Learning a Repertoire of Actions with Deep Neural Networks

Alain Droniou[†,‡], Serena Ivaldi[†,‡,⋆] and Olivier Sigaud[†,‡]
[†]Sorbonne Universités, UPMC Université Paris 06, ISIR UMR7222, Paris, France
[‡]CNRS, Institut des Systèmes Intelligents et de Robotique UMR7222, Paris, France
[⋆]Intelligent Autonomous Systems Lab, FB-Informatik, TU Darmstadt, Germany
Email: {droniou,ivaldi,sigaud}@isir.upmc.fr

Abstract—**We address the problem of endowing a robot with the capability to learn a repertoire of actions using as little prior knowledge as possible. Taking a handwriting task as an example, we apply the deep learning paradigm to build a network which uses a high-level representation of digits to generate sequences of commands, directly fed to a low-level control loop. Discrete variables are used to discriminate different digits, while continuous variables parametrize each digit. We show that the proposed network is able to generalize learned actions to new contexts. The network is tested on trajectories recorded on the iCub humanoid robot.**

I. Introduction

During their development, children learn an increasingly complex set of actions while interacting with their environment. Piaget [1] outlined that they build such a repertoire by progressively adapting known gestures to novel objects (*assimilation*) and by differentiating novel actions for objects which cannot be assimilated smoothly by known actions (*accommodation*).

Despite recent progress in control and perception, robots are still far from such capabilities. On one hand, several works aim to optimize a movement primitive depending on some parameters. For instance, [2] adds context parameters to dynamical movement primitives (DMPs) [3] to generate movements which generalize to new objects. On the other hand, some approaches provide a set of actions for the robot, which learns to associate these actions with different objects and contexts [4]. Some works (e.g. [5], [6], [7], [8], [9]) try to fuse both approaches by learning a repertoire of movement primitives, but are usually limited either by the curse of dimensionality or by the amount of required prior knowledge.

In this work, we present a neural architecture which is able to learn a repertoire of actions with very few priors. The proposed architecture can generalize learned actions to new contexts through a parametrization of each action. It uses gated connections to enable a distributed representation of actions shared by the whole repertoire and, using the deep learning paradigm, it provides a natural framework to use the powerful dimensionality reduction properties of deep networks [10].

We illustrate this architecture on a digit writing task. First, the network learns to generate trajectories for each of the ten digits. In a second experiment, we illustrate the parametrization of actions by teaching the network to draw rotated digits.

II. Related work

Most of the works addressing the problem of learning a repertoire of movement primitives rely on Dynamical Movement Primitives (DMPs) [3] (e.g. [5], [6], [7], [8], [9]), which combine a fixed spring-damper system to ensure convergence and stability properties, and a learnable term to deform the spring-damper trajectory into a given action. Usually, DMPs are parametrized with time (or with phase variables) [3], but other variables can be used to extend the context and provide more flexibility [11], [2]. Though this learnable term can theoretically deform the spring-damper trajectory to any trajectory, it typically necessitates prior knowledge from an external designer who defines adequate parametrized primitives along with their relevant context variables to circumvent the curse of dimensionality. As a result, DMPs are generally restricted to simple reaching or cyclic movements.

On the other hand, reproducing dynamical sequences using neural networks has been studied by several authors. An early tentative is the Elman's network [12], in which the hidden layer is copied at each time step into a *context* layer which is itself fed back to the hidden layer for the next time step. Such recurrent networks are difficult to train due to the vanishing gradient problem [13]. Other techniques such as echo state networks and liquid state machines [14], [15] also use recurrent hidden layers but with randomly chosen weights which remain untrained[1]. Such approaches necessitate large hidden layers whose connectivity matrices are carefully tuned, mainly based on their spectral radius.

Long-short term memories alternatively consist in using complex cells in which a *carousel* can propagate the gradient for an undetermined amount of time and for which several gates modulate the information flow [17]. This has been shown to be efficient to learn long-term dependences, for instance for hand-writing recognition [18].

Neural networks have been used for robotic control [19], for example in a global control architecture to solve inverse and forward kinematics along trajectories planned at a higher level of the architecture.

More recently, deep networks have been applied to sequence prediction [20]. The Conditional Restricted Boltzmann

[1]Except in some approaches where an Hebbian-like learning rule can be used to strengthen or weaken recurrent weights [16].

