UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Learning and Sensorimotor Coordination of Anthropomorphic Robotic Systems

Bruno Duarte Damas

Supervisor: Doctor José Alberto Santos-Victor

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass with Distinction

Jury

Chairperson: Chairman of the IST Scientific Board

Members of the Committee:

Doctor Aude Billard
Doctor Sethu Vijayakumar
Doctor Mário Alexandre Teles de Figueiredo
Doctor José Alberto Rosado dos Santos-Victor
Doctor Alexandre José Malheiro Bernardino

2014

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Learning and Sensorimotor Coordination of Anthropomorphic Robotic Systems

Bruno Duarte Damas

Supervisor: Doctor José Alberto Santos-Victor

Thesis approved in public session to obtain the PhD Degree in
Electrical and Computer Engineering

Jury final classification: Pass with Distinction

Jury

Chairperson: Chairman of the IST Scientific Board

Members of the Committee:

Doctor Aude Billard, Full Professor, École Polytechnique Fédérale de Lausanne, Switzerland

Doctor Sethu Vijayakumar, Professor, The University of Edinburgh, Scotland, UK

Doctor Mário Alexandre Teles de Figueiredo, Professor Catedrático do Instituto Superior Técnico, da Universidade de Lisboa

Doctor José Alberto Rosado dos Santos-Victor, Professor Catedrático do Instituto Superior Técnico, da Universidade de Lisboa

Doctor Alexandre José Malheiro Bernardino, Professor Auxiliar do Instituto Superior Técnico, da Universidade de Lisboa

2014

Para a Rita, para a Laura e para a Júlia.

Resumo

Em robótica, conhecer a maneira como as acções influenciam o ambiente e o estado do robot é um requisito fundamental para uma verdadeira autonomia. Contudo, à medida que os robots vão ficando mais complexos, com um grande esforço de investigação direccionado para robots humanóides com um elevado número de juntas controladas, as suas relações sensório-motoras tornam-se mais difíceis de modelar, conduzindo a situações onde um modelo analítico já não é capaz de fornecer aproximações precisas aos mecanismos robóticos subjacentes. Nesses casos, o recurso a técnicas de aprendizagem artificial pode ser a única maneira de dotar esses sistemas de uma representação fidedigna dos seus mapas sensório-motores.

Esta tese apresenta um mecanismo de aprendizagem computacional, em tempo real, para a estimação de relações genéricas de entrada-saída a partir de um fluxo contínuo de informação. Tal mecanismo é essencial em robots autónomos, onde, de uma forma biologicamente plausível, o robot continuamente aprende e se adapta à medida que interage com o meio ambiente, e onde a experiência adquirida é imediatamente colocada em uso para melhorar a execução das tarefas correntes. O método de aprendizagem proposto é baseado numa abordagem de expectativa-maximização (*expectation-maximization*) que visa ajustar uma mistura infinita de peritos lineares a um fluxo de amostras obtidas em tempo real. Este modelo probabilístico consegue escolher eficientemente o número de peritos que são atribuídos à mistura, desta forma efectivamente controlando a complexidade do modelo resultante. Daqui resulta um algoritmo de aprendizagem incremental e de tempo real, que efectua uma regressão não-linear multivariada numa saída também multivariada, onde a função a aprender é aproximada por uma relação linear na região de influência de cada perito, e que tem a capacidade de activar novos peritos conforme necessário. Uma característica distintiva do método proposto é a sua capacidade para lidar com *multi-funções*, isto é, relações um-para-muitos que naturalmente surgem em alguns domínios da robótica e da visão computacional, recorrendo para isso a um modelo generativo Bayesiano para as predições fornecidas por cada um dos peritos da mistura; como consequência, este método consegue proporcionar relações directas e inversas a partir de um mesmo modelo previamente aprendido.

O modelo sensório-motor aprendido pelo algoritmo proposto apresenta uma grande flexibilidade na maneira como pode ser usado para o controlo de estruturas cinemáti-

cas genéricas: pode ser usado para lidar com mapas sensório-motores tradicionalmente caracterizados por serem difíceis de aprender, como é por exemplo o problema da cinemática inversa ou os modelos onde surgem mudanças súbitas e não assinaladas de contexto sensório-motor; nesses casos, uma predição multimodal é absolutamente necessária e justificada. O algoritmo proposto foi testado intensivamente em simulação, e a sua aplicação ao controlo de braços robóticos foi validada experimentalmente, recorrendo a dois robots humanóides.

Abstract

In robotics, knowing how actions affect the environment and the robot state is a fundamental requirement for autonomy. However, as robots become more complex, with a large research effort put in humanoid robots with a high number of controlled joints, their sensorimotor relations become more difficult to understand, leading to situations where analytical models fail to provide accurate approximations of the underlying robotic mechanisms. In such cases, resorting to machine learning techniques may be the only way to give these systems a reliable representation of their sensorimotor maps.

This thesis presents a computational learning mechanism for estimation of generic input-output relations from a continuous stream of sensorimotor data in a real-time, online fashion. This is a mandatory requirement for autonomous robots, where, in a biologically plausible way, the robot continuously learns and adapts as it interacts with the surrounding environment, and where the acquired experience is immediately used for improving the execution of the current robot task. The proposed learning method is based on a generalized expectation-maximization approach to fit an infinite mixture of linear experts to an online stream of data samples. This probabilistic model can efficiently choose the number of experts that are allocated to the mixture, this way effectively controlling the complexity of the resulting model. The result is an incremental, online and localized learning algorithm that performs nonlinear, multivariate regression on multivariate outputs by approximating the target function by a linear relation within each expert input domain, and that can allocate new experts as needed. A distinctive feature of the proposed method is the ability to learn multi-valued functions, *i.e.*, one-to-many mappings that naturally arise in some robotic and computer vision domains, using an approach based on a Bayesian generative model for the predictions provided by each of the mixture experts; as a consequence, this method is able to directly provide forward and inverse relations from the same learned mixture model.

The sensorimotor model learned using the proposed algorithm exhibits a large flexibility in the way it can be used for control of generic kinematic structures: it can be used to cope with traditionally difficult to learn sensorimotor maps arising, for instance, in inverse kinematics or in sudden, unsignaled changes of sensorimotor contexts, where the need for multi-valued prediction is fully justified. The proposed learning algorithm has been thoroughly tested in simulation, and its application to control of robotic limbs has

been experimentally validated on two humanoid robots.

Agradecimentos

A minha primeira palavra de agradecimento vai para o meu orientador, prof. José Santos-Victor, pelo apoio, pelo constante incentivo ao desenvolvimento da minha autonomia e, principalmente, por me ter acolhido de forma tão calorosa no Vislab: neste laboratório encontrei um ambiente e um companheirismo notáveis a todos os níveis, que muito mitigaram a solidão que inevitavelmente resulta do abraçar a tarefa gigante em que consiste a elaboração de uma tese de doutoramento. Obrigado pois aos meus colegas e amigos do Vislab: Manuel, Ricardo, Plínio, Ravin, Luka, Matteo, Nuno, Alex, entre tantos outros, pelo apoio e pelas discussões e ideias trocadas ao longo destes anos. Em particular, uma palavra de agradecimento para o Lorenzo, pela colaboração profícua iniciada há alguns anos e que ainda hoje se mantém.

Não posso também deixar de referir o apoio dada pela Escola Superior de Tecnologia do Instituto Politécnico de Setúbal, que me possibilitou o desenvolvimento do meu trabalho doutoral em paralelo com as minhas actividades docentes; em particular, quero dirigir o meu agradecimento a todos os colegas e amigos do Departamento de Sistemas e Informática, com quem trabalho há já tantos anos nesta escola.

Quero também agradecer aos membros do júri da minha defesa de tese por todas as sugestões feitas e correcções pertinentes indicadas, agradecendo particularmente aos professores Sethu Vijayakumar e Aude Billard por terem tido a amabilidade de se deslocarem propositadamente a Lisboa para assistir à defesa da minha tese.

Mas a palavra maior de gratidão vai para a Rita, pela forma como me conseguiu aturar nos últimos anos: a sua paciência foi mais do que infinita.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The last decades have presented a remarkable growth of the field of robotics: there has been a huge increase in the complexity of available robots, with current humanoid robots, consisting of tens of independently actuated joints and equipped with a multitude of different sensors, taking their first steps towards mimicking human behaviour. To tackle such highly complex robotic mechanisms, more sophisticated methods were developed to control these robots, to make them execute the desired tasks.

However, as robots complexity increases, building the analytical models for their *sensorimotor* relations, relating their actions to corresponding sensory feedback, needed for movement control, turns out to be more and more difficult and time-consuming. This is a direct consequence of the large number of degrees of freedom of the current humanoid robots, which can make the task of deriving closed form expressions for robot kinematics and dynamics impractical. More important, it may be difficult to accurately measure some physical parameters like friction, and the existence of highly nonlinear physical interactions, such as actuator unmodelled nonlinearities, backlash, time drifts in the kinematic and dynamic characteristics of the robot, produced by material wear, calibration errors in cameras used for end-effector tracking, soft or deformable parts and unmodelled mass distributions, among other effects, make the task of obtaining adequate and accurate models for such kind of systems infeasible (Peters and Schaal, 2006). In this context, sensorimotor relations coming from interactions with objects and the external environment are even more difficult to model with sufficient precision.

An alternative to hand-made models of the robot sensorimotor relations, based on human insights into physics and laws of motion, is to *learn* such models directly from the information coming from the robot sensors, during its operation and interaction with the environment. For a variety of reasons, this is a very attractive approach: first, the complexity of the underlying robotic interactions can be more efficiently acquired through learning, since the model is directly estimated from sensory feedback. This converts

the problem of accurately modelling every physical interaction to the broader problem of choosing an appropriate form and complexity for the learning mechanism that can efficiently represent the underlying physical model structure, as suggested by the acquired data. This way, difficult or even unknown physical interactions can be approximated by the learned model. Secondly, it is not needed to predict every possible scenario for the robot and environment evolution and to program the robot accordingly, since, during operation, the learning algorithm will ideally build its internal model based on the robot experiences and sensory feedback, that can in turn be used for controlling it. Finally, it is worth of notice that by endowing a robot with efficient learning capabilities almost no outside intervention is required for its operation: this is a fundamental aspect of true autonomy. As a side consequence, some unexpected, different ways of performing the required tasks may emerge during the learning process, not predicted by human design or engineering.

Robotic sensorimotor learning can be regarded at various levels, depending on how robot perceptions and actions are defined. At a higher level a soccer playing humanoid robot, for instance, may want to decide from a set of possible actions like dribbling a ball forward, passing it to a team-mate or shooting it to the goal. In this case, perceptions may consist of players positions and velocities, as well as the robot own localization and direction of movement. Deciding how to act or make a plan based on previous experience then typically will resort to Artificial Intelligence (AI) techniques like decision tree learning or reinforcement learning. On the other side, at a much lower level, the same robot may wish to know what commands should send to its motors in order to, say, advance the right leg while maintaining its balance. This is the realm of statistical learning methods like nonlinear regression or density estimation of real-valued quantities. This idea of adaptive robotic systems that respond to environment changes and that can use the past experience to continuously improve their performance has its origins in the field of cybernetics, back in the mid of the twentieth century.

Of course, learning of basic sensorimotor skills must first be achieved before a robot can learn and make plans at a higher level. This hierarchy, with its different levels of relations to learn, has a close relationship to a human developmental perspective, where, for instance, basic reaching and grasping skills must be properly acquired before a baby can learn to manipulate objects. This developmental paradigm assumes a newborn to be endowed with a genetically programmed core set of reflexes, and that the ability to solve and learn various tasks is built upon previously acquired simpler skills, learned at precedent behavioural stages (Payne and Isaacs, 2001).

This dissertation will focus on a low level perspective of sensorimotor learning, where basic relations between the robot motor commands and the consequently perceived environmental changes are to be efficiently learned and employed, in a flexible way, for subsequent control. In this context, the type of information to be inferred from the learned

model will have a decisive impact on the structure of the computational model used for learning. A sensorimotor model can be viewed as a causal relation that associates, in a stochastic manner, a current robot perceived state $\boldsymbol{s}_t$ and an action $\boldsymbol{a}_t$ to a consequent state $\boldsymbol{s}_{t+1}$, *i.e.*, a mapping $(\boldsymbol{s}_t, \boldsymbol{a}_t) \mapsto \boldsymbol{s}_{t+1}$. Such *forward model* can be used to predict consequences of actions; this is the case, for instance, of the forward kinematics model of a robotic mechanism, that is described by a map $\boldsymbol{q} \mapsto \boldsymbol{x}$, relating robot joint positions to a corresponding end-effector task space posture. In this case, the action $\boldsymbol{a}_t$ corresponds to the vector of joints positions $\boldsymbol{q}$, while the current robot state $\boldsymbol{s}_t$ is irrelevant for this model. Another view of the forward kinematics takes the form $(\boldsymbol{q}, \nabla\boldsymbol{q}) \mapsto \nabla\boldsymbol{x}$, where small changes on the end-effector task space posture are taken as a consequence of varying the joints positions in a given robot configuration. This formulation is useful from a control point of view, since it shows how changes in actuation values reflect in the robot state.

However, most of the times a control perspective is more interested in obtaining the actions $\boldsymbol{a}_t$ that will drive the robot or the environment from state $\boldsymbol{s}_t$ to a new state $\boldsymbol{s}_{t+1}$. This information can be directly obtained if an *inverse model* is used, providing a mapping $(\boldsymbol{s}_t, \boldsymbol{s}_{t+1}) \mapsto \boldsymbol{a}_t$. Examples of such formulation include the inverse dynamics and the inverse kinematics models used in robotics. In the former case a relation $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \mapsto \boldsymbol{u}$ is sought, where, in current robot state $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, a motor command $\boldsymbol{u}$ is provided that will provoke a desired change in the robot state, represented by $\ddot{\boldsymbol{q}}$. In the latter example, a static relation $\boldsymbol{x} \mapsto \boldsymbol{q}$ is modelled, giving joint positions $\boldsymbol{q}$ that will lead to a desired end-effector task space posture. As in the forward kinematics case, the static relation represented by the inverse kinematics model does not explicitly need the current robot state.

The work described in this dissertation puts a strong emphasis in the problem of learning these kind of sensorimotor models in a fast and flexible manner, with the minimal human intervention. The application of these models to a robotic framework, where they are to be used for control and acting, imposes severe constraints in the learning algorithm that are usually not present in other domains, as described in the next section.

1.2 Sensorimotor Learning in Robotics

To the casual reader it may come with some surprise that the task of learning robotic sensorimotor relations, using them for controlling the movements of the robot, is still a difficult and not completely solved problem: after all, from a human point of view, it is a trivial task to move, for instance, our right arm to reach and pick a cup of coffee laying in the table in front of us. It cannot be forgotten, however, that each of us took months of continuous learning, during the first years of our existence, to be able to perform these kind of reaching and manipulation operations in a seamless and effortless way —

watching a few months old baby trying to accomplish the same kind of tasks is sufficient to make evident the difficulties that may arise when these sensorimotor relations are not yet completely learned. The human brain is the ultimate learning machine, and if years of training are required to become proficient at playing tennis or a musical instrument, we cannot expect our efforts towards the building of robots that can autonomously learn their basic sensorimotor maps to be without some slings and arrows.

There are many difficulties, introduced by the need for autonomous operation of a robot, that a competent learning algorithm must be capable of dealing with, specially if a large time span — eventually the full lifetime of the robot — is aimed for; such issues are identified and discussed in the following sections.

1.2.1 The Curse of Dimensionality

High dimensional spaces suffer from what is usually known as the "curse of dimensionality" (Bellman, 1957): given an input space with dimension d, the number of sample points needed to cover this space with the same detail and resolution grows exponentially with the increase of the number of dimensions of this space. This phenomenon can be easily understood if a hypercube is used to represent the input space: in this case, if k regularly spaced points are used along each direction of the input space to partition the input, the data grid thus obtained has a total of k^d points. This poses severe problems for the learning algorithm, since statistical distributions describing the sensorimotor data will need a prohibitive number of training samples to be properly estimated. On the other hand, fully exploring these high dimensional spaces in a robotic context takes an exponential operation time: for large values of d, this can easily surpass the expected lifetime of the robot.

Deeply related to the aforementioned curse of dimensionality, a direct and non intuitive consequence of high dimensional spaces is the fact that most of the data density becomes concentrated near the boundaries of the input space, making it more difficult for a learning algorithm to make predictions, as there is the need to extrapolate from the acquired data instead of interpolating between them. Taking the example presented by Hastie, Tibshirani, and Friedman (2009), if N data points uniformly distributed in a p-dimensional unit ball are considered, the median distance from the ball centre to the closest data point is given by $(1 - 0.5^{1/N})^{1/d}$: for $N = 500$ and $d = 10$, this results in a value of 0.52, which means that more than half of the data points are closer to the boundary of the space than to the origin of the unit ball; for $d = 20$, this value rises to 0.72.

This also has important consequences for probability distributions describing this input space: as dimension d increases the probability mass ceases to become concentrated in the centre of the distribution, moving away to the distribution tails. As an illustrative example of this phenomenon, it is a well known fact that, for a multivariate Normal

distribution with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$, the squared Mahalanobis distance to $\boldsymbol{\mu}$, given by $(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$, follows a chi-squared distribution with d degrees of freedom. Using this distribution, for $\boldsymbol{\Sigma}=\sigma^2\mathbf{I}$ and $d=1$, this means that $\|\boldsymbol{x}-\boldsymbol{\mu}\|<3\sigma$ with 0.997 probability, *i.e.*, the distance of a random point $\boldsymbol{x}$ to the distribution centre $\boldsymbol{\mu}$ will almost always be lower than 3 standard deviations, with most probability density located near $\boldsymbol{\mu}$; for $d=10$ the same probability lowers to 0.468, and if $d=20$ is considered a probability of 0.017 is obtained, meaning that almost all probability density has been pushed towards the distribution tails. As a consequence, the notions of dispersion and concentration associated with these probability distributions start to lose their usual connotation as the space dimension increases.

Altogether, as the input dimension increases, the notion of locality starts to break, as the average difference between the distance to the closest and furthest data points goes to zero: this means that all data points are approximately at the same distance from each other, destroying the notion of neighbourhood. This has a dramatic impact on most learning algorithms, as they usually rely on some sort of distance or similarity measure between training points.

1.2.2 Computational Complexity

Integrating a learning algorithm in an autonomous robotic platform imposes a careful management of the computational resources available to the learning tasks. Ideally, the learning mechanism should be able to process the information acquired by the robot during its normal operation. Altogether, the following is usually required:

- Learning must be performed *online*. This means that, when a decision must be made resorting to the computational model learned so far, not all training data has yet been presented to the algorithm. This is clearly a fundamental requirement for the learning algorithm if autonomous operation is necessary for the robot, where the update of the sensorimotor model is performed while simultaneously using it for control, in opposition to the offline processing of the data after all the acquisition experiments take place.

- As a consequence of the previous item, learning must be done in a *real-time* manner, taking into account the robot computational resources and the desired control rate. The learning method must have an update algorithm that can process training points at least as fast as they are acquired, and must provide predictions hastily enough to pace with the control sampling rate.

- A scenario where the learned model is potentially used during the lifetime of a robot results in the necessity of an *incremental* learning scheme, where, due to the real-time requisite described above, not all training points can be stored in

memory. Some learning methods maintain a fixed size subset of the training data, choosing at each learning iteration which point to discard, in order to maintain the most representative data. An alternative to this sparsity based approach is to keep in memory a set of quantities that can summarize the training data seen so far, for instance a set of sufficient statistics under a statistical generative model or the current parameters of an artificial neural network.

1.2.3 Uncertainty

Data coming from robotic applications is noisy, and a machine learning algorithm must be able to deal with the randomness in the training data. Among other desirable properties, a learning mechanism should be able to correctly infer the amount of output noise present in different regions of its input space, and should also be robust enough with respect to the presence of possible outliers in the acquired data.

The uncertainty presented in the input part of the training data must also be taken into account: in general, its probability distribution is not known in advance and some learning algorithms can be particularly sensitive to temporal shifts of these distributions. This happens, for instance, when, instead of randomly sampling data from the whole input space, the training process is focused in particular regions of this space, as naturally occurs when learning is performed online, or when data comes from a stream of sequential samples coming from robot sensors and actuators. In these cases, a degradation of the prediction performance of some learning algorithms can occur over input regions not visited for a long period, a phenomenon known as *negative interference* (Atkeson, Moore, and Schaal, 1997a).

In a robotic learning context, there is the possibility of deciding where to sample the next training point: this research field is known as *active learning*, where the emphasis is put in efficiently learning a given relation without wasting resources needed for the acquisition of the training set. This is a particularly useful mechanism when learning online some sensorimotor maps: a reasonable approach consists in choosing an input point to sample that is believed to decrease the global prediction error the most. Since the true value of the relation to learn is not known, this usually amounts to picking an input point that, together with the corresponding observed output, is expected to reduce the prediction variance the most (Settles, 2009), according to the bias-variance decomposition of the prediction error, further detailed in Section 2.2.

1.2.4 Model Complexity

Choosing an appropriate structure complexity for the learning model is a crucial step that can dramatically influence the resulting prediction error. Such complexity can appear in many forms in different learning models, like the number of layers and units in an artificial

neural network, the number of components of a mixture model or the characteristic input length-scale in kernel regression, to name just a few examples. The learned model will exhibit a large bias when this complexity is chosen to be lower than what would be needed to properly represent the training data. If, on the other hand, the structure complexity for the learner is too high, the resulting model will overfit the data: as a consequence, predictions will have a large variance across different training sets. In both cases, the prediction error on an independent testing set will increase, a phenomenon known as the bias/variance trade-off (Geman, Bienenstock, and Doursat, 1992).

In robotic applications, an automatic learning of the correct model complexity is highly desired, as it avoids the need for human insights about the nature of the sensorimotor map being learned; keeping the model simple enough is necessary to save the computational resources needed for learning and prediction. Automatic adjustment of the model complexity also permits the correct learning of different regions of the input space with potentially different structural complexities. However, estimation of the correct model complexity for a set of training data is still a difficult problem, specially when an online learning scheme is desired.

1.2.5 Redundancy and Multi-Valued Relations

The particular structure of the sensorimotor models needed for controlling a robot introduces some difficulties that generally don't arise in typical applications of nonlinear regression algorithms: learning the forward kinematics model of a robotic manipulator, for instance, can usually be easily done resorting to modern function approximation algorithms. However, when controlling the manipulator, it may be more interesting to learn its inverse model, that predicts which joint angles $\boldsymbol{q}$ should be used to reach a given end-effector posture. It turns out that normally there isn't a unique solution for this problem, as most robotic manipulators can achieve the same task space posture using different joint space configurations. These kinematics inverse models can be represented by multi-valued functions, *i.e.*, one-to-many relations also known as *multifunctions* or *multimaps*. Furthermore, some mechanisms can exhibit multi-valued sensorimotor relations even for forward models: this happens for instance in parallel robotic devices (Merlet, 2006). Another situation of interest is the presence of hidden variables that can change the sensorimotor data that is presented to a learning algorithm during the training phase: a classical example is the effect that an unsignaled change of the load at the robot end-effector may have over its dynamical relation. As a consequence, since the learning algorithm is not aware of such changes of the sensorimotor context, the training data set appears to have been generated from a multi-valued relation from inputs to outputs, consisting of different function branches, one for each context value, as depicted in Figure 1.1.

Trying to directly learn such multi-valued models using standard regression algorithms is condemned to an utter failure, as these methods are not prepared to learn these kind

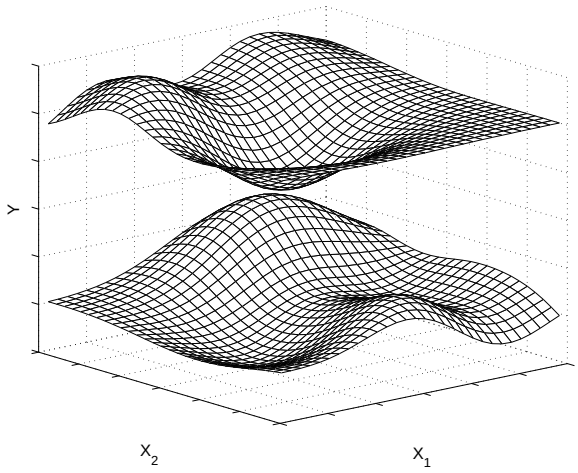
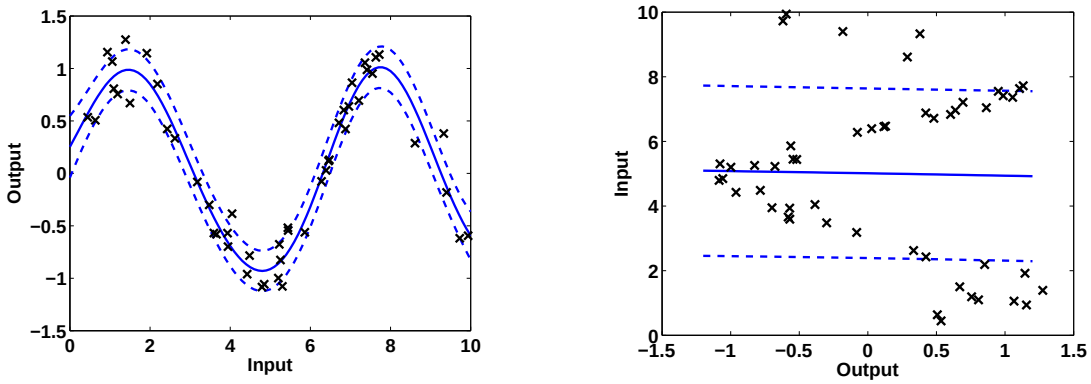


Figure 1.1: Training data arising from switching sensorimotor contexts implicitly defines a multi-valued relation to be learned

of relations; instead, they will usually average the different solutions to the problem, as shown in Figure 1.2, where a Gaussian process (Rasmussen and Williams, 2006), a widely used nonlinear function approximation method, is used to learn a forward and inverse probabilistic model from the same training data. This averaging may have catastrophic



(a) Forward relation.

(b) Inverse relation.

Figure 1.2: Learning a forward and an inverse relation using a Gaussian process. To left is the learned forward model: training data is depicted by crosses, the learned conditional expectation is given by a solid line and the uncertainty interval (one standard deviation) is represented by the two dashed lines. The inverse model, learned using the same training points (with the input and output roles reversed) is shown in the right figure.

consequences, since the space of solutions for these sensorimotor mappings normally is not a convex set: averaging distinct solutions will not generate a valid solution for the learning problem. This situation is depicted in Figure 1.3, where it can be seen that averaging two valid inverse kinematics solutions will not produce a valid inverse solution for the kinematics problem. Additionally, when the robot is redundant, having more actuated

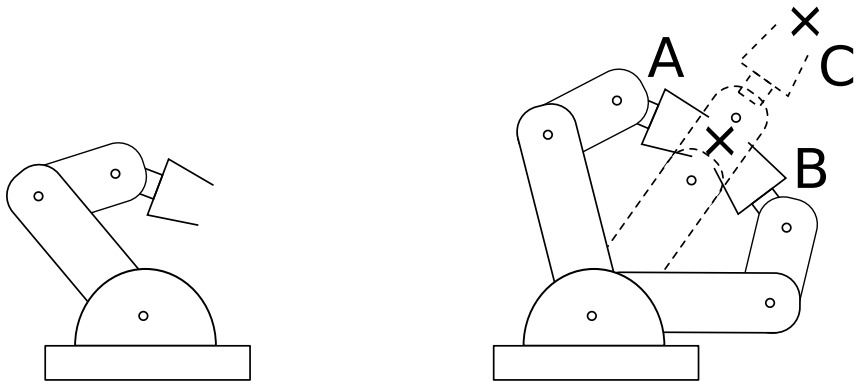


Figure 1.3: The inverse kinematics learning problem: (a) A 3 degrees of freedom planar manipulator; (b) Configurations A and B produce the same end-effector position, but their average, labelled C, does not.

degrees of freedom than the dimensionality of the task space, the space of solutions for the inverse kinematics problem is continuous, with dimension equal to the difference between the dimensionality of the joints and task space, as shown in Figure 1.4, and the inverse model can no longer be represented by a multi-valued relation.

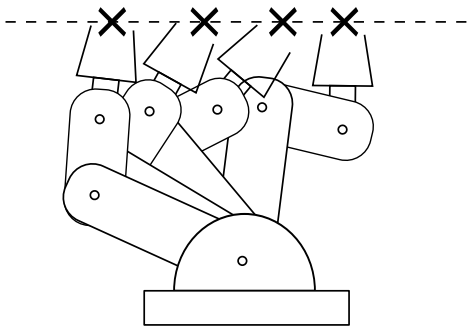


Figure 1.4: The redundancy problem: the task space is one dimensional, corresponding to vertical position of the end-effector (depicted by a dotted line), and since the robot has a higher number of controlled joints, there is a continuum of solutions to this inverse kinematics problem.

Altogether, these complex forward and inverse models, needed for robot control, are perhaps some of the most challenging problems a learning algorithm can face, and the reason why standard machine learning techniques are not yet trivially used in every robotic application.

1.2.6 Adaptation to Changing Environments

In many situations, robotic environments are not stationary: this means that there may exist slow drifts in the sensorimotor relations being learned, that must be accounted for to prevent the learning model from getting biased by old data. Such adaptation to these slow-varying relations can be done in different ways, according to the internal structure of

the learning model: some algorithms introduce some sort of time decay in their internal models, for instance in their internal parameters or sufficient statistics. Other methods, based on the storage of a subset of training data as an accurate representation of the underlying model, may show a higher propensity to discard old data in favour of recently acquired training points.

Other robotic tasks, however, involving for instance handling and manipulation of different objects, cause abrupt changes in the environment and the mappings to be learned. The kinematics mapping from robot joint angles to end-effector position, for instance, changes whenever different tools are used; another classical example is the change in the robot dynamics due to the variation of the load of the end-effector. This is known as learning under varying contexts, where an unobserved context variable changes the map to be learned. Such context can generally be a discrete variable, corresponding to the case where only a finite, albeit unknown, number of different contexts exist, or continuous, indicating a smooth change on the mapping to learn. The most straightforward answer to this problem is to use the same form of adaptation that is used for slow drifts of the sensorimotor maps: of course, it is terribly inefficient to relearn the complete mapping every time the context changes, specially when there is an effective chance that a previously learned context may be presented again to the robot. As an alternative, the learning algorithm should keep enough information in its internal model to be able to successfully represent the robot model and to make accurate predictions under environment context switching.

1.3 Objectives and Contributions

This dissertation provides a statistical learning framework specially suited for use in autonomous robotic applications, in the context of generic sensorimotor maps estimation and consequent use for control purposes. That means, in the first place, that such learning mechanism must be able to stand online operation in a real-time environment; the deep desire for an extended robot autonomy also implies that almost no information about the model to learn will be available. This discards the use of parametric models like those used for learning a body schema (Bongard, Zykov, and Lipson, 2006; Sturm, Plagemann, and Burgard, 2008; Hersch, Sauser, and Billard, 2008), where a given kinematic structure is assumed and the focus is put on estimating the parameters that describe such model. This latter approach has some evident advantages, as a fixed sensorimotor structure is generally easier to learn; however, one one side, parametric models increase the effort put on modelling the robot; on the other side, as stated earlier, they cannot take into account unmodelled or hard to describe effects and interactions, and thus potentially limit the performance of the resulting model.

The proposed learning framework builds upon the mixture of experts concept (Jacobs,

Jordan, et al., 1991) to provide a learning algorithm that can satisfactorily deal with the challenges posed by many robotic control architectures. At its core, a mixture of experts follows a divide and conquer strategy: the input space is softly partitioned among experts, and each expert approximates the map to be learned only within a localized region of the input space. This localized learning approach can be used to approximate multi-valued functions, as long as interference between experts sharing the same regions of the input space is avoided: this can overcome one of the main difficulties that arise when learning some forward and inverse sensorimotor models of robots.

A large number of mixtures of experts architectures have been proposed in the last two decades, from mixtures of neural networks (Bishop, 1994) to the more recent mixtures of Gaussian processes (Rasmussen and Ghahramani, 2002). Among these, the mixture of linear models (Xu, Jordan, and Hinton, 1995), where each expert approximates the map to learn by a linear relation from inputs to outputs, has some properties that make its use particularly compelling in robotic applications. Its simple structure makes possible to derive simple algorithms that can efficiently update the mixture whenever a new training point arrives, or to quickly provide estimates whenever required. This is a fundamental requirement for online, real-time learning. Also, the linear nature of the relation to be approximated by each expert makes it possible to easily invert this relation: as a consequence, inverse predictions can be obtained from the learned mixture. This is an enormous advantage and greatly increases the learned model flexibility, as it can now be used to predict both forward and inverse sensorimotor models from the same computational structure.

This dissertation presents a generative statistical model for the mixture of linear experts architecture that takes into account the partitioning of the input space and the linear relation within each expert input domain. Additionally, some convenient prior and hyper-prior distributions for the mixture parameters are introduced, to allow the mixture to autonomously estimate some nuisance parameters that define the mixture complexity, like the input length-scale or the output noise of the relation to learn.

Training resorts to an online and incremental version of a generalized expectation-maximization procedure that can efficiently process new data points as they are acquired. Adjusting the overall model complexity, in particular the number of experts assigned to the mixture, does not adopt a fully Bayesian scheme: although sound and formally elegant, such approach could hopelessly compromise the algorithm real-time capabilities; instead, an automatic model selection, resorting to a complexity penalization criterion, is used. The resulting mixture, as it will be discussed in detail in Chapter 3, can be viewed as an infinite mixture where, at a given time, only a finite subset of its components are active, effectively contributing to this mixture. For this reason, this probabilistic model was coined the Infinite Mixture of Linear Experts (IMLE) model.

A learned computational model for a sensorimotor relation only has some use if a

prediction mechanism is also provided. Here, the critical issue is to avoid the blending of distinct prediction solutions when the relation exhibits a multi-valued nature, either for forward or inverse prediction. One of the contributions of this dissertation is a mechanism for the prediction stage that can automatically identify and obtain the correct set of solutions for a given query, using the mixture of linear experts learned so far. This multi-valued prediction scheme can be used to obtain forward and inverse predictions for a given input or output query, respectively; in fact, the same reasoning used to obtain a set of multi-valued inverse predictions can be employed for prediction over an arbitrary subset of variables belonging to the joint input-output space, as a function of a query consisting of the remaining variables of this space. Besides such predictions, the IMLE model can also provide uncertainties associated with each prediction; it can also calculate the local estimated Jacobian at any query point, and it also makes available an estimate of the prediction variance reduction that is expected if a particular location of the input space is sampled to provide a new training point for the IMLE model — this way, active learning schemes can be implemented that can supposedly lead to a more principled exploration of the input space.

The sensorimotor models learned using the algorithm proposed in this dissertation exhibit a large flexibility in the way they can be used for control of generic kinematic and dynamic structures: they can be used to cope with traditionally difficult to learn sensorimotor maps, like the ones that arise, for instance, in inverse kinematics. It is proposed in this dissertation a practical application of such learned kinematics model to an end-effector position tracking problem, where the predicted Jacobian is used for closed-loop control and the inverse predictions, provided by the same model, contribute the information needed for planning a trajectory in an open-loop control mode. In this way, both open and closed-loop control schemes can be readily applied using the same learned sensorimotor model. The online capabilities of the proposed learning algorithm even dismiss the need for a previously learned model, as learning can be performed online during the robot operation.

Another contribution of this dissertation is related to the existence of non-observable sensory variables that can produce sudden, unsignaled changes on a robotic sensorimotor context, thus effectively changing the sensorimotor relation, as viewed from a robot perspective. This is another robotic domain where learning is traditionally difficult: the fact that different sensorimotor relations, resulting from different hidden contexts, can be interpreted as a single relation with multi-valued outputs, allows a straightforward use of the IMLE algorithm to learn these robotic sensorimotor maps.

With all these ideas in mind, the thesis herein defended is that *this specially developed learning framework is a feasible and efficient way of learning, in an online fashion, generic robotic sensorimotor maps, that provides a large flexibility in the way it can be used for controlling a robot and that, additionally, also achieves a prediction accuracy at*

least comparable to competing online sensorimotor learning algorithms, while providing additional prediction features that cannot be found in these other approaches.

Previous publications related to the work presented here include (Damas and Santos-Victor, 2012) and (Damas and Santos-Victor, 2013), where the multi-valued online learning algorithm here described was firstly introduced. Experimental validation on humanoid robotic platforms was performed in collaboration with Lorenzo Jamone: the use of the IMLE model for simultaneous open and closed-loop control of the position of the hand of an anthropomorphic robot arm is reported in (Damas, Jamone, and Santos-Victor, 2013), while its success in dealing with multi-valued relations arising in sensorimotor maps with hidden context switches is described in (Jamone, Damas, et al., 2013b) and (Jamone, Damas, et al., 2013a).

1.4 Dissertation Outline

This dissertation is organized as follows:

- Chapter 2 describes the sensorimotor learning problem in detail, discussing how a sensorimotor model can be used for control. A review of the major machine learning techniques is also presented, making reference to the strengths of each approach and equally mentioning their main limitations in a real-time and demanding robotic sensorimotor learning context.

- Chapter 3 presents the IMLE probabilistic model, together with an online algorithm to train it in an efficient manner. A prediction method for obtaining a set of multi-valued solutions from the learned mixture, either in forward or inverse prediction, is also discussed in this chapter, and analytical expressions for the prediction Jacobian and the estimate for the prediction uncertainty reduction are derived in it.

- Chapter 4 provides an extensive experimental evaluation of the IMLE algorithm with respect to single and multi-valued prediction, comparing its performance, in an online setting, to other state-of-the-art function approximation algorithms.

- Chapter 5 considers the application of the IMLE model to sensorimotor learning and control of robotic devices. It is shown how the IMLE model can be used to learn a sensorimotor relation in an online fashion, and how it can be used to simultaneously provide forward and inverse predictions. It is also shown in this chapter how the IMLE model deals with switching of hidden contexts in a robotic learning and control setting.

- Finally, Chapter 6 concludes this dissertation, presenting a summary of contributions presented in this dissertation and pointing directions of future work.

Chapter 2

Sensorimotor Model Learning

"Few would deny that the most powerful statistical tool is graph paper."

Geoffrey Watson, 1964, *Smooth Regression Analysis*

It is hard to conceive intelligent behaviour without movement: every interaction with the world, like the simple act of grabbing some object, but also more sophisticated communication mechanisms like speech or writing, requires the capability to change the environment, which can only be accomplished by some sort of motion commands (Wolpert, Ghahramani, and Flanagan, 2001).

Sensory information about the environment and the agent state, be it a robot or a human being, is also fundamental: under this perspective, such information can be regarded as the necessary input of an information processing system that converts this sensory feedback into motor commands that can, in turn, be used to change the environment. Some neurological findings even point out that a constant flux of information, coming from proprioceptive sensors in the human body, is perhaps a fundamental requirement for the human consciousness (Damásio, 1999).

The first attempts to build intelligent machines, however, put the emphasis on symbolic

computation, somewhat neglecting the essential role of the body and the capability to generate movement in the creation of intelligent behaviour. In the golden era of symbolic artificial intelligence (AI), in the second half of the twentieth century, the advent of the Turing machine (Turing, 1937) lead to the creation of the first computers, which enabled a practical attempt to create an artificial mind. These were times of great expectations, and prominent artificial intelligence scientists like John McCarthy, Marvin Minsky, Allen Newell and Herbert Simon, among many others, had the firm belief that twenty years, back then, would be sufficient to be able to replicate a human brain (Russell and Norvig, 1995).

It turned out that this symbolic approach was overly optimistic in its intents, and by the eighties it was suffering from a steady decline. Some philosophers and scientists were strong critics of the pure symbolic manipulation approach to the building of an intelligent mind, like John Searle (Searle, 1980), Rodney Brooks (Brooks, 1990) and Hubert Dreyfus (Dreyfus, 1992). One of the major criticisms was that symbolic reasoning draw a clear distinction between body and mind, a duality coming back from the work of Descartes and that was no longer supported by modern philosophy, psychology or recent neurology findings.

In the same time, different approaches to artificial intelligence came to light or were revived, like the *connectionism* or *behavioural robotics*, that took an information processing approach to artificial intelligence that did not rely on symbolic manipulation. Connectionism, for instance, believes that intelligent behaviour emerges as a result of interconnected networks of simpler components (Hinton, McClelland, and Rumelhart, 1986; Hinton, 1991): although not new, this concept underwent a renewal with the rediscovery of the backpropagation algorithm, that revived the neural networks model first suggested by McCulloch and Pitts (1943). The field of behavioural robotics also considers intelligence to be a quality that emerges from the interaction of simple processes taking information from sensors and converting it to actions performed by the robot on the environment (Brooks, 1991). A particular example of this view is the *subsumption architecture* (Brooks, 1986; Brooks, 1991): here intelligent behaviour emerges from a hierarchical architecture of simple behaviour modules, where higher level modules perform more abstract tasks and control the lower layers. These and other approaches can be viewed under the broader concept of cybernetics, a interdisciplinary field that studies regulatory systems: under this paradigm there is a kind of symbiotic relation between the concepts of intelligence and motor capabilities. Without movement, intelligence has no means of expressing itself; without intelligence, the motor machinery of a biological or artificial system lacks the capability to generate autonomous behaviour, a fundamental requisite in demanding and permanently evolving environments.

A concept that is central to intelligent behaviour is that of an *internal model*. To be

able to achieve its desired goals and complete some given tasks, a biological or artificial brain must necessarily have some kind of internal representation of how does a motor command influence the environment, that is able to make predictions and to anticipate the consequences of its actions (Ghahramani, 2013). Without internal models for the consequences of actions, as perceived by the sensory system, an artificial or biological brain cannot hope to achieve more than mere random movements.

Even when a system apparently does not have an internal representation of its sensorimotor model, there is always some kind of information that is used to guide the system while it interacts with the environment, that can be interpreted as an implicit model for this interaction: it may consist, for instance, in genetically transmitted information encoding basic movement abilities, fitted by natural evolution over millions of years, or, for artificial systems, in empirical knowledge, provided by a human expert, on how should an artificial system act in order to perform the desired tasks.

Nevertheless, implicit models have only a very limited capability to represent the richness of the complex environments an intelligent system interacts with. Higher autonomy and superior intelligence are related to the ability to generate more complex behaviours, which in turn depends on the representation capabilities of the internal models used in this process. Also, the environment dynamics may show a large variability and may change unpredictably over the time, a situation a hard-coded internal model cannot easily cope with.

Adaptive internal models, that build and change, over the time, their representation of the influence of the robot actions on the surroundings and its own state, are therefore required to endow a robot with a high level of autonomy, that allows the improvement of its response performance in face of unpredicted situations.

Of course, robotic sensorimotor learning can be regarded at various levels, depending on how robot perceptions, actions and goals are defined. In general, basic, low-level sensorimotor skills must first be properly acquired before a robot can learn more complex tasks. Newborn infants, for instance, must first learn how to reach objects before being able to grasp them, and only then can they start learning how to manipulate these objects, a learning process that takes place during their first months of life (Payne and Isaacs, 2001). This developmental perspective, where learning and the ability to solve complex tasks is acquired during different stages of the human development, and where the learning of new tasks builds upon previously acquired skills, as been adopted in artificial systems under the *developmental robotics* paradigm (Asada, MacDorman, et al., 2001; Lungarella, Metta, et al., 2003; Lopes and Santos-Victor, 2007; Asada, Hosoda, et al., 2009).

As stated in Chapter 1, this dissertation main focus is the learning of low-level sensorimotor models, that relate the robot motor commands to the perceived environmental changes. These models are essential in a hierarchical architecture, and allow higher cognitive processes to internally simulate actions and predict its consequences. This is a

simulation perspective of the internal sensorimotor models. Second, the question of how to generate movement to achieve a certain goal, in terms of the environment state, needs also to be addressed. This can be viewed as a *control perspective*, where the sensorimotor models are used to generate desired motions, as required by higher cognitive abilities.

The next sections will discuss these two different perspectives, from the point of modelling and learning. Section 2.1 will first address the kind of models that are required for simulation and control, pointing out the main issues that may arise when learning such models. Then, in Section 2.2, a review of several machine learning techniques that can be useful for the adaptation of these internal models is provided, highlighting the major strengths and limitations of these methods in robotic sensorimotor learning contexts.

2.1 Internal Models for Sensorimotor Coordination

Let the robot state, $\boldsymbol{s}_t$, be defined as a vector that summarizes all the relevant information regarding the environment and the robot, needed to accomplish some task, at a given time instant t. This state is usually obtained resorting to proprioceptive and exteroceptive sensors, that convey information regarding the robot and external environment condition. While sometimes the robot state may consist of raw sensor data, many times some kind of processing and feature extraction is needed to provide this information. This is the case of signals coming from vision or audio sensors: using such raw data directly for real-time robot control, due to its complexity and high dimensionality, is in principle condemned to failure. In such cases, preprocessing of the sensor information is an essential requisite, in order to obtain a minimal set of features that are relevant to the task the robot is supposed to accomplish. These features can be, for instance, the position and velocity of tracked subjects in a video sequence coming from robot cameras, such as objects, humans or the robot own end-effector, or the segmentation into known phonemes of a perceived speech. Extracting relevant information from raw sensory data is a large research subject on its own that will not be addressed in this dissertation: here it will be simply assumed the existence of a state vector $\boldsymbol{s}_t$, without regarding how it was actually obtained.

Also, an action vector $\boldsymbol{a}_t$ must be defined, containing the set of commands a robot can issue at time t, that can potentially modify its state. These commands can correspond to physical quantities driving the robot actuators, like the input voltages of DC motors actuating the robot joints, but can also be high level actions like, for instance, a request to move the robot end-effector to a given position in the Cartesian space or a request to grab an object. In these cases the robot will need another kind of mechanisms to translate these actions to low level actuator commands. As a low level actuation is assumed in this dissertation, the action vector will be represented by a vector of real values (as opposed to symbolic actions, usually represented by a finite enumeration of possible discrete actions).

2.1.1 Forward Models

A sensorimotor model is a robot internal model for the way its actions influence the environment. This model implicitly defines a causal relation from actions to environment changes, that can be naturally represented by the following *forward model*:

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{a}_t, \boldsymbol{s}_t) \ . \tag{2.1}$$

This forward model allows the robot to predict the consequences of its actions: given a current state $\boldsymbol{s}_t$, this model predicts the state at a consequent instant $\boldsymbol{s}_{t+1}$ as a consequence of performing action $\boldsymbol{a}_t$. Although a discrete time model is adopted in the following discussion, it is easy to extend all the considerations made to a continuous time model, by replacing the state at the next time step in equation (2.1) by the time derivative of the state. Note also that, as stated in the previous discussion, it is assumed in this model that the state is completely observable. If this was not the case, equation (2.1) would have to be extended with the observation model

$$\boldsymbol{y}_t = h(\boldsymbol{s}_t) \ ,$$

where $\boldsymbol{y}_t$ would correspond to the observed variables, while the unobserved full state $\boldsymbol{s}_t$ would have to be estimated from samples $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$. The *state estimation* issue will not be further considered in this dissertation: in the remaining of this chapter it will always be assumed a full access to the system state, *i.e.*, that $\boldsymbol{y}_t = \boldsymbol{s}_t$.

This internal forward model may be represented by a deterministic relation. Sometimes, however, due to the stochastic nature of the environment, due to the presence of sensor noise or to the existence of an incomplete state vector, where some hidden variables also influence the evolution of the environment, it may be more convenient to represent the forward model by a stochastic relation: in this case, instead of a single deterministic value, equation (2.1) represents a probability density function over the possible values of $\boldsymbol{s}_{t+1}$.

The state vector $\boldsymbol{s}_t$ can be understood as the *context* under which the action $\boldsymbol{a}_t$ is performed. For dynamical systems the information conveyed by this state vector is essential, since to predict its evolution both the current action and the past history of the system, summarized in $\boldsymbol{s}_t$, are required. On the other hand, static forward models, like the forward kinematics of a robot, do not need any contextual information: in this situation equation (2.1) can simply be written as $\boldsymbol{s}_{t+1} = f(\boldsymbol{a}_t)$.

Learning forward models can usually be done in a straightforward manner by resorting to standard supervised learning techniques, where the input data is given by $\boldsymbol{x} = (\boldsymbol{a}_t, \boldsymbol{s}_t)$ and the corresponding output is $\boldsymbol{y} = \boldsymbol{s}_{t+1}$; the task is then to learn the relation $\boldsymbol{y} = f(\boldsymbol{x})$ from training examples $(\boldsymbol{x}_i, \boldsymbol{y}_i)$; this is known as a *direct modelling* approach (Jordan and

Wolpert, 1999).

Since the true forward model is ordinarily a single-valued function, assigning a single consequence $\boldsymbol{y}$ for every possible input $\boldsymbol{x}$, most function approximation methods can in principle be applied to build an internal model of the forward relation. There are however some exceptions: the forward kinematic model of a parallel manipulator, for instance, typically exhibits multiple possible solutions for the same configuration of the actuator variables (Merlet, 2006). To exactly know the position of the end-effector, for a given joint configuration, some additional information about the evolution of the robot state is required. In a broader perspective, multi-modality on the forward model may occur whenever some relevant information about the robot or environment is not taken into consideration in the internal state representation $\boldsymbol{s}_t$. This makes the learning of the forward model a much more difficult task, as many learning algorithms are not prepared to deal with multi-valued outputs. All these issues will be covered in detail in the next section.

Forward models are used to make predictions about the evolution of the environment and to anticipate the outcomes of actions. This can be used, for instance, to replace actual sensory feedback in noisy and time delayed systems, to cancel the sensory effects of the movement of the robot sensors or to provide expected actions outcomes that can be used for planning (Miall and Wolpert, 1996; Wolpert, Ghahramani, and Flanagan, 2001). To control a robot, if only a finite set of discrete actions exist, the forward model can be used to choose the action $\boldsymbol{a}_t$ that will drive the current robot state closest to the desired state. This dissertation, however, is primarily concerned with continuous actions: in this case, a controller can exploit a forward model by considering the variation on the output that a change of the action vector induces, *i.e.*, by taking the derivative of the output $\boldsymbol{s}_{t+1}$ with respect to the action vector $\boldsymbol{a}_t$. This quantity is known as the *Jacobian* of the transformation represented by the forward model and has an extensive set of applications on robotic domains. On a kinematic level, for instance, the Jacobian of the forward model can be inverted to provide the variation on the joint quantities that would produce a desired change in the position of the end-effector. This concept is the basis for *resolved motion rate control* (RMRC) (Whitney, 1969), a form of robotic control scheme that acts at a velocity level in the joint and task spaces. This and other control schemes will be further detailed in Chapter 5.

Resolved motion rate control performs a local inversion of the linearized forward model at the current state $\boldsymbol{s}_t$ (Whitney, 1969; Salaün, Padois, and Sigaud, 2010). The forward kinematics is usually a highly nonlinear function: consequently, such inversion cannot be used to move directly to the desired task space position, and the command vector $\boldsymbol{a}_t$ must be permanently updated as the state vector $\boldsymbol{s}_t$ evolves. Since a controller, in a broad perspective, needs to choose an action that will lead to a desired state, it is natural that, instead of locally inverting a forward model, an *inverse model* of the environment is

instead considered, as will be discussed next.

2.1.2 Inverse Models

An inverse sensorimotor model is represented by the relation

$$\boldsymbol{a}_t = f^{-1}(\boldsymbol{s}_{t+1}, \boldsymbol{s}_t) \; ; \tag{2.2}$$

this is the same model as the forward model presented in equation (2.1), with the roles of $\boldsymbol{a}_t$ and $\boldsymbol{s}_{t+1}$ reversed, and thus can be seen as a transformation from a desired next state $\boldsymbol{s}_d = \boldsymbol{s}_{t+1}$ to motor controls $\boldsymbol{a}_t$. For open-loop control systems, inverse models are fundamental, as they provide a way to convert desired states to corresponding motor commands that will lead to such states, without relying in any kind of sensory feedback.

Some sensorimotor forward models can be described by injective functions, and thus the corresponding inverse models are still proper functions: this happens, for instance, in forward and inverse dynamics models. In this case the direct modelling method can be readily applied to control (Miller, 1987; Kuperstein, 1988), as illustrated in Figure 2.1. Nevertheless, learning inverse models is in general a more difficult task than learning

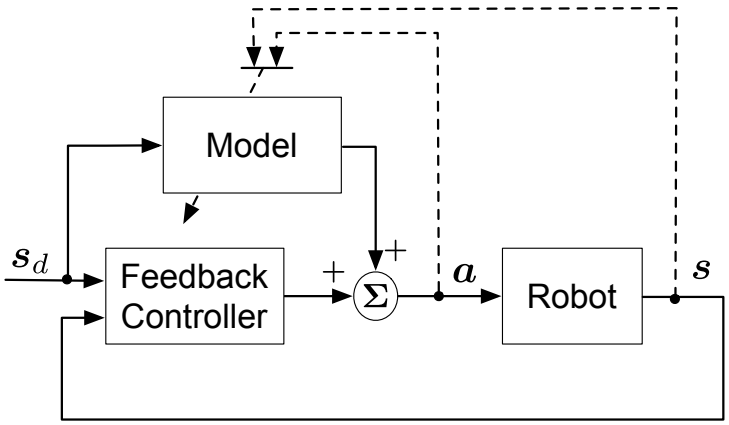


Figure 2.1: The direct modelling approach applied to inverse model learning, used together with a feedback controller. Figure taken from (Nguyen-Tuong and Peters, 2011b).

forward models, due to the model redundancy or the *many-to-one* characteristic of the forward mapping. As a consequence, the inverse model is not a proper function, and thus it cannot be adequately learned resorting to the direct modelling approach, where the training points consist of inputs $\boldsymbol{x}_t = (\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$ and outputs $\boldsymbol{y}_t = \boldsymbol{a}_t$ — note the reversed roles of $\boldsymbol{s}_{t+1}$ and $\boldsymbol{a}_t$, when compared to forward model learning.

Feedback error learning uses an external feedback controller to guide the inverse model learning process (Kawato, Furukawa, and Suzuki, 1987). Contrary to direct modelling, learning has to be done online, during the operation of the robot. The actual motor commands sent to the robot are the sum of the feedback controller output and $\boldsymbol{a}_t$, the

current output of the inverse model, as depicted in Figure 2.2. The fundamental point of

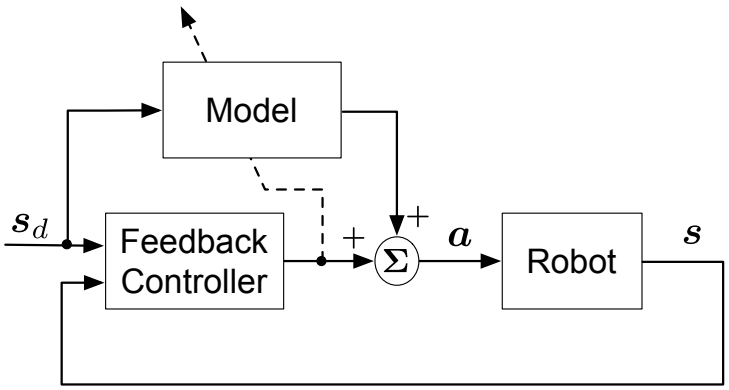


Figure 2.2: The feedback error learning approach to inverse modelling. Figure taken from (Nguyen-Tuong and Peters, 2011b).

feedback error learning is that the output of the feedback controller provides an error signal that is used to adjust the internal parameters of the inverse model: this can be interpreted as providing the commands that are sent to the robot as targets, or *desired responses*, for the inverse model. In such situation the output of the external feedback controller corresponds to the prediction error made by the inverse model, *i.e.*, the difference between the actual motor commands $\boldsymbol{a}_t$ and $\hat{\boldsymbol{a}}_t$, the current output of the inverse model: as learning proceeds the inverse model output converges to the desired robot commands and the feedback error goes to zero. This means that, after the learning process has converged, the inverse model effectively replaces the feedback controller, which nevertheless can still be used to compensate unwanted external disturbances.

Feedback error learning is *goal-oriented*, as opposed to direct modelling. This means that the learning process is influenced by the desired state for the robot: the difference between this state and the actual state is used to guide the learning process via the feedback controller, thus choosing and learning a specific solution for the inverse model. Another goal oriented approach is known as *distal supervised learning* (Jordan and Rumelhart, 1992): here, the multi-modality of the output of the inverse model is circumvented by the use of a previously trained internal forward model of the robot. This forward model is used in a composite controller, where the output of the inverse model being learned feeds the already trained forward model. In this way, the input of the composite controller is the next step desired state $\boldsymbol{s}_d$, while its output is the state predicted by the forward model, $\hat{\boldsymbol{s}}_t$, as illustrated in Figure 2.3. During training the forward model is kept fixed, and only the inverse model is adapted using the *performance error*, the difference between desired and actual observed states. The key point here is that the composite controller should behave as an identity system, mapping desired states into actual robot states: since only the inverse model is allowed to adapt, in the end the inverse model generates inputs $\boldsymbol{a}_t$ that will drive the robot state to the desired values.

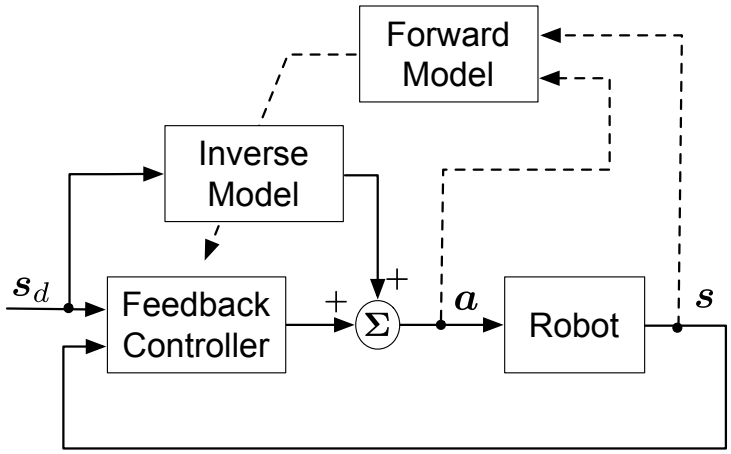


Figure 2.3: The distal learning approach to inverse modelling. Figure taken from (Nguyen-Tuong and Peters, 2011b).

Even if an imperfect forward model is used, the inverse controller in a distal learning scheme is able to track accurately the desired robot states, as its adaptation is made resorting to the performance error, not taking the output of the learned forward model into account. In spite of that, the existence of an approximate forward model, learned from training data, is still crucial for obtaining the inverse model, as it provides the information about how a variation of the action $\boldsymbol{a}_t$ can change the robot state — this is essential to accomplish a successful adaptation of the inverse model (Jordan and Wolpert, 1999). As feedback error learning, the distal learning approach is only able to find a single solution for the inverse model, that depends on the adaptation process and the particular temporal sequence of training data. If a particular type of inverse model is sought, additional penalty terms can be added to the cost function being minimized by the learner (Jordan, 1992).

Besides the feedback error and distal learning approaches, other learning algorithms also have goal-oriented procedures to find a way around the ill-posedness and multi-modality nature of the inverse model. These methods only find a single solution for the inverse problem, and redundancy resolution is usually achieved by introducing some kind of penalty on the cost function used to train the model. This technique is used, for instance, in learning inverse kinematics models (D'Souza, Vijayakumar, and Schaal, 2001) or to learn internal models of the full operational space mappings (Peters and Schaal, 2008).

Finally, another type of architecture uses the mixture of experts concept, described in detail in Section 2.2.6, to introduce a modular approach to the inverse model learning. This model, termed *modular selection and identification for control* (MOSAIC) by Wolpert and Kawato (1998), consists of a set of inverse models paired with corresponding forward models. The action $\boldsymbol{a}_t$ provided by the global model is given by a subset of the inverse models set, according to the prediction error made by the corresponding forward model:

the better a forward model is at predicting the current state of the robot, the more importance has the associated inverse model, as given by the *responsibility* signal of the forward model. This formulation allows, for the same desired robot output, the existence of different predictions made by the model, according to the specific current context. The responsibility signal is also used during learning, to assign training samples to the paired models that are more likely to correspond to the current context. The MOSAIC model has a strong inspiration on internal models of the human brain (Wolpert, Miall, and Kawato, 1998), and a training procedure and some simple example applications have been demonstrated in the work of Haruno, Wolpert, and Kawato (2001). Its architecture is shown in Figure 2.4.

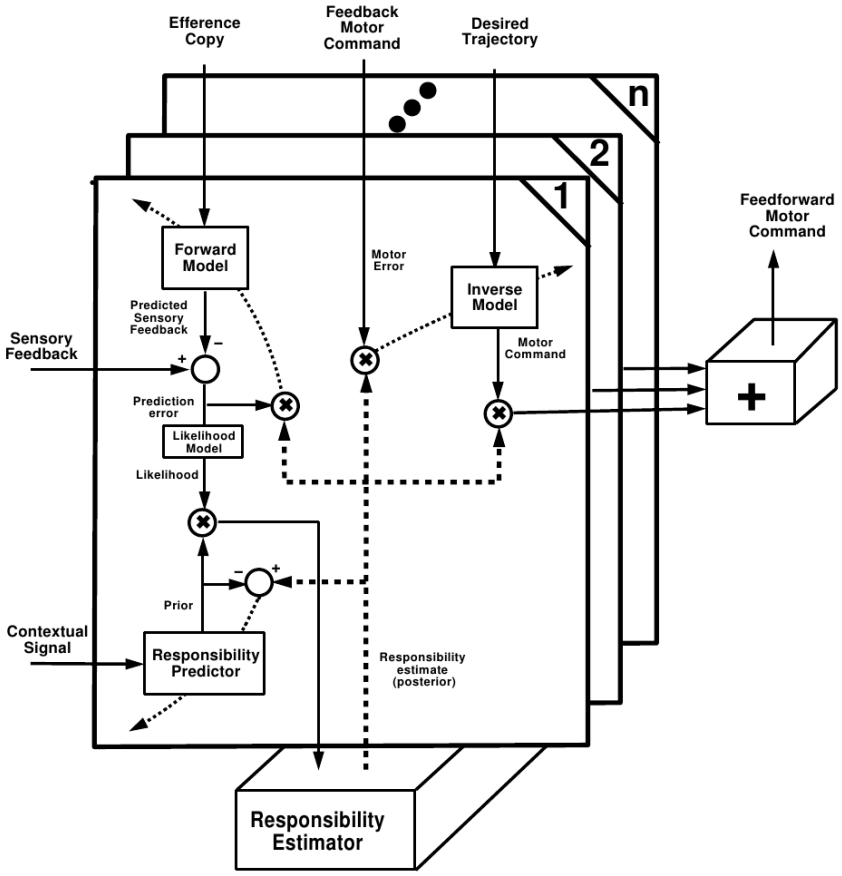


Figure 2.4: The MOSAIC architecture. Figure taken from (Wolpert and Kawato, 1998).

An interesting aspect of inverse model learning is that it must resort to direct modelling algorithms to train some of its components, even if these strategies were initially developed to avoid the direct use of this technique to the inverse model learning problem. As a consequence, direct modelling approaches, that make use, in a typical *regression* context, of input-output training data $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ to adapt the internal models, are essential to both forward and inverse model learning. Learning such models can be done using a large variety of procedures, as it will be described in the next section.

2.2 Learning Methods for Sensorimotor Models

In a regression setting, a supervised learning algorithm can be seen as a function mapping a training set $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$ to an estimate, or a probability distribution, of an unknown relation $\boldsymbol{y} = f(\boldsymbol{x})$. This estimate can be subsequently used to provide predictions at input locations different from the training data: such ability to *generalize* beyond the training data is a characteristic of every learning algorithm.

In order to perform output generalization at unseen inputs, a learning algorithm must adopt, implicitly or explicitly, a preference that favors some classes of functions, known as the inductive bias of a learning algorithm (Mitchell, 1997). Linear regression (Section 2.2.1), for instance, strongly restricts the function to estimate by considering it to be a member of the class of linear functions; Gaussian process regression (Section 2.2.4), on the other hand, assumes a correlation between outputs $\boldsymbol{y}_i$ that depends on the distance between corresponding inputs $\boldsymbol{x}_i$, according to some distance metric. The success and good generalization error of a learning algorithm crucially depends on its inductive bias doing a good job approximating the structure and the regularities of the true function to be learned. Without any restriction on the class of possible true functions, responsible for generating the observed training set $\mathcal{D}$, a learning algorithm has no means of choosing a plausible outcome $\boldsymbol{y}$ at a new input $\boldsymbol{x}$, and its performance with respect to generalization error will be equal to every other algorithm, including random prediction, a situation coined as the *no free lunch theorem* by Wolpert (1996).

Learning algorithms can roughly be divided into *parametric* and *nonparametric* models of the training data. In the first class a specific structure is assumed for the function to approximate, governed by a finite set of parameters: this structure can be though of as the inductive bias of the learning algorithm. Given these parameters — or their current probability distributions — a prediction can be performed without the need to resort to the training data. The linear regression model (Section 2.2.1) and adaptive expansions of a fixed number of basis (Section 2.2.3) are two examples of parametric learning models.

Nonparametric models, on the other hand, do not impose any particular structure to the function being approximated: they can be seen as possessing an infinite number of parameters, and typically depend on the full training dataset $\mathcal{D}$ to perform predictions (Ghahramani, 2013). Memory based learning (Section 2.2.2) and kernel machines (Section 2.2.4) fall in this class of learning methods, where the inductive bias is usually set by defining a structure for the kernel that represents the covariance between output samples or, in memory based learning, the distance metrics that measure the influence of training points in the prediction.

A third class of learning algorithms consists of parametric models whose structure is not completely defined *a priori*: while still assuming a parametric structure for the function to be learned, that structure itself (*e.g.* number of components in a mixture

model or number of basis functions in adaptive basis expansions) is allowed to adapt during the learning process; they can still be interpreted as parametric, but taking a further step that extends the class of functions they are able to represent and somewhat putting them a bit closer to nonparametric approaches.

Parametric algorithms have a set of parameters that are adjusted to make their predictions closely match the provided training examples. This can be done for instance by risk minimization, by defining a loss function over the prediction errors produced by the algorithm and choosing the parameters that minimize such loss. A probabilistic approach considers instead the maximization of the likelihood of the training data, where the parameters of the model are chosen to maximize the plausibility of the training data given the current probabilistic model: this is known as *maximum likelihood* (ML) inference, and the prediction distribution at a query point $\boldsymbol{x}_q$ is given by

$$p(\boldsymbol{y}|\boldsymbol{x}_q, \mathcal{D}) = p(\boldsymbol{y}|\boldsymbol{x}_q, \hat{\boldsymbol{\Theta}}) \ , \tag{2.3}$$

where $\hat{\boldsymbol{\Theta}}$ is the value of the parameter vector that maximizes $p(\mathcal{D}|\boldsymbol{\Theta})$, the likelihood of the training dataset given the parameter vector $\boldsymbol{\Theta}$.

A closely related training method treats the parameters as random variables, seeking the parameter values that maximize their posterior distribution after seeing the training data, a training process that is known as *maximum a posteriori* (MAP) inference. The prediction distribution for MAP inference is still given by equation (2.3), but now $\hat{\boldsymbol{\Theta}}$ corresponds to the value that maximizes the posterior distribution of $\boldsymbol{\Theta}$, $p(\boldsymbol{\Theta}|\mathcal{D})$.

Finally, a full Bayesian approach to the training procedure uses the full *a posteriori* distributions for the parameters in the prediction phase — as opposed to the use of point estimates of the parameters, as in MAP inference. This takes into consideration the uncertainty in the parameters values, and the bias introduced by the priors for these parameters helps guarding against overfitting. Using this inference technique, a prediction distribution at a query point $\boldsymbol{x}_q$ is obtained through marginalization on the model parameter vector $\boldsymbol{\Theta}$, according to

$$p(\boldsymbol{y}|\boldsymbol{x}_q, \mathcal{D}) = \int p(\boldsymbol{y}|\boldsymbol{x}_q, \boldsymbol{\Theta}) p(\boldsymbol{\Theta}|\mathcal{D}) d\boldsymbol{\Theta} \ . \tag{2.4}$$

As for nonparametric models, their training phase usually consists, due to the inherent lack of parameters, in automatically making choices about the quantities that define the inductive bias introduced in the learning process, responsible for the overall complexity of the learned model. Such quantities of interest can be, for instance, the penalty parameter of smoothing splines or, for kernel based methods like memory based learning and kernel machines, the *hyperparameters* of the kernel functions that define the characteristic

input length-scale and/or the output noise of the function to learn. Parametric methods with varying structure, on the other hand, often have the need to autonomously adjust the complexity of the underlying function: an emblematic example is the choice of the number of components in a mixture model (Section 2.2.6).

The more complex a model is the better it will fit the training data, as the function space generated by the learning algorithm is less constricted. However, most of the times the error in an independent test set will start to increase when the model complexity is increased too much: this is known as *overfitting*, an undesired phenomenon that occurs when a learning algorithm specializes too much on the training data. As the complexity of the model increases there is a larger set of hypothesis for the function that generated the training data, and the learning algorithm will be able to pick the ones that provide the best approximating error, thus reducing its bias over the training set. Nonetheless, the variance of the learning algorithm, measured over all possible training sets, will increase, as it reflects the sensitivity of the resulting approximated function to a particular choice of $\mathcal{D}$. This can increase the prediction error at a random test point $\boldsymbol{x}_q$, due to the following well-known decomposition of the prediction error into learner bias and variance:

$$\mathbb{E}[\underbrace{(\hat{f}(\boldsymbol{x}_q) - f(\boldsymbol{x}_q))^2}_{\text{squared error}}] = \underbrace{(\mathbb{E}[\hat{f}(\boldsymbol{x}_q)] - f(\boldsymbol{x}_q))^2}_{\text{bias}^2} + \underbrace{\mathbb{E}[(\hat{f}(\boldsymbol{x}_q) - \mathbb{E}[\hat{f}(\boldsymbol{x}_q)])^2]}_{\text{variance}} . \tag{2.5}$$

Here $f(\boldsymbol{x}_q)$ is the true function value at $\boldsymbol{x}_q$ while $\hat{f}(\boldsymbol{x}_q)$ is the learner prediction; expectations are taken over all possible training sets. While increasing the complexity of the learning algorithm helps reducing its learning bias, the variance nonetheless will get larger and start to dominate the bias term, increasing the overall expected prediction squared error. This behaviour, depicted in Figure 2.5, is known as the *bias/variance trade-off* (Geman, Bienenstock, and Doursat, 1992), and learning algorithms try to set their complexity in such a way that an optimal balance between bias and variance is found, resulting in a low generalization error in an independent test set.

Unfortunately, the variance of a learner cannot be assessed based on the training error alone: for this reason a number of techniques were developed to estimate the generalization error of a learning algorithm from the training data:

Cross Validation randomly splits the training set in two parts, one for training the learning algorithm and the other for assessing the generalization error. Training is then repeated, with another portion of the data used for the validation set, until all data is cycled and all points are used once as validation data (Stone, 1974). The average prediction error on the validation set is then a good estimate of the true generalization error, and can be used to perform model selection or choosing a parameter from a discrete set of possible values. In the extreme case where only

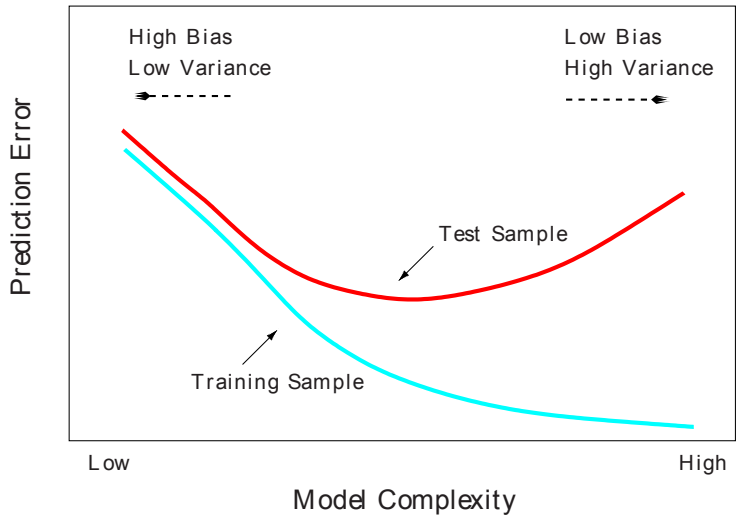


Figure 2.5: The bias/variance trade-off. Figure taken from (Hastie, Tibshirani, and Friedman, 2009).

a sample, out of a training set with M points, is used at a time to evaluate the *out-sample* error, the training procedure has to be repeated M times: this is known as *leave-one-out* cross-validation.

Bayesian Evidence takes advantage of the fact that a more complex probabilistic model m will spread its predictive probability mass over a larger set of possible hypothesis for the underlying function, thus reducing its probability density for a single parameter value (Ghahramani, 2013). Evidence is also known as marginal likelihood due to the fact that the quantity $p(\mathcal{D})$ is used for model comparison, where the parameters $\boldsymbol{\Theta}$ are marginalized out. Evidence maximization automatically provides a bias-variance trade-off, since it tends to penalize more complex models by assigning them less probability, as represented in Figure 2.6. This phenomenon is known as the *Bayesian Occam's razor*, where the simplest models that fit the training data well are chosen (MacKay, 2003).

Penalty Methods impose, during training, a penalty that discourages overly complex models. The *Akaike information criterion* (AIC) (Akaike, 1974) introduces a penalty that linearly grows with the number of independent parameters of a model, based on a information-theoretical approach that resembles the entropy maximization principle in thermodynamics. The *Bayesian information criterion* (BIC) (Schwarz, 1978) tends to inflict a heavier penalty on complex models when compared to AIC, and it is derived as an approximation to the Bayesian evidence described above. Other methods take an approach based on information and theory of coding to derive expressions for the penalty, like the *minimum description length* (MDL) (Rissanen, 1978), that can be shown to have a strong link to BIC (Hastie, Tibshirani, and

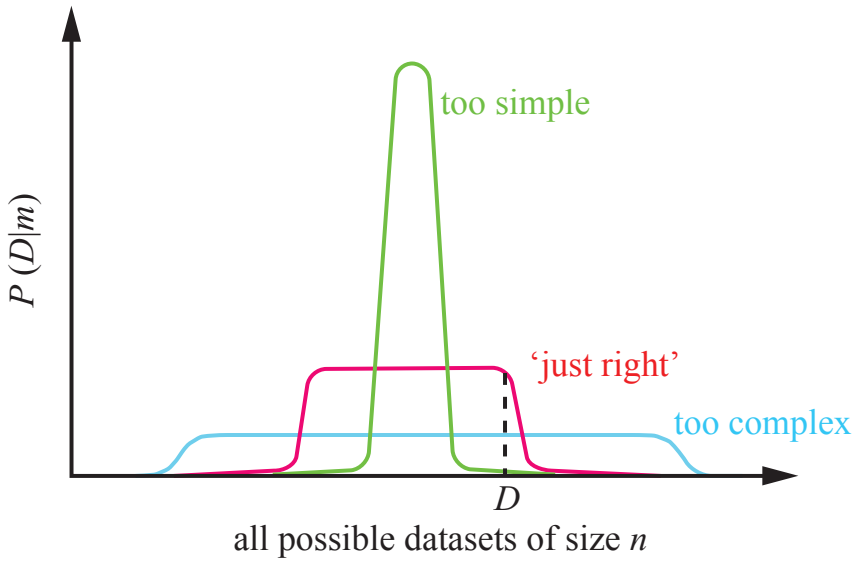


Figure 2.6: The Bayesian Ockam's Razor. Figure taken from (Ghahramani, 2013).

Friedman, 2009), and the *minimum message length* (Wallace and Boulton, 1968).

The remaining of this section will attempt to provide a short review of some techniques used for regression and function approximation, with a special focus on the issues that may have more impact in a robotics sensorimotor learning context.

2.2.1 Linear Regression

Perhaps one of the most simple methods for function approximation is the linear regression model, that assumes a scalar output y to be related to an input vector $\boldsymbol{x} = [x_1,\ x_2,\ \cdots, x_d]^T$ according to a linear relation, corrupted with noise ε:

$$y = \beta_0 + \sum_{k=1}^{d} \beta_k x_k + \varepsilon \ . \tag{2.6}$$

In this linear regression problem an estimate for the unknown regression coefficients β_k is sought that correctly explains the training data $(\boldsymbol{x}_i, y_i)$, for $i = 1 \ldots N$. Although a scalar output y is considered in the following text, the linear regression model can be easily extended to multivariate outputs. Figure 2.7 illustrates the linear regression model concept. Assuming a zero mean, *i.i.d.* Gaussian model for the noise, the likelihood of the training set can be written as

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta}) = \prod_{i=1}^{N} p(y_i|\boldsymbol{x}_i, \boldsymbol{\beta}) \sim \mathcal{N}(\boldsymbol{X}^T\boldsymbol{\beta}, \sigma_n^2\mathbf{I}) \ , \tag{2.7}$$

where $\boldsymbol{X}$ is a $(d+1) \times N$ matrix whose columns correspond to the training points $\boldsymbol{x}_i$, augmented with an initial element equal to 1, $\boldsymbol{y}$ is a vector collecting the targets y_i and

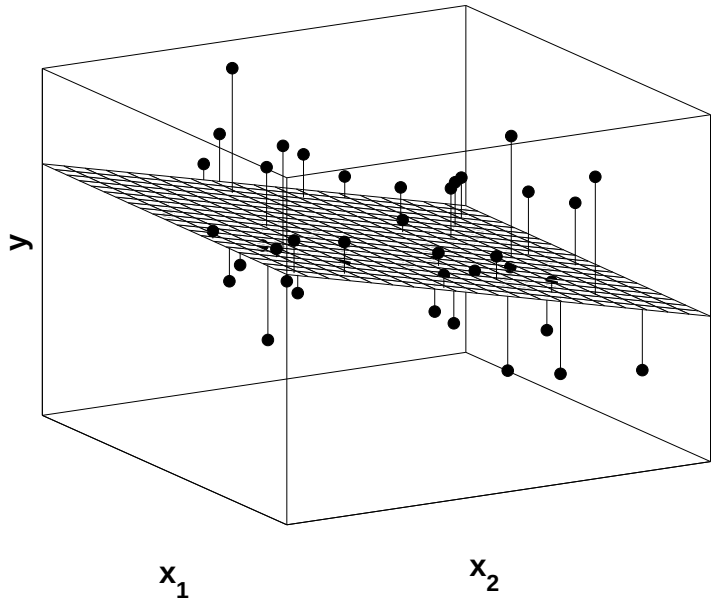


Figure 2.7: The linear regression model fits a hyperplane that minimizes the sum of squared approximation errors of the training data.

