Robotics and Autonomous Systems 135 (2021) 103690

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Robot skill learning in latent space of a deep autoencoder neural network

Rok Pahič [a,b,*], Zvezdan Lončarević [a,b], Andrej Gams [a], Aleš Ude [a,c]

[a] Humanoid and Cognitive Robotics Lab., Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[c] Faculty of Electrical Engineering, University of Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 1 April 2020
Received in revised form 30 October 2020
Accepted 2 November 2020
Available online 11 November 2020

Keywords:
Skill learning
Latent space representations
Deep autoencoder neural networks

ABSTRACT

Just like humans, robots can improve their performance by practicing, i. e. by performing the desired behavior many times and updating the underlying skill representation using the newly gathered data. In this paper, we propose to implement robot practicing by applying statistical and reinforcement learning (RL) in a latent space of the selected skill representation. The latent space is computed by a deep autoencoder neural network, with the data to train the network generated in simulation. However, we show that the resulting latent space representation is useful also for learning on a real robot.

Our simulation and real-world results demonstrate that by exploiting the latent space of the underlying motor skill representation, a significant reduction of the amount of data needed for effective learning by Gaussian Process Regression (GPR) can be achieved. Similarly, the number of RL epochs can be significantly reduced. Finally, it is evident from our results that an autoencoder-based latent space is more effective for these purposes than a latent space computed by principal component analysis.

1. Introduction

One of the main prerequisites for robots to operate outside of structured environments is the ability to continuously learn and adapt actions and motor skills [1]. By acting in the real world and accumulating new knowledge, a robot can gradually improve its performance [2], which is an important step towards achieving the dream of lifelong robot learning [3].

Learning complete actions and/or skills from scratch is in most cases not feasible because the search space is too large [4]. A better approach is often to initiate the learning process by observing a skilled teacher performing the desired task, i.e. by observing human demonstrations [5]. However, unless the robot can generalize from the available human demonstrations, it is unlikely that the accumulated knowledge would be directly applicable in all possible states of the real world [6]. Skill transfer from a human might also not achieve the same outcome when performed by a robot due to the correspondence problem [7], or due to the nature of the task itself.

To adapt to the current state of the environment and perform the desired task, a new skill instance could be synthesized using a database of previous executions of the applicable skill. If the skill instances in the database are related to each other through a known set of parameters describing the task, then a good skill instantiation for the current task description can be computed by statistical generalization of skill executions stored in the database [8]. In this paper, we refer to this set of parameters as the *goal* of an action or a *query point*. If statistical generalization does not result in an appropriate action to fulfill the desired task, then the computed skill parameters must be adapted, e. g. by reinforcement learning (RL).

Reinforcement learning provides a framework and a set of tools for learning of sophisticated and hard-to-engineer behaviors [9]. However, the high number of degrees of freedom (DOFs) typical for robots as well as the continuous state and action space make RL notoriously difficult in practical applications [10]. The use of parameterized policies, policy search methodologies [11], and the exploitation of initial knowledge, e. g. from human demonstration, can somewhat alleviate this problem. Nevertheless, it is often necessary to reduce the dimensionality of the RL problem to make it tractable.

Different dimensionality reduction methods have been applied in the past to reduce the learning space. A parametric representation for robot control policies, e. g. the well-known Dynamic Movement Primitives (DMP) [12], provides a relatively low-dimensional representation of the action space. However, the dimensionality of the DMP parameter space is still rather

* Corresponding author at: Humanoid and Cognitive Robotics Lab., Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia.
E-mail addresses: rok.pahic@ijs.si (R. Pahič), zvezdan.loncarevic@ijs.si (Z. Lončarević), andrej.gams@ijs.si (A. Gams), ales.ude@ijs.si (A. Ude).

https://doi.org/10.1016/j.robot.2020.103690

high for learning [6]. One of the best known general methods for dimensionality reduction is Principal Component Analysis (PCA), which projects the data onto the vector space spanned by basis vectors defined by the variance of the data [13]. Dimensionality reduction and regression have been combined in Locally Weighted Projection Regression (LWPR) [14], an algorithm that achieves nonlinear function approximation in high dimensional spaces even in the presence of redundant and irrelevant input dimensions. Another common method are the (deep) autoencoder networks [15], where the data is pushed through the layer with the smallest number of neurons — the latent space.

1.1. Main contributions

In this paper we propose and experimentally evaluate the process of obtaining new skills by exploiting a latent space representation defined by a deep autoencoder (AE) neural network. Our main aim was to show that both generalization and RL can be applied more effectively in latent space than in the original action space, which in our case is defined by the DMP parameters describing the desired skill.

One of the learning aspects under consideration was the problem of accumulating the database for learning. Statistical skill learning needs a database to generalize from and training of a deep autoencoder requires an even larger database [16]. In this paper we show that an autoencoder trained on a database of simulated data can be used to compute the latent space of real robot actions.

We verified the proposed approaches by learning robotic ball throwing in latent spaces. Our experiments show that by computing an autoencoder from simulated robot throwing data, we obtain an effective latent space representation for reinforcement learning and statistical generalization of throwing movements. Furthermore, this way faster convergence of RL methods can be achieved and a smaller database of throwing actions is needed to synthesize accurate real-world throwing movements by statistical generalization. This is intuitive because the dimensionality of the resulting optimization problems is smaller in latent spaces.

2. Related work

Dynamic movement primitives (DMP) are often taken as motor representation in reinforcement learning (RL) [9,10]. The dimensionality of learning DMP parameters in combination with tactile and visual feedback has been deemed too large in some practical applications [17], prompting RL in latent space of actuator redundancies. Latent spaces defined by autoencoders were applied for this purpose [17,18]. Deep autoencoders and variational autoencoders have also been used to train movement primitives in a low-dimensional latent space [19,20]. It is clear that a deep autoencoder neural network can greatly reduce the dimensionality of the movement representation. However, depending on the size of the latent space, it can also reduce the accuracy of the representation [20]. Another way to reduce the dimensionality of the search space is by constraining the parameter space with statistical generalization [6,21]. This way the learning process can be sped up. However, constraining the learning space can leave out some valid solutions. Exploration in RL can also be constrained to proceed only along the most significant directions in the parameter space [22,23].

Statistical learning using a database of example skill executions and task descriptors has been applied to compute DMP parameters in the past. Methods such as Gaussian Process Regression (GPR) [24], Locally Weighted Regression (LWR) [8,25], and deep neural networks [26] have been applied for this purpose. The application of GPR was extended to compliant DMPs [27], Cartesian DMPs [28], arc-length DMPs [29], and for the autonomous generation of the training database [2]. In this paper, we analyze the performance of GPR when statistical learning is done in a latent space defined by either autoencoders or PCA and compare the results to the standard generalization in the DMP parameter space.

Task parameters were also applied for learning of parameterized skills [30], where the authors propose adapting the DMP weights of a single demonstration based on the task parameter. Parameterized skill memories [31] enable task specific generalization from a low number of examples and are based on DMPs. Methods based on other trajectory representations were developed, too. For example, the Mixture of Motor Primitives (MoMP) [32] was used to obtain a task policy that is composed of several movement primitives weighted by their ability to generate successful movements in the given task context. Zhou et al. [33] propose to apply mixture density network for the mapping from the task parameter query to the parameters describing the movement primitives distribution. The approach of Calinon [34] is centered around task-specific Gaussian Mixture Models (TP-GMM). An extensive list of papers on learning of task-parameterized movements is provided in [34].

Statistical learning in latent spaces was analyzed in several papers. Zuo et al. [35] show that latent spaces can be explored in a controlled manner and argue that this complements various inference methods. Variational autoencoders (VAE) were used to compute latent spaces in this work. Le et al. [36] demonstrated that supervised dimensionality reduction architectures can provide improved generalization performance, which is also the key feature of our approach. Perhaps the most similar to ours is the work reported by Yoo et al. [37], where GPR is applied in the latent space defined by VAE. However, unlike in our work, where latent space generalization is applied for synthesizing new robot trajectories, generalization in Yoo et al. [37] was applied to visual data.