Machines (CRBM) take as input several time steps which are fed to the hidden layer and influence the prediction of the next input. This architecture is able to correctly model time series such as human limbs movements for different walking gaits [21]. CRBM has been extended to Factorized CRBM [21], in which gated connections decrease the number of learned parameters. Other approaches can also use recurrent connections at the hidden layer [22].

III. Background: Deep networks and gated connections

Though neural networks with a single hidden layer are universal approximators, the number of units required to approximate a given function can grow exponentially with the desired accuracy [23]. For a large set of functions, the number of required units decreases by stacking multiple levels of hidden layers [24], but then they are hard to train due to the vanishing gradient problem [13] and the presence of many poor local optima [25]. To overcome this issue, [10] proposed to pre-train each layer to learn a good representation of its input. Different possible pre-training algorithms are surveyed in [26]. One of the most popular paradigms is the autoencoding approach, which aims at minimizing the reconstruction error of the input data. Given a visible input[2] $\mathbf{v}$ and a hidden layer $\mathbf{h}$, it learns the encoding

$$\mathbf{h} = \sigma(W_1\mathbf{v} + \mathbf{b_h}) \quad (1)$$

where W_1 is a learned weight matrix, $\mathbf{b_h}$ is a bias term and σ is usually a non-linear activation function such as a sigmoid function: $\sigma(x) = \frac{1}{1+\exp(-x)}$. This encoding is then decoded to reconstruct the input using

$$\hat{\mathbf{v}} = \sigma(W_2\mathbf{h} + \mathbf{b_v}) \quad (2)$$

where W_2 is another learned weight matrix. Autoencoders are trained by backpropagating the reconstruction error (e.g. $||\mathbf{v} - \hat{\mathbf{v}}||_2^2$, or the cross-entropy in the binary input case). Several regularization techniques can be used, for instance weight tying, sparsity [27], denoising corrupted input [28] or penalization of the hidden layer sensitivity to input [29].

When the network learns a relation between inputs, for instance the transformation between two images or the temporal evolution of a variable, it is useful to use gated connections [30], [21], [31]. Factorizing these connections reduces the number of weights and makes learning easier [32]. Given two different input $\mathbf{x}$ and $\mathbf{y}$, such connections introduce an intermediate *factors* layer, which performs an element-wise multiplication between projections $\mathbf{f_x}$ and $\mathbf{f_y}$ of both inputs. The resulting factors $\mathbf{f_h}$ are then projected to the hidden layer $\mathbf{h}$ (see Fig. 1):

$$\mathbf{f_x} = W_x\mathbf{x}, \mathbf{f_y} = W_y\mathbf{y} \quad (3)$$

$$\mathbf{f_h} = \mathbf{f_x} * \mathbf{f_y} \quad (4)$$

[2]In the following, we denote vectors with bold letters, and matrices with capital letters.

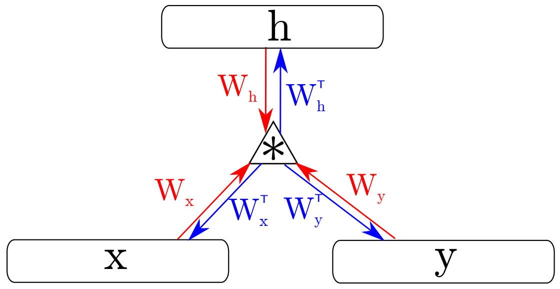

Fig. 1. Illustration of factored gated connections. Given two input $\mathbf{x}$ and $\mathbf{y}$, their projections through W_x and W_y are multiplied in an intermediate *factors* layer $*$, before being projected on the hidden layer h. Such connections are useful to learn representations of multi-dimensional relations between different variables. The role of the three external layers can be exchanged, making it possible to compute a reconstruction of $\mathbf{y}$ given $\mathbf{x}$ and $\mathbf{h}$, and *vice versa*: the two input are projected on the factors layer (red arrows), and the resulting factors are projected on the third layer (blue arrow).

$$\mathbf{h} = \sigma(W_h^\top \mathbf{f_h}) \quad (5)$$

where $*$ denotes the element-wise multiplication. The role of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ can be exchanged, allowing for instance to compute a reconstruction of $\mathbf{y}$ given $\mathbf{x}$ and $\mathbf{h}$.