$\boldsymbol{\beta}$ is the $(d+1)$-dimensional vector of regression coefficients β_k, for $0 \leq k \leq d$.

With some algebraic manipulation, the likelihood in equation (2.7) can be arranged as a Gaussian distribution in the regression vector $\boldsymbol{\beta}$, resulting in (Bishop, 2006)

$$p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}\left((\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{y},\ \sigma_n^2(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\right)\ ; \tag{2.8}$$

as a consequence, the output prediction for an input query $\boldsymbol{x}_q$ has the distribution

$$p(y|\boldsymbol{x}_q, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{x}_q^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{y},\ \sigma_n^2\left(1 + \boldsymbol{x}_q^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\boldsymbol{x}_q\right)\right)\ . \tag{2.9}$$

This is a well known result, and the mean of the posterior distribution for $\boldsymbol{\beta}$ coincides with the *ordinary least squares* (OLS) solution for the linear regression problem, where the regression vector $\boldsymbol{\beta}$ is chosen to minimize the residual sum of squares

$$RSS(\boldsymbol{\beta}) = \sum_{i=1}^{N}\left(y_i - \beta_0 - \sum_{k=1}^{d} x_{ik}\beta_k\right)^2 = (\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta})\ . \tag{2.10}$$

The above OLS solution, however, presents some drawbacks: in high dimensional input spaces the OLS will typically produce high variance estimates, meaning that there is a risk of overfitting, specializing the regression coefficients on the noise. Also, in many datasets arising from robotic sensorimotor relations, the input values may be heavily underconstrained, with irrelevant or highly correlated input dimensions: this means that the full rank assumption on $\boldsymbol{X}\boldsymbol{X}^T$ may be violated, causing numerical instabilities in the required matrix inversion (Ting, Mistry, and Peters, 2006).

To tackle these problems, a large number of approaches were proposed to reduce the effective number of input dimensions needed to correctly explain the linear relation between input vectors and corresponding target responses, and thus reduce the variance in the regression coefficients estimates. Some of these methods are briefly presented bellow:

Principal Component Regression (PCR) replaces the original regressor variables by their principal components[1], discarding the smallest eigenvalue components (Kendall, 1957). This orthogonalizes the regression problem and makes computations more numerically stable.

Partial Least Squares Regression (PLS) (Wold, 1975), a technique widely used in chemometrics, performs univariate regressions along the input directions that exhibit the strongest correlation between input and output — note that PCR above only looks at the input correlation (Frank and Friedman, 1993), and thus can potentially eliminate low variance input directions that can, nevertheless, have a strong influence on the output response.

Ridge regression (also known as *Tikhonov regularization*) (Hoerl and Kennard, 1970) shrinks the regression coefficients by imposing a penalty on their magnitude, by changing the cost (2.10) to $(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|^2$. This results in a solution $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\mathbf{I})^{-1}\boldsymbol{X}\boldsymbol{y}$. Alternatively, ridge regression can be seen as defining a prior Gaussian distribution for $\boldsymbol{\beta}$, with zero mean and $\lambda\mathbf{I}$ covariance, *i.e.*, $p(\boldsymbol{\beta}) = \mathcal{N}(\mathbf{0}, \lambda\mathbf{I})$. In this case the above solution for $\boldsymbol{\beta}$ coincides with the posterior mean for the regression coefficients[2].

LASSO, or Least Absolute Shrinkage and Selection Operator (Tibshirani, 1996), considers a L_1 penalty on the coefficients, instead of the L_2 penalty used in ridge regression. It can be shown that this induces sparsity on the coefficients $\boldsymbol{\beta}$, making a subset of them to go exactly to zero. In a probabilistic sense the LASSO can be interpreted as putting a Laplace prior on the regression coefficients. This prior can be represented in a hierarchical manner, as done by Figueiredo (2003), which opens the door to the use of different hyperpriors, such as the uninformative Jeffreys prior that can achieve a sparse solution without any hyperparameter to tune (Figueiredo, 2003). Several efficient methods for computing the LASSO solutions exist, such as the least angle regression and shrinkage (LARS) (Efron, Hastie, et al., 2004); Bayesian versions of LASSO can provide interval estimates for the regression parameters (Park and Casella, 2008).

Subset Regression retains only a subset of the input variables that best explains the training data, using this subset to obtain the OLS solution. As the two previous

[1] The principal component analysis will be further detailed in Section 2.2.5.

[2] Since, in general, a shrinkage of the intercept β_0 is not desired, ridge regression and some of these methods estimate instead the regression vector with the intercept β_0 excluded, calculating this coefficient separately.

methods, it can be seen as imposing a penalty on the norm of the coefficients vector $\boldsymbol{\beta}$, but this time a L_0 norm is used for regularization. From a probabilistic point of view this equivalent to using a hierarchical Bernoulli-Gaussian model (Soussen, Idier, et al., 2011). The selection of the best subset of predictors can be done using several different techniques, like the *leaps and bounds* procedure (Furnival and Wilson, 1974) or stepwise regression (Derksen and Keselman, 1992).

Sparse Bayesian Learning (SBL) (Tipping, 2001) takes an empirical Bayes perspective to the regression problem: its probabilistic model is similar to ridge regression, but the prior covariances on each of the regression coefficients β_k are assumed different and unknown. These are estimated from the marginal distribution for the data, where the vector $\boldsymbol{\beta}$ is integrated out, and then these estimates are used to obtain the posterior distribution for the regression coefficients. It can be shown that the resulting estimate for vector $\boldsymbol{\beta}$ becomes sparse: in the context of neural networks (Section 2.2.3), these technique is known as *Automatic Relevance Determination* (ARD) (Neal, 1996; MacKay, 1995). The *Variational Bayesian Least Squares* (VBLS) approach proposed in Ting, D'Souza, et al. (2010) can be seen as a computationally efficient version of SBL, that can be applied to high dimensional problems in a real-time environment: this can be particularly useful for robotic sensorimotor learning tasks (Ting, Kalakrishnan, et al., 2009).

Since many robotic sensorimotor maps are highly nonlinear, some concerns about the pertinence and applicability of the linear regression model may naturally arise. However, note that in some problems the use of a linear model is sufficient: an example is the rigid body dynamic (RBD) formulation of a robot dynamics, where all the robot unknown parameters appear linearly in the rigid body dynamics relations (Atkeson, An, and Hollerbach, 1986; An, Atkeson, and Hollerbach, 1988). In this case, parameter identification can be done resorting to the linear regression model, although some issues may arise as there is no mechanism to ensure physical correctness of the estimated inertial parameters, *e.g.*, positive mass parameters (Nakanishi, Cory, et al., 2008). Even when the linear assumption does not hold, the linear model can be used as part of more sophisticated methods, like local learning (Section 2.2.2) or mixture models (Section 2.2.6). Also, many kernel methods (Section 2.2.4) perform a linear regression in a very high (possibly infinite) dimensional feature space, implicitly defined by the kernel function.

Finally, note that of special importance for sensorimotor learning is the ability of these linear regression methods to cope well with online learning schemes, in order to be able to deal with large volumes of sensorimotor data. It is well known that the OLS solution can be computed exactly in a recursive fashion (Ljung, 2002); in order to be gracefully adapted to robotic problems, other linear regression methods should also provide some kind of incremental schemes.

2.2.2 Memory Based Methods

Linear regression models can be seen as global parametric models, that approximate the underlying function by a global model, parametrized by a linear relation from inputs to their corresponding response. Memory based methods, on the other hand, are non-parametric methods that generate local predictions based on the training samples in the vicinity of the query point; this kind of learning scheme is also known as *lazy learning* or *instance based learning* (Mitchell, 1997), and is characterized by the lack of an underlying parametric structure, where the predictions are obtained from a local approximation based on the training data.

The simplest memory based method for regression is the well known nearest neighbour approximator, that simply predicts, for a query point $\boldsymbol{x}_q$, the output $\hat{y}$ taken from the training sample whose input $\boldsymbol{x}_i$ is closest to $\boldsymbol{x}_q$. This estimate, however, has a large variance with respect to the training data and changes abruptly as the query point $\boldsymbol{x}_q$ is changed. A natural extension is then to consider the k-nearest neighbours method, that generates a prediction by averaging the responses corresponding to the k training points closest to the query point[3], as given by

$$\hat{y}(\boldsymbol{x}_q) = \frac{1}{k} \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x}_q)} y_i \ , \tag{2.11}$$

where $N_k(\boldsymbol{x}_q)$ is the k-neighbourhood of $\boldsymbol{x}_q$ in the training sample.

As the number of neighbours increases the prediction variance is reduced, but at a cost of an increasing bias. In the limit, if all data points are considered, a global, constant prediction is obtained that simply corresponds to the average of all training points. Choosing the optimal number of neighbours to use is thus the fundamental question concerning this method, and can be considered as the training phase of the learning algorithm. This is a manifestation of the classical bias-variance dilemma, and a common approach in this case is to choose the neighbourhood size resorting to cross-validation techniques.

Smoothing kernels can make the nearest-neighbours prediction continuous by using a weighted average of the training points, where lower weights $w_i(\boldsymbol{x}_q)$ are given to training points farther from the query point, according to the Nadaraya-Watson formula (Nadaraya, 1964; Watson, 1964):

$$\hat{y}(\boldsymbol{x}_q) = \sum_{i=1}^{N} w_i(\boldsymbol{x}_q) y_i \ , \quad \text{with} \qquad w_i(\boldsymbol{x}_q) = \frac{k(\boldsymbol{x}_q, \boldsymbol{x}_i)}{\sum_{j=1}^{N} k(\boldsymbol{x}_q, \boldsymbol{x}_j)} \ . \tag{2.12}$$

The function $k(\boldsymbol{x}_q, \boldsymbol{x}_i)$ is a smoothing kernel, a positive, decreasing function of the distance $\|\boldsymbol{x}_q - \boldsymbol{x}_i\|$. Many different kernels can be used, and most of them can be written

[3]Sometimes, instead of defining the number of neighbours one chooses instead the radius of a hypersphere centred on $\boldsymbol{x}_q$, averaging the responses of the training points that fall inside that hypersphere.

as $k(\boldsymbol{x}_q, \boldsymbol{x}_i) = k(d)$, where $d = \|\boldsymbol{x}_q - \boldsymbol{x}_i\|/\lambda$ is a distance function; the *squared-exponential* kernel (also known as the Gaussian kernel), for instance, is given by $k(d) = \exp(-0.5d^2)$, while the *Epanechnikov* and the *tri-cube* kernels are given respectively by

$$k(d) = \begin{cases} (1 - d^2) & \text{if } d \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

and

$$k(d) = \begin{cases} (1 - d^3)^3 & \text{if } d \leq 1 \\ 0 & \text{otherwise} \end{cases} . \tag{2.14}$$

These are only some examples of popular kernels, and, in general, the particular choice for the weighting kernel is not critical; other choices for $k(d)$ are described, for instance, in (Atkeson, Moore, and Schaal, 1997a).

The smoothing parameter λ defines the width of the kernel, implicitly defining the range over which a training point contributes to a prediction: large values correspond to a broadener neighbourhood being considered, with many training points effectively used in the prediction average in (2.12). This lowers the variance of the prediction, but increases its bias. On the other side, if small values of λ are used the bias vanishes, but the predicted function becomes more wiggly, starting to specialize over the training samples output noise.

The Nadaraya-Watson formula, like the k-nearest neighbours method, locally approximates the function to be learned by a constant, given by the weighted average of the training samples. *Local linear regression* extends this formulation by locally approximating the function by a linear relation (Cleveland, 1979). This results in the *weighted least squares* solution to the linear regression problem, given by

$$\hat{y}(\boldsymbol{x}_q) = \boldsymbol{x}_q^T (\boldsymbol{X}\boldsymbol{W}(\boldsymbol{x}_q)\boldsymbol{X}^T)^{-1}\boldsymbol{X}\boldsymbol{W}(\boldsymbol{x}_q)\boldsymbol{y} , \tag{2.15}$$

where the $N \times N$ weight matrix $\boldsymbol{W}(\boldsymbol{x}_q)$ is diagonal with ith diagonal element $k(\boldsymbol{x}_q, \boldsymbol{x}_i)$. The prediction is similar to the OLS solution, but now some weights are used to account the influence of the training points in the regression problem, according to the kernel being used (compare to the prediction mean in equation (2.9)). Figure 2.8 presents the reconstructed function based on the training data shown, using a Gaussian kernel with $\lambda = 0.2$. The local linear model obtained in the vicinity of a query point is also presented.

As with k-nearest neighbours, choosing an appropriate smoothing parameter can be done in several different ways: leave-one-out cross-validation is a popular choice, as a simple, closed form expression for its sum of squared residuals can be derived.

Note that higher order local polynomials can be fitted to the data, like quadratic

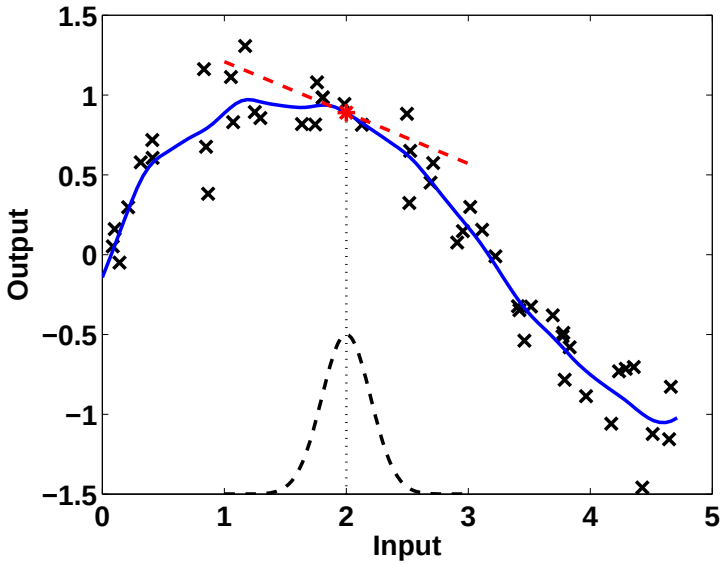


Figure 2.8: Locally weighted learning fitting of a set of noisy training data. The dashed line in the bottom of the figure shows the kernel weight around the query point $x = 2.0$, while the dashed straight line depicts the local linear approximation around the same point.

and cubic polynomials. Increasing the order of the fitted polynomial helps reducing the bias, while increasing the variance: choosing the degree for the polynomial is an open question, but in general lower order ones are preferred (Hastie and Loader, 1993; Hastie, Tibshirani, and Friedman, 2009). In practice, for robotic sensorimotor learning, local linear approximations are a good compromise between prediction error and computational complexity. For more details on memory based methods, Atkeson, Moore, and Schaal (1997a), Loader (1999), and Loader (2012) provide much information on this subject. Some applications to control and robotic applications can be found in (Atkeson, Moore, and Schaal, 1997b), where in particular it is described the use of memory based schemes for learning both forward and inverse sensorimotor relations.

However, two major problems may arise with memory based methods when real-time robotic applications are considered. The first is the need to keep all training points in memory: since the data acquisition process can be very fast, there is the danger of not being able to generate predictions in real-time, due to an increasingly large number of computations needed to calculate a prediction. In this case a solution is to keep a small enough representative subset of the training data, but this implies some care on choosing what points should be maintained. Also, erroneous training points kept in the training dataset, arising from outliers, may have a strong negative impact on the predictions.

Another problem is the fact that boundary effects become more noticeable in high dimensions, as discussed in Chapter 1: these nonparametric, memory based methods are particularly prone to these effects, and Loader (2012) suggests against using them for more than a couple of input dimensions.

More recently some other memory based approaches were proposed, with better flexibility and generalization error. One example is the Bayesian approach to local linear regression of Ting, D'Souza, et al. (2008), that locally finds the relevant input dimensions and automatically discards outliers. Gaussian process regression, described in Section 2.2.4, like the other memory based methods presented above, can also be viewed as a linear smoother, by means of its *equivalent kernel*, with the important distinction that its width automatically shrinks with the increase of density of training data (Silverman, 1984; Rasmussen and Williams, 2006). The excellent prediction performance and probabilistic principled approach of the Gaussian process regression method have in part contributed, in the last decade, to the loss of popularity of some of the memory based learning methods presented above in this section.

2.2.3 Adaptive Basis Expansions and Neural Networks

In general, the regressor vector in the linear regression model in Section 2.2.1 is not limited to the input vector $\boldsymbol{x}$: any transformations of these variables can be considered. For instance, polynomial basis expansions can be considered if the regressor vector is extended to accommodate all the interactions between input variables up to a given order, *e.g.*, $\boldsymbol{x} = [x_1,\ x_2,\ x_1^2,\ x_2^2,\ x_1 x_2]^T$ fits a quadratic relation in a 2-dimensional input space. Another well known basis expansion of the inputs is the piecewise polynomial fitting that can be done resorting to *B-splines* (De Boor, 1978). Overall, the basis expansion model can be written as

$$y = \beta_0 + \sum_{j=1}^{M} \beta_j \Phi_j(\boldsymbol{x}) \ . \tag{2.16}$$

However, the *basis functions* $\Phi_j(\boldsymbol{x})$ need to be set in advance and, as the input dimensionality grows, typically an increasingly large number of basis functions is required to cover the input space. The fact that the input training data typically lies in a space with a lower dimension than the input space (Schaal, Vijayakumar, and Atkeson, 1998), together with a lack of knowledge of the relation to be estimated, encourages the learning of the parameters that characterize the basis functions, instead of setting them beforehand. The class of *generalized additive models* is a particular case of adaptive basis expansions, where the basis Φ_j only operate along a particular input dimension, according to

$$y = \beta_0 + \sum_{j=1}^{M} \beta_j \Phi_j(x_j) \ ;$$

this model can be fitted by iteratively adapting each Φ_j in turn, using the current estimates for the other basis, until the procedure converges. This is known as the *backfitting* algorithm for additive models (Hastie and Tibshirani, 1986).

There is a vast literature on learning with *adaptive basis expansions*, and several differ-

ent learning methods can be viewed under this perspective: *classification and regression trees* (CART) (Breiman, Friedman, et al., 1984) and *multivariate adaptive regression splines* (MARS) (Friedman, 1991), for instance, fall in this category. Classification and regression trees recursively partition the input space by lines that are parallel to the input coordinate axes. This results in a set of mutually exclusive regions of the input space: the basis Φ_j then correspond to the indicator functions over these regions, while the coefficients β_j specify the constant predictions provided by the model in the corresponding regions. Multivariate adaptive regression splines, on the other hand, can be interpreted as a modification of CART that leads to an improved performance under a regression setting: it uses, for its basis functions Φ_j, a set of tensor products of regression splines to represent the target function to be approximated.

Another popular method chooses the basis functions to be $\Phi_j(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}_j)$, where $k(\boldsymbol{x}, \boldsymbol{x}_j)$ assigns a similarity measure that decreases with the distance to the basis centre $\boldsymbol{x}_j$; this function can be chosen as one of the kernels described in Section 2.2.2. This kind of basis expansions are known as *radial basis functions networks* (RBF) (Broomhead and Lowe, 1988; Buhmann, 2003): besides the weights β_j, the learning process usually may try to adapt the centre and parameters of the distance metric of each of the basis.

Artificial neural networks (NN), also known as multilayer perceptrons, are probably the most widely used adaptive basis methods for sensorimotor model learning. Their name comes from the artificial intelligence field, from the attempts to find a representation model for the information processing in the human brain (McCulloch and Pitts, 1943; Widrow and Hoffman, 1960; Rosenblatt, 1962). From a machine learning perspective, a feed-forward neural network consists of a series of layers of adaptive basis functions, where the output of a layer feeds the following layer. The first layer is known as the input layer, the last one is the output layer and all the remaining, intermediary layers are labeled as hidden layers. Figure 2.9 depicts the basic structure of a neural network with a single hidden layer, although a typical neural network may contain several of these.

Typically the first hidden layer projects the input data into a direction given by the weights vector $\boldsymbol{w}_{xz}$, and then performs a univariate nonlinear transformation, also known as the *activation function*, over the result of the projection, according to

$$z_k = \sigma(\boldsymbol{w}_{xz}^T \boldsymbol{x}) \ , \tag{2.17}$$

where z_k is the kth output of the first layer. This is a model that bears a strong resemblance to the *projection pursuit regression* model (Friedman and Stuetzle, 1981). For regression, the *sigmoid* activation function $\sigma(v) = 1/(1 + e^{-v})$ is commonly used. The final layer usually performs a linear regression on the intermediate units z_k, according to

$$y_i = \boldsymbol{w}_{zy}^T \boldsymbol{z} \ . \tag{2.18}$$

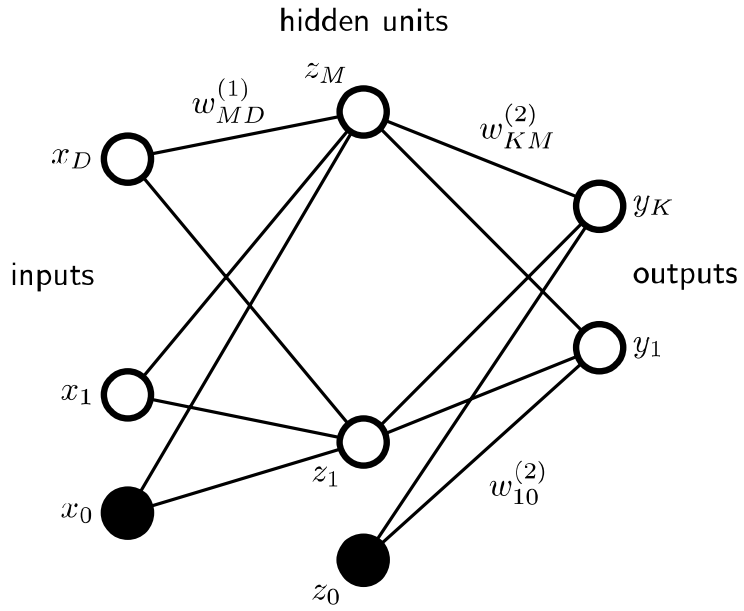


Figure 2.9: Network diagram for a feed-forward neural network with a single hidden layer. Figure taken from (Bishop, 2006).

Here, $\boldsymbol{z}$ denotes the vector that aggregates the variables z_k, corresponding to the latent variables in the hidden layer. It is customary to extend the input and intermediate layers to accommodate a constant term, known as the bias, as shown in Figure 2.9.

The training phase adjusts the weights of each layer in order to fit the network outputs to the training data. This can be done resorting to the popular *back-propagation* algorithm (Rumelhart, Hinton, and Williams, 1986), an efficient gradient descent algorithm for neural networks that has helped to boost a renewed interest in the neural networks architecture, back in the eighties. More recently, the *extreme learning machine* (Huang, Zhu, and Siew, 2006), which randomly chooses hidden nodes and analytically determines the output weights of the network, has shown to be much faster than the back-propagation learning algorithm.

This kind of neural networks is considered an universal approximator, capable of estimate an arbitrary continuous function to any given precision (Kreinovich, 1991), as long as the network has a sufficient number of hidden units. Also, such networks have the ability to perform automatic feature extraction, where the relevant features are implicitly represented in its hidden layers. The fact that the backpropagation algorithm can be carried out online is a pertinent advantage for robotic applications.

Training a neural network requires some attention regarding overfitting: one of the earliest methods to avoid an excessive specialization on the training data was known as *early stopping*, where training ended when no improvement was observed on the prediction error over an independent test data. *Weight decay* is an alternative approach, that consists in defining a regularization term on the loss function — from a Bayesian perspective, this

corresponds to the introduction of a prior on the neural network weights.

From a practical point of view, the choice of topology for the neural network — number of layers, number of units in each hidden layer and types of connections — may have a strong influence in the accuracy of the final model, and there are some techniques that automatically deal with this problem (Bishop, 2006). Another issue, of particular importance for sensorimotor learning, is the fact that standard neural networks are *global* approximation methods, where each unit influences the prediction result in very different regions of the input space. This is a problem when the input data distribution is not stationary, as typically happens when acquiring real-time training data from robotic devices: in this situation, catastrophic negative interference may occur, where recurrently training on a region of the input space may lead to the destruction of the model prediction abilities on other regions of the same space (Atkeson, Moore, and Schaal, 1997a).

Using neural networks to perform general function approximation has lost some of its popularity in the last decade, in favour of more recent nonparametric kernel methods, described in Section 2.2.4; in fact, it can be shown that by defining some priors over the network parameters, a neural network may converge to a Gaussian process (Neal, 1996).

Recurrent neural networks (RNN) are an alternative to the feed-forward neural networks described above: they have recurrent connections linking their units, responsible for the generation of feedback signals in the network. This recurrence leads to the existence of an implicit dynamical memory, coded in the hidden layers, and consequently to an internal state representation that stores contextual information. This architecture has significant advantages for the representation of time series and temporal evolution of dynamic systems: historically, this was the main reason for the creation of this kind of networks.

Since connections between units form directed cycles, the network behaves as a dynamical system, where complex, almost chaotic responses can be potentially obtained. Due to this behaviour, full training of the network, resorting for instance to gradient descent based methods to set the network weights, is traditionally much harder, with a computational expensive training phase and a lack of convergence guarantees.

To overcome these limitations, a new approach to RNN was pursued, consisting in the use of fixed random weights for the recurrent network. In this setting, only the network weights connecting to outputs are learned, while the rest of the network is left untrained. As a consequence, assuming the recurrent network to have a much larger number of units than the number of network inputs, a complex nonlinear transformation of the input signal into a high dimensional vector, encoded in the network hidden units, is achieved. This scheme has some resemblance to kernel methods discussed in Section 2.2.4: like these algorithms, a projection of the inputs is made into a high dimensional space, although in this case there is an explicit representation of that projection. Then, a simple linear regression is performed in that space to obtain the output. However, contrary to kernel

machines, the latent representation may depend also on the output values of the network and the previous values of its hidden units.

This idea was pursued independently in the pioneer works of Jaeger (2003), which developed the *echo state network*, and Natschläger, Maass, and Markram (2002), with their *liquid state machine*. During this period the *Backpropagation-Decorrelation* online algorithm was also presented by Steil (2004) to train these kind of networks. Later, it was proposed by Verstraeten, Schrauwen, et al. (2007) that the ideas supporting this type of networks should be unified under the name of *reservoir computing*. Under this paradigm, the part of the network with fixed weights is known as the *reservoir*.

In its most general representation, a reservoir network dynamics can be described by the following dynamic equations (Schrauwen, Verstraeten, and Van Campenhout, 2007):

$$\begin{aligned}\boldsymbol{z}(k+1) &= \boldsymbol{\sigma}(\boldsymbol{W}_{res}^{res}\boldsymbol{z}(k) + \boldsymbol{W}_{in}^{res}\boldsymbol{x}(k) + \boldsymbol{W}_{out}^{res}\boldsymbol{y}(k) + \boldsymbol{W}_{bias}^{res}) \\ \boldsymbol{y}(k+1) &= \boldsymbol{W}_{res}^{out}\boldsymbol{z}(k) + \boldsymbol{W}_{in}^{out}\boldsymbol{x}(k) + \boldsymbol{W}_{out}^{out}\boldsymbol{y}(k) + \boldsymbol{W}_{bias}^{out}\ .\end{aligned}$$

Here $\boldsymbol{W}_{res}^{res}$, $\boldsymbol{W}_{in}^{res}$, $\boldsymbol{W}_{out}^{res}$ and $\boldsymbol{W}_{bias}^{res}$ are the weight matrices of the links connecting the reservoir, input, output and bias units to the reservoir units. In the same manner, $\boldsymbol{W}_{res}^{out}$, $\boldsymbol{W}_{in}^{out}$, $\boldsymbol{W}_{out}^{out}$ and $\boldsymbol{W}_{bias}^{out}$ are the weight matrices of the links connecting the reservoir, input, output and bias units to the output. An example of such network architecture, as implemented in the work of Reinhart and Steil (2009), is depicted in Figure 2.10.

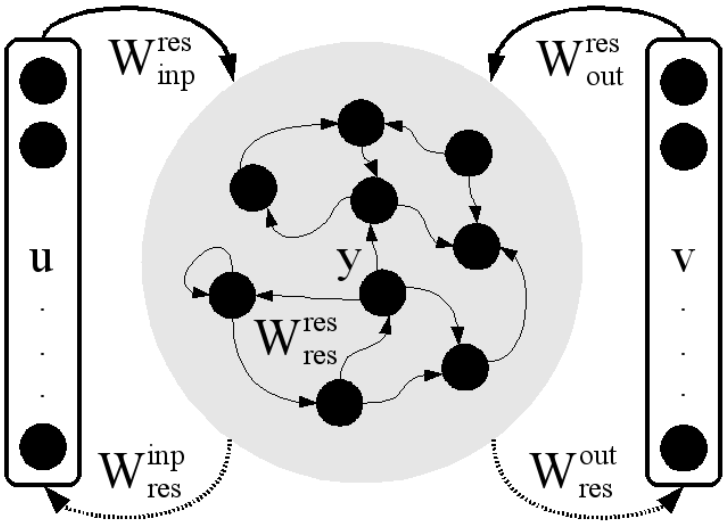


Figure 2.10: Diagram for a reservoir computing network. Figure taken from (Reinhart and Steil, 2009).

Reservoir networks can be efficiently trained, resorting for instance to optimization of reservoir weights based on neural intrinsic plasticity (Steil, 2007), or other methods like evolutionary algorithms (Chatzidimitriou and Mitkas, 2013). Still, one of the problems with reservoir computing is the fact that currently there is no sufficient knowledge on how

the reservoir characteristics affect the learning performance. This stems from the fact that it is harder to identify and set the inductive bias of the reservoir networks, at least compared to other machine learning algorithms. The highly complex dynamic behaviour of a reservoir network, which in certain conditions can approach a chaotic regime, makes these kind of networks to have a black-box behaviour when used for function approximation. The lack of a probabilistic foundation is also a handicap for these architectures, since typically there is no uncertainty treatment whatsoever.

Some of the early applications of neural networks to sensorimotor learning go back to the work of Miller (1989), where feed-forward neural networks are used to control the position and orientation of an object in the field of view of a video camera mounted on the end of a robot arm, in a real-time manner. Demers and Kreutz-Delgado (1992) also use a feed-forward neural network to learn the inverse kinematics of a robotic manipulator. In this work, due to the multi-valued nature of the inverse kinematics relation, an unsupervised learning pre-processing step is first used to partition the input space into regions where the inverse kinematics relation is invertible.

The versatility of the neural networks paradigm has made possible the development of some approaches specifically directed at multi-valued function learning. Among them, there is the work of Shizawa (1996), based on the regularization network of Poggio and Girosi (1990), while Brouwer (2004) and Lee and Lee (2001) use feedforward networks to learn multi-valued relations. These approaches, nevertheless, have very limited applications, as the number of multi-valued solutions must be known beforehand and their validity was only tested in very low dimensional toy datasets. Tomikawa and Nakayama (1998), on the other hand, use a recurrent neural network that, after training, can dynamically be driven to one of the multi-valued solutions of the multimap, depending on the initial value of the output of the network. However, a set of multi-valued solutions is not available, and, like the previous works, it has only been applied to simple toy datasets.

The work of Patino, Carelli, and Kuchen (2002) present some stability proofs and analysis for the inverse dynamics controllers of robotic manipulators that employ neural networks that were previously trained in offline mode. Yet, perhaps most of the works that use a neural network architecture in a robotic context try to emulate biologically realistic neural circuit models, for instance to generate movement of a robot arm (Joshi and Maass, 2005; Butz, Herbort, and Hoffmann, 2007; Herbort, Butz, and Pedersen, 2010; Hemion, Joublin, and Rohlfing, 2012). Such biological plausibility comes at a cost of fairly complex network architectures, that are difficult to train and to generalize to other sensorimotor learning problems. This happens for instance in the work of Hemion, Joublin, and Rohlfing (2012): while their neural fields representation of the overall input-output joint density distribution provides a simple way of getting multi-valued forward and inverse predictions, their approach requires a number of network units that exponentially

grows with the number of input and output dimensions. This, of course, makes its use infeasible for online learning of even sensorimotor models with moderate dimensions.

Recurrent neural networks and reservoirs, due to their rich and complex dynamic behaviour, have also been extensively used for robotic learning and control. Reinhart and Steil (2009) learn an inverse kinematics internal model for the iCub robot (Tsagarakis, Metta, et al., 2007) using reservoirs; in this work, only one inverse solution is obtained, based on the initial condition of the network and its dynamical evolution. Later, the same authors used combined ideas of reservoir computing and extreme learning machines in an associative network (Reinhart and Steil, 2011), represented in Figure 2.11. This network

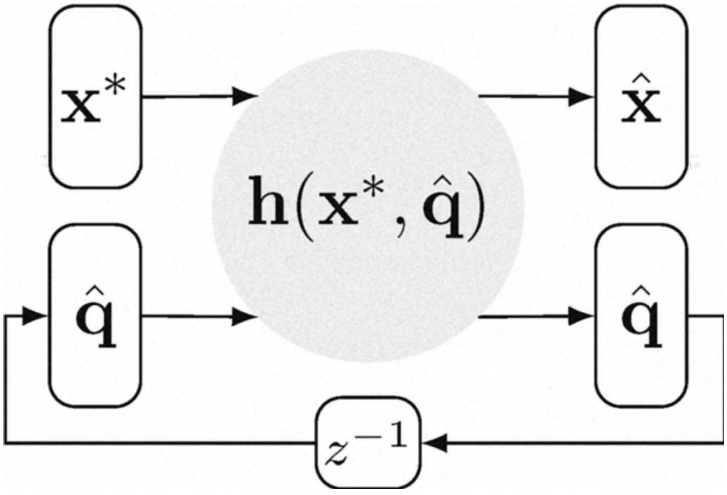


Figure 2.11: Diagram for the associative network of Reinhart and Steil (2011), used for simultaneous inverse and forward kinematics learning. In this example there is a feedback connection on the joints values, and consequently the network is used in inverse kinematics prediction mode. Feedback of the task space vector would result in a forward prediction. Figure taken from (Reinhart and Steil, 2011).

can associate input and output values by iterating the output feedback-driven network dynamics, in a similar way to reservoir networks with output feedback and the earlier work of Tomikawa and Nakayama (1998): this allows forward and inverse kinematics solutions to be recovered from the same network. Multiple basins of attraction for a single input are created by extending the training data by synthesized sequences, to promote attraction to the training data samples. However, it is a bit unclear how the generalization of the network is controlled and what mechanisms exist to avoid overfitting; also, although multi-valued solutions are implicitly stored in the associative network, only one solution can be obtained at a time, again depending on the dynamical evolution of the associative network. Recently an extension of this work was developed to deal with arms and torso coupling in a humanoid upper body (Reinhart and Steil, 2012). Also recently, Hartmann, Boedecker, et al. (2012) combined recurrent neural networks and Gaussian process regression (Section 2.2.4) to learn online the inverse dynamics of a musculoskeletal

robot.

2.2.4 Gaussian Process Regression and Kernel Machines

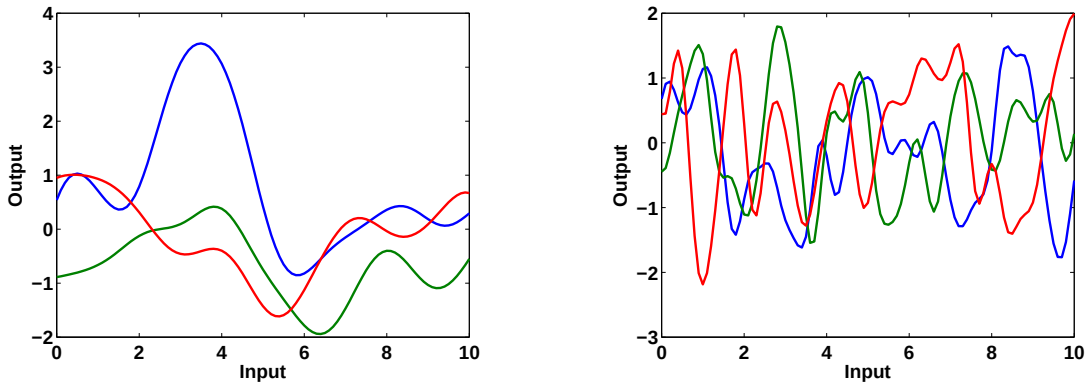
A Gaussian process (GP) can be viewed as a generalization of the Gaussian distribution, where instead of random scalars and vectors there are random functions. More formally, and taking the definition in Rasmussen and Williams (2006), *a Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.* This latter distribution is fully specified by its mean $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}_l, \boldsymbol{x}_m)$,

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] \,, \tag{2.19}$$

$$k(\boldsymbol{x}_l, \boldsymbol{x}_m) = \mathbb{V}(f(\boldsymbol{x}_l), f(\boldsymbol{x}_m)) = \mathbb{E}[(f(\boldsymbol{x}_l) - m(\boldsymbol{x}_l))(f(\boldsymbol{x}_m) - m(\boldsymbol{x}_m))] \,, \tag{2.20}$$

where $f(\boldsymbol{x})$ is the function to be learned. The specification of $m(\boldsymbol{x})$ and $k(\boldsymbol{x}_l, \boldsymbol{x}_m)$ implicitly defines a probabilistic distribution over functions f. Usually $m(\boldsymbol{x})$ is assumed zero, and so the covariance function, defined by the kernel function $k(\boldsymbol{x}_l, \boldsymbol{x}_m)$, is directly responsible for the characteristics of the class of functions f generated by the Gaussian process. To show how the choice of the kernel affects the type of functions generated by the Gaussian process, in Figure 2.12 some random samples taken from Gaussian processes are depicted, corresponding to the squared-exponential covariance function with two different length-scales λ,

$$k(\boldsymbol{x}_l, \boldsymbol{x}_m) = \exp\left(-\frac{|\boldsymbol{x}_l - \boldsymbol{x}_m|^2}{2\lambda^2}\right) \,. \tag{2.21}$$



(a) $\lambda = 1$. (b) $\lambda = 1/3$.

Figure 2.12: Random samples drawn from a Gaussian process using a squared-exponential covariance function, with two different length-scales.

When used in a regression setting, a Gaussian process is used as a prior distribution

for the function f to estimate. Training data $(\boldsymbol{x}_i, y_i)$, for $i = 1 \dots N$, is then assumed to be drawn from this prior distribution, as well as any given query point $\boldsymbol{x}_q$ for which a prediction $\hat{y}$ is desired. The training outputs and the desired prediction are jointly Gaussian, according to the definition of a Gaussian process:

$$\begin{bmatrix} \boldsymbol{y} \\ \hat{y} \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) & \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_q) \\ \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_q)^T & k(\boldsymbol{x}_q, \boldsymbol{x}_q) \end{bmatrix}\right) . \tag{2.22}$$

Here, $\boldsymbol{y}$ is a vector comprising the output training values y_i and $\boldsymbol{X}$ is a $d \times N$ matrix whose columns are the training points $\boldsymbol{x}_i$. The matrix $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$ collects the covariances between training points, *i.e.*, its element (i, j) is the covariance between the ith and the jth training points. The vector $\boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_q)$, on the other side, gathers the covariances between the training data and the query point.

The fundamental aspect of Gaussian process regression (GPR) is that a posterior distribution for a function value $\hat{y}$ can be obtained that takes into account the *observed* values $\boldsymbol{y}$ in the training set. In a Bayesian setting, this is equivalent to obtaining the distribution for the output $\hat{y}$ conditioned on the observed training output values $\boldsymbol{y}$, *i.e.*, $p(\hat{y}|\boldsymbol{x}_q, \boldsymbol{y}, \boldsymbol{X})$. From a practical point of view, this effectively narrows the space of functions generated by the Gaussian process prior, making them agree, in a probabilistic sense, with the observed training data.

In a realistic setting it is assumed that the observations y_i are noisy and that the true function doesn't necessarily coincide with the training output values. The effect of addictive, i.i.d. Gaussian noise in the outputs, with variance σ_n^2, can be easily incorporated in this probabilistic model by changing the variance expression for a data point, making

$$\mathbb{V}(y_i) = \mathbb{V}(f(\boldsymbol{x}_i), f(\boldsymbol{x}_i)) = k(\boldsymbol{x}_i, \boldsymbol{x}_i) + \sigma_n^2 . \tag{2.23}$$

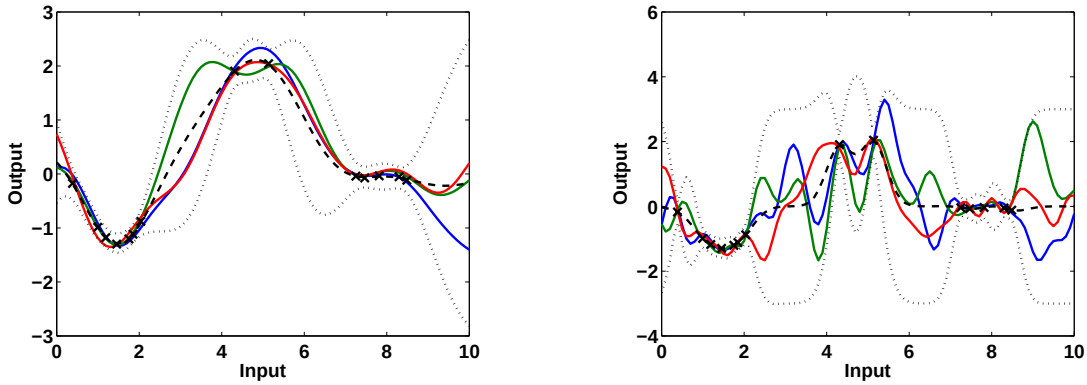
Accounting for this noise in the model in equation (2.22) is equivalent to adding σ_n^2 to $k(\boldsymbol{x}_q, \boldsymbol{x}_q)$ and a diagonal matrix $\sigma_n^2 \mathbf{I}$ to the Kernel matrix $\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$. Predictive distributions for the output corresponding to a query point $\boldsymbol{x}_q$ can then be easily obtained by conditioning the joint Gaussian distribution of equation (2.22) on the training data: this results also in a Gaussian distribution for $\hat{y}$, with mean and variance given by

$$\mathbb{E}[\hat{y}] = \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_q)^T (\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \mathbf{I})^{-1} \boldsymbol{y} \qquad \text{and} \tag{2.24}$$

$$\mathbb{V}[\hat{y}] = \sigma_n^2 + k(\boldsymbol{x}_q, \boldsymbol{x}_q) - \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_q)^T (\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 \mathbf{I})^{-1} \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_q) . \tag{2.25}$$

In Figure 2.13 some sample functions taken from the posterior distributions are shown, corresponding to the priors shown in Figure 2.12, together with the training data that originated this distribution.

The kernel covariance structure and its parameters (commonly known as the *hyper-*



(a) $\lambda = 1$. (b) $\lambda = 1/3$.

Figure 2.13: Random samples drawn from the posterior Gaussian process. The black dashed line is the mean of the posterior distribution, while the dotted lines depict a 3 standard deviations confidence interval.

parameters) define the characteristics of the functions to be learned; they can deeply influence the final prediction results, and can be viewed as the inductive bias of a GP model. Still, most of the cases there does not exist enough information about the function to learn that unequivocally defines the covariance function to use. It is common in this situation to infer the hyperparameters from the training data, a procedure that can be seen as training the Gaussian process model. There are various methods to train a Gaussian process, and by far the most used ones are *marginal likelihood maximization* and *cross-validation* (Rasmussen and Williams, 2006).

Gaussian process regression can also be understood as a Bayesian linear regression performed in an infinite feature space, according to the model

$$y = \sum_{j=1}^{\infty} \beta_j \Phi_j(\boldsymbol{x}) \; . \tag{2.26}$$

This model can be interpreted as the infinite counterpart of the basis expansion introduced in Section 2.2.3 (Equation 2.16). Usually the features, represented by basis functions $\Phi_j(\boldsymbol{x})$, need not to be explicitly known, since the predictive distributions depend only on dot products of these infinite basis functions vectors, which in turn are conveniently represented by the kernel functions $k(\boldsymbol{x}_l, \boldsymbol{x}_m)$. This is commonly known as the *kernel trick*, which allows to perform exact calculations on infinite feature spaces resorting only to a finite number of sample points. This apparently non intuitive result is explained in the context of *reproducing kernel Hilbert spaces* (RKHS), which is the foundation for other nonparametric learning methods like *smoothing splines* and *support vector machines* (Wahba, 1990; Girosi, Jones, and Poggio, 1995; Evgeniou, Pontil, and Poggio, 2000).

Smoothing splines (Green and Silverman, 1994; Wahba, 1990) can be interpreted as a

nonparametric extension of B-splines, where a knot is placed in every training point and the complexity of the resulting fit is controlled by a penalty term in the residual sum of squares to be minimized. Their extension to multivariate inputs is known as *thin-plate splines*; however, unlike the univariate case, the computational complexity for thin-plate splines makes the training procedure infeasible for a moderate number of training points, unless some regularization techniques are employed (Hastie, Tibshirani, and Friedman, 2009).

Support vector regression (SVR) (Evgeniou, Pontil, and Poggio, 2000; Smola and Schölkopf, 2004; Schölkopf and Smola, 2002) is another nonparametric function approximation method whose popularity rivals with Gaussian process regression. Like GPR, it makes use of the kernel trick to lift the regression problem to an implicit high dimensional feature space. Different from it, however, it lacks a probabilistic nature, resulting instead from the minimization of a penalized error function, where the ϵ-insensitive error function, defined by

$$L_\epsilon(\boldsymbol{e}) = \begin{cases} 0 & \text{if } |\boldsymbol{e}| \leq \epsilon \\ |\boldsymbol{e}| - \epsilon & \text{otherwise} \end{cases},$$

is used instead of the usual squared error. Such error function does not penalize small approximation errors, and the use of the L_1 norm, like LASSO, does not penalize outliers so strongly as the sum of squared residuals. As a consequence, support vector machines (SVM) naturally induces sparse solutions, where only a subset of the training data — the *support vectors* — effectively contribute to the prediction.

Gaussian process and support vector regression are currently considered state-of-the-art function approximation methods, and there is no definite evidence in favour of one of these machine learning algorithms with respect to approximation performance. While SVR is more robust to the presence of outliers in the training set, its non probabilistic nature may become a severe handicap concerning robotic sensorimotor learning, where many times the uncertainty of the prediction is also an important quantity of interest. Also, in general GPR seems to be more flexible: the kernel hyperparameters can be learned via evidence maximization, and a GP prior can be easily incorporated into a hierarchical Bayesian model.

The number of training points is an important issue in nonparametric kernel methods, since computational considerations severely limit the allowable number of such points. To overcome such limitations, sparse versions of GPR were devised to speed up learning, that choose and only take into account a subset of the training data (Smola and Bartlett, 2001; Csató and Opper, 2002; Seeger, Williams, and Lawrence, 2003; Snelson and Ghahramani, 2006; Keerthi and Chu, 2006; Candela and Rasmussen, 2005; Lázaro-Gredilla, 2010).

Real-time sensorimotor learning requires an online training scheme: this is not addressed by GPR or SVR, at least in their original form, due to their batch nature and

the consequent need to maintain all training points in memory. There has been a large number of adaptations to these algorithms that provide incremental versions of them, *e.g.* (Csató and Opper, 2002; Engel, Mannor, and Meir, 2002; Ma, Theiler, and Perkins, 2003; Nguyen-Tuong and Peters, 2011a). However, their ability to process large volumes of sensorimotor data, with high sampling rates, has not yet been properly addressed and remains to be proven, although recently Nguyen-Tuong and Peters (2011a) presented an efficient algorithm for management of a reduced dictionary of training data, to be consequently used in kernel regression, that is shown to be adequate for real-time learning.

Another approaches to reduce the computational burden of these kernel methods take instead a local perspective, learning a set of GPR or SVR models that are only valid in a local region of the input space. They can be interpreted under the *mixture model* paradigm, that also covers many other computational models: this topic is covered in detail in Section 2.2.6.

There is a vast history of successful applications of kernel learning to robotics: Pelossof, Miller, et al. (2004) use SVR to find optimal object grasps by a robotic hand; GPR is applied to adaptive control by Kocijan, Murray-Smith, et al. (2004), while reinforcement learning and optimal control also use GPR in Rottmann and Burgard (2009) and Deisenroth, Rasmussen, and Peters (2009); GPR is used in conjunction with Bayesian filtering to estimating the state of an autonomous micro-blimp (Ko and Fox, 2009); and Nguyen-Tuong and Peters (2010) show how to integrate *a priori* robotic inverse dynamics model knowledge within a GPR framework, to mention only a few examples.

Despite their current state-of-the-art approximation performance, there are nevertheless some issues in the adaptation of kernel methods to sensorimotor learning: besides the aforementioned computational cost, there is no simple way of obtaining inverse predictions from a trained forward model. Also, their single-valued nature prevents their use in learning models whose outputs may be potentially multi-modal. Nguyen-Tuong and Peters (2012) show how to use kernel regression to control a redundant robot in its task space: the multi-valued nature of the problem is solved using a *single* local model that only considers training data in the vicinity of the query point; this, however, has the undesirable property of constantly discarding the acquired information that describes the input-output relation in other regions of the sensorimotor space.

2.2.5 Unsupervised Learning Approaches

Supervised learning based on training examples $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ can be converted to an unsupervised *density estimation* problem, by means of the conditional densities obtained from the learned joint density function over both inputs and outputs, $p(\boldsymbol{z})$, where $\boldsymbol{z} = [\boldsymbol{x}^T \boldsymbol{y}^T]^T$. Forward and inverse relations can be obtained from a learned joint density $p(\boldsymbol{z}|\boldsymbol{Z})$, where $\boldsymbol{Z}$ comprises all training points $\boldsymbol{z}_i$, by taking the conditional densities $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{Z})$ or

$p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{Z})$ (Ghahramani, 1994).

Memory based methods, for instance, implicitly define a joint density distribution by storing all the training points, from which forward or inverse predictions can be readily obtained (Atkeson, Moore, and Schaal, 1997b). These methods, however, present a serious drawback: since the prediction is obtained from a weighted average of the training samples, multi-valued solutions, coming from a multi-modal conditional distribution, are simply merged together in a single estimate.

Sensorimotor models can be represented by a low dimensional manifold in the joint input-output space, usually with the same dimension as the input space. Several dimensionality reduction techniques exist that search for reduced dimensionality, latent representations of the unsupervised training data points $\boldsymbol{z}_i$: among them, perhaps the simplest one is the Principal Component Analysis (PCA) (Pearson, 1901). This procedure seeks another coordinate system to represent the training data, where the input variables become decorrelated, and is based on the SVD decomposition of the (centred) matrix $\boldsymbol{Z}$. It can be shown that the new orthogonal variables, called principal components, are organized in such a way that the projection of the training data onto the first principal component has the highest variance among all linear combinations of the original variables, that the second principal component has the highest variance among all linear combinations of variables orthogonal to the first principal component, and so on (Murphy, 2012).

In the new coordinate system, selecting only the first q variables (corresponding to the first q principal components) and discarding the remaining variables is equivalent to finding an affine hyperplane of rank q that best describes the data, *i.e.*, for which the sum of squared errors between the training data and its projection onto the hyperplane is minimized. This operation can be seen as discarding the directions of the joint input-output space that have the least variability of the training data.

The PCA for a given training data is easy to obtain and can be calculated either incrementally (Warmuth and Kuzmin, 2008) or in an iterative fashion (Roweis, 1998); forward and inverse conditional predictions can also be easily obtained from the PCA model: these are two highly desirable properties for sensorimotor learning. However, as can be immediately noted, the linear approximation to the training data is usually a overly optimistic assumption for most of the models to be learned: in these cases nonlinear dimensionality reduction algorithms are probably of much better use. There is a vast literature on this subject, and among the most well known methods there can be found *self-organizing maps* (SOM) (Kohonen, 2001), *principal curves and surfaces* (Hastie and Stuetzle, 1989), *multidimensional scaling* (Torgerson, 1958), *locally-linear embedding* (Roweis and Saul, 2000) and the *Isomap* algorithm (Tenenbaum, Silva, and Langford, 2000), among many others. However, most of these algorithms are very computational demanding, and many of them are batch algorithms that require the presence of all

training points. Some incremental variants of these algorithms exist (Law and Jain, 2006; Jia, Yin, et al., 2009; Li, Jiang, et al., 2011), but the lack of a probabilistic model for many of these nonlinear dimensionality reduction techniques prevents an easy calculation of conditional densities, needed for prediction in forward or inverse models: this somewhat discourages their use for model learning in a demanding robotics learning environment.

Any density estimation scheme that operates in the joint input-output space can in principle be used to provide forward and inverse conditional distributions: one recent example is for instance the work of Bocsi, Nguyen-Tuong, et al. (2011), that uses joint kernel support estimation (Lampert and Blaschko, 2009) to model the joint probability distribution, and uses this model to learn the inverse kinematics relation of a redundant robot. Training of this model resorts to One-Class SVM (Schölkopf, Platt, et al., 2001): this, however, is a computationally expensive procedure that must be performed offline.

A different approach for density estimation takes a local perspective, representing a general probability density in the input-output space as a mixture of local and simpler distributions. The ubiquitous *mixture of Gaussians* has the form

$$p(\boldsymbol{z}) = \sum_{k=1}^{M} \pi_k p_k(\boldsymbol{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \ , \tag{2.27}$$

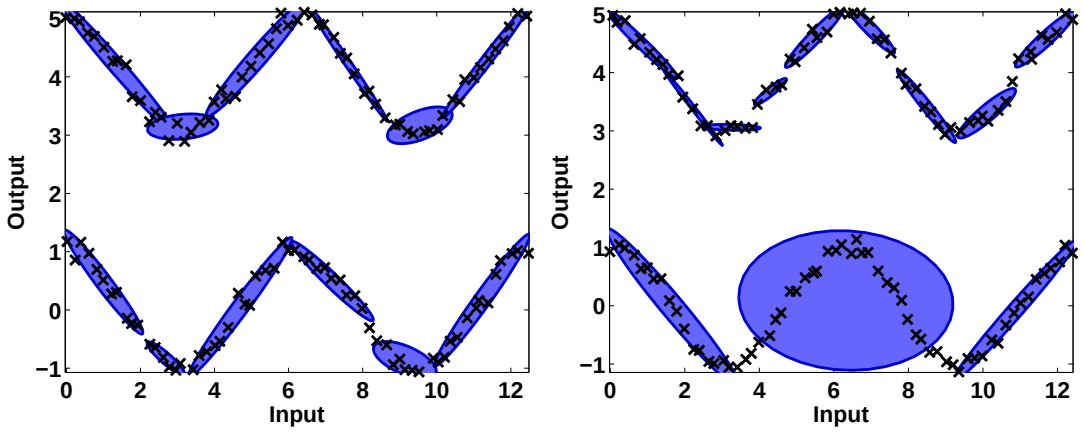
where each distribution $p_k()$ is normally distributed with mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$, and where π_k is the *a priori* probability that a sample $\boldsymbol{y}_i$ is generated from the k*th* Gaussian distribution (McLachlan and Peel, 2000). Of interest are also the *mixture of factor analysers* (Ghahramani and Hinton, 1996) and the *mixture of PCA's* (Tipping and Bishop, 1999); these methods locally perform dimensionality reduction, introducing some structure in the covariance matrices of the underlying Gaussian densities, and can be pictured as mixtures of "flattened" Gaussians.

Forward and inverse conditional distributions can be directly obtained from the trained mixture models, due to their probabilistic nature. During training, the main difficulty in estimating the mixture parameters is the presence of a latent, unobservable variable w_{ij} that signals which mixture component j was responsible for generating training sample $\boldsymbol{z}_i$ — if these latter variables were known the estimation of the mixture parameters would be trivial. Formulating the problem as a generative model with unobserved latent variables naturally enters the domain of the *expectation-maximization* (EM) algorithm (Dempster, Laird, and Rubin, 1977), which in fact is one of the most popular approaches for training these mixtures. The EM algorithm naturally can be adapted for online learning schemes (Neal and Hinton, 1999; Capp and Moulines, 2009) — this is a highly desirable property for robotic applications.

Another problem that arises in these kind of mixture models is the choice of the number of components that constitute the mixture. Several extensions to the mixture model exist that are capable of automatically selecting an appropriate number of components:

the infinite mixture concept presented by Rasmussen (2000), for instance, is a Bayesian method that assigns a Dirichlet process prior on the mixing proportions of the mixture, responsible for the automatic generation of the correct number of components. This, however, usually requires an offline, computational expensive training based on Markov chain Monte Carlo sampling methods, not suitable for real-time learning, although some variational techniques can be used to accelerate the training process. Other extensions rely on the EM algorithm, either using a greedy approach to grow the mixture to an appropriate number of components (Vlassis and Likas, 2002; Verbeek, Vlassis, and Kröse, 2003) or, using the opposite idea, starting with a large number of components and automatically shrinking the number of components of the mixture to a reasonable value. This can be achieved, among other techniques, by considering a Bayesian approach and carefully chosen priors (Figueiredo and Jain, 2002) or through reducing the uncertainty of missing data, using the mutual information between the missing and incomplete data (Li, Zhang, and Jiang, 2005).

The works of Calinon, Guenter, and Billard (2007) and Lopes and Damas (2007) are two examples of how the mixture of Gaussians framework can be applied to sensorimotor learning. However, note that unsupervised learning using mixture of Gaussians is a more difficult problem than its supervised counterpart, usually conducting to worse results, as it ignores that joint data, apart from noise corruption, lies in a lower dimensional manifold; this phenomenon can be observed in Figure 2.14. Other approaches, like the mixture



(a) (b)

Figure 2.14: Gaussian mixture model for supervised learning: (a) a mixture of 12 Gaussian models in the input-output space is used to successfully approximate training data coming from a multi-valued relation; (b) a different run of the EM algorithm for the mixture, applied to the same training data, results in a different, sub-optimal solution, where the predicted conditional output variance is much higher than the true data output noise in some regions of the input-space.

of factor analysers, alleviate this matter by assuming data to be generated from a lower dimensional space, but training this model requires the estimation of the projection of

training data into this space. This additional latent data, known as the factors, represent the coordinates of a data point in the lower dimensional space, and its estimation considerably complicates the training phase, as the EM algorithm that is traditionally used to train this mixture becomes prone to a larger number of local, sub-optimal likelihood maxima. This is perhaps the main drawback of these unsupervised approaches, when applied to multi-valued supervised learning and prediction: even if they work reasonably well in low dimensional problems, as the input dimension grows their performance will likely start to drop, resulting in convergence to sub-optimal solutions that do not take the problem structure into account.

2.2.6 Mixtures of Experts

The divide and conquer principle can be seen as the foundation of a *mixture model*, where a potentially hard to approximate input-output relation is divided into a set of simpler learning problems. Mixture models for unsupervised learning were already addressed in Section 2.2.5; in a supervised learning context, each model of the mixture approximates the conditional output distribution in a particular region of the input space, according to

$$p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\Theta}) = \sum_{j=1}^{M} p(\boldsymbol{y}|\boldsymbol{x}, w_j, \boldsymbol{\Theta}) p(w_j|\boldsymbol{x}, \boldsymbol{\Theta}) \ , \tag{2.28}$$

where w_j is a latent variable that indicates that the jth mixture component is responsible for the generation of the observed output $\boldsymbol{y}$ and $\boldsymbol{\Theta}$ is the set of parameters that define the mixture, to be learned from the data. This probabilistic model is known as the *mixture of experts* model (MOE), where $p(\boldsymbol{y}|\boldsymbol{x}, w_j, \boldsymbol{\Theta})$ represents the conditional output distribution provided by expert j and $p(w_j|\boldsymbol{x}, \boldsymbol{\Theta})$ defines, for every input vector $\boldsymbol{x}$, the probability that expert j is responsible for generating an output $\boldsymbol{y}$. This is known as the *gate function*, that partitions the input space in a set of distinct regions, implicitly defining the relative strength of the experts in a particular input location.

Mixtures of experts were originally introduced by (Jacobs, Jordan, et al., 1991): like CART (Breiman, Friedman, et al., 1984) and MARS (Friedman, 1991), this statistical model can be regarded as a method that performs a partitioning of the input space, where soft splits, defined by the gate $p(w_j|\boldsymbol{x}, \boldsymbol{\Theta})$, are used instead of the hard decision boundaries typical of CART and MARS. The mixture of experts model was also extended into a hierarchical representation, where the final predictions of distinct mixture of experts models are fed into a top level gate, much in the style of a multilayer neural network (Jordan and Jacobs, 1994).

In the original MOE formulation the gate function was defined by a softmax function, according to

$$p(w_j|\boldsymbol{x}, \boldsymbol{\Theta}) = \frac{\exp(\beta_j(x, \boldsymbol{\Theta})}{\sum_{k=1}^{M} \exp(\beta_k(x, \boldsymbol{\Theta})} \ , \tag{2.29}$$

where the activation function $\beta_j(x, \boldsymbol{\Theta})$ was linear in $\boldsymbol{x}$ and thus split the input space along an hyperplane of arbitrary orientation. An alternative model was later devised by Xu, Jordan, and Hinton (1995), where the gate responsibilities $p(w_j|\boldsymbol{x}, \boldsymbol{\Theta})$ were defined according to Bayes' theorem,

$$p(w_j|\boldsymbol{x}, \boldsymbol{\Theta}) = \frac{p(\boldsymbol{x}|w_j, \boldsymbol{\Theta})p(w_j|\boldsymbol{\Theta})}{\sum_{k=1}^{M} p(\boldsymbol{x}|m_k, \boldsymbol{\Theta})p(m_k|\boldsymbol{\Theta})} \,. \tag{2.30}$$

Using this gate model with density distributions $p(\boldsymbol{x}|w_j, \boldsymbol{\Theta})$ taken from the exponential family, as proposed in the work of Xu, Jordan, and Hinton (1995), has two major advantages: in one hand it makes possible the training of the mixture using the Expectation-Maximization procedure; on the other hand it restricts the influence of the experts to confined regions of the input space — for this reason, this alternative model is also known as the *localized* MOE.

One of the greatest strengths of the mixture of experts model is its inherent versatility, since a large variety of learning architectures can be chosen for the experts:

Linear regression experts: In this architecture each expert models the input-output relation by a linear function (Section 2.2.1). This is one of the earliest models found in the literature (Jordan and Jacobs, 1994; Xu, Jordan, and Hinton, 1995; Waterhouse, Mackay, and Robinson, 1996), and its simple formulation typically allows a fast training procedure, like the online version presented by Sato and Ishii (2000). While the mixture of linear experts can be initially though as a piecewise linear approximation of the input-output relation, the fact that soft splits of the input space are used can effectively result in a highly nonlinear conditional prediction. The work of Bishop and Svensén (2002) and Ueda and Ghahramani (2002) extends this model by taking a Bayesian approach, while Bo, Sminchisescu, et al. (2008) present an efficient training procedure for a Bayesian mixture of linear experts.

Locally weighted projection regression (LWPR) (Vijayakumar, D'Souza, and Schaal, 2005) and XCSF (Wilson, 2002) are two popular non probabilistic algorithms that share many characteristics with the mixture of linear experts concept: while sharing the divide-and-conquer approach of the MOE architectures, using linear models to represent the input-output relation in localized regions of the input space, they use training algorithms different from the standard probabilistic MOE approaches. LWPR has been widely used for online, real-time learning of robotic tasks, and has its origins in the local linear regression model (Atkeson, Moore, and Schaal, 1997a; Schaal, Atkeson, and Vijayakumar, 2002), where the need to handle large volumes of sensorimotor data has led to a careful management of available resources, by means of a representation of training points by locally linear models sufficient statistics. This algorithm uses a gradient descent on the prediction error, based on

a stochastic leave-one-out cross-validation algorithm, to adapt the distance metrics of the receptive fields that partition the input space; within each receptive field, a linear relation from input to output is obtained via an incremental partial least squares algorithm that efficiently deals with redundant and high dimensional input spaces. XCSF, on the other hand, updates the input distance metric of each model resorting to a steady state genetic algorithm, while each linear model is fitted using a recursive least squares algorithm.

Neural networks experts: Perhaps the most significant use of a neural network to model the conditional prediction of each expert is given by the mixture density network (MDN) (Bishop, 1994; Bishop, 2006). Contrary to the majority of MOE approaches that seek to partition the input space into smaller regions, this MDN was developed with the specific purpose of providing generic conditional output probability distributions, with a special focus on multi-valued models. Each expert of the mixture, modelled as a neural network, is supposed to approximate a particular branch of the multi-valued function to be learned, thus partitioning the output space for each input point.

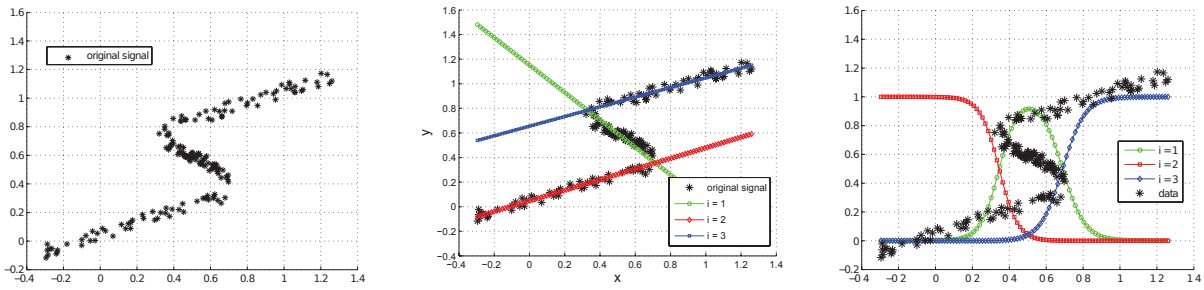
Smoothing kernel experts: The recent work of Huang, Li, and Wang (2013) uses a mixture of smoothing kernel experts to specifically model multi-valued relations, much in the same way as the mixture density network described above. An EM algorithm is derived in this paper, together with some corresponding asymptotic properties. Selection of the kernel bandwidth, however, is performed offline, as well as the main EM iterations. Furthermore, application of the proposed algorithm is limited to the univariate input, univariate output situation.

Gaussian process experts: The need to overcome the inversion of large covariance matrices, corresponding to the full training dataset of the original Gaussian process model, and the handling of nonstationary covariance and noise are perhaps the main motivations for the use of Gaussian processes in the MOE architecture. Assigning different GP experts to smaller regions of the input space effectively reduces the effective number of training points of each expert and makes the required matrix inversions computationally feasible. This also permits the use of independent hyperparameters for each GP, thus allowing different input length-scales and output noises in different regions of the input space. The need for different kernel functions in distinct regions of the input space is also the main motivation for the use of SVM experts in the MOE model (Cao, 2003; Lima, Coelho, and Von Zuben, 2007).

Even if the use of mixtures of GP experts overcomes the problems described above, heavy computational resources are still needed to train such mixture, which is typically performed offline, as in the works of Tresp (2001), Rasmussen and Ghahramani (2002) and Meeds and Osindero (2006). Recently there has been an effort to

develop more efficient training techniques for the mixture of GP experts (Nguyen-Tuong, Seeger, and Peters, 2009a; Yuan and Neubauer, 2009; Yang and Ma, 2011); nonetheless, computational complexity is usually a delicate issue when mixtures of Gaussian processes are considered.

As described above, mixtures of experts can be used to partition both the input and the output space. Input space partitioning can lead to simpler and more computational efficient learning solutions; on the other hand, allowing different experts to share the same input space regions, while allocated to different output space regions, naturally enables learning of multi-valued functions as those typically arising in inverse models. This situation is depicted in Figure 2.15, where three experts are used to model a multi-valued relation. Among many other supervised learning algorithms, this easy handling of multi-



(a) (b) (c)

Figure 2.15: Modelling of a multi-valued relation by a mixture of three experts: (a) training data, coming from a multi-valued relation (note the region around $x = 0.5$); (b) the linear input-output relations for each of the experts; (c) Gate probabilities assigned to each of the experts, as a function of x. Figures taken from (Yuksel, Wilson, and Gader, 2012).

valued functions is perhaps a distinctive property of the mixture of experts architecture. This characteristic has been explicitly addressed in several works (Bishop, 1994; Bishop and Svensén, 2002; Kanaujia and Metaxas, 2006; Sminchisescu, Kanaujia, and Metaxas, 2007; Qin and Carreira-Perpinan, 2008); other works have reported this property of MOE, using datasets where output multi-modality occurs (Rasmussen and Ghahramani, 2002; Meeds and Osindero, 2006; Yuan and Neubauer, 2009).

Bayesian schemes are known to provide better regularization against overfitting, as opposed to traditional maximum likelihood inference (Bishop, 2006), and so it comes as no surprise that several Bayesian variations of the MOE architecture, for different types of gates and experts, have been proposed in the last years, using either maximum *a posteriori* procedures based on the EM algorithm (Kanaujia and Metaxas, 2006; Sminchisescu, Kanaujia, and Metaxas, 2007; Bo, Sminchisescu, et al., 2008), variational methods (Waterhouse, Mackay, and Robinson, 1996; Bishop and Svensén, 2002; Ueda and Ghahramani, 2002) or Markov chain Monte Carlo (MCMC) techniques like Gibbs sampling (Rasmussen and Ghahramani, 2002; Meeds and Osindero, 2006). Many of these

methods were also used to automatically find an appropriate number of experts for the mixture or the model structure in a hierarchical approach, through maximization of the marginal likelihood (Ueda and Ghahramani, 2002; Bishop and Svensén, 2002), although requiring significant computations. The use of a Dirichlet process model for the gate in the works of Rasmussen and Ghahramani (2002) and Meeds and Osindero (2006) introduces the *infinite mixture* concept, where the number of experts is not limited *a priori* and is inferred from the training data. Training, however, resorts to a computationally expensive MCMC algorithm, which can hardly be used in an online setting. Other approaches adopt carefully chosen priors to induce sparsity in the mixture (Kanaujia and Metaxas, 2006), much in the style of the work of Figueiredo and Jain (2002). In general, all these model selection techniques are similar to the ones described in Section 2.2.5, and thus non Bayesian strategies based on growing the mixture of experts (Saito and Nakano, 1996; Fritsch, Finke, and Waibel, 1997), pruning it (Jacobs, Peng, and Tanner, 1997) or simultaneously growing and pruning the mixture (Waterhouse and Robinson, 1995) have also been devised.

Mixtures of experts have been successfully applied to sensorimotor learning problems: Lima, Coelho, and Von Zuben (2007), for instance, use a mixture of SVM experts to learn nonlinear system dynamics; a mixture of Gaussian processes is used to learn online the inverse dynamics model of a robotic arm (Nguyen-Tuong, Seeger, and Peters, 2009b); a MDN is trained in the work of Qin and Carreira-Perpinan (2008) to learn the inverse kinematics of a PUMA arm, and the multiple solutions provided by the network are used to plan feasible trajectories in the joint space. One to many relations are also learned by Grollman and Jenkins (2010), where an infinite mixture of sparse Gaussian processes is used to approximate a policy in a learning from demonstration context, and Bo, Sminchisescu, et al. (2008) use the multi-valued learning capability of the mixture of experts architecture to retrieve multiple 3D reconstruction hypothesis from 2D images.

2.3 Discussion

There is currently no out-of-the-box learning solution that can tackle all the peculiarities of robotics sensorimotor learning, and every learning algorithm presented here shows some weaknesses with respect to some of the issues that arise in a demanding robotic context. Nonparametric, Bayesian methods in principle provide an elegant way to learn arbitrary functions in high dimensional spaces, showing an extremely good performance and having the ability to automatically choose the model complexity in a principled way. However, most of the times this comes at a painful computational cost, completely putting aside the possibility of learning online while the robot tries to execute some desired tasks.

Online learning, on the other hand, requires incremental algorithms, since, with small sample times, the large volume of training data that is collected by a learning algorithm

cannot be realistically kept in memory and fully used for inference and prediction. This loss of information, however, prevents an online algorithm from getting the full picture of the training data, and as a consequence makes it harder to produce more informed decisions regarding the choice of the model complexity.

Local learning schemes present some advantages: they are less prone to the curse of dimensionality, as they construct their models in the input-output regions covered by the training data, and are not affected by the destructive interference phenomenon that may occur when a global approximation method is trained intensively in a particular region of the input-output space. Locality, however, means that a partitioning of the space is required: this again makes the choice of model complexity a harder problem.

Most classes of learning algorithms assume a single-valued model: when confronted with training data coming from a multi-valued model they usually generate a single prediction that is close to the average of the possible true solutions. This will typically produce inaccurate predictions, as the space of solutions for a multi-valued problem is usually not convex: the average of solutions is usually not part of the solution space. Having the ability to learn multi-valued models definitely enhance the representation capability for the learned model, but on the other hand it makes the learning problem significantly more difficult, introducing some conceptual issues that do not arise in single-valued regression.

Finally, most of the discussed learning algorithms do not have the mechanisms to provide inverse predictions, *i.e.*, given an output query $\boldsymbol{y}_q$ they cannot generate an estimate or probability distribution for the corresponding input vector $\boldsymbol{x}$. This is in part a consequence of the multi-modality that generally occurs in inverse mappings. Learning methods that can provide simultaneous forward and inverse predictions from their internal models have significant advantages over competing algorithms, as they can estimate both the forward and inverse model of the sensorimotor map being learned. Bearing all these considerations in mind, the next chapter will introduce a learning algorithm, specially suited for robotic applications, that tries to address all the issues raised during this exposition.

Chapter 3

The Infinite Mixture of Linear Experts

This chapter describes the learning algorithm proposed in this dissertation: the previous chapter outlined some specific challenges arising when learning generic sensorimotor maps, and the infinite mixture of linear experts model (IMLE) here presented is an attempt to provide a probabilistic model that can deal with the aforementioned issues.

In particular, the main topic of this chapter is generic nonlinear regression from $\mathbb{R}^d$ to $\mathbb{R}^D$, where d and D are respectively the input and output dimensions of the sensorimotor map to be learned. Online training of the internal model representing this map is a fundamental requisite, as a real-time adaptation mechanism is sought that can be run during the interaction of a robot with its surroundings. Also, the learning model should be flexible enough to accommodate very generic input-output relations, and should be able to provide forward and inverse predictions from the same learned internal model that can be readily exploited for sensorimotor control: as discussed before, this means that the prediction mechanism must be able to deal with multi-valued input-output relations.

The mixture of experts architecture described in the previous chapter has some appealing properties that justify its use for the learning model that will be introduced in this chapter: local experts, responsible only for describing the input-output relation in a particular region of this space, are less prone to dimensionality issues, and the fact that different experts can describe different relations in the same region of the input space make them suited for multi-valued function learning. Additionally, assigning linear models to describe the input-output relation of each expert may potentially lead to a fast training procedure, with the additional benefit that this relation can be easily inverted, for each expert, in order to build an inverse prediction scheme.

This chapter first introduces, in section 3.1, the infinite mixture of linear experts probabilistic model. It then follows, in section 3.2, with the description of a generalized expectation-maximization algorithm (Dempster, Laird, and Rubin, 1977) to train this

mixture: in particular, an online, incremental training scheme is suggested, and special attention is given to the process of enlarging the number of active components of the mixture.

Given the IMLE model learned so far, providing forward and inverse predictions based on the state of this model is a sensitive matter, specially when data comes from a multi-valued relation: obtaining predictions using the current state of the mixture model, as well as corresponding uncertainty estimates, is covered in section 3.3; this same section also covers the procedure required to produce inverse predictions from the same learned model. Also, it is described in this section how to obtain the Jacobian of the relation, *i.e.*, the matrix of all first-order partial derivatives of the IMLE forward map: this Jacobian is essential for closed-loop control using the IMLE model, as will be considered in chapter 5.

Another quantity of interest that can be obtained from this model is the input space direction along which a new training point should be sampled in order to reduce the overall uncertainty the most. This calculations fall under a research topic known as *active learning*, where, in a broad sense, it is desired to obtain an efficient sampling scheme that results in a good quality estimated input-output map, using the least number of training examples. This is examined in section 3.4.