3. DMP latent space representations

We start by introducing the movement representation utilized throughout this paper, followed by the description of two latent space representations that can be used for training of motor skills encoded by the selected representation: autoencoder-based latent spaces and PCA-based latent spaces.

3.1. DMP parameter space

Dynamic Movement Primitives (DMPs) have been designed to represent any smooth robot motion. They can be used to provide a parametric representation of motor skills in the context of robot skill learning [12]. However, each particular motor skill usually spans only a low-dimensional manifold in the space of all possible robot movements. It should therefore be possible to map DMPs representing a desired skill to a lower dimensional parameter space (latent space). The idea is that learning in low-dimensional latent spaces should be faster and easier than learning in the full, usually high-dimensional parameter space.

According to Ijspeert et al. [12], a DMP that controls the motion of one degree of freedom is defined by a second order differential equation system (1)–(2) and phase equation (3)

$$\tau \dot{z} = \alpha_z(\beta_z(g - y) - z) + f(x), \tag{1}$$

$$\tau \dot{y} = z, \tag{2}$$

$$\tau \dot{x} = -\alpha_x x, \tag{3}$$

where $y \in \mathbb{R}$ is the robot control parameter while $x \in \mathbb{R}$ is the phase, which is introduced to avoid explicit time dependency.

There are a number of parameters that need to be specified, including the temporal scaling factor $\tau > 0$, the time constants $\alpha_z, \beta_z, \alpha_x > 0$, and the weights $\boldsymbol{\omega} \in \mathbb{R}^N$ that define a nonlinear forcing term $f(x)$

$$f(x) = \frac{\sum_{i=1}^{N} \omega_i \psi_i(x)}{\sum_{i=1}^{N} \psi_i(x)} x(g - y_0),$$
$$\psi_i(x) = \exp\left(-\frac{1}{2\delta_i^2}(x - c_i)^2\right).$$

The parameters of the forcing term are used to create a system response that follows any smooth point-to-point trajectory from the initial configuration y_0 to the final configuration g. c_i are the centers of N radial basis functions $\psi_i(x)$, while the parameters δ_i control their widths.

The parameters α_z, β_z, α_x, c_i and δ_i are usually set to constant values, whereas the initial and final position y_0 and g, time constant τ, and the weights $\boldsymbol{\omega}$ are obtained from the data. See [8, 12] for more details about how to compute these parameters. The desired commands for the robot (y, $\dot{y}$ and possibly $\ddot{y}$) are computed by integrating Eqs. (1)–(3) using Euler integration with the initial values set to $y = y_0,\ z = \tau\dot{y} = 0,\ x = 1$.

For robots with more degrees of freedom, each degree has its own control variables y and z, starting configuration y_0, final configuration g and weights $\boldsymbol{\omega}$. Time constant τ and phase x are shared between the degrees of freedom, which ensures that the motion of all the degrees of freedom is synchronized. Thus the full DMP parameter space has $d_{\text{DMP}} = (N + 2)n_{\text{DOF}} + 1$ parameters.

3.2. Autoencoder-based latent space

A deep autoencoder (AE) is a type of neural network, often applied for dimensionality reduction [19]. Deep autoencoders with nonlinear layers enable good dimensionality reduction while keeping the most relevant part of motion information in the reduced representation. During training, the AE learns how to copy its input data (in our case DMPs) to the output with the highest possible precision. Two parts comprise an autoencoder: an encoder and a decoder network (see Fig. 1). In the encoder part, data are pushed through the layers until they reach the layer with the smallest number of neurons (bottleneck). The decoder part expands the bottleneck layer so that the output data $\tilde{\boldsymbol{\theta}}^{DMP}$ match the input data $\boldsymbol{\theta}^{DMP}$ as well as possible. Thus we have $\mathbf{F}_{\text{dec}} \approx \mathbf{F}_{\text{enc}}^{-1}$. The latent space is defined by neurons of the bottleneck layer. We denote its dimension by d_{AE}, $d_{\text{AE}} < d_{\text{DMP}}$.

To train an autoencoder network, we need to gather a large number of skill executions and represent them with DMPs $\boldsymbol{\theta}_i^{\text{DMP}} \in \mathbb{R}^{d_{\text{DMP}}}, i = 1, \ldots, m$. The following criterion function is then optimized

$$\boldsymbol{\zeta}^\star = \arg\min_{\boldsymbol{\zeta}} \frac{1}{m} \sum_{i=1}^{m} \left\| \boldsymbol{\theta}_i^{\text{DMP}} - \mathbf{F}_{\text{dec}}\left(\mathbf{F}_{\text{enc}}\left(\boldsymbol{\theta}_i^{\text{DMP}}\right)\right) \right\|^2, \tag{4}$$

where $\boldsymbol{\zeta}^\star$ are the autoencoder parameters (weights and biases of neurons in the AE network). The precise structure of the deep AE neural network is usually determined experimentally by a network designer.

Once the network has been trained, we can compute the latent space representation of any given DMP $\boldsymbol{\theta}^{\text{DMP}} \in \mathbb{R}^d_{\text{DMP}}$ by applying the encoder part of the network,

$$\boldsymbol{\theta}^{\text{AE}} = \mathbf{F}_{\text{enc}}\left(\boldsymbol{\theta}^{\text{DMP}}\right) \in \mathbb{R}^{d_{\text{AE}}}. \tag{5}$$

Similarly, the decoder part of the network maps the latent space representation $\boldsymbol{\theta}^{\text{AE}}$ back to the DMP parameter space

$$\tilde{\boldsymbol{\theta}}^{\text{DMP}} = \mathbf{F}_{\text{dec}}\left(\boldsymbol{\theta}^{\text{AE}}\right) \in \mathbb{R}^{d_{\text{DMP}}}. \tag{6}$$

3.3. PCA-based latent space

Principal Component Analysis (PCA) [38] is a classical machine learning procedure for dimensionality reduction. It can be described as the orthogonal projection onto a low dimensional linear subspace such that the variance of the projected data is maximized. The main difference between autoencoders and the PCA is that the AE provides a nonlinear transformation to the latent space, whereas PCA results in a linear transformation.

Given the training data $\{\boldsymbol{\theta}_i^{\text{DMP}}\}_{i=1}^m$, $\boldsymbol{\theta}_i^{\text{DMP}} \in \mathbb{R}^{d_{\text{DMP}}}$, PCA is performed by computing the mean of the data $\bar{\boldsymbol{\theta}}^{\text{DMP}} = 1/m \sum_{i=1}^m \boldsymbol{\theta}_i^{\text{DMP}}$ and by forming the matrix $\boldsymbol{W}_{\text{PCA}} \in \mathbb{R}^{d_{\text{DMP}} \times d_{\text{PCA}}}$ composed of column eigenvectors associated with the largest eigenvalues of the data covariance matrix

$$\boldsymbol{S} = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}_i^{\text{DMP}} - \bar{\boldsymbol{\theta}}^{\text{DMP}}\right)\left(\boldsymbol{\theta}_i^{\text{DMP}} - \bar{\boldsymbol{\theta}}^{\text{DMP}}\right)^{\text{T}}, \tag{7}$$

where $d_{\text{PCA}} < d_{\text{DMP}}$ is the minimum number of eigenvectors needed to describe the variance in the data $\{\boldsymbol{\theta}_i^{\text{DMP}}\}_{i=1}^m$ with the required accuracy. d_{PCA} is often determined experimentally, but automated methods are also possible.