IV. Architecture

In this section, a general mathematical formulation of the adressed task is presented. Then, the proposed implementation using deep networks is detailed.

A. Mathematical framework

A general representation of action is given by

$$\mathbf{q_t} = f(\mathbf{s_t}, \mathbf{m_t}, \mathbf{c_t}) \quad (6)$$

where $\mathbf{q_t}$ is the command at time t (e.g. torques, Cartesian or joints velocities, $\ldots$), $\mathbf{s_t}$ is the state of the system at time t, $\mathbf{m_t}$ corresponds to a memory of past states and $\mathbf{c_t}$ is a general context variable describing for instance the current goal of the system. We distinguish between state $\mathbf{s}$ and context $\mathbf{c}$ in the way it evolves while the system is performing an action: $\mathbf{s_t}$ describes an instantaneous state (for instance the current joint positions) which evolves at each time step and is strongly related to the course of the action currently performed, while the context $\mathbf{c_t}$ is intended to be a slowly varying input describing an overall situation, such as a symbolic representation of the pursued goal.

For the sake of clarity, throughout this paper we consider the example of a robotic writing task, in which we control the end-effector of a robot in the three dimensional Cartesian space to write different digits. Thus, we flesh-out (6) with the following variables: $\mathbf{q_t}$ encodes the desired Cartesian velocities $\dot{\mathbf{x}}_\mathbf{t}$ of the end-effector, $\mathbf{s_t}$ is the current Cartesian position $\mathbf{x_t}$ of the end effector, $\mathbf{m_t}$ stores some memory about past positions (this will be further described in the following), and $\mathbf{c_t}$ is a symbolic representation of the action being performed (i.e. the digit being written), represented by a boolean action vector $\mathbf{a}$ containing only zeros, except for the desired digit for which

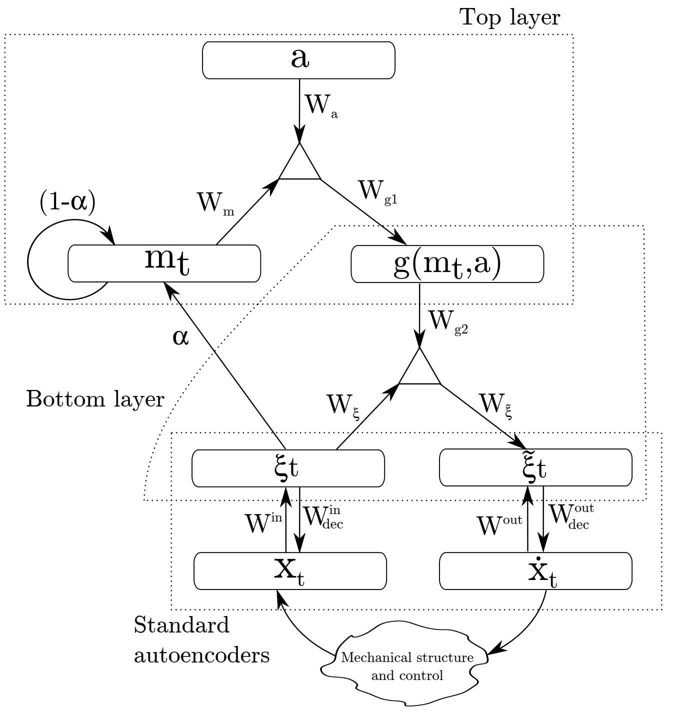

Fig. 2. Architecture of the network used for the experiment. The Cartesian positions $\mathbf{x_t}$ are fed into a standard autoencoder which learns a representation ξ. This representation is used to compute a trace $\mathbf{m_t}$ of the positions, and to compute the desired velocity. A sub-network uses the trace $\mathbf{m_t}$ and the overall goal $\mathbf{a}$ to compute an intermediate representation of an instantaneous goal $\mathbf{g}(\mathbf{m_t}, \mathbf{a})$. The arrows denote the direction of the projections computed by the corresponding matrices (transposed matrices are used to project backward, except for the two lower standard autoencoders). They are oriented according to the information flow inside the network when it is used to generate a desired velocity $\dot{\mathbf{x}}_\mathbf{t}$ given a position $\mathbf{x_t}$, an overall goal $\mathbf{a}$ and a current progress $\mathbf{m_t}$.