3.1 Probabilistic Model

The infinite mixture of linear experts assumes the following generative model for a sample point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the input vector and $\boldsymbol{y}_i \in \mathbb{R}^D$ is the corresponding output response:

$$\boldsymbol{y}_i | \boldsymbol{x}_i, w_{ij}; \boldsymbol{\Theta} \sim \mathcal{N}\left(\boldsymbol{\mu}_j + \boldsymbol{\Lambda}_j(\boldsymbol{x}_i - \boldsymbol{\nu}_j), \boldsymbol{\Psi}_j\right) \ , \tag{3.1a}$$

$$\boldsymbol{x}_i | w_{ij}; \boldsymbol{\Theta} \sim \mathcal{N}\left(\boldsymbol{\nu}_j, \boldsymbol{\Sigma}_j\right) \ , \tag{3.1b}$$

$$p(w_{ij}; \boldsymbol{\Theta}) = \frac{m_j}{M} \ , \quad \text{with} \quad M = \sum_{k=1}^{\infty} m_k \ . \tag{3.1c}$$

Here w_{ij} denotes a latent or hidden indicator variable that equals 1 if data point i was generated by linear model j and 0 otherwise, with $1 \leq j \leq \infty$ and $\sum_j w_{ij} = 1$; these variables can be gathered, for each i, in an infinite binary vector $\boldsymbol{w}_i$ whose j^{th} component is equal to w_{ij}. For notational convenience, sometimes the shorthand notation w_{ij} will be used to denote the event $w_{ij} = 1$, as in above equations. Also, since $p(w_{ij}; \boldsymbol{\Theta})$ does not depend on a particular point i, sometimes the variable w_j will be used to denote the event that linear model j generates an unspecified sample point. The parameter m_j indicates if expert j is activated, effectively contributing to the mixture: it is equal to 1 if expert j is active and 0 otherwise. Each of the latent indicator variables w_{ij} has consequently an associated probability of $1/M$ if linear model j contributes to the mixture, where M is the

total number of active experts. Perhaps a more common approach to this kind of mixture models is to assign different *a priori* probabilities for the mixture components, by making $p(w_{ij}; \boldsymbol{\Theta}) = \pi_j$, and then learn the values for the parameters π_j from the training data; however, this uniform probability distribution on the latent variables is a more natural approach to the online regression problem, since fully learned mixture coefficients depend heavily on the input training data distribution: this can vary greatly in an online data acquisition setting. Note also that, although using a probabilistic mixture representation, the ultimate goal of this model is to describe a mapping from inputs to outputs: in this context, assigning the same importance to different parts of this mapping seems to make more sense.

Given w_{ij}, input $\boldsymbol{x}_i$ follows a Normal distribution with parameters $\boldsymbol{\nu}_j$ and $\boldsymbol{\Sigma}_j$, while output $\boldsymbol{y}_i$ follows a linear relation from $\boldsymbol{x}_i$, with mean $\boldsymbol{\mu}_j$, linear map matrix $\boldsymbol{\Lambda}_j$ and diagonal covariance matrix $\boldsymbol{\Psi}_j$, corresponding to uncorrelated Gaussian noise in $\boldsymbol{y}$, as depicted in Figure 3.1, for a single expert. This model, apart the uniform distribution for w_{ij}, is similar to the one presented by Xu, Jordan, and Hinton (1995) and Sato and Ishii (2000), where each expert j models a linear relation from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ in some region of the input domain, defined by input centre $\boldsymbol{\nu}_j$ and covariance $\boldsymbol{\Sigma}_j$, this way softly partitioning the input space among the experts.

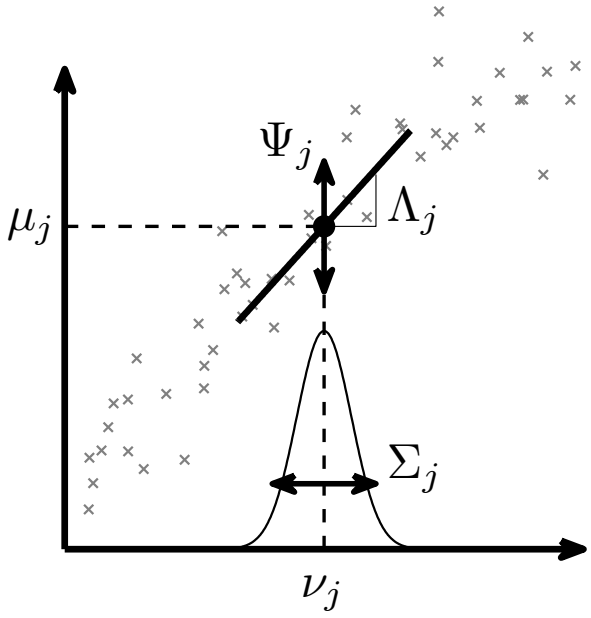


Figure 3.1: A schematic representation, for univariate input and output data, of the parameters of an individual expert in the IMLE model.

Differently from these previous works, however, the following priors for the parameters

of the active experts in the mixture, *i.e.*, for which $m_j = 1$, are defined:

$$\boldsymbol{\nu}_j|\boldsymbol{\Sigma}_j \sim \mathcal{N}(\boldsymbol{\nu}_{0j},\ \frac{1}{n_\nu}\boldsymbol{\Sigma}_j)\ , \tag{3.2a}$$

$$\boldsymbol{\Sigma}_j|\bar{\boldsymbol{\Sigma}} \sim W^{-1}(n_\Sigma\bar{\boldsymbol{\Sigma}},\ n_\Sigma)\ , \tag{3.2b}$$

$$\boldsymbol{\Lambda}_j(k)|\boldsymbol{\Psi}_j(k) \sim \mathcal{N}\left(\boldsymbol{\Lambda}_0 = \mathbf{0},\ \frac{\boldsymbol{\Psi}_j(k)}{n_\Lambda}\mathbf{I}\right)\ , \tag{3.2c}$$

$$\boldsymbol{\mu}_j|\boldsymbol{\Psi}_j \sim \mathcal{N}\left(\boldsymbol{\mu}_{0j},\ \frac{1}{n_\mu}\boldsymbol{\Psi}_j\right)\ , \tag{3.2d}$$

$$\boldsymbol{\Psi}_j(k)|\bar{\boldsymbol{\Psi}} \sim \mathcal{G}^{-1}\left(\frac{n_\Psi}{2},\ \frac{n_\Psi}{2}\psi_k\right)\ , \tag{3.2e}$$

as well as the following Bernoulli prior for m_j:

$$m_j \sim Bern\left(\frac{1}{j}\right)\ . \tag{3.2f}$$

The prior on the activations m_j imposes an increasing penalty on the number of linear experts the learning phase tries to allocate: this will be analysed with more detail in Section 3.2.2. As for the remaining priors, W^{-1} and $\mathcal{G}^{-1}$ denote multivariate Inverse-Wishart and univariate Inverse-Gamma distributions, respectively. $\boldsymbol{\Psi}_j(k)$ is the k^{th} element of the diagonal of $\boldsymbol{\Psi}_j$, while $\boldsymbol{\Lambda}_j(k)$ corresponds to the k^{th} row of $\boldsymbol{\Lambda}_j$. Constants n_ν, n_Σ, n_μ, n_Λ and n_Ψ determine the "strength" of the respective priors, $\boldsymbol{\nu}_{0j}$, $\bar{\boldsymbol{\Sigma}}$, $\boldsymbol{\mu}_{0j}$, $\boldsymbol{\Lambda}_0$ and $\bar{\boldsymbol{\Psi}}$, expressed as an equivalent number of "fake" data points. These distributions are chosen for convenience, since they are conjugate priors for the observed data distribution. The purpose of the common prior distribution for covariance matrices $\boldsymbol{\Sigma}_j$, governed by diagonal matrix $\bar{\boldsymbol{\Sigma}}$, is threefold: it introduces some regularization, so that $\boldsymbol{\Sigma}_j$ has always an inverse; it ensures that the experts input regions shapes do not differ too much from each other, and finally, it prevents non-neighbouring experts from competing for the same data in the initial phase of the learning process of each expert — a serious problem occurring in mixtures of experts models, referred for instance in (Schaal and Atkeson, 1998; Vijayakumar, D'Souza, and Schaal, 2005), thus enforcing the principle of localized learning. The Normal prior on $\boldsymbol{\nu}_j$, on the other hand, controls the degree of "mobility" of this parameter: $n_\nu = 0$ makes it dependent solely on the training data, while $n_\nu = \infty$ leads to a fixed centre, as typically occurs in radial basis networks or in the LWPR algorithm introduced in Chapter 2; the prior on $\boldsymbol{\mu}_j$ has the same purpose of controlling the influence of training data on this parameter. Finally, the Inverse-Gamma prior on $\boldsymbol{\Psi}_j$ defines a kind of "average" noise that is shared by all experts, while the prior $\boldsymbol{\Lambda}_0 = \mathbf{0}$ for the rows of $\boldsymbol{\Lambda}_j$ performs a coefficient shrinkage similar to ridge regression; its main purpose, however, is to impose a regularization mechanism, in order to make the matrix inversion required when estimating this parameter always full rank. Such prior introduces some bias in the

regression coefficients, and consequently n_Λ should be kept to a low value in order to make such undesired effect negligible.

Diagonal matrices $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$, with diagonal elements σ_k and ψ_k respectively, represent the prior knowledge for the common structure of $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\Psi}_j$: these values strongly depend on the specific map to learn, particularly its characteristic input length-scale and the output noise. It is highly desirable that such hyperparameters are learned from training data, and so some vague hyper-priors are defined for these, here represented by scaled inverse chi-squared distributions, to avoid relying on such problem specific information:

$$\sigma_k \sim \text{Scale-Inv-}\chi^2(n_\sigma, \sigma_{0k}) \ , \tag{3.3a}$$

$$\psi_k \sim \text{Scale-Inv-}\chi^2(n_\psi, \psi_{0k}) \ , \tag{3.3b}$$

with σ_{0k} and ψ_{0k} standing respectively for the k^{th} diagonal elements of $\boldsymbol{\Sigma}_0$ and $\boldsymbol{\Psi}_0$, diagonal matrices representing the initial guesses for $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$. Free parameters n_σ and n_ψ control these hyper-priors strength: setting $n_\sigma = 0$ and $n_\psi = 0$ results in the uninformative priors $p(\sigma_k) \propto 1/\sigma_k$ and $p(\psi_k) \propto 1/\psi_k$, respectively. The (infinite) parameter vector $\boldsymbol{\Theta}$ that defines this mixture, to be learned from the data, is consequently given by $\boldsymbol{\Theta} = \left\{\bar{\boldsymbol{\Sigma}}, \bar{\boldsymbol{\Psi}}\right\} \cup \left\{\boldsymbol{\nu}_j, \boldsymbol{\Sigma}_j, \boldsymbol{\mu}_j, \boldsymbol{\Lambda}_j, \boldsymbol{\Psi}_j, m_j\right\}_{(1 \le j \le \infty)}$, and the graphical model corresponding to the above described probabilistic model is shown in Figure 3.2.

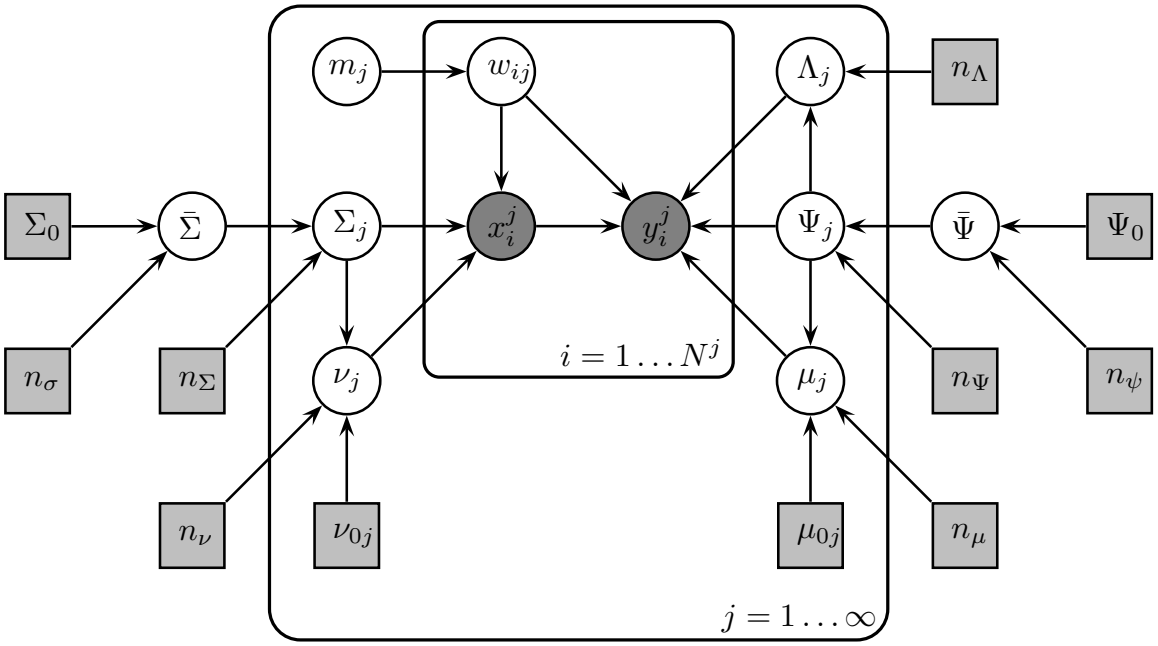


Figure 3.2: Graphical model representing the infinite mixture of linear experts. Lightly shaded rectangular boxes represent fixed parameters. Observed data points are grouped according to their label: $\boldsymbol{x}_i^j$ and $\boldsymbol{y}_i^j$ represent the i$^{\text{th}}$ data point generated by expert j, while N^j is the total number of training points generated by such expert.

3.2 Training

Given $\boldsymbol{X}$, a collection of input training data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$ and $\boldsymbol{Y}$, the corresponding output $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_N\}$, the classical Bayesian inference approach to the above model first builds the posterior parameter vector distribution given the observed training data, $p(\boldsymbol{\Theta}|\boldsymbol{X}, \boldsymbol{Y})$, and then uses it to obtain $p(\boldsymbol{y}|\boldsymbol{x}_q, \boldsymbol{X}, \boldsymbol{Y})$ and $p(\boldsymbol{x}|\boldsymbol{y}_q, \boldsymbol{X}, \boldsymbol{Y})$, respectively the forward and inverse posterior predictive distributions, by marginalization over the parameter vector $\boldsymbol{\Theta}$ and the latent variables $\boldsymbol{W} = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_N\}$, *i.e.*,

$$p(\boldsymbol{y}|\boldsymbol{x}_q, \boldsymbol{X}, \boldsymbol{Y}) = \sum \int p(\boldsymbol{y}|\boldsymbol{x}_q, \boldsymbol{\Theta})\; p(\boldsymbol{\Theta}, \boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})\; d\boldsymbol{\Theta}$$

and

$$\begin{aligned} p(\boldsymbol{x}|\boldsymbol{y}_q, \boldsymbol{X}, \boldsymbol{Y}) &= \sum \int p(\boldsymbol{x}|\boldsymbol{y}_q, \boldsymbol{\Theta})\; p(\boldsymbol{\Theta}, \boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})\; d\boldsymbol{\Theta} \\ &= \sum \int \frac{p(\boldsymbol{y}_q|\boldsymbol{x}, \boldsymbol{\Theta}) p(\boldsymbol{x}|\boldsymbol{\Theta})}{p(\boldsymbol{y}_q|\boldsymbol{\Theta})}\; p(\boldsymbol{\Theta}, \boldsymbol{W}|\boldsymbol{X}, \boldsymbol{Y})\; d\boldsymbol{\Theta}\;, \end{aligned}$$

where the sum is over all possible values of $\boldsymbol{W}$. These expressions are however analytically intractable: Markov chain Monte Carlo (MCMC) methods (Andrieu, Freitas, et al., 2003) or variational techniques (Beal, 2003) are two popular training methods for Bayesian models that can provide approximations to the above integrals. Still, these methods, and MCMC in particular, are computationally expensive, and it is difficult to adapt them to an online, incremental learning scheme. Moreover, particular attention must be given to the parameters m_j that define the number of components that compose the mixture. A standard technique consists in defining a Dirichlet prior for the mixture proportions that can automatically find the most appropriate number of components for the mixture, but training such model has high computational demands that are not compatible with online and incremental learning: this will be discussed in more detail in Section 3.2.2.

Instead of following these Bayesian procedures to train the proposed mixture model, in this dissertation an alternative approach is pursued, based on the EM algorithm of Dempster, Laird, and Rubin (1977) to find a maximum a posteriori (MAP) estimate for the unknown parameter vector $\boldsymbol{\Theta}$, due to its easy adaptation to online learning schemes.

The log-likelihood of parameter vector $\boldsymbol{\Theta}$, given the complete training data $\{\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}\}$, is given by

$$L(\boldsymbol{\Theta}; \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}) = \log \left[p(\boldsymbol{\Theta}) \prod_{i=1}^{N} p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}_i; \boldsymbol{\Theta}) p(\boldsymbol{x}_i|\boldsymbol{w}_i; \boldsymbol{\Theta}) p(\boldsymbol{w}_i; \boldsymbol{\Theta}) \right]\;,$$

where $p(\boldsymbol{\Theta})$ encompasses the priors defined in (3.2) and (3.3). The probabilities appearing

in the above equation are given by

$$p(\boldsymbol{y}_i|\boldsymbol{x}_i, \boldsymbol{w}_i; \boldsymbol{\Theta}) = \prod_{j=1}^{\infty} p(\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ij}; \boldsymbol{\Theta})^{w_{ij}} \ ,$$
$$p(\boldsymbol{x}_i|\boldsymbol{w}_i; \boldsymbol{\Theta}) = \prod_{j=1}^{\infty} p(\boldsymbol{x}_i|w_{ij}; \boldsymbol{\Theta})^{w_{ij}} \quad \text{and}$$
$$p(\boldsymbol{w}_i; \boldsymbol{\Theta}) = \prod_{j=1}^{\infty} p(w_{ij}; \boldsymbol{\Theta})^{w_{ij}} \ .$$

Maximization of this log-likelihood with respect to $\boldsymbol{\Theta}$ is not feasible due to the presence of the latent variables $\boldsymbol{W}$. However, application of the EM algorithm to this log-likelihood produces a sequence of estimates $\hat{\boldsymbol{\Theta}}^t$ that are guaranteed to converge to a local maxima of the log-likelihood of the observed data $\{\boldsymbol{X}, \boldsymbol{Y}\}$. This algorithm alternates between the expectation step (E-Step), which calculates the Q-function $Q(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}^t)$, the conditional expectation of $L(\boldsymbol{\Theta}; \boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{W})$ with respect to the latent variables $\boldsymbol{W}$, for the current value of $\hat{\boldsymbol{\Theta}}^t$, and the maximization step (M-Step), that finds the new value of $\hat{\boldsymbol{\Theta}}^{t+1}$ given the previous expectation.

3.2.1 E-Step

The log-likelihood, given by

$$L(\boldsymbol{\Theta}; \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{W}) = \log p(\boldsymbol{\Theta}) + \sum_{i=1}^{N} \sum_{j=1}^{\infty} w_{ij} \left[\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ij}) + \log p(\boldsymbol{x}_i|w_{ij}; \boldsymbol{\Theta}) + \log p(w_{ij}; \boldsymbol{\Theta})\right] \ , \tag{3.4}$$

is clearly linear with respect to the latent variables w_{ij}, and hence it suffices, to obtain $Q(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}^t)$, to calculate $h_{ij}^t = \mathbb{E}[w_{ij}|\boldsymbol{Y}, \boldsymbol{X}; \hat{\boldsymbol{\Theta}}^t]$, the estimate of the posterior probability that data point i was effectively generated by expert j, also called the *responsibility* that expert j has generated data point i. Since w_{ij} only depends on training point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ this results, using Bayes' theorem,

$$\begin{aligned} h_{ij}^t &\equiv \mathbb{E}[w_{ij}|\boldsymbol{Y}, \boldsymbol{X}; \hat{\boldsymbol{\Theta}}^t] \\ &= \mathbb{E}[w_{ij}|\boldsymbol{y}_i, \boldsymbol{x}_i; \hat{\boldsymbol{\Theta}}^t] \\ &= p(w_{ij}|\boldsymbol{y}_i, \boldsymbol{x}_i; \hat{\boldsymbol{\Theta}}^t) \\ &= \frac{p(\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ij}; \hat{\boldsymbol{\Theta}}^t) p(\boldsymbol{x}_i|w_{ij}; \hat{\boldsymbol{\Theta}}^t) \ \hat{m}_j^t}{\sum_{k=1}^{\infty} p(\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ik}; \hat{\boldsymbol{\Theta}}^t) p(\boldsymbol{x}_i|w_{ik}; \hat{\boldsymbol{\Theta}}^t) \ \hat{m}_k^t} \ . \end{aligned}$$

As a result, the Q-function becomes

$$\begin{aligned} Q(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}^t) &\equiv \mathbb{E}_W[L(\boldsymbol{\Theta}; \boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{W}); \hat{\boldsymbol{\Theta}}^t] \\ &= \log p(\boldsymbol{\Theta}) + \sum_{i=1}^{N} \sum_{j=1}^{\infty} h_{ij}^t \left[\log p(\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ij}; \boldsymbol{\Theta}) + \log p(\boldsymbol{x}_i|w_{ij}; \boldsymbol{\Theta}) + \log p(w_{ij}; \boldsymbol{\Theta})\right] , \end{aligned} \tag{3.5}$$

where the responsibilities h_{ij}^t depend on the current estimate $\hat{\boldsymbol{\Theta}}^t$.

The complete data log-likelihood in (3.4) belongs to the *exponential family*, and thus the Q-function depends only on training data through $\boldsymbol{S}^t$, the expected value of the sufficient statistics vector, given the observed data and the current value of the parameter vector $\hat{\boldsymbol{\Theta}}^t$. For the mixture model (3.1-3.2), it comprises the terms S_{hj}^t, $\boldsymbol{S}_{hxj}^t$, $\boldsymbol{S}_{hyj}^t$, $\boldsymbol{S}_{hxxj}^t$, $\boldsymbol{S}_{hyyj}^t$ and $\boldsymbol{S}_{hyxj}^t$, for $1 \leq j \leq \infty$, defined as follows:

$$\begin{aligned} S_{hj}^t &= \textstyle\sum_{i=1}^N s_{hj}^t(i) & s_{hj}^t(i) &= h_{ij}^t , \\ \boldsymbol{S}_{hxj}^t &= \textstyle\sum_{i=1}^N \boldsymbol{s}_{hxj}^t(i) & \boldsymbol{s}_{hxj}^t(i) &= h_{ij}^t \boldsymbol{x}_i , \\ \boldsymbol{S}_{hyj}^t &= \textstyle\sum_{i=1}^N \boldsymbol{s}_{hyj}^t(i) & \boldsymbol{s}_{hyj}^t(i) &= h_{ij}^t \boldsymbol{y}_i , \\ \boldsymbol{S}_{hxxj}^t &= \textstyle\sum_{i=1}^N \boldsymbol{s}_{hxxj}^t(i) & \boldsymbol{s}_{hxxj}^t(i) &= h_{ij}^t \boldsymbol{x}_i \boldsymbol{x}_i^T , \\ \boldsymbol{S}_{hyxj}^t &= \textstyle\sum_{i=1}^N \boldsymbol{s}_{hyxj}^t(i) & \boldsymbol{s}_{hyxj}^t(i) &= h_{ij}^t \boldsymbol{y}_i \boldsymbol{x}_i^T , \\ \boldsymbol{S}_{hyyj}^t &= \textstyle\sum_{i=1}^N \boldsymbol{s}_{hyyj}^t(i) & \boldsymbol{s}_{hyyj}^t(i) &= h_{ij}^t \boldsymbol{y}_i \boldsymbol{y}_i^T . \end{aligned} \tag{3.6}$$

The quantity $\boldsymbol{s}_i^t = \{s_{hj}^t(i), \boldsymbol{s}_{hxj}^t(i), \boldsymbol{s}_{hyj}^t(i), \boldsymbol{s}_{hxxj}^t(i), \boldsymbol{s}_{hyxj}^t(i), \boldsymbol{s}_{hyyj}^t(i)\}_{(1 \leq j \leq \infty)}$ is also defined for convenience, so that $\boldsymbol{S}^t = \sum_{i=1}^N \boldsymbol{s}_i^t$.

Performing the above E-Step requires the availability of all observed data, which of course is not admissible during online training. Neal and Hinton (1999) present a view of the standard EM algorithm that allows for partial E-Steps to be implemented, resulting in an incremental version of EM. It consists, at iteration t, in performing an update of the sufficient statistics using solely a particular data point i, according to $\boldsymbol{S}^t = \boldsymbol{S}^{t-1} + \boldsymbol{s}_i^t - \boldsymbol{s}_i^{t-1}$, instead of the whole dataset as in equations (3.6). For a continuous stream of data, each point is visited and used only once, and thus its index i can be associated with corresponding iteration number t. The partial E-Step can be written in this case as

$$\boldsymbol{S}^t = \boldsymbol{S}^{t-1} + \boldsymbol{s}^t , \tag{3.7}$$

where $\boldsymbol{s}^t \equiv \boldsymbol{s}_i^t$ for data point $i = t$. A more general result, provided recently by Capp and Moulines (2009), defines more general conditions for convergence of online EM algorithms, suggesting the following E-Step:

$$\bar{\boldsymbol{S}}^t = \bar{\boldsymbol{S}}^{t-1} + \gamma_t(\boldsymbol{s}^t - \bar{\boldsymbol{S}}^{t-1}) ,$$

where γ_t is a step-size and $\bar{\boldsymbol{S}}$ is an alternative set of sufficient statistics. Setting $\gamma_t = t^{-\alpha}$, for $\alpha \in (0.5, 1]$, guarantees the algorithm convergence under some mild assumptions, while introducing a time decay in the sufficient statistics that may be beneficial when slowly time varying data is presented to the algorithm: such situation may eventually occur within the context of robotic applications. Sufficient statistics $\bar{\boldsymbol{S}}$ and $\boldsymbol{S}$ are related according to $\bar{\boldsymbol{S}}^t = \gamma_t \boldsymbol{S}^t$, and thus the above equation can be reformulated to become

$$\boldsymbol{S}^t = \lambda_t \boldsymbol{S}^{t-1} + \boldsymbol{s}^t \ , \quad \text{where} \quad \lambda_t = \gamma_{t-1}(\gamma_t^{-1} - 1) \ ; \tag{3.8}$$

this is the decaying statistics formulation presented in (Sato and Ishii, 2000). Setting $\gamma_t = t^{-1}$ corresponds to having $\lambda_t = 1$, *i.e.*, an accumulation of the sufficient statistics with no forgetting over the time, equivalent to (3.7).

3.2.2 M-Step

M-Step picks the parameter vector $\boldsymbol{\Theta}$ that maximizes the current value of the Q-function. The fact that most of the priors are conjugate to the data likelihood can be used to arrive at the following new estimates — see Appendix A for details —, for $j \in \mathcal{M}^t$, where $\mathcal{M}^t = \{j \in \mathbb{N} : \hat{m}_j^t = 1\}$ is the set containing the experts effectively contributing to the mixture at iteration t:

$$\hat{\boldsymbol{\nu}}_j^{t+1} = \frac{\boldsymbol{S}_{hxj}^t + n_\nu \boldsymbol{\nu}_{0j}}{\boldsymbol{S}_{hj}^t + n_\nu} \ , \tag{3.9a}$$

$$\hat{\boldsymbol{\Sigma}}_j^{t+1} = \frac{\boldsymbol{S}_{hxxj}^t - (\boldsymbol{S}_{hj}^t + n_\nu)\hat{\boldsymbol{\nu}}_j^{t+1}\hat{\boldsymbol{\nu}}_j^{t+1^T} + n_\Sigma \hat{\bar{\boldsymbol{\Sigma}}}^{t+1} + n_\nu \boldsymbol{\nu}_{0j}\boldsymbol{\nu}_{0j}^T}{\boldsymbol{S}_{hj}^t + n_\Sigma + d + 2} \ , \tag{3.9b}$$

$$\hat{\boldsymbol{\Lambda}}_j^{t+1} = \left(\boldsymbol{S}_{hyxj}^t - \frac{\boldsymbol{S}_{hyj}^t + n_\mu \boldsymbol{\mu}_{0j}}{\boldsymbol{S}_{hj}^t + n_\mu}(\boldsymbol{S}_{hxj}^t)^T\right)\left(n_\Lambda \mathbf{I} + \boldsymbol{S}_{hxxj}^t - \frac{\boldsymbol{S}_{hxj}^t(\boldsymbol{S}_{hxj}^t)^T}{\boldsymbol{S}_{hj}^t + n_\mu}\right)^{-1} \ , \tag{3.9c}$$

$$\hat{\boldsymbol{\mu}}_j^{t+1} = \frac{\boldsymbol{S}_{hyj}^t + n_\mu \boldsymbol{\mu}_{0j}}{\boldsymbol{S}_{hj}^t + n_\mu} + \hat{\boldsymbol{\Lambda}}_j^{t+1}\left(\hat{\boldsymbol{\nu}}_j^{t+1} - \frac{\boldsymbol{S}_{hxj}^t}{\boldsymbol{S}_{hj}^t + n_\mu}\right) \quad \text{and} \tag{3.9d}$$

$$\hat{\boldsymbol{\Psi}}_j^{t+1} = \frac{n_\Psi \hat{\bar{\boldsymbol{\Psi}}}^{t+1} + \operatorname{diag}\left\{\boldsymbol{S}_{hyyj}^t - \hat{\boldsymbol{\Lambda}}_j^{t+1}(\boldsymbol{S}_{hyxj}^t)^T - (\hat{\boldsymbol{\mu}}_j^{t+1} - \hat{\boldsymbol{\Lambda}}_j^{t+1}\hat{\boldsymbol{\nu}}_j^{t+1})(\boldsymbol{S}_{hyj}^t + n_\mu \boldsymbol{\mu}_{0j})^T\right\}}{n_\Psi + S_{hj}^t + 2} \ ; \tag{3.9e}$$

$\operatorname{diag}\{\cdot\}$ denotes a diagonal matrix equal to the diagonal of its argument. For the common input variance parameter $\bar{\boldsymbol{\Sigma}}$, however, the partial derivatives of $Q(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}^t)$ with respect to each σ_k must be obtained and equated to zero, resulting in

$$\hat{\sigma}_k^{t+1} = \frac{\left(\frac{M^{t+1}}{2} - \frac{n_\sigma + 1}{n_\Sigma}\right) + \sqrt{\left(\frac{M^{t+1}}{2} - \frac{n_\sigma + 1}{n_\Sigma}\right)^2 + 2\frac{n_\sigma}{n_\Sigma}\sigma_{0k}\sum_{j \in \mathcal{M}^{t+1}} \hat{\boldsymbol{\Sigma}}_j^{-1}(k)^{t+1}}}{\sum_{j \in \mathcal{M}^{t+1}} \hat{\boldsymbol{\Sigma}}_j^{-1}(k)^{t+1}} \ , \tag{3.9f}$$

where $\hat{\mathbf{\Sigma}}_j^{-1}(k)^{t+1}$ denotes the k^{th} element of the diagonal of the inverse of $\hat{\mathbf{\Sigma}}_j^{t+1}$ and M^t corresponds to the effective number of experts in the mixture at iteration t, *i.e.*, $M^t = \sum_{j=1}^{\infty} \hat{m}_j^t$. Using the same procedure, a similar result holds for $\bar{\mathbf{\Psi}}$:

$$\hat{\psi}_k^{t+1} = \frac{\left(\frac{M^{t+1}}{2} - \frac{n_\psi+1}{n_\Psi}\right) + \sqrt{\left(\frac{M^{t+1}}{2} - \frac{n_\psi+1}{n_\Psi}\right)^2 + 2\frac{n_\psi}{n_\Psi}\psi_{0k}\sum_{j\in\mathcal{M}^{t+1}} \hat{\mathbf{\Psi}}_j^{-1}(k)^{t+1}}}{\sum_{j\in\mathcal{M}^{t+1}} \hat{\mathbf{\Psi}}_j^{-1}(k)^{t+1}} \,. \tag{3.9g}$$

Equations (3.9b) and (3.9f), on one hand, and (3.9e) and (3.9g), on the other, are coupled, without an explicit closed form solution for the parameters being estimated. To deal with this issue the maximization step is relaxed and each of these parameters is maximized individually, conditionally on the others remaining fixed. This corresponds to the expectation conditional maximization algorithm, a particular case of the generalized variant of the EM algorithm, where the M-Step is modified to an update that improves the Q-function, without necessarily maximizing it (Dempster, Laird, and Rubin, 1977). Solving for the values of $\hat{m}_j^{t+1}$ that maximize the likelihood is however intractable, as it requires evaluating all the infinite combinations of values for m_j and picking the one that maximizes the Q-function. A procedure to set the values of $\hat{m}_j^{t+1}$ at each iteration of the EM algorithm is discussed next.

Growing the Mixture:

Activations m_j define the number of experts constituting the current mixture, and play a key role in defining the complexity of the global probabilistic model. Choosing the appropriate number of components for a mixture is a difficult problem and several methods have been proposed to deal with it; among them, Bayesian methods provide an elegant framework that automatically generates a trade-off between the fitness of the data to the model and the complexity of the same model. Moreover, the infinite mixture models based on the Dirichlet process nonparametric prior for the mixing coefficients allow for generative models where the number of components of the mixture is not defined *a priori* (Antoniak, 1974; Rasmussen and Ghahramani, 2002; Meeds and Osindero, 2006).

Unfortunately, training these infinite mixtures usually requires either computational expensive Markov Chain Monte Carlo sampling methods or variational approaches that typically rely on some sort of truncation that imposes a bound on the admissible number of components for the mixture. Furthermore, the desired online training for this probabilistic model creates some additional difficulties: while, in an offline setting, Bayesian methods can in principle efficiently grow or annihilate mixture components, when operating incrementally the full set of training points is no longer available, and decisions concerning the allocation or removal of components of the mixture must be made resorting only to the most recently available training points and the current mixture state — this is a far more demanding learning challenge. Sato (2001) derives an online variational

Bayesian algorithm for learning mixture models, but it requires the maintenance of parallel hypothesis about the number of components that can easily become too computational expensive. Some recent works also view the variational Bayesian learning model under an online perspective, but are based on non-deterministic approaches based on Gibbs sampling (Wang and Blei, 2012) or require processing the training points in smaller batches of data (Gomes, Welling, and Perona, 2008). For all these methods, an adequate computational speed that allows the processing of hundreds or thousands of samples per second, as required for online learning schemes for robotic applications, is yet to be shown.

As opposed to Bayesian methods, it is more difficult to assess an optimal value for the number of components of a mixture when using a EM algorithm for training: a well known drawback of these EM based techniques is the fact that the maximized observed data likelihood will never decrease when new components are added to the mixture (Figueiredo and Jain, 2002). This allows the derivation of a broad class of criteria to decide when to add new components, ranging from only allowing the existence of a single component, which for the IMLE model would be equivalent to performing a global linear regression on the data, to the activation of a new component for each data point processed, which in turn would correspond to some kind of memory based learning approach, where predictions would be made resorting to all available training data. As a consequence, most EM deterministic methods impose some kind of penalty over the number of mixture components during optimization, such as the Akaike's information criterion (AIC) (Akaike, 1974), the Bayesian inference criterion (BIC) (Schwarz, 1978) or the minimum message length criterion (MML) (Wallace and Boulton, 1968), to name just a few: a comprehensive comparison and review of these kind of penalty methods can be found in (McLachlan and Peel, 2000).

In the probabilistic model given in (3.1-3.2), the prior distribution on m_j plays the role of such penalty, making a high number of experts increasingly less probable. However, changing, at iteration t, the value of a particular parameter m_{j+} from 0 to 1 will result in the following variation of the Q-function:

$$\begin{aligned}
Q(\boldsymbol{\Theta}^+, \hat{\boldsymbol{\Theta}}^t) - Q(\boldsymbol{\Theta}, \hat{\boldsymbol{\Theta}}^t) &= \log \frac{p(\boldsymbol{\Theta}^+)}{p(\boldsymbol{\Theta})} + \sum_{i=1}^{N} \sum_{j=1}^{\infty} h_{ij}^t \log \frac{p(w_{ij}; \boldsymbol{\Theta}^+)}{p(w_{ij}; \boldsymbol{\Theta})} \\
&= \log \frac{p(m_{j+} = 1)}{p(m_{j+} = 0)} + \sum_{i=1}^{N} \sum_{j=1}^{\infty} h_{ij}^t \log \frac{\frac{1}{M^t+1}}{\frac{1}{M^t}} \\
&= \log \frac{\frac{1}{j^+ + 1}}{1 - \frac{1}{j^+ + 1}} + \log \frac{1}{M^t + 1} \sum_{i=1}^{N} \sum_{j=1}^{\infty} h_{ij}^t \\
&= \log \frac{1}{j^+} + N \log \frac{M^t}{M^t + 1}
\end{aligned}$$

where $\boldsymbol{\Theta}^+$ is the parameter vector corresponding to $m_{j+} = 1$; equations (3.1c) and (3.2f)

are used in the above derivation, together with the fact that responsibilities h_{ij} sum to 1 over the experts set.

This is a problem: activating a new expert will always lead to a decrease of the Q-function; moreover, this decrease does not even depend on the training data observed so far by the probabilistic model. However, although momentarily decreasing the Q-function, activating a new expert can, nevertheless, increase the observed data likelihood in the subsequent iteration. Of course, under the online paradigm herein followed it is not possible to calculate the likelihood of the entire observed data, as each training point is discarded after the corresponding update of the mixture sufficient statistics. However, at the end of M-Step, the log-likelihood of the next training point $(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1})$, under the new parameter vector $\hat{\boldsymbol{\Theta}}^{t+1}$, can be evaluated. This log-likelihood is given, considering also the priors on $\boldsymbol{\Theta}$, by $L^{t+1}(\hat{\boldsymbol{\Theta}}^{t+1}) = \log[p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|\hat{\boldsymbol{\Theta}}^{t+1})p(\hat{\boldsymbol{\Theta}}^{t+1})]$, where $p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|\hat{\boldsymbol{\Theta}}^{t+1})$ is obtained from the complete data likelihood by marginalizing out the latent variables w_{ij}:

$$\begin{aligned} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|\hat{\boldsymbol{\Theta}}^{t+1}) &= \sum_{j=1}^{\infty} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|w_j, \hat{\boldsymbol{\Theta}}^{t+1})p(w_j|\hat{\boldsymbol{\Theta}}^{t+1}) \\ &= \frac{1}{M^t}\sum_{j=1}^{M^t} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|w_j, \hat{\boldsymbol{\Theta}}^{t+1}) \ . \end{aligned}$$

If, alternatively, expert $j^+ = M^t + 1$ is activated at the end of the M-Step, by making $\hat{m}_{j^+}^{t+1} = 1$ and initializing $\boldsymbol{\mu}_{0j^+}$ and $\boldsymbol{\nu}_{0j^+}$ to $\boldsymbol{y}_{t+1}$ and $\boldsymbol{x}_{t+1}$ respectively, the alternative log-likelihood $L^{t+1}(\hat{\boldsymbol{\Theta}}_+^{t+1}) = \log[p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|\hat{\boldsymbol{\Theta}}_+^{t+1})p(\hat{\boldsymbol{\Theta}}_+^{t+1})]$ is obtained, where $\hat{\boldsymbol{\Theta}}_+^{t+1}$ is a modified version of parameter vector $\hat{\boldsymbol{\Theta}}^{t+1}$, with expert j^+ activated, and where the next training point likelihood is now given by:

$$\begin{aligned} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|\hat{\boldsymbol{\Theta}}_+^{t+1}) &= \sum_{j=1}^{\infty} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|w_j, \hat{\boldsymbol{\Theta}}_+^{t+1})p(w_j|\hat{\boldsymbol{\Theta}}_+^{t+1}) \\ &= \frac{1}{M^t+1}\left[\sum_{j=1}^{M^t} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|w_j, \hat{\boldsymbol{\Theta}}_+^{t+1}) + p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1})\right] \\ &= \frac{M^t}{M^t+1} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|\hat{\boldsymbol{\Theta}}^{t+1}) + \frac{1}{M^t+1} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1}|w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) \ , \end{aligned}$$

since $p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}}_+) = p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}})$ for $j \neq j^+$.

Activating expert j^+ will increase the log-likelihood of the next training point if $L^{t+1}(\hat{\boldsymbol{\Theta}}_+^{t+1}) > L^{t+1}(\hat{\boldsymbol{\Theta}}^{t+1})$, and this can be used as a criterion for deciding when to activate a new expert. This approach, however, will often lead to too many local models being allocated: instead, a statistical approach of activating a new expert only when strong evidence supports the alternative parameter vector $\hat{\boldsymbol{\Theta}}_+^{t+1}$ against the null hypothesis $\hat{\boldsymbol{\Theta}}^{t+1}$ is considered. This later parameter vector can be seen as a special case of $\hat{\boldsymbol{\Theta}}_+^{t+1}$, with less one mixture component: this suggests using a likelihood ratio test to compare

them, where the test statistic $T = 2L^{t+1}(\hat{\boldsymbol{\Theta}}_+^{t+1}) - 2L^{t+1}(\hat{\boldsymbol{\Theta}}^{t+1})$ approximately follows a chi-squared distribution with degrees of freedom equal to the difference of free number of parameters between $\hat{\boldsymbol{\Theta}}_+^{t+1}$ and $\hat{\boldsymbol{\Theta}}^{t+1}$ (O'Hagan and Forster, 1994). At the time of activation of a new expert j^+ only $\boldsymbol{\mu}_{0j+}$ and $\boldsymbol{\nu}_{0j+}$ are effectively defined, and so the change on the number of free parameters is equal to $d+D$. Let $\mathcal{X}^2(p_0, d+D)$ be the critical value of a chi-squared distribution with $d+D$ degrees of freedom, corresponding to the number of free parameters introduced in the mixture: a new expert should be activated, according to the likelihood ratio, if, for a significance value p_0,

$$T = 2L^{t+1}(\hat{\boldsymbol{\Theta}}_+^{t+1}) - 2L^{t+1}(\hat{\boldsymbol{\Theta}}^{t+1}) > \mathcal{X}^2(p_0, d+D) \; . \tag{3.10}$$

Changing m_{j+} only affects $p(\hat{\boldsymbol{\Theta}}_+^{t+1})$ through the term $p(m_{j+})$, given by (3.2f); since all the remaining terms are not modified, this results in

$$\frac{p(\hat{\boldsymbol{\Theta}}_+^{t+1})}{p(\hat{\boldsymbol{\Theta}}^{t+1})} = \frac{p(m_{j+}=1)}{p(m_{j+}=0)} = \frac{\frac{1}{M^t+1}}{1-\frac{1}{M^t+1}} = \frac{1}{M^t} \; ,$$

and, after some calculations, equation (3.10) indicates that a new expert j^+ should be consequently activated whenever

$$\sum_{j=1}^{M^t} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1} | w_j, \hat{\boldsymbol{\Theta}}^{t+1}) \leq \frac{1}{1+\frac{1-e^{-0.5\mathcal{X}^2(p_0,d+D)}}{M^t}} \cdot \frac{e^{-0.5\mathcal{X}^2(p_0,d+D)}}{M^t} \cdot p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1} | w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) \; , \tag{3.11}$$

where expert j^+ parameters are equal to their prior values (expert j^+ has not yet accumulated any sufficient statistics). Note that the probability in the right-side of the equation above can be decomposed as

$$p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1} | w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) = p(\boldsymbol{y}_{t+1} | \boldsymbol{x}_{t+1}, w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) p(\boldsymbol{x}_{t+1} | w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) \; ,$$

where, according to (3.1),

$$\begin{aligned} \boldsymbol{y}_{t+1} | \boldsymbol{x}_{t+1}, w_{j+}; \boldsymbol{\Theta} &\sim \mathcal{N}\left(\boldsymbol{\mu}_{0j} + \boldsymbol{\Lambda}_0(\boldsymbol{x}_{t+1} - \boldsymbol{\nu}_{0j}), \hat{\hat{\boldsymbol{\Psi}}}^{t+1}\right) \; , \\ \boldsymbol{x}_{t+1} | w_{j+}; \boldsymbol{\Theta} &\sim \mathcal{N}\left(\boldsymbol{\nu}_{0j}, \hat{\hat{\boldsymbol{\Sigma}}}^{t+1}\right) \; ; \end{aligned}$$

since $\boldsymbol{\nu}_{0j+} = \boldsymbol{x}_{t+1}$ and $\boldsymbol{\mu}_{0j+} = \boldsymbol{y}_{t+1}$, this results in

$$\begin{aligned} p(\boldsymbol{y}_{t+1}, \boldsymbol{x}_{t+1} | w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) &= p(\boldsymbol{y}_{t+1} | \boldsymbol{x}_{t+1}, w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) p(\boldsymbol{x}_{t+1} | w_{j+}, \hat{\boldsymbol{\Theta}}_+^{t+1}) \\ &= \frac{1}{\sqrt{(2\pi)^D |\hat{\hat{\boldsymbol{\Psi}}}^{t+1}|}} \cdot \frac{1}{\sqrt{(2\pi)^d |\hat{\hat{\boldsymbol{\Sigma}}}^{t+1}|}} \; . \end{aligned}$$

The right side of (3.11) introduces an increasingly penalty on the activation of new

experts as the value of M increases. Adjustable parameter p_0 can regulate the propensity to activate new experts: the lower its value the higher the critical value of the chi-squared distribution will be, making the experts activation criterion harder to be met, resulting in fewer components in the mixture. In general, equation (3.11) indicates that a new expert should be activated when the next acquired training point is poorly explained by the current probabilistic model. This is a sensible approach to mixture grow in online algorithms: LWPR, for instance, creates a new linear model each time an input training point $\boldsymbol{x}_i$ fails to activate the nearest receptive field by more than a given fixed threshold. Equation (3.11), on the other hand, imposes a varying threshold that depends on the current number of experts of the mixture and on the current estimates for hyperparameters $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$. Contrary to LWPR, this activation scheme takes into account both the input and output part of each training point, as required to adequately learn multi-valued functions, thus preventing interference between different branches of a multi-valued relation during training; it can also be viewed as a time varying threshold, that changes as the input length-scale and output noise estimates, represented by $\hat{\bar{\boldsymbol{\Sigma}}}$ and $\hat{\bar{\boldsymbol{\Psi}}}$ respectively, are learned from training data by the IMLE algorithm.

Outliers Detection

A customary pre-processing step when learning is performed offline is to discard isolated points that clearly have a large deviation from the true input-output relation responsible for the observed training data. Removing such *outliers* is however a much more sensitive subject when incremental learning is considered, as the notion of "isolated point" fades when it is no longer possible to look at the whole training set.

One possible approach is then to consider a special class w_0 that is assumed to be responsible for the generation of these outliers: assuming this class to have a prior probability $p(w_0|\boldsymbol{\Theta})$ and that some observation model $p(\boldsymbol{y}, \boldsymbol{x}|w_0, \boldsymbol{\Theta})$ exists for this class, then the probability that a training point $(\boldsymbol{y}, \boldsymbol{x})$ is an outlier generated by w_0, given the current mixture parameters, follows from Bayes' rule,

$$p(w_0|\boldsymbol{y}, \boldsymbol{x}, \hat{\boldsymbol{\Theta}}) = \frac{p(\boldsymbol{y}, \boldsymbol{x}|w_0, \hat{\boldsymbol{\Theta}})p(w_0|\hat{\boldsymbol{\Theta}})}{\sum_{j=1}^{\infty} p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}})p(w_j|\hat{\boldsymbol{\Theta}}) + p(\boldsymbol{y}, \boldsymbol{x}|w_0, \hat{\boldsymbol{\Theta}})p(w_0|\hat{\boldsymbol{\Theta}})} \,. \tag{3.12}$$

This posterior probability for w_0 is dominant over the posterior probabilities for the mixture experts if $p(w_0|\boldsymbol{y}, \boldsymbol{x}, \hat{\boldsymbol{\Theta}}) > 0.5$, and this relation can be used to mark a training point as an outlier, removing it from the training set before it is used in the learning process.

A natural choice for $p(w_0|\hat{\boldsymbol{\Theta}})$ is to make it depend on the number of experts activated so far, by making, for instance, $p(w_0|\hat{\boldsymbol{\Theta}}) = 1/M$; considering instead a constant value for this prior would make the outlier posterior probability closer to one as the number of

active experts increased, irrespectively of the values of $p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}})$, which is, of course, an undesired result. With this outlier class w_0 the distribution over the models responsible for generating the training data is no longer a proper distribution, as

$$\begin{aligned} p(w_0|\hat{\boldsymbol{\Theta}}) + \sum_{j=1}^{\infty} p(w_j|\hat{\boldsymbol{\Theta}}) &= \frac{1}{M} + \sum_{j=1}^{M} \frac{1}{M} \\ &= \frac{1}{M} + 1 \\ &> 1 \ ; \end{aligned}$$

however, this is not a problem as far as posterior distribution (3.12) is concerned, since any normalization factor for this distribution would cancel out in (3.12).

How should the outlier observation model $p(\boldsymbol{y}, \boldsymbol{x}|w_0, \boldsymbol{\Theta})$ be defined? The most straightforward answer is simply to assign it a constant probability, making $p(\boldsymbol{y}, \boldsymbol{x}|w_0, \boldsymbol{\Theta})$ an improper distribution. A significant problem that arises with such approach is that in such case the posterior distribution for w_0, given by equation (3.12), heavily depends on the characteristics of the relation to be learned, since the input length-scale and the output noise of the relation influence the distribution $p(\boldsymbol{y}, \boldsymbol{x}|\hat{\boldsymbol{\Theta}})$ via the normalization constants appearing in the Gaussian distributions for $p(\boldsymbol{y}|\boldsymbol{x}, \hat{\boldsymbol{\Theta}})$ and $p(\boldsymbol{x}|\hat{\boldsymbol{\Theta}})$.

To take a better view on this issue, note that the criterion to decide when an acquired training point $(\boldsymbol{y}, \boldsymbol{x})$ is an outlier, given by $p(w_0|\boldsymbol{y}, \boldsymbol{x}, \hat{\boldsymbol{\Theta}}) > 0.5$ for current estimate $\hat{\boldsymbol{\Theta}}$, results in

$$\sum_{j=1}^{M} p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}}) < p(\boldsymbol{y}, \boldsymbol{x}|w_0, \hat{\boldsymbol{\Theta}}) \ .$$

Thus, for a constant outlier observation model $p(\boldsymbol{y}, \boldsymbol{x}|w_0, \boldsymbol{\Theta}) = K$, the above criterion leads to a simple comparison of $\sum_{j=1}^{M} p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}})$ against a fixed threshold, which must be adequately adjusted for every specific relation to be learned.

Instead, the criterion for outlier detection can be made approximately invariant with respect to input length-scale and output noise of the relation by making it depend on the current estimates for these quantities, represented respectively by $\hat{\bar{\boldsymbol{\Sigma}}}$ and $\hat{\bar{\boldsymbol{\Psi}}}$, making

$$p(\boldsymbol{y}, \boldsymbol{x}|w_0, \hat{\boldsymbol{\Theta}}) = \frac{1}{\sqrt{(2\pi)^D |\hat{\bar{\boldsymbol{\Psi}}}|}} \cdot \frac{1}{\sqrt{(2\pi)^d |\hat{\bar{\boldsymbol{\Sigma}}}|}} \cdot e^{-0.5\mathcal{X}^2(p_0, d+D)} \ ,$$

where, as before, $\mathcal{X}^2(p_0, d+D)$ is the critical value, for significance level p_0, of a chi-squared distribution with $d+D$ degrees of freedom, representing the inverse cumulative distribution function of this distribution evaluated at probability $1 - p_0$.

What is the justification for the above expression? If a newly activated, not yet trained expert (for which $\hat{\boldsymbol{\Psi}}_j = \hat{\bar{\boldsymbol{\Psi}}}$, $\hat{\boldsymbol{\Lambda}}_j = \boldsymbol{\Lambda}_0 = \mathbf{0}$ and $\hat{\boldsymbol{\Sigma}}_j = \hat{\bar{\boldsymbol{\Sigma}}}$) is considered, the expression above can be seen as the evaluation of the probability density function of such expert at a

point $(\boldsymbol{y}, \boldsymbol{x})$ that lies over the equidensity contour that encircles the input-output region, centred at $(\hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\nu}}_j)$, corresponding to a $1 - p_0$ probability. Free parameter p_0 defines what equidensity contour of the density function is used to evaluate this probability. This makes the outlier detection criterion more robust with respect to the input-output relation characteristics, resulting in a value of $p(\boldsymbol{y}, \boldsymbol{x}|w_0, \hat{\boldsymbol{\Theta}})$ that is always comparable to the values of $p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}})$, irrespectively of the input and output dimensions, the estimated input length-scale and the estimated output noise.

There is a close relation between the outlier detection criterion and equation (3.11), used to decide when to activate a new expert in the mixture: alternatively defining $p(w_0|\hat{\boldsymbol{\Theta}}) = (1/M)^2$, meaning that the event that a training point is an outlier become much less probable as more experts are assigned to the mixture, results in the following outlier detection criterion, for a training point $(\boldsymbol{y}, \boldsymbol{x})$:

$$\sum_{j=1}^{M} p(\boldsymbol{y}, \boldsymbol{x}|w_j, \hat{\boldsymbol{\Theta}}) \leq \frac{1}{M} \cdot \frac{1}{\sqrt{(2\pi)^D |\hat{\boldsymbol{\Psi}}|}} \cdot \frac{1}{\sqrt{(2\pi)^d |\hat{\boldsymbol{\Sigma}}|}} \cdot e^{-0.5\chi^2(p_0, d+D)} \,. \tag{3.13}$$

This result is very similar to (3.11), since the first factor on right side of (3.11) quickly approaches 1 as the number of active experts M increases, thus showing that the mechanisms for activating a new expert or recognizing an outlier are essentially the same, and correspond to identifying training points poorly explained by the current mixture. So how does one know then if a training point satisfying (3.11) is an outlier or, alternatively, an indication that a new expert is needed in the corresponding region of the input-output space? This is challenging problem: for single-valued regression, outliers can be detected as training points that have enough support from the current model in the input space, while presenting a large deviation from it in the output space; the same, however, does not happen in multi-valued regression, where this situation may simply correspond to a yet unseen branch of the multi-valued function being learned. Furthermore, the online assumption prevents taking a look at the whole training set, where identification of outliers would in principle be made easier by searching for isolated points.

Perhaps the only way to provide a partial answer to this issue is to make some assumptions about the nature of the distribution of the training data. In particular, if it is assumed that training data is temporally correlated, as most of the times is the case in online learning, a new data point will have a high probability of being poorly explained by the current mixture if the same occurred with the previous point; in contrast, the same does not happen with statistically independent outliers. Observing a sequence of consecutive training points satisfying (3.11) is then more likely to be caused by a lack of fit of the points to the current model than the (very unlikely) occurrence of a sequence of consecutive outliers. This insight is the base for the final criterion to decide when to enlarge the mixture: a new expert should be activated when the previous $N_{outlier}$ training points were

all considered outliers, for $N_{outlier} \geq 1$ (in practice even $N_{outlier} = 1$ should be enough for most cases). While slightly delaying the activation of a new mixture component when a training point is not well represented by the current mixture, such mechanism introduces some robustness towards the existence of outlier data, that would otherwise lead to the creation of unnecessary linear experts to explain such points.

The downside of this approach is the assumption taken on the correlation of training points: these are assumed to be generated from temporal trajectories in the input space; additionally, situations where multi-valued data is generated by randomly sampling between different relation branches are not considered, instead assuming that, apart some occasional switches, consecutive training points are taken from the same branch. If this assumption is not met the algorithm may require more training samples to converge, or may simply produce a mixture model that does not represent the underlying data in a satisfactory manner; however, note that this temporal correlation assumption is almost always met when robotic real-time data acquisition processes are considered.

3.2.3 Computational Complexity

Learning with IMLE is very fast: for a new observation $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, a complete update of the mixture parameters consists of:

1. Model creation: deciding if a new expert should be activated (Equation 3.11);
2. E-Step: assigning responsibilities h_{ij} to active experts (Equation 3.5);
3. E-Step: updating the sufficient statistics (Equation 3.8);
4. M-Step: obtaining the new value for $\hat{\boldsymbol{\Theta}}$ (Equations 3.9).

All these calculations have a computational complexity of $\mathcal{O}(M(d^2 + dD))$, since the matrix inversions required in this process can be efficiently performed using the Sherman-Morrison formula to perform a rank-one update for these quantities. The computational complexity of a complete update of IMLE parameters is consequently linear in D and M, the number of experts, and quadratic in d, the number of input dimensions, making it directly comparable to the best state-of-the-art online algorithms in terms of computational complexity per training point. This computational complexity can be further reduced if a Cholesky decomposition for the inverse of S_{hxxj} is instead maintained and updated in each iteration, once again using rank-one updates, potentially reducing the computational complexity to roughly half when obtaining this quantity and the associated covariance matrices. Also, this complexity can be made linear in d if the input distance metrics $\boldsymbol{\Sigma}_j$ are constrained to be diagonal.

3.3 Prediction

At any instant, the IMLE model can be used to generate predictions, using the state of the mixture that results from the training process with the data points acquired until then. The *forward prediction* provides an output prediction $\hat{\boldsymbol{y}}$ for a particular input query $\boldsymbol{x}_q$; on the other hand, the *inverse prediction* finds an input prediction $\hat{\boldsymbol{x}}$ that is believed to be the responsible for the generation of a desired output $\boldsymbol{y}_q$. Unless the relation to be learned comes from an injective relation, it is very common to have different input values $\boldsymbol{x}$ generating the same output $\boldsymbol{y}$: as a consequence, the inverse prediction mechanism must be able to provide more than a single solution for a query $\boldsymbol{y}_q$. In general, as discussed in previous chapters, there are also some situations that can lead to a forward multi-valued relation, and consequently the forward prediction algorithm must also be prepared to generate multiple solutions for the same input query. Besides the forward and inverse point estimates, given respectively by $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{x}}$, predictions should be accompanied by corresponding uncertainty estimates, represented by covariance matrices $\hat{\boldsymbol{R}}$; also, in the context of forward prediction, the expected output variation resulting from a small perturbation of the input query $\boldsymbol{x}_q$ may be of great relevance: as described in this section, the IMLE model provides the means to obtain all these quantities.

This section will first focus on forward prediction, starting with the single-valued case and then going to the multi-valued situation. After that, inverse prediction is discussed, followed by Jacobian prediction, that finds the aforementioned output change resulting from an infinitesimal change in the input; a quick examination of the computational complexity of the proposed prediction mechanisms concludes the section.

3.3.1 Conditional Probability Distribution and Forward Prediction

Under a full Bayesian paradigm, a forward prediction for an input query $\boldsymbol{x}_q$ is represented by a distribution $p(\boldsymbol{y}|\boldsymbol{x}_q, \boldsymbol{Y}, \boldsymbol{X})$, where the dependence on learned mixture parameters and latent variables is marginalized out. The same occurs in inverse prediction, where now $p(\boldsymbol{x}|\boldsymbol{y}_q, \boldsymbol{Y}, \boldsymbol{X})$ is considered. However, these posterior distributions cannot be analytically calculated for most probabilistic models: instead, EM based learning algorithms will normally provide predictions based on $\hat{\boldsymbol{\Theta}}$, the point estimate for the parameter vector being learned. For IMLE forward prediction, this results in

$$p(\boldsymbol{y}|\boldsymbol{x}_q, \hat{\boldsymbol{\Theta}}) = \sum_{j \in \mathcal{M}} w_j^y(\boldsymbol{x}_q) p(\boldsymbol{y}|\boldsymbol{x}_q, w_j, \hat{\boldsymbol{\Theta}}) \ , \tag{3.14a}$$

where

$$w_j^y(\boldsymbol{x}_q) = p(w_j|\boldsymbol{x}_q, \hat{\boldsymbol{\Theta}}) = \frac{p(\boldsymbol{x}_q|w_j, \hat{\boldsymbol{\Theta}})}{\sum_{k \in \mathcal{M}} p(\boldsymbol{x}_q|w_k, \hat{\boldsymbol{\Theta}})} \ , \tag{3.14b}$$

and where w_j and w_k are a shorthand for $w_{qj} = 1$ and $w_{kj} = 1$; $p(\boldsymbol{y}|\boldsymbol{x}_q, w_j, \hat{\boldsymbol{\Theta}})$ and $p(\boldsymbol{x}_q|w_j, \hat{\boldsymbol{\Theta}})$ follow from (3.1), with $\boldsymbol{\Theta}$ replaced by its estimate.

Even if marginalizing out all the unknown parameters $\boldsymbol{\Theta}$ cannot be analytically performed to provide a full Bayesian conditional distribution for the output, at least some of these parameters can, due to their conjugacy relation to the likelihood of the observed data. Doing so has the benefit of incorporating the uncertainty in these parameters in the final prediction. In particular, given the training data, experts activations $\hat{m} = \{\hat{m}_j\}_{(1 \leq j \leq \infty)}$ and current predictions for hyperparameters $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$, the conditional prediction at an input query point $\boldsymbol{x}_q$ becomes

$$p(\boldsymbol{y}|\boldsymbol{x}_q, \boldsymbol{Y}, \boldsymbol{X}, \hat{\bar{\boldsymbol{\Sigma}}}, \hat{\bar{\boldsymbol{\Psi}}}, \hat{m}) = \sum_{j \in \mathcal{M}} w_j^y(\boldsymbol{x}_q) p(\boldsymbol{y}|\boldsymbol{x}_q, w_j, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Psi}}}) \;, \tag{3.15a}$$

with

$$w_j^y(\boldsymbol{x}_q) = \frac{p(\boldsymbol{x}_q|w_j, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Sigma}}}) p(w_j|\hat{m})}{\sum_{k=1}^{\infty} p(\boldsymbol{x}_q|w_k, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Sigma}}}) p(w_k|\hat{m})} = \frac{p(\boldsymbol{x}_q|w_j, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Sigma}}})}{\sum_{k \in \mathcal{M}} p(\boldsymbol{x}_q|w_k, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Sigma}}})} \;, \tag{3.15b}$$

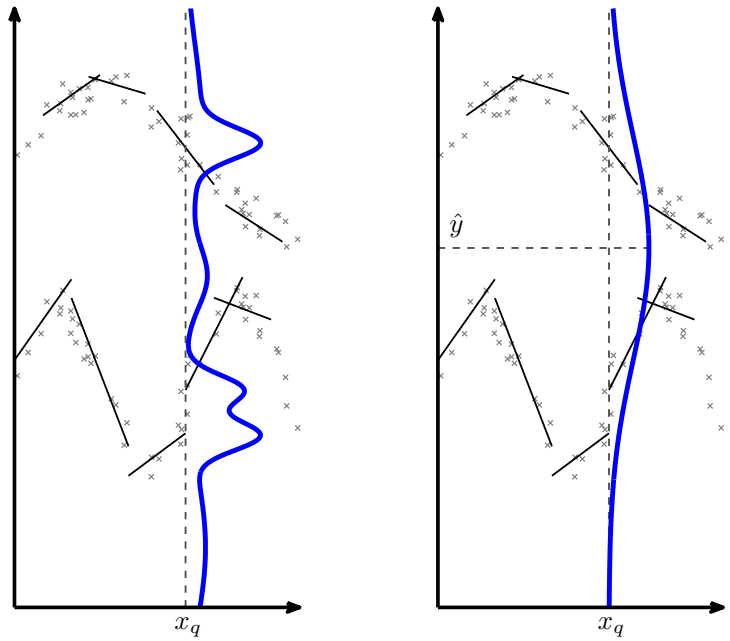
and where the dependence on training data $(\boldsymbol{Y}, \boldsymbol{X})$ is summarized using the sufficient statistics $\boldsymbol{S}^t$; distributions for $p(\boldsymbol{y}|\boldsymbol{x}_q, w_j, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Psi}}})$ and $p(\boldsymbol{x}_q|w_j, \boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Sigma}}})$ are derived in the appendix and are given by (A.6) and (A.1), respectively. To keep the notation simple $\hat{\boldsymbol{\Theta}}^*$ will be used to refer to $(\boldsymbol{S}^t, \hat{\bar{\boldsymbol{\Sigma}}}, \hat{\bar{\boldsymbol{\Psi}}}, \hat{m})$ in the remaining of the section. The conditional density (3.15a) can be understood as a weighted mixture of M Normal densities, each corresponding to a point estimate provided by a different expert, together with an uncertainty value, and where the mixture weights are given by the posterior probabilities that the query point was generated by each expert. An example of such conditional distribution is depicted in Figure 3.3a.

Given the above conditional density, how can a point estimate for $\boldsymbol{y} = f(\boldsymbol{x}_q)$ be obtained, together with an uncertainty measure of this estimate? This is a critical issue: for single-valued forward prediction, such estimate and corresponding uncertainty can be obtained from (3.15a) by taking its mean and variance,

$$\hat{\boldsymbol{y}} = \mathbb{E}[\boldsymbol{y}|\boldsymbol{x}_q, \hat{\boldsymbol{\Theta}}^*] = \sum_j w_j^y \hat{\boldsymbol{y}}_j \quad \text{and} \tag{3.16a}$$

$$\hat{\boldsymbol{R}} = \mathbb{V}[\boldsymbol{y}|\boldsymbol{x}_q, \hat{\boldsymbol{\Theta}}^*] = \sum_j w_j^y \boldsymbol{R}_j^y + \sum_j w_j^y (\hat{\boldsymbol{y}}_j - \hat{\boldsymbol{y}})(\hat{\boldsymbol{y}}_j - \hat{\boldsymbol{y}})^T \;, \tag{3.16b}$$

where $\hat{\boldsymbol{y}}_j$ and $\boldsymbol{R}_j^y$ are respectively the mean and variance of $p(\boldsymbol{y}|\boldsymbol{x}_q, w_j, \hat{\boldsymbol{\Theta}}^*)$, and where from now on the dependence on $\boldsymbol{x}_q$ is dropped for notational convenience. This is an approach followed by many mixture models and it works reasonably well under the single-valued hypothesis, although it tends to overestimate the true variance of the data due to the cross variance between expert estimates, given by the last term in (3.16b). However,



(a) (b)

Figure 3.3: The IMLE model after the training phase: linear models are depicted, in their input activation regions, by straight line segments, while some training points are represented by small grey crosses. Superimposed on this model is the output predictive distribution for a particular input query point $\boldsymbol{x}_q$, represented by a thick blue line, as given by (a) the conditional mixture distribution given by equations (3.15), and (b) the single-valued prediction model given by (3.16).

when multi-valued functions are considered, this approach will only be able to generate a single estimate, together with a large value of the associated uncertainty, as depicted in Figure 3.3b. As reported for instance by Ghahramani and Jordan (1994) and Ghahramani (1994), merging together the distinct solutions provided by each expert might result in a poor overall estimate for a non-convex solution space, where the weighted mean of different experts predictions might itself be far from the true value to estimate: this is clearly unacceptable as far as multi-valued prediction is concerned.

An alternative approach is to search for the modes of the conditional distribution (3.15a), as their location can be a good indicator of the true values of the underlying multi-valued function. This is done, for instance, in (Carreira-Perpiñán, 2000; Qin and Carreira-Perpinan, 2008); however, the distribution (3.15a) can have many low weight spurious modes, corresponding to the contributions of distant experts; in this case some kind of filtering must be done to remove them. Even after removing low weighted components, the topography of the mixture can be very complex in prediction spaces with more than one dimension, as analysed by Ray and Lindsay (2005), where, perhaps counterintuitively, there may exist more modes than mixture components.

Yet another approach, followed in this dissertation, consists in considering a hypoth-

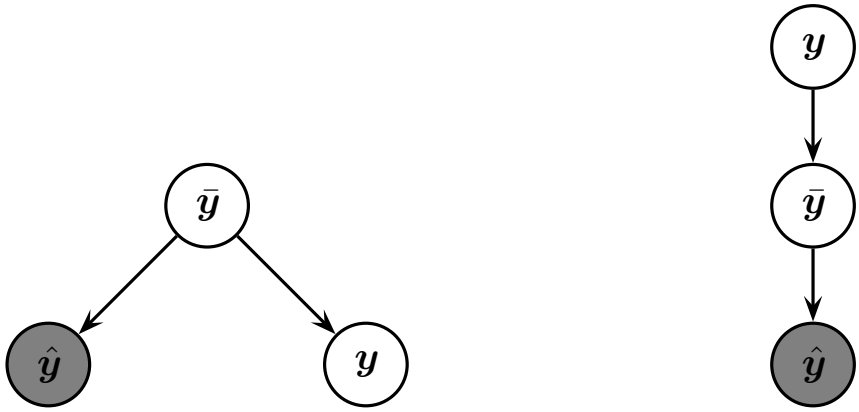


Figure 3.4: Multivariate linear regression graphical model. In (a), an unobserved true mean $\bar{\boldsymbol{y}}$ generates a prediction $\hat{\boldsymbol{y}}$ (observed) and a generic output point $\boldsymbol{y}$. In (b), the same model is rewritten: in this equivalent graphical model an unobserved conditional output mean $\bar{\boldsymbol{y}}$ is generated from an also unknown conditional output distribution $\boldsymbol{y}$, from which, in turn, results a prediction $\hat{\boldsymbol{y}}$.

esised probabilistic model for the true conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$, and then to infer the posterior distribution for its parameters given the current mixture. Such probabilistic model will be described next.

3.3.2 Conditional Generative Probabilistic Model

When single-valued regression is considered and the IMLE model only contains one component, it is relatively simple to infer the conditional distribution at a query point $\boldsymbol{x}_q$, given the current mixture state. The prediction $\hat{\boldsymbol{y}}(\boldsymbol{x}_q)$, provided by the sole linear expert of the mixture, is given by (A.3); this estimate relates to an unknown true conditional mean value $\bar{\boldsymbol{y}}(\boldsymbol{x}_q)$ according to (A.9), while, according to the linear model, observations $\boldsymbol{y}$ are obtained from $\bar{\boldsymbol{y}}$ with variance $\hat{\boldsymbol{\Psi}}_j$ (see equation A.7). This results in the graphical model represented in Figure 3.4, where

$$\boldsymbol{y}|\bar{\boldsymbol{y}}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\bar{\boldsymbol{y}}, \hat{\boldsymbol{\Psi}}) \tag{3.17}$$

and

$$\hat{\boldsymbol{y}}|\bar{\boldsymbol{y}}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\bar{\boldsymbol{y}}, \gamma\hat{\boldsymbol{\Psi}}) \; . \tag{3.18}$$

Consequently, as stated in Appendix A (equation A.6), the posterior distribution for an output sample $\boldsymbol{y}$, given estimate $\hat{\boldsymbol{y}}$, is given by

$$\boldsymbol{y}|\hat{\boldsymbol{y}}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\hat{\boldsymbol{y}}, (1+\gamma)\hat{\boldsymbol{\Psi}}) \; .$$

This result is the well known posterior predictive distribution for the linear regression

model, where the uncertainty in the regression parameters is incorporated in the posterior variance for a new data point $\boldsymbol{y}(\boldsymbol{x}_q)$.

The above equations have a different interpretation that is particularly useful for the mixture case, where the distribution of an observation $\boldsymbol{y}$ is considered unknown and is to be estimated, while the point estimates $\hat{\boldsymbol{y}}_j$, provided by the different components of the mixture, are taken as observations. With this "reverse role" model in mind, equation (3.17) becomes

$$\bar{\boldsymbol{y}}_j|\boldsymbol{y}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\boldsymbol{y}, \hat{\boldsymbol{\Psi}}_j) \ ,$$

where again "observation" $\hat{\boldsymbol{y}}_j$ is calculated using (3.18).

Of course, each linear expert has more or less influence in the final prediction, according to the weights w_j^y; the standard way to incorporate these weights in the generative model is to make the variances of the corresponding observations $\hat{\boldsymbol{y}}_j$ inversely proportional to these weights, following the traditional probabilistic view of weighted least squares and best linear unbiased estimators (Gelman, Carlin, et al., 2004; Vijayakumar, D'Souza, and Schaal, 2005). The above equation thus must be modified to

$$\bar{\boldsymbol{y}}_j|\boldsymbol{y}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\boldsymbol{y}, \hat{\boldsymbol{\Psi}}_j/w_j^y) \ ,$$

and using

$$\hat{\boldsymbol{y}}_j|\bar{\boldsymbol{y}}_j, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\bar{\boldsymbol{y}}_j, \gamma_j\hat{\boldsymbol{\Psi}}_j)$$

results in

$$\hat{\boldsymbol{y}}_j|\boldsymbol{y}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\boldsymbol{y}, \boldsymbol{R}_j) \ , \qquad \text{with} \qquad \boldsymbol{R}_j \equiv \left(\gamma_j + \frac{1}{w_j^y}\right)\hat{\boldsymbol{\Psi}}_j = \varphi_j\hat{\boldsymbol{\Psi}}_j \ . \tag{3.19}$$

The corresponding graphical model is depicted in Figure 3.5. The posterior predictive distribution for an output $\boldsymbol{y}$ at a query location $\boldsymbol{x}_q$ is consequently

$$\boldsymbol{y}|\hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\hat{\boldsymbol{y}}, \hat{\boldsymbol{R}}) \ , \tag{3.20}$$

where

$$\hat{\boldsymbol{y}} = \Big(\sum_j \boldsymbol{R}_j^{-1}\Big)^{-1}\Big(\sum_j \boldsymbol{R}_j^{-1}\hat{\boldsymbol{y}}_j\Big) \qquad \text{and} \tag{3.21a}$$

$$\hat{\boldsymbol{R}} = \Big(\sum_j \boldsymbol{R}_j^{-1}\Big)^{-1} \ . \tag{3.21b}$$

If all experts output noise predictions $\boldsymbol{\Psi}_j$ are the same, and if the uncertainty on the mixture parameters is neglected by making $\gamma_j = 0$, it is worth of notice that point estimate given by equation (3.21a) becomes equal to the mixture conditional mean (3.16a); in the general case, however, the above posterior distribution assigns lower weights to experts

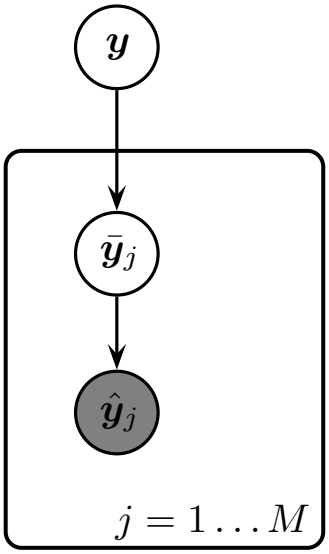


Figure 3.5: Single-valued prediction graphical model.

predictions with higher uncertainties.

Different outputs $\boldsymbol{y}^{(k)}$ must be considered when data points originate from a multi-valued relation, for $k = 1 \cdots K$, where K is the number of multi-valued outputs for the same input query $\boldsymbol{x}_q$. In this situation, to provide a meaningful generative probabilistic model for the output distribution, it is imperative to define the latent indicator variables s_{jk} that signal if $\bar{\boldsymbol{y}}_j$ was indeed generated from multi-valued output $\boldsymbol{y}^{(k)}$. This leads to

$$\bar{\boldsymbol{y}}_j | s_{jk}, \boldsymbol{y}^{(k)}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\boldsymbol{y}^{(k)}, \hat{\boldsymbol{\Psi}}_j / w_j^y) \ ,$$

and again using (3.18) the following result is obtained:

$$\hat{\boldsymbol{y}}_j | s_{jk}, \boldsymbol{y}^{(k)}, \hat{\boldsymbol{\Theta}}^* \sim \mathcal{N}(\boldsymbol{y}^{(k)}, \boldsymbol{R}_j) \ . \tag{3.22}$$

This generative model is depicted in Figure 3.6, where $\boldsymbol{s}_j$ is a vector of size K that aggregates the indicator variables s_{jk} corresponding to expert j, such that one of its elements is equal to one and all the remaining components are zero. The probability distribution of $\hat{\boldsymbol{y}}_j$ given this vector can therefore be expressed as

$$p(\hat{\boldsymbol{y}}_j | \boldsymbol{s}_j, \hat{\boldsymbol{\Theta}}^*) = \prod_{k=1}^{K} p(\hat{\boldsymbol{y}}_j | s_{jk}, \boldsymbol{y}^{(k)}, \hat{\boldsymbol{\Theta}}^*)^{s_{jk}} \ ,$$

from which follows the complete data log-likelihood

$$L\left(\hat{\boldsymbol{y}}_{1\cdots M}, \boldsymbol{s}_{1\cdots M}, \boldsymbol{y}^{(1\cdots K)}\right) = \sum_j \sum_k s_{jk} \log p(\hat{\boldsymbol{y}}_j | s_{jk}, \boldsymbol{y}^{(k)}, \hat{\boldsymbol{\Theta}}^*) \ .$$

Even if a closed form posterior distribution for $\boldsymbol{y}^{(k)}$ cannot be obtained due to the

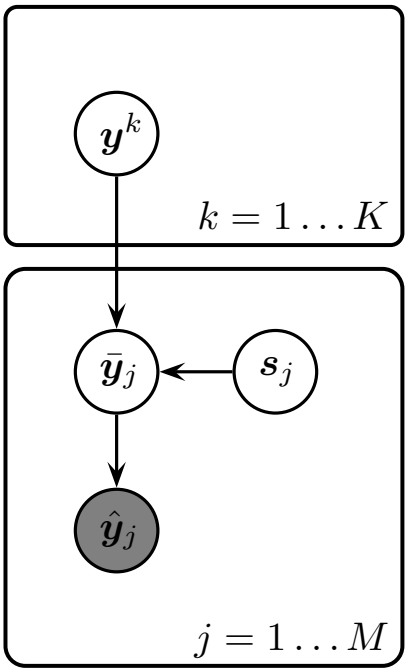


Figure 3.6: Multi-valued prediction graphical model.

presence of the indicator variables s_{jk}, the above likelihood can be used to iterate through a simple EM procedure, in order to obtain $\hat{\boldsymbol{y}}^{(k)}$, an estimate for $\boldsymbol{y}^{(k)}$:

$$\begin{aligned} h_{jk}(t) &= E[s_{jk}|\hat{\boldsymbol{y}}_j, \hat{\boldsymbol{y}}^{(k)}(t), \hat{\boldsymbol{\Theta}}^*] = p(s_{jk}|\hat{\boldsymbol{y}}_j, \hat{\boldsymbol{y}}^{(k)}(t), \hat{\boldsymbol{\Theta}}^*) \\ &= \frac{p(\hat{\boldsymbol{y}}_j|s_{jk}, \hat{\boldsymbol{y}}^{(k)}(t), \hat{\boldsymbol{\Theta}}^*)}{\sum_l p(\hat{\boldsymbol{y}}_j|s_{jl}, \hat{\boldsymbol{y}}^{(l)}(t), \hat{\boldsymbol{\Theta}}^*)} \,, \qquad \text{(E-Step)} \qquad (3.23a) \\ \hat{\boldsymbol{y}}^{(k)}(t+1) &= \Big(\sum_j h_{jk}(t) \boldsymbol{R}_j^{-1} \Big)^{-1} \Big(\sum_j h_{jk}(t) \boldsymbol{R}_j^{-1} \hat{\boldsymbol{y}}_j \Big) \,. \qquad \text{(M-Step)} \qquad (3.23b) \end{aligned}$$

It usually takes only a few iterations for the EM algorithm to converge, using the initialization procedure described in the next section. After convergence, experts are assigned to multi-valued predictions $\hat{\boldsymbol{y}}^{(k)}$ according to the final value of h_{jk}; this leads to the following estimates for each multi-valued branch k,

$$\hat{\boldsymbol{y}}^{(k)} = \Big(\sum_j \boldsymbol{R}_j^{-1} \Big)^{-1} \Big(\sum_j \boldsymbol{R}_j^{-1} \hat{\boldsymbol{y}}_j \Big) \qquad \text{and} \qquad (3.24a)$$

$$\hat{\boldsymbol{R}}^{(k)} = \Big(\sum_j \boldsymbol{R}_j^{-1} \Big)^{-1} \,. \qquad (3.24b)$$

where the sums are over experts assigned to each particular multi-valued prediction $\hat{\boldsymbol{y}}^{(k)}$.