For any given DMP $\boldsymbol{\theta}^{\text{DMP}} \in \mathbb{R}^{d_{\text{DMP}}}$, its projection onto the PCA-based latent space is computed as follows

$$\boldsymbol{\theta}^{\text{PCA}} = \boldsymbol{W}_{\text{PCA}}^{\text{T}}\left(\boldsymbol{\theta}^{\text{DMP}} - \bar{\boldsymbol{\theta}}^{\text{DMP}}\right). \tag{8}$$

The formula below can be applied to map latent space parameters $\boldsymbol{\theta}^{\text{PCA}} \in \mathbb{R}^{d_{\text{PCA}}}$ back to the initial DMP parameter space

$$\tilde{\boldsymbol{\theta}}^{\text{DMP}} = \boldsymbol{W}_{\text{PCA}}\boldsymbol{\theta}^{\text{PCA}} + \bar{\boldsymbol{\theta}}^{\text{DMP}}. \tag{9}$$

4. Learning in AE- and PCA-based latent spaces

The main aim of this paper is to show that skill learning methodologies such as reinforcement learning and statistical learning can be implemented more effectively by exploiting low-dimensional latent space skill representations. In this section, we outline the application of reinforcement learning using a variant of PoWER method and the application of statistical learning method Gaussian Process Regression (GPR), both for skill learning in latent spaces. For experimental analysis, we implemented both learning approaches in the AE- and PCA-based latent space as well as in the full DMP parameter space.

4.1. Reinforcement learning in latent spaces

Reward weighted policy learning with importance sampling, which is a variant of Policy Learning by Weighting Exploration with the Returns (PoWER) method [10], was selected for testing the performance of RL in latent spaces. This method uses a parameterized skill policy and a reward function to maximize the expected return of skill performance trials. Its advantages are that it can be used with any policy representation (important when comparing the performance of different representations) and is robust with respect to reward functions. We provide our implementation of this method in Appendix A.

RL in both the DMP parameter space and latent spaces estimates the movement parameters using Eqs. (A.2)–(A.3). The only difference when RL is implemented in the latent space is that the estimated parameters $\boldsymbol{\theta}_n^{\text{AE}}$ and $\boldsymbol{\theta}_n^{\text{PCA}}$ need to be transformed back to the DMP parameter space to control the robot. In the case of AE-based latent space representation, this is performed using the decoder network

$$\tilde{\boldsymbol{\theta}}_n^{\text{DMP}} = \mathbf{F}_{\text{dec}}\left(\boldsymbol{\theta}_n^{\text{AE}}\right). \tag{10}$$

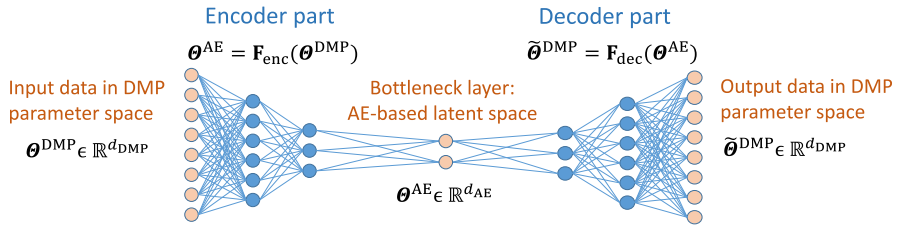


Fig. 1. Simple example AE network: the encoder part $\mathbf{F}_{\mathrm{enc}} : \mathbb{R}^{d_{\mathrm{DMP}}} \mapsto \mathbb{R}^{d_{\mathrm{AE}}}$ of the network is to the left of the bottleneck layer (with bottleneck neurons as output) and the decoder part $\mathbf{F}_{\mathrm{dec}} : \mathbb{R}^{d_{\mathrm{AE}}} \mapsto \mathbb{R}^{d_{\mathrm{DMP}}}$ to the right of the bottleneck layer (with bottleneck neurons as input).

Similarly, in the case of PCA-based latent space representation, we apply the formula

$$\tilde{\boldsymbol{\theta}}_n^{\mathrm{DMP}} = \boldsymbol{W}_{\mathrm{PCA}} \boldsymbol{\theta}_n^{\mathrm{PCA}} + \bar{\boldsymbol{\theta}}^{\mathrm{DMP}}. \tag{11}$$

Index n denotes the current iteration step of RL.

The reduced dimensionality of latent spaces is expected to have a positive effect on the convergence and stability of RL algorithms.

4.2. Gaussian processes regression in latent spaces

To test the performance of statistical skill learning in latent spaces, we applied Gaussian Process Regression (GPR) to estimate skills represented in the full DMP parameter space [24] or in AE- and PCA-based latent space. GPR has been selected because it had been demonstrated [39] that it outperforms other regression methods in difficult learning problems such as estimating the inverse dynamics of a seven degrees of freedom robot arm. See [39] for more details on the practical implementation of GPR.

For statistical learning, we need to gather example skill performances $\boldsymbol{\theta}_i^{DMP}$ together with the associated task descriptors $\boldsymbol{q}_i$, $i = 1, \ldots, m$, where m is the number of example skill executions. GPR then computes a mapping function that for each new desired task descriptor $\boldsymbol{q}_d$ predicts the corresponding control policy $\boldsymbol{\theta}_d^{\mathrm{DMP}}$ when learning takes place in full DMP parameter space or $\boldsymbol{\theta}_d^{\mathrm{AE}}$ and $\boldsymbol{\theta}_d^{\mathrm{PCA}}$ when learning takes place in AE- and PCA-based latent space, respectively. Thus, the following transformation functions are computed by GPR:

$$\mathbf{G}_{\mathrm{DMP}}(\{\boldsymbol{\theta}_i^{\mathrm{DMP}}, \boldsymbol{q}_i\}_{i=1}^m) : \boldsymbol{q}_d \mapsto \boldsymbol{\theta}_d^{\mathrm{DMP}}, \tag{12}$$

$$\mathbf{G}_{\mathrm{AE}}(\{\boldsymbol{\theta}_i^{\mathrm{AE}}, \boldsymbol{q}_i\}_{i=1}^m) : \boldsymbol{q}_d \mapsto \boldsymbol{\theta}_d^{\mathrm{AE}}, \tag{13}$$

$$\mathbf{G}_{\mathrm{PCA}}(\{\boldsymbol{\theta}_i^{\mathrm{PCA}}, \boldsymbol{q}_i\}_{i=1}^m) : \boldsymbol{q}_d \mapsto \boldsymbol{\theta}_d^{\mathrm{PCA}}. \tag{14}$$

After computing the AE- and PCA-based latent space parameters using Eqs. (13) and (14), respectively, we can compute the associated DMP control policy for skill execution by applying the same transformation as in the case of RL, i.e. Eqs. (10) and (11).

5. Experimental setup

The experiments in this paper focus on the task of robotic ball throwing at a target. We treat throwing as a planar problem in the vertical plane, i. e., in the saggital plane of the robot. The orientation of the plane is assumed to be correct. There is no loss of generality due to this assumption as we can reorient the robot towards the target if the orientation is not correct. The target, in our case a basket, is therefore displaced in the distance and the height from the robot's base.

We used a 7 DOF robot arm Mitsubishi PA-10 for robotic throwing. Three DOFs of the robot, which contribute to its motion in the saggital plane, were used to realize ball throwing. The simulation setup is depicted in Fig. 2a, while the real-world setup is shown in Fig. 2b. We used MuJoCo [40] for dynamic simulation of robot throwing. We put a ball holder into the robot hand both in simulation and on the real robot and the ball is placed onto the holder, but is not firmly attached. Thus when the throwing action is carried out, both in dynamic simulation and on the real robot, the ball detaches itself from the holder once the hand motion starts slowing done, i.e. after the acceleration falls to zero and becomes negative. This is different than in human ball throwing where the ball is usually firmly held by the fingers before being released.

Throwing was selected because it was previously studied in the context of statistical generalization [8] and reinforcement learning [41]. It can thus provide benchmarks to compare the effectiveness of different methods when applied in the latent space or in the full motion space defined by DMP parameters.

