Denoising Auto-encoders for Learning of Objects and Tools Affordances in Continuous Space

Atabak Dehban[1], Lorenzo Jamone[1], Adam R. Kampff[2] and José Santos-Victor[1]

***Abstract*— The concept of affordances facilitates the encoding of relations between actions and effects in an environment centered around the agent. Such an interpretation has important impacts on several cognitive capabilities and manifestations of intelligence, such as prediction and planning. In this paper, a new framework based on denoising Auto-encoders (dA) is proposed which allows an agent to explore its environment and actively learn the affordances of objects and tools by observing the consequences of acting on them. The dA serves as a unified framework to fuse multi-modal data and retrieve an entire missing modality or a feature within a modality given information about other modalities. This work has two major contributions. First, since training the dA is done in continuous space, there will be no need to discretize the dataset and higher accuracies in inference can be achieved with respect to approaches in which data discretization is required (e.g. Bayesian networks). Second, by fixing the structure of the dA, knowledge can be added incrementally making the architecture particularly useful in online learning scenarios. Evaluation scores of real and simulated robotic experiments show improvements over previous approaches while the new model can be applied in a wider range of domains.**

I. Introduction

The intelligence of an agent is related to its ability to succeed in an environment [1]. This success depends on the agent's ability to choose appropriate actions in order to achieve some objective or goal. Humans, as an inspiration for designing artificial intelligent agents, can solve complex tasks routinely by choosing proper actions from a vast repertoire of possibilities. It is very difficult, if not impossible, to achieve this efficiency in action planning without a mechanism of predicting the consequences of action execution especially in environments that are not fully known to agents. According to Pezzulo et al. [2] the ability to predict the outcome of different actions is a manifestation of mastering the sensorimotor knowledge, where sensorimotor knowledge is defined as "practical knowledge of the ways movement gives rise to changes in stimulation" [3].

One way to encode the aforementioned dynamic relationships is through the concept of affordances. Affordances can be defined as sensorimotor contingency patterns in the perceptual stimulus [4] which arise from action possibilities that an environment can offer an agent. The term was first introduced in J. J. Gibson's influential work [5] and has gathered the attention of many researchers from diverse fields such as developmental psychology, neuroscience and robotics.

Tackling the problem of prediction in sensorimotor space through the concept of affordances facilitates the design of robots that are expected to behave in unstructured environments because this approach allows the robot to predict how an object would behave in response to actions even without any prior interaction with that particular object and thus, solving the problem of object manipulation independent of object recognition.

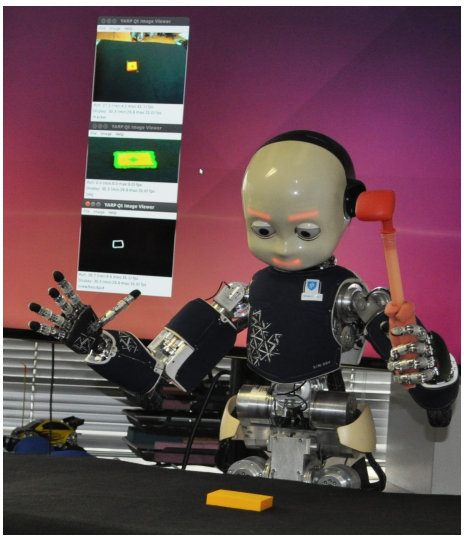


Fig. 1. Experimental setup with iCub humanoid

In robotics, affordances are most commonly defined as an acquired relation by an agent between the triplet: (a) *entity* or *object* (b) *action* or *behavior* and (c) *effects* [6]. According to this definition, for an agent to acquire the knowledge of affordances, it must already have a means to acknowledge objects, be able to execute at least some primitive actions and also to notice the effects caused by action execution.

This is also the scenario which will be presented in this work. More specifically, we will explain how a denoising Auto-encoder (dA) [7] , which are inspired from Multi-Layer Perceptrons (MLPs), can be trained to find relations between objects' and tools' shape descriptors with actions and the effects of executing those actions - the relative displacement of object. dA has been selected to encode the knowledge of affordances for several reasons. First, since they are a variant of neural networks, they can be applied in online learning scenarios. Second, they impose no constrain on the type of inputs they can use (continuous or discrete) and third, they were applied successfully in fusing multi-modal information and retrieve one modality given the others [8].