Additionally, the relative weight of each multi-valued prediction is given by

$$w^{(k)} = \sum_j w_j^y \ , \tag{3.24c}$$

where w_j^y is given by (3.15b) and the sum, as usual, is performed over the experts constituting prediction k.

3.3.3 Generative Model Statistical Validation

The above generative model, while providing a way to estimate the mean and the variance of a set of multi-valued output predictions $\boldsymbol{y}^{(k)}$ at a specific input query $\boldsymbol{x}_q$, cannot decide on the most adequate number of multi-valued solutions K to consider. Intuitively, a specific number of multi-valued solutions K for an input query $\boldsymbol{x}_q$ is acceptable if the experts individual predictions $\hat{\boldsymbol{y}}_j$ assigned to the same particular solution $\boldsymbol{y}^{(k)}$ agree, in a broad sense, with the value of that solution. This means that the deviation of experts predictions $\hat{\boldsymbol{y}}_j$ from the corresponding solution $\boldsymbol{y}^{(k)}$ should not be too large when compared to $\boldsymbol{R}_j$. Of course, as the value of K increases so does the fitness of the individual predictions to the solutions $\boldsymbol{y}^{(k)}$: a minimum value of K should be chosen that provides an adequate agreement of experts predictions $\hat{\boldsymbol{y}}_j$ with the generative model presented in the previous section.

For each $\hat{\boldsymbol{y}}_j$, the quantity $(\hat{\boldsymbol{y}}_j - \boldsymbol{y}^{(k)})^T \boldsymbol{R}_j^{-1} (\hat{\boldsymbol{y}}_j - \boldsymbol{y}^{(k)})$ follows a chi-squared distribution with D degrees of freedom, given that $\hat{\boldsymbol{y}}_j$ was indeed generated by solution k according to (3.22). Under the hypothesis that

(a) the probabilistic model presented in the previous section is indeed responsible for generating experts predictions $\hat{\boldsymbol{y}}_j$;

(b) the value K is the correct number of multi-valued solutions for the forward prediction problem at input query $\boldsymbol{x}_q$; and

(c) the EM procedure described above correctly grouped the experts predictions into K different multi-valued solutions,

then the statistic T_k follows a chi-squared distribution for every solution k,

$$T_k = \sum_j (\hat{\boldsymbol{y}}_j - \hat{\boldsymbol{y}}^{(k)})^T \boldsymbol{R}_j^{-1} (\hat{\boldsymbol{y}}_j - \hat{\boldsymbol{y}}^{(k)}) \quad \sim \chi^2_{(M_k - 1)D} \ , \tag{3.25}$$

where again the sums are over experts assigned to solution k and M_k is the number of such experts. A low value for this statistic indicates a good fit of observations $\hat{\boldsymbol{y}}_j$ to the estimated solutions $\hat{\boldsymbol{y}}^{(k)}$; on the other hand, the current set of solutions $\hat{\boldsymbol{y}}^{(k)}$ is considered to be badly explained by the data if the p-value for any solution k is lower than a given significance level α_{multi}: in such case the above hypothesis is rejected.

A practical detail of the above test is that $\hat{\boldsymbol{y}}_j$ will not effectively have an arbitrary large variance for corresponding small values of w_j^y, as expected from (3.19) and (3.22), where $\boldsymbol{R}_j$ goes to infinity as w_j^y approaches zero: as a consequence, T_k will assume much lower values than the ones expected under the null hypothesis distribution, since in a typical prediction scenario most of the experts will have very low values of w_j^y, making their contribution to the prediction at a particular query point negligible. This will make the rejection of the goodness of fit hypothesis stated above much harder to be met; even more problematic is the fact that the number of experts that do not significantly contribute to the prediction, with $w_j^y \approx 0$, and that were nevertheless assigned to a prediction $\hat{\boldsymbol{y}}^{(k)}$, will severely influence this statistical test, by "artificially" increasing the degrees of freedom of the associated chi-squared distribution. The solution to this problem is to replace $M_k - 1$ in (3.25) by $1/\sum(w_j^y)^2 - 1$, the effective degrees of freedom of the mixture; it was empirically found that this adjustment provides a better fit of the statistic to its corresponding distribution, under the null hypothesis.

3.3.4 Multi-valued Prediction

The previous sections provided the mechanisms that, for a particular value of K, generate a set of multi-valued predictions $\hat{\boldsymbol{y}}^{(k)}$, while coming up with a statistical validation tool for the results thus obtained. The question that remains to be answered is how to find an appropriate number of multi-valued solutions for the conditional prediction problem. Following the parsimonious principle known as Ockham's razor, the multi-valued prediction algorithm starts with a single-valued estimation, *i.e.*, $K = 1$: if the statistical test described in the previous section rejects this single-valued solution, K is increased by one and the EM iterations in (3.23) are performed. If the statistical test described in the previous section rejects at least one of the two solutions, K is again incremented: this procedure is repeated until a value of K is found for which the test fails to reject the null hypothesis for any of the solutions thus obtained. When this happens a set of multi-valued solutions is considered to be found, represented by posterior means $\hat{\boldsymbol{y}}^{(k)}$ and variances $\hat{\boldsymbol{R}}^{(k)}$, for $1 \leq k \leq K$, as given in (3.24).

To speed up the prediction process, each EM procedure initializes $\hat{\boldsymbol{y}}^{(k)}$ with the values found in the previous run of the algorithm, while the extra solution starts near the solution k that produced the smallest p-value in the previous statistical test, this way dividing in two the solution responsible for the null hypothesis rejection. This initialization greatly accelerates and stabilizes the convergence of the EM iterations for each value of K. Note also that, during this process, the significance level α_{multi} controls the final number of solutions found: the lower its value the harder it is to reject the null hypothesis, and less solutions are likely to be found. On the other hand, increasing α_{multi} helps to separate different solutions but, as an unwanted consequence, predictions $\hat{\boldsymbol{y}}_j$ for neighbour experts may stop being merged together due to the function curvature around $\boldsymbol{x}_q$.

Of course, if the input-output relation is known to be single-valued, there is no need to conduct this prediction procedure: it suffices then to use equations (3.21), using all the active experts in the mixture.

3.3.5 Inverse Prediction

The conditional inverse probability distribution for an output query $\boldsymbol{y}_q$ is given by

$$p(\boldsymbol{x}|\boldsymbol{y}_q, \hat{\boldsymbol{\Theta}}) = \sum_{j \in \mathcal{M}} w_j^x(\boldsymbol{y}_q) p(\boldsymbol{x}|\boldsymbol{y}_q, w_j, \hat{\boldsymbol{\Theta}}) \;, \tag{3.26}$$

where

$$w_j^x(\boldsymbol{y}_q) = p(w_j|\boldsymbol{y}_q, \hat{\boldsymbol{\Theta}}) = \frac{p(\boldsymbol{y}_q|w_j, \hat{\boldsymbol{\Theta}})}{\sum_{k \in \mathcal{M}} p(\boldsymbol{y}_q|w_k, \hat{\boldsymbol{\Theta}})} \;. \tag{3.27}$$

These expressions are the inverse prediction counterparts of (3.14a) and (3.14b); the probability densities $p(\boldsymbol{x}|\boldsymbol{y}_q, w_j, \hat{\boldsymbol{\Theta}})$ and $p(\boldsymbol{y}_q|w_j, \hat{\boldsymbol{\Theta}})$ appearing above can be obtained from the joint input-output distribution $p(\boldsymbol{v}|w_j, \hat{\boldsymbol{\Theta}})$, where $\boldsymbol{v}^T = [\boldsymbol{x}^T \; \boldsymbol{y}^T]$ is a vector where the input $\boldsymbol{x}$ and output $\boldsymbol{y}$ are stacked together. This joint distribution is Normal, with mean $\bar{\boldsymbol{v}}_j^T = [\boldsymbol{\nu}_j^T \; \boldsymbol{\mu}_j^T]$ and covariance matrix

$$\boldsymbol{R}_j^v = \begin{bmatrix} \boldsymbol{\Sigma}_j & \boldsymbol{\Sigma}_j \boldsymbol{\Lambda}_j^T \\ \boldsymbol{\Lambda}_j \boldsymbol{\Sigma}_j & \boldsymbol{\Psi}_j + \boldsymbol{\Lambda}_j \boldsymbol{\Sigma}_j \boldsymbol{\Lambda}_j^T \end{bmatrix} \;;$$

from this result it immediately follows that

$$\boldsymbol{x}|\boldsymbol{y}_q, w_j; \hat{\boldsymbol{\Theta}} \sim \mathcal{N}(\hat{\boldsymbol{x}}_j(\boldsymbol{y}_q), \boldsymbol{R}_j^x) \qquad \text{and} \tag{3.28a}$$

$$\boldsymbol{y}_q|w_j; \hat{\boldsymbol{\Theta}} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_j, \hat{\boldsymbol{\Psi}}_j + \hat{\boldsymbol{\Lambda}}_j \hat{\boldsymbol{\Sigma}}_j \hat{\boldsymbol{\Lambda}}_j^T) \;, \tag{3.28b}$$

where $\hat{\boldsymbol{x}}_j(\boldsymbol{y}_q)$ and $\boldsymbol{R}_j^x$ are given respectively by

$$\hat{\boldsymbol{x}}_j(\boldsymbol{y}_q) = \hat{\boldsymbol{\nu}}_j + \boldsymbol{R}_j^x \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1} (\boldsymbol{y}_q - \hat{\boldsymbol{\mu}}_j) \quad \text{and} \tag{3.28c}$$

$$\boldsymbol{R}_j^x = (\hat{\boldsymbol{\Sigma}}_j^{-1} + \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1} \hat{\boldsymbol{\Lambda}}_j)^{-1} \;. \tag{3.28d}$$

The above conditional distributions can be used to produce a set of inverse conditional point estimates, together with corresponding uncertainties, using a procedure similar to the one described in the previous section for forward prediction. Note that, contrary to forward prediction, it is not possible to incorporate the mixture parameters uncertainty in the inverse model, since there is no closed form expression for the result of compounding $p(\boldsymbol{x}|\boldsymbol{y}_q, w_j, \hat{\boldsymbol{\Theta}})$ or $p(\boldsymbol{y}_q|w_j, \hat{\boldsymbol{\Theta}})$, given respectively by (3.28a) and (3.28b), with the posterior distributions of the mixture parameters, given the training data observed so far. As a consequence, the inverse predictions are calculated using only the point estimates for mixture parameters $\hat{\boldsymbol{\Theta}}$. The generative model for inverse prediction, depicted in Figure 3.7,

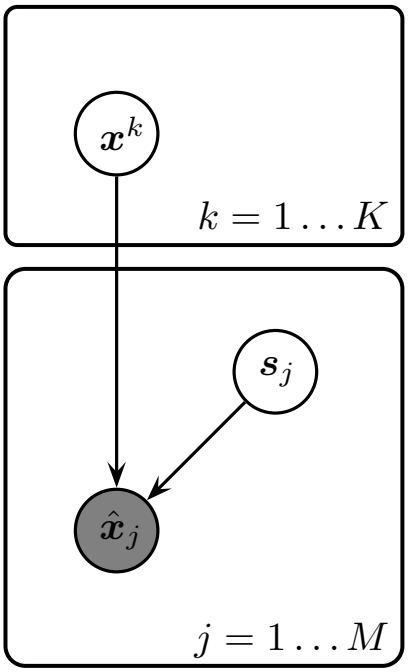


Figure 3.7: Inverse prediction graphical model.

thus become a simplified version of the forward prediction model of Figure 3.6, with

$$\hat{\boldsymbol{x}}_j | s_{jk}, \boldsymbol{x}^{(k)}, \hat{\boldsymbol{\Theta}} \sim \mathcal{N}(\boldsymbol{x}^{(k)}, \boldsymbol{R}_j) \ , \tag{3.29}$$

where now $\boldsymbol{R}_j$ is given by

$$\boldsymbol{R}_j \equiv \frac{1}{w_j^x} \boldsymbol{R}_j^x \ .$$

With this generative model in mind, the remaining procedure for obtaining a set of inverse multi-valued predictions for a query $\boldsymbol{y}_q$ is exactly the same as the one presented in the previous section, and will be omitted here for brevity. However, besides the lack of integration of uncertainty in the mixture parameters in the final prediction, note that, contrary to forward prediction, inverse prediction produces full covariance matrices for the uncertainty on the calculated point estimates for the conditional inverse solutions.

Of special importance is the fact that, in general, the input space dimension d may be greater than the output space dimension D: this means that, in this case, the true solution space $\boldsymbol{x}(\boldsymbol{y}_q)$ is continuous and has dimension $d - D$. In sensorimotor learning of a robot kinematic relation, this situation happens if the robot is redundant, with more degrees of freedom than the dimension of the task space considered. Therefore, some care must be used when looking at the set of inverse predictions provided by the multi-valued prediction algorithm, as it can only be interpreted as a set of *estimated samples* of the true, unknown continuous solution space $\boldsymbol{x}(\boldsymbol{y}_q)$.

Finally, it is worth of notice that the same reasoning for obtaining an inverse prob-

ability distribution from the IMLE model can also be used to produce more general predictions. Assuming the joint vector $\boldsymbol{v}$ to comprise an arbitrary query part $\boldsymbol{v}_q$ and an answer part $\boldsymbol{v}_a$, such that $\boldsymbol{v} = \boldsymbol{v}_q \cup \boldsymbol{v}_a$, the conditional distribution $p(\boldsymbol{v}_a|\boldsymbol{v}_q, w_j, \hat{\boldsymbol{\Theta}})$ can be straightforwardly derived from the joint distribution, for each expert j, and then the multi-valued prediction procedure previously presented in this chapter can be applied. However, it should be stressed out that, as in the inverse prediction situation, a continuous solution space is expected when the dimension of the query vector $\boldsymbol{v}_q$ is lesser than the input space dimension d, and that in that case the set of multi-valued solutions eventually obtained can only be viewed as a noisy sampling of the continuous solution space.

3.3.6 Jacobian Prediction

The forward prediction provided by IMLE at a given input query $\boldsymbol{x}_q$ is given, for a particular solution k found by the clustering procedure presented in Section 3.3.2, by

$$\hat{\boldsymbol{y}}^{(k)}(\boldsymbol{x}_q) = \hat{\boldsymbol{R}}^{(k)}(\boldsymbol{x}_q) \sum_j \boldsymbol{R}_j^{-1}(\boldsymbol{x}_q)\hat{\boldsymbol{y}}_j(\boldsymbol{x}_q) \ ,$$

where $\hat{\boldsymbol{R}}^{(k)}(\boldsymbol{x}_q)$ is the output expected variance, according to the current IMLE model, for solution k, evaluated at input point $\boldsymbol{x}_q$, given by

$$\hat{\boldsymbol{R}}^{(k)}(\boldsymbol{x}_q) = \Big(\sum_j \boldsymbol{R}_j^{-1}(\boldsymbol{x}_q) \Big)^{-1} \ ,$$

and where the sum is performed over the set of experts assigned to solution k. Quantities $\boldsymbol{R}_j(\boldsymbol{x}_q)$ and $\hat{\boldsymbol{y}}_j(\boldsymbol{x}_q)$ are respectively expert j output variance and prediction at point $\boldsymbol{x}_q$, defined in equations (3.19) and (A.3), and here repeated for convenience:

$$\boldsymbol{R}_j(\boldsymbol{x}_q) = \varphi_j(\boldsymbol{x}_q)\hat{\boldsymbol{\Psi}}_j \qquad \text{and} \qquad \hat{\boldsymbol{y}}_j(\boldsymbol{x}_q) = \hat{\boldsymbol{\Lambda}}_j(\boldsymbol{x}_q - \hat{\boldsymbol{\nu}}_j) + \hat{\boldsymbol{\mu}}_j \ .$$

This prediction can be expressed as a weighted average of the individual expert predictions,

$$\hat{\boldsymbol{y}} = \sum_j \boldsymbol{W}_j \hat{\boldsymbol{y}}_j \ , \qquad \text{with} \quad \boldsymbol{W}_j = \Big(\sum_k \boldsymbol{R}_k^{-1} \Big)^{-1} \boldsymbol{R}_j^{-1} = \hat{\boldsymbol{R}}\boldsymbol{R}_j^{-1} \ ,$$

where the explicit dependence on the input point $\boldsymbol{x}_q$ and on a particular solution k was dropped for readability.

The derivative of the IMLE prediction $\hat{\boldsymbol{y}}$ with respect to the input variable $\boldsymbol{x}$ can be interpreted as the estimated Jacobian of the true input-output relation, a D by d matrix where entry (m, n) gives the partial derivative of the mth output component with respect to the nth input dimension. This matrix can be obtained, for each multi-valued solution k, by taking the derivative of $\hat{\boldsymbol{y}}$ with respect to the input vector $\boldsymbol{x}$, using the above

expression. This results in

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} = \sum_j \boldsymbol{W}_j \hat{\boldsymbol{\Lambda}}_j + \sum_j \boldsymbol{W}_j (\hat{\boldsymbol{y}} - \hat{\boldsymbol{y}}_j) \boldsymbol{\zeta}_j(\boldsymbol{x}) \;, \tag{3.30}$$

where $\boldsymbol{\zeta}_j(\boldsymbol{x})$ is defined in equation (B.5a), in Appendix B; all the intermediate steps taken to deduce equation (3.30) are also therein presented. As a side effect of these calculations, the derivative of uncertainty $\hat{\boldsymbol{R}}$ with respect to the input vector is also calculated and given by

$$\frac{d\hat{\boldsymbol{R}}}{d\boldsymbol{x}} = \hat{\boldsymbol{R}} \sum_j \boldsymbol{W}_j \boldsymbol{\zeta}_j(\boldsymbol{x}) \;. \tag{3.31}$$

Note that since $\hat{\boldsymbol{R}}$ is a D by D matrix, its derivative with respect to the input vector $\boldsymbol{x}$ results in a third order tensor; however, as $\hat{\boldsymbol{R}}$ is a diagonal matrix, there is a slight abuse of notation in equation (3.31) above: entry (m, n) of the matrix in the right side of the equation denotes the partial derivative of the mth element of the diagonal of $\hat{\boldsymbol{R}}$ with respect to the nth input dimension.

3.3.7 Computational Complexity

Forward prediction requires the quantities $\hat{\boldsymbol{y}}_j$, γ_j, w_j^y and $\hat{\boldsymbol{\Psi}}_j^{-1}$ to be available for every active expert j: calculating these values has a cost of $\mathcal{O}(M(d^2 + dD))$, the same computational complexity of an update of the IMLE model for a new training point. After that, equations (3.21a) and (3.21b) are used to obtain a single-valued prediction, with a cost of $\mathcal{O}(MD)$: this is usually much faster than the previous step of obtaining the aforementioned quantities for each expert. Multi-valued prediction multiplies this cost by N_{iter}, the number of EM iterations used in (3.23), and by K, the final number of multi-valued solutions obtained, therefore having a total computational cost of $\mathcal{O}(MDN_{iter}K)$.

The above reasoning holds for inverse prediction, exception made to the inversion of full covariance matrices required to obtain $\hat{\boldsymbol{x}}_j$, in (3.28c), and the weights w_j^x via distributions $p(w_j|\boldsymbol{y}_q, \hat{\boldsymbol{\Theta}})$, given by (3.28b), which have a higher computational cost. A full matrix inversion may sometimes present numerical problems and lack of accuracy, but fortunately this inversion is not required to obtain $\hat{\boldsymbol{x}}_j$ and w_j^x: the quantity $\boldsymbol{U}_j = \boldsymbol{R}_j^x \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1} = (\hat{\boldsymbol{\Sigma}}_j^{-1} + \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1} \hat{\boldsymbol{\Lambda}}_j)^{-1} \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1}$ needed to compute $\hat{\boldsymbol{x}}_j$ may be obtained by solving

$$\left(\hat{\boldsymbol{\Sigma}}_j^{-1} + \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1} \hat{\boldsymbol{\Lambda}}_j\right) \boldsymbol{U}_j = \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1}$$

using a robust Cholesky decomposition of $\hat{\boldsymbol{\Sigma}}_j^{-1} + \hat{\boldsymbol{\Lambda}}_j^T \hat{\boldsymbol{\Psi}}_j^{-1} \hat{\boldsymbol{\Lambda}}_j$, which is faster and more accurate than performing the full matrix inversion. After that, an expert j inverse prediction is given by

$$\hat{\boldsymbol{x}}_j(\boldsymbol{y}_q) = \hat{\boldsymbol{\nu}}_j + \boldsymbol{U}_j (\boldsymbol{y}_q - \hat{\boldsymbol{\mu}}_j) \;.$$

This result may also be used to avoid the matrix inversion required to obtain $p(w_j|\boldsymbol{y}_q, \hat{\boldsymbol{\Theta}})$ in (3.28b): using the Woodbury matrix identity,

$$\begin{aligned}\left(\hat{\boldsymbol{\Psi}}_j + \hat{\boldsymbol{\Lambda}}_j\hat{\boldsymbol{\Sigma}}_j\hat{\boldsymbol{\Lambda}}_j^T\right)^{-1} &= \hat{\boldsymbol{\Psi}}_j^{-1} - \hat{\boldsymbol{\Psi}}_j^{-1}\hat{\boldsymbol{\Lambda}}_j\left(\hat{\boldsymbol{\Sigma}}_j^{-1} + \hat{\boldsymbol{\Lambda}}_j^T\hat{\boldsymbol{\Psi}}_j^{-1}\hat{\boldsymbol{\Lambda}}_j\right)^{-1}\hat{\boldsymbol{\Lambda}}_j^T\hat{\boldsymbol{\Psi}}_j^{-1} \\ &= \hat{\boldsymbol{\Psi}}_j^{-1} - \hat{\boldsymbol{\Psi}}_j^{-1}\hat{\boldsymbol{\Lambda}}_j\boldsymbol{U}_j \ .\end{aligned}$$

The above computations to obtain $\boldsymbol{U}_j$ and $(\hat{\boldsymbol{\Psi}}_j + \hat{\boldsymbol{\Lambda}}_j\hat{\boldsymbol{\Sigma}}_j\hat{\boldsymbol{\Lambda}}_j^T)^{-1}$ do not depend on a specific query point $\boldsymbol{y}_q$, and thus can be reused, at a null computational cost, for inverse predictions at different queries $\boldsymbol{y}_q$, as long as the mixture parameters do not change. Furthermore, if computation time is a serious issue, such computations may be performed, for each expert, only after relevant changes to its parameters are made; these changes can be detected by monitoring the number of points used to train each expert, given by S_{hj} in (3.6).

3.4 Active Uncertainty Reduction

Sensorimotor learning, most of the times, does not consist in a pure supervised learning scheme: autonomous agents have the ability to choose the input training points used during the learning phase, as opposed to supervised learning, where input-output data pairs are externally provided to the learner. In this situation, under an online setting, a learning mechanism may actively and iteratively choose which input location of the sensorimotor map being learned should be sampled to be presented to the learning algorithm in a subsequent iteration, usually having the reduction of its prediction error in mind. In a broad sense, *active learning* "studies the closed-loop phenomenon of a learner selecting actions or making queries that influence what data are added to its training set", to reduce the data a learner requires to achieve a given performance (Cohn, Ghahramani, and Jordan, 1996).

Active learning — also coined optimal experimental design in a regression setting — is an active field of research, and there are many different strategies to optimally select the next training point to "harvest": for a good review of active learning techniques, the survey of Settles (2009) is suggested. Among these methods, the prediction uncertainty reduction principle presented in (Cohn, Ghahramani, and Jordan, 1996) is of particular relevance for use with the IMLE model, as its probabilistic reasoning adequately fits the prediction mechanisms presented earlier in this chapter. This active learning approach starts by acknowledging that the expected prediction error of a learning algorithm, given by

$$\int \mathbb{E}_{y,\mathcal{D}}\left[(\hat{y}(\boldsymbol{x}) - y(\boldsymbol{x}))^2\right] p(\boldsymbol{x})d\boldsymbol{x} \ , \tag{3.32}$$

depends only on the training data through $\mathbb{V}[\hat{y}(\boldsymbol{x})]$, the variance of the predictor $\hat{y}(\boldsymbol{x})$,

assuming the squared prediction bias of this predictor to be negligible when compared to its variance. In the above equation a scalar output y is considered, and the expectation is taken over $p(y|\boldsymbol{x})$ and training sets $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{Y})$ used to train the learning algorithm responsible for prediction $\hat{y}(\boldsymbol{x})$. This result directly follows from the decomposition of the prediction error provided by Geman, Bienenstock, and Doursat (1992):

$$\begin{aligned}\mathbb{E}_{y,\mathcal{D}}\left[(\hat{y}(\boldsymbol{x}) - y(\boldsymbol{x}))^2\right] =& \mathbb{E}_y\left[(y(\boldsymbol{x}) - \mathbb{E}_y[y|\boldsymbol{x}])^2\right] + \left(\mathbb{E}_{\mathcal{D}}[\hat{y}|\boldsymbol{x}] - \mathbb{E}_y[y|\boldsymbol{x}]\right)^2 + \\ &+ \mathbb{E}_{\mathcal{D}}\left[(\hat{y}(\boldsymbol{x}) - \mathbb{E}_{\mathcal{D}}[\hat{y}(\boldsymbol{x})])^2\right] \ ;\end{aligned}$$

in this expression the first term in the right side is the true variance of the data, the second term is the squared bias of the predictor and the third one its variance, $\mathbb{V}[\hat{y}(\boldsymbol{x})]$. Note that the first two terms cannot be computed, as the true distribution $p(y|\boldsymbol{x})$ is not known.

Choosing a new point $\tilde{\boldsymbol{x}}$ to query for the corresponding response $\tilde{y}$, that is expected to lower the prediction error the most, amounts consequently to picking the value of $\tilde{\boldsymbol{x}}$ that maximizes the average predictor variance decrease over the input space,

$$\arg\max_{\tilde{\boldsymbol{x}}} \int \left(\mathbb{V}[\hat{y}(\boldsymbol{x})] - \tilde{\mathbb{V}}[\hat{y}(\boldsymbol{x})]\right) p(\boldsymbol{x}) \ d\boldsymbol{x} \ ,$$

where $\tilde{\mathbb{V}}[\hat{y}(\boldsymbol{x})]$ is the expected predictor variance at input location $\boldsymbol{x}$ after data point $(\tilde{\boldsymbol{x}}, \tilde{y})$ has been incorporated in the training set. The above expression can be easily extended to a multivariate case, by considering instead

$$\arg\max_{\tilde{\boldsymbol{x}}} \int \boldsymbol{a}^T \cdot \left(\mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})] - \tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\boldsymbol{x})]\right) p(\boldsymbol{x}) \ d\boldsymbol{x} \ , \tag{3.33}$$

where $\boldsymbol{a}$ is a D-dimensional vector of inverse weights that assigns more or less importance to each output component in this active learning setting. Again, this is only valid assuming the variance of the estimator to dominate its squared bias, *i.e.*, that the learner is approximately *unbiased*.

Since $\mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})]$ does not depend on the new training point $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$, the point $\tilde{\boldsymbol{x}}$ that maximizes the above quantity is equal to the one that minimizes $\boldsymbol{a}^T\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\boldsymbol{x})]$, *i.e.*,

$$\arg\min_{\tilde{\boldsymbol{x}}} \int \boldsymbol{a}^T \cdot \tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\boldsymbol{x})] \ p(\boldsymbol{x}) \ d\boldsymbol{x} \ ;$$

for a scalar output, this is the original formulation in (Cohn, Ghahramani, and Jordan, 1996).

From a practical viewpoint, obtaining the integral in (3.33) is analytically intractable; one way to circumvent this problem is to evaluate $\mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})] - \tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\boldsymbol{x})]$ only at some random points $\boldsymbol{x}$ drawn according to $p(\boldsymbol{x})$, as suggested in the original paper of Cohn, Ghahra-

mani, and Jordan (1996). However, as the number of input dimensions d increases there is an exponential increase in the number of such points required to produce a good approximation of the above integral. This goes against the scalability properties that are desired for the IMLE learning algorithm. Another approach, followed in this work, is to assume that after querying $\tilde{\boldsymbol{x}}$ the prediction variation $\mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})]$ will decrease the most in a region around $\tilde{\boldsymbol{x}}$; as a consequence, the point $\tilde{\boldsymbol{x}}$ that produces the largest decrease of prediction variation, evaluated at the same point $\tilde{\boldsymbol{x}}$, as given by

$$\arg\max_{\tilde{\boldsymbol{x}}} \ \Delta\mathbb{V}(\tilde{\boldsymbol{x}}) \ ,$$

where scalar $\Delta\mathbb{V}(\tilde{\boldsymbol{x}})$ is defined as

$$\Delta\mathbb{V}(\tilde{\boldsymbol{x}}) = \boldsymbol{a}^T \cdot \Big(\mathbb{V}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})] - \tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})] \Big) \ , \tag{3.34}$$

can be considered a reasonable, if somewhat crude, approximation to the value given by (3.33). This approximation can be justified by the local nature of the IMLE algorithm: adding $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ to the training set will mostly affect experts that are sufficiently activated during the E-Step of the training procedure, having non-negligible values of h_{ij}. As a consequence, prediction will change the most in the input region influenced by these experts, containing the point $\tilde{\boldsymbol{x}}$.

The variance of the estimator $\hat{\boldsymbol{y}}(\boldsymbol{x})$ is

$$\begin{aligned}
\mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})] &= \mathbb{V}\Big[\sum_j \boldsymbol{W}_j \hat{\boldsymbol{y}}_j \Big] \\
&= \sum_j \mathbb{V}\Big[\boldsymbol{W}_j \hat{\boldsymbol{y}}_j \Big] \\
&= \sum_j \boldsymbol{W}_j^2 \mathbb{V}[\hat{\boldsymbol{y}}_j] \\
&= \sum_j \boldsymbol{W}_j^2 \gamma_j \hat{\boldsymbol{\Psi}}_j \\
&= \hat{\boldsymbol{R}}^2 \sum_j \boldsymbol{R}_j^{-2} \gamma_j \hat{\boldsymbol{\Psi}}_j \ ,
\end{aligned}$$

as the variability of the estimator only comes from the training data output noise — the input values $\boldsymbol{X}$ are assumed fixed and chosen by the learner — and the values of $\boldsymbol{W}_j$ only depend on these inputs; also, it is assumed that, given $\boldsymbol{X}$, the predictions $\hat{\boldsymbol{y}}_j(\boldsymbol{x})$ are independent. Note that $\hat{\boldsymbol{R}}$, $\boldsymbol{R}_j$ and γ_j all depend on $\boldsymbol{x}$, although that is omitted in the above derivation for readability. Finally, the above derivation for the variance of $\hat{\boldsymbol{y}}(\boldsymbol{x})$ takes into account the fact that all matrices involved are diagonal ones: in the general case, $\mathbb{V}\left[\boldsymbol{W}_j \hat{\boldsymbol{y}}_j\right]$ would be equal to $\boldsymbol{W}_j \mathbb{V}\left[\hat{\boldsymbol{y}}_j\right] \boldsymbol{W}_j^T$, of course.

As expected, the variance of the predictor is always less or equal than the estimated

data variance $\hat{\boldsymbol{R}}$ for each output dimension, as the latter also includes the predictor variance. This can be demonstrated by noting that

$$\boldsymbol{R}_j^{-1}\gamma_j\hat{\boldsymbol{\Psi}}_j = \left(\gamma_j + \frac{1}{w_j^y}\right)^{-1}\hat{\boldsymbol{\Psi}}_j^{-1}\gamma_j\hat{\boldsymbol{\Psi}}_j = \frac{\gamma_j w_j^y}{\gamma_j w_j^y + 1}$$

where without losing generality a scalar output is assumed. As a consequence,

$$\begin{aligned}
& \boldsymbol{R}_j^{-1}\gamma_j\hat{\boldsymbol{\Psi}}_j &<& \ 1 \\
\iff \quad & \boldsymbol{R}_j^{-2}\gamma_j\hat{\boldsymbol{\Psi}}_j &<& \ \boldsymbol{R}_j^{-1} \\
\iff \quad & \sum_j \boldsymbol{R}_j^{-2}\gamma_j\hat{\boldsymbol{\Psi}}_j &<& \ \sum_j \boldsymbol{R}_j^{-1} = \hat{\boldsymbol{R}}^{-1} \\
\iff \quad & \hat{\boldsymbol{R}}^2\sum_j \boldsymbol{R}_j^{-2}\gamma_j\hat{\boldsymbol{\Psi}}_j &<& \ \hat{\boldsymbol{R}} \\
\iff \quad & \mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})] &<& \ \hat{\mathbb{V}}[\boldsymbol{y}(\boldsymbol{x})]\ .
\end{aligned}$$

The expected predictor variance after presenting $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ to the learner, $\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})]$, is

$$\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\boldsymbol{x})] = \sum_j \tilde{\boldsymbol{W}}_j^2\mathbb{E}_{\tilde{\boldsymbol{y}}}[\hat{\boldsymbol{y}}_j] = \tilde{\boldsymbol{R}}^2\sum_j \tilde{\boldsymbol{R}}_j^{-2}\tilde{\gamma}_j\mathbb{E}_{\tilde{\boldsymbol{y}}}[\hat{\boldsymbol{\Psi}}_j]\ ,$$

where $\tilde{\boldsymbol{W}}_j$, $\tilde{\boldsymbol{R}}_j$ and $\tilde{\gamma}_j$ denote respectively the new values of $\boldsymbol{W}_j$, $\boldsymbol{R}_j$ and γ_j after $\tilde{\boldsymbol{x}}$ is incorporated in the mixture, and $\mathbb{E}_{\tilde{\boldsymbol{y}}}[\hat{\boldsymbol{y}}_j]$ and $\mathbb{E}_{\tilde{\boldsymbol{y}}}[\hat{\boldsymbol{\Psi}}_j]$ are the expected values for $\hat{\boldsymbol{y}}_j$ and $\hat{\boldsymbol{\Psi}}_j$, averaged over possible values of $\tilde{\boldsymbol{y}}$, also after $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ is added to the mixture — here, $\tilde{\boldsymbol{y}}$ is treated as a random variable as it is not yet observed.

Picking the value of $\tilde{\boldsymbol{x}}$ that maximizes (3.34) is not analytically feasible and it involves the search in a d-dimensional space: pool-based selective sampling, *i.e.*, randomly generating a set of $\tilde{\boldsymbol{x}}$ candidates according to the input distribution and choosing the best one according to (3.34), may be computationally efficient for very low dimensional problems, but unfortunately the number of such candidates required to cover the input space with the same resolution grows exponentially as the input dimension increases. An alternative to pool-based selective sampling, also suggested in (Cohn, Ghahramani, and Jordan, 1996), is to hill-climb the criterion (3.34) by making use of the analytical derivative of $\Delta\mathbb{V}(\tilde{\boldsymbol{x}})$,

$$\frac{d}{d\tilde{\boldsymbol{x}}}\Delta\mathbb{V}(\tilde{\boldsymbol{x}}) = \boldsymbol{a}^T \cdot \left(\frac{d}{d\tilde{\boldsymbol{x}}}\mathbb{V}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})] - \frac{d}{d\tilde{\boldsymbol{x}}}\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})]\right)\ , \tag{3.35}$$

which can provide a local maximum of $\Delta\mathbb{V}(\tilde{\boldsymbol{x}})$. Some gory details concerning this derivative are presented in Appendix C.

The hill-climbing procedure described above can produce a new training candidate $\tilde{\boldsymbol{x}}$ arbitrarily far from the last acquired point $\boldsymbol{x}_t$. On the other hand, most of the times real-time learning of sensorimotor maps introduces continuity contingencies, since it is

not possible to instantaneously change the robot input $\boldsymbol{x}$ by a large amount between acquisition samples, due to acceleration limitations. This suggests using a local, greedy approach to the active learning problem, where, at each learning iteration, a new training point $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ is sampled along the direction, provided by (3.35), that results in the largest expected prediction variance reduction for the current robot state: this direction can be found by simply evaluating the above derivative at current point $\boldsymbol{x}_t$.

Finally, a brief note pertaining multi-valued prediction should be added: all the above equation apply in a straightforward manner to the multi-valued case, as long as the summations needed to obtain $\mathbb{V}[\hat{\boldsymbol{y}}(\boldsymbol{x})]$ and $\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\boldsymbol{x})]$ are only performed over the sets of experts obtained through the clustering procedure presented in section 3.3.4. As a consequence, a set of K "optimal" sampling directions are obtained, one for each of the multi-valued predictions obtained for current input point $\boldsymbol{x}_t$. Which one should then be used to maximize the prediction uncertainty reduction? This is equivalent to asking which of the outputs, predicted by the IMLE algorithm, corresponds to the actual robot state, and while more sophisticated mechanisms may be employed to estimate such latent context (Petkos and Vijayakumar, 2007), a simple solution to this problem is to consider the multi-valued prediction closer to the last observed output $\boldsymbol{y}_t$, and then to use the corresponding set of experts to provide the input direction along which a new training point should be sampled. This approach, unfortunately, cannot be directly applied to a pool-based sampling scheme, as it is no longer trivial to find a correspondence between the multi-valued function branch for the last seen output $\boldsymbol{y}_t$ and the set of multi-valued predictions at a random input location: what to do in this case is beyond the scope of this thesis.

3.5 Discussion

This chapter presented the IMLE probabilistic model: it was shown how to train it in an incremental and online fashion, automatically dealing with a number of issues that are of great importance for sensorimotor learning. In particular, the EM based training procedure automatically handles outliers, enlarges the model whenever needed, assigning more experts to the current mixture, and continuously adapts the hyper-priors values concerning the characteristic input length-scale and output noise of the relation to be learned, shared by all experts in the mixture.

A prediction procedure, using the current of the IMLE model, was also introduced, capable of providing forward and inverse multi-valued predictions, as well as the estimated Jacobian at a given query point. Also, an active learning scheme was derived for this learning architecture, that provides a method to choose the next training input $\tilde{\boldsymbol{x}}$ that, in a statistical manner, is expected to produce the largest reduction on the prediction variance and, therefore, on the prediction error. Overall, these prediction mechanisms

and training procedure make the IMLE algorithm a highly versatile method for real-time sensorimotor learning. The next chapter will attempt to demonstrate the capabilities of this method, by intensively testing and comparing it to other state-of-the-art learning algorithms, in a diversity of different and demanding situations.

Chapter 4

Experimental Evaluation

This chapter is devoted to the evaluation of the IMLE model in several different experimental settings, specially focusing on large training sets arising from continuous streams of data that particularly suit online learning. This model is compared to other online learning algorithms, namely LWPR (Vijayakumar, D'Souza, and Schaal, 2005), probably one of the most widely used state-of-the-art online learning methods for robotic applications, SOGP, a sparse online approximation for Gaussian process regression (Csató and Opper, 2002), and ROGER, an online infinite mixture of SOGP experts (Grollman and Jenkins, 2010). All these algorithms have their C++ implementation code available, and their most recent version to this date is used in the comparisons (Klanke and Vijayakumar, 2009; Grollman, 2008; Grollman, 2009). A C++ implementation of the IMLE algorithm is also freely available for download and is described in Appendix D.

The IMLE learning model is also compared to standard GPR: this is not an online algorithm in its standard formulation, but it can give some insights on the expected performance loss when going to an online operation setting. In the following experiments, a Gaussian likelihood to be used with an isotropic squared exponential covariance function is specified, using exact inference for training and prediction. Optimal values for the input length-scale of the kernels and for the output noise, the free hyperparameters of the model, are obtained using standard optimization techniques over the training set, using the GPML Matlab code (Rasmussen and Nickisch, 2010).

Parameters of interest for tuning the LWPR algorithm comprehend $\boldsymbol{D}_{init}$, α_{init}, w_{gen} and penalty γ: for details on these parameters meaning, please consult the related documentation. Additionally, in all experiments *diagOnly* is set to *false* and $useMeta$ and *updateD* to *true*. The SOGP model is trained using a Gaussian kernel: the remaining tuning parameters for this algorithm are σ_k^2, the kernel width, σ_0^2, the expected output noise for the function to learn, and β, the maximum number of training basis points to be kept by the algorithm. SOGP resembles standard GPR if no upper limit is set to this number of basis points. Besides the (common) parameters for each of the SOGP experts, ROGER most important quantities to be defined beforehand are P, the number of par-

ticles for the mixture, and α, the Chinese Restaurant Process concentration parameter that drives the propensity of new SOGP experts to be created within each particle. In general, the higher the number of particles and SOGP capacity the slower ROGER will run.

As for IMLE, there are 11 parameters that can be tuned to change the resulting behaviour of the algorithm: some of them typically don't need any tweaking and the following experiences, unless otherwise noted, will keep them with their default values, namely $n_\Lambda = 0.1$ (a small value is needed for regularization), $n_\nu = 0$ and $n_\mu = 0$ (experts locations fully learned). As the input space dimension increases, a stronger prior on input covariance matrices $\mathbf{\Sigma}_j$ and output noise $\mathbf{\Psi}_j$ is needed to make the learning process relatively invariant with respect to the trajectory nature of the training data acquisition process: a good rule of thumb is to set $n_\Sigma = n_\Psi = n_\sigma = 2^d$ and then choose n_ψ based on the confidence on the value of $\mathbf{\Psi}_0$, with smaller values corresponding to a larger uncertainty on this parameter. A typical value for the forgetting factor lies in the range $\alpha = 0.99 \sim 0.999$; the remaining parameters, $\mathbf{\Psi}_0$, $\mathbf{\Sigma}_0$ and p_0 have a strong influence in the experts activation process (3.11), and ultimately on the number of local experts created during the training phase. $\mathbf{\Psi}_0$ represents the expected output noise variance, while $\mathbf{\Sigma}_0$ corresponds to the input activation region for which the function to be learned can be approximately represented by a linear relation.

While setting and tuning such apparently high number of free parameters may appear to be challenging at first, the convergence of the probabilistic model is not very sensitive to specific values of these parameters. Perhaps the most important issue when considering the tuning of IMLE free parameters is to ensure a correct convergence of $\bar{\mathbf{\Sigma}}$ and $\bar{\mathbf{\Psi}}$, the input length-scale and output noise estimates; this problem is discussed thoroughly in the following text.

4.1 Single-valued Function Approximation

This section evaluates the IMLE model performance with respect to single-valued function approximation, applying the algorithm to four different learning problems with increasing input dimension and comparing its performance to LWPR, SOGP and GPR. Some care must be taken when confronting these different learning schemes: in general, increasing the model complexity for each of the algorithms will produce smaller approximation errors, while incurring in some heavier computational cost. The number of local linear models activated by IMLE and LWPR is a good measure of model complexity for these online algorithms: its final value, after the training process, is a consequence of the choices for their tunable parameters and the training data itself. Since IMLE and LWPR have the same computational complexity per training point, the final number of activated models provides a fair comparison ground for IMLE and LWPR in terms of the approximation er-

ror/computational complexity trade-off. GPR and SOGP model complexity, on the other hand, is measured by the number of stored training points used for posterior prediction over the test data. For GPR this number is set beforehand, while SOGP learns a sparse subset of the training data to be used for prediction, possibly limiting the maximum number of these inducing points to a value of β. IMLE and LWPR computational demands are linear in the number of local models, while GPR and SOGP are much more penalized by the increase of the number of stored training points or inducing points, respectively.

The amount of information implicitly available to the algorithms is another important issue for a fair comparison between them: for regression, the input length-scale of the data and the noise level present in the output are two critical properties of the function to be learned. GPR learns them offline by optimizing the likelihood of the training data with respect to these hyperparameters. Since GPR is an offline algorithm, its prediction performance strongly depends on the stored training points input locations: if they efficiently cover all the input space GPR is expected to outperform methods based on local linear approximations, in terms of prediction error. Yet, such highly desirable informative training set may be unavailable or can be difficult to generate, as in typical robotic applications, where visiting all the input space can be very time consuming.

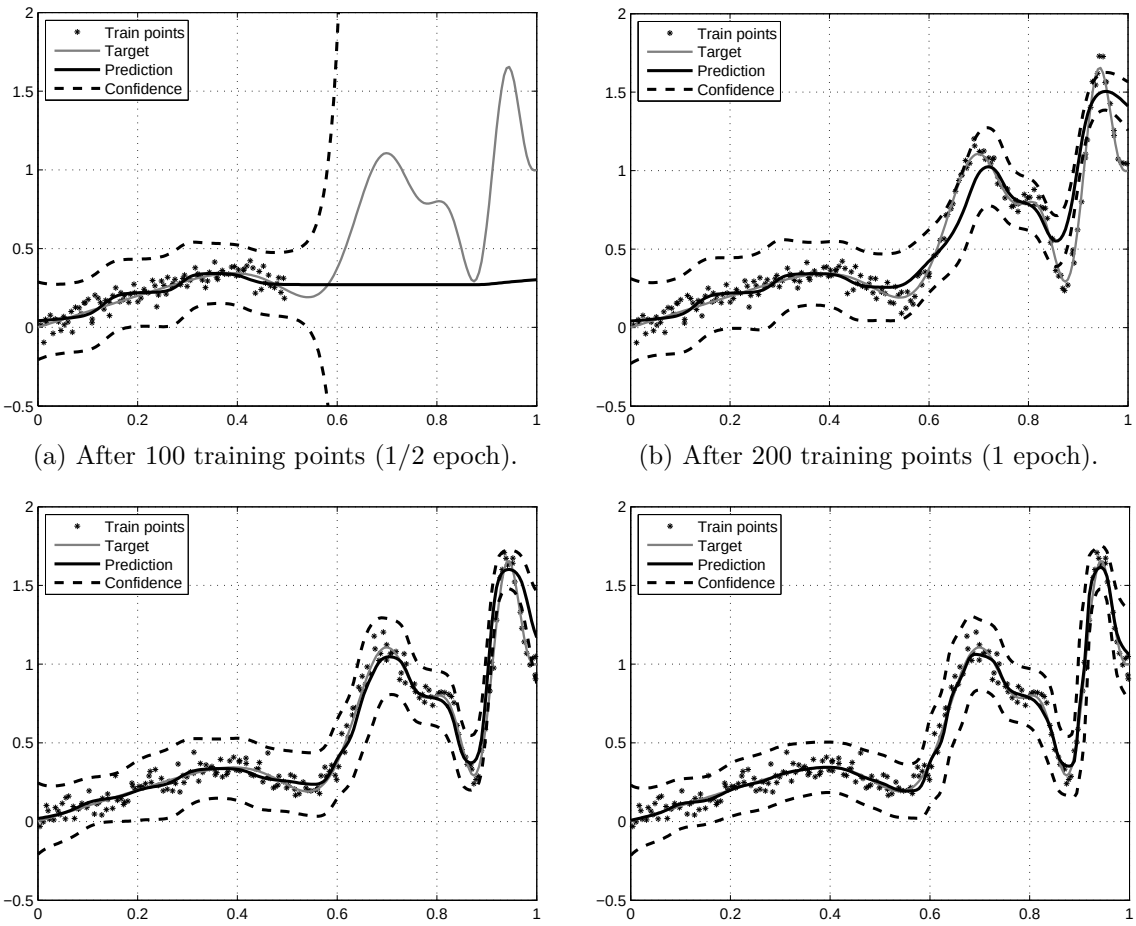
Online methods alleviate this dependence on an initial representative training set by learning their models on the fly, adapting them as new training data arrives. This is achieved in SOGP by maintaining a representative subsample of the data. Parameters σ_k^2 and σ_0^2, however, are not adapted during the learning process, and thus SOGP must rely on a good initialization of its input length-scale and output noise parameters: adequate values can be obtained, for instance, from an initial offline optimization, similar to GPR. Both IMLE and LWPR learn the input length-scale and output noise for each of their local linear models — in fact, they learn a full input distance metric, represented by covariance matrices $\boldsymbol{\Sigma}_j$ and $\boldsymbol{D}_j^{-1}$ respectively. They differ in the way they initialize these quantities: LWPR initializes the input distance metrics to a constant value $\boldsymbol{D}_{init}$; IMLE, on the other hand, puts a common prior on $\boldsymbol{\Sigma}_j$, defining then a vague hyperprior for $\bar{\boldsymbol{\Sigma}}$. While LWPR initialization strongly influences the number of receptive fields created during learning, in IMLE the information conveyed in hyperpriors parameters $\boldsymbol{\Sigma}_0$ and $\boldsymbol{\Psi}_0$ can quickly lose importance if n_σ and n_ψ are small. This capability to learn, in an online fashion, the characteristic input length-scale and output noise makes IMLE more robust to poor parameter initialization and less dependent of problem specific knowledge; this complex probabilistic model structure, however, can make the learning convergence depend more on the training data input distribution, since hyperparameters $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$ strongly influence the behaviour of newly activated experts — which in turn also contribute to the estimation of $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$. Ultimately, as discussed in Section 4.1.3, this can lead to convergence to poor local maxima of the likelihood function.

4.1.1 Toy Example, $\mathbb{R}^1 \mapsto \mathbb{R}^1$

The first function approximation problem on which the IMLE model is tested is taken from (Schaal and Atkeson, 1993) and (Vijayakumar, D'Souza, and Schaal, 2005), and consists in an univariate regression problem whose target function, defined between 0 and 1, is given by

$$y = x - \sin^3(2\pi x^3)\cos(2\pi x^3)\exp(x^4) \; .$$

Data is taken in a sequential manner from this relation, and output is corrupted with Gaussian noise with standard deviation equal to 0.05. The IMLE model parameters for this problem follow the general rules stated above, and were set to $\alpha = 0.999$, $n_\Lambda = 0.1$, $n_\nu = n_\mu = 0$ and $n_\Sigma = n_\Psi = n_\sigma = n_\psi = 2$. The remaining parameters, $\mathbf{\Psi}_0$, $\mathbf{\Sigma}_0$ and p_0, were set respectively to $0.01\mathbf{I}$, $0.01\mathbf{I}$ and 0.1. The IMLE single-valued prediction provided at different stages of the learning process is shown in Figure 4.1: as can be seen in



(a) After 100 training points (1/2 epoch).

(b) After 200 training points (1 epoch).

(c) After 1 000 training points (5 epochs).

(d) After 10 000 training points (50 epochs).

Figure 4.1: Single-valued prediction provided by the IMLE model for the univariate toy example. The prediction confidence shown in the figures represents a 3 standard deviation interval, as given by the variance estimate provided by IMLE at each test point.

Figure 4.1b, after a single sweep of the input range the IMLE model already provides a very decent approximation to the target function, and after 5 sweeps an accurate approximation can already be found in Figure 4.1c.

It is also instructive to watch what happens to the IMLE model prediction when training data is absent from a particular input range of the target function. Figure 4.2 depicts such situation, where now there is no training data from input range $[0.4, 0.6]$ presented to the learning algorithm. Note that the prediction uncertainty measure, as

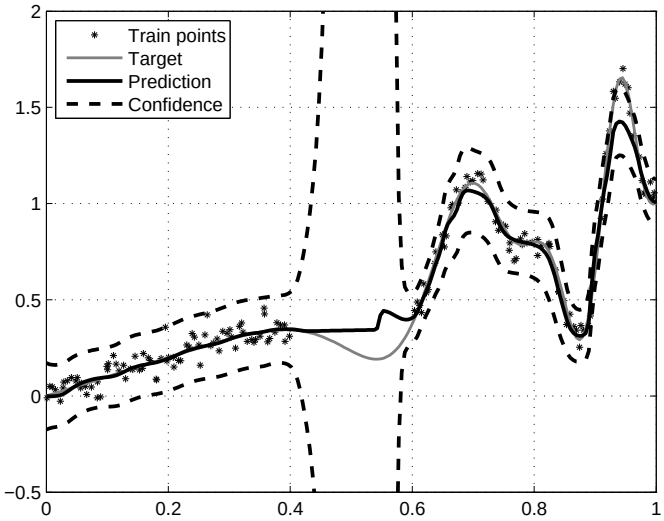


Figure 4.2: Single-valued prediction provided by the IMLE model for the univariate toy example, after 5 epochs of training, with data missing between 0.4 and 0.6.

given by the IMLE model, quickly grows as the test point moves away from input regions covered by the mixture linear experts. This is a consequence of equation (3.21b): as a test point $\boldsymbol{x}_q$ moves away from a linear expert j input activation region the corresponding value of w_j^y goes to 0, and correspondingly $\boldsymbol{R}_j$ goes to infinity, as given by equation (3.19). This is a desired property of the prediction process: since at $\boldsymbol{x}_q$ there are no linear experts supporting the input query location, the provided prediction should be accompanied by a large variance, signalling the fact that such prediction should be considered highly unreliable.

4.1.2 Cross Function, $\mathbb{R}^2 \mapsto \mathbb{R}^1$

The next experience runs the IMLE model and other competing algorithms on a sequential stream of data taken from the cross function suggested in (Vijayakumar, D'Souza, and Schaal, 2005), a two-dimensional input, univariate output function displayed in Figure 4.3, given by the relation

$$y = \max\{\exp(-10x_1),\ \exp(-50x_2),\ 1.25\exp(-5x_1^2 - 5x_2^2)\}\ ,$$

where x_1 and x_2 denote respectively the first and second input vector components. The

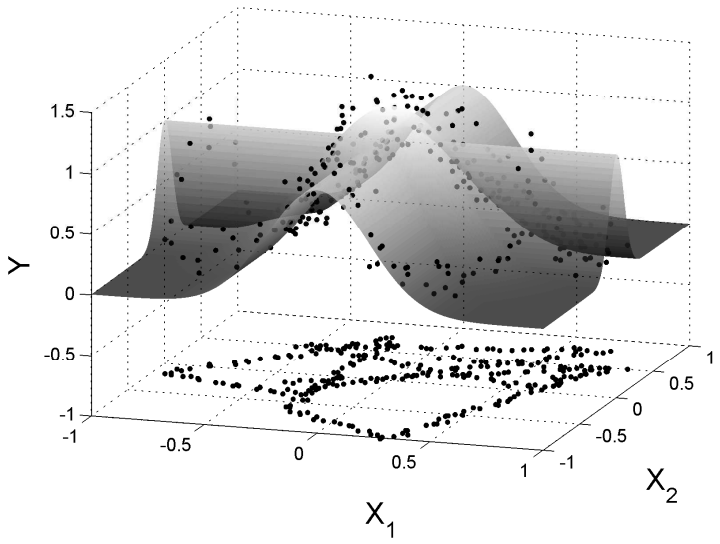


Figure 4.3: Target function and a sample of the training data (also projected on the x-plane to enhance the trajectory nature of data acquisition).

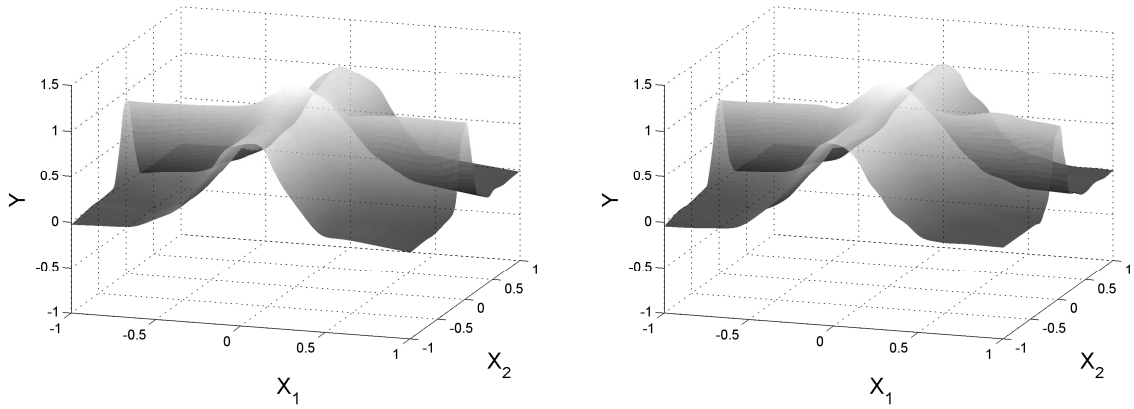
training set consists of points sampled from a random trajectory performed in the input space, together with corresponding output data, for which Gaussian noise with 0.1 standard deviation was added: a small sample of such training data can be seen in Figure 4.3, superimposed on the target function.

All the suggested values for LWPR parameters presented in the cross 2D example given in LWPR source code (Klanke and Vijayakumar, 2009), namely $\boldsymbol{D}_{init} = 50\mathbf{I}$, were adopted. SOGP was left with its default parameters of $\sigma_k^2 = \sigma_0^2 = 0.1$, with no limit on the number of inducing points. As for IMLE, $\boldsymbol{\Sigma}_0 = 0.02\mathbf{I}$ was chosen to match LWPR initial input covariance matrix, while defining $\boldsymbol{\Psi}_0 = 0.1^2$ and making $n_\Sigma = n_\Psi = n_\sigma = n_\psi = 2^d = 4$. For comparison purposes two values for the parameter p_0, $p_0 = 0.1$ and $p_0 = 0.2$, were considered. Finally, some offline algorithms were also considered whose training was conducted in batch mode: besides a standard GP model, Gaussian Mixture Regression was also considered by adapting a mixture of Gaussians in the joint input-output space to the training data.

All the online algorithms were trained on a set of 200 000 sequential points coming from a random trajectory in the input space, and the prediction root mean square error (RMSE) was evaluated on a noiseless test grid of 200 by 200 equally spaced input points and corresponding output values. As for the offline methods, the GP model was trained using a random, non sequential training set with two different sizes ($N = 1\,000$ and $N = 5\,000$), while the GMM used the Matlab Statistics Toolbox to adapt a mixture of 60 Gaussian models to an also random, non sequential dataset with 10 000 training points.

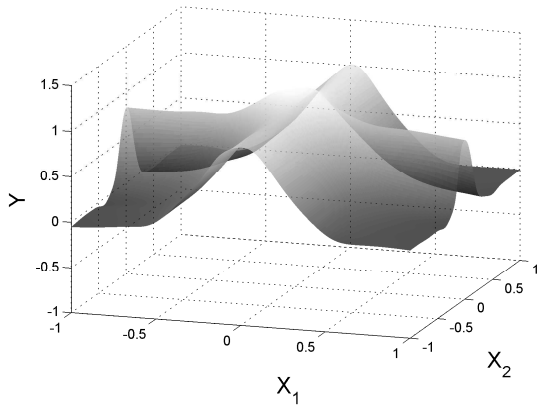
Figure 4.4 shows a typical reconstruction of the original target function by the different

learning algorithms. Also, it is shown in Figure 4.5 the evolution of RMSE and number

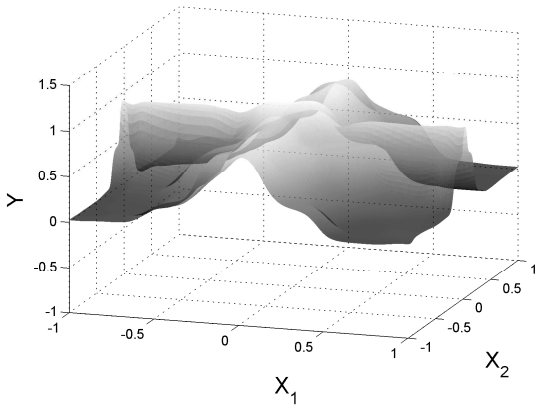


(a) IMLE ($p_0 = 0.1$): RMSE = 0.0285.

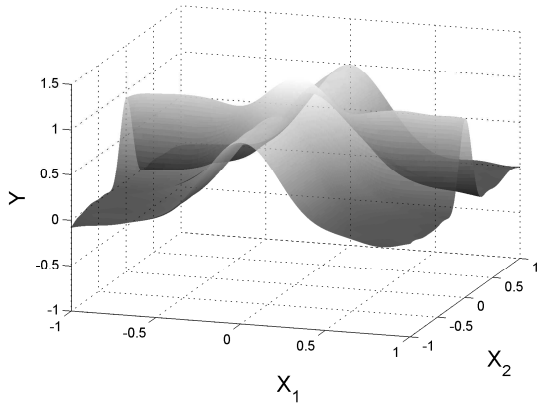
(b) IMLE ($p_0 = 0.2$): RMSE = 0.0206.



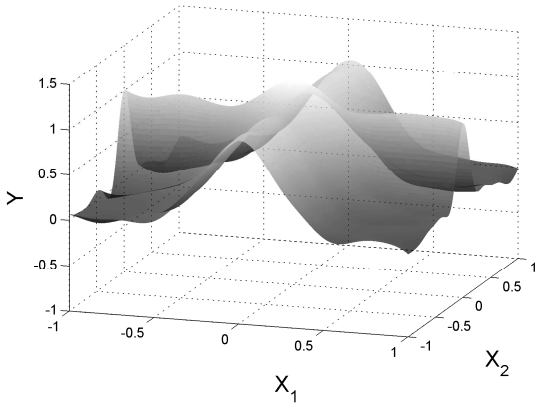
(c) LWPR: RMSE = 0.0494.



(d) GMR: RMSE = 0.0989.



(e) SOGP: RMSE = 0.0145.



(f) GP (5 000 points): RMSE = 0.0261.

Figure 4.4: Reconstructed cross function.

of created experts as a function of the processed training points, for the online learning methods; the final results for all learning methods, after training, are summarised in Table 4.1.

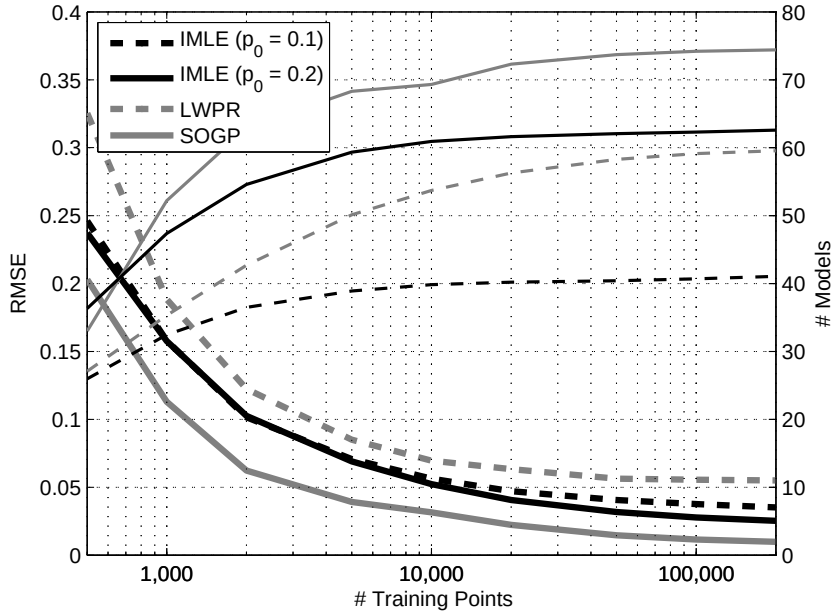


Figure 4.5: Learning curves for the Cross 2D function: RMSE and number of models created. For better visualization, the number of stored induced points of SOGP is scaled by 10. Also, for accuracy, the results presented for IMLE and LWPR are averages over 100 random trials.

Method	RMSE	# Models	CPU time (s)	
			Train	Test
$IMLE_{0.1}$	0.0351	41.04	6.3	6.6
$IMLE_{0.2}$	0.0252	62.58	9.5	9.9
LWPR	0.0550	59.55	2.3	3.1
SOGP	0.0145	744	2 600.9	593.9
GPR_{1000}	0.0563	—	—	—
GPR_{5000}	0.0261	—	—	—
GMR_{10000}	0.1691	60	—	—
$greedyGMR_{10000}$	0.2951	50.1	—	—

Table 4.1: Final RMSE, number of models and spent CPU time for the Cross experiment. For IMLE, LWPR and SOGP, training was performed online, using a 200 000 points training set. The 40 000 points test set was used 9 times during the training phase to obtain the RMSE evolution for the different learning algorithms. The GMR results here presented are an average over 10 different trials. Notice that the number of models is not shown for GPR as this is a global method; also, the offline algorithms were trained in batch mode, using a reduced training set: this makes their training and testing times not comparable to the training and testing times spent by the online learners, and thus these quantities are not presented in the table for GPR and GMR.

It can be seen that SOGP achieves the best RMSE, but at a much higher computational cost, compared to the other online methods. LWPR, on the other hand, is the fastest algorithm, but has a worse function approximation error when compared to IMLE, even

when IMLE resorts to less local models ($p_0 = 0.1$). Note that increasing p_0 results in a better function approximation, at a penalty on the number of linear experts activated and consequent increase on computation time. As for offline GPR, low error rates can be obtained if the training set is large enough, but this comes at a prohibitively cost in terms of offline computation time and memory required to perform the necessary matrix inversions. Also worth of notice is the fact that a random set was used to train the GP: as stated before, this may not be easy to generate in many practical real-time applications. This may also explain why SOGP has a better RMSE than GPR while using significantly less training points for prediction, since SOGP keeps only the most informative points taken out of the full training set.

Finally, note the poor performance of the GMR algorithm based on a mixture of Gaussians: the average RMSE achieved over 10 independent trials is significantly higher than the RMSE corresponding to the other methods. The GMM greedy learning approach of Verbeek, Vlassis, and Kröse (2003) was also tested in this setting, where each component of the mixture is added in a sequential manner, which in principle can potentially prevent the EM algorithm from becoming trapped in a poor local maxima of the likelihood of the training data. The corresponding results are also presented in Table 4.1: however, using this algorithm lead to even worse results. This experience seems to suggest that using a mixture of Gaussian models in the joint input-output space to describe a relation from inputs to outputs is difficult, as the training procedure may easily lead to sub-optimal final mixture models. While, as reported for instance in (Figueiredo and Jain, 2002), such mixture models may be appropriate to describe training data that is indeed generated from such mixture models, things start to become more complicated when data is generated from a d-dimensional subspace embedded in an input-output space with $d+D$ dimensions, even for low values of d, as seen in this example. Possibly some better results would be obtained if some constraints were defined on the covariances of the mixture components, together with the introduction of some kind of regularization mechanisms: but in the end introducing such constraints and regularization priors would lead to a generative model very similar to the IMLE model.

4.1.3 The PUMA 560 Serial Robot, $\mathbb{R}^6 \mapsto \mathbb{R}^3$

The Unimation PUMA 560 is a well known six degrees of freedom industrial robotic arm: its forward kinematics function is described, for instance, in (Craig, 1989). To evaluate the single-valued prediction capabilities of IMLE, the PUMA 560 robot kinematics was simulated, defining a 10 cm tool extending along the z-axis of the frame associated with the 6^{th} joint, and random trajectories over the joint space of the robot were generated, considering the multivariate sensorimotor map output to be the corresponding 3D position of the tool tip. The kinematic function to be learned is thus a map from an input space of dimension 6 to an output space of dimension 3, even if the last joint is irrelevant as it

only changes the tool orientation. Note that the fully stretched arm plus tool measures more than 90 cm, which makes the range of each of the output variables to be almost 2 meters.

Both LWPR and IMLE learning algorithms were trained with a set of 10 million training points, and the final achieved RMSE on a different test set comprising 100 000 points was evaluated. Output training values were corrupted with Gaussian noise, with standard deviation equal to 2 cm — this corresponds approximately to 1/100 of the output range, and can model, for instance, moderate noise in a vision based end-effector tracking process. SOGP, unfortunately, was left out of the comparisons: its parameters turned out to be difficult to tune and its behaviour unstable and very slow in face of a large stream of highly correlated input data. Standard GPR was also considered, by generating a random set of training data for hyperparameters optimization and testing on the independent test set: once again, as stated before, the results thus obtained do not compare fairly to the other online algorithms, as one of the most challenging difficulties that arise with this dataset is the massive, sequential and correlated nature of the training data.

Experimental results are shown in Table 4.2, where the RMSE for two different instances of the GPR algorithm, trained with 1 000 and 5 000 data points, is presented.

Method	RMSE	#Models	CPU time (s)	
			Train	Test
IMLE	0.0245	668	4 181	141
LWPR	0.0560	4 338	11 974	725
GPR_{1000}	0.0513	—	—	—
GPR_{5000}	0.0144	—	—	—

Table 4.2: Results on the PUMA dataset for IMLE, LWPR and GPR learning algorithms: final RMSE, number of models and spent CPU time.

Parameters for the IMLE model were set in the usual manner, defining $\alpha = 0.999$, $p_0 = 0.1$ and $n_\sigma = n_\Sigma = n_\Psi = 2^d = 64$. To avoid giving IMLE any information regarding the input length-scale and output noise characteristics of the function to be learned, parameters related to these quantities were set to broad, uninformative values $\mathbf{\Psi}_0 = \mathbf{I}$ and $\mathbf{\Sigma}_0 = \mathbf{I}$, and n_ψ was set to a low value of 8 in order to quickly decay the influence of $\mathbf{\Psi}_0$ in the estimate for $\bar{\mathbf{\Psi}}$. As for LWPR, the combination of parameters that made LWPR achieve the lowest error was picked, paying attention not to let the number of created receptive fields grow to unacceptable values, namely $\gamma = 10^{-8}$, $w_{gen} = 0.1$, $\boldsymbol{D}_{init} = 10\mathbf{I}$ and $\alpha_{init} = 50$. Tuning these parameters to have roughly the same number of allocated models as IMLE never produced a RMSE of less than 10 cm on each output dimension. Figure 4.6 shows the corresponding learning curves for IMLE and LWPR. Due to the conservative large parameter values for $\mathbf{\Psi}_0$ and $\mathbf{\Sigma}_0$, IMLE converged slowly in the initial

learning phase, but quickly recovered after convergence of $\bar{\mathbf{\Psi}}$ and $\bar{\mathbf{\Sigma}}$ to better output noise and input length-scale estimates, respectively.

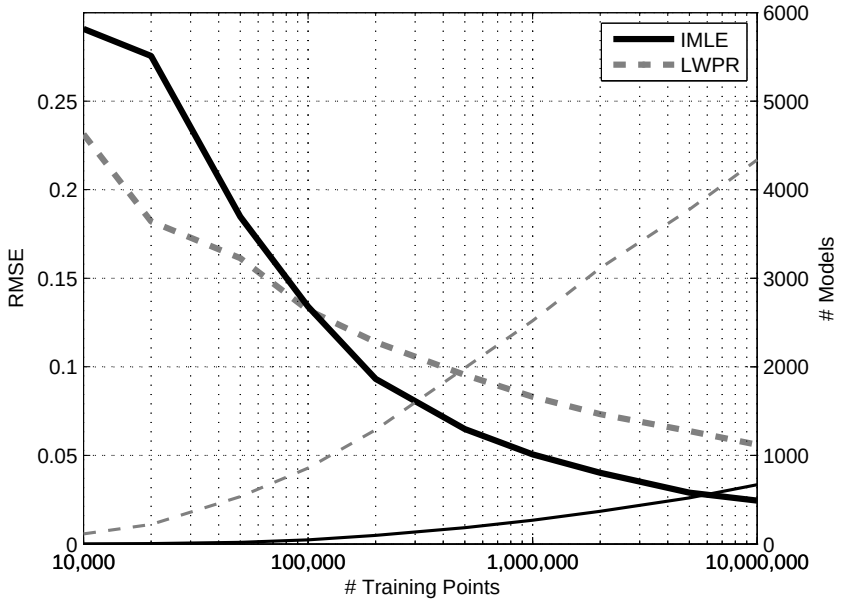


Figure 4.6: Learning curves for the PUMA dataset, for a typical parameter configuration.

It is important to stress out that, in order to provide a fair comparison to the IMLE model, the LWPR algorithm was tested over an exhaustive combination of parameters, namely $\gamma \in \{10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}\}$, $\alpha_{init} \in \{10, 20, 50, 150, 250\}$, $w_{gen} \in \{0.1, 0.2\}$ and $\boldsymbol{D}_{init} \in \{10, 20\}\mathbf{I}$. These parameters affected the final error through the number of receptive fields they tended to create: the larger this number the lesser the error, as expected. The same happened to IMLE when varying its tunable parameters, with $\mathbf{\Psi}_0 \in \{1.0, 0.0016, 0.0001\}\mathbf{I}$, $\mathbf{\Sigma}_0 \in \{4.0, 1.0, 0.25\}\mathbf{I}$, $n_\psi \in \{64, 32, 8\}$, $n_\sigma \in \{64, 32, 8\}$ and $p_0 \in \{0.03, 0.1, 0.3\}$. In general, for both learning algorithms, a lower RMSE would generally be achieved at a cost of an increasingly number of local linear models.

It is very illustrative to depict the final RMSE as a function of the number of models created, for each parameter configuration of IMLE or LWPR, as in Figure 4.7, since the number of local models affects the computational training and prediction time in a similar way for these two algorithms. In the figure three different convergence behaviours can be identified for the IMLE model: the first one, marked IMLE (Global), corresponds to a low value of $\mathbf{\Psi}_0 = 0.0001\mathbf{I}$. Since this value of $\mathbf{\Psi}_0$ is lower than the actual output noise, IMLE activates a large number of linear experts in the initial learning phase, each one covering all the input space (due to a high $\mathbf{\Sigma}_0$) and a particular region of the output space. This is of course an undesirable behaviour, that goes against the principle of localized learning. The second case, identified as IMLE (Overfitting), is a consequence of choosing a low value of $\mathbf{\Sigma}_0$ in combination with a low/medium value of n_σ, allowing the individual $\mathbf{\Sigma}_j$ to shrink to a point where each new training point has a high probability of activating

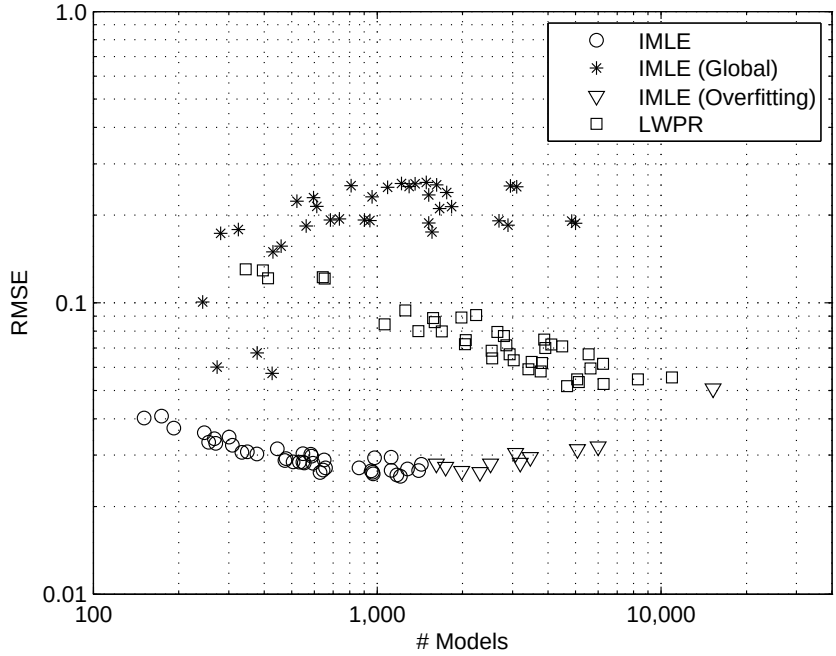


Figure 4.7: Effect of parameters variation on approximation error and number of local models created. Each marker represents a particular configuration of parameters of the learning algorithm, and its location reflects the number of local linear models created (x-axis) and RMSE (y-axis) achieved after training using data arising from the PUMA 560 kinematics model.

a new expert. This may result in a snow ball effect, where more experts contribute to a decrease of $\bar{\mathbf{\Sigma}}$, which in turn will lead to smaller $\mathbf{\Sigma}_j$ and consequently more experts being activated as the parameter values are pushed towards the boundary of the parameter space. As seen in Figure 4.7, this does not result in a reduction of RMSE in the test set, a good indicator that IMLE is overfitting in that situation. These are, however, two extreme situations, caused by setting $\mathbf{\Psi}_0$ or $\mathbf{\Sigma}_0$ to smaller values than the output noise and input length-scale, respectively. All other combinations of parameters exhibit a good robustness regarding the training convergence. Note that, in this case, IMLE has a much better performance than LWPR, achieving a much smaller RMSE while activating considerably less local models.