5.1. Generating AE and PCA-based latent spaces

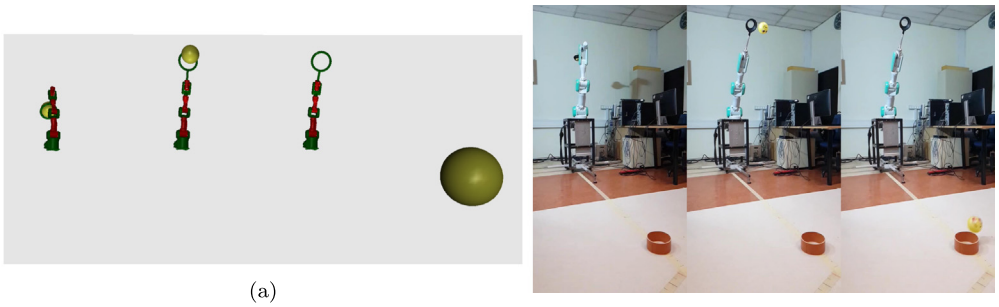
To compute the AE- and PCA-based latent spaces, we first generated the training dataset

$$\mathbf{D} = \left\{\boldsymbol{\theta}_j^{\mathrm{DMP}}\right\}_{j=1}^{P}, \tag{15}$$

which consists of P robot throwing trajectories representing the DMP parameters $\boldsymbol{\theta}_j^{DMP}$ describing the joint motion of the three degrees freedom that are relevant for throwing. With $N = 20$ DMP weights for each DOF we get 60 weights and together with 3 starting points, 3 goals and 1 common time constant this adds up to 67-dimensional DMP parameter space $\boldsymbol{\theta}^{DMP}$.

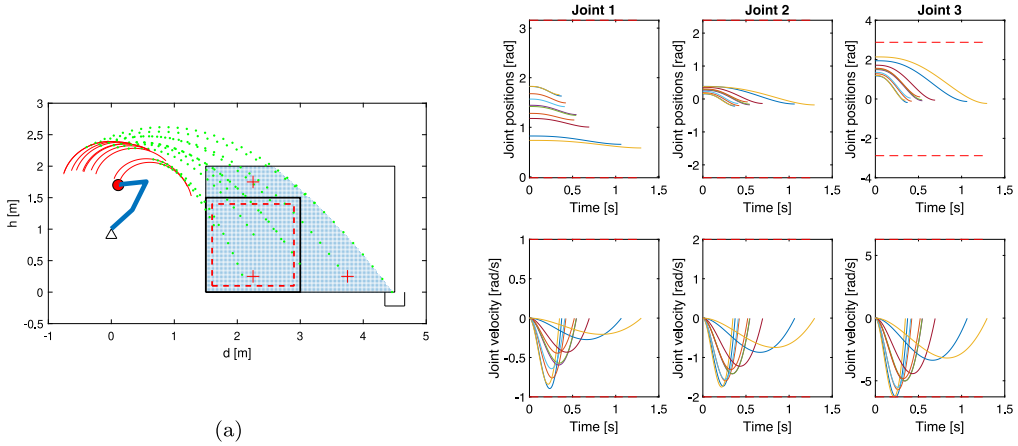
The example throwing trajectories for training a deep autoencoder neural network were generated using the procedure described in Appendix B. A database for a target grid with the distances in the range from 1.5 to 4.5 m and the heights in the range from 0 to 2 meters was generated, with 48 equally spaced target points per meter in both dimensions. We discarded all trajectories and targets that resulted in joint positions or velocities outside of real robot joint and joint velocity limits. This way we gathered 9824 example throwing trajectories (see also Fig. 3a). The targets for which executable robot trajectories could be generated are marked with small blue dots in a target rectangle of 3 m $\times$ 2 m (the black rectangle). A subset of trajectories associated with targets in the black square was used to train GPR, while a subset of trajectories associated with targets in the red dashed square was used for testing the performance of GPR. The red crosses mark the targets for testing reinforcement learning. The data are available for download at [42].

To compute an AE-based latent space and reduce the dimensionality of the learning problems, we designed a deep AE neural network with a 3-dimensional bottleneck layer, which defines its latent space. The size of the latent space was determined experimentally by reducing its size until the accuracy of transformation from latent space to DMP parameter space started to drop significantly. The resulting AE network was comprised of 5 hidden layers with 15, 10, 3, 10, and 15 neurons, as shown in Fig. 4a. For the activation function of each hidden layer we used $\mathbf{y} = \tanh(\mathbf{W}_{\mathrm{AE}}\boldsymbol{\theta} + \mathbf{b}_{\mathrm{AE}})$, where $\boldsymbol{\zeta}^{\star} = \{\mathbf{W}_{\mathrm{AE}}, \mathbf{b}_{\mathrm{AE}}\}$ are the AE parameters and $\boldsymbol{\theta}$ denotes the input to the neurons. The activation function



(a) (b)

Fig. 2. Experimental setup for evaluation of reinforcement learning and statistical learning in different spaces. (a) Mujoco dynamic simulation. (b) Mitsubishi PA-10 robot in its initial posture (left), after the release of the ball (center), and when the ball lands (right).



(a) (b)

Fig. 3. (a) Kinematic simulation for creating the datasets for training and testing, with the training and testing areas marked by rectangles. Examples of the computed end-effector trajectories for ball throwing are presented in red with the corresponding ball flight trajectories in green. (b) The computed joint trajectories and joint velocities corresponding to the end-effector trajectories. Dashed red lines mark the boundary values for joint positions and velocities. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of the output layer was linear. For autoencoder training we used 70% of trajectories contained in dataset (15). The remaining 15% of data were used for validation and 15% for testing.

The three-dimensional AE-based latent space can also be used to visualize the data. Fig. 4b shows a 3-dimensional plot of the computed latent values. The spread and the connected shape of the latent space data promise a good generalization performance.

Using dataset (15), we also conducted PCA analysis and obtained the following largest eigenvalues: $\boldsymbol{\lambda} = [1445980, 5769, 3623, 1058, 19, 1, \ldots]$. We chose the PCA latent space dimension to be equal to 3, even though it would also be possible to choose the PCA latent space dimension of 4 based on these values. We used 3 to fairly compare learning in PCA- and AE-based latent spaces. Namely, in our experiments the additional fourth latent space dimension significantly slowed down the reinforcement learning in the PCA-based latent space compared to when only 3 dimensions were used.

5.2. Dataset for statistical learning

Just like the computation of latent spaces, statistical learning needs training data, too. As discussed in Section 4.2, besides DMP parameters, statistical learning also requires the corresponding query points. We created three datasets to evaluate statistical learning with three different movement representations (DMP parameter space and AE- and PCA-based latent space of DMP parameters). Data in black square in Fig. 3a was used to create these datasets, which resulted in datasets composed of throwing trajectories for distance and height values in the range of 1.5 m–3 m and 0 m–1.5 m. The distance and height were discretized into 10 equidistant values, thus altogether we obtained 100 pairs of query points and throwing trajectories. The resulting throwing trajectories were used to compute the 67 dimensional DMP representation (see Section 3.1) and then transformed into a 3 dimensional AE- and PCA-based latent space representation (see Section 3). These trajectory representations were then used to generate executable robot trajectories in dynamic simulation, where the new landing distances and heights were computed. Note that these were slightly different for each representation as the mapping to latent spaces and DMP integration result in slightly different movements (see Fig. 5).