The particular architecture of dA has another desired property to model object affordances. In order to better explain this property, we contrast dA to its predecessor MLPs. If an MLP was used to encode the knowledge of affordances, one would need to train a network which uses objects and tools features and actions, and outputs the effect. Then it was necessary to train another network

*This work is partially supported by the Portuguese FCT grant PD/BD/105776/2014 and by the EU Project LIMOMAN [PIEF-GA-2013-628315].

[1]A. Dehban, L. Jamone and J. Santos-Victor are with the Institute for Systems and Robotics, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal. {adehban, ljamone jasv}@isr.ist.utl.pt

[2]Adam R.Kampff is with Champalimaud Neuroscience Programme, Champalimaud Centre for the Unknown, Lisbon, Portugal and Sainsbury Wellcome Centre for Neural Circuits and Behaviour (SWC), London, UK. adam.kampff@neuro.fchampalimaud.org

with objects and tools features and effects, to infer the action, and so on. In addition, different combinations would all require a dedicated network, e.g. given object features and effects, infer tool features and actions. On the other hand, a dA architecture fuses all the modalities together and thus, one network can be used to infer all arbitrary combinations of missing and available features and modalities (Of course not all queries will lead to meaningful results. For example one can provide tool and object features to the network and the network will provide answers for actions and effects, which in this case it would be hard to interpret these outputs).

The learning of affordances happens as the robot performs various actions on different combinations of objects and tools while observing the displacement of the object after 5 seconds. These data is then used to train a dA, which serves both as a cross-modal memory retriever [9] and an interpolator since it learns useful cross-modal relations. The objects' and tools' shape descriptors are the geometric properties of objects and tools surfaces which were defined in [10] and [11], [12]. We have first compared the scores of dA with those reported by Gonçalves et al. [12] using the same dataset. In order to show the applicability of dA in real scenarios, the model is also trained and tested on a dataset that was collected by doing experiments on real iCub humanoid robot [13] (Fig. 1). In our experiments, the robot can successfully imitate a human demonstrator by selecting the correct tool and action.

The rest of the paper is organized as follows. After reviewing the state of the art (Sec. II), in Sec. III the features of tools and objects that were used in the dA to learn the affordances are introduced. Afterward, actions that were executed by the robot are presented and we have also explained what are the effects that were measured by the robot after action execution. Next we present our dA and explain how it can be used to learn about affordances. A description of the experimental setup is presented in Sec. IV and the results of these experiments are reported, which show the estimation and generalization properties of the dA over different datasets. These results are also compared to the state-of-the-art. Finally, in Sec. V we draw our conclusions.

II. Related Work

Fitzpatrick et al. [14] were among the first researchers who have applied object affordances to reason about action effects. In their experiment, a robot learns about objects *rollability* affordance by executing *poking* action on it from different angles and observe the resulting motion behavior. They have shown that the robot can use this acquired knowledge in order to imitate an observed action to achieve similar results by selecting the right poke angle. However, this work is focused on interaction aspects of robots and objects and not on learning an association between visual features and actions with effects ,thus, the learned knowledge cannot be generalized to new and unseen objects.

Stoytchev [15] extends the idea to incorporate the concept of tool. In this work the robot learns about the affordances of various tools by doing motor babbling with them and observing the effect. The robot learns the associations between tool colors and behaviors and effects. The robot can update its representation during execution if inconsistencies are detected, e.g. breaking of a tool. Since the tool feature that they have used in their model was color, which in general can be independent of affordances of tools, their model cannot associate new tools to the ones that were observed in the past and it must restart the motor exploration step for each new tool that is introduced.

In a more recent article, Katz et al. [16] have used visual geometric features of objects for autonomous pile manipulation with a robotic arm. The features were autonomously calculated from images of a camera and a user would perform manual labeling of predefined affordance classes such as grasp or push. Since they have used visual geometric features, their model can guess labels from experience for novel objects. However, the requirement for hand-labeled dataset limits the robot's potential to discover new action possibilities.