4.1.4 The SARCOS Inverse Dynamics Dataset, $\mathbb{R}^{21} \mapsto \mathbb{R}^7$

The SARCOS anthropomorphic robotic arm is a seven degrees of freedom manipulator that has been used to test several function approximation algorithms. The learning task considered here is the estimation of its inverse dynamics from training examples: this is a nonlinear map from a 21-dimensional input space, consisting of positions, velocities and accelerations of each joint to a 7-dimensional output space comprising the corresponding joint torques. Such learned model can then be used to estimate the torques that achieve a desired trajectory in the joint space. The dataset consists of 48 933 training points and

4 449 test points, taken from trajectories performed with the real robot. Output values are normalized by the variance of the outputs of the test set, to make the results presented here directly comparable to the ones in (Rasmussen and Williams, 2006; Vijayakumar, D'Souza, et al., 2002). The IMLE model results on this dataset are compared to the following models, taken from the aforementioned works:

Linear Regression (LR): A linear regression model is fitted to the data, to provide a baseline for comparison;

Rigid Body Dynamics (RBD): This is a parametric, physics based model for the inverse dynamics function that is estimated using a least-squares approach, using the available training data;

LWPR: This model was trained using diagonal distance metrics $\boldsymbol{D}_j$, cycling through the training data over 6 000 000 iterations: this roughly corresponds to 123 passes over the full training data.

GPR: Due to the computational infeasibility of using the full training data for optimization of hyperparameters and prediction over test data, a subset of regressors method was employed, with size 4 096. A squared exponential covariance function was used, and its hyperparameters were optimized accordingly, using a subset of the training data.

As for IMLE, "out-of-the-box" default parameters were used, setting conservative large values for its noise and length-scale parameters, namely $\boldsymbol{\Psi}_0 = \mathbf{I}$ and $\boldsymbol{\Sigma}_0 = \mathbf{I}$, while setting $p_0 = 0.1$. Using the input dimension to set up the value of n_Σ, n_Ψ and n_σ, according to the rule of thumb presented before, would result in overly large prior strengths; instead, one must resort to the notion that, for actual robot movements, the generated data tends to be low dimensional, with around 4 — 6 effective dimensions (Schaal, Vijayakumar, and Atkeson, 1998). This is a standard assumption in robotic applications, in order to circumvent the curse of dimensionality: learning a sensorimotor map would require a full exploration of the input space if this condition did not hold. In such case, as the input dimension increased, the time required for exploration and learning would grow exponentially, making the learning task infeasible from a practical point of view. In the following experiments, it was found that a value of 7 effective dimensions could provide satisfactory learning behaviour, and thus some of the IMLE model parameters were set accordingly, by making $n_\Sigma = n_\Psi = n_\sigma = 2^7$. The value of n_ψ was set to a low value of 8, due to the high uncertainty on $\boldsymbol{\Psi}_0$: note however that varying n_ψ over a considerable wide interval around this value did not change IMLE convergence behaviour significantly.

Differently from GPR and LWPR, the IMLE model can directly provide multivariate output predictions without the need to train different models for each of the output

dimensions. If the maps from inputs to each output variable have the same length-scale properties, as frequently happens in robotic sensorimotor maps, a huge economy of computational resources can be attained, as each local linear model can describe the interaction between the inputs and the full output vector. It is show, in Table 4.3, the prediction error when the output consists only of the first joint torque, comparing it to the results presented in (Rasmussen and Williams, 2006) for LWPR, GPR, RBD and LR. The IMLE model was also used to learn the full output torque vector: for both situations, the Mean Square Error (MSE) and number of activated experts after (a) a full pass over the training data and (b) 10 consecutive passes over the same data are presented[1]. For confirmation purposes, IMLE was also trained with $p_0 = 0.0$, resulting in

Method	MSE	#Models
LR_{1D}	0.075	—
RBD_{1D}	0.104	—
LWPR_{1D}	0.040	260
GPR_{1D}	0.011	—
IMLE_{1D} (1 epoch)	0.019	313
IMLE_{1D} (10 epochs)	0.010	563
IMLE_{7D} (1 epoch)	0.018	271
IMLE_{7D} (10 epochs)	0.010	550

Table 4.3: Prediction performance results on the SARCOS dataset for LR, RBD, LWPR, GPR and IMLE methods. IMLE_{1D} is the model corresponding to a map from $\mathbb{R}^{21}$ to $\mathbb{R}^1$, while IMLE_{7D} is the model obtained from the full map from $\mathbb{R}^{21}$ to $\mathbb{R}^7$.

a model with a single expert that provided a global linear approximation to the training data: as expected, a MSE equal to that of the LR model was then obtained.

It is noteworthy the extremely good convergence of the IMLE algorithm: after only a single pass through the training data IMLE achieves a better approximation error than LWPR (after more than 100 passes through the same data) while activating a comparable number of linear models. If more points are presented to IMLE, cycling through the data 10 times, a MSE comparable to state-of-the-art GPR is even achieved. It is also worth of notice the fact that IMLE performance did not change much when a full 7-dimensional output vector was considered for the learning task: the MSE remained the same, while, perhaps surprisingly, the number of experts even dropped a bit[2]. No computational time increase was noticed in this situation, as the slightly increase in computation due to this higher output dimension was balanced by a smaller number of activated models. In this aspect IMLE compares very favourably to GPR and LWPR, which would require 7 times more computational power to learn the full inverse dynamics map.

[1] To make the results directly comparable to the ones found in (Rasmussen and Williams, 2006), Table 4.3 shows the MSE instead of the customary RMSE: to obtain this latter value it suffices to take the square root of the presented values, of course.

[2] This probably is explained by a lesser tendency for overfitting when the output dimension increases.

4.2 Multi-valued Function Approximation

Next, the IMLE model prediction capabilities are evaluated under a multi-valued target function scenario. As stated in section 3.2.2, learning multi-valued functions in an online fashion poses several additional problems: without further information, it is difficult to distinguish noise or outliers from new multi-valued function branches. Additionally, it is no longer possible to set large values for $\mathbf{\Sigma}_0$ and $\mathbf{\Psi}_0$, hoping that the learning process finds adequate values for the input length-scale and output noise estimate, as multi-valued relations may then be interpreted as single-valued functions with a large output noise. There is also the problem of time-varying functions: this issue can be easily addressed in online single-valued algorithms by introducing some kind of forgetting mechanisms, to allow for a quick adaptation of the internal model to the time-varying data. However, things become a bit more unclear when multi-valued data is considered, as fast time-changing training data may become, from an online learning algorithm point of view, undistinguishable from data originated by a multi-valued input-output relation, without any clear distinction between these two situations.

The IMLE model was compared, whenever possible, to the ROGER algorithm during the following tests. This latter algorithm, however, consists in an infinite mixture of SOGP's, thus inheriting the same limitations mentioned in the previous section, namely the difficulty in getting a good parameter configuration and the slow operation for medium or high dimensional input spaces. Additionally, ROGER implementation code does not provide a set of multi-valued solutions for the function being approximated, instead sampling a single solution from the infinite mixture. This may prove to be an highly undesirable feature when robotic applications are considered, as ROGER predictions will permanently probabilistically switch between different multi-valued branches, which may become unacceptable from a control point of view, where typically smooth movements are desired. To provide a fair comparison to IMLE, ROGER code was adapted to provide a finite set of prediction solutions in the following way: for each input query, ROGER was asked to repeatedly obtain a forward prediction for the same input query (100 different trials), this way generating a random sample of prediction over its set of particles; the distinct solutions thus obtained were gathered to become the set of predicted multi-valued solutions.

Approximation error was measured by taking, for each input query, the multi-valued prediction closest to the true output. Note that the important problem of choosing a solution among the set of multi-valued prediction produced by the algorithm is not addressed in the following experiments. This can be seen as a context estimation, and it is a requirement when the multi-valued model is to be used for control purposes, as discussed in Chapter 5. In the following experiments α_{multi} was set to 0.1: the higher this value the higher the number of solutions found for the same query, on average; however,

it was found that the value of this parameter did not influence much the final prediction when the branches of the multi-valued function to be learned happen to be well separated in the output space, and that setting it on a range of, *e.g.*, $\alpha_{multi} \in [0.01, 0.2]$ resulted many times in similar prediction results.

4.2.1 Synthetic Datasets

To illustrate the fundamental differences between multi and single-valued function approximation algorithms, a simple toy example is presented next, consisting of a multi-valued target sinusoidal function. Standard function approximation learning methods, like LWPR or GPR, typically behave poorly in this setting, due to the multi-valued nature of the true input-output relation to be learned. Training data was generated from this relation by alternating sequential sweeps over each of the two branches of the multi-valued function, given respectively by $f_1(x) = cos(x)$ and $f_2(x) = cos(x) + 4$, and each output sample y_i was corrupted with Gaussian noise with standard deviation equal to 0.1. Figure 4.8 represents a sample of the training data, while Figure 4.9 depicts the ap-

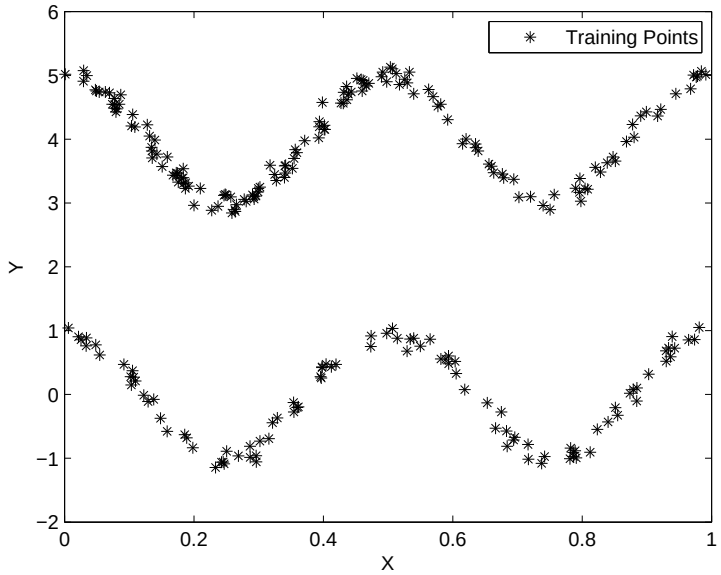


Figure 4.8: Training data coming from a very simple multi-valued learning example.

proximations provided by IMLE, ROGER, LWPR and GPR to this input-output relation, taken after training.

As expected, LWPR and GPR tend to average the two branches of the multi-valued function, while IMLE and ROGER correctly identify the two solutions. In particular, ROGER achieves an almost perfect approximation to the true function (RMSE = 0.014), when compared to IMLE (RMSE = 0.039).

The next multi-valued function approximation example is taken from the work of Shizawa (1996), that suggests a synthetic dataset consisting in a multi-valued function

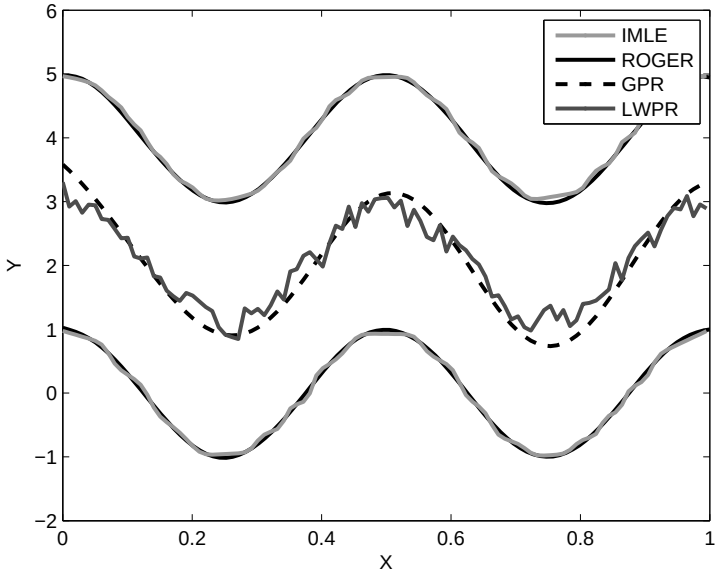
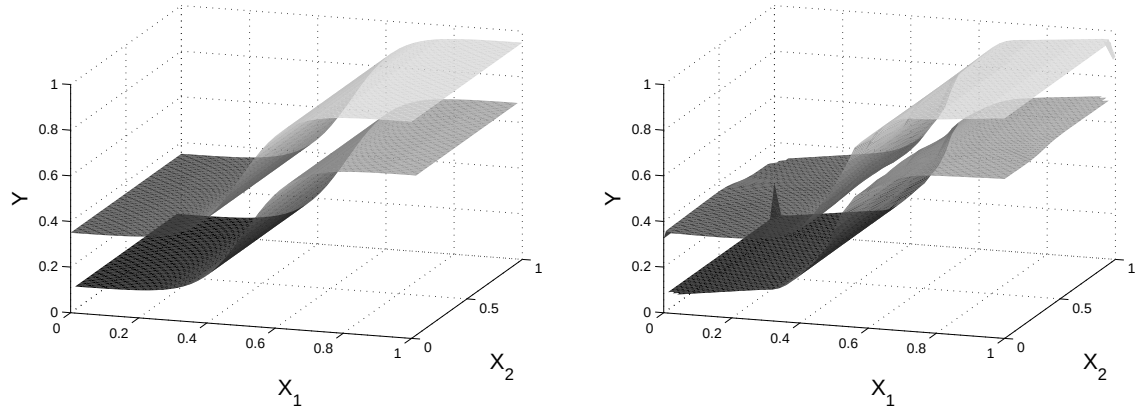


Figure 4.9: Reconstructing the relation shown in Figure 4.8 using four distinct learning algorithms.

from $\mathbb{R}^2$ to $\mathbb{R}^1$, with two distinct branches, described by equations

$$f_1(x_1, x_2) = \frac{0.6}{1 + e^{-15(x_1 - 0.5)}} + 0.1 \quad \text{and} \quad f_2(x_1, x_2) = \frac{0.6}{1 + e^{-15(x_1 - 0.5)}} + 0.35 \; .$$

Such target function is depicted in Figure 4.10a, while IMLE multi-valued prediction, after performing a 10 000 points training phase, is represented in Figure 4.10b.



(a) Target function. (b) IMLE prediction.

Figure 4.10: Multi-valued toy example suggested by Shizawa (1996).

This toy example is particularly challenging for multi-valued prediction: the two function branches are very close to each other and can easily become merged during learning. IMLE achieves a RMSE of 0.017 in a grid-like independent, noise free test set. ROGER,

on the other hand, was not able to generate a good estimate: it systematically produced, for every tested parameter configuration, more than 20 SOGP experts. This led to a severe output prediction interference, with typically more than 10 solutions generated for each input query[3].

In the two previous examples the multi-valued functions have two distinct, well separated function branches. It is very instructive to examine what happens when such branches blend together, as depicted for instance in Figure 2.15. This is shown in Figure 4.11, where the final multi-valued prediction after a 50 000 points training phase is shown, together with some representative training points of this S-shaped function.

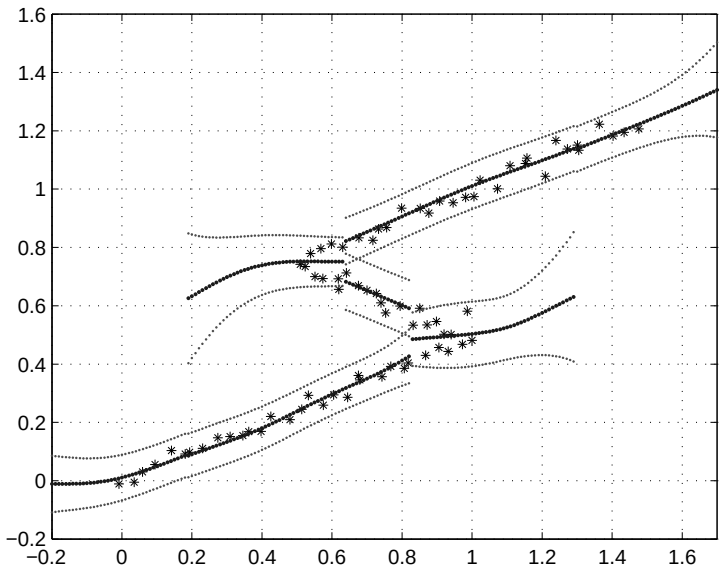


Figure 4.11: Multi-valued prediction of the S-shaped function. A subset of the training data is represented by stars, while the IMLE model predicted outputs for each input query are represented by black dots. The grey dots represent a 3 standard deviation confidence interval for each predicted value.

It can be noticed that the final multi-valued prediction is not completely representative of the true distribution of the underlying data. This is a consequence of the local nature of the prediction process: for a particular query point, the prediction is obtained using only the output prediction provided by each linear expert at the same query point. There is no way of knowing that a particular multi-valued prediction does not correspond to the observed training data.

There is not a simple answer to this issue: one possible direction of research, that will not be followed in this dissertation, could consist in also exploiting the information provided by the mixture in a region around the query point. Note however that the

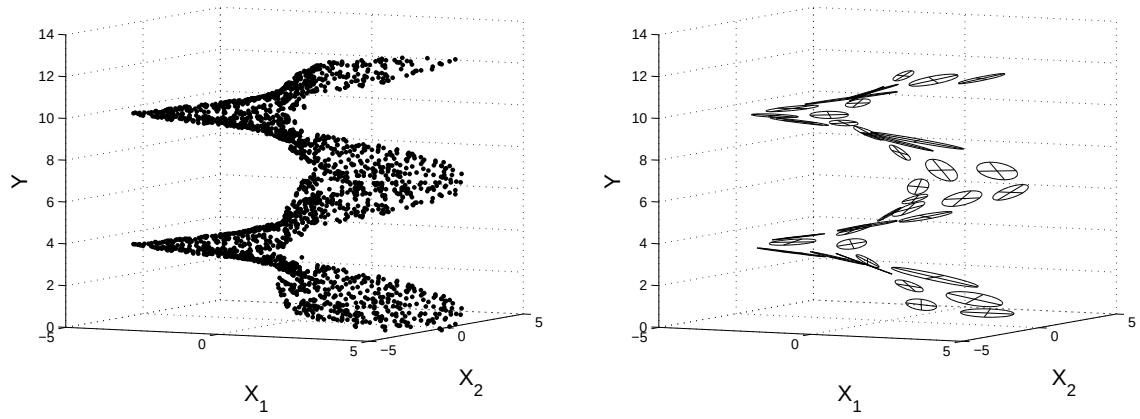
[3]This behaviour was unfortunately observed in the following experiments, and so ROGER was removed from the remaining tests. Note that ROGER good results shown in Figure 4.9 were a consequence of a careful choice of its parameters: small changes in these values would typically lead to more than two solutions being predicted in many regions of the input space.

example provided in Figure 4.11 is perhaps an extreme case of such behaviour. In general this is not a critical issue, as can be observed in the next examples concerning inverse and forward multi-valued prediction.

Another toy example, presented now by Lee and Lee (2001), consists in a cylindrical spiral surface, again a multi-valued function from $\mathbb{R}^2$ to $\mathbb{R}^1$, described by

$$y = tan^{-1}(\frac{x_2}{x_1}) \ , \qquad \text{where } y \in [0, 4\pi[, \text{ and} \qquad 0.2 < \sqrt{x_1^2 + x_2^2} < 5 \ .$$

The co-domain of the target function makes each input point to have two distinct solutions. IMLE was trained with random trajectories generated over the target function, and after 50 000 points it achieved a RMSE of less than 0.065 on an independent random test set. The multi-valued predicted outputs, after training, for this test set are shown in Figure 4.12a, and the corresponding IMLE linear models, generated during the training phase,are represented in Figure 4.12b.



(a) IMLE multi-valued prediction over the test set.

(b) The 39 linear experts allocated by the IMLE model after training.

Figure 4.12: IMLE prediction using the toy example suggested by Lee and Lee (2001).

The final toy example here presented consists in a randomly generated piecewise constant target function from $\mathbb{R}^2$ to $\mathbb{R}^1$, shown in Figure 4.13. Although not multi-valued, many function approximation algorithms will not be able to properly learn it, as their smoothness assumptions can severely conflict with the discontinuities of the target function. This will typically result in either an overfitting to the data, when the complexity of the model is increased to approximate the function in the vicinity of the discontinuities, or in an oversmoothing behaviour, where the predictions average the two values of the target function near the function transitions.

Multi-valued learning algorithms based on mixtures, like IMLE, can provide an elegant solution to this problem. During training, there is no output interference near the discontinuities, since in those regions the data is simply interpreted as coming from a multi-valued function. When predicting, a single-valued solution can nevertheless be pro-

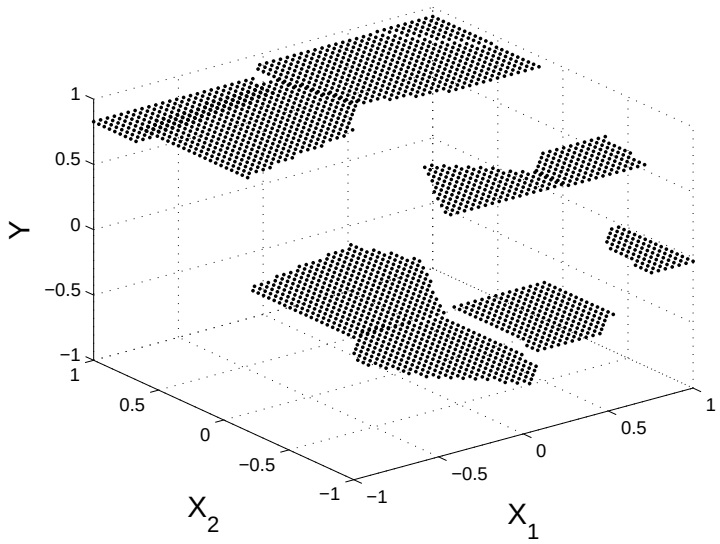
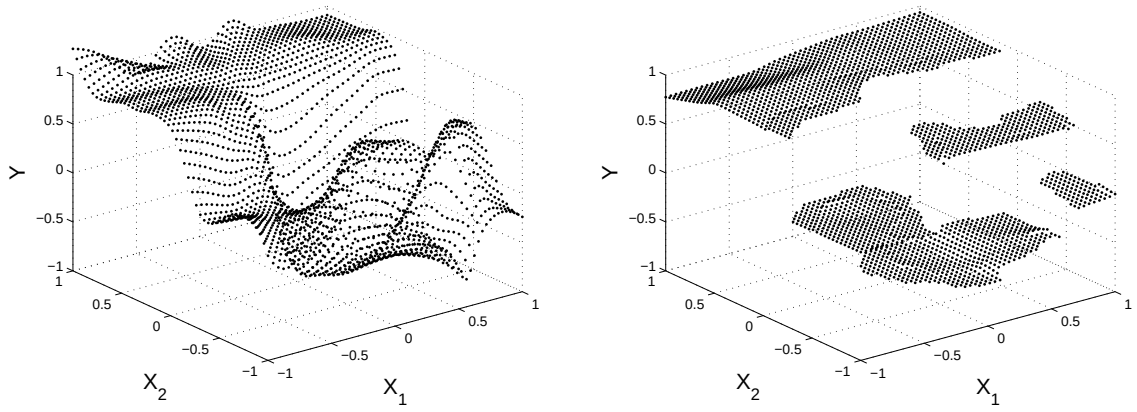


Figure 4.13: Discontinuous target function to learn.

vided by simply taking the most important solution from the multi-valued prediction set, according to the sum of weights $w_j^y(\boldsymbol{x}_q)$ of the experts that contribute to that solution, as given by equation (3.24c), instead of the standard single-valued prediction that simply averages over all experts individual estimates. This procedure is able to generate sudden transitions of the prediction, like the algorithm developed by Toussaint and Vijayakumar (2005) to specifically deal with discontinuous functions. Prediction results are shown in Figure 4.14, for IMLE and a single-valued learner, represented in this case by LWPR.



(a) Single-valued prediction (LWPR). (b) Best multi-valued prediction (IMLE).

Figure 4.14: Reconstruction of a discontinuous target function using a single and multi-valued learner.

Note that the purpose of this experiment is mainly to provide the proof of concept that IMLE (and other multi-valued learning algorithms) can also efficiently approximate discontinuous functions without the need for any special modifications; although a full comparison of IMLE to other learning algorithms capable of efficiently approximating

functions with discontinuities is certainly a topic worth investigating, such direction of research will not be further pursued in this section.

4.2.2 iCub Inverse Dynamics Learning Under Different Loads, $\mathbb{R}^{12} \mapsto \mathbb{R}^4$

As stated in Chapter 1, the inverse dynamics learning problem consists in approximating the relation $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \mapsto \boldsymbol{u}$ using training points sampled from this relation, where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are vectors containing respectively the positions, velocities and accelerations of the joints of a robot and where $\boldsymbol{u}$ is the motor command that actuates these same joints. This sensorimotor map can then be employed to control the dynamics of such robot, as it provides, for a given robot state represented by $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, the motor command $\boldsymbol{u}$ required to generate a desired joints vector acceleration $\ddot{\boldsymbol{q}}$.

In the next example, training data acquired from a real robot is provided to the IMLE model to approximate its inverse dynamics map: the robot used to obtain the training and test sets is the humanoid robot iCub, a 53 degrees of freedom humanoid robot for research in embodied cognition (Tsagarakis, Metta, et al., 2007; Metta, Natale, et al., 2010; Parmiggiani, Maggiali, et al., 2012; Natale, Nori, et al., 2013), depicted in Figure 4.15. During data acquisition only 4 of the iCub joints were actuated, corresponding to the

Figure 4.15: The iCub robot holding a water bottle: this changes its inverse dynamics relation.

shoulder and elbow joints of the right arm. Training consisted in performing random movements in this joint space, while acquiring the positions, velocities and accelerations of these joints, together with the corresponding commands sent to iCub motors. A training set of 20 000 points was acquired in this way, to be posteriorly used by the IMLE algorithm to approximate this sensorimotor map from $\mathbb{R}^{12}$ to $\mathbb{R}^4$.

After this data acquisition was performed a second experiment was conducted, similar to the previous one, but where now a bottle of water was held by the iCub right hand. As before, 20 000 training points were acquired by randomly moving the iCub right arm.

The inverse dynamics relation is well known to be a single-valued function, where knowledge of the input triplet $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ suffices to obtain the corresponding motor command $\boldsymbol{u}$. These two experiments thus correspond to data acquisition from two distinct single-valued inverse dynamics relations.

Things become interesting when the context within which the experiments are performed, *i.e.*, the presence or absence of the water bottle, is unknown to the robot, either during training or during exploitation of the learned model for control purposes. In such situation, and from the robot point of view, the sensorimotor map becomes a multi-valued relation with two distinct branches, as the same configuration $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ can be associated with two different motor commands, depending on whether the robot is holding the water bottle or not. Learning in the presence of hidden contexts is a challenging problem and is discussed with more detail in Section 5.3.

To evaluate the IMLE model prediction capabilities for this multi-valued problem, the experiments proceeded as follows: first, the IMLE model was trained with the dataset corresponding to the first context, where no bottle was held by the robot. After these 20 000 points were presented sequentially to the learning algorithm, the second sequence, corresponding to the presence of the bottle, was then fed to the IMLE model. After that, the original sequence was again used to train the IMLE model. It is important to notice that, during this procedure, the change of context was never signalled in any way to the learning algorithm: from its perspective, there was a single training phase consisting of a sequence of 60 000 data points taken from the same, possibly multi-valued, relation. During this training phase the performance of the IMLE model was evaluated using two different test sets of 2 000 points each, consisting in sequences of data points acquired in the same conditions as the training data, *i.e.*, during the random movement of iCub joints with and without the water bottle. The results are shown in Figure 4.16.

The initial phase of the training process shows an evolution typical of many online learning algorithms: an increase of the complexity of the probabilistic model, represented by the number of allocated linear experts, results in a corresponding decline of the prediction error. When the context is changed and training data starts being acquired while the iCub robot holds the bottle the RMSE suffers a sudden and expected increase, as the test set where this error is evaluated has also changed to reflect the new iCub dynamic configuration. In this situation, as training goes on, the RMSE decreases and more linear experts are assigned to the mixture, in a very similar way to the initial training phase. However, differently from the first part of the learning process, a fast increase of the average number of solutions found on the test set is observed, that quickly converges to a value very close to 2. This means that the internal model corresponding to the first context was not forgotten by the IMLE model, that can now predict two different motor commands that, in a given joint configuration, will produce some desired joint acceleration. These two multi-valued solutions are, of course, the motor commands corresponding to the two

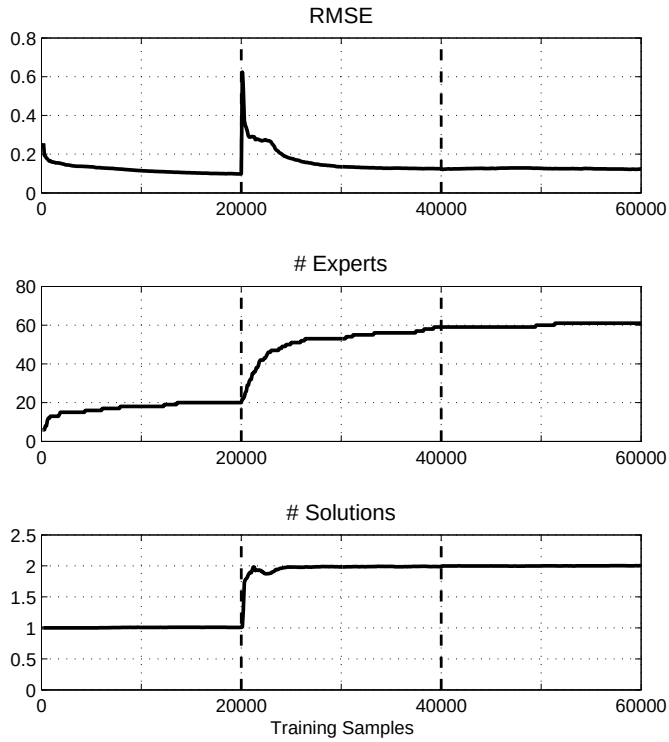


Figure 4.16: iCub inverse dynamics learning curves: RMSE, number of allocated experts and average number of solutions found during the learning process.

distinct dynamic configurations of the iCub robot that arise when this robot is holding the bottle and when it is not.

The final training phase just confirms the adequate learning of the multi-valued relation by IMLE: when training points, generated from the initial context, are presented again to this model the RMSE suffers almost no change and the average number of predicted multi-valued solutions is kept around 2. Also, no sudden increase of the number of allocated experts was observed, meaning that the linear experts that modelled the first dynamical relation did not suffer any kind of interference from the second context training data and that the information kept in their parameters was not forgotten. Just for comparison, a simple linear regression model, trained and evaluated using the same data, has a prediction error value of RMSE = 0.331.

4.3 Inverse Prediction

As stated in Chapter 2, the IMLE model can be used at any time of the learning process to generate predictions for both forward and inverse relations. The following text describes some experiments that assess the inverse prediction capabilities of the IMLE probabilistic model.

4.3.1 PUMA 560 Serial Robot, $\mathbb{R}^3 \mapsto \mathbb{R}^3$

In this experiment the first three joints of the PUMA 560 robot arm described in Section 4.1.3 are used to position the end-effector. The kinematic function for this robot configuration is a map from $\mathbb{R}^3$ to $\mathbb{R}^3$: while the forward map consists of a single-valued function, the inverse kinematics is multi-valued and can exhibit up to 4 solutions.

The IMLE model was trained using 1 million points taken from a simulated random trajectory in the joint space of the PUMA robot, and a noiseless random test trajectory of 100 000 points was then used to evaluate the IMLE prediction capabilities. Gaussian noise was also added to the observed output training points, with standard deviation equal to 0.02, corresponding approximately to 1/100 of the output range. Ten different trials were performed, and the average prediction error is presented in Table 4.4.

#Models	Forward RMSE (cm)			Inverse RMSE (%)		
	x	y	z	θ_1	θ_2	θ_3
163.0	1.54	1.61	1.13	3.71	3.91	1.43

Table 4.4: Forward and inverse prediction performance results on the PUMA 560 kinematic map, where only the first three joints are actuated and considered.

Figure 4.17 also presents, for the PUMA robot, the frequency of the number of solutions found by IMLE inverse prediction, comparing it to the real value, obtained by explicitly solving the inverse kinematics equations. The discrepancy between these numbers can

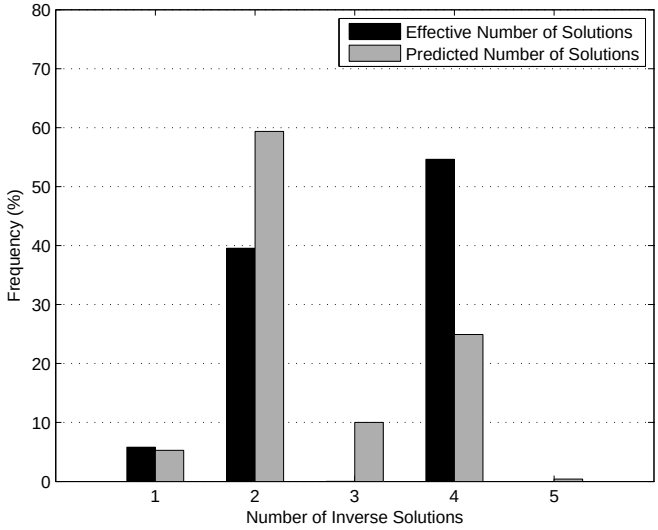


Figure 4.17: Frequency of the number of solutions found by IMLE inverse prediction for the PUMA 560 problem.

be explained by the fact that close to the workspace boundary of the PUMA 560 there are pairs of inverse solutions that become close to each other: in this situation IMLE

inverse prediction tends to merge these solutions. Increasing the value of α_{multi} would reduce this behaviour; however, due to the curvature of the map to learn, this would have the undesired side effect of predictions of neighbour experts being erroneously taken as separate solutions. In general, choosing a value for α_{multi} is a compromise between this two effects. Nevertheless, IMLE still achieves a good inverse prediction error rate, since in the workspace boundary the merged solution provided by IMLE is approximately the average of two reasonable similar true solutions.

4.3.2 Parallel 3-RPR Robot, $\mathbb{R}^3 \mapsto \mathbb{R}^3$

As a second example, the 3-RPR parallel manipulator described in (Merlet, 2006) is used to evaluate IMLE forward and inverse prediction capabilities. This robot consists in an end-effector connected to a fixed base through three prismatic links, each connecting to the base and end-effector using free, unactuated rotational joints, as represented in Figure 4.18. Its movement is restricted to the x-y plane: actuating on link lengths p_1, p_2

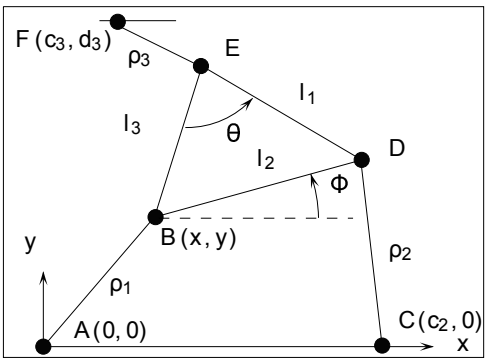


Figure 4.18: A parallel 3-RPR robot geometric scheme. Figure taken from (Merlet, 2006).

and p_3 changes the x-y end-effector position and θ orientation on this plane. The kinematic map for this mechanism is also a map from $\mathbb{R}^3$ to $\mathbb{R}^3$. Parallel robots typically exhibit a duality relation to serial chains with respect to the forward and inverse kinematics nature: while their inverse relation is usually unique and straightforward to calculate, obtaining a closed formula for the end-effector position and orientation as a function of actuator values is difficult and frequently gives rise to multiple valid solutions. This mechanism is known to have up to six different solutions for the same actuator configuration, which makes learning its forward kinematics infeasible for most standard single-valued function approximation techniques.

As in the previous example, the IMLE model was trained using 1 million points taken from a simulated random trajectory, and Gaussian noise with standard deviation equal to 1/100 of the output range was added to the output part of the training samples. The movement of the 3-RPR parallel manipulator was restricted to a square of 40 cm by 40 cm in the centre of the mechanism, while the angle was constrained to the interval $[-\pi/2; \pi/2]$.

As in the previous experiment, the obtained model was tested over a noiseless random sequence of 100 000 points, and Table 4.5 presents the final results averaged over 10 such different trials.

#Models	Forward RMSE (%)			Inverse RMSE (cm)		
	x	y	θ	p_1	p_2	p_3
91.3	0.70	0.70	1.12	0.27	0.26	0.26

Table 4.5: Forward and inverse prediction performance results on the parallel 3-RPR robot kinematic map.

The trained IMLE model found on average 1.88 forward solutions per test point, which, given the constrained workspace, agrees with the expected number of solutions. For every point on the test set only a single inverse solution was found: this is in total agreement with the single-valued nature of the inverse kinematics for these kind of mechanisms.

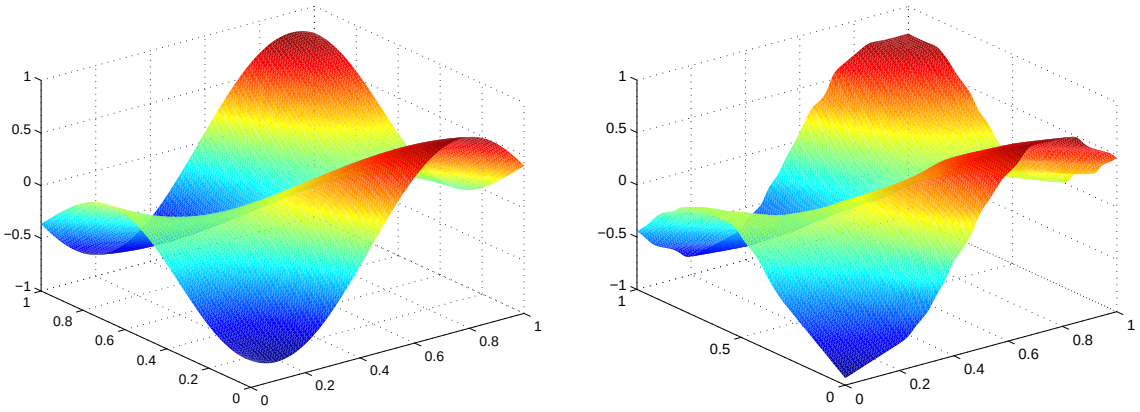
4.3.3 PUMA 560 Serial Robot, $\mathbb{R}^2 \mapsto \mathbb{R}^1$

When the input dimension of a sensorimotor map is greater than its output dimension the inverse problem becomes ill-posed, in the sense that the input locations that generate a given output value do not correspond to a finite set of multi-valued solutions: instead, generally a continuous solution space, with dimension equal to the difference between input and output dimensions of the map must be considered.

In this situation, one cannot expect the IMLE model to find such continuous inverse solution space, since the clustering procedure presented in Section 3.3.4 can only produce a finite set of inverse point estimates. Nevertheless, the inverse predictions thus obtained can in principle provide a useful sampling of such space.

To illustrate the IMLE model inverse prediction mechanism a simple kinematic mechanism is learned in this section: for ease of visualisation, a sensorimotor map from $\mathbb{R}^2$ to $\mathbb{R}^1$ is considered, consisting of the map from the second and third joints of the PUMA 560 robot described earlier in this chapter to the x coordinate of the corresponding end-effector. This, of course, corresponds to a redundant kinematic structure, as the input dimension is greater than the output dimension. Learning this map poses no problem to the IMLE model: Figure 4.19 shows the target function and the corresponding IMLE forward estimate, taken after a training phase of 100 000 points corrupted with 2 cm standard deviation Gaussian noise. After training, the IMLE model allocated 61 linear experts.

Given the learned model, a set of output queries was then presented to the IMLE algorithm, corresponding to the contour lines depicted in Figure 4.20, and a set of multi-valued inverse predictions was obtained for each query. As discussed in Chapter 3, the parameter α_{multi} strongly influences the number of solutions obtained when a multi-valued prediction is requested: the higher this value, the higher the number of solutions found



(a) Target function.

(b) Prediction provided by the IMLE model after training.

Figure 4.19: The PUMA kinematic function from $\mathbb{R}^2$ to $\mathbb{R}^1$.

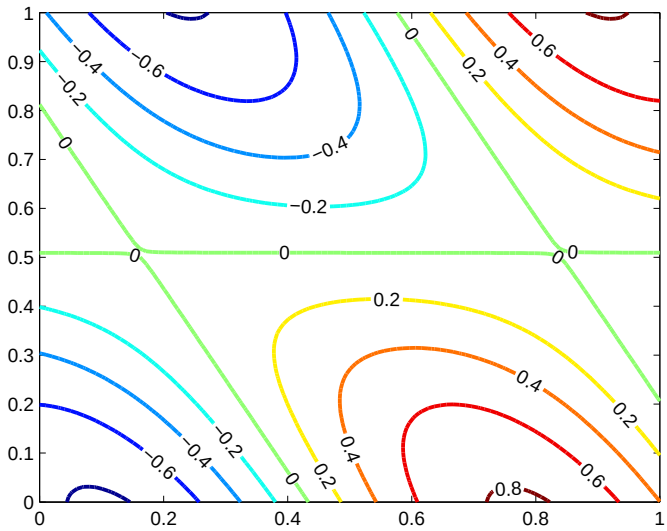


Figure 4.20: Contour plot describing the inverse kinematics of the PUMA robot. Each contour corresponds to the input subspace that generates the same x-position of the end-effector.

by the clustering procedure, on average, and to substantiate this statement, Table 4.6 presents the number of inverse solutions found by the IMLE algorithm, for a set of different output query points, using 4 different values of α_{multi}.

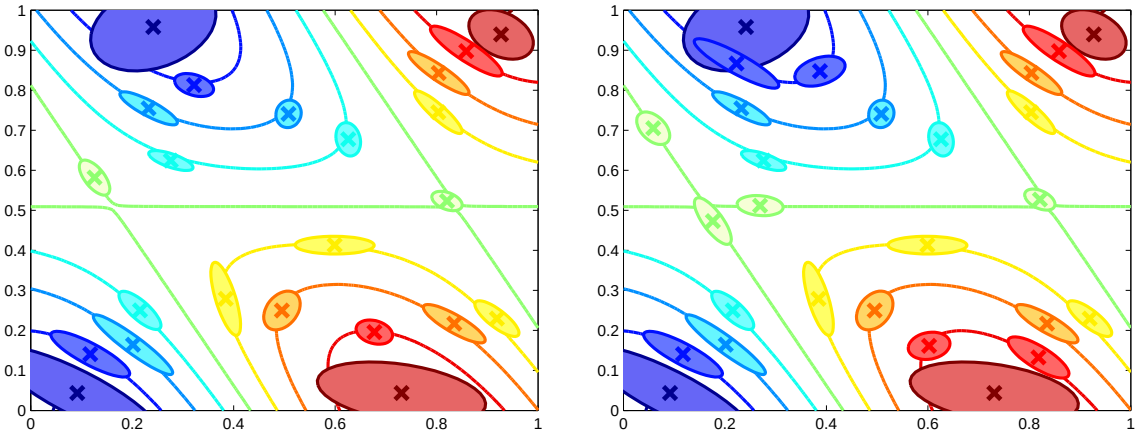
Using a higher value of α_{multi} has the desired consequence of providing a finer sampling of the inverse solution space; however, as a consequence, some spurious inverse solutions may then appear, corresponding to inverse predictions provided by experts that, despite the fact that their output space region of influence does not conveniently support the output query location, have nevertheless some influence on the final set of inverse predictions, due to a high value of α_{multi}.

Inverse predictions found by the IMLE model, for the different values of α_{multi} presented in Table 4.6, are shown in Figure 4.21, superimposed over the contours describing

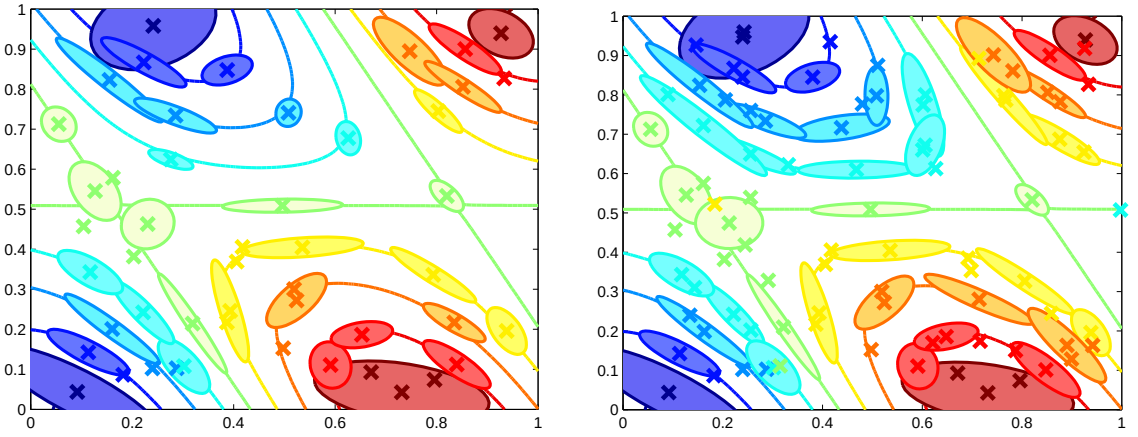
α_{multi}	$\boldsymbol{y}_q$								
	-0.8	-0.6	-0.4	-0.2	0.0	0.2	0.4	0.6	0.8
0.2	2	2	3	3	2	4	3	2	2
0.8	2	3	3	3	4	4	3	3	2
0.95	2	4	6	5	9	8	6	5	4
0.99	3	7	12	16	13	16	11	9	4

Table 4.6: Number of inverse solutions found by the IMLE algorithm for the $\mathbb{R}^2 \mapsto \mathbb{R}^1$ PUMA kinematic function, for different values of α_{multi}.

the true inverse relation. Also shown in this figure are the predicted variances provided by IMLE, represented by the ellipsoids centred at each inverse prediction. Such predic-



(a) $\alpha_{multi} = 0.2$ (b) $\alpha_{multi} = 0.8$



(c) $\alpha_{multi} = 0.95$ (d) $\alpha_{multi} = 0.99$

Figure 4.21: Inverse prediction provided by the IMLE model, for different values of α_{multi}.

tions and variances constitute a Gaussian mixture model describing the inverse solution space, and additionally each Gaussian has a weight given by $w^{(k)}$, as defined in (3.27). Figure 4.21 shows a remarkable agreement between such Gaussian mixture and the true inverse solution space: as expected, there are more inverse predictions populating the

inverse solution space as the value of α_{multi} is increased. As discussed above, an undesired side effect in this situation is the lack of accuracy of some of the inverse predictions found: in particular, this can be observed in Figure 4.21d. Fortunately, such spurious predictions can be easily detected, as they correspond to predictions with a low value of the associated weight. Also, as a consequence of such low value of $w^{(k)}$, their corresponding prediction variances become increasingly larger, signalling an increase of the uncertainty on such prediction values. One can easily remove these low confidence inverse predictions, for instance, by simply imposing a threshold on the weights $w^{(k)}$, under which a multi-valued prediction is simply discarded. This was done in Figure 4.21, where each inverse prediction whose weight was less than 5% was considered unreliable: these predictions are still marked in Figures 4.21a–4.21d, but they lack, in the figures, the ellipsoids representing their variance.

4.4 Active Learning

The last experiments presented in this chapter have the primary purpose of shedding some light to the active learning strategies devised in Section 3.4 for the IMLE model. It is important to understand why is this proposed active learning scheme preferable over other sampling schemes: one could legitimately ask if a simpler sampling scheme, based for instance in picking training points from locations with a high level of output uncertainty, would not suffice. To show how could such sampling policy fail, a simple univariate toy example, represented by the relation

$$y = \begin{cases} 2x & \text{if } x \leq 0.5 \\ -2x + 2 & \text{if } x > 0.5 \end{cases} \tag{4.1}$$

will be used: some training data generated from this relation is depicted in Figure 4.22. Note that the left half of the function has a stronger noise level, with a standard deviation five times larger than the right half of the function. This heteroscedasticity has a profound impact on active learning schemes based on variance maximization, as the tendency to explore regions of the input space with a high level of output uncertainty may definitely compromise a full coverage of this same space. Figure 4.23a shows the evolution of this quantity as the training of the IMLE model is undertaken. In this particular example, data was acquired by conducting consecutive sweeps of the input range, from left to right. After only 10 training points are presented to the IMLE model there is a large uncertainty in the right half of the input space, as the training data seen so far was only originated from the left half of the input domain. As training goes on the output variance estimate becomes more accurate, and after $N = 5\,000$ training points the distinct nature of output noise in the two halves of the input domain can be clearly devised.

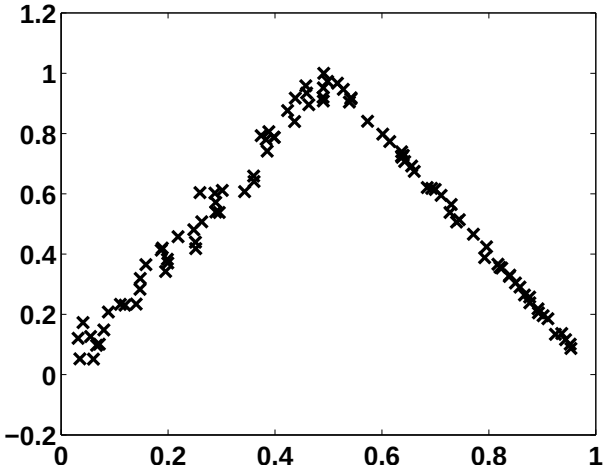
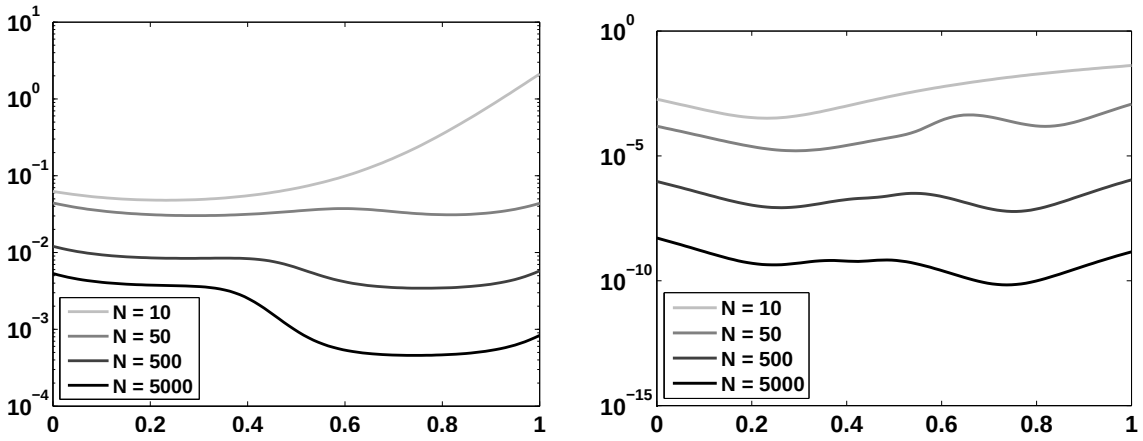


Figure 4.22: Some training data generated from the target function described in (4.1). Note the higher level of output noise in the first half of the function.



(a) Output variance.

(b) Expected prediction variance reduction.

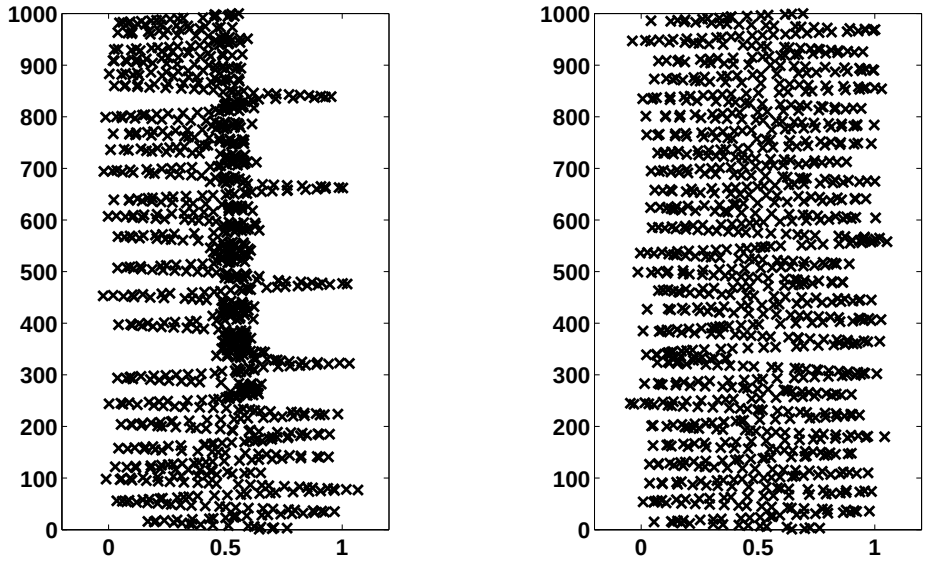
Figure 4.23: Evolution of output variance and expected prediction variance reduction as a function of $\boldsymbol{x}$, at different IMLE model training stages.

A different picture can be seen in Figure 4.23b, representing the expected prediction variance reduction if a particular value of $\boldsymbol{x}$ is picked to collect a training sample to be presented to the IMLE model, as given by equation (3.34). Even if a slight preference appears to be given to the left half of the input domain after a training phase of $N = 5\,000$ points, this is only caused by the presence of a higher level of noise in that region: acquisition of a few more samples on the left half of the input range would in principle lower the value of the expected prediction variance reduction in that region. This figure also shows an important characteristic of equation (3.34): it tends to give higher preference to the input space boundaries and to regions where the most dominant linear expert changes, according to the input regions defined by $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\nu}_j$.

The two different active learning criteria described above were used, in the next experiment, to influence the data acquisition process during training, using a pool-based

selective sampling scheme. Data was picked by performing straight line trajectories in the input space, by taking a succession of input target points to reach. These target points were chosen according to the variance and the expected prediction variance reduction criteria: every time a new input target point was needed a set of 20 random input points was generated within the input range, and the one maximizing the considered criterion was picked as the new target point.

Figure 4.24 shows the input trajectories obtained using such active learning scheme, for the two criteria presented above. It comes immediately into attention that actively



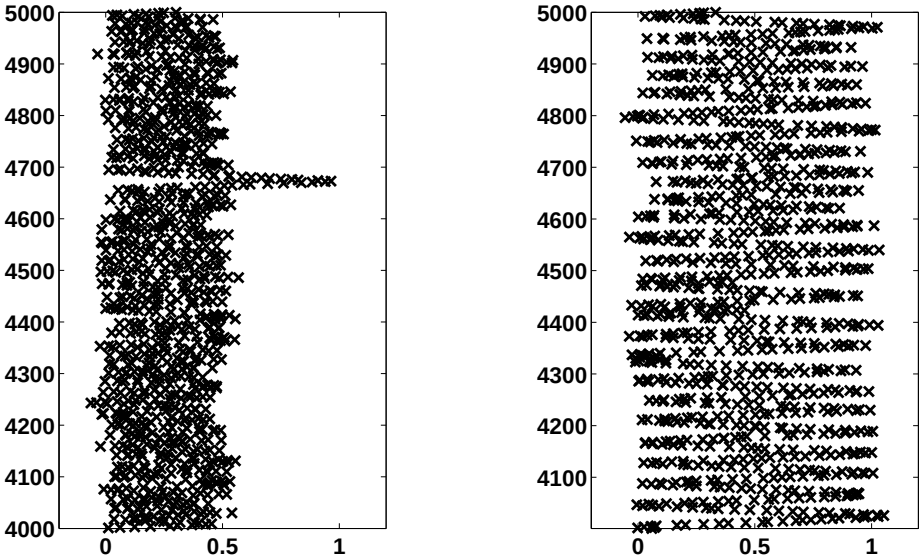
(a) Output variance.

(b) Expected prediction variance reduction.

Figure 4.24: Input space trajectories as training is performed, for two different exploration policies — initial stage of the learning process. The y coordinates denotes the training iteration.

picking training points that maximize the output variance has the undesired consequence of getting the exploration stuck in the particular region of the input space corresponding to a higher noise level. This phenomenon even becomes more accentuated as the learning proceeds, as can be visualized in Figure 4.25: in the final training phase, after 4 000 data points were presented to the IMLE model, an active learning scheme based on output variance maximization keeps exploring the input region corresponding to the higher output noise level, completely neglecting the remaining regions of this space. Picking training points that maximize the expected prediction variance reduction, on the other hand, produces a more uniform exploration of the input space.

Figure 4.26 may contribute to better understand the exploration process underlying these different active learning approaches: it depicts the percentage of time spent in a particular input region, after a training phase of 5 000 points, by considering a partition of the input range in a set of 10 equally spaced bins. The active learning scheme introduced in



(a) Output variance.

(b) Expected prediction variance reduction.

Figure 4.25: Input space trajectories as training is performed, for two different exploration policies — final stage of the learning process.

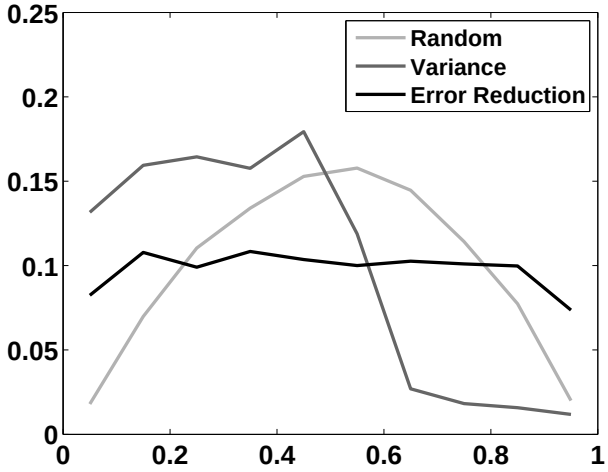
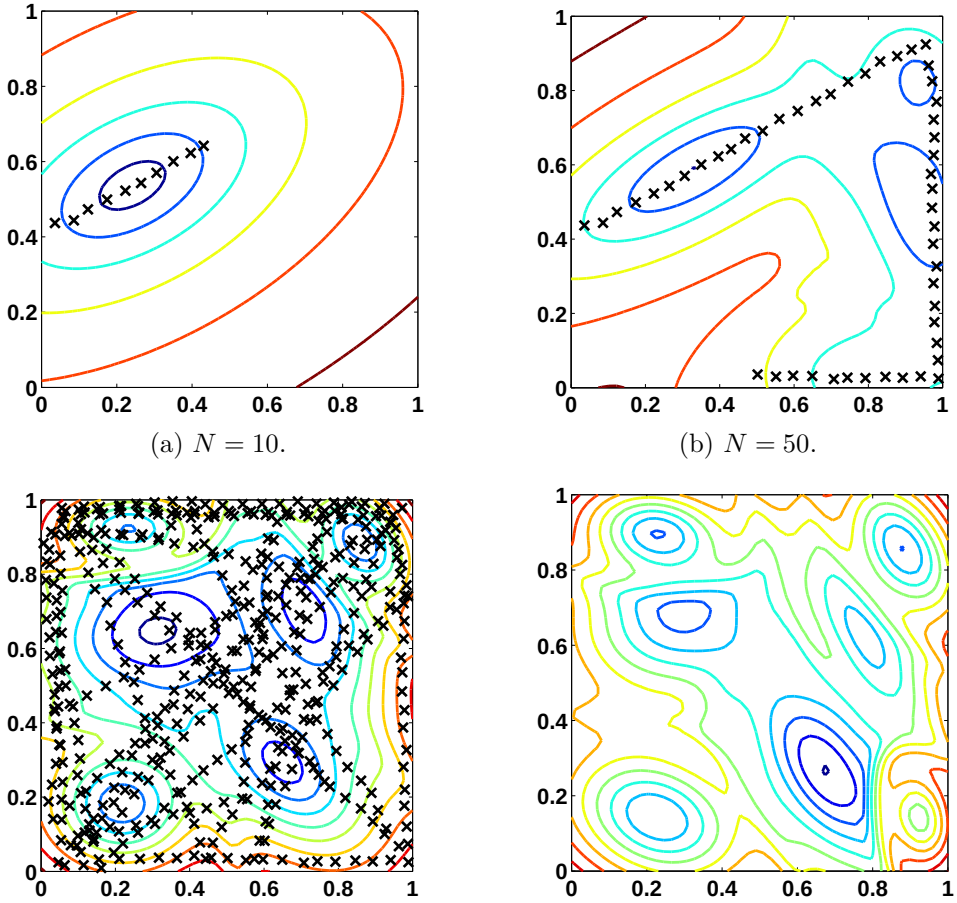


Figure 4.26: Percentage of time spent in a particular input region.

Section 3.4 effectively produces an almost uniform sampling of the input range, while the criterion that picks training points based on their expected output variance concentrates its efforts in the exploration of the left half of the input range, corresponding to a higher noise level. Another result concerning random exploration of the input space is also shown in this figure: due to its random nature, the input regions close to the input space boundaries are less visited.

Next, the PUMA 560 testbed is used to illustrate some characteristics of the IMLE model active learning mechanism. For visualization purposes, the same low-dimensional map from $\mathbb{R}^2$ to $\mathbb{R}^1$ considered in Section 4.3 is also used in the following experiments.

Figure 4.27 shows how the input space is sampled when the same pool-based sampling



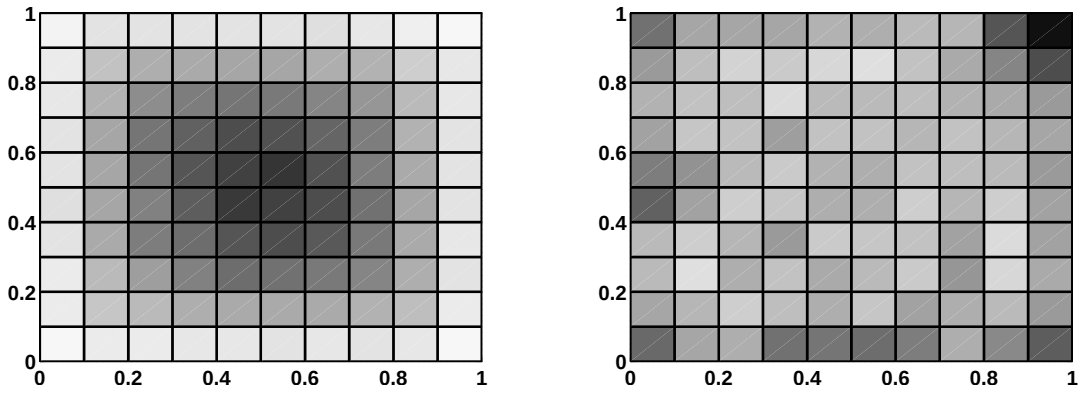
(a) $N = 10$. (b) $N = 50$.

(c) $N = 500$. (d) $N = 5\,000$.

Figure 4.27: The evolution of the variance reduction measure as the IMLE model is trained. Contours indicate equal expected variance reduction as a result of sampling a new training point at a specific location. Also depicted are the training points acquired at each situation (exception made to Figure 4.27d, which would lead to unnecessary figure clutter.

scheme described in the previous experiment is used to generate the input space acquisition trajectory in an online way. It is worth of notice that such active learning mechanism will usually lead to a more intensive exploration of the input space boundaries. This behaviour is caused by the higher variance reduction expected if a new training point is sampled near the input space limits, as can be observed in Figure 4.27. As a consequence, the input space is sampled in a more uniform way: random sampling, on the other hand, produces more frequent visits to interior regions of the input space. These behaviours are depicted in Figure 4.28.

Does this active learning scheme improve the convergence of the learning process to low prediction errors? Figure 4.29 shows that an active learning sampling scheme is beneficial, specially in an early learning stage, as compared to random trajectory generation. In this figure the random sampling is compared to the pool-based active learning scheme



(a) Random sampling. (b) Active learning.

Figure 4.28: 2D grid representing the percentage of time spent in a particular input location. Darker squares denote a higher percentage of training points coming from these regions.

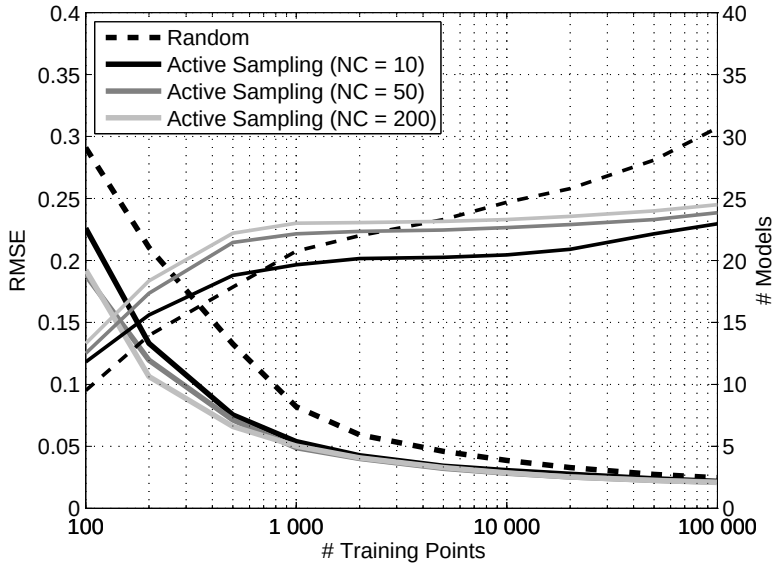


Figure 4.29: Learning curves for the PUMA 560 active learning example ($\mathbb{R}^2 \mapsto \mathbb{R}^1$), showing the evolution of the RMSE and the number of allocated experts for 4 different sampling schemes. Results shown are the average over 20 independent trials.

provided by the IMLE model, for 3 different sizes of such pool, corresponding to 10, 50 and 200 random input candidates, represented by variable NC in the figure. It can be seen that active sampling initially allocates more experts than the random sampling strategy, and that, as a consequence, the RMSE drops to significantly lower values than the ones corresponding to random sampling. However, as the learning process goes on, eventually the random sampling creates enough linear experts to produce values of RMSE comparable to the ones corresponding to the active learning scheme.

Similar learning curves can be obtained, for the PUMA 560 testbed, if a higher number of input and output dimensions is considered: learning the $\mathbb{R}^3 \mapsto \mathbb{R}^3$ sensorimotor map,

first introduced in Section 4.3, has the results depicted in Figure 4.30, where, after a training phase consisting of 100 000 sample points, the RMSE corresponding to the active learning schemes is approximately half the random sampling RMSE.

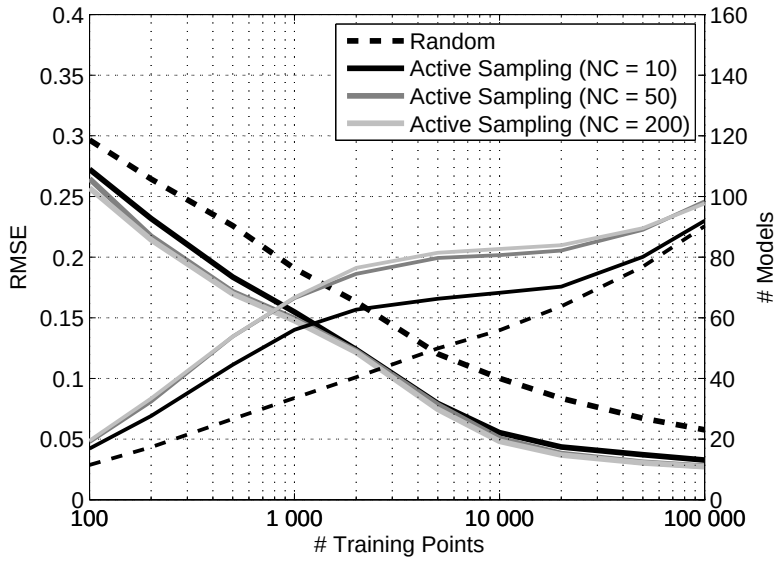


Figure 4.30: Learning curves for the PUMA 560 active learning example ($\mathbb{R}^3 \mapsto \mathbb{R}^3$), showing the evolution of the RMSE and the number of allocated experts for 4 different sampling schemes. Results shown are the average over 20 independent trials.

However, pool-based sampling becomes less practical as the input dimension increases, since the size of the pool must increase accordingly, thus requiring more processing time to obtain the next candidate point to sample. In this situation, the gradient of the variance reduction criterion, as defined in equation (3.35) and Appendix C, can be used to locally find the input direction along which the next training point should be sampled. There are however some technical details with such approach: the predisposition of this active learning scheme to sample the boundary of the input space will inevitably drive the acquisition process outside the allowable input range, and as a consequence some sort of penalty on the distance to the input space boundary must be introduced in the trajectory generation mechanism to prevent sampling outside this range. Also, there are no guarantees that the achieved trajectories will not enter some kind of deadlocks, where the same region of the input space is sampled over and over again, as the dynamics of the learning and active sampling processes are highly complex. This is a topic that surely deserves additional investigation, but that will not be further addressed in this dissertation, as it goes a bit beyond the scope of this thesis.

4.5 Discussion

The IMLE probabilistic model can be incrementally trained, and predictions can be provided during the training process, in a fully online fashion. All the experiments described in this chapter served the primary purpose of validating the IMLE model performance in different learning situations: it was shown first that the IMLE model is a competitive learning algorithm for traditional single-valued forward prediction, achieving a prediction accuracy comparable to several state-of-the-art function approximation algorithms. After that the IMLE model was tested in several different multi-valued experimental settings that demonstrated its good performance in such domains: these are regression problems that are not conveniently dealt with by most supervised learning algorithms. This multi-valued prediction capability also allows the generation of inverse predictions from the same IMLE model that is used for forward prediction, and this was demonstrated in this chapter too. Finally, it was shown how the variance reduction estimate provided by the same IMLE model can be used to conduct the data acquisition procedure in order to accelerate the convergence of the learning algorithm, under an active learning context. The following chapter will show how to make use of the versatility of the IMLE probabilistic model to control anthropomorphic robotic limbs, by learning their corresponding sensorimotor maps and using these internal models to provide the information required by their control processes.

Chapter 5

Sensorimotor Coordination

This chapter illustrates the application of the IMLE model to online learning and control of robotic limbs, putting a special emphasis on situations where traditional learning algorithms may fail, due to the lack of ability to deal with a multi-valued nature of the training data. Section 5.1 introduces the model based control problem, providing a short review of some standard approaches to this matter. Section 5.2 shows how the IMLE model can be used to control a robotic arm using either open or closed-loop control schemes, making use of its ability to provide both forward and inverse predictions from the same learned model. Finally, Section 5.3 illustrates the use of a single IMLE model to learn a robotic sensorimotor relation where a hidden change of context can occur without any kind of notification to the learning and control algorithms — in this particular case, the change of the kinematics structure by inclusion of a tool of unknown geometry.

5.1 Robotic Model Based Control

The major question that arises in model based control of robotic mechanisms is how to choose, based on a previously acquired sensorimotor model, the actuation values that will drive the robot state to a particular desired value. In the following discussion, the control of the end-effector position, expressed as a vector $\boldsymbol{x}$ comprising the end-effector coordinates in a known referential will be considered. Note, however, that the techniques described in this section can be readily applied to control different components of the robot state, other than the end-effector position; in particular, orientation of the same end-effector can also be considered by an appropriate coding of this orientation in the task space vector $\boldsymbol{x}$.

At a kinematic level, the relation between joint positions $\boldsymbol{q}$ and end-effector task space position $\boldsymbol{x}$ is defined by the *kinematics function*

$$\boldsymbol{x} = \boldsymbol{f}_{kin}(\boldsymbol{q}) \ . \tag{5.1}$$

As stated before, when the dimension of the joint space is equal to the dimension of the task space, the kinematics forward relation of a serial robot is a proper, single-valued function, while the inverse relation is usually multi-valued — parallel robot exhibit the opposite behaviour. If the dimension of the joint space is greater than the dimension of the task space the robot is said to be redundant. This means that, for fixed task space positions, the robot still can move along certain directions of the joint space, along a constraint surface, known as the *self-motion* manifold. This redundancy allows the robot to use the extra degrees of freedom, for instance, to avoid obstacles or to minimize the energy spent to actuate its joints.

Another important concept is the notion of singularities, regions of the joint space where the robot loses some degrees of freedom for its movements. Putting the robot near singular configurations is highly undesirable, as the joint velocities needed to move the robot in some task space directions become unbounded. Formally, singularities occur in a particular joint configuration when the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$, whose ijth element is given by $\frac{\partial f^i_{kin}(\boldsymbol{q})}{\partial \boldsymbol{q}^j}$, is rank deficient. Redundant robots can use the extra degrees of freedom to avoid joint limits and singularities.

While the kinematic function describes a static relation that describes the transformation operated by a robot from joint to task space, the *dynamics* equation relates joints accelerations $\ddot{\boldsymbol{q}}$ to corresponding torques $\boldsymbol{\tau}$. Contrary to the kinematics function, this is a well-posed one-to-one relation between vectorial spaces of the same dimension, given by the *inverse dynamics* equation

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) \ , \tag{5.2}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is the mass or inertia matrix, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ aggregates the centrifugal and Coriolis forces and $\boldsymbol{G}(\boldsymbol{q})$ denotes the gravity term. As can be seen in the above equation, this relation depends on the robot current values of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, that can be understood as the context of the dynamics equation. Since the inertia matrix is positive definite, the above equation can be easily inverted to obtain

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}(\boldsymbol{q})^{-1}(\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q})) \ .$$

The following sections will describe how these kinematic and dynamic sensorimotor maps can be exploited to make the robot follow a desired task space trajectory.

5.1.1 Static Control

Given a desired end-effector $\boldsymbol{x}_d(t)$, robotic control at a pure kinematic level uses the kinematic function (5.1) to obtain a joint vector $\boldsymbol{q}$ that will place the end-effector at the desired position. If an inverse model $\boldsymbol{q} = \boldsymbol{f}^{-1}_{kin}(\boldsymbol{x})$ is available, either an analytical

or a learned one, the choice of the joint vector $\boldsymbol{q}$ is generally guided by some kind of minimization of a cost criterion, such as the manipulability index, the distance to joint limits, the distance to obstacles or the proximity of singularities (Baillieul and Martin, 1990), as a kinematic inverse model for a redundant mechanism does not provide a single solution for the control problem.

Forward kinematic models of serial mechanisms are normally much easier to obtain: when such forward model is available a solution for the static control problem can be found using numerical optimization techniques that also deal with the redundancy resolution issue. One of the simplest solutions for this problem is the use of a Newton–Raphson algorithm to solve the equation $\boldsymbol{f}_{kin}(\boldsymbol{q}) - \boldsymbol{x} = \boldsymbol{0}$ (Peiper, 1968). Global methods find a solution that minimizes a desired cost function by looking at the global joints and task space paths: in (Nakamura and Hanafusa, 1987) the globally optimal redundancy control problem is solved strictly by using Pontryagin's maximum principle, while Kazerounian and Wang (1988) integrate joint velocities found using the differential kinematics equations to produce the joint angles corresponding to the desired solution. Local methods, on the other hand, only look at the instantaneous cost: this makes the optimization problem simpler and easier to solve in real-time, but contrary to global methods they are prone to singularities and less than optimal global solutions (Baillieul and Martin, 1990). Local methods typically make use of the differential kinematic equations, described in detail in Section 5.1.2, to calculate a sequence of joint values that will drive the end-effector to the desired position (Shamir and Yomdin, 1988). The *cyclic coordinate descent method* of Wang and Chen (1991) can also be viewed as a local method, where, starting from an initial robot configuration, each joint value is updated in turn, trying to minimize the distance between the end-effector position $\boldsymbol{x}$ and the desired position $\boldsymbol{x}_d$. A comparison of global and local methods for numerical inverse kinematics is presented in the works of Suh and Hollerbach (1987), Martin, Baillieul, and Hollerbach (1989) and Baillieul and Martin (1990), while a comparison of real-time applications of some of these numerical methods is discussed by Tolani, Goswami, and Badler (2000).

5.1.2 Velocity Control

The aim of a velocity based control scheme, also known as *resolved motion rate control* (RMRC) is to find joint velocities $\dot{\boldsymbol{q}}$ that will generate some desired task space positions $\boldsymbol{x}_d$ and velocities $\dot{\boldsymbol{x}}_d$. The reference task space velocity is usually given by

$$\dot{\boldsymbol{x}}_r = \dot{\boldsymbol{x}}_d + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x}) \;, \tag{5.3}$$

where $\boldsymbol{x}$ is the current end-effector position and $\boldsymbol{K}_p$ is a gain diagonal matrix. While omitted for readability, in this chapter all the above variables depend on a particular value of time, *i.e.*, $\boldsymbol{q}$, $\boldsymbol{x}$, $\boldsymbol{x}_r$ and $\boldsymbol{x}_d$, among others, should be read as $\boldsymbol{q}(t)$, $\boldsymbol{x}(t)$, $\boldsymbol{x}_r(t)$ and

$\boldsymbol{x}_d(t)$, respectively.

Differentiation of equation (5.1) results in the well known relation

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \ , \tag{5.4}$$

where the matrix $\boldsymbol{J}(\boldsymbol{q})$ is the *Jacobian* matrix that, at a given joint configuration $\boldsymbol{q}$, relates task space velocities and joint velocities. If the Jacobian matrix is invertible, this means that the task is not redundant and the robot is not in a singular configuration; in this case, the solution for the control problem is straightforward and is given by

$$\dot{\boldsymbol{q}}_a = \boldsymbol{J}^{-1}\dot{\boldsymbol{x}}_r \ , \tag{5.5}$$

where for notational convenience the dependence of the Jacobian on the current joint position vector $\boldsymbol{q}$ has been dropped.