the corresponding bit is set to one. With these notations, (6) becomes:

$$\dot{\mathbf{x}}_\mathbf{t} = f(\mathbf{x_t}, \mathbf{m_t}, \mathbf{a}). \tag{7}$$

We further factorize (7), introducing an instantaneous goal function, $g(\mathbf{m_t}, \mathbf{a})$, which depends on the action being performed $\mathbf{a}$ (the "overall goal") and its current progress $\mathbf{m_t}$:

$$\dot{\mathbf{x}}_\mathbf{t} = f(\mathbf{x_t}, g(\mathbf{m_t}, \mathbf{a})) \tag{8}$$

Notably, such a factorization of the representations reduces the dimensionality of the input when the dimensionality of the output of g is smaller than the dimensions of $\mathbf{m_t}$ and $\mathbf{a}$; further, it allows function f to learn synergies which can be re-used by any of the actions from the repertoire.

B. Neural network implementation

The proposed network (8) contains two sublayers (Fig. 2):

- The top one corresponds to the computation of g which takes $\mathbf{m_t}$ and $\mathbf{a}$ as input.
- The bottom one corresponds to the computation of f and takes $\mathbf{x_t}$ and the output of the first sub-network as input.

The positions $\mathbf{x}$ of the end-effector are first encoded by an autoencoder using sigmoid units at the hidden layer, and linear units at the visible layer. This results in a "normalized" input (each unit is between 0 and 1) fed to the remaining network. To deal with an arbitrary range of values, the weights of this autoencoder are not tied: the encoder uses a weight matrix W^{in}, while the decoder uses a weight matrix W^{in}_{dec}. The output of this encoder is denoted by:

$$\xi_\mathbf{t} = \sigma(W^{in}\mathbf{x_t}). \tag{9}$$

Similarly, Cartesian velocities are encoded by a second autoencoder using matrices W^{out} and W^{out}_{dec}, whose output is denoted by:

$$\tilde{\xi}_\mathbf{t} = \sigma(W^{out}\dot{\mathbf{x}}_\mathbf{t}). \tag{10}$$

In the current implementation of the network, $\mathbf{m_t}$ is related to the past positions by a simple exponential window:

$$\mathbf{m_t} = (1-\alpha)\mathbf{m_{t-1}} + \alpha\xi_\mathbf{t}. \tag{11}$$

Each sub-network uses gated connections, which results in a factorization of representations at each layer. Thus, the g function is computed as

$$\mathbf{g}(\mathbf{m_t}, \mathbf{a}) = \sigma\left(W_{g_1}\left((W_m\mathbf{m_t}) * (W_a\mathbf{a})\right)\right) \tag{12}$$

and the h function is computed as

$$\tilde{\xi}_\mathbf{t} = \sigma\left(W_{\tilde{\xi}}\left((W_\xi\xi_\mathbf{t}) * (W_{g_2}\mathbf{g}(\mathbf{m_t}, \mathbf{a}))\right)\right). \tag{13}$$

Training this architecture consists in learning all connectivity matrices W_* in (9), (10), (12) and (13). This is done by minimizing the reconstruction error of predicted Cartesian velocities $\hat{\dot{\mathbf{x}}}$ given the positions and the vector $\mathbf{a}$, using a standard gradient descent on the error $\sum_t ||\hat{\dot{\mathbf{x}}}_\mathbf{t} - \dot{\mathbf{x}}_\mathbf{t}||^2$.

According to the deep learning paradigm, each sub network is pre-trained independently. First, the autoencoders are trained to learn a representation ξ of $\mathbf{x}$ and a representation $\tilde{\xi}$ of $\dot{\mathbf{x}}$ by minimizing the reconstruction error of $\mathbf{x}$ and $\dot{\mathbf{x}}$ respectively. Then, the lower level gated network is trained to learn a representation $\mathbf{g_t}$, given $\xi_\mathbf{t}$ and $\tilde{\xi}_\mathbf{t}$, by minimizing the distance with their reconstructions $\xi^\mathbf{recons}_\mathbf{t}$ and $\tilde{\xi}^\mathbf{recons}_\mathbf{t}$:

$$\mathbf{g_t} = \sigma(W^\top_{g_2}(W_\xi\xi_\mathbf{t} * W^\top_{\tilde{\xi}}\tilde{\xi}_\mathbf{t})) \tag{14}$$

$$\xi^\mathbf{recons}_\mathbf{t} = \sigma(W^\top_\xi(W_{g_2}\mathbf{g_t} * W^\top_{\tilde{\xi}}\tilde{\xi}_\mathbf{t})) \tag{15}$$

$$\tilde{\xi}^\mathbf{recons}_\mathbf{t} = \sigma(W_{\tilde{\xi}}(W_\xi\xi_\mathbf{t} * W_{g_2}\mathbf{g_t})). \tag{16}$$