In the above data generation procedure, the query points were defined as

$$\mathbf{q} = [d, h]^{\mathrm{T}}, \tag{16}$$

where d is the distance and h the height of the throwing target. We obtained the following datasets for GPR training

$$\mathcal{G}^{\mathrm{DMP}} = \left\{\boldsymbol{\theta}_j^{\mathrm{DMP}}, \mathbf{q}_j^{\mathrm{DMP}}\right\}_{j=1}^{R}, \tag{17}$$

$$\mathcal{G}^{\mathrm{AE}} = \left\{\boldsymbol{\theta}_j^{\mathrm{AE}}, \mathbf{q}_j^{\mathrm{AE}}\right\}_{j=1}^{R}, \tag{18}$$

$$\mathcal{G}^{\mathrm{PCA}} = \left\{\boldsymbol{\theta}_j^{\mathrm{PCA}}, \mathbf{q}_j^{\mathrm{PCA}}\right\}_{j=1}^{R}. \tag{19}$$

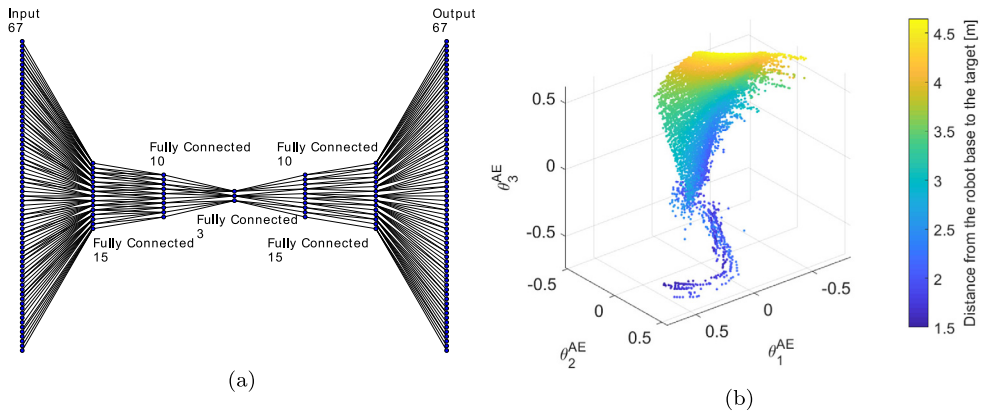


Fig. 4. (a) Illustration of the designed autoencoder structure. (b) Example points in the latent space, computed by projecting the training data shown in Fig. 3 to the latent space.

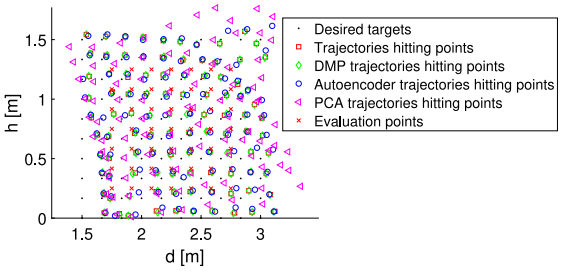


Fig. 5. Query points of datasets for statistical generalization and targets for testing.

where $\boldsymbol{\theta}_j^{\mathrm{DMP}}$, $\mathbf{q}_j^{\mathrm{DMP}}$ are the DMP throwing trajectories and the associated queries, while $\boldsymbol{\theta}_j^{\mathrm{AE}}$, $\mathbf{q}_j^{\mathrm{AE}}$ and $\boldsymbol{\theta}_j^{\mathrm{PCA}}$, $\mathbf{q}_j^{\mathrm{PCA}}$ are their AE- and PCA-based latent space counterparts, respectively.

6. Experimental results

First, we compared the effectiveness of PCA- and AE-based latent spaces by comparing their reproduction error. Then we evaluated the implemented learning processes with different trajectory representations in dynamic simulation and on a real robot.

6.1. Reproduction error of AE- and PCA-based latent spaces

The latent spaces generated by the proposed autoencoder network and principal component analysis are both three-dimensional and from the perspective of dimensionality comparable for learning. A good indication for how well each method has learnt the latent space is its reproduction error. We define the reproduction error as the difference between the original DMPs (trajectories) and trajectories computed by first applying Eq. (5) or (8) to respectively project the original DMPs onto the AE- and PCA-based latent spaces, followed by an application of Eq. (6) or (9) to map the latent space representations back to the DMP representation. The reproduction error for the jth joint trajectory can thus be computed as follows

$$E(j) = \frac{1}{T_j} \sum_{i=1}^{T_j} \|\mathbf{y}_j^{\mathrm{AE/PCA}}(x_{i,j}) - \mathbf{y}_j^{\mathrm{DMP}}(x_{i,j})\|, \tag{20}$$

where $x_{i,j} = x(t_{i,j})$ are the phases, $\mathbf{y}_j^{\mathrm{AE/PCA}}(x_{i,j})$ the robot joint configurations obtained by integrating the jth output DMP as calculated by the AE or PCA, and $\mathbf{y}_j^{\mathrm{DMP}}(x_{i,j})$ the robot joint configurations obtained by integrating the original jth DMP without latent space projection. T_j denotes the number of points for the jth DMP. A subset of the data described in Section 5.1 was used for testing (1474 examples), while the rest of the data was used to train AEs and compute PCA.

Table 1
Reproduction errors resulting from the projection onto the latent spaces computed by AE, PCA, and AE with linear activation functions. The results represent the average error (20) over all DMPs from the test set.

	Joint trajectory error [rad]
AE	0.0157 ± 0.0003
PCA	0.0781 ± 0.0016
AE with linear activation functions	0.0516 ± 0.0019

To evaluate the importance of handling nonlinear transformations, we performed an ablation analysis in which we removed the nonlinear activation functions and retrained the AE with only linear activation functions. We then compared the performance of the autoencoder with and without nonlinear activation functions. Note that the AE with linear activation functions has the same number of tunable parameters for learning as the AE with nonlinear activation functions. However, when only linear activation functions are present, AE can be shortened after training to an encoder matrix and a decoder matrix, each having the same number of parameters as the PCA matrix.

Results in Table 1 show that the average reproduction error is smaller with AEs than with PCA. This is consistent with other results presented later in Sections 6.2 and 6.3. In general, a better reproductive performance can be expected when using AE-based latent spaces compared to PCA-based latent spaces. This is due to the ability of the AEs to perform nonlinear approximations [15]. Our results in Table 1 confirm this with an expected decrease in performance when nonlinear activation functions are removed. However, AEs with linear activation functions still have a smaller reproduction error than PCA. This might be because unlike with PCA, the latent space dimensions of the AEs do not have to be orthogonal.

6.2. Reinforcement learning experiments

The main focus of our RL experiments was to evaluate the stability and speed of convergence.

6.2.1. Dynamic simulation

Reinforcement learning for each trajectory representation was first tested in dynamic simulation with throwing at three different targets (see Fig. 3a). For each target, we carried out the reinforcement learning process (A.2) 15 times, altogether 45 times

for each representation. Each RL session was stopped if the robot hit the target or the number of rollouts exceeded 100. For all experiments, no matter the target or trajectory representation, we used the same initial approximation for the throwing trajectory, encoded into the representation that was being tested. As a result of each throw $\boldsymbol{\tau}_i$, we measured the shortest distance between the target $\boldsymbol{q}_T$ and the point on the throwing trajectory $\boldsymbol{q}_i$. The measured distance was used to compute the terminal reward for reinforcement learning

$$R(\tau_i) = \exp(-\|\boldsymbol{q}_T - \boldsymbol{q}_i\|^2). \tag{21}$$

The exploration noise was tuned for each learning space separately. It was lowered in each step to 98% of the previous exploration noise. Each fifth trial was executed without adding exploration noise to test the convergence. For AE-based latent space we took into account that activation function tanh is restricted to the interval $[-1, 1]$, thus we forced the latent space values with added noise into this interval.

The convergence of reinforcement learning for this task is shown in Fig. 6a, which shows the average error for each trajectory representation, computed as the smallest distance between the target and the ball. We also compared the speed of convergence by counting iterations until the first hit, where a throw was counted as a hit if the computed distance was less than 2 cm. The results are shown in the bar graph in Fig. 6b.