Doğer et al. [17] have proposed a goal oriented affordance control for mobile robots based on the formalism of [6]. The robot explores the world with its primitive actions and collects the related changes in feature space as effects. The effects are then clustered using k-means to form the effect-id. Afterward for each effect-id, a supervised algorithm such as SVM is trained to map relevant features to effects. Ugur et al. [18] has further developed this idea by encoding the effects and objects in the same feature space and have utilized the learned mappings to perform various object manipulation tasks using a robotic arm. In this approach, the clustering algorithm that is used to find effect-ids and the subsequent number of SVMs, needs the whole dataset in a batch. Thus, this approach is not suitable in online learning scenarios.

Montesano et al. [10] have proposed a Bayesian Network (BN) to encode the knowledge of affordances. The BN captures the structural dependencies between actions, object features -which were clustered using Xmeans-, and effects and can effectively deal with uncertainties in the real world. It also provides a unified framework to not only learn, but also to use object affordances. This model can be used to imitate observed actions by predicting the effects caused by its own action execution on different objects. Since the model only relies on object's visual features, it can easily extend its knowledge to novel objects.

Osório et al. [19] have used Gaussian Mixture Models (GMMs) to use continuous features and effects with BNs. Their results obtained from simulation datasets suggest using continuous values improves the performance of BNs in the presence of noisy data or when the size of training set is large. However, to train the GMM, they needed to know the sensor noise distribution which limits the applicability of this approach in real scenarios.

Using icub humanoid, Tikhanoff et al. [20] have proposed a method to learn the pulling affordance of a tool. The icub robot preforms several experiments with a specific tool and a specific object and learns to map the initial relative configuration of the tool with respect to the object to the object's displacement as the result of pulling. Because the learned maps are specific to particular tools and objects, generalization to novel tools and objects requires new experiments to be performed.

More recently, Gonçalves et al. [11] have extended the use of BNs to incorporate the concept of intermediate objects i.e. tools. This model can be trained to relate tools' and objects' features to actions and effects. The performance of different architectures of BNs are evaluated in [12]. In this work, the dataset that was used to train the BN was collected in iCub simulator [21] but was successfully applied to predict the effects on the real robot.

One drawback of the aforementioned methods is that although features are initially calculated in the continuous space, some form of data clustering has to be applied to discretize the data. Automatic data clustering requires the whole batch of dataset to find clusters which are going to be used in the BN. On the other hand, ad hoc data clustering also significantly influences the final performance of the network. Moreover, learning the structure of the BN also requires the whole dataset. In addition to these, by clustering the features and effects into several bins, the accuracy of the prediction

will be at most equal to the accuracy of clusters which also hinders the performance.

This work is built upon the work of [12] as it uses the same features to represent the affordances. However, there are significant differences compared to all the mentioned approaches. First, since this architecture is trained on continuous features and effects, the accuracy of predictions will not be affected by the bin sizes and higher precisions could be obtained. Second the method can be applied in online scenarios where information is added to the network in an incremental manner.

III. Computational Methodology for Learning Object Affordances

In this section we will outline the affordances learning process which is similar to the process described in [11]. In this scenario, it is assumed that the robot already posses an elementary set of capabilities to perform some actions, calculate object and tool's geometric visual descriptors and to measure the effects caused by the execution of actions as explained in Sec. I. The robot is presented with one object and one tool and it calculates their visual descriptors, then one of the predefined actions will be executed on the object with the tool which the robot is holding in hand. The effect of this action execution is also recorded. The set of object and tool features, action and effects constitutes one sample in the dataset.

A. Visual descriptors

According to affordances theory, visual features of the environment should immediately give rise to action inference. To this end, we will extract the visual descriptors from the 2D silhouette of object and tool. This task entails the segmentation of the object from its surrounding background. This segmentation can easily become complex specially when one needs to segment unknown objects in cluttered scenes. This work is not focused on solving this problem in its general form, thus we have limited the robots playground on a table with colorful toys as objects and tools. With this simplification, the objects and tools can be easily segmented by using color thresholds and thus, connected components of pixels in 2D can be retrieved. The geometric visual descriptors that were introduced in [10] and [11] are then calculated based on the connected components. These features are **Area**, **Convexity**, **Eccentricity**, **Compactness**, **Circleness** and **Squareness** which are useful since they allow the model to generalize to novel objects and tools.Consider a situation where the goal of the human or robot is to point. This task can be effectively carried out using numerous objects such as a pen, a fork or a spoon. Although all these objects belong to different categories, they will become similar given the goal i.e. desired effect. On the contrary, if the goal becomes eating from a bowl of soup, an architecture which knows about affordances would choose the spoon over fork even though they might be conceptually belong to the same category of cutlery.