Then, the upper level gated network is trained to infer the intermediate goal $\mathbf{g_t}$ provided by the lower level network, given $\mathbf{m_t}$ and $\mathbf{a}$, by minimizing the difference between $\mathbf{g_t}$ and $\mathbf{g}(\mathbf{m_t}, \mathbf{a})$ (12). Finally, a global gradient descent involving all matrices is performed to fine-tune the network to minimize the global prediction error of $\dot{\mathbf{x}}_\mathbf{t}$.

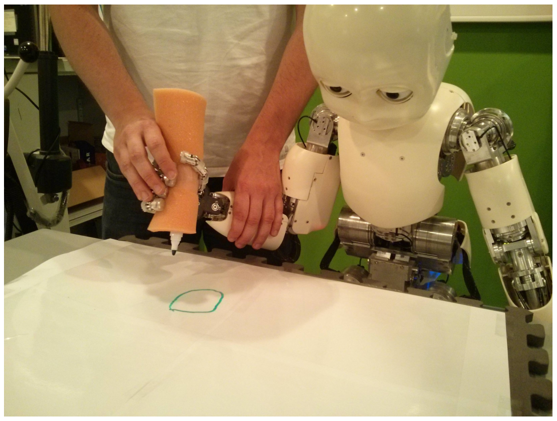

Fig. 3. Experimental setup. The robot is controlled in zero-torque mode, while a human operator moves its arm to write the digits between 0 and 9. The three dimensional Cartesian positions of the end-effector are recorded at approximately 100Hz.

TABLE I
NUMBER OF UNITS USED FOR THE EXPERIMENT

Layer	Number of units
$\mathbf{a}$	10
$\mathbf{m_t}$	100
factors ($\mathbf{m_t}$,$\mathbf{a}$)	1000
$\mathbf{g}(\mathbf{m_t}, \mathbf{a})$	100
$\xi_\mathbf{t}$	100
factors ($\mathbf{g}$,$\xi_\mathbf{t}$)	100
$\tilde{\xi}_\mathbf{t}$	100

V. EXPERIMENTS

We test the architecture on a writing task with the iCub humanoid robot [33]. We record the arm trajectories of the robot being manipulated by a human operator to write the digits between 0 and 9 [34]. Each digit basically corresponds to a different action from a repertoire. A total of 76 trajectories for each digit are recorded and used to train the network. For each trajectory, the initial position is considered as the origin of the three dimensional Cartesian frame. Each trajectory then consists of the sequence of Cartesian positions sampled at approximately 100Hz (providing between approximately 100 points for “short” digits like 1 and 500 points for longer digits like 8). Figure 3 illustrates the experimental setup. Some of the recorded trajectories are shown in Fig. 4.

The number of units in each layer of the network is given in Table I. The time constant α of the exponential window is set to 0.02, and we use a learning rate of 0.001 with a momentum of 0.95. The memory $\mathbf{m_t}$ is set to 0 at the beginning of each trajectory.

A. *Repertoire of actions*

First, we train the network to learn the repertoire of actions corresponding to the ten digits between 0 and 9. For this purpose, the vector $\mathbf{a}$ consists of a boolean vector with ten units and, for each trajectory, the unit corresponding to the performed digit is set to 1 (human labeling). The network

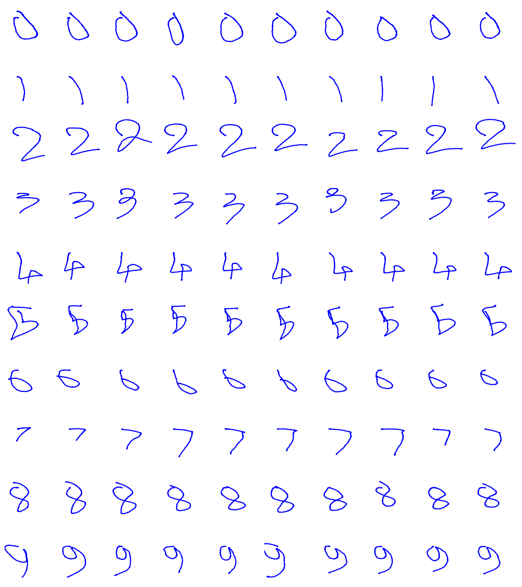

Fig. 4. Samples of recorded trajectories from the iCub robot. Each trajectory is plotted in a 12x8 cm box. Only two dimensions are plotted, the third being constant for most digits, except for 4 and 5 which necessitate to lift the pen.