Equation (5.4) admits an infinite number of solutions $\dot{\boldsymbol{q}}_a$, for a desired value of $\dot{\boldsymbol{x}}$, if the robot is redundant, with a joint space dimensionality greater than the task space dimensionality, due to the existence of a null space of the transformation $\boldsymbol{J}$, defined by the set of joint values that satisfy the relation

$$\boldsymbol{J}\dot{\boldsymbol{q}} = \boldsymbol{0} \ .$$

As a consequence, any vector $\dot{\boldsymbol{q}}_N$ belonging to the null space of $\boldsymbol{J}$ can be added to an arbitrary velocity vector $\dot{\boldsymbol{q}}$ without changing the corresponding task velocity, since

$$\boldsymbol{J}(\dot{\boldsymbol{q}} + \dot{\boldsymbol{q}}_N) = \boldsymbol{J}\dot{\boldsymbol{q}} + \boldsymbol{J}\dot{\boldsymbol{q}}_N = \boldsymbol{J}\dot{\boldsymbol{q}} + \boldsymbol{0} = \boldsymbol{J}\dot{\boldsymbol{q}} \ .$$

Also, when the robot is redundant it is no longer possible to define the inverse $\boldsymbol{J}^{-1}$. One solution is to replace this inverse by a generalized inverse $\boldsymbol{J}^{\dagger}$: the relation (5.5) then becomes

$$\dot{\boldsymbol{q}}_a = \boldsymbol{J}^{\dagger}\dot{\boldsymbol{x}}_r \ .$$

Note that the generalized inverse may not be unique: this can be remedied if the *Moore-Penrose pseudoinverse* is chosen, a special generalized inverse that is unique and that exists for every matrix, irrespectively of its rank.

A more general approach to velocity based control is to consider the actuation joint velocity $\dot{\boldsymbol{q}}_a$ to be given by the solution of the following minimization problem,

$$\arg\min_{\dot{\boldsymbol{q}}} \quad (\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_r)^T \boldsymbol{W}_x (\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_r) + L(\boldsymbol{q}, \dot{\boldsymbol{q}}) \ , \tag{5.6}$$

where $\boldsymbol{W}_x$ is a diagonal matrix of weights and $L(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a general loss function; setting

this function to

$$L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r)^T \boldsymbol{W}_q (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r) + \lambda H(\boldsymbol{q} + \tau \dot{\boldsymbol{q}}) \ ,$$

where $\dot{\boldsymbol{q}}_r$ is some reference joint space velocity, $\boldsymbol{W}_q$ is another diagonal matrix of weights, $H(\boldsymbol{q})$ is some penalty on the current joints positions, that tries to drive them to a given rest position, τ is an infinitesimal that goes to 0 and λ is a scalar weight, allows the derivation of some well-known velocity controllers, as will be shown in the following discussion. With such loss function, the function to be minimized has a strong resemblance to the coherence enforcement of Hersch (2009), where the control of movement in both task and joint space is achieved using a Vector Integration To Endpoint (VITE) dynamical system (Bullock and Grossberg, 1988). Setting the derivative of (5.6) to $\mathbf{0}$ and solving for $\dot{\boldsymbol{q}}$ results in

$$\dot{\boldsymbol{q}}_a = \left(\dot{\boldsymbol{q}}_r - \lambda \boldsymbol{W}_q^{-1} \nabla H\right) + \boldsymbol{W}_q^{-1} \boldsymbol{J}^T \left(\boldsymbol{J} \boldsymbol{W}_q^{-1} \boldsymbol{J}^T + \boldsymbol{W}_x^{-1}\right)^{-1} \left[\dot{\boldsymbol{x}}_r - \boldsymbol{J}\left(\dot{\boldsymbol{q}}_r - \lambda \boldsymbol{W}_q^{-1} \nabla H\right)\right] \ , \tag{5.7}$$

where ∇H is the gradient of H at the current joint position.

Least Norm Method

Setting $\lambda = 0$, $\boldsymbol{W}_q = \mathbf{I}$, $\dot{\boldsymbol{q}}_r = \mathbf{0}$ and $\boldsymbol{W}_x = w_x \mathbf{I}$, with $w_x \to \infty$, results in

$$\dot{\boldsymbol{q}}_a = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T)^{-1} \dot{\boldsymbol{x}}_r = \boldsymbol{J}^\dagger \dot{\boldsymbol{x}}_r \ ,$$

where $\boldsymbol{J}^\dagger = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T)^{-1}$ is the Moore-Penrose pseudoinverse of $\boldsymbol{J}$; this is one of earliest solutions to velocity control problem, proposed by Whitney (1969), and known as the least norm method. In this case, the cost is given by $L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}^T \dot{\boldsymbol{q}} = |\dot{\boldsymbol{q}}|^2$, and the fact that the weights in $\boldsymbol{W}_x$ go to infinity makes this problem equivalent to minimizing the norm of $\dot{\boldsymbol{q}}$, under the constraint given by equation (5.4): hence the "least norm" designation.

Weighted Least Norm Method

If a general weight matrix $\boldsymbol{W}_q$ is used instead, that assigns different importance to each of the joints, the solution in equation (5.7) is simplified to

$$\dot{\boldsymbol{q}}_a = \boldsymbol{W}_q^{-1} \boldsymbol{J}^T (\boldsymbol{J} \boldsymbol{W}_q^{-1} \boldsymbol{J}^T)^{-1} \dot{\boldsymbol{x}}_r \ .$$

This is a weighted version of the least norm solution, first suggested by Whitney (1972), where high values of the elements of the diagonal of $\boldsymbol{W}_q$ prevent fast movements of the corresponding joints.

Damped Least Squares Method

The least norm method (and its weighted variant) has numerical problems when $(\boldsymbol{J}\boldsymbol{J}^T)$ becomes ill-conditioned: this happens near singularities, where the robot cannot move in the desired direction due to its kinematic properties. This undesired phenomenon can be circumvented if the constraint (5.4) is relaxed, and this can be achieved by setting a finite value for the weights in $\boldsymbol{W}_x$. Making $\boldsymbol{W}_x = \alpha^{-1}\mathbf{I}$, while keeping $\lambda = 0$ and $\boldsymbol{W}_q = \mathbf{I}$, results in the *damped least squares* control scheme, proposed simultaneously by Nakamura and Hanafusa (1986) and Wampler (1986), with solution

$$\dot{\boldsymbol{q}}_a = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T + \alpha\mathbf{I})^{-1}\dot{\boldsymbol{x}}_r \ .$$

The existence of $\alpha\mathbf{I}$ makes the matrix always invertible, but as a consequence the solution no longer exactly satisfies (5.4). A weighted variant of this solution can be readily obtained by considering an arbitrary positive diagonal matrix $\boldsymbol{W}_q$, leading to

$$\dot{\boldsymbol{q}}_a = \boldsymbol{W}_q^{-1}\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{W}_q^{-1}\boldsymbol{J}^T + \alpha\mathbf{I})^{-1}\dot{\boldsymbol{x}}_r \ .$$

Gradient Projection Method

Finally, avoiding joint vector values too close to the corresponding joint physical limits can be enforced by defining a function $H(\boldsymbol{q})$ that imposes higher penalties in regions close to the joint space boundary. This method was first proposed by Liegeois (1977) and can be obtained from equation (5.7) by considering $\boldsymbol{W}_q = \mathbf{I}$, $\lambda > 0$ and making $\boldsymbol{W}_x$ go again to infinity, resulting in

$$\begin{aligned}\dot{\boldsymbol{q}}_a &= \boldsymbol{J}^T\left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1}\dot{\boldsymbol{x}}_r - \lambda\left(\mathbf{I} - \boldsymbol{J}^T\left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1}\boldsymbol{J}\right)\nabla H \\ &= \boldsymbol{J}^\dagger\boldsymbol{x}_r - \lambda\left(\mathbf{I} - \boldsymbol{J}^\dagger\boldsymbol{J}\right)\nabla H \ .\end{aligned} \tag{5.8}$$

Since $\boldsymbol{W}_x \to \infty$ enforces a perfect task constraint, the joints limits avoidance is performed in the null space of the Jacobian transformation, via the null space projector $(\mathbf{I} - \boldsymbol{J}^\dagger\boldsymbol{J})$.

Practical Considerations

All these velocity based control schemes only require the Jacobian of the forward kinematics relation, at every possible joint position $\boldsymbol{q}$, to be able to operate normally. Any sensorimotor motor learning algorithm that is able to provide an estimate of this Jacobian can in principle be used to provide the necessary information needed to control a robot at the velocity level, as done, for instance, by Salaün, Padois, and Sigaud (2010), using the LWPR learning algorithm (Vijayakumar, D'Souza, and Schaal, 2005). Another approach

is to learn the map $(\boldsymbol{q}, \dot{\boldsymbol{x}}_r) \mapsto \dot{\boldsymbol{q}}_a$ directly (D'Souza, Vijayakumar, and Schaal, 2001): however, due to the redundancy of the kinematic function, this problem is ill-posed, with a continuum of possible solutions $\dot{\boldsymbol{q}}_a$ that generate the same task space velocity $\dot{\boldsymbol{x}}_r$ under the same context $\boldsymbol{q}$. While it can be shown that, in the vicinity of a particular joint value $\boldsymbol{q}$, any solution obtained from averaging the training samples $\dot{\boldsymbol{x}}_r^i$ will result in a solution $\hat{\dot{\boldsymbol{q}}}_a$ that will produce the desired task space velocity $\dot{\boldsymbol{x}}_r$ (D'Souza, Vijayakumar, and Schaal, 2001; Peters and Schaal, 2008), globally these solutions may not provide a coherent controller due to the nonlinearity of the forward model. As a consequence, the work of D'Souza, Vijayakumar, and Schaal (2001) biases the learning algorithm by training it with carefully chosen data that will produce only an inverse solution, an approach that certainly lacks the generality required for autonomous learning.

Desired velocity commands $\dot{\boldsymbol{q}}_a$ can be directly fed to low-level controllers, responsible for accurate tracking of such reference commands. This approach, however, does not take into consideration the complex dynamics of a robot manipulator, ignoring the interactions between different links. The dynamics equation (5.2) can be used to calculate the actuation torques that will produce a desired joint vector acceleration $\ddot{\boldsymbol{q}}_a$; such acceleration is not provided by any of these velocity control schemes, but it may be obtained from numerical differentiation of $\dot{\boldsymbol{q}}_a$, making

$$\ddot{\boldsymbol{q}}_a \approx \frac{\dot{\boldsymbol{q}}_a - \dot{\boldsymbol{q}}_a^{old}}{T} ,$$

where T is the sampling period and $\dot{\boldsymbol{q}}_a^{old}$ is the desired joints velocity calculated in the previous control iteration. This approach is taken, for instance, by Nakanishi, Cory, et al. (2008) and Salaün, Padois, and Sigaud (2010). Note, however, that these approaches also need a learned model for the robot dynamics.

Perhaps the main drawback of velocity control methods is the fact that, by operating at a velocity level, they do not take into account desired acceleration references $\ddot{\boldsymbol{x}}_d$, as can be seen in expression (5.3). The lack of such reference for the task space acceleration many times leads to highly overdamped, non compliant controlled systems (Nakanishi, Cory, et al., 2008). Acceleration based control methods, presented in the next section, are a natural solution to this problem if desired acceleration profiles are to be considered.

5.1.3 Acceleration Control

Control methods at the acceleration level compute a joint acceleration vector $\ddot{\boldsymbol{q}}$ from a reference task space acceleration $\ddot{\boldsymbol{x}}_r$. This reference acceleration can be calculated, given desired temporal profiles for the task space position, velocity and acceleration, represented respectively by $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d$ and $\ddot{\boldsymbol{x}}_d$, by

$$\ddot{\boldsymbol{x}}_r = \ddot{\boldsymbol{x}}_d + \boldsymbol{K}_v(\dot{\boldsymbol{x}}_d - \dot{\boldsymbol{x}}) + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x}) . \tag{5.9}$$

As before, $\boldsymbol{x}$ denotes the current end-effector position and $\boldsymbol{K}_p$ is a matrix gain over the current position error. Vector $\dot{\boldsymbol{x}}$ is the end-effector velocity and $\boldsymbol{K}_v$ is a matrix gain over the velocity error. Equation (5.9) turns the tracking problem into a reference attractor in the task space, with gains $\boldsymbol{K}_v$ and $\boldsymbol{K}_p$ (Peters, Mistry, et al., 2005; Udwadia, 2003). This reference acceleration vector can be related to a corresponding joint space acceleration by the following equation,

$$\ddot{\boldsymbol{x}} = \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\ddot{\boldsymbol{q}} \,, \tag{5.10}$$

obtained from the time differentiation of (5.4), where $\dot{\boldsymbol{J}}$ denotes the time derivative of the Jacobian.

Acceleration control suffers from the same redundancy resolution issues that arise in velocity control, and obtaining an acceleration vector $\ddot{\boldsymbol{q}}$ that fulfils the task space acceleration requirements can follow the same approach. Thus, such solution $\ddot{\boldsymbol{q}}_a$ can be viewed as the vector that minimizes the following cost,

$$\arg\min_{\ddot{\boldsymbol{q}}} \quad (\boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \ddot{\boldsymbol{x}}_r)^T \boldsymbol{W}_x (\boldsymbol{J}\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \ddot{\boldsymbol{x}}_r) + L(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \,. \tag{5.11}$$

Setting $L(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = (\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_0)^T \boldsymbol{W}_q (\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_0)$, for instance, where $\ddot{\boldsymbol{q}}_0$ is the acceleration required to perform some secondary task, and making $\boldsymbol{W}_x \to \infty$ results in

$$\begin{aligned} \ddot{\boldsymbol{q}}_a &= \boldsymbol{W}_q^{-1} \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{W}_q^{-1}\boldsymbol{J}^T)^{-1} (\ddot{\boldsymbol{x}}_r - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) + \left[\mathbf{I} - \boldsymbol{W}_q^{-1}\boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{W}_q^{-1}\boldsymbol{J}^T)^{-1}\boldsymbol{J}\right] \ddot{\boldsymbol{q}}_0 \\ &= \boldsymbol{W}_q^{-\frac{1}{2}} \left(\boldsymbol{J}\boldsymbol{W}_q^{-\frac{1}{2}}\right)^{\dagger} (\ddot{\boldsymbol{x}}_r - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) + \boldsymbol{W}_q^{-\frac{1}{2}} \left[\mathbf{I} - \left(\boldsymbol{J}\boldsymbol{W}_q^{-\frac{1}{2}}\right)^{\dagger} \left(\boldsymbol{J}\boldsymbol{W}_q^{-\frac{1}{2}}\right)\right] \boldsymbol{W}_q^{\frac{1}{2}} \ddot{\boldsymbol{q}}_0 \,, \end{aligned} \tag{5.12}$$

which can be seen as the enforcement of the task $\ddot{\boldsymbol{x}}_r$ while maintaining the joints accelerations $\ddot{\boldsymbol{q}}_a$ as close to $\ddot{\boldsymbol{q}}_0$ as possible — as can be seen in equation (5.12), the secondary task acceleration is projected into the null space of the main task, thus guaranteeing that the resulting acceleration vector $\ddot{\boldsymbol{q}}_a$ still fulfils the desired task space acceleration $\ddot{\boldsymbol{x}}_r$.

Reference accelerations $\ddot{\boldsymbol{q}}_a$ have a simple translation to corresponding joint torques, given by equation (5.2), and it is no surprise that a resolved acceleration controller is almost always used in conjunction with an inverse dynamics model, leading to a torque vector given by

$$\boldsymbol{\tau} = \boldsymbol{M}\ddot{\boldsymbol{q}}_a + \boldsymbol{C} + \boldsymbol{G} \,;$$

when $\boldsymbol{W}_q = \mathbf{I}$ and $\ddot{\boldsymbol{q}}_0 = \boldsymbol{M}^{-1}(\boldsymbol{\tau}_0 - \boldsymbol{C} - \boldsymbol{G})$ the resulting control law is the well known combination of a resolved acceleration based controller with a dynamical model, as first proposed by Hsu, Mauser, and Sastry (1989).

Given the correspondence between desired acceleration and torques, other approaches consider instead, in equation (5.11), a loss function

$$L(\boldsymbol{\tau}) = (\boldsymbol{\tau} - \boldsymbol{\tau}_0)^T \boldsymbol{N} (\boldsymbol{\tau} - \boldsymbol{\tau}_0) \,,$$

where $\boldsymbol{\tau}_0$ represents the torque corresponding to some desired secondary task. It is easy to show, using the dynamics relation $\boldsymbol{\tau} = \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C} + \boldsymbol{G}$, that $L(\boldsymbol{\tau})$ corresponds to the cost (5.11) if the substitutions $\boldsymbol{W}_q = \boldsymbol{MNM}$ and $\ddot{\boldsymbol{q}}_0 = \boldsymbol{M}^{-1}(\boldsymbol{\tau}_0 - \boldsymbol{C} - \boldsymbol{G})$ are made. It can be shown that plugging these two expressions into (5.12), after some simplifications, results in a desired torque vector given by

$$\boldsymbol{\tau} = \boldsymbol{N}^{-\frac{1}{2}} \left(\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^{-\frac{1}{2}}\right)^{\dagger} \left(\ddot{\boldsymbol{x}}_r - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\boldsymbol{M}^{-1}(\boldsymbol{C} + \boldsymbol{G})\right) + \\ + \boldsymbol{N}^{-\frac{1}{2}} \left[\mathbf{I} - \left(\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^{-\frac{1}{2}}\right)^{\dagger} \left(\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^{-\frac{1}{2}}\right)\right] \left(\boldsymbol{N}^{-\frac{1}{2}}\boldsymbol{\tau}_0\right) .$$

This is the framework presented by Peters, Mistry, et al. (2005): by choosing different values for $\boldsymbol{N}$ several known controllers can be obtained: making $\boldsymbol{N} = \boldsymbol{M}^{-2}$, for instance, results again in the controller of Hsu, Mauser, and Sastry (1989), while $\boldsymbol{N} = \boldsymbol{M}^{-1}$ recovers the controller of Khatib (1987).

Practical Considerations

Besides the estimation of the Jacobian $\boldsymbol{J}$, resolved acceleration controllers also need current joint velocities $\dot{\boldsymbol{q}}$ and an estimate for $\dot{\boldsymbol{J}}$, as can be seen from equation (5.12). Numerically differentiating the Jacobian may introduce some estimation noise, and should be handled with care. Also, cascading the controller with a dynamical model of the robot, to obtain the actuation torques, also requires the learning of the inverse dynamics model.

The kinematics model in (5.10) and the dynamics model in (5.2) become coupled when the overall torque controller cannot be expressed as a cascade of a resolved acceleration controller and an inverse dynamics model, as in the work of Hsu, Mauser, and Sastry (1989); as a consequence, matrices $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{G}$ may appear in the control law in a unstructured manner. This complicates the learning process, as it means that these quantities also need to be estimated from the data, independently of the estimation of the kinematic or dynamic sensorimotor maps. Like in the velocity control case, direct learning of the sensorimotor map needed for control, represented by the relation $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{x}}_r) \mapsto \ddot{\boldsymbol{\tau}}_a$, is not feasible using standard learning approaches, due to the kinematics redundancy. In this case the introduction of a penalty function on the torques $\boldsymbol{\tau}$ may remove this ambiguity, as done for instance in (Peters and Schaal, 2008). Note, however, that in this case the training algorithm directs the learning process towards a particular solution for the operational space control problem, somehow discarding the possibility of applying different control schemes using the same learned model.

5.2 Open and Closed-Loop Control Using IMLE

Resolved acceleration and resolved motion rate controllers described in Sections 5.1.2 and 5.1.2 fall within the class of closed-loop control, where feedback sensory data is used to track the desired end-effector trajectory. In this type of control, this feedback usually comes in the form of current readings of position, velocity or acceleration of the end-effector.

Closed-loop methods are known for their robustness towards external perturbations and model uncertainties: resolved motion rate control, for instance, works reasonably well even when the Jacobian estimate is not accurate, as long as the actuated joint velocities, calculated using the estimated Jacobian, move the end-effector closer to the desired task space point. In the absence of noise and perturbations, when a perfect kinematic model is available, straight line trajectories towards the desired end-effector position are obtained when RMRC schemes are used (Liegeois, 1977). However, since such methods need sensory feedback, they suffer, in their original form, from excessive sensor noise levels, sometimes resulting in robot jerky motions. Also, they are condemned to failure whenever signals from the sensors cease to be available, for instance when the end-effector is occluded in a vision based tracking system.

Open-loop controllers, on the other hand, rely on an accurate inverse sensorimotor model to provide the actuations that will drive the end-effector to the desired state. They are insensitive to sensor noise or sensor failure, as they do not use feedback sensory information in the control process. Still, they are not as robust as closed-loop controllers with respect to unexpected movement perturbations and inaccurate learned models.

The IMLE model provides an elegant solution to model based control of robotic end-effector position tracking, as it can provide forward, inverse and Jacobian estimates from the same model, that can be used for both open and closed-loop control. In closed-loop mode, IMLE provides Jacobian estimates that are used to control the robot joints, using either velocity or acceleration control schemes. If an open-loop control is desired, the multiple solutions resulting from IMLE inverse prediction can be used to plan a trajectory in the joint space, as described with more detail in Section 5.2.2. This results in a general, learning based control scheme that can be switched from closed to open-loop at any time, either due to a high noise level in the sensors, occlusions in visual feedback or simple sensor failure. In fact, IMLE can be used to compare sensor readings to their expected values, as predicted from its forward model, in this way enabling an automatic switch to open-loop mode whenever there is a large discrepancy in these values. This is a distinctive characteristic of the IMLE probabilistic model: as discussed in Chapter 2, while many learning algorithms were successfully applied to operational space control, none of these methods provides a learned model that can be simultaneously used in a feedback and feedforward control scheme.

The rest of this section provides experimental results supporting the effectiveness of the IMLE model in dealing with both open and closed-loop control, in both an online and offline learning setting.

5.2.1 Closed Loop Position Tracking

In the following experiments, the gradient projection method originally proposed by Liegeois (1977), as given by equation (5.8), is adopted to control the end-effector position in task space, where the null space of the main task is used to keep the joints values as far as possible from their physical limits, using

$$H(\boldsymbol{q}) = \frac{1}{N_q} \sum_{i=1}^{N_q} \left(\frac{\boldsymbol{q}_i - \mathbf{a}_i}{\mathbf{a}_i - \boldsymbol{q}_i^{max}} \right)^2 \quad , \qquad \text{with } \mathbf{a}_i = \frac{\boldsymbol{q}_i^{max} + \boldsymbol{q}_i^{min}}{2} \; ;$$

the constant N_q denotes the overall number of joints and $\boldsymbol{q}^{min}$ and $\boldsymbol{q}^{max}$ are the lower and upper joints limits.

Motor velocity commands $\dot{\boldsymbol{q}}_a(t)$ are calculated, at each time step, according to equations (5.3) and (5.8). When the system is close to singularities a regularization term is introduced in the pseudo-inverse, similar to the regularization term of the damped least squares method of Nakamura and Hanafusa (1986) and Wampler (1986), to avoid numerical instabilities. The Jacobian $\boldsymbol{J}(\boldsymbol{q}(t))$ is obtained, at each time step, from the current IMLE model, taking the estimate of the learned map local slope at input query point $\boldsymbol{q}(t)$.

Following a trajectory in the task space, represented by a sequence of task space points $\boldsymbol{x}_d^{(i)}$, can be accomplished, using this closed-loop controller, by simply feeding the desired task space points, together with desired velocities $\dot{\boldsymbol{x}}_d^{(i)}$, in sequence to the controller, switching to the next reference point $\boldsymbol{x}_d^{(i+1)}$ whenever the current point $\boldsymbol{x}_d^{(i)}$ is reached.

5.2.2 Open Loop Trajectory Planning

Given a trajectory in the task space, represented by a sequence of task space points $\boldsymbol{x}_d^{(i)}$, together with corresponding desired velocities $\dot{\boldsymbol{x}}_d^{(i)}$, the open loop trajectory planning problem consists in finding joints position profiles, $\boldsymbol{q}_d(t)$, eventually together with velocity and accelerations given by $\dot{\boldsymbol{q}}_d(t)$ and $\ddot{\boldsymbol{q}}_d(t)$ respectively, that will generate a task space trajectory satisfying the desired position and velocity constraints. This problem, as previously discussed in this chapter, is ill-posed, as redundant robots exhibit, for each desired task space position, a continuum of inverse solutions. To generate a feasible joint space trajectory, first IMLE is used to obtain, for each task space point $\boldsymbol{x}_d^{(i)}$, a set of inverse kinematics solutions $\boldsymbol{q}_j^{(i)}$. After that, a single solution for each of the inverse kinematics sets of solutions is picked, by imposing a penalty on the overall joint space displacement and on the predicted forward error for the whole trajectory, choosing the

joint space sequence that minimizes such cost. Finally, position, velocity and acceleration profiles are generated for each joint that respect the velocity and acceleration constraints for the joints. These steps are further detailed in the following text.

Inverse Prediction

The IMLE model can directly provide inverse predictions $\hat{\boldsymbol{q}}_j^{(i)}$ for a given query $\boldsymbol{x}_d^{(i)}$, calculated using the probabilistic model learned so far. In general, as discussed in Chapter 3, inverse predictions may be not as accurate as predictions taken from the forward model, due to the multi-valued nature of the inverse kinematics map. However, since forward and Jacobian estimates can also be readily obtained from the same learned model, these estimates can be used to improve $\hat{\boldsymbol{q}}_j^{(i)}$, the inverse kinematics solutions provided by the IMLE model for each of the desired positions in the task space. Assuming a first order approximation to the estimated forward kinematics map, it is straightforward to obtain the new inverse prediction $\boldsymbol{q}_j^{(i)}$ that will exactly produce the desired output $\boldsymbol{x}_d^{(i)}$, in a least norm sense. This quantity is the solution of the relation,

$$\hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^{(i)}) + \boldsymbol{J}(\hat{\boldsymbol{q}}_j^{(i)}) \left(\boldsymbol{q}_j^{(i)} - \hat{\boldsymbol{q}}_j^{(i)}\right) = \boldsymbol{x}_d^{(i)} \ ,$$

where $\hat{\boldsymbol{q}}_j^{(i)}$ is an inverse solution provided by IMLE, $\boldsymbol{q}_j^{(i)}$ is the corrected valued for this inverse solution, and $\boldsymbol{J}(\hat{\boldsymbol{q}}_j^{(i)})$ and $\hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^{(i)})$ are respectively the Jacobian and forward predictions evaluated at $\hat{\boldsymbol{q}}_j^{(i)}$, as predicted by IMLE. This results in the following corrected inverse estimate:

$$\boldsymbol{q}_j^{(i)} = \hat{\boldsymbol{q}}_j^{(i)} + \boldsymbol{J}^{\dagger}(\hat{\boldsymbol{q}}_j^{(i)}) \left(\boldsymbol{x}_d^{(i)} - \hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^{(i)})\right) \ ; \tag{5.13}$$

in a small neighbourhood of $\hat{\boldsymbol{q}}_j^{(i)}$ this correction will drive the error $(\boldsymbol{x}_d^{(i)} - \hat{\boldsymbol{x}}(\hat{\boldsymbol{q}}_j^{(i)}))$ approximately to zero, according to the learned model.

Trajectory Optimization

Having a set of candidate inverse solutions for each desired task space point, $\boldsymbol{q}_j^{(i)}$, the main issue is then how to choose an appropriate solution from each of these sets. A sensible approach is to pick the inverse solutions in a way that the overall joints displacement is minimized, avoiding large jumps in the joint space, and also to prefer inverse solutions with a low forward error, as estimated from the learned model.

To achieve such behaviour, the approach proposed in (Qin and Carreira-Perpinan, 2008) is closely followed, where a joint space trajectory is obtained, from the full set of inverse solutions, by minimization of a global penalty of the form

$$\sum_{i=1}^{N} \|\boldsymbol{q}^{(i)} - \boldsymbol{q}^{(i-1)}\| + \lambda \sum_{i=1}^{N} \|\boldsymbol{x}_d^{(i)} - \hat{\boldsymbol{x}}(\boldsymbol{q}^{(i)})\|^2 \ , \tag{5.14}$$

for $\lambda \geq 0$. The first term penalizes large joint space variations, encouraging short trajectories, while the second term imposes a penalty on inaccurate solutions given by inverse prediction. Note that, contrary to the work of Qin and Carreira-Perpinan (2008), the IMLE model is used to get the predictions $\hat{\boldsymbol{x}}(\boldsymbol{q}^{(i)})$ — in (Qin and Carreira-Perpinan, 2008), the Mixture Density Network used for learning the inverse kinematics map cannot generate forward predictions, and so the forward error must be calculated, either using an analytical model for the direct kinematics or an independent learning algorithm. Also, another fundamental difference is the online nature of the IMLE algorithm: in (Qin and Carreira-Perpinan, 2008) the inverse kinematics model must be learned offline, before being used for control.

Minimization of (5.14) is computationally very cheap, using Dijkstra's algorithm over a directed acyclic graph; compared to the computations involved in obtaining forward and inverse predictions, its cost is negligible.

Generation of motor commands

After the calculation of a joint space trajectory, corresponding to the desired task space trajectory, it is needed to generate temporal positions, velocity and acceleration profiles for each of the joints, that can be fed to low level PID joint controllers or to an inverse dynamics model, responsible for the generation of a set of joint torques that will achieve the desired joint profiles. Additionally, it may be of interest to traverse each of the task space points with a given desired velocity $\dot{\boldsymbol{x}}_d^{(i)}$. Fortunately, these velocities can easily be mapped back to the joint space, once again using the pseudo-inverse of the estimated Jacobian $\boldsymbol{J}^{\dagger}$, making

$$\dot{\boldsymbol{q}}_d^{(i)} = \boldsymbol{J}^{\dagger}(\boldsymbol{q}_d^{(i)})\ \dot{\boldsymbol{x}}_d^{i}\,. \tag{5.15}$$

Position, velocity and acceleration temporal profiles that achieve such task, described by $\boldsymbol{q}_d^{(i)}(t)$, $\dot{\boldsymbol{q}}_d^{(i)}(t)$ and $\ddot{\boldsymbol{q}}_d^{(i)}(t)$ respectively, can then be obtained for each joint, by resorting to classic joint trajectory generation based on, for instance, cubic polynomials or Bang-Bang acceleration policies (Craig, 1989), that take the acceleration and velocity limitations of each joint into account. These temporal profiles can then be fed into the low level joint controllers or to an inverse dynamics model. Note that this control scheme does not need information about the end-effector position (or velocity or acceleration): control is performed in a pure open-loop manner, and the trajectory planning is executed without resorting to task space feedback.

5.2.3 Experimental Evaluation

The following experiments are performed on the iCub simulator (Tikhanoff, Fitzpatrick, et al., 2008), a software that uses ODE (Open Dynamic Engine) for realistically simulating the movement and physical interaction with objects of iCub, a 53 degrees of freedom hu-

Figure 5.1: A snapshot of the iCub Simulator used in the experiments.

manoid robot for research in embodied cognition (Tsagarakis, Metta, et al., 2007; Metta, Natale, et al., 2010; Parmiggiani, Maggiali, et al., 2012; Natale, Nori, et al., 2013) — Figure 5.1 displays a snapshot of this simulator.

The right arm and the waist of the robot are actuated to control the end-effector position in the 3D Cartesian space, using the controllers presented in the two previous sections. The joint space vector $\boldsymbol{q}$ used in the experiments has 7 degrees of freedom, corresponding to the shoulder yaw, pitch and roll rotations (elevation/depression, adduction/abduction and rotation of the arm), the elbow flexion/extension, and the waist yaw, roll and pitch rotations (rotation, adduction/abduction, elevation/depression of the trunk). The corresponding joint limits are defined in Table 5.1. The task space end-effector position $\boldsymbol{x}$, on the other hand, consists in the 3D position of the hand, taken with respect to a fixed reference frame placed on the ground, between the robot feet.

	arm				waist		
$\boldsymbol{q}^{min}$	$-80°$	$0°$	$0°$	$20°$	$-30°$	$-30°$	$-10°$
$\boldsymbol{q}^{max}$	$0°$	$80°$	$80°$	$80°$	$30°$	$30°$	$30°$

Table 5.1: Joints limits of the iCub robot simulator.

Open and Closed-Loop Trajectory Following

The first experiment compares the performance of the proposed open and closed-loop control schemes when following a desired trajectory in the task space. In this and the following experiments, the robot was asked to perform a square-like pattern in a X-Y plane in the task space, with side length equal to 10 cm, using its end-effector. One of

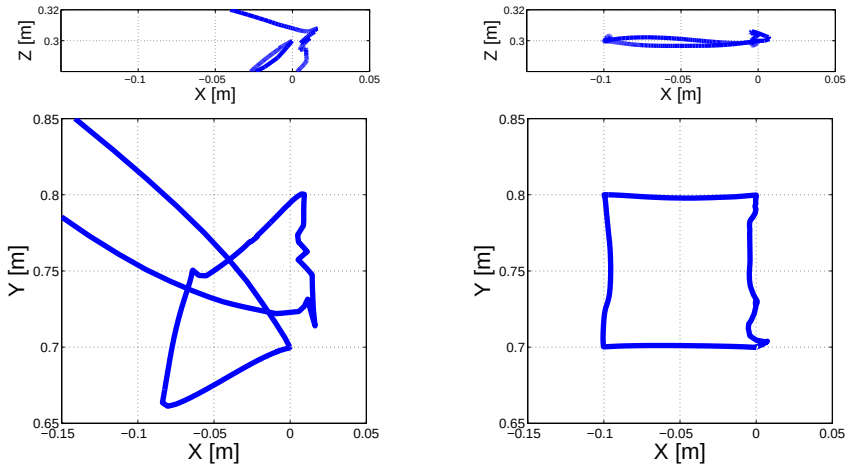
the main advantages of the IMLE algorithm is the ability to work in an online manner, learning and updating its internal parameters while using the same model to make the predictions needed to execute the desired actions. This motivates using the model provided by the IMLE learner to try to perform the desired trajectory from scratch, using an untrained IMLE probabilistic model and updating the IMLE probabilistic model on the fly, presenting it with the training points as they are acquired during the robot movement.

Figure 5.2 shows the results obtained when trying to draw the square-like pattern, both for open-loop (in the left) and closed-loop control (in the right). The robot repeated the movement for several iterations, trying to follow the desired square-like trajectory in each iteration.

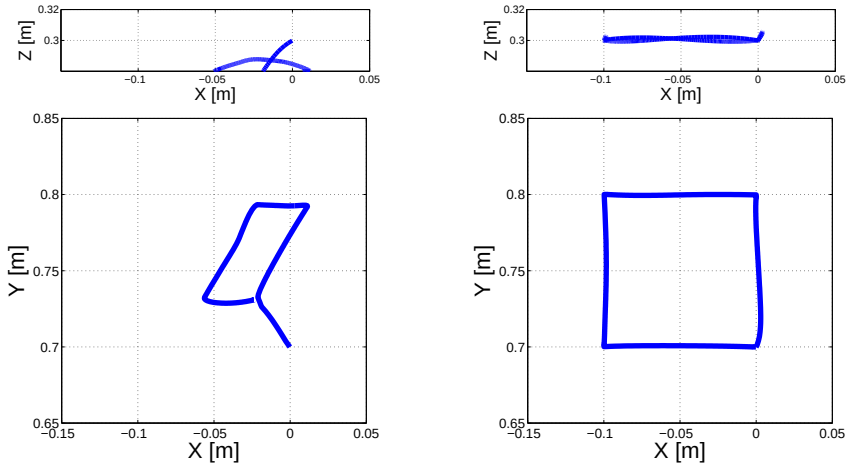
The open-loop controller planned, at each iteration, the joint space velocity and position profiles that would take the end-effector to each of the edges of the square, in sequence, stopping at each of them. Additionally, some task space via points were defined between the square edges, to guarantee a reasonably straight trajectory in the task space, between the edges of the square. As for the closed-loop controller, it sufficed to set its end-effector position reference to each of the edges of the desired square-like trajectory: each time the end-effector was close enough to the target point or a time limit was reached the reference in the task space would change to the next edge. There was no need to set via points for the closed loop controller, since, as stated before, RMRC schemes are known to generate straight line task space trajectories when the Jacobian estimate is accurate enough.

As expected, a poor controller performance was observed during the first iterations of the movement, specially for open-loop control, due to a not yet properly learned model. However, as shown in Figure 5.2, after about 10 iterations the kinematics model had been learned and the robot could then properly follow the desired trajectory, for both open and closed-loop control. Note that in the first iterations the closed-loop controller was more successful at performing the task: this is a consequence of the online learning setup, since the closed-loop controller Jacobian estimation at each time step used an IMLE model that was also constantly being updated and improved. In contrast, the open-loop controller planned the whole trajectory in the beginning of the movement, and would only use the updated IMLE model in the next movement iteration. Alternatively, the open-loop controller could be easily adapted to re-plan the trajectory at a given time rate, to exploit the online update of the kinematics model.

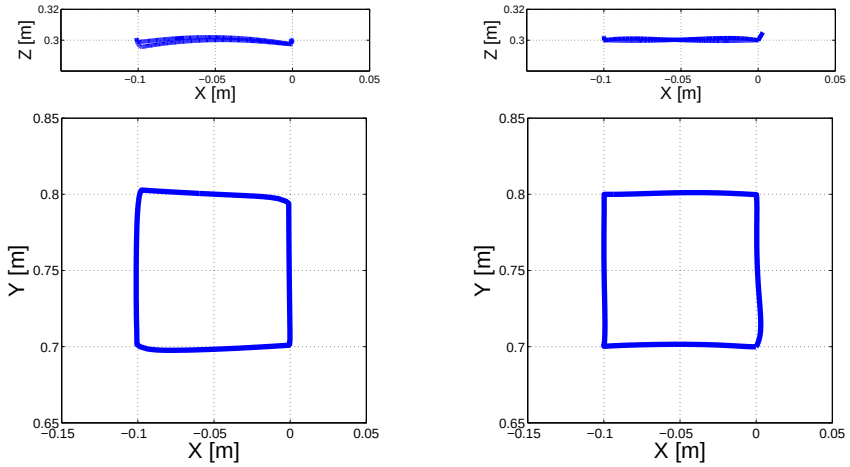
It can be rightfully argued that controlling the robot while simultaneously learning the model in an online fashion avoids the potential problems that can arise when estimating the inverse kinematics of a redundant robot, as the training data being fed to the learning algorithm originates mostly from a single joint space trajectory, biasing the learning algorithm to a specific solution of the inverse kinematics. Motivated by the desired to evaluate the IMLE inverse prediction capabilities in a more general setting, a random exploration



(a) First Iteration.



(b) Second Iteration.



(c) Tenth Iteration.

Figure 5.2: Online Learning and Trajectory Following: open-loop (left) and closed-loop (right).

in the joint space of the robot was performed: this motor babbling was executed until 100 000 training points were acquired and processed by the IMLE algorithm, in order to cover the whole workspace. After that, the task described in the previous experiment was repeated. There were no noticeable changes on the final open-loop trajectory, and the tracking error remained in the same level: this shows that, despite the redundancy in the kinematics map, the open-loop controller based on the inverse predictions provided by IMLE could still achieve a good accuracy while performing the desired task.

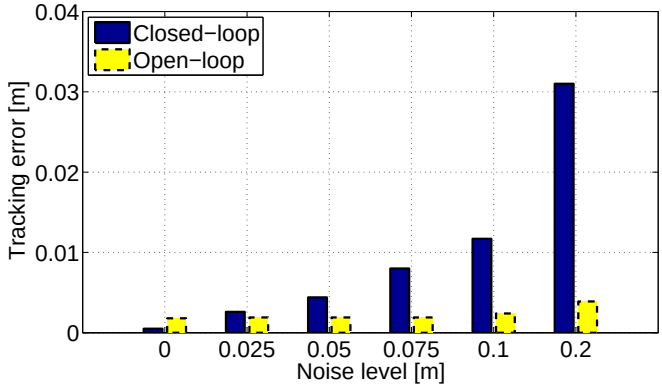
Sensitivity to Sensor Noise

Next, the performance of both open and closed-loop controllers under different sensor noise levels is analysed. While a performance drop is to be expected for both controllers when the noise increases, as a consequence of less precise learned IMLE models, a larger sensitivity of the closed-loop controller to the noise is also anticipated, as this controller relies on sensor readings to calculate the actuation at each control step. This is confirmed in Figure 5.3, where the RMSE is depicted for both open and closed-loop controllers as a function of the noise level, for the square-like target trajectory, after the kinematics model has been properly learned, and where a noise level of X means a noise uniformly distributed between -X and X. The jerkiness of the motion is also depicted in this figure: as expected, the jerk for the closed-loop controller quickly increases with the output noise level, while the open-loop jerk is kept at negligible values due to the smooth joint profiles generated by the trajectory planner, using cubic polynomials.

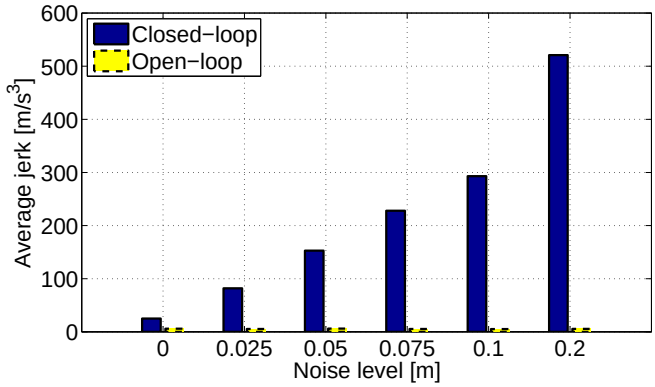
Figure 5.4 shows the attained trajectories, for open and closed-loop, for three different noise levels. Also, shown in Figure 5.5 is a comparison of the first joint position profile under a severe sensor noise scenario, for both types of controllers; the obtained results for the other joints were similar.

Sensor Failure

Sometimes the sensors reading the end-effector task space position may fail: this may be due to a sensor malfunction or, for vision based systems, simply a consequence of some kind of end-effector occlusion. When such kind of situation occurs an open-loop control scheme must be used, as the sensory feedback that makes the current task space end-effector position $\boldsymbol{x}(t)$ is no longer available. Figure 5.6 depicts the catastrophic trajectory that arises when, using a closed-loop controller, a sensor malfunction is simulated at a specific instant. In the right side of the same figure is shown the resulting trajectory when a composite controller is used for the same situation: this controller uses the Jacobian based RMRC scheme during normal operation, and when the end-effector position feedback fails a seamless switch to open-loop control, based on the same learned model used for closed-loop control. As soon as the failure is detected, the composite controller switches to



(a) Average error (RMSE).



(b) Average jerk (m/s^3).

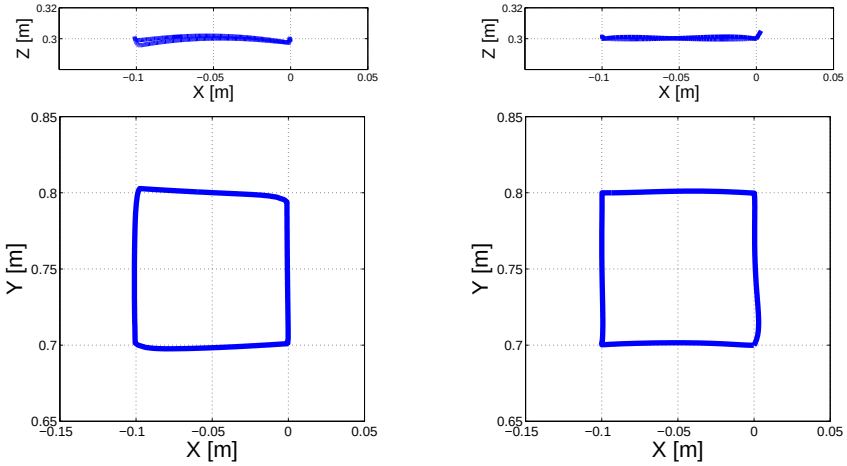
Figure 5.3: Average error and jerk over the square-like trajectory, for different sensor noise levels.

open-loop mode and a new trajectory is planned and executed that does not depend on task space feedback.

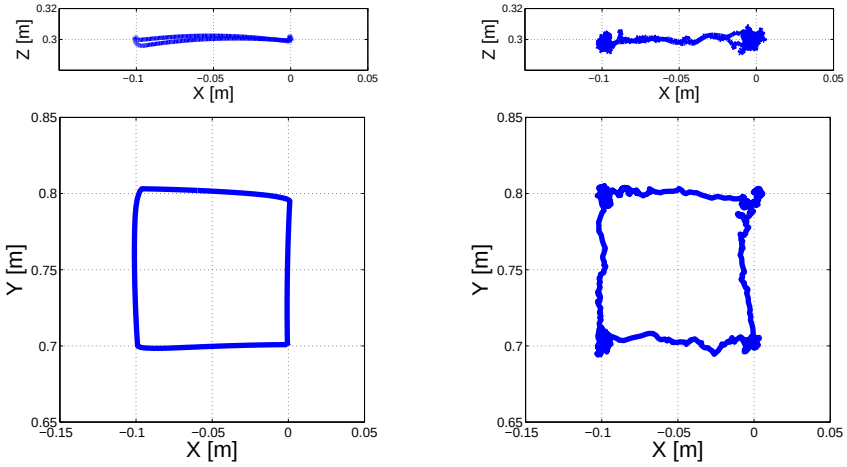
As shown in the figure, the composite controller based on both open and closed-loop control shows no noticeable degradation on the task performance when the failure occurs. Above each end-effector trajectory is also depicted the temporal profile for the first joint of the arm. Note that even when a sensor failure is not properly communicated to the controlling algorithm, resulting in a situation where the sensory feedback simply returns erroneous values, the control system can detect such situation by permanently comparing the sensor readings to the corresponding predicted values, taken from forward predictions given by the IMLE model.

5.3 Learning and Control of Switched Systems Using IMLE

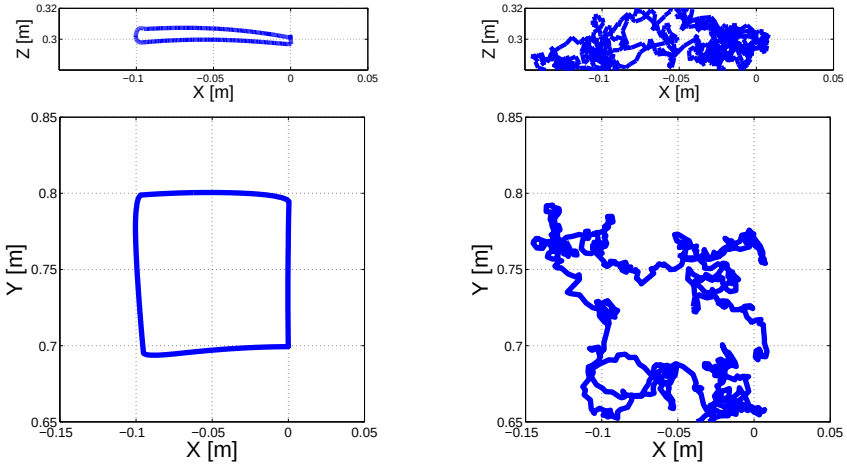
Many robotic tasks, involving handling and manipulation of different objects, make the environment and the mappings to be learned non-stationary. The kinematics mapping from robot joint angles to end-effector position, for instance, changes whenever different



(a) No noise.

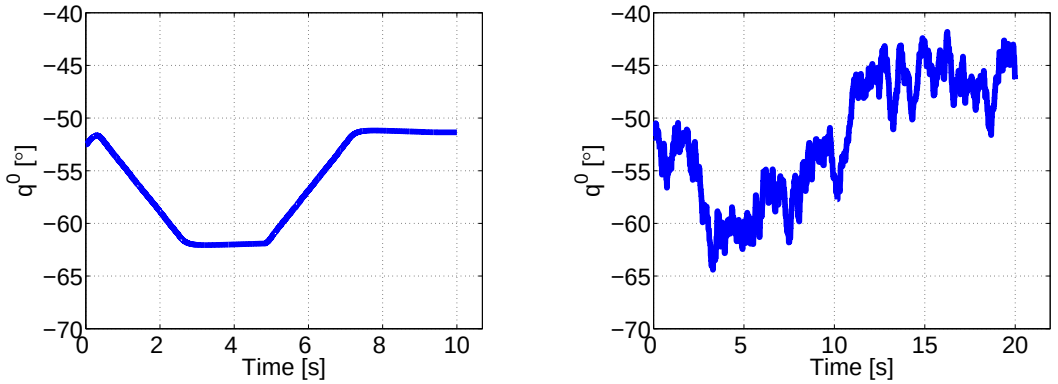


(b) Moderate noise level.



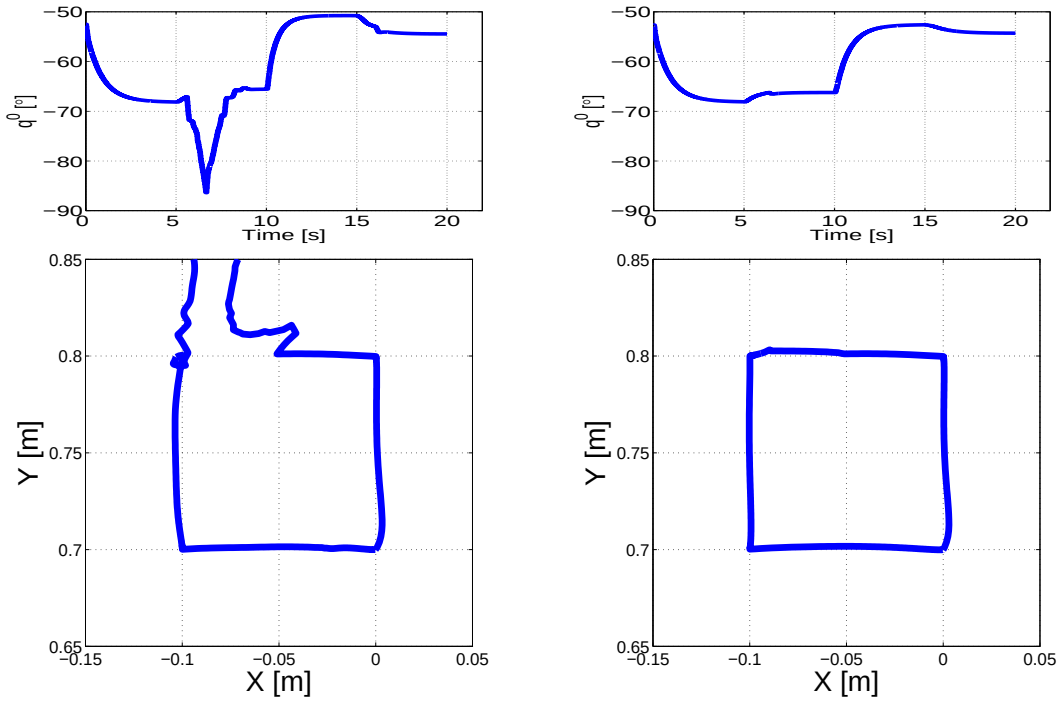
(c) High noise level.

Figure 5.4: Trajectory following after the learning phase, for several noise levels: open-loop (left) and closed-loop (right).



(a) Open-loop. (b) Closed-loop.

Figure 5.5: Position profiles for the first joint, as the robot executes the task with a high level of sensor noise.



(a) Closed-loop controller. (b) Composite controller, changing to open-loop when the fault is detected.

Figure 5.6: Simulating an end-effector position sensor fault.

tools are used; another classical example is the change in the robot dynamics due to the variation of the load of the end-effector. This is known as learning under a varying context, where an unobserved context variable changes the map to be learned. Such context can generally be a discrete variable, corresponding to the case where only a finite, albeit unknown, number of different contexts exist, or continuous, indicating a smooth change on the mapping to learn. The most straightforward answer to this problem is to introduce some form of adaptation in the learning algorithms, making them forget past

experience through the use of some kind of forgetting factors mechanism. Of course, it is terribly inefficient to relearn the complete mapping every time the context changes, specially when there is an effective chance that a previously learned context may be presented again to the robot. Another approach to this problem, for the discrete case, is to keep a set of models that describe the robot model for each different context: some of the earliest work on this subject is given by the works of Narendra and Balakrishnan (1997) on adaptive control and the MOSAIC architecture of Haruno, Wolpert, and Kawato (2001), while (Petkos and Vijayakumar, 2007) constitutes a more recent vision on this matter. This latter work identifies three critical issues when learning multiple models for use in robot control. The first issue is how to identify the correct number of models to use without any problem specific information. The other two issues are how to estimate the current context, given that the correct number of models to use is known, and how to use such estimation for either controlling the robot or further training the models. The adaptive control of Narendra and Balakrishnan (1997), for instance, considers that an appropriate number of models is given *a priori*, already trained and exhibiting good performance within each context; the MOSAIC architecture, on the other hand, assumes that some perceptual cues are available that can guide a correct context estimation in the early learning process, that can in turn successfully assign the perceived data points to a predefined number of models — which in practice turns out to be a very optimistic assumption. Additionally, the results presented for the MOSAIC model are somewhat limited to a very simple system consisting of an object moving along a single direction axis. Finally, the approach presented in (Petkos and Vijayakumar, 2007) ambitiously claims the ability to deal with continuously varying contexts, although, as the authors admit, their method only holds under changes in the mass of the object being manipulated — it could not be applied, for instance, when varying a robot link length while trying to learn its forward kinematics. Their assumption of an explicit latent context variable also brings some problems when the current context needs to be inferred for training purposes: in the continuous case they need to resort to two models previously trained using context labelled data before they are able to generalize to unseen contexts, while in the discrete case a bootstrap, based on a EM procedure over a batch of unlabelled data points, is required when no trained models exist yet — this however, goes against the online, incremental philosophy of LWPR (Vijayakumar, D'Souza, and Schaal, 2005), the function approximation algorithm used in the corresponding simulations.

Some recent works focus more specifically on the problem of adapting the robot kinematics under different tools operation. In (Nabeshima, Kuniyoshi, and Lungarella, 2006) a simple 2-joints planar manipulator is controlled using an analytical model of the Jacobian, and when a tool is added to the kinematic chain the corresponding Jacobian is obtained through multiplication of the analytical Jacobian by a linear constant matrix, which is learned exploiting the temporal integration of visual and tactile information dur-

ing motor exploration. Another approach is proposed in (Rolf, Steil, and Gienger, 2010), where a recurrent neural network parametrized with the length of the tool is used to estimate the inverse kinematics of a humanoid robot. However, the length of the tool must be known in advance to train and query the neural network. Additionally, training is done using circular trajectories in a fixed plane: this procedure learns a subspace of much lower dimensionality than the joints space dimension being used. Another major limitation of these works is their inability to account for other than rigid transformations, *i.e.*, their inability to cope with flexible or deformable tools.

The IMLE model framework allows modelling the map to be learned directly as an unknown multi-valued function: this approach bears some significant advantages over the other aforementioned approaches to estimation and control of discrete varying context systems: on one hand, there is no need to maintain a bank of single-valued function approximation models, since IMLE produces a discrete set of solutions for each input query point; the number and values for this set of solutions depend on the specific input query location and the information gathered so far by the algorithm. This also avoids the need to define or estimate the number of single-valued models to use. Secondly, the IMLE training process, based on the EM algorithm, automatically and transparently assigns responsibilities to each of the local models for each training point, with no need to explicitly maintain an estimate for the hidden context variable. This even allows for the existence of a different number of contexts in different locations of the input space. Choosing an appropriate control action is also very simple using IMLE: assuming some form of continuity and smoothness, a particular solution, for a given query point, can be picked by simply choosing the predicted solution closest to the most recently observed output point.

To evaluate the performance of the IMLE algorithm under a discrete varying context situation, this algorithm is used to learn the kinematics map of two distinct humanoid robots: section 5.3.1 describes the experiments made using the iCub simulator, while section 5.3.2 introduces the Kobian robot and presents the experimental results obtained using this humanoid robot. The same task space closed-loop control scheme is used to track a desired trajectory in both situations: such controller, once again, is based on the approach first introduced by Liegeois (1977), already presented in section 5.2.1.

In the following experiments the online multi-valued estimation performance of the IMLE model is evaluated and compared to the estimation provided by LWPR, a single-valued function approximation algorithm, in a switching context scenario. It is important to stress out that this, of course, is not a fair comparison, as LWPR simply cannot cope with multi-valued data and thus is condemned to a poor prediction performance in this situation. Nevertheless, providing these comparisons may help understanding the technical limitations of classical single-valued function approximation schemes in the presence of training data arising from switching sensorimotor contexts, and how a multi-

valued learning algorithm like IMLE can circumvent such limitations.

5.3.1 Experimental Results: iCub Simulator

The same joint vector $\boldsymbol{q}$ and corresponding limits used in section 5.2.3 are also used in the following experiments, carried using the iCub simulator, and thus their description will be omitted here for brevity. Differently from previous experiments, the end-effector position $\boldsymbol{x}$ can be considered the robot hand position or the location of the tip of a tool, depending on whether the robot is holding such tool or not. Two different tools where considered in the following simulations: a 28 cm stick tool and a 48×30 cm L-shaped tool, displayed in Figure 5.7. This makes the kinematics map to be learned to vary in time, as the task space position $\boldsymbol{x}$ depends, for the same joint configuration $\boldsymbol{q}$, on the presence or not of a tool held by iCub.

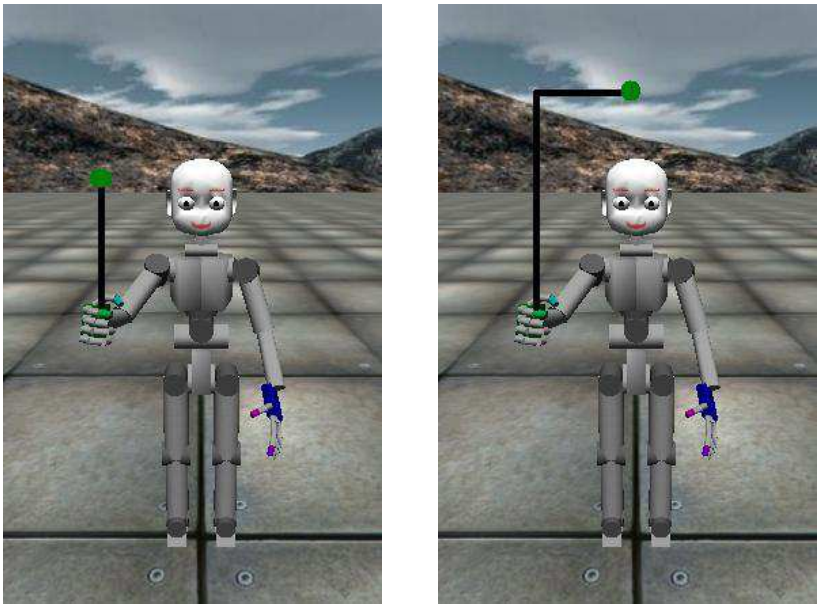


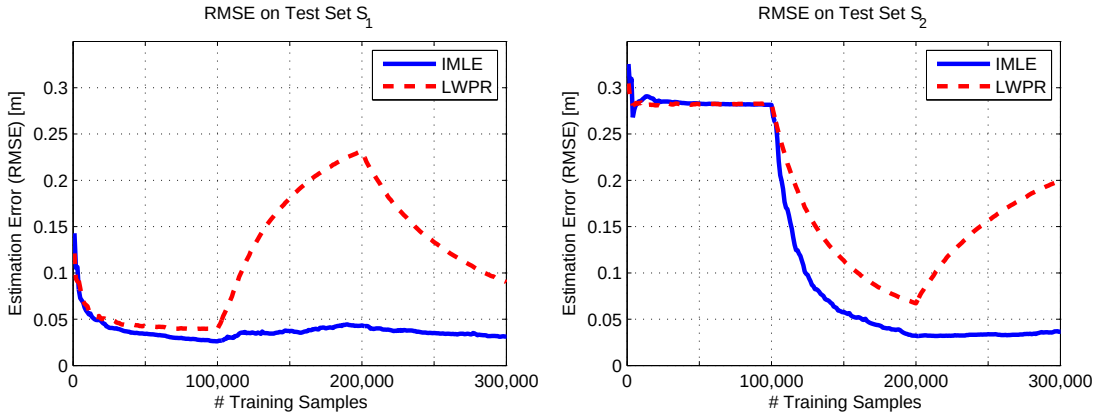
Figure 5.7: Snapshots of the iCub Simulator grabbing the two different tools used in the experiments: on the left, the 28 cm stick tool, on the right, the 48×30 cm L-shaped tool.

Motor babbling and model estimation

The first experiment consists in training the IMLE and LWPR learning algorithms with data pairs $(\boldsymbol{q}_i, \boldsymbol{x}_i)$ sampled from random trajectories performed by the robot in its joint space. During this motor babbling phase the robot moved to random reference configurations in the joint space, using a low-level joint position controller, spanning the whole space within the robot limits defined in Table 5.1. The first 100 000 training points were acquired with no tool held by iCub, *i.e.*, considering $\boldsymbol{x}$ to be the iCub right hand coordinates. After that, the 28 cm stick tool (see left image in Figure 5.7) was attached to

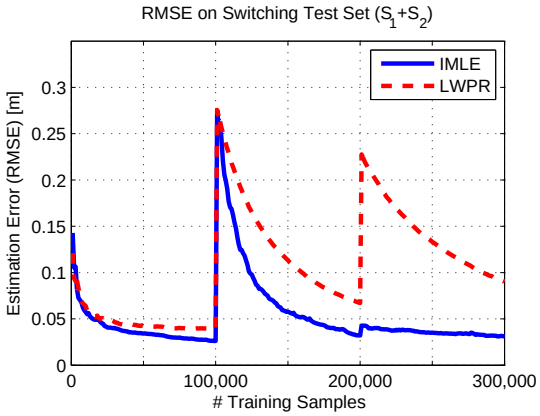
the robot hand, without informing the learning algorithms of such change in the forward kinematics. After the acquisition of more 100 000 training points, the tool was removed and the robot continued the motor babbling for more 100 000 points.

During this procedure, the root mean square error (RMSE) over two independent test sets of 3 000 samples each, S_1 and S_2, was calculated: the joint values $\boldsymbol{q}$ were the same in both test sets, while the $\boldsymbol{x}$ values corresponded to the positions of the end-effector, considering either the hand position (S_1) or the position of the tip of the 28 cm stick tool (S_2). The results thus obtained are shown in Figure 5.8.



(a) Test set S_1 (no tool).

(b) Test set S_2 (28 cm stick tool).



(c) Test set is changed according to the tool used during training: first S_1 is used, then S_2 and finally S_1 again.

Figure 5.8: Forward kinematic prediction error with a switching context: IMLE and LWPR learning algorithms are trained first without the tool (until sample 100 000), then with the 28 cm stick tool (until sample 200 000) and then again without the tool (until sample 300 000). Figures above show the observed RMSE in different test sets.

As expected, the IMLE model performs much better than a single-valued learner: after being presented with training data coming from the two branches of the multi-valued relation, corresponding to the two different kinematic contexts, this model is able to successfully predict the task space position of the end-effector in both situations, *i.e.*, with

and without a tool attached to iCub hand. It is important to stress again that the presence of the tool is never signalled to the learning algorithms any time during the training process: from their point of view, they were trained with 300 000 undistinguishable training points. The reason for IMLE apparent superiority is, of course, the fact that it can maintain information regarding both multi-valued branches in its internal model. As for LWPR, it can be seen that its performance is quite similar to IMLE during the first phase of training, showing an RMSE comparable to IMLE in both test sets S_1 and S_2. When the second training phase starts, by changing the context and introducing the tool in the kinematic chain, some fundamental differences can already be found between the two algorithms, with IMLE converging faster in the presence of this new training data. This is a consequence of LWPR necessity to forget the relation learned using the first context, fully retraining its internal model to be able to approximate the new kinematic relation. This can be observed in Figure 5.8a, where a sudden increase in the RMSE evaluated over S_1 is observed when training with data coming from the alternative context. On the other hand, in the same situation, there is only a slight increase in the RMSE over S_1 for the IMLE model, showing that the new training data causes little interference on the linear experts assigned to the kinematics map corresponding to the first context.

When the context changes again, with training data being generated from the original kinematics relation, relating the joint vector to the iCub hand position with no tool being held, the prediction error of the IMLE model suffers almost no change (see Figure 5.8c), as its internal model kept the information corresponding to the initial context training data. The LWPR model, as expected, must again fully retrain its parameters, leading to a huge instantaneous increase of the prediction error when the tool is removed.

There is a lower bound on a single-valued learning algorithm prediction error in the presence of multi-valued relations: for a two-valued function like the one presented in the above experiment, this value can easily be found if the two branches of this relation are assumed to differ by a constant distance: this is precisely what happens when a tool is added to the iCub kinematic chain. In this case, the mean square error of a single-valued predictor $\hat{\boldsymbol{x}}(\boldsymbol{q})$ on the test set $\bar{S} = S_1 \cup S_2$ is, assuming S_1 and S_2 to have the same number of test points N_{test},

$$\begin{aligned} MSE^{\bar{S}}(\hat{\boldsymbol{x}}) &= \frac{MSE^{S_1}(\hat{\boldsymbol{x}}) + MSE^{S_2}(\hat{\boldsymbol{x}})}{2} \\ &= \frac{1}{2N_{test}} \left(\sum_{i=1}^{N_{test}} \|\boldsymbol{x}_i^{S_1} - \hat{\boldsymbol{x}}(\boldsymbol{q}_i)\|^2 + \sum_{i=1}^{N_{test}} \|\boldsymbol{x}_i^{S_2} - \hat{\boldsymbol{x}}(\boldsymbol{q}_i)\|^2 \right) , \end{aligned}$$

where $\boldsymbol{x}_i^{S_1}$ and $\boldsymbol{x}_i^{S_2}$ are test outputs coming respectively from S_1 and S_2, sharing the same input $\boldsymbol{q}_i$. It is relatively simple to see that the predictor $\hat{\boldsymbol{x}}^*$ that minimizes the above error is the one that produces, for each $\boldsymbol{q}_i$, a prediction corresponding to the average of the true solutions, *i.e.*, $\hat{\boldsymbol{x}}^*(\boldsymbol{q}_i) = (\boldsymbol{x}_i^{S_1} + \boldsymbol{x}_i^{S_2})/2$. In this situation, the mean square error

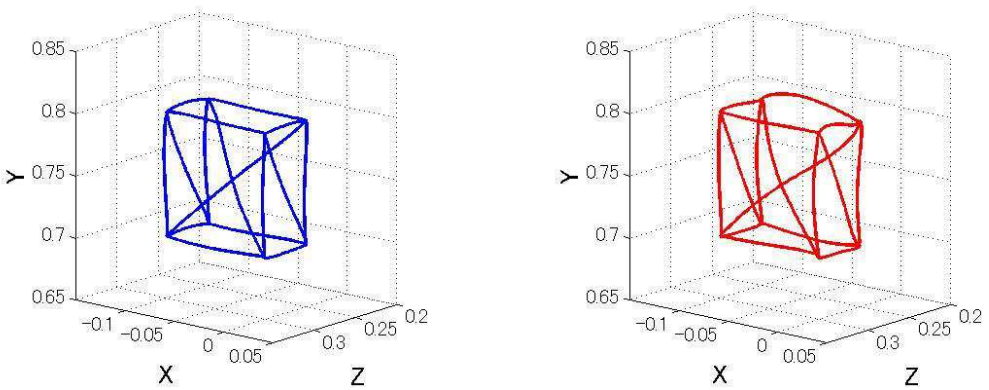
is $MSE^{\bar{S}}(\hat{\boldsymbol{x}}^*) = (\Delta L/2)^2$, where ΔL is the constant value by which the two branches of the multi-valued relation differ: in the above experiment this corresponds to the length of the displacement of the end-effector $\boldsymbol{x}$ caused by the introduction of the tool, *i.e.*, $\Delta L = 28\,\text{cm}$. Multi-valued learning algorithms, by being able to generate a set of multi-valued predictions for the same input query, are of course not troubled by this bound.

Task space control

The next experiment evaluates the performance of the IMLE algorithm while controlling the end-effector (be it the iCub hand or the tip of one of the tools), after training with data points generated by different contexts. The test movement consists of a sequence of 16 target positions that results in a cube with a 10 cm side, including some of its diagonals.

Three different training sets with 100 000 points each are considered, T_1, T_2 and T_3, corresponding respectively to the kinematics map when the end-effector is the iCub hand (no tool attached), when the stick tool is held by iCub and when the L-shaped tool is instead considered. The cube-like trajectory, correspondingly, can be performed using iCub hand, the stick tool or the L-shaped tool: these experiments are denoted respectively by S_1, S_2 and S_3. For illustration purposes, it is also shown in the following figures the trajectories attained by a single-value learner, here once more represented by the LWPR algorithm.

Figure 5.9 shows the execution of the test movement with the hand, after motor babbling without any tool (T_1). This is a standard single-valued learning scenario, and

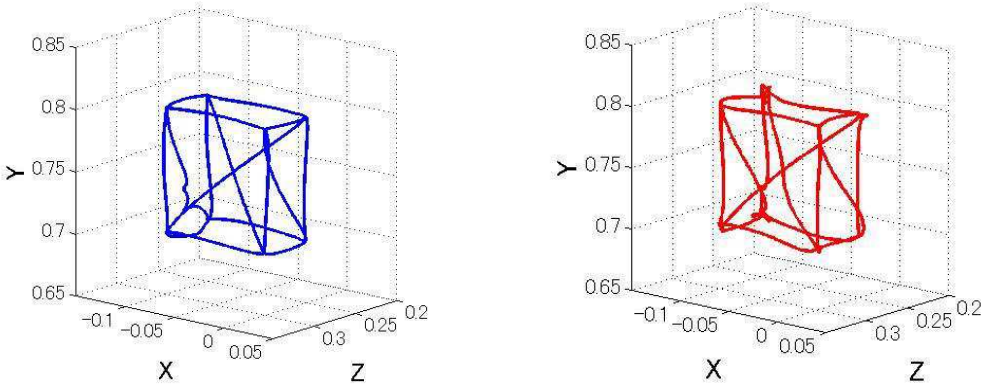


(a) IMLE. (b) LWPR.

Figure 5.9: Task space test trajectory using the hand (S_1), after training using data coming from the same context (T_1).

both algorithms achieve satisfactory performance. After that, both algorithms are also trained using the dataset T_2, corresponding to the stick tool, and the test movement is attempted using the same tool: the observed trajectories are depicted in Figure 5.10: while some minor degradation on the obtained trajectories can be observed — particularly

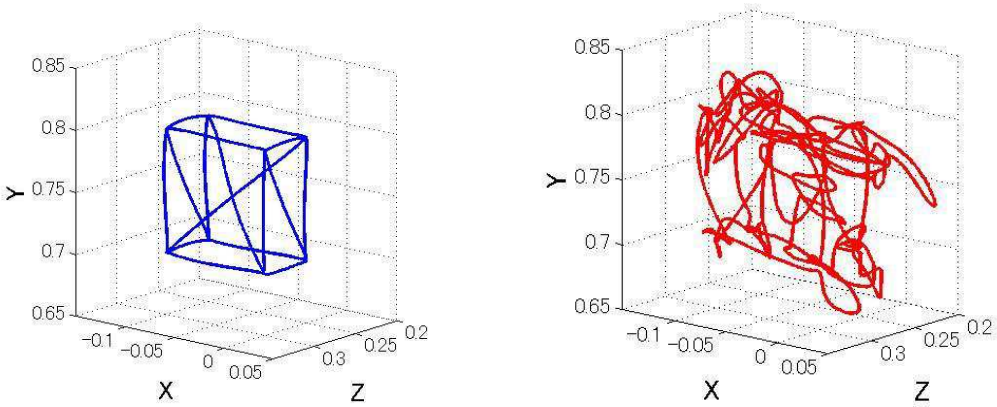
in those generated by the LWPR algorithm — the overall results are still acceptable.



(a) IMLE. (b) LWPR.

Figure 5.10: Task space test trajectory using the stick tool (S_2), after the training sequence $T_1 \mapsto T_2$.

If, however, the test movement is attempted without the stick tool, using the iCub hand, after the same training sequence took place ($T_1 \mapsto T_2$), some very distinct trajectories are produced, as can be observed in Figure 5.11. Here, LWPR can no longer perform



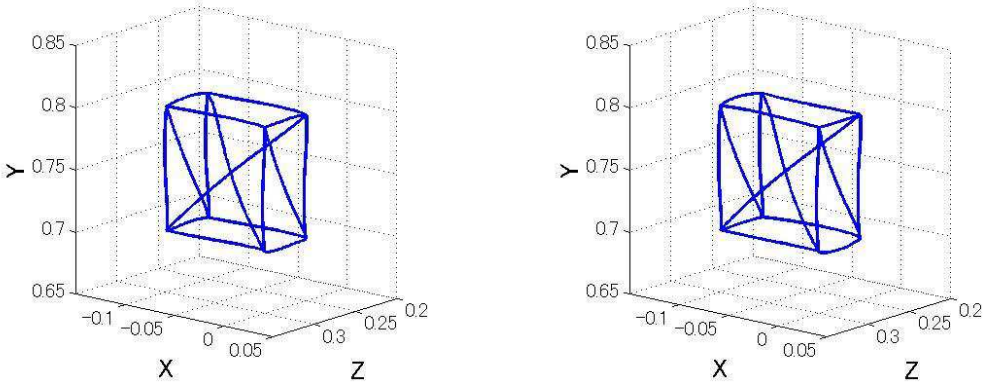
(a) IMLE. (b) LWPR.

Figure 5.11: Task space test trajectory using the hand (S_1), after the training sequence $T_1 \mapsto T_2$.

the cube trajectory using the iCub hand, as it adaptation to the stick tool context made the algorithm forget the original joints to hand position kinematics map. IMLE, on the other hand, does not exhibit any performance drop, still having the ability to perform the desired trajectory using the iCub hand.

After these tests, the training procedure was resumed, using again dataset T_1, corresponding to the original kinematics map, where no tool was held by iCub. Figure 5.12 shows the cube-shaped task performed by the IMLE learner, using either the hand or the tip of the stick tool. As can immediately be perceived from this figure, training using

data coming from the two different contexts (stick tool and no tool) does not cause any kind of interference in the learning process, and after this training takes place the robot can successfully perform the desired task using either its hand or the stick tool tip.



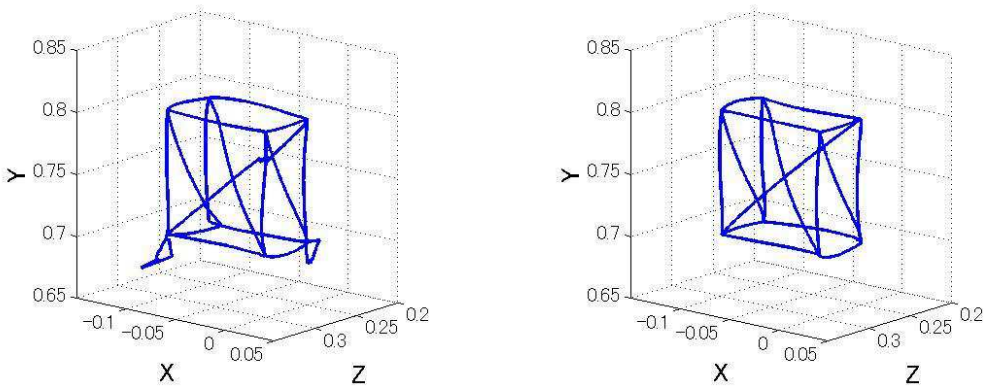
(a) Hand (no tool attached). (b) Stick tool held.

Figure 5.12: Task space test trajectory using IMLE, after the training sequence $T_1 \mapsto T_2 \mapsto T_1$. In the left figure the trajectory is performed using the hand with no tool attached, while in the right figure the stick tool is held.

It is interesting to study what happens when the IMLE learner tries to perform the cube-shaped test trajectory using the stick tool, without being previously trained with data generated from such kinematic relation: the obtained trajectories are depicted in Figure 5.13. Of course, since the IMLE learner had only been trained with no tool attached to its hand and had not yet seen any training data corresponding to this distinct scenario, its performance was expected to be very poor. However, differently from the previous experiments, the IMLE model was allowed to operate in a full online mode, *i.e.*, training data was acquired while the test trajectory was being tracked and it was immediately fed to the learning algorithm as soon as it was available, allowing the controller to improve its performance during the execution of the task. As seen in the figure, convergence is fast, and by the third attempt to track the cube-like trajectory the IMLE learner already achieves satisfactory results.

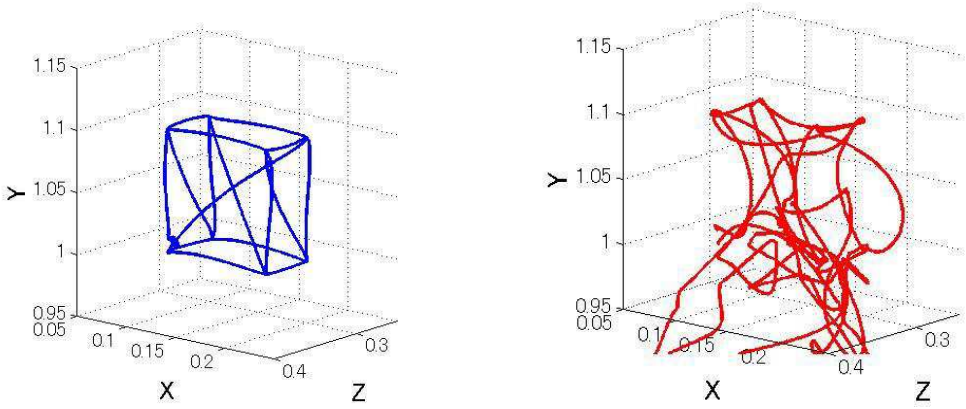
This experiment shows that the IMLE model allows new tools, corresponding to different sensorimotor contexts, to be dynamically introduced during normal operation and control of the end-effector position, without any kind of signalling of such change and not requiring a motor babbling phase for this new context, thus leading to a large degree of autonomy for the robot.