B. Actions and effects

The actions that are considered in this work are categorical and predefined and are constituted of a subset of different pushes (*left, right, pull close* or *push away*) which are preformed on the object while the tool is held in robot's hand. The IDs of these actions are directly fed to the affordances architecture.

The effects of action execution are considered as the 2D displacement of the object's center of mass along the lateral and longitudinal directions on the table's plane. To calculate this value, the position of the object on the image plane before the action execution and 5 seconds after action execution is recorded. The difference between these two values serves as the effect. In longitudinal movement, the positive direction means closer to the robot (down in image space) and the negative direction means farther away from the robot (up in image space). In lateral movement, positive and negative values are used to encode right and left movements, respectively.

The way the visual features and effects are introduced to the model is where our proposed solution differentiates from approaches that rely on data discretization. After the aforementioned features and effects are calculated from camera images, they are fed directly into the dA and are used to train the network without any need to group them into different bins.

C. Denoising auto-encoders for learning and using object affordances

We propose to use an over-complete denoising Auto-encoder (dA) to capture the structure of the affordance dataset. The dA for this work consists of one encoding layer g^{-1} and one decoding layer g (Fig. 2). A simple auto-encoder takes an input vector $\mathbf{x} \in [0,1]^m$ and maps it to a hidden representation $\mathbf{z} \in [0,1]^n$ through a deterministic mapping $\mathbf{z} = g^{-1}(\mathbf{x})$ where:

$$g^{-1}(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1}$$

parameterized by $\theta = \{\mathbf{W}, \mathbf{b}\}$. $\mathbf{W}$ is an $n \times m$ weight matrix and $\mathbf{b}$ is an $n \times 1$ bias vector. After learning, the network tries to reconstruct its input via the mapping $\hat{\mathbf{x}} = g(\mathbf{z})$ and we have:

$$g(\mathbf{z}) = s\left(\mathbf{W}'\mathbf{z} + \mathbf{b}'\right) \tag{2}$$

which similarly is parameterized by $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. In (1) and (2) $s(\cdot)$ is the squashing function (usually sigmoid). In order to reduce the number of auto-encoder parameters and hence, avoid the problem of overfitting, one can optionally constrain the weight matrices to $\mathbf{W}' = \mathbf{W}^T$, in which case the auto-encoder is said to have *tied weights*. Suppose $\mathbf{x}^{(i)}$ to be one sample point from the training dataset $\mathscr{D}$, thus for each $\mathbf{x}^{(i)}$ the corresponding hidden representation $\mathbf{z}^{(i)}$ is determined and used to reconstruct the output vector $\hat{\mathbf{x}}^{(i)}$.

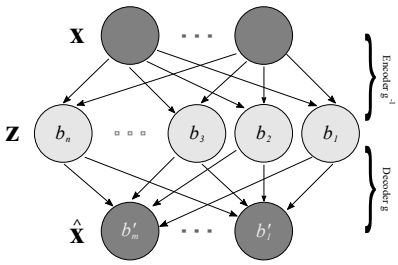


Fig. 2. Schematics of an auto-encoder where x is the input, z is the hidden representation and $\hat{\mathbf{x}}$ is the reconstruction. b_is are the bias terms associated with hidden layer units and b'_is are bias terms associated with the output layer.

The parameters θ and θ' are optimized to minimize the average reconstruction error:

$$\theta^*, \theta'^* = \arg\min_{\theta,\theta'} \frac{1}{d} \sum_{i=1}^{d} \mathscr{L}\left(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)}\right) \tag{3}$$

where d is the number of samples in $\mathscr{D}$. The loss function $\mathscr{L}$ can be a simple squared error $\mathscr{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|$. In case of variables which are between zero and one, the loss function is more commonly defined as *sum of Bernoulli cross-entropies*:

$$\mathscr{L}(\mathbf{x},\hat{\mathbf{x}}) = -\sum_{k=1}^{m}[x_k \log \hat{x}_k + (1-x_k)\log(1-\hat{x}_k)] \qquad (4)$$

To see how the above formula can be interpreted as a loss function, consider the case where $x_k = 0$. The first term in (4) becomes zero and $\mathscr{L}(x_k,\hat{x}_k) = -\log(1-\hat{x}_k)$. The training algorithm tries to minimize this value and thus $\hat{x}_k$ approaches x_k. A similar argument applies to the case where $x_k = 1$.