is trained on the recorded trajectories using mini-batches of 1000 points, i.e. 1000 Cartesian positions along with the corresponding Cartesian velocities. These mini-batches are obtained by concatenating trajectories randomly chosen from the dataset until 1000 points are obtained. The network is then tested to generate trajectories. For this purpose, we close the loop between positions $\mathbf{x_t}$ and predicted velocities $\hat{\dot{\mathbf{x}}}_\mathbf{t}$ by simulating the movement of the end-effector according to:

$$\mathbf{x_{t+1}} = \mathbf{x_t} + 0.001 \times \eta \times (\hat{\dot{\mathbf{x}}}_\mathbf{t} + \nu) \tag{17}$$

where η is a noise modeling a variable control loop delay (Poisson noise, $\lambda = 10$)[3] and ν is a noise modeling an approximate command of the robot, induced for instance by an imprecise kinematic model (independent Gaussian noise, $std = 0.0025 m/s$, which approximately corresponds to an average of 5% error between the desired velocity and its execution). Both noises aim to test the robustness of the learned trajectories. The process is started from the initial position $(0, 0, 0)$ and is iterated for a number of time steps equal to the mean duration of recorded trajectories for the considered digit. Figure 5 shows some generated trajectories.

B. *Parametrization of actions*

In the previous experiment, we trained the network to learn a set of different primitives based on a symbolic

[3]This generates a control loop delay $0.001 \times \eta$ whose mean is 0.01s and standard deviation is about 0.003s. It ranges approximately between 0 and 0.025s

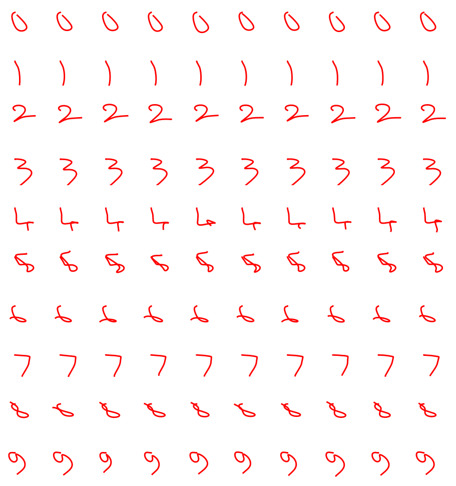

Fig. 5. Trajectories generated by the network. Each trajectory is plotted in a 12x8 cm box. Noise is added during the generation of each trajectory to evaluate the robustness of the network.

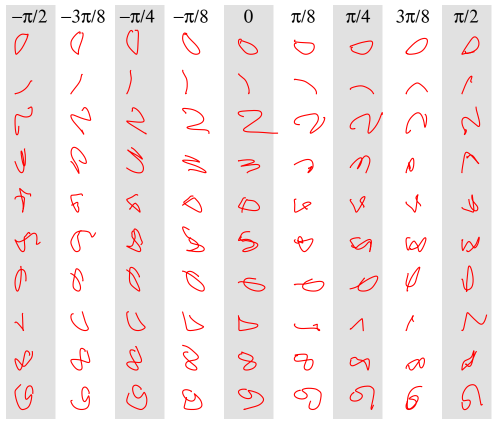

Fig. 6. Trajectories generated by the network for different rotation angles. Digits on gray background correspond to rotation angles which were presented to the network during training (only one trajectory for each (digit, rotation angle) pair was provided for training). Other trajectories show the generalization capabilities of the network.

representation of actions. However, the network can also handle a parametrization of such actions. To illustrate this capability, we generate new trajectories from the recorded dataset, by rotating one sample of each digit by an angle $\Theta \in \{-\pi/2, -\pi/4, 0, \pi/4, \pi/2\}$ (we apply the corresponding rotation matrices to all points of the trajectories). This provides a total of 50 trajectories for the training set (5 trajectories for each digit). This contrasts with the previous experiment based on 760 demonstrations, more than usually used by DMP-based approaches. The rotation parameter is added to the top layer $\mathbf{a}$ as two additional units taking values $\frac{1}{2}(1 + cos(\Theta))$ and $\frac{1}{2}(1 + sin(\Theta))$. The resulting top layer $\mathbf{a}$ is thus composed of 10 binary units to represent the digits, and two real-valued units representing the rotation angle. All the other parameters are the same as for the previous experiment.