The final experiment introduces a third sensorimotor context, corresponding to the insertion of the *L*-shaped tool in the iCub kinematics chain (depicted in the right image in Figure 5.7). Motor babbling is performed using training data coming from the T_3 dataset, and then the test trajectory is performed using the *L*-shaped tool: results are depicted in Figure 5.14. Note that target positions used in this test are shifted in space, as the robot



(a) First iteration. (b) Third iteration.

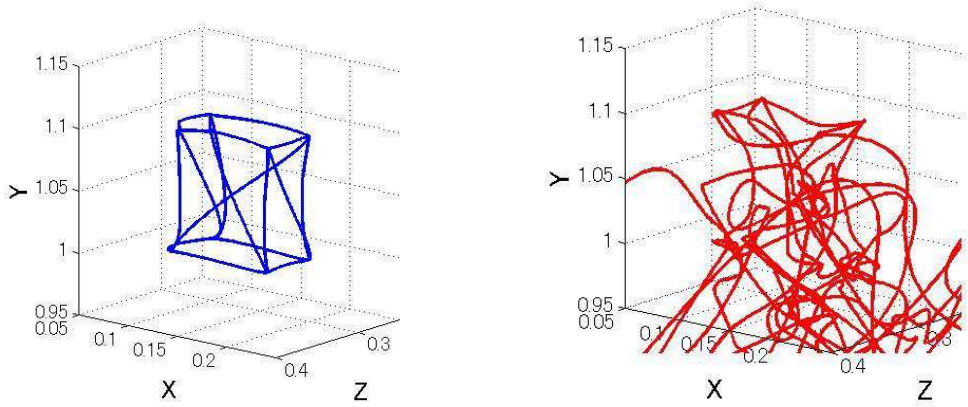
Figure 5.13: Task space test trajectory using IMLE, after training with dataset T_1. The IMLE model learns, in an online manner, to control its end-effector using an unobserved tool, updating its internal parameters as the test trajectory is performed.



(a) IMLE. (b) LWPR.

Figure 5.14: Task space test trajectory using the L-shaped tool (S_3), after the training sequence $T_1 \mapsto T_2 \mapsto T_1 \mapsto T_3$.

cannot physically reach the previous workspace when holding this new tool. Also worth of notice is the incapacity of the LWPR learner to track this test trajectory, even after being trained with a dataset corresponding to the same sensorimotor context. This is probably due to LWPR internal mechanisms making more difficult to forget previous experiments as the number of training points increases. It is instructive to analyse what happens if both algorithms are trained again using the original context dataset T_1: in this situation the training sequence $T_1 \mapsto T_2 \mapsto T_1 \mapsto T_3 \mapsto T_1$ has been presented to IMLE and LWPR learners. In this situation, Figure 5.15 depicts the cube-like test trajectories performed by both algorithms, using the L-shaped tool: the advantage of using a multi-valued learning algorithm to estimate a sensorimotor map in the presence of different contexts should be apparent from the observation of these figures.



(a) IMLE. (b) LWPR.

Figure 5.15: Task space test trajectory using the L-shaped tool (S_3), after the training sequence $T_1 \mapsto T_2 \mapsto T_1 \mapsto T_3 \mapsto T_1$.

As a final note, during the experiments, the IMLE model was allocating around 40 experts to approximate the robot forward kinematics, raising to about 130 and 280 experts after the inclusion of the stick tool and the L-shaped tool, respectively. The number of allocated linear models has a strong influence on the computational burden of the algorithm: a large number of mixture components may indicate some sort of undesired learning overfitting, due to an incorrect initialization of the IMLE model parameters. However, these are reasonably low numbers, considering the dimension of the explored space, that do not introduce any kind of real-time processing constraints.

5.3.2 Experimental Results: Kobian

Kobian (Zecca, Endo, et al., 2008) is a 48 degrees of freedom full humanoid robot, that has been designed to integrate the bipedal walking skill of Wabian (Ogura, Aikawa, et al., 2006) to the emotion expression capabilities of the human-like head robot WE-4 (Miwa, Okuchi, et al., 2002). This robot, depicted in Figure 5.16, has a size similar to that of an average Japanese woman, while its overall weight is 62 kg. Its joints configuration is also illustrated in this figure, distributed in the following manner: 12 joints in the legs, 3 in the waist, 14 in the arms, 8 in the hands, 4 in the neck and 7 in the head.

In the following experiments, the right arm of the Kobian robot is controlled, using 4 of its joints, to position its end-effector in a desired position of the workspace. This end-effector is represented by a visual marker, consisting of a green ball, attached either to the wrist or to the tip of a 35 cm long flexible rubber tube, as represented in Figure 5.17. The end-effector position in the workspace is represented by the visual marker position in the images captured by the cameras installed in Kobian eyes: this position, obtained

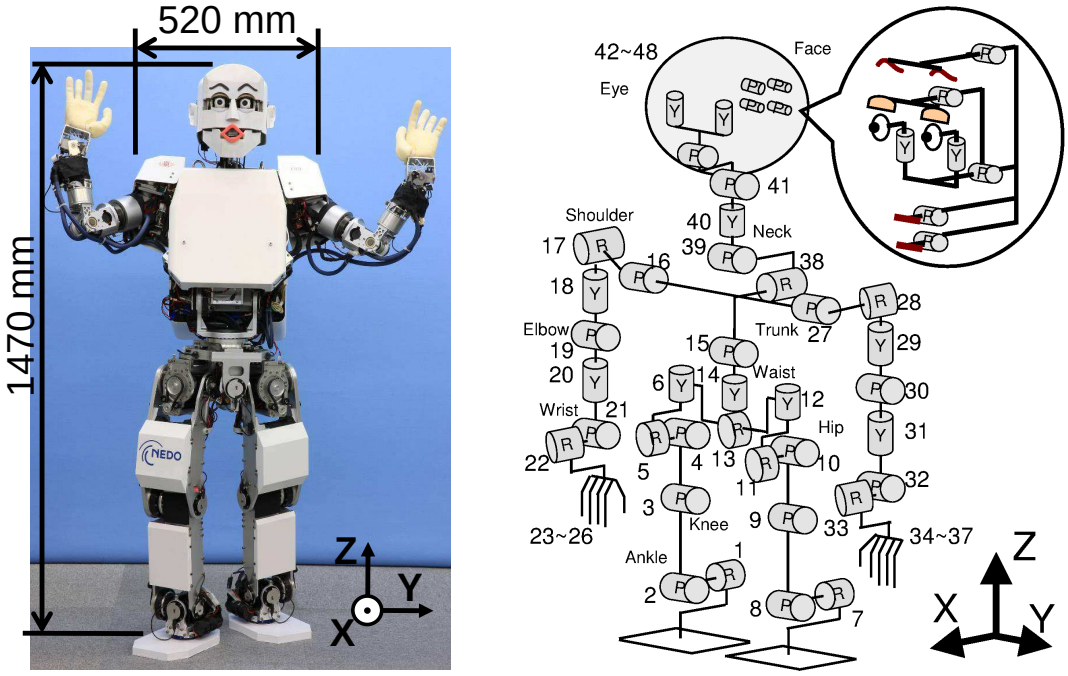


Figure 5.16: The Kobian humanoid robot.

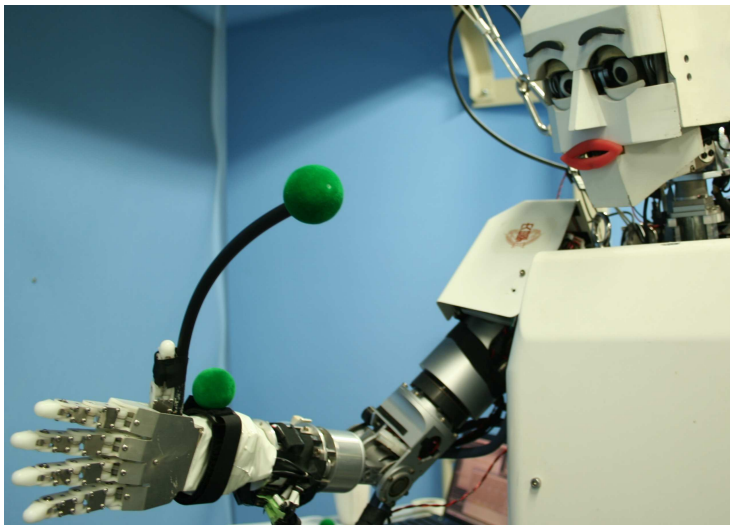


Figure 5.17: The Kobian humanoid robot with the flexible rubber tool. Green balls (*i.e.*, visual markers) are attached to both the wrist and the tip of the tool.

through color based segmentation, is defined as

$$\boldsymbol{x}_v = \begin{bmatrix} u_R \\ v_R \\ u_L - u_R \end{bmatrix} ,$$

where u_R and v_R are respectively the horizontal and vertical coordinates of the centre of the marker in the right camera image, while u_L and v_L are the same coordinates as seen

in the left image. The usual alignment and placement of the cameras in the eyes makes $v_L = v_R$; $u_L - u_R$, the difference in the horizontal position of the marker in both cameras, on the other hand, provides depth information concerning its 3D position in the head reference frame.

The actuated joints, used in the closed-loop control of the task space end-effector position, are the right arm shoulder pitch, yaw and roll rotations, corresponding to the elevation/depression, adduction/abduction and rotation of the arm, and the elbow flexion/extension; these joints are represented by the vector $\boldsymbol{q}_{arm} = [\theta_{sp}\ \theta_{sy}\ \theta_{sr}\ \theta_e]$ and are depicted respectively as joints 16, 17, 18 and 19 in Figure 5.16. Since one of the following experiments also requires moving the Kobian neck, additionally the vector $\boldsymbol{q}_{head} = [\theta_{ny}\ \theta_{np}]$ is also defined, comprising the neck yaw and pitch rotations, corresponding to the Kobian head rotation and elevation/depression, represented by joints 40 and 39 in Figure 5.16). The joint limits are defined in Table 5.2.

	arm				head	
$\boldsymbol{q}^{min}$	$-70°$	$-30°$	$-15°$	$-90°$	$-40°$	$-10°$
$\boldsymbol{q}^{max}$	$-60°$	$-5°$	$15°$	$-40°$	$40°$	$20°$

Table 5.2: Joints limits of the Kobian robot.

As in the previous experiments, performed using the iCub simulator, this experimental setup makes possible the study of the IMLE model multi-valued prediction capabilities in the presence of a switching context scenario. As before, the inclusion or removal of the flexible tool is never signalled to the learning algorithm in any way, neither during motor babbling nor during task space control. Note that the use of a flexible tool, made of soft rubber, makes the kinematic chain extremely difficult to analytically model, as the shape of this tool changes unpredictably with the arm configuration, due to the influence of gravity.

5.3.3 Visual-motor Coordination

The vector $\boldsymbol{x}_v$ provides the end-effector position with respect to the head reference frame, which is not the most convenient frame to be used for visual-motor coordination if the head is allowed to move. An alternative encoding for this end-effector position can be considered if instead it is assumed that the head joints $\boldsymbol{q}_{head}$ are independently moved to bring the end-effector position in the images to a fixed and pre-determined location — in these experiments, the centre of of the right image is considered, *i.e.*, $u_R = 0$ and $v_R = 0$. This *fixation* behaviour can be achieved by generating head joint velocities according to

$$\begin{bmatrix} \dot{\boldsymbol{q}}^0_{head} \\ \dot{\boldsymbol{q}}^1_{head} \end{bmatrix} = \begin{bmatrix} K_0 & 0 \\ 0 & K_1 \end{bmatrix} \begin{bmatrix} \Delta u_R \\ \Delta v_R \end{bmatrix} = - \begin{bmatrix} K_0 & 0 \\ 0 & K_1 \end{bmatrix} \begin{bmatrix} u_R \\ v_R \end{bmatrix} ,$$

where K_0 and K_1 are positive gains, $\dot{\boldsymbol{q}}^0_{head}$ and $\dot{\boldsymbol{q}}^1_{head}$ are the instantaneous velocities of the joints $\boldsymbol{q}^0_{head}$ and $\boldsymbol{q}^1_{head}$ respectively, and $\Delta u_R = u^d_R - u_R$ and $\Delta v_R = v^d_R - v_R$ are the current deviations, measured in the right camera image, from the desired image position for the end-effector, given in this situation by $u^d_R = v^d_R = 0$.

When fixation is achieved, leading to $\Delta u_R = \Delta v_R = 0$, the end-effector position with respect to the Kobian torso can be encoded using the head configuration $\boldsymbol{q}_{head}$ (Metta, Sandini, and Konczak, 1999), using the alternative end-effector position vector

$$\boldsymbol{x}_m = \begin{bmatrix} \boldsymbol{q}^0_{head} \\ \boldsymbol{q}^0_{head} \\ u_L - u_R \end{bmatrix} .$$

Using this latter encoding, the sensorimotor learning problem consists in estimating the kinematics relation $\boldsymbol{x}_m = \boldsymbol{f}(\boldsymbol{q}_{arm})$, a $\mathbb{R}^4$ to $\mathbb{R}^3$ multi-valued relation that depends on the hidden context that indicates if the flexible tool is held by the Kobian robot or not, and where the $\boldsymbol{x}_m$ vector is obtained after fixation of the marker by the head controlling system.

To evaluate the IMLE model learning and prediction capabilities using this physical robot, first a motor babbling phase is conducted, following the same procedure presented in Section 5.3.1. A low level controller was used to move the arm joints to random configurations within the joint limits, and the head controller was used to bring the robot into fixation of the visual marker: after this fixation took place, the corresponding input-output pair consisting of joint values $\boldsymbol{q}_{arm}$ and end-effector position, given by $\boldsymbol{x}_m$, was presented as a training point to the IMLE model. After 500 training points were collected in this manner, the visual marker was moved to the tip of the flexible tool, and 500 more training points were acquired. After that, the marker was repositioned in its original configuration and 300 more points were acquired and used to train and test the IMLE algorithm. The prediction error, evaluated over two independent test sets S_1 and S_2 of 100 points each, corresponding respectively to the two different sensorimotor contexts, is presented in Figure 5.18. This error is evaluated using the test set corresponding to the training phase of the learning algorithm: S_1 for the first 500 training points, then S_2 and finally S_1 again for the remaining 300 points. For illustration purposes it is also shown the prediction results using LWPR, to stress again the competitive handicap of single-valued learning algorithms in these switching context sensorimotor environments.

After the motor babbling phase, the trained IMLE probabilistic model is used to track a star-like pattern, here denoted by the STAR test. A sequence of 16 target visual positions that draws a virtual star in the right camera, shown in Figure 5.19, is fed into the head controller: each reference position is changed every 2 seconds, and so the full test trajectory has a total duration of 32 seconds. Simultaneously, the Kobian arm is actuated, using the gradient projection controller of Liegeois (1977), presented in Section 5.1.2 and

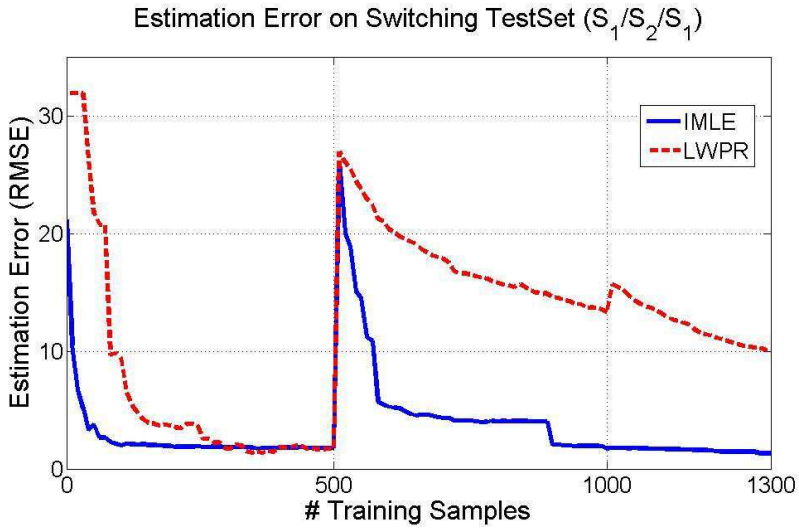


Figure 5.18: Prediction RMSE of the IMLE and LWPR algorithms during Kobian motor babbling. First part of training uses the hand as end-effector, then a flexible tool is introduced at training sample 500 and finally, at training sample 1000, the tool is removed.

also used in this section to control the iCub simulator, in order to follow the trajectory given by $\boldsymbol{x}_d = \boldsymbol{x}_m$, using equation (5.3) and the Jacobian provided by the trained IMLE model.

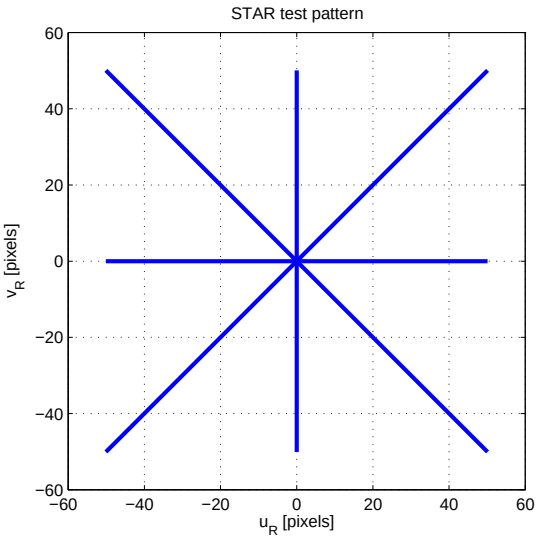
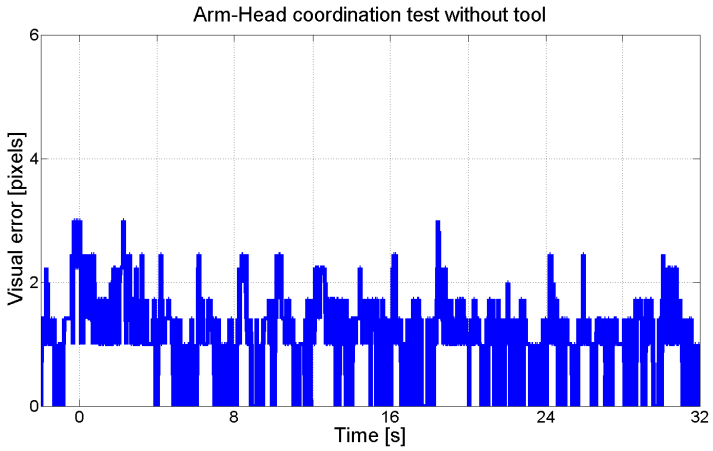
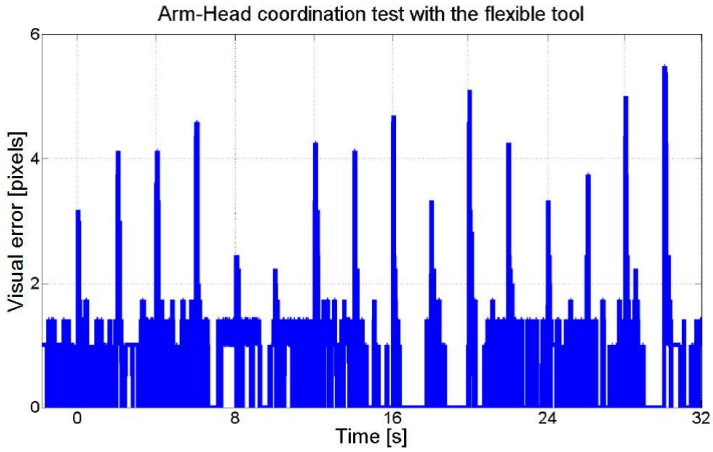


Figure 5.19: STAR pattern used in the Kobian experiments.

Whenever the head starts to move to fixate the next target of the STAR pattern, the arm is thus controlled to follow the head motion, in order to position its end-effector in the centre of the right camera image. During this procedure the third component of $\boldsymbol{x}_d$, the desired end-effector position, is kept at a constant value that keeps the Kobian arm within its workspace. Figure 5.20 shows the visual error, measured in pixels in the right camera image, *i.e.*, the difference between the marker location in the right camera image and the



(a) No tool.



(b) Flexible tool held.

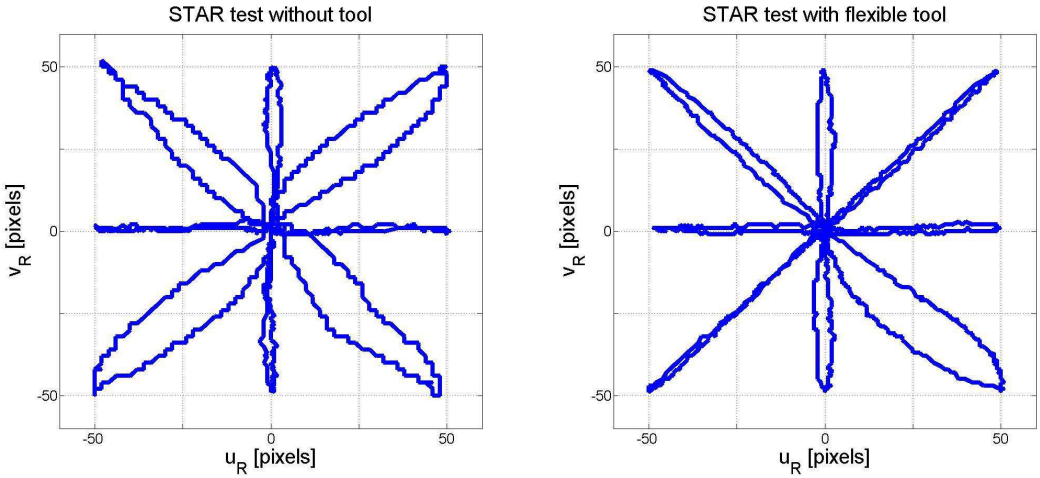
Figure 5.20: End-effector visual position error during the arm-head coordination experiment.

centre of the same image, when the experiment is performed with and without the flexible tool, using the same IMLE learned model. The larger error peaks in Figure 5.20b are due to oscillations of the flexible tool, produced by the higher accelerations in the starting phase of each sub-trajectory of the STAR pattern, but otherwise this error remains low, irrespectively of the sensorimotor context in which the experiment is performed.

5.3.4 Visually Guided Reaching

Next, to further test the performance of the task space control using the learned kinematics, a visually guided reaching experiment is executed, under the aforementioned sensorimotor context switching situation. This time the head is kept fixed, and thus the vector $\boldsymbol{x}_v$ is used to encode the task space end-effector position. The same motor babbling described in the previous section is performed, and after this learning phase, where the IMLE model is trained using the visual marker in Kobian hand and in the tip of the

flexible tool, the STAR pattern is executed, again in both sensorimotor contexts. The position of the marker in the right camera image is depicted in Figure 5.21: again, these



(a) No tool. (b) Flexible tool held.

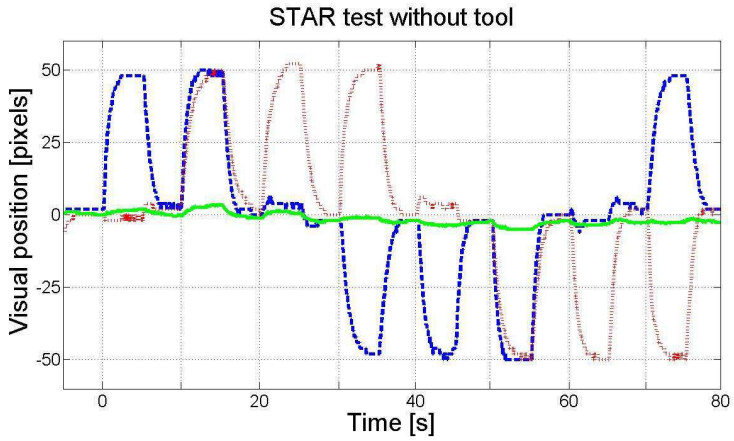
Figure 5.21: End-effector visual position during the visually guided reaching experiment.

are satisfactory results, where the desired trajectory can be performed with the visual marker placed at two different positions in the arm kinematic chain — it is important to stress again that the learner is never signalled whether the flexible tool is being used or not.

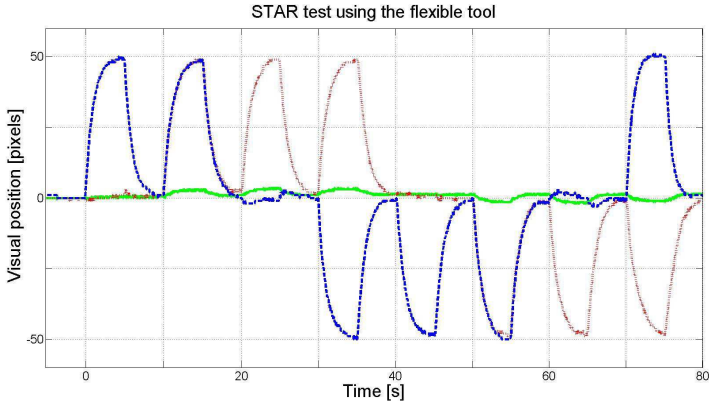
Finally, it is worth of notice that this experiment produces smooth trajectories in both the joint and task spaces: Figure 5.22 depicts the task space trajectories, as a function of time, obtained while performing the STAR test pattern, while Figure 5.23 presents the corresponding joint trajectory profiles.

5.4 Discussion

This chapter presented several applications of the IMLE learning algorithm to task space trajectory control of a robot end-effector, where its inverse and multi-valued prediction capabilities were emphasised. First, it was shown the way its multi-valued prediction capability allows the learner to recover multiple solutions for the inverse kinematics problem, corresponding to different branches of the inverse map, or a sample of the continuum space of inverse solutions when redundant robots are considered. This characteristic, combined with its forward and Jacobian prediction, makes IMLE a highly versatile learning algorithm that can be used to learn the kinematic sensorimotor map online, while simultaneously performing a task space trajectory control that uses the same learned model to choose the commands that should be sent to the robot actuators in the subsequent time step. While other learning algorithms have also been successfully applied to this kind of goal directed exploration (Salaün, Padois, and Sigaud, 2010), a distinctive feature



(a) No tool.

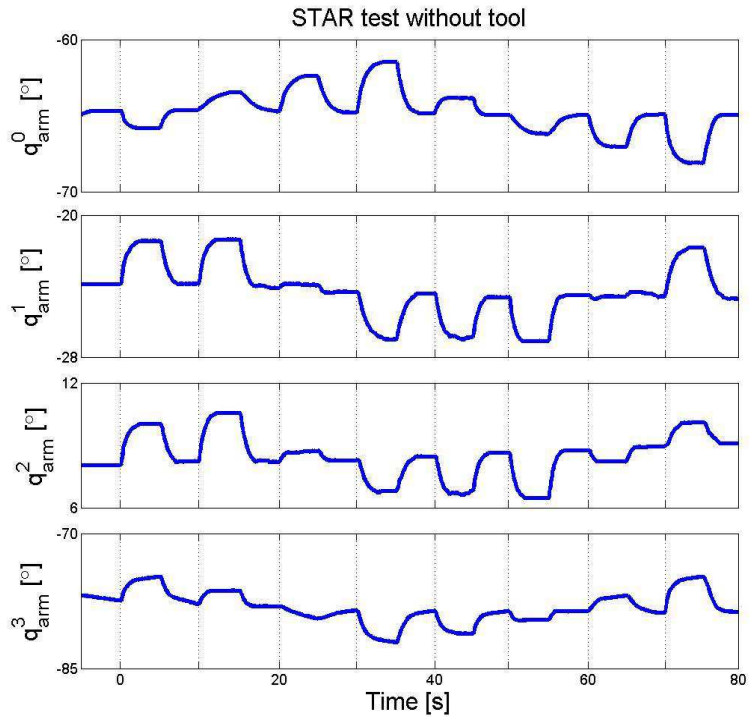


(b) Flexible tool held.

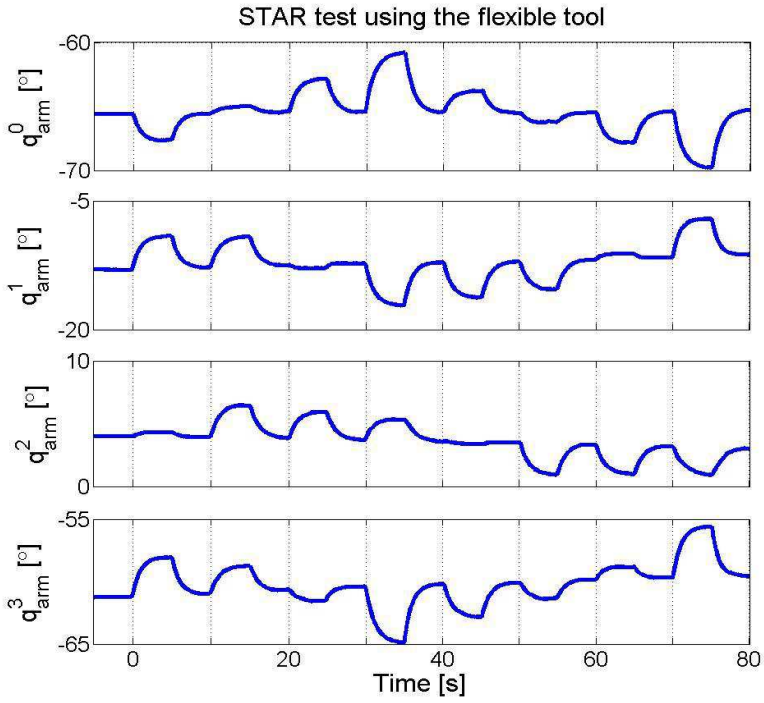
Figure 5.22: End-effector visual position as a function of time.

of the IMLE probabilistic model is its ability to generate forward, inverse and Jacobian predictions from the same learned model. This characteristic can be used to control the task space end-effector position under a pure closed-loop RMRC scheme, updating the IMLE model while executing a desired task, and also to perform an open-loop trajectory planning and tracking when the feedback signal is not present or when it has too much noise or delay. The online nature of the IMLE learning algorithm even makes the use of a composite controller possible, that activates a closed or open-loop controller whenever needed during the operation of the robot: as shown above, this type of controller can seamless switch from closed to open-loop, without any considerable degradation of the task being performed.

Then, the IMLE model was used to learn the the kinematic model of a redundant robot under switching sensorimotor contexts, corresponding to the use of tools of different lengths and shapes. The use of such tools effectively changed the end-effector location and, consequently, the kinematic properties of the robot arm; as it turns out, this is challenging robotic environment for a learning algorithm, as the changes on the sensorimotor



(a) No tool.



(b) Flexible tool held.

Figure 5.23: Actuated arm joints positions as a function of time.

context are not signalled to the learner. This is a situation where the IMLE learning algorithm can truly stand out, since this context switching sensorimotor relation can per-

fectly be modelled by a multi-value relation, where each branch of the relation represents a distinct context. Differently from previous works, the IMLE probabilistic model makes no assumptions about the kinematic properties of the tool; also, no information is given to the robot about the current tool being used, or when a change or removal of the tool is performed. Moreover, the number of different contexts represented by the model doesn't need to be decided *a priori*, but it is automatically determined by the learning algorithm, during the prediction phase, based on the training samples.

The IMLE learning model can also be applied to a diversity of promising applications in the robotics domain, not covered in this chapter. It was shown, in Chapter 4, that IMLE can successfully learn the inverse dynamics relation of a robot: the multi-valued prediction capabilities of the IMLE model make possible, in principle, to learn this model when different loads are applied to the robot being controlled, using a procedure similar to the one presented in Section 5.3. However, some additional considerations must be taken into account in this case, as the sensorimotor map being learned, represented by the relation $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \mapsto \boldsymbol{u}$, has the controlled variable $\boldsymbol{u}$ as output, while the context $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and the causal consequence $\ddot{\boldsymbol{q}}$ are gathered to form the input vector of this map. As a consequence, it is not straightforward to identify the current context, during operation of the robot, based only in the last seen output sample, as done in Section 5.3.

Finally, equations (5.7) and (5.6) deserve a closer look: setting convenient values for diagonal matrices $\boldsymbol{W}_x$ and $\boldsymbol{W}_q$ can in principle lead a hybrid control scheme, where open and closed loop controllers are simultaneously used. In this situation, a balance between closed-loop control, represented by the term $(\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_r)^T \boldsymbol{W}_x (\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_r)$, and open-loop control, represented by the $(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r)^T \boldsymbol{W}_q (\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r)$ term, can be achieved. This balance can even be dynamically changed during online control: when a low confidence is assigned to open-loop trajectories $\dot{\boldsymbol{q}}_r$, planned using the inverse solutions provided by the IMLE algorithm, the value of $\boldsymbol{W}_q$ can be made smaller; conversely, when $\dot{\boldsymbol{x}}_r$ is affected by a large noise level or when the extreme situation of sensor failure occurs, $\boldsymbol{W}_x$ can be reduced, making the open-loop control dominant. Also, the penalty $H(\boldsymbol{q})$ can be modified to consider the uncertainty reduction cost described in Section 3.4: this can potentially result in the extremely interesting behaviour of having the robot actively exploring the null space of the Jacobian of the desired task, in order to improve the IMLE model, while simultaneously performing the same desired task. All these are topics that are being currently investigated.

Chapter 6

Discussion and Concluding Remarks

The infinite mixture of linear experts presented in this dissertation is a probabilistic model that can efficiently deal with nonlinear function approximation in an online, incremental manner, comparable to current state-of-the-art online learning methods. It consists of a collection of linear experts, together with appropriate priors on the parameters being learned and a mechanism that efficiently grows the number of experts when the need to explain outfitted data arises for newly acquired samples. Its training is based on the generalized expectation-maximization algorithm, where the expectation step is extended to allow for incremental updating of the sufficient statistics of the mixture of experts and the maximization step includes the allocation of a new expert each time a training point is poorly explained by the current mixture. Put together, this results in a very fast and scalable online learning algorithm, capable of processing hundreds or thousands of samples per second, coming from continuous streams of data. This is a difficult learning setting: when no knowledge is provided about the characteristics of the function to learn this information must be estimated from a sequence of correlated training data that may correspond to only a small subset of the full input-output space. The IMLE was tested in this kind of situation, for single-valued regression, and showed how it could equal or even surpass current state-of-the-art online learning algorithms, in terms of convergence of prediction error and complexity of the overall model.

However, a distinctive feature of IMLE, when compared to other online supervised learning algorithms, is its ability to deal with multi-valued estimates for the same query data. The applications for such kind of prediction capabilities range from learning forward models of parallel robots to learning switching models, where the function to be approximated can alternate between different configurations over time. This constitutes a even more challenging learning problem: besides the limitations coming from learning online from a stream of data, the expert allocation dilemma comes into play, where it is difficult to distinguish between noise and outliers, in one hand, and a mixture requiring a new component, on the other. Additionally, underlying non-stationary relations make the problem even more difficult to learn.

It was also shown in this dissertation that the same procedure used to obtain a set of multi-valued forward solutions can be applied to inverse queries, making IMLE capable of delivering both forward and inverse multi-valued predictions from the same model, without need for further training. This is a consequence of directly learning a multivariate output relation from $\mathbb{R}^d$ to $\mathbb{R}^D$, instead of a set of D distinct univariate output maps. Such multi-valued learning capability can also prove to be useful when learning discontinuous functions, for which an undesirable prediction smoothing typically occurs in the vicinity of the discontinuities when using standard function approximation algorithms.

Finally, the exploitation of the IMLE model under an active learning context was also described: the same model used to generate forward and inverse predictions can be used to provide an estimate of the expected prediction uncertainty reduction at any given input point, and also makes available the derivatives of such estimate with respect to the input vector. This way, a more parsimonious exploration of the input space can be conducted while achieving the same approximation error in the same space.

As any other learning method, specially under such demanding and unfriendly experimental settings, the IMLE algorithm also presents some limitations, that in part are a consequence of the multi-valued nature of the data and the online training scheme; still, these limitations are potential topics for further work and future improvement:

- The IMLE training algorithm currently lacks a mixture shrinking mechanism, that would be responsible for removing either experts providing wrong predictions or redundant mixture components. The experiments presented in Chapter 4 show that IMLE will activate new experts more and more sparingly as the training progresses, but eventually, after a lifetime of learning, too much experts may become activated, resulting in an increase of computational resources consumption. Moreover, in the event of some episodic outlier bursts, some experts may be activated to represent such erroneous training data. This dissertation does not provide a definite answer to this problem — as discussed is section 3.2.2, growing the mixture under incremental assumptions, for multi-valued data, is a delicate matter, and shrinking the same mixture in a principled way is even more troublesome. The main difficulty here is the fact that there is no simple way to detect experts wrongly activated by an outlier (or a sequence of outliers), as they cannot easily be distinguished from experts originating from a sporadic training on a new branch of the multi-valued function being learned. Such "incorrect" experts can be heuristically detected, for instance, by a strong deviation from the characteristic input length-scale or output noise (strong disagreement between $\mathbf{\Sigma}_j$ and $\bar{\mathbf{\Sigma}}$ or between $\mathbf{\Psi}_j$ and $\bar{\mathbf{\Psi}}$) or by a low support on training data (low accumulation of sufficient statistics after a long period of training). However, removing an expert under this conditions doesn't come without the risk of lowering the likelihood of the training data, since no guarantees exist that such expert doesn't represent the true distribution of the multi-valued function

being learned. In general, choosing an expert to be removed from the mixture — due for instance to capacity limitations imposed to the mixture — is not easy to do without access to the full set of training points, and this hinders the creation of a probabilistically supported shrinking mechanism for the mixture.

- Learning of multi-valued relations rises some issues and questions that are not fully answered in this dissertation. In order to avoid output interference during training, there must be a clear separation between multi-valued branches: when this does not happen there is the risk that the output regions between these branches become populated with inconsequential experts. There is also the time-varying of the training data issue: when the relation to be learned exhibits slow drifts during the training process, the IMLE model can in principle adapt to these variations by means of the forgetting factor in the E-Step update of the sufficient statistics. But what rate of change is allowed for an input-output relation before the IMLE model starts interpreting it as a multi-valued relation? This is an open issue.

- Another delicate issue is the presence of highly correlated or irrelevant data. Feature selection is a very desirable property of a function approximation algorithm, that allows to learn the input subspace that effectively contributes to the output variation: this typically makes the prediction more stable and less influenced by irrelevant or strongly correlated input dimensions. In a Bayesian setup, feature selection can be implemented if adequate priors for loading matrices $\mathbf{\Lambda}_j$ are defined: adopting a Gaussian prior — like IMLE currently does — leads to the ridge regression (Hoerl and Kennard, 1970), while a Laplacian prior induces the LASSO (Tibshirani, 1996). Both these priors involve the choice of a hyperparameter to control the degree of regularization and sparseness. Figueiredo (2003) uses a Jeffrey's prior to overcome the necessity of such hyperparameter. More recently, Ting, D'Souza, et al. (2010) proposed a slightly different formulation of the generative model corresponding to the linear regression performed by each expert, which together with a careful choice of priors for the elements of $\mathbf{\Lambda}_j$ can lead to a fast and efficient high dimensional feature selection and regression. IMLE has a large space for customization by choice of different priors for the mixture parameters, and in principle any of these priors can be integrated into the IMLE model: the key challenge is then the preservation of the current computational complexity of the training algorithm.

- The training procedure is not completely insensitive to variations in the input data distribution: the sampling rate of training points and their input correlation somehow can influence the convergence of the algorithm. An emblematic example for this issue is the situation where the algorithm is trained with the same data point, over and over again: this may result in some mixture parameters converging to the boundaries of the parameter space, a very undesirable situation, particularly

in the beginning of the training procedure, when the values of $\bar{\mathbf{\Sigma}}$ and $\bar{\mathbf{\Psi}}$ have not yet settled. This issued can be circumvented in an ad hoc manner, accepting only consecutive training points whose input parts differ by a minimum amount; overall, this problem has not an easy solution in a probabilistic setting, and plagues many algorithms based on mixture models.

- A correct convergence of the hyperparameters $\bar{\mathbf{\Sigma}}$ and $\bar{\mathbf{\Psi}}$ is also a delicate topic: the IMLE model tries to learn the characteristic input length-scale and output noise of the data, but a particularly bad choice of tuning parameters can result in convergence to values that are not compatible with the true nature of the data, pushing the training procedure to unfavourable local maxima of the likelihood. Even so, as shown in Section 4.1.3, the IMLE model is sufficient robust to such initial choices, as long as they are reasonably within the true values of the input length-scale and output noise of the data. Under a historical perspective, the initial approach to online sensorimotor learning taken on this work was based on an unsupervised mixture of Gaussian models, from which conditional distributions provided the required forward and inverse relations (Lopes and Damas, 2007). However, soon it was realized that learning these relations in an unsupervised way was prone to poor convergence of the training algorithm; additionally, each different sensorimotor map to learn needed a careful choice of initial parameters for the mixture, a frustrating trial-by-error tuning process that went against the robot autonomy that was the final goal of the undergoing work. Mixtures of Factor Analysers (Ghahramani and Hinton, 1996), by performing a local dimensionality reduction that took the problem structure into account, were also considered, but they proved to have the same convergence issues, this time due to the increased effort put on estimating a larger set of latent variables. Under this context, the current solution presented in this thesis, adopting instead a supervised learning paradigm that made the training process more tractable, lead to a huge improvement over the unsupervised learning approach initially followed.

- The existence of a large number of tunable parameters in the IMLE model, together with the aforementioned possibility of poor convergence of the learning algorithm due to a bad choice of the tuning parameters related with $\bar{\mathbf{\Sigma}}$ and $\bar{\mathbf{\Psi}}$, makes this learning architecture somehow difficult to be easily employed by the casual user. Having an "out-of-the-box" behaviour for the software that implements the IMLE model, leading to a satisfactory convergence behaviour without the need to tune any kind of parameters would be an evident improvement for this work. However, it must be kept in mind that some information must be conveyed to the algorithm in the form of tunable parameters, as sensorimotor relations may be very distinct in their nature and their input and output ranges may also be very different: in general, some kind learning bias must be provided.

- Finally, it would be desirable to have a fully Bayesian mechanism for training the mixture, as these kind of methods choose the right model complexity automatically, thus controlling the mixture tendency to overfit. This can be done, for instance, by defining Dirichlet priors over the mixture proportions, under a variational training approach. The main challenge with this kind of approach would be retaining the online and scalability properties of the algorithm: management of the number of active components in a mixture is computationally intensive in a Bayesian paradigm, and the incremental requirements for the algorithm turn the Bayesian solution even harder. Some recent works on stochastic variational inference (Hoffman, Blei, et al., 2013) may point a promising direction worth investigating, but to this date a fully incremental Bayesian approach to the mixture of experts model, that can be trained in real-time in high dimensional spaces, is not yet available.

These issues, nevertheless, should not be considered major weaknesses of the proposed model, as some of them are common to many online and offline learning algorithms, while the others result from the multi-valued assumption for the training data, that significantly complicates the learning process.

The IMLE model provides a general framework for online learning of generic sensorimotor relations, that can be seamlessly applied to a variety of robotic problems that would traditionally need distinct approaches to the learning problem. It has been shown, during this dissertation, how the same learned model can be used to (i) generate forward predictions, useful for simulation and prediction of consequences of actions; (ii) provide inverse predictions, that implicitly define an inverse model that can be used for open-loop control of end-effector task space position; (iii) estimate the Jacobian of the sensorimotor map, a fundamental component for closed-loop velocity and acceleration tracking of task space end-effector position; and (iv) explore the sensorimotor space under an active learning paradigm, as the learned model also makes available not only an uncertainty estimate at each input location, but also an estimate for the expected variation of this model uncertainty, as a consequence of the choice of a particular action. Altogether, the proposed IMLE model for learning and prediction, presented in this dissertation, constitutes a cohesive framework that can hopefully be used as a basic building block of a higher-level architecture, responsible for endowing a robot with a greater level of autonomy.

Appendix A

IMLE Posterior Distributions

The fact that distributions (3.2) are conjugate to data likelihood (3.1) can be used to derive the posterior distribution for the mixture parameters at iteration t, given the current estimates for $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$. Since the Normal-Inverse Wishart is a conjugate prior for the centre and covariance matrix of a multivariate Normal data distribution (Gelman, Carlin, et al., 2004), the posterior distribution for $\boldsymbol{\nu}_j$ and $\boldsymbol{\Sigma}_j$ becomes, for each expert j,

$$\boldsymbol{\nu}_j, \boldsymbol{\Sigma}_j | \boldsymbol{S}^t, \bar{\boldsymbol{\Sigma}} \sim \mathcal{N}W^{-1}(\boldsymbol{\nu}^*,\ n_\nu^*,\ \boldsymbol{\Sigma}^*,\ n_\Sigma^*) \ ,$$

where

$$\begin{aligned}
n_\nu^* &= n_\nu + S_{hj} \ , \\
\boldsymbol{\nu}^* &= \frac{n_\nu \boldsymbol{\nu}_{0j} + \boldsymbol{S}_{hxj}}{n_\nu + S_{hj}} \ , \\
n_\Sigma^* &= n_\Sigma + S_{hj} \ \text{ and} \\
\boldsymbol{\Sigma}^* &= n_\Sigma \bar{\boldsymbol{\Sigma}} + n_\nu \boldsymbol{\nu}_{0j} \boldsymbol{\nu}_{0j}^T + \boldsymbol{S}_{hxxj} - n_\nu^* \boldsymbol{\nu}_j^* \boldsymbol{\nu}_j^{*T} \ .
\end{aligned}$$

Equations (3.9a) and (3.9b) directly follow from the preceding equations, as the maximum value for the Normal-Inverse Wishart distribution is achieved by $\hat{\boldsymbol{\nu}}_j = \boldsymbol{\nu}^*$ and $\hat{\boldsymbol{\Sigma}}_j = \boldsymbol{\Sigma}^*/(n_\Sigma^* + d + 2)$. The predictive distribution $p(\boldsymbol{x}_q | w_j, \boldsymbol{S}^t, \bar{\boldsymbol{\Sigma}})$ can then be obtained if the marginal distribution of $p(\boldsymbol{x}_q, \boldsymbol{\Sigma}_j, \boldsymbol{\nu}_j | w_j, \boldsymbol{S}^t, \bar{\boldsymbol{\Sigma}})$ is taken with respect to unknown parameters $\boldsymbol{\Sigma}_j$ and $\boldsymbol{\nu}_j$, which results in a multivariate t-Student distribution with $n_\Sigma^* - d + 1$ degrees of freedom,

$$\boldsymbol{x}_q | w_j, \boldsymbol{S}^t, \bar{\boldsymbol{\Sigma}} \sim t_{n_\Sigma^* - d + 1} \left(\boldsymbol{\nu}^*, \frac{n_\nu^* + 1}{n_\nu^* (n_\Sigma^* - d + 1)} \boldsymbol{\Sigma}^* \right) \ . \tag{A.1}$$

The Normal-Inverse Gamma distribution is a conjugate prior for the noise and regression coefficients under the standard linear regression likelihood model (O'Hagan and Forster, 1994). A different parametrization of equation (3.1a) can be used to infer the pos-

terior distribution of $\boldsymbol{\Lambda}_j$, $\boldsymbol{\mu}_j$ and $\boldsymbol{\Psi}_j$ at iteration t, by defining $\tilde{\boldsymbol{\Lambda}}_j \equiv [\boldsymbol{\Lambda}_j,\ \bar{\boldsymbol{\mu}}_j]$, where $\bar{\boldsymbol{\mu}}_j \equiv \boldsymbol{\mu}_j - \boldsymbol{\Lambda}_j\boldsymbol{\nu}_j$, and expanding the input vector to accommodate a constant term, $\tilde{\boldsymbol{x}} \equiv [\boldsymbol{x}^T,\ 1]^T$; this makes possible rewriting (3.1a) as $\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ij}, \boldsymbol{\Theta} \sim \mathcal{N}(\tilde{\boldsymbol{\Lambda}}_j\tilde{\boldsymbol{x}}_i, \boldsymbol{\Psi}_j)$ and changing the corresponding priors (3.2c) and (3.2d) accordingly, $\tilde{\boldsymbol{\Lambda}}_j(k)|\boldsymbol{\Psi}_j(k) \sim \mathcal{N}(\tilde{\boldsymbol{\Lambda}}_0(k), \boldsymbol{\Psi}_j(k)\tilde{\boldsymbol{R}}_\Lambda^{-1})$, with $\tilde{\boldsymbol{\Lambda}}_0 = [\boldsymbol{0},\ \boldsymbol{\mu}_{0j}]$ and

$$\tilde{\boldsymbol{R}}_\Lambda^{-1} = \begin{bmatrix} n_\Lambda \mathbf{I} & \boldsymbol{0} \\ \boldsymbol{0} & \cdots n_\mu \end{bmatrix} .$$

The posterior distribution for $(\tilde{\boldsymbol{\Lambda}}_j(k), \boldsymbol{\Psi}_j(k))$ then becomes

$$\tilde{\boldsymbol{\Lambda}}_j(k), \boldsymbol{\Psi}_j(k)|\boldsymbol{S}^t, \bar{\boldsymbol{\Psi}} \sim \mathcal{N}G^{-1}\left(\tilde{\boldsymbol{\Lambda}}^*(k),\ \tilde{\boldsymbol{R}}_\Lambda^*,\ \frac{n_\Psi^*}{2},\ \frac{\boldsymbol{\Psi}^*(k)}{2}\right) , \tag{A.2}$$

where

$$\begin{aligned}
\tilde{\boldsymbol{R}}_\Lambda^* &= (\tilde{\boldsymbol{R}}_\Lambda^{-1} + \tilde{\boldsymbol{S}}_{hxxj})^{-1} , \\
\tilde{\boldsymbol{\Lambda}}^* &= (\tilde{\boldsymbol{S}}_{hyxj} + \tilde{\boldsymbol{\Lambda}}_0\tilde{\boldsymbol{R}}_\Lambda^{-1})\tilde{\boldsymbol{R}}_\Lambda^* = [\boldsymbol{\Lambda}^*,\ \ \bar{\boldsymbol{\mu}}^*] , \\
n_\Psi^* &= n_\Psi + S_{hj} , \\
\boldsymbol{\Psi}^* &= n_\Psi\bar{\boldsymbol{\Psi}} + \operatorname{diag}\left\{\boldsymbol{S}_{hyyj} + \tilde{\boldsymbol{\Lambda}}_0\tilde{\boldsymbol{R}}_\Lambda^{-1}\tilde{\boldsymbol{\Lambda}}_0^T - \tilde{\boldsymbol{\Lambda}}^*(\tilde{\boldsymbol{R}}_\Lambda^*)^{-1}\tilde{\boldsymbol{\Lambda}}^{*T}\right\} ,
\end{aligned}$$

with $\tilde{\boldsymbol{S}}_{hyxj} = [\boldsymbol{S}_{hyxj},\ \ \boldsymbol{S}_{hyj}]$ and

$$\tilde{\boldsymbol{S}}_{hxxj} = \begin{bmatrix} \boldsymbol{S}_{hxxj} & \boldsymbol{S}_{hxj} \\ \boldsymbol{S}_{hxj}^T & S_{hj} \end{bmatrix}$$

accounting for the extended input vector $\tilde{\boldsymbol{x}}$. Noting that $\tilde{\boldsymbol{R}}_\Lambda^*$ can be written as

$$\tilde{\boldsymbol{R}}_\Lambda^* = \begin{bmatrix} \boldsymbol{R}_\Lambda^* & -\boldsymbol{R}_\Lambda^*\frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu} \\ -\frac{\boldsymbol{S}_{hxj}^T}{S_{hj}+n_\mu}\boldsymbol{R}_\Lambda^* & R_{\bar{\boldsymbol{\mu}}}^* \end{bmatrix} , \quad \text{where}$$

$$\boldsymbol{R}_\Lambda^* = \left(n_\Lambda\mathbf{I} + \boldsymbol{S}_{hxxj} - \frac{\boldsymbol{S}_{hxj}\boldsymbol{S}_{hxj}^T}{S_{hj}+n_\mu}\right)^{-1} \ \text{ and } \ R_{\bar{\boldsymbol{\mu}}}^* = \frac{1}{S_{hj}+n_\mu} + \frac{\boldsymbol{S}_{hxj}^T}{S_{hj}+n_\mu}\boldsymbol{R}_\Lambda^*\frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu} ,$$

easily leads to

$$\begin{aligned}
\boldsymbol{\Lambda}^* &= \left(\boldsymbol{S}_{hyxj} - \frac{\boldsymbol{S}_{hyj} + n_\mu\boldsymbol{\mu}_{0j}}{S_{hj}+n_\mu}\boldsymbol{S}_{hxj}^T\right)\boldsymbol{R}_\Lambda^* , \\
\bar{\boldsymbol{\mu}}^* &= \frac{\boldsymbol{S}_{hyj} + n_\mu\boldsymbol{\mu}_{0j}}{S_{hj}+n_\mu} - \boldsymbol{\Lambda}^*\frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu} \qquad \text{and} \\
\boldsymbol{\Psi}^* &= n_\Psi\bar{\boldsymbol{\Psi}} + \operatorname{diag}\left\{\boldsymbol{S}_{hyyj} - \boldsymbol{\Lambda}^*\boldsymbol{S}_{hyxj}^T - \bar{\boldsymbol{\mu}}^*(\boldsymbol{S}_{hyj} + n_\mu\boldsymbol{\mu}_{0j})^T\right\} .
\end{aligned}$$

Equation (A.2) can be split as $p(\tilde{\boldsymbol{\Lambda}}_j(k)|\boldsymbol{\Psi}_j(k), \boldsymbol{S}^t)p(\boldsymbol{\Psi}_j(k)|\boldsymbol{S}^t, \bar{\boldsymbol{\Psi}})$, where

$$\tilde{\boldsymbol{\Lambda}}_j(k)|\boldsymbol{\Psi}_j(k), \boldsymbol{S}^t \sim \mathcal{N}(\tilde{\boldsymbol{\Lambda}}^*(k),\ \boldsymbol{\Psi}_j(k)\tilde{\boldsymbol{R}}^*_\Lambda) \qquad \text{and}$$
$$\boldsymbol{\Psi}_j(k)|\boldsymbol{S}^t, \bar{\boldsymbol{\Psi}} \sim \mathcal{G}^{-1}\left(\frac{n^*_\Psi}{2},\ \frac{\boldsymbol{\Psi}^*(k)}{2}\right) ,$$

and consequently $\boldsymbol{\Lambda}_j(k)$ and $\bar{\boldsymbol{\mu}}_j(k)$ are jointly Normal given $\boldsymbol{\Psi}_j(k)$, with

$$\boldsymbol{\Lambda}_j(k)|\boldsymbol{\Psi}_j(k), \boldsymbol{S}^t \sim \mathcal{N}(\boldsymbol{\Lambda}^*(k),\ \boldsymbol{\Psi}_j(k)\boldsymbol{R}^*_\Lambda) ,$$
$$\bar{\boldsymbol{\mu}}_j(k)|\boldsymbol{\Psi}_j(k), \boldsymbol{S}^t \sim \mathcal{N}(\bar{\boldsymbol{\mu}}^*(k),\ \boldsymbol{\Psi}_j(k)R^*_{\bar{\mu}})$$

and cross-variance equal to $-\boldsymbol{R}^*_\Lambda \frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu}$. As a consequence, $\boldsymbol{\mu}_j(k)$ is also normally distributed given $\boldsymbol{\Psi}_j(k)$, with $\boldsymbol{\mu}_j(k)|\boldsymbol{\Psi}_j(k), \boldsymbol{S}^t \sim \mathcal{N}(\boldsymbol{\mu}^*(k),\ \boldsymbol{\Psi}_j(k)R^*_\mu)$, where

$$\boldsymbol{\mu}^* = \frac{\boldsymbol{S}_{hyj} + n_\mu \boldsymbol{\mu}_{oj}}{S_{hj}+n_\mu} + \boldsymbol{\Lambda}^*\left(\hat{\boldsymbol{\nu}}_j - \frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu}\right) \qquad \text{and}$$
$$R^*_\mu = \frac{1}{S_{hj}+n_\mu} + \left(\frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu} - \hat{\boldsymbol{\nu}}_j\right)^T \boldsymbol{R}^*_\Lambda \left(\frac{\boldsymbol{S}_{hxj}}{S_{hj}+n_\mu} - \hat{\boldsymbol{\nu}}_j\right) .$$

Equations (3.9c–3.9e) arise from the previous posterior distributions, since $\hat{\boldsymbol{\Lambda}}_j = \boldsymbol{\Lambda}^*$, $\hat{\boldsymbol{\mu}}_j = \boldsymbol{\mu}^*$ and $\hat{\boldsymbol{\Psi}}_j = \boldsymbol{\Psi}^*/(n^*_\Psi + 2)$ are the modes of the respective distributions.

In order to obtain $p(\boldsymbol{\Lambda}_j|\boldsymbol{S}^t)$ and $p(\boldsymbol{\mu}_j|\boldsymbol{S}^t)$ Equation (A.2) must be marginalized with respect to $\boldsymbol{\Psi}_j$: a well known result states that the resulting distribution becomes a multivariate t-Student with n^*_Ψ degrees of freedom. When this value is large the distribution can be approximated by a multivariate Normal: for $\boldsymbol{\Lambda}_j$ and $\boldsymbol{\mu}_j$ this results in

$$\boldsymbol{\Lambda}_j(k)|\boldsymbol{S}^t \underset{a}{\sim} \mathcal{N}(\boldsymbol{\Lambda}^*(k),\ \boldsymbol{R}^*_\Lambda\ \boldsymbol{\Psi}^*(k)/n^*_\Psi) \quad \text{and}$$
$$\boldsymbol{\mu}_j(k)|\boldsymbol{S}^t \underset{a}{\sim} \mathcal{N}(\boldsymbol{\mu}^*(k),\ R^*_\mu\ \boldsymbol{\Psi}^*(k)/n^*_\Psi) .$$

Finally, the posterior predictive distribution for $\boldsymbol{y}|\boldsymbol{x}_q$ is obtained by noting that the marginalization of $p(\boldsymbol{y}_i|\boldsymbol{x}_i, w_{ij}, \boldsymbol{\Theta})$ with respect to the parameters $\boldsymbol{\Psi}_j$, $\boldsymbol{\mu}_j$ and $\boldsymbol{\Lambda}_j$ yields again a t-Student distribution, with n^*_Ψ degrees of freedom, mean equal to

$$\hat{\boldsymbol{y}}_j(\boldsymbol{x}_q) = \tilde{\boldsymbol{\Lambda}}^*\tilde{\boldsymbol{x}}_q = \hat{\boldsymbol{\Lambda}}_j(\boldsymbol{x}_q - \hat{\boldsymbol{\nu}}_j) + \hat{\boldsymbol{\mu}}_j \tag{A.3}$$

and variance given by

$$\boldsymbol{R}^y_j(\boldsymbol{x}_q) = (1 + \tilde{\boldsymbol{x}}_q^T \tilde{\boldsymbol{R}}^*_\Lambda \tilde{\boldsymbol{x}}_q)\frac{\boldsymbol{\Psi}^*}{n^*_\Psi} = (1 + \gamma_j(\boldsymbol{x}_q))\frac{\boldsymbol{\Psi}^*}{n^*_\Psi} \approx (1 + \gamma_j(\boldsymbol{x}_q))\hat{\boldsymbol{\Psi}}_j , \tag{A.4}$$

where the factor

$$\gamma_j(\boldsymbol{x}_q) = \frac{1}{S_{hj} + n_\mu} + \left(\boldsymbol{x}_q - \frac{\boldsymbol{S}_{hxj}}{S_{hj} + n_\mu}\right)^T \boldsymbol{R}^*_\Lambda \left(\boldsymbol{x}_q - \frac{\boldsymbol{S}_{hxj}}{S_{hj} + n_\mu}\right) \tag{A.5}$$

reflects the uncertainty on the estimates $\hat{\boldsymbol{\mu}}_j$ and $\hat{\boldsymbol{\Lambda}}_j$ used in the posterior prediction: note that as the training size increases this term vanishes in (A.4), while the fundamental source of noise due to $\boldsymbol{\Psi}_j$ remains. Put together, this results in

$$\boldsymbol{y}|\boldsymbol{x}_q, w_j, \boldsymbol{S}^t, \bar{\boldsymbol{\Psi}} \underset{a}{\sim} \mathcal{N}(\hat{\boldsymbol{y}}_j(\boldsymbol{x}_q),\ \boldsymbol{R}^y_j(\boldsymbol{x}_q)) \ , \tag{A.6}$$

where again the t-Student distribution is approximated to a Normal one, under the assumption of a large value of n^*_Ψ. It may be useful to view this result under a different and equivalent formulation, given by the hierarchical model

$$\boldsymbol{y}|\bar{\boldsymbol{y}}_j, w_j, \boldsymbol{S}^t, \bar{\boldsymbol{\Psi}} \underset{a}{\sim} \mathcal{N}(\bar{\boldsymbol{y}}_j,\ \hat{\boldsymbol{\Psi}}_j) \ , \tag{A.7}$$

$$\bar{\boldsymbol{y}}_j|\boldsymbol{x}_q, \boldsymbol{S}^t, \bar{\boldsymbol{\Psi}} \underset{a}{\sim} \mathcal{N}(\hat{\boldsymbol{y}}_j(\boldsymbol{x}_q),\ \gamma_j(\boldsymbol{x}_q)\hat{\boldsymbol{\Psi}}_j) \ , \tag{A.8}$$

which can be interpreted as a sample point $\boldsymbol{y}(\boldsymbol{x}_q)$ being generated, with noise $\hat{\boldsymbol{\Psi}}_j$, from an unknown mean $\bar{\boldsymbol{y}}_j$, for which posterior distribution (A.8) is available given the current set of sufficient statistics and input query $\boldsymbol{x}_q$. The last equation can also be rewritten as

$$\hat{\boldsymbol{y}}_j|\boldsymbol{x}_q, \boldsymbol{S}^t, \bar{\boldsymbol{\Psi}} \underset{a}{\sim} \mathcal{N}(\bar{\boldsymbol{y}}_j(\boldsymbol{x}_q),\ \gamma_j(\boldsymbol{x}_q)\hat{\boldsymbol{\Psi}}_j) \ , \tag{A.9}$$

which can be interpreted, for current mixture parameters, as a point estimate $\hat{\boldsymbol{y}}$ being generated from a true value $\bar{\boldsymbol{y}}_j(\boldsymbol{x}_q)$ with an uncertainty represented by variance $\gamma_j(\boldsymbol{x}_q)\hat{\boldsymbol{\Psi}}_j$.

Appendix B

IMLE Prediction Derivatives

This appendix presents some useful results concerning the derivatives of the IMLE prediction with respect to the input variable $\boldsymbol{x}$. This prediction is given, for a given solution k found by the clustering procedure presented in Section 3.3.2, by

$$\hat{\boldsymbol{y}}^{(k)}(\boldsymbol{x}) = \hat{\boldsymbol{R}}^{(k)}(\boldsymbol{x}) \sum_j \boldsymbol{R}_j^{-1}(\boldsymbol{x}) \hat{\boldsymbol{y}}_j(\boldsymbol{x}) \ ,$$

where $\hat{\boldsymbol{R}}^{(k)}(\boldsymbol{x})$ is the output expected variance, according to the current IMLE model, for solution k, evaluated at input point $\boldsymbol{x}$. This is given by

$$\hat{\boldsymbol{R}}^{(k)}(\boldsymbol{x}) = \Big(\sum_j \boldsymbol{R}_j^{-1}(\boldsymbol{x}) \Big)^{-1} \ .$$

For readability, unless denoted otherwise, in the remaining text the explicit dependence on the input point $\boldsymbol{x}$ and solution k will be dropped. Also, without loss of generality, all the following results will be derived assuming the output dimension D to be equal to 1: this avoids the notation burden concerning some matrix calculus that would otherwise be required. Note that the matrices $\boldsymbol{R}_j$ are all diagonal, and so the final results obtained in this appendix can be readily extended to the output multivariate case $D > 1$.

The prediction can be expressed as a weighted average of the individual expert predictions:

$$\hat{\boldsymbol{y}} = \sum_j \boldsymbol{W}_j \hat{\boldsymbol{y}}_j \ , \qquad \text{with} \quad \boldsymbol{W}_j = \Big(\sum_k \boldsymbol{R}_k^{-1} \Big)^{-1} \boldsymbol{R}_j^{-1} = \hat{\boldsymbol{R}} \boldsymbol{R}_j^{-1} \ ,$$

where $\hat{\boldsymbol{R}}$ is again the output variance and $\boldsymbol{R}_j$ and $\hat{\boldsymbol{y}}_j$ are respectively expert j variance and prediction at point $\boldsymbol{x}$, given respectively by (3.19) and (A.3),

$$\boldsymbol{R}_j = \varphi_j \hat{\boldsymbol{\Psi}}_j \qquad \text{and} \qquad \hat{\boldsymbol{y}}_j = \hat{\boldsymbol{\Lambda}}_j (\boldsymbol{x} - \hat{\boldsymbol{\nu}}_j) + \hat{\boldsymbol{\mu}}_j \ ,$$

where $\varphi_j = \gamma_j + 1/w_j^y$, with γ_j and w_j^y given respectively by (A.5) and (3.15b).

The derivative of the IMLE prediction $\hat{\boldsymbol{y}}$ with respect to the input variable $\boldsymbol{x}$, that

can be interpreted as the Jacobian of the estimated input-output map, is given by

$$\begin{aligned}\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} &= \frac{d}{d\boldsymbol{x}}\Big[\hat{\boldsymbol{R}}\sum_j \boldsymbol{R}_j^{-1}\hat{\boldsymbol{y}}_j\Big] \\ &= \frac{d\hat{\boldsymbol{R}}}{d\boldsymbol{x}}\sum_j \boldsymbol{R}_j^{-1}\hat{\boldsymbol{y}}_j + \hat{\boldsymbol{R}}\sum_j \frac{d}{d\boldsymbol{x}}\left[\boldsymbol{R}_j^{-1}\hat{\boldsymbol{y}}_j\right] .\end{aligned}$$

The first derivative appearing in the right side of the above equation is the derivative of the output uncertainty with respect to $\boldsymbol{x}$,

$$\begin{aligned}\frac{d\hat{\boldsymbol{R}}}{d\boldsymbol{x}} &= \frac{d}{d\boldsymbol{x}}\Big[\Big(\sum_j \boldsymbol{R}_j^{-1}\Big)^{-1}\Big] \\ &= -\Big(\sum_j \boldsymbol{R}_j^{-1}\Big)^{-2}\sum_j\Big(-\boldsymbol{R}_j^{-2}\frac{d\boldsymbol{R}_j}{d\boldsymbol{x}}\Big) \\ &= \hat{\boldsymbol{R}}^2\sum_j \boldsymbol{R}_j^{-2}\frac{d\boldsymbol{R}_j}{d\boldsymbol{x}} \\ &= \hat{\boldsymbol{R}}\sum_j \hat{\boldsymbol{R}}\boldsymbol{R}_j^{-1}\boldsymbol{R}_j^{-1}\hat{\boldsymbol{\Psi}}_j\frac{d\varphi_j}{d\boldsymbol{x}} ,\end{aligned}$$

and noting that $\boldsymbol{R}_j^{-1}\hat{\boldsymbol{\Psi}}_j = \varphi_j^{-1}\mathbf{I}$ and $\hat{\boldsymbol{R}}\boldsymbol{R}_j^{-1} = \boldsymbol{W}_j$, the above expression becomes

$$\frac{d\hat{\boldsymbol{R}}}{d\boldsymbol{x}} = \hat{\boldsymbol{R}}\sum_j \varphi_j^{-1}\boldsymbol{W}_j\frac{d\varphi_j}{d\boldsymbol{x}} . \tag{B.1}$$

Since

$$\begin{aligned}\frac{d}{d\boldsymbol{x}}\Big[\boldsymbol{R}_j^{-1}\hat{\boldsymbol{y}}_j\Big] &= -\boldsymbol{R}_j^{-2}\frac{d\boldsymbol{R}_j}{d\boldsymbol{x}}\hat{\boldsymbol{y}}_j + \boldsymbol{R}_j^{-1}\frac{d\hat{\boldsymbol{y}}_j}{d\boldsymbol{x}} \\ &= -\boldsymbol{R}_j^{-1}\boldsymbol{R}_j^{-1}\hat{\boldsymbol{\Psi}}_j\hat{\boldsymbol{y}}_j\frac{d\varphi_j}{d\boldsymbol{x}} + \boldsymbol{R}_j^{-1}\frac{d}{d\boldsymbol{x}}\left[\hat{\boldsymbol{\Lambda}}_j(\boldsymbol{x}-\hat{\boldsymbol{\nu}}_j)+\hat{\boldsymbol{\mu}}_j\right] \\ &= -\boldsymbol{R}_j^{-1}\varphi_j^{-1}\hat{\boldsymbol{y}}_j\frac{d\varphi_j}{d\boldsymbol{x}} + \boldsymbol{R}_j^{-1}\hat{\boldsymbol{\Lambda}}_j ,\end{aligned}$$

the Jacobian becomes

$$\begin{aligned}\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} &= \hat{\boldsymbol{R}}\Big(\sum_j \varphi_j^{-1}\boldsymbol{W}_j\frac{d\varphi_j}{d\boldsymbol{x}}\Big)\Big(\sum_j \boldsymbol{R}_j^{-1}\hat{\boldsymbol{y}}_j\Big) + \hat{\boldsymbol{R}}\sum_j \boldsymbol{R}_j^{-1}\Big(\hat{\boldsymbol{\Lambda}}_j - \varphi_j^{-1}\hat{\boldsymbol{y}}_j\frac{d\varphi_j}{d\boldsymbol{x}}\Big) \\ &= \hat{\boldsymbol{y}}\sum_j \varphi_j^{-1}\boldsymbol{W}_j\frac{d\varphi_j}{d\boldsymbol{x}} - \sum_j \varphi_j^{-1}\boldsymbol{W}_j\hat{\boldsymbol{y}}_j\frac{d\varphi_j}{d\boldsymbol{x}} + \sum_j \boldsymbol{W}_j\hat{\boldsymbol{\Lambda}}_j \\ &= \sum_j \boldsymbol{W}_j\hat{\boldsymbol{\Lambda}}_j + \sum_j \varphi_j^{-1}\boldsymbol{W}_j(\hat{\boldsymbol{y}}-\hat{\boldsymbol{y}}_j)\frac{d\varphi_j}{d\boldsymbol{x}} .\end{aligned} \tag{B.2}$$

The derivative in the previous expression is equal to

$$\frac{d\varphi_j}{d\boldsymbol{x}} = \frac{d\gamma_j}{d\boldsymbol{x}} - \left(\frac{1}{w_j^y}\right)^2 \frac{d}{d\boldsymbol{x}} w_j^y \ ,$$

with

$$\frac{d\gamma_j}{d\boldsymbol{x}} = 2\left(\boldsymbol{x} - \frac{\boldsymbol{S}_{hxj}}{S_{hj} + n_\mu}\right)^T \boldsymbol{R}_\Lambda^* \tag{B.3}$$

and

$$\begin{aligned}
\frac{d}{d\boldsymbol{x}} w_j^y &= \frac{d}{d\boldsymbol{x}}\left[\frac{p_j(\boldsymbol{x})}{\sum_k p_k(\boldsymbol{x})}\right] \\
&= \frac{1}{\sum_k p_k(\boldsymbol{x})} \frac{d}{d\boldsymbol{x}} p_j(\boldsymbol{x}) - \frac{p_j(\boldsymbol{x})}{\left(\sum_k p_k(\boldsymbol{x})\right)^2} \sum_k \frac{d}{d\boldsymbol{x}} p_k(\boldsymbol{x}) \\
&= \frac{1}{\sum_k p_k(\boldsymbol{x})}\left(\frac{d}{d\boldsymbol{x}} p_j(\boldsymbol{x}) - w_j^y \sum_k \frac{d}{d\boldsymbol{x}} p_k(\boldsymbol{x})\right) \ ,
\end{aligned}$$

where R_Λ^* is defined in Appendix A and $p_j(x)$ is the probability density for $\boldsymbol{x}$ given that it was generated by expert j, given by (A.1) of the same Appendix. This probability follows a multivariate t-Student distribution,

$$p_j(\boldsymbol{x}) \propto \left[1 + \frac{n_\nu^*}{n_\nu^* + 1}(\boldsymbol{x} - \boldsymbol{\nu}^*)^T (\boldsymbol{\Sigma}^*)^{-1} (\boldsymbol{x} - \boldsymbol{\nu}^*)\right]^{-\frac{n_\Sigma^* + 1}{2}} \ ,$$

and so

$$\begin{aligned}
\frac{d}{d\boldsymbol{x}} p_j(\boldsymbol{x}) &= -\frac{n_\Sigma^* + 1}{2}\left[1 + \frac{n_\nu^*}{n_\nu^* + 1}(\boldsymbol{x} - \boldsymbol{\nu}^*)^T (\boldsymbol{\Sigma}^*)^{-1} (\boldsymbol{x} - \boldsymbol{\nu}^*)\right]^{-1} 2\frac{n_\nu^*}{n_\nu^* + 1}(\boldsymbol{x} - \boldsymbol{\nu}^*)^T (\boldsymbol{\Sigma}^*)^{-1} p_j(\boldsymbol{x}) \\
&= -\boldsymbol{\kappa}_j(\boldsymbol{x}) p_j(\boldsymbol{x}) \ ,
\end{aligned}$$

with

$$\boldsymbol{\kappa}_j(\boldsymbol{x}) = (n_\Sigma^* + 1)\left[\frac{n_\nu^* + 1}{n_\nu^*} + (\boldsymbol{x} - \boldsymbol{\nu}^*)^T (\boldsymbol{\Sigma}^*)^{-1} (\boldsymbol{x} - \boldsymbol{\nu}^*)\right]^{-1} (\boldsymbol{x} - \boldsymbol{\nu}^*)^T (\boldsymbol{\Sigma}^*)^{-1} \tag{B.4}$$

and where n_Σ^*, n_ν^*, and $\boldsymbol{\Sigma}^*$ are defined in Appendix A for each j. Consequently,

$$\begin{aligned}
\frac{d}{d\boldsymbol{x}} w_j^y &= \frac{1}{\sum_k p_k(\boldsymbol{x})}\left(-\boldsymbol{\kappa}_j(\boldsymbol{x}) p_j(\boldsymbol{x}) + w_j^y \sum_k \boldsymbol{\kappa}_k(\boldsymbol{x}) p_k(\boldsymbol{x})\right) \\
&= w_j^y \left(\frac{\sum_k \boldsymbol{\kappa}_k(\boldsymbol{x}) p_k(\boldsymbol{x})}{\sum_k p_k(\boldsymbol{x})} - \boldsymbol{\kappa}_j(\boldsymbol{x})\right) \\
&= w_j^y \left(\sum_k w_k^y \boldsymbol{\kappa}_k(\boldsymbol{x}) - \boldsymbol{\kappa}_j(\boldsymbol{x})\right) \ ,
\end{aligned}$$

and finally equations (B.2) and (B.1) become

$$\frac{d\hat{\boldsymbol{y}}}{d\boldsymbol{x}} = \sum_j \boldsymbol{W}_j \hat{\boldsymbol{\Lambda}}_j + \sum_j \boldsymbol{W}_j (\hat{\boldsymbol{y}} - \hat{\boldsymbol{y}}_j) \boldsymbol{\zeta}_j(\boldsymbol{x}) \qquad \text{and} \tag{B.5a}$$

$$\frac{d\hat{\boldsymbol{R}}}{d\boldsymbol{x}} = \hat{\boldsymbol{R}} \sum_j \boldsymbol{W}_j \boldsymbol{\zeta}_j(\boldsymbol{x}) \;, \tag{B.5b}$$

with

$$\begin{aligned} \boldsymbol{\zeta}_j(\boldsymbol{x}) &= \varphi_j^{-1} \frac{d\varphi_j}{d\boldsymbol{x}} \\ &= \frac{1}{1 + \gamma_j w_j^y} \left(w_j^y \frac{d\gamma_j}{d\boldsymbol{x}} + \boldsymbol{\kappa}_j(\boldsymbol{x}) - \sum_k w_k^y \boldsymbol{\kappa}_k(\boldsymbol{x}) \right) \end{aligned} \tag{B.6}$$

and where $d\gamma_j/d\boldsymbol{x}$ and $\boldsymbol{\kappa}_j(\boldsymbol{x})$ are given by (B.3) and (B.4), respectively.