The above formulation cannot capture the inter-feature dependencies of our dataset since we have an over complete hidden layer and each hidden unit can just learn to replicate its input. In this case the reconstruction error of the network for the training set will be very small while the network has not learned any useful relations.

One approach to make an over-complete auto-encoder to learn useful features is by adding random noise to input, but force the network to reconstruct the denoised version of the input [7]. The network cannot reconstruct its input from noisy observations, unless it could capture the structure of the dataset. In this case, the learning algorithm tries to converge $\mathbf{z}$ to a representation which is useful to reconstruct the input when some features are not available.

The input of the dA is a corrupted version $\tilde{\mathbf{x}}$, of the original input $\mathbf{x}$ according to a stochastic mapping $\tilde{\mathbf{x}} \sim q(\tilde{\mathbf{x}}|\mathbf{x})$. In our experiments, for each component of a feature vector, we have conducted a Bernoulli experiment with probability of success v and if the experiment was successful, that component was forced to be 0.5. Since there is no information in the choice of the corrupted component, the dA has to use the information from the available components to infer the value of the corrupted feature. This corrupted input $\tilde{\mathbf{x}}$ replaces the original input $\mathbf{x}$ in (1).

IV. Evaluation Scores and Results

In this section we will discuss the training process of the proposed over-complete denoising auto-encoder and validate its performance on two datasets. One of these datasets was collected using iCub simulator and the other was collected from the real robot.

A. Training process of denoising auto-encoder

The implementation of dA was done in Theano [22] which is an open source numerical computation library for Python. Theano was selected since it provides a convenient interface to do numerical computations on GPUs and can be easily integrated with other Python libraries such as NumPy [23] and scikit-learn [24].

As explained in the previous section, the input to the dA must lie between zero and one. Thus, it is necessary to preprocess the data before feeding it into the network. In this step, each feature of the dataset was first scaled to have zero mean and unit variance ($\sigma = 1$). Afterward all the components which were outside three standard deviation range were saturated to the value of three standard deviation(3σ) to remove the outliers. The features were then scaled in the interval $[0.1, 0.9]$ to comply with the general guidelines in [25].

Empirically, adagrad training algorithm [26] with momentum found better minimums in our training sets. Adagrad tries to adapt a new learning rate for each feature at each time step. In this way, the learning algorithm pays more attention to rare, but informative features while slowly learning from the frequent ones.

We have found that increasing the number of hidden units does not significantly improve the reconstruction errors on the training set. Two explanations are proposed for this behavior. First, by increasing the number of hidden layers, the training algorithm cannot find better minimums for the weights and biases. The second explanation is that we have already reached the limits of our training set. Careful examination of the dataset revealed the existence of contradicting samples (the same tools and objects along with the same action results in completely different effects, i.e. bigger than 5σ) due to the realistic scenarios in our datasets. These observations make the second explanation more probable.

Since the whole datasets were available and this work is not addressing the issue of online learning, we have used 5 fold cross validation to choose the learning rate from the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the number of hidden units from the set $\{20, 22, 24, 26, 28\}$. Nevertheless, the performance of the dA is not very sensitive to the choice of these hyper parameters and our initial guesses were almost as good as the results obtained from cross validation. This robustness of our results with regard to hyper parameters can be attributed to the fact that the set of features we have chosen to represent tools and objects are relatively uncorrelated which we will address while we discuss each dataset.

B. Simulation dataset

In this scenario, 2353 trials were done using the iCub simulator. In each experiment, the robot first looks at the object and tool from different angles and calculates the features introduced in Sec. III-A (this dataset was gathered by Gonçalves et al. [11] and we have used it with the permission of the author). Then, while holding the tool in hand, chooses one of the actions from the set of actions in Sec. III-B. More details about the shape of tools and objects can be found in [11]. Each sample of this dataset has 15 components ($m = 15$, 6 object features, 6 tool features, 1 action tag and 2 effects). In the correlation coefficient matrix calculated from the object and tool features, only 30% of the elements had an absolute value bigger than 0.5 (outside the main diagonal). Table I shows hyper parameters that were used for this dataset.