After training, the network is used to generate trajectories with rotation angles $\Theta = k\frac{\pi}{8}$ for k in $\{-4, \cdots, 4\}$. No noise is added for the generation (17), the control loop delay is set to 10ms. Figure 6 shows the resulting trajectories.

VI. Discussion

In our work, the role of the memory layer, even in its very simple current implementation (11), is visible in Fig. 5 for digits with sharp corners (e.g. digits 2 and 9) and for digits with cross points (e.g. digits 4, 6 and 8). It actually allows the network to produce different desired velocities for a same current position $\mathbf{x}$, without explicitly modeling features such as corners or cross points. In a DMP approach, this would require either carefully chosen parametrized primitives, or a segmentation of such trajectories into several segments. However, since our approach does not use prior knowledge, it can prevent the network from learning very specific features, like for digit 5 for which the link between the lower part of the digit and the horizontal upper line is not considered as a more important point of the trajectory than all others. Improving such trajectories would necessitate either an optimization framework such as reinforcement learning to drive the network towards better trajectories, or a more sophisticated memory $\mathbf{m_t}$ able to extract and store relevant information about important via-points. This latter property is typically hand-crafted in classical DMPs, but recent probabilistic approaches aim at capturing such features [35].

Our approach aims at endowing control architectures with the powerful dimensionality reduction capabilities of deep networks [10]. First, it separates the lowest level of control from the high level trajectory planning by introducing the intermediate goal representation $\mathbf{g}(\mathbf{m_t}, \mathbf{a})$. Moreover, using gated connections, it factorizes knowledge from which we can expect an increasingly faster learning of new trajectories when the amount of knowledge increases. However, further experiments with more complex tasks are necessary to quantify such expected improvements.

Few assumptions are made about the input of the proposed architecture. Other state/command variables can be used, but also other representations of high level actions can be provided as input. For instance, a representation based on clustering of another modality, such as images or spoken words, can be used to trigger the execution of one action or another. This paves the way for loosely supervised interaction scenarios, for instance for programming by demonstration experiments. The possibility to use the high-level representation of context to parametrize the execution of an action also increases the expressiveness of our approach compared to the current theory of associative skill memories, which, as outlined in [11], makes the assumption that "the movement generated for a particular skill should be as stereotypic as possible" and leaves context sensitivity for future work.

VII. Conclusion and future work

In this paper, we proposed an architecture able to learn a repertoire of actions based on a factorization of representations. It relies on standard deep learning principles, that is the minimization of a reconstruction error by gradient descent. This architecture can be naturally plugged into a wider architecture, which can use powerful dimensionality reduction capabilities of deep networks to provide a compact and meaningful representation of perception. In turn, this can be used to efficiently learn actions grounded in contexts from complex environments, which is a key step for affordance learning. This will be the subject of future work.

In the experiments we discussed, the memory $\mathbf{m_t}$ has been hard-coded as an exponential window over past positions (11). This restricts the network function to learning only local temporal regularities, whose spread depends on the chosen time constant. Further work will consist in learning the recurrent weights from the memory layer, possibly involving long short term memory units [17], to make the network more generic. This will require top-down interactions from the upper layer $\mathbf{a}$, to be able to select and memorize relevant information depending on the overall goal. Direct interactions between the memory $\mathbf{m_t}$ and the goal $\mathbf{a}$ are also necessary in order to learn to segment and cluster continuous time sequences.

Another direction for future work consists in stacking several layers of this architecture. The top layer $\mathbf{a}$ of the proposed architecture could be considered as an intermediate goal for a higher, more global objective. Such an approach could be fruitful to efficiently learn sequences of actions.

References

[1] J. Piaget, *The Origins of Intelligence in Children*. WW Norton & Co, 1952.