Appendix C

IMLE Uncertainty Reduction

This appendix derives, in more detail, the calculations needed to obtain equation (3.35) in Chapter 3, here repeated for convenience:

$$\frac{d}{d\tilde{\boldsymbol{x}}}\Delta\mathbb{V}(\tilde{\boldsymbol{x}}) = \boldsymbol{a}^T \cdot \left(\frac{d}{d\tilde{\boldsymbol{x}}}\mathbb{V}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})] - \frac{d}{d\tilde{\boldsymbol{x}}}\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})]\right) .$$

In the following text the dependence on $\tilde{\boldsymbol{x}}$ will be omitted for most quantities, for the sake of notational simplicity, although keeping in mind that most of these quantities depend on the particular value of $\tilde{\boldsymbol{x}}$. The first term on the right side of the above equation can be expressed as

$$\begin{aligned}
\frac{d\mathbb{V}[\hat{\boldsymbol{y}}]}{d\tilde{\boldsymbol{x}}} &= \frac{d}{d\tilde{\boldsymbol{x}}}\sum_j \boldsymbol{W}_j^2\mathbb{V}[\hat{\boldsymbol{y}}_j] \\
&= 2\sum_j \boldsymbol{W}_j\mathbb{V}[\hat{\boldsymbol{y}}_j]\frac{d\boldsymbol{W}_j}{d\tilde{\boldsymbol{x}}} + \sum_j \boldsymbol{W}_j^2\frac{d\mathbb{V}[\hat{\boldsymbol{y}}_j]}{d\tilde{\boldsymbol{x}}} \\
&= 2\sum_j \boldsymbol{W}_j^2\mathbb{V}[\hat{\boldsymbol{y}}_j]\left(\hat{\boldsymbol{R}}^{-1}\frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}} - \boldsymbol{\zeta}_j\right) + \sum_j \boldsymbol{W}_j^2\frac{d\mathbb{V}[\hat{\boldsymbol{y}}_j]}{d\tilde{\boldsymbol{x}}} \\
&= 2\,\hat{\boldsymbol{R}}^{-1}\mathbb{V}[\hat{\boldsymbol{y}}]\frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}} - 2\sum_j \boldsymbol{W}_j^2\mathbb{V}[\hat{\boldsymbol{y}}_j]\boldsymbol{\zeta}_j + \sum_j \boldsymbol{W}_j^2\frac{d\mathbb{V}[\hat{\boldsymbol{y}}_j]}{d\tilde{\boldsymbol{x}}} ,
\end{aligned}$$

where, borrowing some results from Appendix B, the derivative of $\boldsymbol{W}_j$ with respect to $\tilde{\boldsymbol{x}}$ was calculated according to

$$\begin{aligned}
\frac{d\boldsymbol{W}_j}{d\tilde{\boldsymbol{x}}} &= \frac{d}{d\tilde{\boldsymbol{x}}}[\hat{\boldsymbol{R}}\boldsymbol{R}_j^{-1}] \\
&= \frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}}\boldsymbol{R}_j^{-1} + \hat{\boldsymbol{R}}\frac{d\boldsymbol{R}_j^{-1}}{d\tilde{\boldsymbol{x}}} \\
&= \hat{\boldsymbol{R}}\Big(\sum_k \boldsymbol{W}_k\boldsymbol{\zeta}_k\Big)\boldsymbol{R}_j^{-1} - \hat{\boldsymbol{R}}\boldsymbol{R}_j^{-2}\hat{\boldsymbol{\Psi}}_j\frac{d\varphi_j}{d\tilde{\boldsymbol{x}}}
\end{aligned}$$

$$
\begin{aligned}
&= \boldsymbol{W}_j\Big(\sum_k \boldsymbol{W}_k\boldsymbol{\zeta}_k\Big) - \boldsymbol{W}_j\boldsymbol{R}_j^{-1}\hat{\boldsymbol{\Psi}}_j\frac{d\varphi_j}{d\tilde{\boldsymbol{x}}} \\
&= \boldsymbol{W}_j\hat{\boldsymbol{R}}^{-1}\frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}} - \boldsymbol{W}_j\varphi_j^{-1}\frac{d\varphi_j}{d\tilde{\boldsymbol{x}}} \\
&= \boldsymbol{W}_j\left(\hat{\boldsymbol{R}}^{-1}\frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}} - \boldsymbol{\zeta}_j\right) .
\end{aligned}
$$

Obtaining the derivative of $\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]$ with respect to $\tilde{\boldsymbol{x}}$, the predictor variance after the same $\tilde{\boldsymbol{x}}$ has been added to the training set, results in very complex, albeit analytically tractable, derivations, due to all the computations performed when IMLE is trained with a new point $\tilde{\boldsymbol{x}}$. To overcome these difficulties, two approximations are made in the following text. The first one assumes that training the IMLE model with a new input point $\tilde{\boldsymbol{x}}$ does not significantly change the weights $\boldsymbol{W}_j$ used to predict $\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})$. This is equivalent to saying that an input direction is sought that achieves the maximum prediction uncertainty reduction through minimization of individual experts uncertainties, as these quantities are the only other source of uncertainty variation in (3.35) if the weights $\boldsymbol{W}_j$ are assumed to be fixed. As a consequence,

$$
\frac{d\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]}{d\tilde{\boldsymbol{x}}} = 2\,\hat{\boldsymbol{R}}^{-1}\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]\frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}} - 2\sum_j \boldsymbol{W}_j^2\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}_j]\boldsymbol{\zeta}_j + \sum_j \boldsymbol{W}_j^2\frac{d\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}_j]}{d\tilde{\boldsymbol{x}}} ,
$$

and the derivative with respect to $\tilde{\boldsymbol{x}}$ of the uncertainty reduction after training with $\tilde{\boldsymbol{x}}$ becomes

$$
\frac{d}{d\tilde{\boldsymbol{x}}}\Big(\mathbb{V}[\hat{\boldsymbol{y}}]-\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]\Big) = 2\,\hat{\boldsymbol{R}}^{-1}\Big(\mathbb{V}[\hat{\boldsymbol{y}}]-\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]\Big)\frac{d\hat{\boldsymbol{R}}}{d\tilde{\boldsymbol{x}}} - 2\sum_j \boldsymbol{W}_j^2\Big(\mathbb{V}[\hat{\boldsymbol{y}}_j]-\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}_j]\Big)\boldsymbol{\zeta}_j + \sum_j \boldsymbol{W}_j^2\frac{d}{d\tilde{\boldsymbol{x}}}\Big(\mathbb{V}[\hat{\boldsymbol{y}}_j]-\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}_j]\Big) .
$$

Obtaining each expert individual prediction variance at $\tilde{\boldsymbol{x}}$ is straightforward: according to Appendix A,

$$
\mathbb{V}[\hat{\boldsymbol{y}}_j] = \gamma_j\hat{\boldsymbol{\Psi}}_j ,
$$

where

$$
\gamma_j = \frac{1}{S_{hj}} + \left(\tilde{\boldsymbol{x}} - \frac{\boldsymbol{S}_{hxj}}{S_{hj}}\right)^T \boldsymbol{R}_\Lambda^*\left(\tilde{\boldsymbol{x}} - \frac{\boldsymbol{S}_{hxj}}{S_{hj}}\right) ,
$$

with

$$
\begin{aligned}
\boldsymbol{R}_\Lambda^* &= \left(\boldsymbol{S}_{hxxj} - \frac{\boldsymbol{S}_{hxj}\boldsymbol{S}_{hxj}^T}{S_{hj}}\right)^{-1} \\
&= \boldsymbol{S}_{hxxj}^{-1} + \frac{(\boldsymbol{S}_{hxxj}^{-1}\boldsymbol{S}_{hxj})(\boldsymbol{S}_{hxxj}^{-1}\boldsymbol{S}_{hxj})^T}{S_{hj} - \boldsymbol{S}_{hxj}^T\boldsymbol{S}_{hxxj}^{-1}\boldsymbol{S}_{hxj}} .
\end{aligned}
$$

In the above equations the values of n_μ and $n_\Lambda\mathbf{I}$ have been incorporated in sufficient

statistics S_{hj} and $\boldsymbol{S}_{hxxj}$ respectively, to ease the notation in the calculations that follow: hence the apparent difference relative to the same equations presented in Appendix A.

The derivative of expert j uncertainty is

$$\frac{d\gamma_j}{d\tilde{\boldsymbol{x}}} = 2\Big(\tilde{\boldsymbol{x}} - \frac{\boldsymbol{S}_{hxj}}{S_{hj}}\Big)^T \boldsymbol{R}^*_\Lambda \ ,$$

and therefore

$$\frac{d\mathbb{V}[\hat{\boldsymbol{y}}_j]}{d\tilde{\boldsymbol{x}}} = 2\hat{\boldsymbol{\Psi}}_j\Big(\tilde{\boldsymbol{x}} - \frac{\boldsymbol{S}_{hxj}}{S_{hj}}\Big)^T \boldsymbol{R}^*_\Lambda \ .$$

As for $\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]$, it is importance to notice that this quantity depends on the values of $\hat{\boldsymbol{\Psi}}_j$, which are affected by the addition of a new training point $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$. However, the purpose of this active learning scheme is to choose the new training point $\tilde{\boldsymbol{x}}$: the value of $\tilde{\boldsymbol{y}}$ has not yet been observed. It can be estimated, though, using the current mixture state and the prediction mechanism presented in Section 3.3, and thus $\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}]$ must be interpreted as an expected value over values of $\boldsymbol{y}$, as predicted by the IMLE model.

The expected value of $\hat{\boldsymbol{\Psi}}_j$, after including $\tilde{\boldsymbol{x}}$ can be obtained using equations (3.9), after completely updating the sufficient statistics with training point $(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}}))$, where $\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}}) = \mathbb{E}[\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{x}}]$. This requires extensive calculations, that can become somewhat overwhelming when the derivative of the updated value of $\hat{\boldsymbol{\Psi}}_j$ with respect to $\tilde{\boldsymbol{x}}$ is also considered. This motivates the second approximation made herein, where $\hat{\boldsymbol{\Psi}}_j$ is assumed to change much less than γ_j when $(\tilde{\boldsymbol{x}}, \mathbb{E}[\tilde{\boldsymbol{y}}])$ is used to train the IMLE model. This assumption puts the emphasis on searching potential training points that reduce the uncertainty coefficient γ_j, as opposed to reducing the full prediction uncertainty of expert j. For this reason,

$$\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}_j] = \tilde{\gamma}_j \hat{\boldsymbol{\Psi}}_j \ ,$$

with

$$\tilde{\gamma}_j = \frac{1}{\tilde{S}_{hj}} + \Big(\tilde{\boldsymbol{x}} - \frac{\tilde{\boldsymbol{S}}_{hxj}}{\tilde{S}_{hj}}\Big)^T \tilde{\boldsymbol{R}}^*_\Lambda \Big(\tilde{\boldsymbol{x}} - \frac{\tilde{\boldsymbol{S}}_{hxj}}{\tilde{S}_{hj}}\Big) \ ,$$

where now

$$\begin{aligned}\tilde{\boldsymbol{R}}^*_\Lambda &= \left(\tilde{\boldsymbol{S}}_{hxxj} - \frac{\tilde{\boldsymbol{S}}_{hxj}\tilde{\boldsymbol{S}}^T_{hxj}}{\tilde{S}_{hj}}\right)^{-1} \\ &= \tilde{\boldsymbol{S}}^{-1}_{hxxj} + \frac{(\tilde{\boldsymbol{S}}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj})(\tilde{\boldsymbol{S}}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj})^T}{\tilde{S}_{hj} - \tilde{\boldsymbol{S}}^T_{hxj}\tilde{\boldsymbol{S}}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj}} \ .\end{aligned}$$

The updated sufficient statistics of interest, for expert j, after inclusion of $\tilde{\boldsymbol{x}}$ are given

by

$$\begin{aligned}\tilde{S}_{hj} &= \lambda_j S_{hj} + h_j \ , \\ \tilde{\boldsymbol{S}}_{hxj} &= \lambda_j \boldsymbol{S}_{hxj} + h_j \tilde{\boldsymbol{x}}\end{aligned}$$

and

$$\begin{aligned}\tilde{\boldsymbol{S}}^{-1}_{hxxj} &= \left(\lambda_j \boldsymbol{S}_{hxxj} + h_j \tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^T\right)^{-1} \\ &= \frac{1}{\lambda_j}\left(\boldsymbol{S}^{-1}_{hxxj} - \frac{(\boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{x}})(\boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{x}})^T}{\frac{\lambda_j}{h_j} + \tilde{\boldsymbol{x}}^T \boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{x}}}\right) \ ,\end{aligned}$$

where h_j are the responsibilities, calculated in the E-Step, for the fictitious training point $(\tilde{\boldsymbol{x}}, \hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}}))$.

Defining the auxiliary vectors

$$\boldsymbol{f} = \boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{x}}, \qquad\qquad \boldsymbol{g} = \boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj}$$

and the auxiliary scalar quantities

$$\begin{aligned}&a = \tilde{\boldsymbol{x}}^T \boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{x}}^T \boldsymbol{f}, && b = \tilde{\boldsymbol{x}}^T \boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj} = \tilde{\boldsymbol{x}}^T \boldsymbol{g}, && c = \tilde{\boldsymbol{S}}^T_{hxj} \boldsymbol{S}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj} = \tilde{\boldsymbol{S}}_{hxj}\boldsymbol{g}, \\ &d = \frac{\lambda_j}{h_j} + a, && e = 1 - \frac{a}{d}, && i = \frac{b}{d}, \\ &k = c - bi, && m = be\tilde{S}_{hj} - k, && n = \lambda_j \tilde{S}_{hj} - k,\end{aligned}$$

the previous expressions become

$$\begin{aligned}\tilde{\boldsymbol{S}}^{-1}_{hxxj} &= \frac{1}{\lambda_j}\left(\boldsymbol{S}^{-1}_{hxxj} - \frac{\boldsymbol{f}\boldsymbol{f}^T}{d}\right) \ , \\ \tilde{\boldsymbol{S}}^{-1}_{hxxj}\tilde{\boldsymbol{S}}_{hxj} &= \frac{1}{\lambda_j}(\boldsymbol{g} - i\boldsymbol{f})\end{aligned}$$

and

$$\tilde{\boldsymbol{R}}^*_{\boldsymbol{\Lambda}} = \frac{1}{\lambda_j}\left(\boldsymbol{S}^{-1}_{hxxj} - \frac{\boldsymbol{f}\boldsymbol{f}^T}{d} + \frac{1}{n}(\boldsymbol{g} - i\boldsymbol{f})(\boldsymbol{g} - i\boldsymbol{f})^T\right) \ .$$

As a consequence,

$$\begin{aligned}\tilde{\gamma}_j &= \frac{1}{\tilde{S}_{hj}} + \left(\tilde{\boldsymbol{x}} - \frac{\tilde{\boldsymbol{S}}_{hxj}}{\tilde{S}_{hj}}\right)^T \tilde{\boldsymbol{R}}^*_{\boldsymbol{\Lambda}}\left(\tilde{\boldsymbol{x}} - \frac{\tilde{\boldsymbol{S}}_{hxj}}{\tilde{S}_{hj}}\right) \\ &= \frac{1}{\tilde{S}_{hj}} + \frac{1}{\lambda_j}\left(\tilde{\boldsymbol{x}} - \frac{\tilde{\boldsymbol{S}}_{hxj}}{\tilde{S}_{hj}}\right)^T \left(\boldsymbol{S}^{-1}_{hxxj} - \frac{\boldsymbol{f}\boldsymbol{f}^T}{d} + \frac{1}{n}(\boldsymbol{g} - i\boldsymbol{f})(\boldsymbol{g} - i\boldsymbol{f})^T\right)\left(\tilde{\boldsymbol{x}} - \frac{\tilde{\boldsymbol{S}}_{hxj}}{\tilde{S}_{hj}}\right)\end{aligned}$$

$$= \frac{1}{\tilde{S}_{hj}} + \frac{1}{\lambda_j}\left(ae - 2be\frac{1}{\tilde{S}_{hj}} + \left(k + \frac{m^2}{n}\right)\frac{1}{\tilde{S}_{hj}^2}\right) .$$

The derivative of $\tilde{\gamma}_j$ can be obtained by first noting that

$$\begin{aligned}
&\frac{d}{d\tilde{\boldsymbol{x}}}a = 2\boldsymbol{f}^T, && \frac{d}{d\tilde{\boldsymbol{x}}}b = \boldsymbol{g}^T + h\boldsymbol{f}^T,\\
&\frac{d}{d\tilde{\boldsymbol{x}}}c = 2h_j\boldsymbol{g}^T, && \frac{d}{d\tilde{\boldsymbol{x}}}d = 2\boldsymbol{f}^T,\\
&\frac{d}{d\tilde{\boldsymbol{x}}}e = -2\frac{e}{d}\boldsymbol{f}^T, && \frac{d}{d\tilde{\boldsymbol{x}}}i = \frac{1}{d}(\boldsymbol{g}^T + (h_j - 2i)\boldsymbol{f}^T),\\
&\frac{d}{d\tilde{\boldsymbol{x}}}k = 2(h_j - i)(\boldsymbol{g}^T - i\boldsymbol{f}^T), && \frac{d}{d\tilde{\boldsymbol{x}}}n = -2(h_j - i)(\boldsymbol{g}^T - i\boldsymbol{f}^T)
\end{aligned}$$

and

$$\frac{d}{d\tilde{\boldsymbol{x}}}m = (e\tilde{S}_{hj} - 2(h_j - i))\boldsymbol{g}^T + ((h_j - 2i)e\tilde{S}_{hj} + 2(h_j - i)i)\boldsymbol{f}^T ;$$

this results in

$$\begin{aligned}
\frac{d\tilde{\gamma}_j}{d\tilde{\boldsymbol{x}}} &= \frac{1}{\lambda_j}\left[2e^2\boldsymbol{f}^T - \frac{2}{\tilde{S}_{hj}}\left(e(h_j - 2i)\boldsymbol{f}^T + e\boldsymbol{g}^T\right) + \frac{2}{\tilde{S}_{hj}^2}(h_j - i)\left(\boldsymbol{g} - i\boldsymbol{f}^T\right) + \right.\\
&\left. + \frac{2}{\tilde{S}_{hj}^2}\frac{m^2}{n^2}(h_j - i)(\boldsymbol{g}^T - i\boldsymbol{f}^T) + \frac{2}{\tilde{S}_{hj}^2}\frac{m}{n}\left((e\tilde{S}_{hj} - 2(h_j - i))\boldsymbol{g}^T + ((h_j - 2i)e\tilde{S}_{hj} + 2(h_j - i)i)\boldsymbol{f}^T\right)\right]\\
&= \frac{2}{\lambda_j}\left[e^2\boldsymbol{f}^T + \left(\frac{m}{n} - 1\right)\frac{e}{\tilde{S}_{hj}}\left((h_j - 2i)\boldsymbol{f}^T + \boldsymbol{g}^T\right) + \frac{h_j - i}{\tilde{S}_{hj}^2}\left(\frac{m}{n} - 1\right)^2(\boldsymbol{g}^T - i\boldsymbol{f}^T)\right] .
\end{aligned}$$

Defining the additional auxiliary variables

$$o = \frac{1}{\tilde{S}_{hj}}\left(\frac{m}{n} - 1\right) \qquad \text{and} \qquad p = (h_j - i)o^2 ,$$

the above equation finally becomes

$$\frac{d\tilde{\gamma}_j}{d\tilde{\boldsymbol{x}}} = \frac{2}{\lambda_j}\left[(e(e + (h_j - 2i)o) - ip)\,\boldsymbol{f}^T + (eo + p)\,\boldsymbol{g}^T\right] ,$$

which leads to

$$\frac{d\tilde{\mathbb{V}}[\hat{\boldsymbol{y}}_j]}{d\tilde{\boldsymbol{x}}} = \frac{2}{\lambda_j}\hat{\boldsymbol{\Psi}}_j\left[(e(e + (h_j - 2i)o) - ip)\,\boldsymbol{f}^T + (eo + p)\,\boldsymbol{g}^T\right] .$$

Appendix D

Software

The IMLE training and prediction algorithm is fully implemented in C++, in the form of a template-based header-only library, freely available at Damas (2013). Using the IMLE library requires the installation of the Eigen library (Guennebaud, Jacob, and Others, 2013), also a template, header-header only library for linear algebra and vector and matrix manipulation. In addition, some components of the widely used Boost C++ libraries (Boost.org, 2013) are also required, in particular `Boost.Math`, for accessing some statistical functions, and `Boost.Serialization`, to provide a mechanism for saving the current IMLE model in a non-volatile memory for posterior use.

The IMLE library has been tested in a Linux environment, using a recent Ubuntu distributions (12.04 LTS): it has been successfully compiled using the GCC compiler, version 4.6.3. This library was also compiled in the Windows 7 environment, using Visual Studio Express 2012, and, although not tested as thoroughly as in the Linux environment, has proven to perform correctly. To ease the building process of software using the IMLE library across different platforms and operating systems some CMake scripts (Cmake.org, 2013) are also provided.

D.1 IMLE Class Interface

The IMLE class is a template class, that needs the input and output dimensions to be defined before an object can be instantiated. The following code, for instance, creates two IMLE models, `imleObj1` and `imleObj2`, that can be used to learn a $\mathbb{R}^4 \mapsto \mathbb{R}^3$ and a $\mathbb{R}^5 \mapsto \mathbb{R}^2$ input-output map, respectively.

```
#include "imle.hpp"

int main(int argc, char **argv)
{
        const int d = 4;
```

```
        const int D = 3;

        IMLE<d,D> imleObj1;

        IMLE<5, 2> imleObj2;

        return 0;
}
```

The IMLE class provides a structure that comprises all parameters that can be tuned to influence the learning algorithm behaviour, that can be used to initialize an IMLE class instance:

```
IMLE<d,D>::Param param;
param.sigma0 = 2.0;
param.wPsi = 10.0;

IMLE<d,D> imleObj(param);
```

There are many other parameters that can be set, besides `sigma0` and `wPsi` presented in the above example: these parameters are described in Chapter 3. Additionally, structure members `saveOnExit` and `defaultSave` signal if the current state of the IMLE model should be saved when a IMLE object ceases to exist and, in that case, the filename where this state should be stored.

The IMLE class also provides some convenient alias to some of the most used vectors and matrices used to interact with the IMLE model:

```
IMLE<d,D>::Z vec1;
IMLE<d,D>::X vec2;
IMLE<d,D>::ZZ mat1;
IMLE<d,D>::XX mat2;
IMLE<d,D>::XZ mat3;
```

In the above example, `vec1` is an input vector and `vec2` is an output vector, with corresponding dimensions d and D; `mat1` is an input covariance matrix, with dimension $d \times d$, `mat2` is an output covariance matrix, with dimension $D \times D$, and `mat3` is a $D \times d$ matrix that can be used, among other things, to store a Jacobian matrix.

It is worth of notice that, in this code, `Z` denotes an input vector and `X` denotes an output vector, which is not consistent with the notation presented in this dissertation, where $\boldsymbol{x}$ and $\boldsymbol{y}$ denote respectively an input and an output vector. This is due to historical

reasons: in earlier works this was the notation used, by the time the IMLE code was developed.

Multi-valued predictions are stored in data structures given by the following types:

```
IMLE<d,D>::ArrayZ;
IMLE<d,D>::ArrayX;
IMLE<d,D>::ArrayZZ;
IMLE<d,D>::ArrayXX;
IMLE<d,D>::ArrayXZ;
```

Here, `ArrayZ` is just an alias to a `std::vector` of Eigen vectors `Z`; in the same way, the other typedefs are just a shorthand to vectors comprising data types `X`, `ZZ`, `XX` and `XZ`, respectively. In this way, the following code prints to the screen the D-dimensional vector that contains the $4th$ multi-valued prediction given by the IMLE model (details on this topic will be given in the following text):

```
IMLE<d,D> imleObj;
IMLE<d,D>::ArrayX predX;

        (...)

predX = imleObj.getMultiplePredictions();
std::cout << predX[3] << std::endl;
```

D.1.1 Constructors

An IMLE object can be created using three different constructors:

```
IMLE<d,D>::Param prm;

IMLE<d,D> imleObj1;
IMLE<d,D> imleObj2(prm);
IMLE<d,D> imleObj3("savedModel.imle");
```

The first constructor simply creates an empty IMLE object, with a default set of parameters. The second constructor also creates an empty IMLE object, but changes its internal parameters, as given by its argument. The last constructor uses a previously saved IMLE model to initialize the IMLE object state. In these constructors, an optional argument defining the size of memory reserved for the experts comprising the mixture can be defined; the following code, for instance, creates an empty IMLE object and preallocates the memory required to store 512 experts:

```
IMLE<d,D> imleObj(512);
```

Note that this number does not limit the number of experts allowed in the mixture and is only defined for computational efficiency[1]: if, at a given moment, more than 512 experts are needed, the IMLE object simply allocates more memory to the mixture.

Finally, any IMLE object can be reset to an empty state, using the method

```
void IMLE<d,D>::reset();
```

This empties the mixture, resetting the IMLE object to its default state. The only information that is kept is the current set of parameters used by the model.

D.1.2 Parameter handling, serialization and state display

An IMLE object can be fully serialized to a file, for posterior use. This is achieved by the `save()` and `load()` methods:

```
bool save(std::string const &filename);
bool load(std::string const &filename);
```

The IMLE parameters can also be changed and accessed any time, using the `setParameters()` and `getParameters()` methods presented bellow, while `displayParameters()` prints the current set of parameters being used by the IMLE model to a `std::ostream`, using the screen as default stream. Also, for convenience, a `loadParameters()` method allows to load a given set of parameters from a XML configuration file; such file can be generated using the static method `createDefaultParamFile()`, that creates a file with the default set of parameters that can be posteriorly changed by the user.

```
void setParameters(Param const &prm);
Param const &getParameters();
void displayParameters(std::ostream &out = std::cout) const;
bool loadParameters(std::string const &fname);
static void createDefaultParamFile(std::string const &fname);
```

Some information regarding the current state of the mixture can be obtained using the following methods:

```
int getNumberOfExperts();
Z const &getSigma();
X const &getPsi();
Experts const &getExperts();
void modelDisplay(std::ostream &out = std::cout) const;
```

[1]The current version of the IMLE C++ code reserves memory for 1024 experts by default, if such number is not provided.

This way, the current number of linear experts of the mixture and the current values of $\bar{\boldsymbol{\Sigma}}$ and $\bar{\boldsymbol{\Psi}}$ can be inspected at any time during the learning process. Additionally, `getExperts()` provides a way of accessing the current set of experts of the mixture, while `modelDisplay()` outputs the complete IMLE internal state to a `std::ostream` — since this includes each expert internal state, this method can easily lead to some considerable amount of data being output to the stream.

D.1.3 Train and Predict

To incrementally update the mixture with a new input-output sample pair $(\boldsymbol{x}, \boldsymbol{y})$ the following method should be used:

```
void update(Z const &z, X const &x);
```

This triggers the online EM update and the automatic allocation of a new expert if needed, as described in Section 3.2.

A single-valued prediction $\hat{\boldsymbol{y}}$ can be produced at an input query $\boldsymbol{x}$ using the method

```
X const &predict(Z const &z);
```

This prediction is obtained using equation (3.21a) in Section 3.3. After using the `predict()` method some other quantities become available and can be accessed using the following methods:

```
X const &getPrediction();
X const &getPredictionVar();
Scal getPredictionWeight();
XZ const &getPredictionJacobian();
X const &getPredictionErrorReduction();
XZ const &getPredictionErrorReductionDerivative();
```

This way, besides the prediction $\hat{\boldsymbol{y}}$, a user can obtain the corresponding variance estimate at the same query point (equation (3.21b), the Jacobian of the estimated input-output map (equation (3.30) and some quantities pertaining the active learning scheme described in Section 3.4 (equations (3.34) and (3.35)). Additionally, a `getPredictionWeight()` method is available, returning the effective percentage of the mixture that was used to generate the prediction, as given by weights $w^{(k)}$ defined in equation (3.24c). Note this quantity is much more relevant for multi-valued prediction methods described next, as, for single-valued prediction, `getPredictionWeight()` will usually return a value very close to 1; if it returns a value lower than 1, this means that the prediction at query point $\boldsymbol{x}$ is poorly supported by the current mixture, *i.e.*, that there are no linear experts covering the location of the input query. In this case, a non negligible weight is assigned to the

outlier model w_0, as described in Section 3.2.2. Finally, the quantities used for an active learning scheme, $\mathbb{V}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})] - \tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})]$ (equation (3.34)) and $\frac{d}{d\tilde{\boldsymbol{x}}}(\mathbb{V}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})] - \tilde{\mathbb{V}}[\hat{\boldsymbol{y}}(\tilde{\boldsymbol{x}})])$ (equation (3.35)), can be obtained using the methods `getPredictionErrorReduction()` and `getPredictionErrorReductionDerivative()`, respectively.

Another single-valued prediction method is given by

```
X const &predictStrongest(Z const &z);
```

This method effectively performs a multi-valued regression at query point $\boldsymbol{x}$, but only considers the strongest solution obtained, according to the prediction weights described above, and discarding the rest, as described in Section 4.2.1. After `predictStrongest()` is used the same methods presented for single-valued prediction can be used to obtain the variance and the Jacobian of the estimated input-output map, among other quantities of interest.

A multi-valued prediction can be generated using the following method:

```
void predictMultiple(Z const &z);
```

After this prediction command is issued, a set of methods similar to the ones presented in the single-valued prediction can be used to obtain some quantities of interest:

```
ArrayX const &getMultiplePredictions();
ArrayX const &getMultiplePredictionsVar();
ArrayScal const &getMultiplePredictionsWeight();
int getNumberOfSolutionsFound();
ArrayXZ const &getMultiplePredictionsJacobian();
ArrayX const &getMultiplePredictionErrorReduction();
ArrayXZ const &getMultiplePredictionErrorReductionDerivative();
```

Note that, additionally, there is a method that provide the number of forward predictions found using the procedure described in Section 3.3. Finally, inverse predictions can be obtained using the IMLE class method

```
void predictInverse(X const &x);
```

After that, as usual, the inverse predictions and the corresponding variances and weights, together with the number of such predictions found by the clustering procedure, can be accessed using the following methods:

```
ArrayZ const &getInversePredictions();
ArrayZZ const &getInversePredictionsVar();
ArrayScal const &getInversePredictionsWeight();
int getNumberOfInverseSolutionsFound();
```

D.2 YARP Module

To facilitate the application of the IMLE C++ library to sensorimotor learning and control in robotic tasks, a YARP module was also developed in the scope of this work. YARP (Yet Another Robot Platform) is an open source middleware specially developed to support software development on humanoid robotics (Metta, Fitzpatrick, and Natale, 2006; Fitzpatrick, Metta, and Natale, 2008; Fitzpatrick, Metta, and Natale, 2013). In this context, a YARP module is an executable program that can run autonomously and that can communicate with simulators, hardware devices belonging to a robot or another modules.

All configuration of IMLE model parameters can be done resorting to standard YARP configuration files, as described in Fitzpatrick, Metta, and Natale (2013). The IMLE module opens two ports for communication with another modules: one of them is used to train the IMLE model by sending it single input-output pairs of training data, while the second port is bidirectional and is used to provide predictions from the current IMLE model. Two different threads are used for training and prediction, and a software lock is used to ensure integrity of the IMLE model, by only allowing one thread at a time to access and change this model.

D.2.1 Data Port

The data port of the IMLE model accepts training points in the following message form:

$$((x_1\ x_2\ \cdots x_d)\ \ (y_1\ y_2 \cdots y_D))\ ,$$

where d and D are respectively the input and output dimensions.

D.2.2 Query Port

The query port provides predictions to queries sent to it. Such queries start with the word *predict*, followed by an optional prediction mode (single-valued prediction is assumed if no prediction mode is provided) and the input query point, in the form

(predict $(x_1\ x_2\ \cdots x_d)$)
(predict SingleValued $(x_1\ x_2\ \cdots x_d)$)
(predict MultiValued $(x_1\ x_2\ \cdots x_d)$)
(predict Strongest $(x_1\ x_2\ \cdots x_d)$) .

The optional *WithJacobian* string can be specified at the end of the message if the Jacobian at the query point is desired, like in

(predict MultiValued $(x_1\ x_2\ \cdots x_d)$ WithJacobian) .

Inverse prediction follows the same message structure, although now *WithJacobian* is not taken into account:

(predict Inverse (y_1 $y_2 \cdots y_D$)) .

The IMLE module replies to these queries with a message with the following structure,

(s_1 s_2 $\cdots$ s_K) ,

where K is the number of solutions found (K will always be equal to 1 when single-valued prediction is considered) and s_k is the kth solution found by the IMLE prediction algorithm. A forward prediction reply has the form

$$s_k = \text{((Prediction } y_1 \; y_2 \cdots y_D\text{) (Variance } v_1 \; v_2 \cdots v_D\text{) (Weight } w\text{) (Jacobian } J_{11} \; J_{12} \; \cdots \; J_{Dd}\text{))} ,$$

where $v_1 \cdots v_D$ are the components of the diagonal covariance matrix that represents the output uncertainty at query point $x_1 \cdots x_D$, w is the solution weight with respect to the full set of multi-valued predictions and $J_{11} \cdots J_{Dd}$ represent the D by d Jacobian matrix that is only provided if a request for it was made in the corresponding query.

Replies to inverse queries follow more or less the same structure,

$$s_k = \text{((Prediction } x_1 \; x_2 \cdots x_d\text{) (Variance } v_{11} \; v_{12} \cdots v_{dd}\text{) (Weight } w\text{))} ,$$

but now a full covariance matrix is returned and no Jacobian estimate is provided.

Bibliography

Akaike, H (1974). "A new look at the statistical model identification". In: *IEEE Transactions on Automatic Control* 19.6, pp. 716–723.

An, Chae H, Christopher G Atkeson, and John M Hollerbach (1988). *Model-based Control of a Robot Manipulator*. Vol. 16. MIT Press Cambridge, MA.

Andrieu, Christophe, Nando De Freitas, Arnaud Doucet, and Michael I Jordan (2003). "An introduction to MCMC for machine learning". In: *Machine learning* 50, pp. 5–43.

Antoniak, C E (1974). "Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems". In: *The Annals of Statistics* 2.6, pp. 1152–1174.

Asada, M., K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, et al. (2009). "Cognitive Developmental Robotics: A Survey". In: *IEEE Transactions on Autonomous Mental Development* 1.1, pp. 12–34.

Asada, Minoru, Karl F. MacDorman, Hiroshi Ishiguro, and Yasuo Kuniyoshi (2001). "Cognitive developmental robotics as a new paradigm for the design of humanoid robots". In: *Robotics and Autonomous Systems* 37.2-3, pp. 185–193.

Atkeson, Christopher G, Chae H An, and John M Hollerbach (1986). "Estimation of Inertial Parameters of Manipulator Loads and Links". In: *The International Journal of Robotics Research* 5, pp. 101–119.

Atkeson, Christopher G, Andrew W Moore, and Stefan Schaal (1997a). "Locally Weighted Learning". In: *Artificial Intelligence Review* 11.1, pp. 11–73.

Atkeson, Christopher G, Andrew W Moore, and Stefan Schaal (1997b). "Locally Weighted Learning for Control". In: *Artificial Intelligence Review* 11.1, pp. 75–113.

Baillieul, J and D P Martin (1990). "Resolution of kinematic redundancy". In: *Proceedings of Symposia in Applied Mathematics*. Vol. 41, pp. 49–89.

Beal, Matthew J (2003). "Variational Algorithms for Approximate Bayesian Inference". PhD thesis, pp. 1–281.

Bellman, Richard (1957). *Dynamic Programming*. A Rand Corporation research study. Princeton University Press.

Bishop, Christopher M (1994). *Mixture density networks*. Tech. rep.

Bishop, Christopher M (2006). *Pattern Recognition and Machine Learning*. Ed. by M Jordan, J Kleinberg, and B Schölkopf. Information science and statistics. Springer, p. 738. arXiv: 0-387-31073-8.

Bishop, Christopher M and Markus Svensén (2002). "Bayesian hierarchical mixtures of experts". In: *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., pp. 57–64.

Bo, Liefeng, Cristian Sminchisescu, Atul Kanaujia, and Dimitris Metaxas (2008). "Fast algorithms for large scale conditional 3D prediction". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8.

Bocsi, Botond, Duy Nguyen-Tuong, Lehel Csató, Bernhard Scholkopf, and Jan Peters (2011). "Learning inverse kinematics with structured prediction". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 698–703.

Bongard, Josh, Victor Zykov, and Hod Lipson (2006). "Resilient machines through continuous self-modeling." In: *Science (New York, N.Y.)* 314.5802, pp. 1118–21.

Boost.org (2013). *Boost {C}++ Libraries, version 1.53*. URL: `http://www.boost.org` (visited on 10/2013).

Breiman, Leo, Jerome H Friedman, Richard A Olshen, and Charles J Stone (1984). "Classification and Regression Trees". In:

Brooks, Rodney A (1986). "A robust layered control system for a mobile robot". In: *IEEE Journal on Robotics and Automation* 2.1, pp. 14–23.

Brooks, Rodney A (1990). "Elephants don't play chess". In: *Robotics and Autonomous Systems* 6.1-2, pp. 3–15.

Brooks, Rodney A (1991). "Intelligence without representation". In: *Artificial Intelligence* 47.1-3, pp. 139–159.

Broomhead, D and David Lowe (1988). "Multivariable functional interpolation and adaptive networks". In: *Complex Systems* 2, pp. 321–355.

Brouwer, R K (2004). "Feed-forward neural network for one-to-many mappings using fuzzy sets". In: *Neurocomputing* 57, pp. 345–360.

Buhmann, Martin Dietrich (2003). *Radial basis functions: theory and implementations*. Vol. 12. Cambridge university press.

Bullock, D and S Grossberg (1988). "Neural dynamics of planned arm movements: emergent invariants and speed-accuracy properties during trajectory formation." In: *Psychological review* 95.1, pp. 49–90.

Butz, Martin V, Oliver Herbort, and Joachim Hoffmann (2007). "Exploiting redundancy for flexible behavior: unsupervised learning in a modular sensorimotor control architecture." In: *Psychological review* 114.4, pp. 1015–46.

Calinon, Sylvain, Florent Guenter, and Aude Billard (2007). "On Learning, Representing, and Generalizing a Task in a Humanoid Robot". In: *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)* 37.2, pp. 286–298.

Candela, Joaquin Quiñonero and Carl Edward Rasmussen (2005). "A unifying view of sparse approximate Gaussian process regression". In: *The Journal of Machine Learning Research* 6, pp. 1939–1959.

Cao, Lijuan (2003). "Support vector machines experts for time series forecasting". In: *Neurocomputing* 51, pp. 321–339.

Capp, Olivier and Eric Moulines (2009). "Online EM Algorithm for Latent Data Models". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71.3, pp. 593–613.

Carreira-Perpiñán, M Á (2000). "Mode-finding for mixtures of Gaussian distributions". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11, pp. 1318–1323.

Chatzidimitriou, Kyriakos C. and Pericles a. Mitkas (2013). "Adaptive reservoir computing through evolution and learning". In: *Neurocomputing* 103, pp. 198–209.

Cleveland, W S (1979). "Robust locally weighted regression and smoothing scatterplots". In: *Journal of the American Statistical Association* 74, pp. 829–836.

Cmake.org (2013). *Cmake – Cross Platform Make, version 2,8*. URL: `http://www.cmake.org` (visited on 10/2013).

Cohn, David A, Zoubin Ghahramani, and Michael I Jordan (1996). "Active Learning with Statistical Models". In: *Journal of Artificial Intelligence Research* 4.1. Ed. by G Tesauro, D Touretzky, and TEditors Leen, pp. 129–145. arXiv: `960310 [cs]`.

Craig, J J (1989). *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc.

Csató, Lehel and Manfred Opper (2002). "Sparse on-line Gaussian processes". In: *Neural Computation* 14.3, pp. 641–668.

Damas, Bruno (2013). *Infinite Mixture of Linear Experts (IMLE) Library, version 1.9*. URL: `http://users.isr.ist.utl.pt/~bdamas/IMLE/` (visited on 10/2013).

Damas, Bruno, Lorenzo Jamone, and José Santos-Victor (2013). "Open and Closed-Loop Task Space Trajectory Control of Redundant Robots Using Learned Models". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.

Damas, Bruno and José Santos-Victor (2012). "An online algorithm for simultaneously learning forward and inverse kinematics". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1499–1506.

Damas, Bruno and José Santos-Victor (2013). "Online learning of single- and multivalued functions with an infinite mixture of linear experts." In: *Neural computation* 25.11, pp. 3044–91.

Damásio, António (1999). *O Sentimento de Si (English: The feeling of what happens: body and emotion and the making of consciousness)*. Publicações Europa-América.

De Boor, C (1978). *A Practical Guide to Splines*. Vol. 27, p. 325.

Deisenroth, Marc Peter, Carl Edward Rasmussen, and Jan Peters (2009). "Gaussian process dynamic programming". In: *Neurocomputing* 72.7-9, pp. 1508–1524.

Demers, David and Kenneth Kreutz-Delgado (1992). "Learning Global Direct Inverse Kinematics". In: *Advances in Neural Information Processing Systems*. Morgan Kaufmann, pp. 589–595.

Dempster, A P, N M Laird, and D B Rubin (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1, pp. 1–38.

Derksen, Shelley and H J Keselman (1992). "Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables". In: *British Journal of Mathematical and Statistical Psychology* 45.2, pp. 265–282.

Dreyfus, Hubert L (1992). *What computers still can't do: a critique of artificial reason*. The MIT Press.

D'Souza, Aaron, Sethu Vijayakumar, and Stefan Schaal (2001). "Learning inverse kinematics". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems.* Vol. 1. Iros. IEEE, pp. 298–303.

Efron, Bradley, Trevor Hastie, Ian Johnstone, and Robert Tibshirani (2004). "Least Angle Regression". In: *The Annals of Statistics* 32.2, pp. 407–499.

Engel, Yaakov, Shie Mannor, and Ron Meir (2002). "Sparse online greedy support vector regression". In: *European Conference on Machine Learning (ECML)*, pp. 84–96.

Evgeniou, Theodoros, Massimiliano Pontil, and Tomaso Poggio (2000). "Regularization Networks and Support Vector Machines". In: *Advances in Computational Mathematics* 13, pp. 1–50.

Figueiredo, Mário A T (2003). "Adaptive sparseness for supervised learning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.9, pp. 1150–1159.

Figueiredo, Mário A T and Anil K Jain (2002). "Unsupervised learning of finite mixture models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.3, pp. 381–396.

Fitzpatrick, Paul, Giorgio Metta, and Lorenzo Natale (2008). "Towards long-lived robot genes". In: *Robotics and Autonomous Systems* 56.1, pp. 29–45.

Fitzpatrick, Paul, Giorgio Metta, and Lorenzo Natale (2013). *YARP – Yet Another Robot Platform, version 2.3.20*. URL: `http://eris.liralab.it/yarp/` (visited on 10/2013).

Frank, Ildiko E and Jerome H Friedman (1993). "A Statistical View of Some Chemometrics Regression Tools". In: *Technometrics* 35, pp. 109–135.

Friedman, Jerome H (1991). "Multivariate adaptive regression splines". In: *The annals of statistics* 19.1, pp. 1–67.

Friedman, Jerome H and Werner Stuetzle (1981). "Projection Pursuit Regression". In: *Journal of the American Statistical Association* 76, pp. 817–823.

Fritsch, Jürgen, Michael Finke, and Alex Waibel (1997). "Adaptively growing hierarchical mixtures of experts". In: *In Advances in Neural Information Processing Systems 9*. MIT Press Cambridge, MA.

Furnival, George M and Robert W Wilson (1974). "Regressions by leaps and bounds". In: *Technometrics* 16.4, pp. 499–511.

Gelman, A, J B Carlin, H S Stern, and D B Rubin (2004). "Bayesian Data Analysis". In: *Champan and Hall/CRC*.

Geman, Stuart, Elie Bienenstock, and René Doursat (1992). "Neural Networks and the Bias/-Variance Dilemma". In: *Neural Computation* 4.1, pp. 1–58.

Ghahramani, Zoubin (1994). "Solving inverse problems using an EM approach to density estimation". In: *Proceedings of the 1993 Connectionist Models Summer School*. Ed. by Michael C Mozer, P Smolensky, David S Touretzky, J L Elman, and A S Weigend. Erlbaum Associates, pp. 316–323.

Ghahramani, Zoubin (2013). "Bayesian non-parametrics and the probabilistic approach to modelling." In: *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* 371.1984, p. 20110553.

Ghahramani, Zoubin and Geoffrey E Hinton (1996). *The EM Algorithm for Mixtures of Factor Analyzers*. Tech. rep.

Ghahramani, Zoubin and Michael I Jordan (1994). "Supervised Learning from Incomplete Data via an EM approach". In: *Advances in Neural Information Processing Systems 6*, pp. 120–127.

Girosi, Federico, Michael Jones, and Tomaso Poggio (1995). "Regularization Theory and Neural Networks Architectures". In: *Neural Computation* 7.2, pp. 219–269.

Gomes, Ryan, Max Welling, and Pietro Perona (2008). "Incremental learning of nonparametric Bayesian mixture models". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 1–8.

Green, Peter J and Bernard W Silverman (1994). *Nonparametric regression and generalized linear models: a roughness penalty approach*. Chapman & Hall London.

Grollman, D H and O C Jenkins (2010). "Incremental learning of subtasks from unsegmented demonstration". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. IEEE, pp. 261–266.

Grollman, Daniel (2008). *Sparse Online Gaussian Process (SOGP) Library, version 2.0*. URL: `http://cs.brown.edu/people/dang/code.shtml` (visited on 01/2012).

Grollman, Daniel (2009). *Realtime Overlapping Gaussian Expert Regression (ROGER) Library, version 1.5*. URL: `http://cs.brown.edu/people/dang/code.shtml` (visited on 01/2012).

Guennebaud, Gaël, Beno\^it Jacob, and Others (2013). *EIGEN library, version 3.1.4*. URL: `http://eigen.tuxfamily.org` (visited on 10/2013).

Hartmann, Christoph, Joschka Boedecker, Oliver Obst, Shuhei Ikemoto, and Minoru Asada (2012). "Real-Time Inverse Dynamics Learning for Musculoskeletal Robots based on Echo State Gaussian Process Regression". In: *Robotics: Science and Systems*.

Haruno, M, Daniel M Wolpert, and Mitsuo Kawato (2001). "Mosaic model for sensorimotor learning and control". In: *Neural computation* 13.10, pp. 2201–20.

Hastie, Trevor and Clive Loader (1993). "Local Regression: Automatic Kernel Carpentry". In: *Statistical Science* 8.2, pp. 120–129.

Hastie, Trevor and Werner Stuetzle (1989). "Principal Curves". In: *Journal of the American Statistical Association* 84, pp. 502–516.

Hastie, Trevor and Robert Tibshirani (1986). "Generalized additive models". In: *Statistical science* 1.3, pp. 297–310.

Hastie, Trevor, Robert Tibshirani, and Jerome H Friedman (2009). *The Elements of Statistical Learning*. Vol. 27. Springer Series in Statistics 2. New York, NY: Springer New York.

Hemion, Nikolas J., Frank Joublin, and Katharina J. Rohlfing (2012). "Integration of sensorimotor mappings by making use of redundancies". In: *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Herbort, Oliver, Martin V Butz, and Gerulf Pedersen (2010). "The SURE_REACH model for motor learning and control of a redundant arm: from modeling human behavior to applications in robotics". In: *From Motor Learning to Interaction Learning in Robots*. Springer, pp. 85–106.

Hersch, Micha (2009). "Adaptive sensorimotor peripersonal space representation and motor learning for a humanoid robot". PhD thesis.

Hersch, Micha, Eric Sauser, and Aude Billard (2008). "Online Learning of the Body Schema". In: *International Journal of Humanoid Robotics* 05.02, pp. 161–181.

Hinton, Geoffrey E (1991). *Connectionist Symbol Processing*. The MIT Press.

Hinton, Geoffrey E, J L McClelland, and David E Rumelhart (1986). "Distributed representations". In: *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*. MIT Press. The MIT Press, pp. 77–109.

Hoerl, A E and R W Kennard (1970). "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 42.1, pp. 55–67.

Hoffman, Matthew D, David M Blei, Chong Wang, and John Paisley (2013). "Stochastic Variational Inference". In: *Journal of Machine Learning Research* 14, pp. 1303–1347.

Hsu, Ping, John Mauser, and Shankar Sastry (1989). "Dynamic control of redundant manipulators". In: *Journal of Robotic Systems* 6.2, pp. 133–148.

Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew (2006). "Extreme learning machine: Theory and applications". In: *Neurocomputing* 70.1-3, pp. 489–501.

Huang, Mian, Runze Li, and Shaoli Wang (2013). "Nonparametric Mixture of Regression Models." In: *Journal of the American Statistical Association* 108.503.

Jacobs, Robert A, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton (1991). "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1, pp. 79–87.

Jacobs, Robert A, Fengchun Peng, and Martin A Tanner (1997). "A Bayesian Approach to Model Selection in Hierarchical Mixtures-of-Experts Architectures". In: *Neural Networks* 10.2, pp. 231–241.

Jaeger, Herbert (2003). "Adaptive Nonlinear System Identification with Echo State Networks". In: *Advances in Neural Information Processing Systems 15*. Ed. by S Thrun S. Becker and K Obermayer. MIT Press.

Jamone, Lorenzo, Bruno Damas, Nobotsuna Endo, José Santos-Victor, and Atsuo Takanishi (2013a). "Incremental development of multiple tool models for robotic reaching through autonomous exploration". In: *Paladyn Journal of Behaviour Robotics* 3.3, pp. 113–127.

Jamone, Lorenzo, Bruno Damas, José Santos-Victor, and Atsuo Takanishi (2013b). "Online Learning of Humanoid Robot Kinematics Under Switching Tools Contexts". In: *IEEE International Conference on Robotics and Automation*.

Jia, Peng, Junsong Yin, Xinsheng Huang, and Dewen Hu (2009). "Incremental Laplacian eigenmaps by preserving adjacent information between data points". In: *Pattern Recognition Letters* 30.16, pp. 1457–1463.

Jordan, Michael I (1992). "Constrained supervised learning". In: *Journal of Mathematical Psychology* 36.3, pp. 396–425.

Jordan, Michael I and Robert A Jacobs (1994). "Hierarchical Mixtures of Experts and the EM Algorithm". In: *Neural Computation* 6.2, pp. 181–214.

Jordan, Michael I and David E Rumelhart (1992). "Forward Models: Supervised Learning with a Distal Teacher". In: *Cognitive Science* 16.3, pp. 307–354.

Jordan, Michael I and Daniel M Wolpert (1999). "Computational Motor Control". In: *The Cognitive Neurosciences, 2nd edition*. Ed. by Michael S. Gazzaniga. MIT Press.

Joshi, Prashant and Wolfgang Maass (2005). "Movement generation with circuits of spiking neurons". In: *Neural computation* 17.8, pp. 1715–38.

Kanaujia, Atul and Dimitris Metaxas (2006). "Learning Ambiguities Using Bayesian Mixture of Experts". In: *8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pp. 436–440.

Kawato, Mitsuo, Kazunori Furukawa, and R Suzuki (1987). "A hierarchical neural-network model for control and learning of voluntary movement". In: *Biological Cybernetics* 57.3, pp. 169–185.

Kazerounian, K. and Z. Wang (1988). "Global versus Local Optimization in Redundancy Resolution of Robotic Manipulators". In: *The International Journal of Robotics Research* 7.5, pp. 3–12.

Keerthi, Sathiya and Wei Chu (2006). "A matching pursuit approach to sparse Gaussian process regression". In: *Advances in Neural Information Processing Systems 18*. Ed. by Y Weiss, B Schölkopf, and J Platt. Cambridge, MA: MIT Press, pp. 643–650.

Kendall, Maurice G (1957). *A Course in Multivariate Analysis*.

Khatib, Oussama (1987). "A unified approach for motion and force control of robot manipulators: The operational space formulation". In: *IEEE Journal on Robotics and Automation* 3.1, pp. 43–53.

Klanke, Stefan and Sethu Vijayakumar (2009). *Locally Weighted Projection Regression (LWPR) Library, version 1.2.3*. URL: `http://wcms.inf.ed.ac.uk/ipab/slmc/research/software-lwpr` (visited on 01/2012).

Ko, Jonathan and Dieter Fox (2009). "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models". In: *Autonomous Robots* 27.1, pp. 75–90.

Kocijan, J, Roderick Murray-Smith, Carl Edward Rasmussen, and A Girard (2004). "Gaussian process model based predictive control". In: *Proceedings of the 2004 American Control Conference*. Vol. 3. IEEE: Institute of Electrical and Electronics Engineers, pp. 2214–2219.

Kohonen, Teuvo (2001). *Self-Organizing Maps*. Vol. 30. Springer.

Kreinovich, Vladik Y A (1991). "Arbitrary nonlinearity is sufficient to represent all functions by neural networks: A theorem". In: *Neural Networks* 4.3, pp. 381–383.

Kuperstein, M (1988). "Neural model of adaptive hand-eye coordination for single postures". In: *Science* 239.4845, pp. 1308–1311.

Lampert, Christoph H and Matthew B Blaschko (2009). "Structured prediction by joint kernel support estimation". In: *Machine Learning* 77.2-3, pp. 249–269.

Law, Martin H and Anil K Jain (2006). "Incremental nonlinear dimensionality reduction by manifold learning". In: *IEEE transactions on pattern analysis and machine intelligence* 28.3, pp. 377–91.

Lázaro-Gredilla, Miguel (2010). "Sparse Gaussian Processes for Large-Scale Machine Learning". PhD thesis.

Lee, K W and T Lee (2001). "Design of neural networks for multi-value regression". In: *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*. Vol. 1. IEEE. IEEE, pp. 93–98.

Li, Haifeng, Keshu Zhang, and Tao Jiang (2005). "The regularized EM algorithm". In: *Proceedings of the national conference on artificial intelligence (AAAI)*. Vol. 20. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, p. 807.

Li, Housen, Hao Jiang, Roberto Barrio, Xiangke Liao, Lizhi Cheng, et al. (2011). "Incremental manifold learning by spectral embedding methods". In: *Pattern Recognition Letters* 32.10, pp. 1447–1455.

Liegeois, Alain (1977). "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 7.12, pp. 868–871.

Lima, Clodoaldo A M, André L V Coelho, and Fernando J Von Zuben (2007). "Hybridizing mixtures of experts with support vector machines: Investigation into nonlinear dynamic systems identification". In: *Information Sciences* 177.10, pp. 2049–2074.

Ljung, Lennart (2002). "Recursive identification algorithms". In: *Circuits, Systems, and Signal Processing* 21.1, pp. 57–68.

Loader, Catherine (2012). "Smoothing: Local regression techniques". In: *Handbook of Computational Statistics*. Springer, pp. 571–596.

Loader, Clive (1999). *Local Regression and Likelihood*. Statistics and Computing. Springer.

Lopes, Manuel and Bruno Damas (2007). "A learning framework for generic sensory-motor maps". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1533–1538.

Lopes, Manuel and José Santos-Victor (2007). "A developmental roadmap for learning by imitation in robots". In: *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society* 37.2, pp. 308–21.

Lungarella, Max, Giorgio Metta, Rolf Pfeifer, and Giulio Sandini (2003). "Developmental robotics: a survey". In: *Connection Science* 15.4, pp. 151–190.

Ma, Junshui, James Theiler, and Simon Perkins (2003). "Accurate on-line support vector regression." In: *Neural computation* 15.11, pp. 2683–703.

MacKay, David (1995). "Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks". In: *Network: Computation in Neural Systems* 6.3, pp. 469–505.

MacKay, David (2003). *Information theory, inference and learning algorithms*.

Martin, D.P., J. Baillieul, and J.M. Hollerbach (1989). "Resolution of kinematic redundancy using optimization techniques". In: *IEEE Transactions on Robotics and Automation* 5.4, pp. 529–533.

McCulloch, Warren S. and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133.

McLachlan, G and D Peel (2000). "Finite Mixture Models". In:

Meeds, Edward and Simon Osindero (2006). "An Alternative Infinite Mixture Of Gaussian Process Experts". In: *Advances in Neural Information Processing Systems 18*. Ed. by Y Weiss, B Schölkopf, and J Platt. Cambridge, MA: MIT Press, pp. 883–890.

Merlet, J P (2006). *Parallel Robots*. Springer-Verlag New York Inc.

Metta, G, G Sandini, and J Konczak (1999). "A developmental approach to visually-guided reaching in artificial systems." In: *Neural networks : the official journal of the International Neural Network Society* 12.10, pp. 1413–1427.

Metta, Giorgio, Paul Fitzpatrick, and Lorenzo Natale (2006). "YARP: Yet Another Robot Platform". In: *International Journal of Advanced Robotic Systems* 3.1, p. 1.

Metta, Giorgio, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, et al. (2010). "The iCub humanoid robot: an open-systems platform for research in cognitive development." In: *Neural networks : the official journal of the International Neural Network Society* 23.8-9, pp. 1125–34.

Miall, R Chris and Daniel M Wolpert (1996). "Forward Models for Physiological Motor Control". In: *Neural Networks* 9.8, pp. 1265–1279.

Miller, W T (1987). "Sensor-based control of robotic manipulators using a general learning algorithm". In: *IEEE Journal on Robotics and Automation* 3.2, pp. 157–165.

Miller, W T (1989). "Real-time application of neural networks for sensor-based control of robots with vision". In: *IEEE Transactions on Systems, Man, and Cybernetics* 19.4, pp. 825–831.

Mitchell, Tom (1997). *Machine Learning*. McGraw Hill, p. 432.

Miwa, Hiroyasu, Tetsuya Okuchi, Hideaki Takanobu, and Atsuo Takanishi (2002). "Development of a new human-like head robot WE-4". In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. IEEE, pp. 2443–2448.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*.

Nabeshima, Cota, Yasuo Kuniyoshi, and Max Lungarella (2006). "Adaptive body schema for robotic tool-use". In: *Advanced Robotics* 20.10, pp. 1105–1126.

Nadaraya, E A (1964). "On Estimating Regression". In: *Theory of Probability & Its Applications* 9.1, pp. 141–142.

Nakamura, Y and H Hanafusa (1986). "Inverse kinematic solutions with singularity robustness for robot manipulator control". In: *ASME Journal of Dynamic Systems, Measurement, and Control* 108.3, pp. 163–171.

Nakamura, Y and H Hanafusa (1987). "Optimal Redundancy Control of Robot Manipulators". In: *The International Journal of Robotics Research* 6.1, pp. 32–42.

Nakanishi, Jun, R Cory, Michael Mistry, Jan Peters, and Stefan Schaal (2008). "Operational Space Control: A Theoretical and Empirical Comparison". In: *The International Journal of Robotics Research* 27.6, pp. 737–757.

Narendra, K.S. and J Balakrishnan (1997). "Adaptive control using multiple models". In: *IEEE Transactions on Automatic Control* 42.2, pp. 171–187.

Natale, Lorenzo, Francesco Nori, Giorgio Metta, and Matteo Fumagalli (2013). "The iCub platform: a tool for studying intrinsically motivated learning". In: *Intrinsically Motivated Learning in Natural and Artificial Systems*. Ed. by Gianluca Baldassarre and Marco Mirolli. Springer, pp. 433–458.

Natschläger, Thomas, Wolfgang Maass, and Henry Markram (2002). "The "Liquid Computer": A Novel Strategy for Real-Time Computing on Time Series". In: *Special Issue on Foundations of Information Processing of TELEMATIK* 8.1, pp. 39–43.

Neal, Radford M (1996). *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics Series. Springer-Verlag.

Neal, Radford M and Geoffrey E Hinton (1999). "A view of the EM algorithm that justifies incremental, sparse, and other variants". In: *Learning in graphical models*, pp. 355–368.

Nguyen-Tuong, Duy and Jan Peters (2010). "Using model knowledge for learning inverse dynamics". In: *IEEE International Conference on Robotics and Automation*, pp. 2677–2682.

Nguyen-Tuong, Duy and Jan Peters (2011a). "Incremental online sparsification for model learning in real-time robot control". In: *Neurocomputing* 74.11, pp. 1859–1867.

Nguyen-Tuong, Duy and Jan Peters (2011b). "Model learning for robot control: a survey". In: *Cognitive processing* 12.4, pp. 319–40.

Nguyen-Tuong, Duy and Jan Peters (2012). "Online Kernel-Based Learning for Task-Space Tracking Robot Control". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.9, pp. 1417–1425.

Nguyen-Tuong, Duy, Matthias Seeger, and Jan Peters (2009a). "Local Gaussian Process Regression for Real Time Online Model Learning". In: *Advances in Neural Information Processing Systems 21*. Ed. by D Koller, D Schuurmans, Y Bengio, and L Bottou, pp. 1193–1200.

Nguyen-Tuong, Duy, Matthias Seeger, and Jan Peters (2009b). "Model Learning with Local Gaussian Process Regression". In: *Advanced Robotics* 23.15, pp. 2015–2034.

Ogura, Yu, Hiroyuki Aikawa, Kazushi Shimomura, Akitoshi Morishima, Hun-ok Lim, et al. (2006). "Development of a new humanoid robot WABIAN-2". In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, pp. 76–81.

O'Hagan, A and J Forster (1994). *Kendall's Advanced Theory of Statistics: Volume 2B; Bayesian Inference*. Halsted Press.

Park, Trevor and George Casella (2008). "The Bayesian Lasso". In: *Journal of the American Statistical Association* 103.482, pp. 681–686.

Parmiggiani, Alberto, Marco Maggiali, Lorenzo Natale, Francesco Nori, Alexander Schmitz, et al. (2012). "The design of the iCub humanoid robot". In: *International Journal of Humanoid Robotics* 9.04.

Patino, H D, R Carelli, and B R Kuchen (2002). "Neural networks for advanced control of robot manipulators". In: *IEEE Transactions on Neural Networks* 13.2, pp. 343–54.

Payne, V G and L Isaacs (2001). *Human Motor Development: A Lifespan Approach*. McGraw-Hill Higher Education.

Pearson, K (1901). "On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine* 2, pp. 559–572.

Peiper, Donald Lee (1968). "The kinematics of manipulators under computer control". PhD thesis.

Pelossof, R, A Miller, P Allen, and T Jebara (2004). "An SVM learning approach to robotic grasping". In: *IEEE International Conference on Robotics and Automation*. Vol. Vol.4, pp. 3512–3518.

Peters, Jan, Michael Mistry, Firdaus Udwadia, R. Cory, J. Nakanishi, et al. (2005). "A unifying methodology for the control of robotic systems". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1. IEEE, pp. 1824–1831.

Peters, Jan and Stefan Schaal (2006). "Learning Operational Space Control". In: *Proceedings of Robotics: Science and Systems*. Philadelphia, USA: The MIT Press.

Peters, Jan and Stefan Schaal (2008). "Learning to Control in Operational Space". In: *The International Journal of Robotics Research* 27.2, pp. 197–212.

Petkos, Georgios and Sethu Vijayakumar (2007). "Context Estimation and Learning Control through Latent Variable Extraction: From discrete to continuous contexts". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. IEEE, pp. 2117–2123.

Poggio, Tomaso and Federico Girosi (1990). "Regularization algorithms for learning that are equivalent to multilayer networks". In: *Science (New York, N.Y.)* 247.4945, pp. 978–82.

Qin, Chao and Miguel A Carreira-Perpinan (2008). "Trajectory inverse kinematics by conditional density modes". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. IEEE, pp. 1979–1986.

Rasmussen, Carl Edward (2000). "The infinite Gaussian mixture model". In: *Advances in Neural Information Processing Systems 12* 12, pp. 554–560.

Rasmussen, Carl Edward and Zoubin Ghahramani (2002). "Infinite Mixtures of Gaussian Process Experts". In: *Advances in Neural Information Processing Systems 14*, pp. 881–888.

Rasmussen, Carl Edward and Hannes Nickisch (2010). *Gaussian Processes for Machine Learning (GPML) Matlab Code, version 3.1*. URL: `http://www.gaussianprocess.org/gpml/code/matlab/doc/` (visited on 01/2012).

Rasmussen, Carl Edward and Christopher K I Williams (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

Ray, Surajit and Bruce G Lindsay (2005). "The topography of multivariate normal mixtures". In: *The Annals of Statistics* 33.5, pp. 2042–2065.

Reinhart, Rene Felix and Jochen Jakob Steil (2009). "Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot iCub". In: *9th IEEE-RAS International Conference on Humanoid Robots*. Ieee, pp. 323–330.

Reinhart, Rene Felix and Jochen Jakob Steil (2011). "Neural learning and dynamical selection of redundant solutions for inverse kinematic control". In: *2011 11th IEEE-RAS International Conference on Humanoid Robots*. IEEE. IEEE, pp. 564–569.

Reinhart, Rene Felix and Jochen Jakob Steil (2012). "Learning whole upper body control with dynamic redundancy resolution in coupled associative radial basis function networks". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1487–1492.

Rissanen, J. (1978). "Modeling by shortest data description". In: *Automatica* 14.5, pp. 465–471.

Rolf, Matthias, Jochen J. Steil, and Michael Gienger (2010). "Learning Flexible Full Body Kinematics for Humanoid Tool Use". In: *International Conference on Emerging Security Technologies*, pp. 171–176.

Rosenblatt, F (1962). *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books.

Rottmann, A and Wolfram Burgard (2009). "Adaptive autonomous control using online value iteration with gaussian processes". In: *IEEE International Conference on Robotics and Automation*, pp. 2106–2111.

Roweis, Sam (1998). "EM algorithms for PCA and SPCA". In: *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*. MIT Press, pp. 626–632.

Roweis, Sam and Lawrence Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding." In: *Science (New York, N.Y.)* 290.5500, pp. 2323–6.

Rumelhart, David E, Geoffrey E Hinton, and R J Williams (1986). "Learning internal representations by error propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Ed. by David E Rumelhart and J McClelland. Vol. 1. The MIT Press, Cambridge, MA, pp. 318–362.

Russell, Stuart and Peter Norvig (1995). *Artificial Intelligence: a Modern Approach*. Prentice-Hall International (UK).

Saito, Kazumi and Ryohei Nakano (1996). "A constructive learning algorithm for an HME". In: *Proceedings of International Conference on Neural Networks (ICNN'96)*. Vol. 2. IEEE, pp. 1268–1273.

Salaün, Camille, Vincent Padois, and Olivier Sigaud (2010). "Learning Forward Models for the Operational Space Control of Redundant Robots". In: *From Motor Learning to Interaction Learning in Robots*. Ed. by Olivier Sigaud and Jan Peters. Vol. 264. Springer Berlin Heidelberg, pp. 169–192.

Sato, Masa-aki (2001). "Online Model Selection Based on the Variational Bayes". In: *Neural Computation* 13.7, pp. 1649–1681.

Sato, Masa-aki and Shin Ishii (2000). "On-line EM Algorithm for the Normalized Gaussian Network". In: *Neural Computation* 12.2, pp. 407–432.

Schaal, S and C G Atkeson (1998). "Constructive Incremental Learning from Only Local Information". In: *Neural Computation* 10.8, pp. 2047–2084.

Schaal, S, S Vijayakumar, and C G Atkeson (1998). "Local dimensionality reduction". In: *Advances in Neural Information Processing Systems*, pp. 633–639.

Schaal, Stefan and Christopher G Atkeson (1993). "Assessing the Quality of Learned Local Models". In: *Advances in Neural Information Processing Systems 6*. Ed. by J D Cowan, G Tesauro, and J Alspector, pp. 160–167.

Schaal, Stefan, Christopher G Atkeson, and Sethu Vijayakumar (2002). "Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning". In: *Applied Intelligence* 17.1, pp. 49–60.

Schölkopf, Bernhard, John C Platt, J Shawe-Taylor, Alexander J Smola, and Robert C Williamson (2001). "Estimating the support of a high-dimensional distribution". In: *Neural Computation* 13.7, pp. 1443–71.

Schölkopf, Bernhard and Alexander Smola (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive computation and machine learning. MIT Press.

Schrauwen, Benjamin, David Verstraeten, and Jan Van Campenhout (2007). "An overview of reservoir computing: theory, applications and implementations". In: *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pp. 471–482.

Schwarz, Gideon (1978). "Estimating the Dimension of a Model". In: *The Annals of Statistics* 6.2, pp. 461–464.

Searle, John R (1980). "Minds, brains, and programs". In: *Behavioral and brain sciences* 3.3, pp. 417–457.

Seeger, Matthias, Christopher K I Williams, and Neil D Lawrence (2003). "Fast forward selection to speed up sparse Gaussian process regression". In: *Workshop on AI and Statistics*. Vol. 9.

Settles, Burr (2009). *Active Learning Literature Survey*. Tech. rep. University of Wisconsin-Madison.

Shamir, T and Y Yomdin (1988). "Repeatability of redundant manipulators: mathematical solution of the problem". In: *IEEE Transactions on Automatic Control* 33.11, pp. 1004–1009.

Shizawa, Masahiko (1996). "Multivalued regularization network-a theory of multilayer networks for learning many-to-h mappings". In: *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* 79.9, pp. 98–113.

Silverman, Bernard W (1984). "Spline Smoothing: The Equivalent Variable Kernel Method". In: *Annals of Statistics* 12, pp. 898–916.

Sminchisescu, Cristian, Atul Kanaujia, and Dimitris Metaxas (2007). "BM3 E: discriminative density propagation for visual tracking." In: *IEEE transactions on pattern analysis and machine intelligence* 29.11, pp. 2030–44.

Smola, Alexander and Peter Bartlett (2001). "Sparse Greedy Gaussian Process Regression". In: *Advances in Neural Information Processing Systems 13*.

Smola, Alexander and Bernhard Schölkopf (2004). "A tutorial on support vector regression". In: *Statistics and computing* 14.3, pp. 199–222.

Snelson, Edward and Zoubin Ghahramani (2006). "Sparse Gaussian Processes using Pseudo-inputs". In: *Advances in Neural Information Processing Systems 18*. Ed. by Y Weiss, B Schölkopf, and J Platt. Cambridge, MA: MIT Press, pp. 1257–1264.

Soussen, Charles, Jérôme Idier, David Brie, and Junbo Duan (2011). "From Bernoulli–Gaussian Deconvolution to Sparse Signal Restoration". In: *IEEE Transactions on Signal Processing* 59.10, pp. 4572–4584.

Steil, Jochen Jakob (2004). "Backpropagation-decorrelation: online recurrent learning with O(N) complexity". In: *IEEE International Joint Conference on Neural Networks*. Vol. 2, pp. 843–848.

Steil, Jochen Jakob (2007). "Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning." In: *Neural Networks* 20.3, pp. 353–64.

Stone, Mervyn (1974). "Cross-Validatory Choice and Assessment of Statistical Predictions". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 36.2, pp. 111–147.

Sturm, Jurgen, Christian Plagemann, and Wolfram Burgard (2008). "Unsupervised body scheme learning through self-perception". In: *IEEE International Conference on Robotics and Automation*, pp. 3328–3333.

Suh, Ki C and J. Hollerbach (1987). "Local versus global torque optimization of redundant manipulators". In: *IEEE International Conference on Robotics and Automation*. Vol. 4. Institute of Electrical and Electronics Engineers, pp. 619–624.

Tenenbaum, J B, V de Silva, and J C Langford (2000). "A global geometric framework for nonlinear dimensionality reduction." In: *Science (New York, N.Y.)* 290.5500, pp. 2319–23.

Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society (B)* 58, pp. 267–288.

Tikhanoff, V, P Fitzpatrick, G Metta, L Natale, F Nori, et al. (2008). "An Open Source Simulator for Cognitive Robotics Research: The Prototype of the iCub Humanoid Robot Simulator". In: *Workshop on Performance Metrics for Intelligent Systems*. National Institute of Standards and Technology, Washington DC.

Ting, Jo-Anne, Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal (2008). "A Bayesian approach to empirical local linearization for robotics". In: *2008 IEEE International Conference on Robotics and Automation*, pp. 2860–2865.

Ting, Jo-Anne, Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal (2010). "Efficient learning and feature selection in high-dimensional regression." In: *Neural computation* 22.4, pp. 831–86.

Ting, Jo-Anne, Mrinal Kalakrishnan, Sethu Vijayakumar, and Stefan Schaal (2009). "Bayesian Kernel Shaping for Learning Control". In: *Advances in Neural Information Processing Systems 21*. Ed. by D Koller, D Schuurmans, Y Bengio, and L Bottou, pp. 1673–1680.

Ting, Jo-Anne, Michael Mistry, and Jan Peters (2006). "A Bayesian Approach to Nonlinear Parameter Identification for Rigid Body Dynamics." In: *Robotics: Science and Systems (RSS)*.

Tipping, Michael E (2001). "Sparse Bayesian Learning and the Relevance Vector Machine". In: *Journal of Machine Learning Research* 1, pp. 211–244.

Tipping, Michael E and Christopher M Bishop (1999). "Mixtures of Probabilistic Principal Component Analyzers". In: *Neural Computation* 11.2, pp. 443–482.

Tolani, Deepak, Ambarish Goswami, and Norman I. Badler (2000). "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs". In: *Graphical Models* 62.5, pp. 353–388.

Tomikawa, Y and K Nakayama (1998). "Approximating many valued mappings using a recurrent neural network". In: *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)* 2, pp. 1494–1497.

Torgerson, Warren S (1958). *Theory and methods of scaling*.

Toussaint, M and S Vijayakumar (2005). "Learning discontinuities with products-of-sigmoids for switching between local models". In: *Proceedings of the 22nd international conference on Machine learning*. ACM, pp. 904–911.

Tresp, Volker (2001). "Mixtures of Gaussian processes". In: *Advances in Neural Information Processing Systems 13*.

Tsagarakis, N G, G Metta, G Sandini, D Vernon, R Beira, et al. (2007). "iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research". In: *Advanced Robotics* 21.10, pp. 1151–1175.

Turing, A M (1937). "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1, pp. 230–265.

Udwadia, F. E. (2003). "A new perspective on the tracking control of nonlinear structural and mechanical systems". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 459.2035, pp. 1783–1800.

Ueda, Naonori and Zoubin Ghahramani (2002). "Bayesian model search for mixture models based on optimizing variational bounds". In: *Neural Networks* 15.10, pp. 1223–1241.

Verbeek, Jakob, Nikos Vlassis, and B Kröse (2003). "Efficient greedy learning of gaussian mixture models." In: *Neural computation* 15.2, pp. 469–85.

Verstraeten, David, Benjamin Schrauwen, M D'Haene, and D Stroobandt (2007). "An experimental unification of reservoir computing methods." In: *Neural Networks* 20.3, pp. 391–403.

Vijayakumar, Sethu, Aaron D'Souza, and Stefan Schaal (2005). "Incremental Online Learning in High Dimensions". In: *Neural Computation* 17.12, pp. 2602–2634.

Vijayakumar, Sethu, Aaron D'Souza, Tomohiro Shibata, Jörg Conradt, and Stefan Schaal (2002). "Statistical Learning for Humanoid Robots". In: *Autonomous Robots* 12.1, pp. 55–69.

Vlassis, Nikos and Aristidis Likas (2002). "A Greedy EM Algorithm for Gaussian Mixture Learning". In: *Neural Processing Letters* 15.1, pp. 77–87.

Wahba, Grace (1990). *Spline models for observational data*.

Wallace, C S and D M Boulton (1968). "An Information Measure for Classification". In: *The Computer Journal* 11.2, pp. 185–194.

Wampler, Charles (1986). "Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods". In: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1, pp. 93–101.

Wang, Chong and David Blei (2012). "Truncation-free Stochastic Variational Inference for Bayesian Nonparametric Models". In: *Advances in Neural Information Processing Systems 25*. Ed. by P Bartlett, F C N Pereira, C J C Burges, L Bottou, and K Q Weinberger, pp. 422–430.

Wang, Li-Chun Tommy and Chih Cheng Chen (1991). "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators". In: *IEEE Transactions on Robotics and Automation* 7.4, pp. 489–499.

Warmuth, Manfred K and Dima Kuzmin (2008). "Randomized Online PCA Algorithms with Regret Bounds that are Logarithmic in the Dimension". In: *Journal of Machine Learning Research* 9, pp. 2287–2320.

Waterhouse, Steve, David Mackay, and Tony Robinson (1996). "Bayesian Methods for Mixtures of Experts". In: *Advances in Neural Information Processing Systems 8 (NIPS)*. Vol. 8. MIT Press.

Waterhouse, Steve and A J Robinson (1995). "Pruning and growing hierachical mixtures of experts". In: *4th International Conference on Artificial Neural Networks*. Vol. 1995. IEEE, pp. 341–346.

Watson, Geoffrey S (1964). "Smooth Regression Analysis". In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 26.4, pp. 359–372.

Whitney, D E (1972). "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators". In: *ASME Journal of Dynamic Systems, Measurement, and Control* 94, pp. 303–309.

Whitney, Daniel (1969). "Resolved Motion Rate Control of Manipulators and Human Prostheses". In: *IEEE Transactions on Man Machine Systems* 10.2, pp. 47–53.

Widrow, B and Matthew Hoffman (1960). "Adaptive Switching Circuits". In: *IRE WESCON Convention Record*, pp. 96–104.

Wilson, S W (2002). "Classifiers that approximate functions". In: *Natural Computing* 1.2, pp. 211–234.

Wold, Herman (1975). "Soft modeling by latent variables: The nonlinear iterative partial least squares approach". In: *Perspectives in probability and statistics, papers in honour of M. S. Bartlett*. Ed. by M S Bartlett and J M Gani, pp. 520–540.

Wolpert, Daniel M, Zoubin Ghahramani, and J Randall Flanagan (2001). "Perspectives and problems in motor learning." In: *Trends in cognitive sciences* 5.11, pp. 487–494.

Wolpert, Daniel M and Mitsuo Kawato (1998). "Multiple paired forward and inverse models for motor control". In: *Neural Networks* 11.7-8, pp. 1317–1329.

Wolpert, Daniel M, R Chris Miall, and Mitsuo Kawato (1998). "Internal models in the cerebellum." In: *Trends in cognitive sciences* 2.9, pp. 338–47.

Wolpert, David H (1996). "The Lack of A Priori Distinctions Between Learning Algorithms". In: *Neural Computation* 8, pp. 1341–1390.

Xu, Lei, Michael I Jordan, and Geoffrey E Hinton (1995). "An Alternative Model for Mixtures of Experts". In: *Advances in Neural Information Processing Systems*. The MIT Press, pp. 633–640.

Yang, Yan and Jinwen Ma (2011). "An efficient EM approach to parameter learning of the mixture of gaussian processes". In: *Advances in Neural Networks–ISNN 2011*, pp. 165–174.

Yuan, Chao and Claus Neubauer (2009). "Variational Mixture of Gaussian Process Experts". In: *Advances in Neural Information Processing Systems 21*. Ed. by D Koller, D Schuurmans, Y Bengio, and L Bottou, pp. 1897–1904.

Yuksel, S. E., J. N. Wilson, and P. D. Gader (2012). "Twenty Years of Mixture of Experts". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.8, pp. 1177–1193.

Zecca, M., N. Endo, S. Momoki, K. Itoh, and A. Takanishi (2008). "Design of the humanoid robot KOBIAN - preliminary analysis of facial and whole body emotion expression capabilities-". In: *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pp. 487–492.