TABLE I
Simulation Dataset Hyper-parameters

Train Parameters				dA Parameters	
epsilon	learning rate	momentum	mini-batch size	# hidden units n	corruption v
10^{-6}	0.2	0.8	200	24	0.3

In order to evaluate the performance of dA, we first compare its performance to previous approaches. This comparison is problematic since the dA outputs deterministic values while the output of the BN reported in [12] are probabilities. On the other hand, dA is trained on continuous data points and outputs a continuous value where as the BN is trained and tested on categorical data.

To reconcile these two approaches, first we have discretized the outputs of the dA after it was trained on continuous data. Furthermore, since the accuracy score of BNs is based on maximum a posteriori estimate, it is the closest measure to a deterministic output thus, we have adopted this score as the basis of our comparison. To measure the accuracy score for the bayesian network, the probability of each effect given the value of actions and features is calculated. If the most probable effect bin is the same as the correct bin, the network gets a reward of 1. The sum of total rewards divided by the number of testset samples in percent is the accuracy score. A random system which chooses one out of five bins gets a score of 20 . On the other hand, the accuracy score for dA is simply the number of correct bin predictions normalized by the number of testset samples.

Two tests were conducted using the simulation dataset. First, a dataset of experiments with 7 objects and tools was divided into a training set containing 80% of the samples and a testset containing the remaining 20%. The dA is trained as explained in Sec. III-C and during test time, all the effects were corrupted by the value 0.5 and all other features and actions were left untouched. Table II shows the accuracy score of the different architectures. According to the results of this table, the dA can provide similar results to the previously reported approaches without the need to be trained on discrete data.

TABLE II

EVALUATION SCORES OF SPLITTING THE DATASET OF 7 OBJECTS INTO 80% OF TRAINING SET AND 20% OF TEST SET

	Baseline	PCA	BDe	K2	dA
Accuracy Sc.(%)	75.90	80.57	83.28	**83.73**	79.13

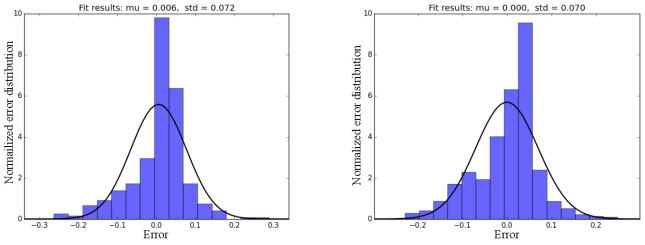
In another experiment, the same model which was trained on the previous step was tested on an unseen object. The results of this experiment are reported in Table III.

TABLE III

EVALUATION SCORES OF THE UNSEEN OBJECT

	Baseline	PCA	BDe	K2	dA
Accuracy Sc.(%)	44.20	73.91	48.42	47.93	**76.52**

The dA score is similar to the reported BN score which was trained on a PCA transformation of object and tool features. In this network, after finding the principal components, only the first two components were chosen as representatives of object and tool features and each component was discretized into two categories. Our experiments show that the reported results of the PCA network can be severely hindered if more components were used or if each component was discretized into more categories. On the other hand, one of the merits of this work is that the performance of the proposed dA is not affected by the choice of data discretization or dimensionality reduction techniques.



(a) split dataset (b) unseen object

Fig. 3. Error distribution and histograms on simulation dataset

After these comparisons, we will evaluate the performance of dA in continuous space to measure how reliable it can estimate the effects. In these evaluations, we have fitted a normal curve to the errors made by the dA. Note that each component of the samples is scaled to be in the interval $[0.1, 0.9]$ but the output of the dA comes directly out of the sigmoid function and thus is in the interval $[0, 1]$. Fig. 3 shows the error histogram and distribution for the above two experiments. According to these plots, 95% of the errors are below 0.21 (3σ) which is about 21% of the maximum possible