[2] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with expanded function approximators," in *Proc. of the IEEE Int. Conf. on Humanoids Robotics*, 2013, pp. 1–7.

[3] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[4] E. Ugur, E. Sahin, and E. Oztop, "Affordance learning from range data for multi-step planning," in *EpiRob*, 2009.

[5] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Int. Conf. on Robotics and Automation (ICRA2009)*, 2009.

[6] G. Neumann, W. Maass, and J. Peters, "Learning complex motions by sequencing simpler motion templates," in *Proc. of the 26th Annual Int. Conf. on Machine Learning*, 2009, pp. 753–760.

[7] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a mixture of motor primitives," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS Int. Conf. on*. IEEE, 2010, pp. 411–416.

[8] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Learning sequential motor tasks," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[9] C. Daniel, G. Neumann, and J. Peters, "Hierarchical relative entropy policy search," in *Int. Conf. on Artificial Intelligence and Statistics*, 2012.

[10] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[11] P. Pastor, M. Kalakrishnan, F. Meier, F. Stulp, J. Buchli, E. Theodorou, and S. Schaal, "From dynamic movement primitives to associative skill memories," *Robot. Auton. Syst.*, vol. 61, no. 4, pp. 351–361, Apr. 2013.

[12] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[14] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[15] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[16] S. Babinec and J. Pospíchal, "Two approaches to optimize echo state neural networks," in *Proc. of the 11th Int. Conf. on Soft Computing, Mendel*, 2005, pp. 39–44.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] A. Graves, H. Bunke, S. Fernndez, M. Liwicki, and J. Schmidhuber, "Unconstrained online handwriting recognition with recurrent neural networks," in *in Advances in Neural Information Processing Systems 20*. MIT Press, 2008.

[19] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, no. 4, pp. 319–340, 2011.

[20] M. Langkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, no. 0, pp. 11 – 24, 2014.

[21] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Two Distributed-State Models For Generating High-Dimensional Time Series," *J. Mach. Learn. Res.*, vol. 12, pp. 1025–1068, 2011.

[22] I. Sutskever, G. E. Hinton, and G. W. Taylor, "The Recurrent Temporal Restricted Boltzmann Machine." in *Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. MIT Press, 2008, pp. 1601–1608.

[23] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[24] O. Delalleau and Y. Bengio, "Shallow versus Deep Sum-Product Networks," in *NIPS*, 2011, pp. 666–674.

[25] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *AISTAT*, vol. 5, 2009.

[26] Y. Bengio, A. Courville, P. Vincent, and U. Montreal, "Representation Learning: A Review and New Perspectives," *arXiv*, vol. 1206.5538v2, pp. 1–34, 2012.

[27] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in *Advances in Neural Information Processing Systems*, 2006.

[28] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. of the 25th Int. Conf. on Machine learning - ICML '08*. New York, New York, USA: ACM Press, 2008, pp. 1096–1103.

[29] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction," in *Proc. of the 28th Int. Conf. on Machine Learning*, 2011, pp. 833–840.

[30] R. Memisevic and G. Hinton, "Unsupervised Learning of Image Transformations," in *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition*. IEEE, 2007.

[31] A. Droniou and O. Sigaud, "Gated autoencoders with tied input weights," in *Proc. of The 30th Int. Conf. on Machine Learning*, 2013, pp. 154–162.

[32] R. Memisevic, "Learning to relate images: Mapping units, complex cells and simultaneous eigenspaces," *ArXiv e-prints*, 2012.

[33] L. Natale, F. Nori, G. Metta, M. Fumagalli, S. Ivaldi, U. Pattacini, M. Randazzo, A. Schmitz, and G. Sandini, "The icub platform: a tool for studying intrinsically motivated learning," in *Intrinsically motivated learning in natural and artificial systems - Ed. Baldassarre, G. and Mirolli, M.* Springer-Verlag, 2013, pp. 433–458.

[34] S. Ivaldi, M. Fumagalli, M. Randazzo, F. Nori, G. Metta, and G. Sandini, "Computing robot internal/external wrenches by means of inertial, tactile and f/t sensors: theory and implementation on the icub," in *Proc. of the 11th IEEE-RAS Int. Conf. on Humanoid Robots - HUMANOIDS*, Bled, Slovenia, 2011, pp. 521–528.

[35] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in Neural Information Processing Systems 26*, 2013, pp. 2616–2624.