It is clear from these graphs that RL in latent spaces converges significantly faster than RL in full DMP parameter space, both in terms of distance to the target and the number of rollouts needed to hit the target. Not only is the convergence faster, but it is also more stable as can be observed from the size of the 95% confidence interval. The reason for this is that RL in full parameter space has a considerable chance to produce throws in which the robot looses the ball before making the actual throw. On the other side, learning in latent spaces limits the exploration to the actual throwing trajectories and thus finds the solution faster. It is also clear that AE-based latent space learning outperformed the PCA-based latent spaces, which is probably due to the nonlinear nature of deep AE neural networks.

6.2.2. Real robot experiments

Reinforcement learning with different representations was also tested on a real Mitsubishi PA-10 robot. The task of the robot was to throw the ball into a basket. To simplify the measurement process, we chose targets on the horizontal line and measured the horizontal difference between the target x_T and the ball landing position x_i. The terminal reward for reinforcement learning was thus computed as

$$R(\tau_i) = \exp(-(x_T - x_i)^2). \tag{22}$$

The size of the basket and the ball were such that the ball could miss the middle of the target by about 3 cm on each side but still land in the basket. The main difference between the aforementioned dynamic simulation and the real robot experiment is that on the real robot we cannot execute trajectories that violate its joint and/or joint velocity limits. A non-executable trial was marked with a 10 m error, and obtained an appropriately extremely low reward.

We selected two targets on the horizontal floor and applied reinforcement learning process (A.2) for each of the two targets 5 times, i. e., a total of 10, and obtained results shown in Fig. 7.

The experiments with the real robot confirm our simulation results. Reinforcement learning in the PCA-based and AE-based latent space is much faster and more stable than learning in the DMP space, which has many problems with generating executable throws that can hold the ball in the throwing spoon. The learning in AE-based and PCA-based latent spaces does not suffer from this problem because it keeps explorative throwing trajectories in the space spanned by the training throwing trajectories. Just like in simulation, we obtained better performance with learning in the AE-based latent space than in the PCA-based latent space.

Table 2
Average throwing error and its variance when applying GPR in three different learning spaces. 49 throws with targets uniformly spread within the selected area were used for generalization. Results represent the distance between the desired target and the closest point on the ball flight trajectory.

	Throwing error [m]
GPR in DMP parameter space	0.046 ± 0.029
GPR in AE-based latent space	0.024 ± 0.015
GPR in PCA-based latent space	0.030 ± 0.019

6.3. Evaluation of statistical learning in latent spaces

Reinforcement learning finds a suitable robot throwing trajectory for the given target. It does not take into account any previously acquired knowledge about throwing at different targets and always needs to start learning from scratch. We applied statistical learning method GPR (Gaussian Process Regression, see [39]) to exploit previous knowledge when throwing at new targets. In this section, we compare the accuracy of GPR for different skill representations.

6.3.1. Performance of GPR

In simulation we tested the performance of statistical generalization with Gaussian process regression (GPR) using data within the black square in Fig. 3a for training and data within the red dashed square for testing. The data at the edge of the training set were not used for testing because the performance of statistical learning deteriorates at the edge of the training area. We created a testing set using a grid of 7×7 testing targets, which were in-between the data points used for training (see Fig. 5). For each trajectory representation, we used the corresponding dataset defined in (12)–(14) to calculate GPR parameters (see [39]). GPR was then used to compute and execute throwing trajectories for each target in the testing set. We used dynamical simulation to perform the throwing actions. Results are shown in Table 2, where the average error and its variance for each representation are shown. In Fig. 8, the throwing errors are presented with a contour plot.

The results in Table 2 clearly show that GPR in the AE-based latent space outperforms the statistical learning in the other two parameter spaces. Not only the average error but also the variance between the results is smaller in case of applying GPR in the AE-based latent space. In Fig. 8 we can observe uniformly low errors for GPR in the E-based latent space. To a degree, the errors are still relatively low but larger in the PCA-based latent space. When applying GPR in the full DMP parameter space, we can observe noticeable differences between the areas of low and high errors.

6.3.2. Incremental dataset augmentation

We also tested how the performance of statistical learning improves as the training dataset used to compute GPR parameters becomes larger. This experiment was performed both in simulation and on the real robot. In both experiments, the dataset augmentation process followed the following procedure:

1. Compute GPR parameters using the initial training set $\{\boldsymbol{\theta}_k, \boldsymbol{q}_k\}_{k=1}^m$.
2. Generate a random target position $\boldsymbol{q}$ within the selected training area and compute the corresponding throwing trajectory $\boldsymbol{\theta}_{m+1}$ using GPR in the appropriate parameter space.

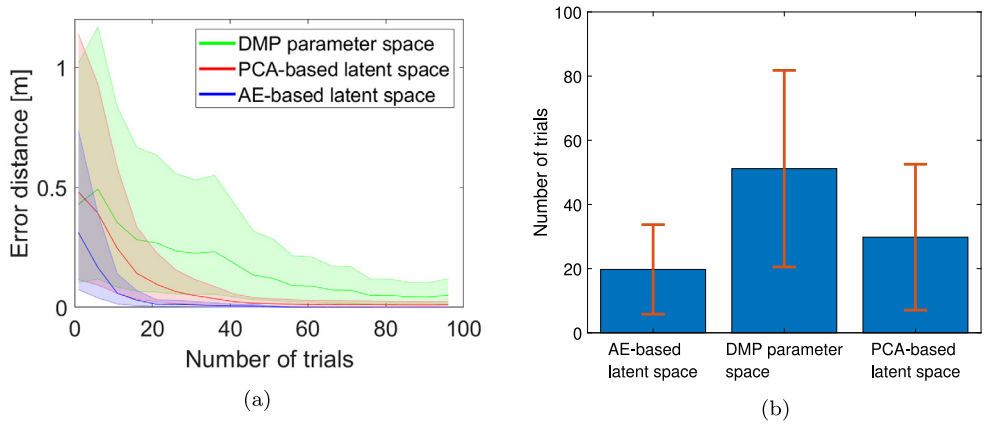


Fig. 6. Results of reinforcement learning on simulated data. (a) Average error distance after the specified number of trials for DMP representation, PCA-based and AE-based latent space representation, with the 95% confidence interval for the exponential distribution of the results. (b) Average number of rollouts to the first hit for reinforcement learning with different trajectory representation and the variance of the results.

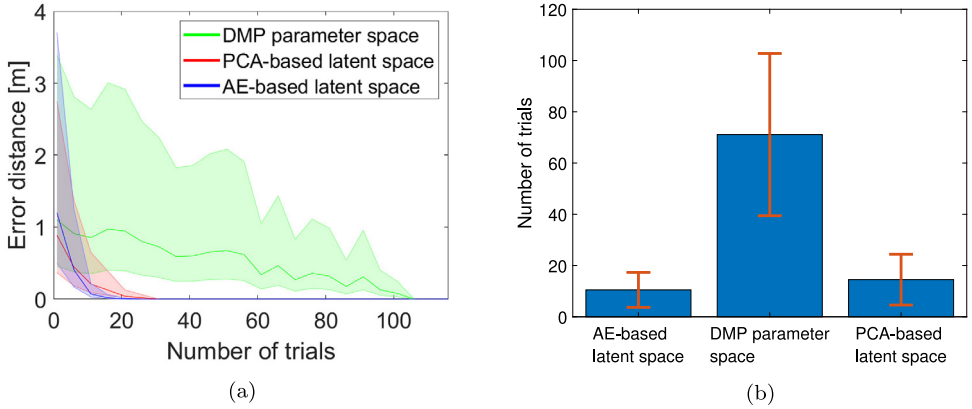


Fig. 7. Reinforcement learning of throwing movements on a real robot. (a) Average error distance after the specified number of trials for DMP representation, PCA-based and AE-based latent space representation, with 95% confidence interval for the exponential distribution of the results. (b) Average number of rollouts to the first hit for reinforcement learning with different trajectory representations and the variance of the results.