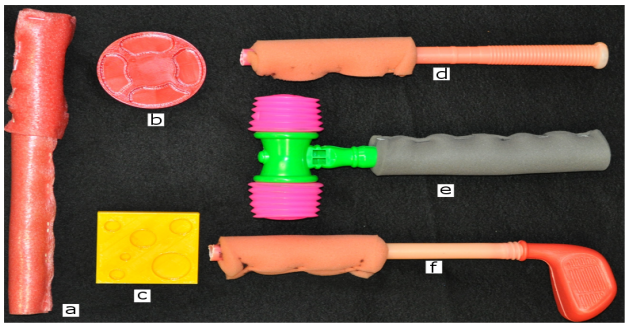


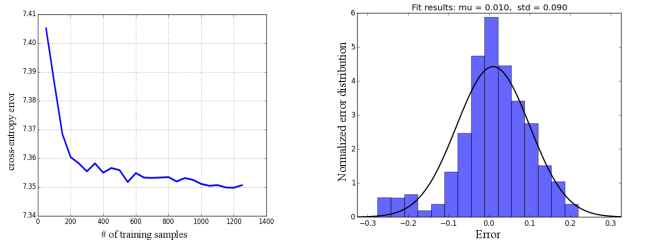
Fig. 4. Objects and tools that were used in the robot experiment. a) stick_1, b) tomato, c) cheese, d) stick_2, e) hammer and f) rake

error. The code to access this dataset along with an implementation of the dA can be found at `https://github.com/atabakd/Continuous_affordances`.

C. Real robot dataset

In order to see how the model behaves when encountered with real world data, another dataset was collected in which the real iCub robot executed actions on different objects and toys (Fig. 4). The experiment was done with two actions {*push*, *pull*}, two objects {*cheese*, *tomato*} and two tools {*rake*, *stick_1*} where each action was repeated twice. Feature *area* was not calculated for objects and tools (similar to the original feature set introduced in [10]). The robot observes the tools and objects from 10 different view points which results in a dataset of size 1600 ($2 \times 2 \times 2 \times 2 \times 10$). Each sample of this dataset has 13 components ($m = 13$, 5 object features, 5 tool features, 1 action tag and 2 effects). In this dataset, 22% of the tool and data features had an absolute correlation coefficient bigger than 0.5. The hyper parameters of this network are similar to Table I, the only difference is that because of smaller number of features, 20 hidden units provided the best cross validation score.

First, we will examine the network's performance with regard to the number of samples. Fig. 5a shows the cross-entropy errors made by the network on a fixed test set as a function of train set size. This plot shows that increasing the number of training samples can improve generalization error.



(a) cross-entropy error (b) Error distribution and histogram of errors for the real dataset

Fig. 5. dA performance on real dataset

As in the first experiment with simulation dataset, 80% of the data was used for training and 20% for testing. Fig. 5b shows the distribution and histogram of the error. The similar performance of dA in simulation and real datasets shows that it is a reliable architecture for reason about object and tool affordances in robotic experiments.

Similar to the simulation dataset, we have conducted new experiments on the robot to see the generalization capabilities. In the new

experiments, the human demonstrator pushes an object {*cheese*} toward the robot and the robot observes the effect. Afterward, the robot is presented with two novel tools {*stick_2, hammer*} and is expected to choose the right tool along with the right action to imitate the observed effects. The experiment was repeated 8 times and each tool was viewed from 4 different angles. The exact same transformation which was used to pre-process the real robot dataset was applied on the new tool and object features. In all the trials, the robot had successfully selected the right tool {*hammer*} with the right action {*pull*}.

V. Conclusions and future work

In this work we have proposed the use of denoising auto-encoders as a computational framework to learn and reason about object affordances. This model is able to find structures which can relate object and tool features with actions and effects. These representations can be later used to 1) predict the effects of action executions on different objects with different tools and 2) imitate effects by selecting the correct action and tool. The comparison of dA performance with previous approaches shows its superior generalization capabilities while maintaining desirable expressiveness power. Infering the action and tool directly from object features and observed effect is one of the main contributions of this work. Our experiments indicates that going from simulation datasets to real datasets does not impose any significant degradation of performance and implies effectiveness of this approach on real robot scenarios.

Denoising auto-encoders are mostly used in the deep learning literature as a pre-training step for multi-layer neural networks. To the knowledge of authors, this work shows the first use of dAs in relatively small datasets with low amount of features. To take this work to where dAs shine the brightest, new scenarios must be defined where the features are not predefined but are learned from camera images and actions are not just labels but the dynamics of joint angles movements. On the other hand, as the results of the current work suggest, dA can be trained to find interesting relations even with small datasets. One possible approach to further investigate the capabilities of the proposed architecture is to use it as an online learning mechanism in real robot scenarios. The dA can benefit from more tool and object features which better covers the space because of its relatively large expressive power.

Acknowlegment

This work is partially supported by an FCT doctoral grant PD/BD/105776/2014 and by the EU Project LIMOMAN [PIEF-GA-2013-628315].

References

[1] S. Legg and M. Hutter, "A universal measure of intelligence for artificial agents," in *International Joint Conference on Artificial Intelligence*, vol. 19. LAWRENCE ERLBAUM ASSOCIATES LTD, 2005, p. 1509.

[2] G. Pezzulo, M. Butz, O. Sigaud, and G. Baldassarre, *Anticipatory Behavior in Adaptive Learning Systems: From Psychological Theories to Artificial Cognitive Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

[3] A. Noë, *Action in perception*. MIT press, 2004.

[4] S. Z. Caiani, "Extending the notion of affordance," *Phenomenology and the Cognitive Sciences*, vol. 13, no. 2, pp. 275–293, 2014.

[5] J. J. Gibson, *The Ecological Approach to Visual Perception*. Boston, MA: Houghton Mifflin, 1979.

[6] E. Sahin, M. Çakmak, M. R. Dogar, E. Ugur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, 2007.

[7] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[8] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 689–696.

[9] K. Noda, H. Arie, Y. Suga, and T. Ogata, "Multimodal integration learning of robot behavior using deep neural networks," *Robotics and Autonomous Systems*, vol. 62, no. 6, pp. 721–736, 2014.

[10] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory motor coordination to imitation," *IEEE Transactions on Robotics*, vol. 24, no. 1, pp. 15–26, 2008.

[11] A. Gonçalves, G. Saponaro, L. Jamone, and A. Bernardino, "Learning visual affordances of objects and tools through autonomous robot exploration," in *Autonomous Robot Systems and Competitions (ICARSC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 128–133.

[12] A. Gonçalves, J. Abrantes, G. Saponaro, L. Jamone, and A. Bernardino, "Learning intermediate object affordances: Towards the development of a tool concept," in *IEEE ICDL-Epirob*, 2014.

[13] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. Von Hofsten, K. Rosander, M. Lopes, J. Santos-Victor *et al.*, "The icub humanoid robot: An open-systems platform for research in cognitive development," *Neural Networks*, vol. 23, no. 8, pp. 1125–1134, 2010.

[14] P. Fitzpatrick and G. Metta, "Grounding vision through experimental manipulation," *Phil. Trans. R. Soc. A: Mathematical, Physical and Engineering Sciences*, pp. 2165–2185, 2003.

[15] A. Stoytchev, "Behavior-grounded representation of tool affordances," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 3060–3065.

[16] D. Katz, A. Venkatraman, M. Kazemi, J. A. Bagnell, and A. Stentz, "Perceiving, learning, and exploiting object affordances for autonomous pile manipulation," *Autonomous Robots*, vol. 37, no. 4, pp. 369–382, 2014.

[17] M. R. Doğar, M. Cakmak, E. Uğur *et al.*, "From primitive behaviors to goal-directed behavior using affordances," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 729–734.

[18] E. Ugur, E. Oztop, and E. Sahin, "Goal emulation and planning in perceptual space using learned affordances," *Robotics and Autonomous Systems*, vol. 59, no. 7, pp. 580–595, 2011.

[19] P. Osório, A. Bernardino, R. Martinez-Cantin, and J. Santos-Victor, "Gaussian mixture models for affordance learning using bayesian networks," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010, pp. 4432–4437.

[20] V. Tikhanoff, U. Pattacini, L. Natale, and G. Metta, "Exploring affordances and tool use on the icub," in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*. IEEE, 2013, pp. 130–137.

[21] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori, "An open-source simulator for cognitive robotics research: The prototype of the icub humanoid robot simulator," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, ser. PerMIS '08. New York, NY, USA: ACM, 2008, pp. 57–61.

[22] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.

[23] S. van der Walt, S. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science Engineering*, vol. 13, no. 2, pp. 22–30, March 2011.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[25] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.

[26] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.