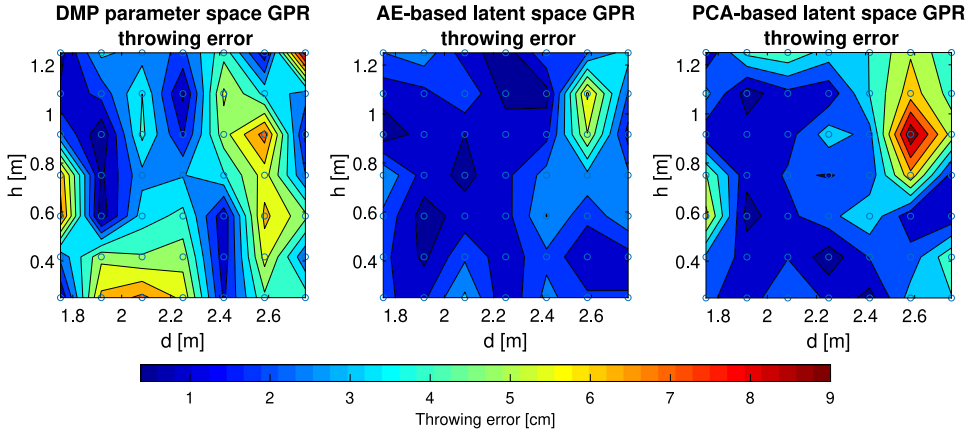


Fig. 8. Throwing error resulting from the throwing trajectories computed by GPR in different learning spaces. Circles represent different targets on a 7 × 7 target grid. Errors are in centimeters and represent the distance between the desired target and the closest point on the ball trajectory.

3. Execute the throwing trajectory in dynamic simulation or with a real robot.
4. Measure the actual ball landing position $\boldsymbol{q}_{m+1}$ and compute the distance between the actual landing position and the desired target position $\boldsymbol{q}$.
5. Augment the training dataset $\{\boldsymbol{\theta}_k, \boldsymbol{q}_k\}_{k=1}^{m+1}$ and compute the new GPR parameters.
6. Increase m and continue with Step 2.

In simulation, we first performed four throws to generate the initial training set at four targets situated at the edge of the testing area, which was the same black square area as in Fig. 3a. The random targets were then generated in the red dashed square area. For each skill representation, we repeated the data augmentation procedure 20 times. Each time we augmented the same initial dataset with 50 randomly selected targets. As the

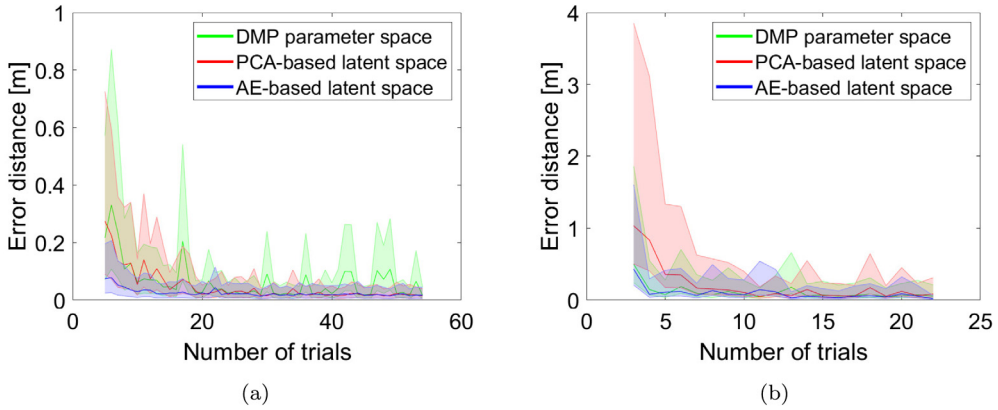


Fig. 9. Improved performance of statistical learning by autonomous dataset augmentation procedure. (a) Simulation experiment. (b) Real robot experiment. Both graphs show the average error of throwing and 95% confidence interval for the exponential distribution of the results for different trajectory representations as a function of the number of points added to the training set used to compute GPR.

datasets became larger, the error of throws decreased for all three skill representations, as shown in Fig. 9a.

The simulation results shown in Fig. 9a demonstrate that learning in the AE-based latent space was the fastest at reducing the error of throwing and achieved the most stable convergence. Learning in the DMP parameter space and PCA-based latent space also converged but needed more data to achieve the same average error. The plot also shows that the process of database augmentation in the full DMP parameter space was less stable, probably due to occasionally adding throwing trajectories with larger errors, as shown in Fig. 8 left.

On the real robot, we performed a similar experiment for targets distributed along the line at the distance from 2 m to 4 m. We first performed 2 throws resulted in the ball landing roughly at the edge of the training area. 20 random targets were then generated within the training area and the dataset augmentation procedure was performed. We repeated this procedure 3 times, starting from the same initial dataset for each trajectory representation. 20 additional throws were performed in each experiment.

The results of these experiments are shown in Fig. 9b. While learning in the AE-based latent space again performed the best, in these experiments we achieved similar performance by learning in the full DMP parameter space. This is probably the result of a small statistical sample and simplification of the experimental task to one dimension.

7. Summary and discussion

In this paper we demonstrated the advantages of motor learning in low-dimensional latent spaces compared to learning in a full motor parameter space, e.g. DMP parameter space. We showed that both reinforcement and statistical learning can be more effective in latent spaces. Even though the data for computing latent spaces were generated in simulation, the computed latent spaces were successfully used also to increase the performance of learning on the real robot. More specifically, our experiments show that the average error of statistical learning in latent spaces is lower and requires fewer data points for good performance. Similarly, reinforcement learning in latent spaces is significantly faster and more stable. In all our experiments, different forms of learning in the AE-based latent space outperformed learning in the PCA-based latent space. The achieved results align with the better approximation performance of deep autoencoder neural networks compared to PCA.

One reason why learning in restricted latent spaces is faster than learning in the full motor parameter space is that latent spaces limit exploration to the part of the motor space that is relevant for the desired task. However, this can be problematic if the latent space does not approximate the task-relevant part of the motor space well. This is the main reason why learning in the AE-based latent space outperformed learning in the PCA-based latent space. Namely, due to its ability to model nonlinear relationships, AE-based latent space is normally a better approximation of the task-relevant part of the motor space. A possible approach to improve the approximation quality of latent spaces is to generate additional data points by applying RL in the full motor parameter space and then add these data points to the dataset used for computing latent spaces.

When training an AE, the AE should not only learn to copy training data perfectly from input to output, but also learn to come as close as possible to the exact representation of training data in the latent space. The approach that emphasizes this aspect is the energy-based model for AE. The PCA already follows this approach by default through the method definition [43]. We believe that by implementing this approach for our AEs, e.g. by making use of denoising AEs or regularization of latent space [44], we can create even better AE latent spaces for motor learning. This is part of our future research.

In our experiments, we augmented the database for statistical learning by randomly generating additional query points (targets) and the associated throwing trajectories. However, this was only possible because we could first analytically compute the associated throwing trajectories using the method described in Appendix B. New data points for learning were then generated by executing the computed throwing trajectories in dynamical simulation or on a real robot. If it is not possible to analytically compute new training trajectories to acquire additional training data, then reinforcement learning, possibly performed in latent spaces, can be used to obtain new training data for statistical learning. This way the training database can be augmented in a fully model-free way.

Another venue for further research is to use imitation learning to acquire the initial database for latent space computation and subsequent learning. For example, human trajectories could be recorded during the desired task execution and used to compute latent spaces. Just like in our experiments, where latent spaces were computed using simulated data and improved the learning on a real robot, we expect that human-demonstrated task executions would provide similar benefits. However, since the variance of human-demonstrated trajectories is typically much larger, it might be necessary to use variational autoencoders to account for these variances, as shown in the work of Chen et al. [20]. The latent space of variational autoencoder could be used in the same way as our current, AE-based latent space.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has received funding from program group Automation, robotics, and biocybernetics (P2-0076) and young researcher grant PR-07602, both supported by the Slovenian Research Agency, and from EU's Horizon 2020 grant ReconCycle (GA no. 871352).

Appendix A. Reward weighted policy learning with importance sampling

The reinforcement learning method PoWER was originally developed by Kober and Peters and is particularly well-suited for learning of trial-based tasks in motor control [10]. Under certain assumptions, including the assumption that there is only terminal reward and that only a single basis function is active at any given time (note that this is only approximately true for DMPs), PoWER updates control policy parameters $\boldsymbol{\theta}_n$ as follows

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \frac{\langle(\boldsymbol{\Theta}_n - \boldsymbol{\theta}_n)R(\boldsymbol{\Theta}_n)\rangle_{w(\boldsymbol{\tau})}}{\langle R(\boldsymbol{\Theta}_n)\rangle_{w(\boldsymbol{\tau})}}, \tag{A.1}$$

where $\boldsymbol{\Theta}_n = \{\boldsymbol{\theta}_k^*\}_{k=1}^n$ denotes the set of all policy parameters $\boldsymbol{\theta}_k^*$ executed until the nth iteration and $R > 0$ the terminal reward received at the end of each trial. The expression $\langle\cdot\rangle_{w(\boldsymbol{\tau})}$ denotes importance sampling, which role is to select a predefined number of best trials to compute the update. The importance sampler can significantly reduce the number of required trials (rollouts) to compute the optimal control policy.

In the context of DMP parameter learning, the update rule (A.1) can be applied to estimate the weights of the DMP forcing term. It is equivalent to

$$\boldsymbol{\theta}_{n+1} = \frac{\sum_{i=1}^m R_{\text{in}(n,i)}\boldsymbol{\theta}^*_{\text{in}(n,i)}}{\sum_{i=1}^m R_{\text{in}(n,i)}}, \tag{A.2}$$

where function $\text{in}(n, i)$ selects the trial with the ith highest reward from the trial set $\{\boldsymbol{\theta}_k^*, R_k\}_{k=1}^n$ and m is the number of best trials used to compute the parameter update. The exploration parameters are computed by adding exploration noise to the current estimate $\boldsymbol{\theta}_n$

$$\boldsymbol{\theta}_n^* = \boldsymbol{\theta}_n + \boldsymbol{\varepsilon}_n. \tag{A.3}$$

Here $\boldsymbol{\varepsilon}_n$ is a zero mean Gaussian noise. Its variance $\boldsymbol{\Sigma}$ is usually a diagonal matrix specified by a user. Higher variance should be used when it is necessary to explore a larger area in the parameter space, but this could take a lot of time to converge, whereas lower variance results in faster convergence but could get stuck in a local minimum.

While a rigorous implementation of PoWER is applicable only to reinforcement learning of weights of the DMP forcing term, its simplified version specified by the update rule (A.1) and equivalently (A.2) can be applied also to learn other DMP parameters. We call the resulting method reward weighted policy learning with importance sampling. In our experiments we used this method to estimate the full DMP parameter set as well as the AE- and PCA-based latent space parameters.

Appendix B. Generation of simulated ball throwing trajectories

Neglecting the air drag, the motion of a free-flying ball can be modeled as follows

$$\boldsymbol{p}(t) = \begin{bmatrix} p_x^r + (t - t_r)v_0\cos(\alpha_r) \\ p_y^r + (t - t_r)v_0\sin(\alpha_r) - \frac{1}{2}(t - t_r)^2 g \end{bmatrix}, \tag{B.1}$$

where t_r denotes the time at which the robot releases the ball, $\boldsymbol{p}_r = [p_x^r, p_y^r]^{\mathrm{T}}$ the ball position at release time, $\dot{\boldsymbol{p}}_r = v_0[\cos(\alpha_r), \sin(\alpha_r)]^{\mathrm{T}}$ the ball velocity at release time, and α_r the angle of motion at release time. g is the gravitational acceleration. Eq. (B.1) can be re-written as a parabola in 2-D plane

$$p_y(t) - p_y^r = \tan(\alpha_r)(p_x(t) - p_x^r) - \frac{1}{2}\frac{g}{v_0^2\cos^2(\alpha_r)}(p_x(t) - p_x^r)^2. \tag{B.2}$$

All values are given in the robot base coordinate system.

Formula (B.2) was used in [8] to compute the angle

$$\alpha_r(\boldsymbol{p}_r, \alpha) = \arctan\left(2\frac{h - p_y^r}{d - p_x^r} - \tan(\alpha)\right) \tag{B.3}$$

and absolute velocity

$$v_0(\boldsymbol{p}_r, \alpha) = \sqrt{-\frac{g(d - p_x^r)^2}{2(\tan(\alpha)(d - p_x^r) - (h - p_y^r))\cos^2(\alpha_r(\boldsymbol{p}_r, \alpha))}} \tag{B.4}$$

of the robotic throwing movement at release time from the given release position $\boldsymbol{p}_r$, the final target position $\boldsymbol{q} = [d, h]^{\mathrm{T}}$, and angle α at which the ball should hit the target. Parameters d and h denote the distance and height of the target.

Unfortunately, the training data computed by hand-specifying the angle α and the release position $\boldsymbol{p}_r$ as in [8] turned out to be too simple to properly evaluate our learning processes. We therefore developed a 2-step procedure to generate a more variable set of throwing movements. In the first step, we calculate the robot joint configuration $\boldsymbol{y}_r$ at release time where the smallest joint velocity $\dot{\boldsymbol{y}}_r$ is needed to hit the target. Let $\mathbf{F}_{\text{fk}}$ denote the forward kinematics relating the 3 robot degrees of freedom $\boldsymbol{y}$ used for throwing in this experiment to the 2 Cartesian dimensions describing the throwing target, $\boldsymbol{p}_r = \mathbf{F}_{\text{fk}}(\boldsymbol{y}_r)$. We formulate the following optimization problem

$$\begin{aligned} &\arg\min_{\boldsymbol{y}_r, \alpha}\left\{\|\boldsymbol{W}_{\text{opt}}\dot{\boldsymbol{y}}_r\|^2 = \left\|\boldsymbol{W}_{\text{opt}}\boldsymbol{J}_r^+ v_0\,(\mathbf{F}_{\text{fk}}(\boldsymbol{y}_r), \alpha)\begin{bmatrix}\cos(\alpha_r(\mathbf{F}_{\text{fk}}(\boldsymbol{y}_r), \alpha)) \\ \sin(\alpha_r(\mathbf{F}_{\text{fk}}(\boldsymbol{y}_r), \alpha))\end{bmatrix}\right\|^2\right\}, \\ &\quad \text{subject to} \\ &\quad -90^\circ \le \alpha \le -35^\circ,\ \boldsymbol{y}^{\min} \le \boldsymbol{y}_r \le \boldsymbol{y}^{\max}, \end{aligned} \tag{B.5}$$

where $\boldsymbol{J}_r \in \mathbb{R}^{2\times 3}$ is the robot Jacobian at release configuration $\boldsymbol{y}_r$. Functions α_r and v_0 are defined by Eq. (B.3) and (B.4), respectively. The weight matrix $\boldsymbol{W}_{\text{opt}} = \text{diag}(1/|\dot{y}_1^{\max}|, 1/|\dot{y}_2^{\max}|, 1/|\dot{y}_3^{\max}|)$ was included to normalize the velocities with the maximum allowed joint velocities. To ensure that the thrown ball lands successfully in the basket, we limited the hitting angle α within a suitable range. The release velocities $\dot{\boldsymbol{y}}_r$ can also be computed at the optimal configuration $\{\boldsymbol{y}_r, \alpha\}$ using forward kinematics and formulas (B.3)–(B.4).

In the second step we use the computed release joint position and velocity values to create the throwing trajectory $\boldsymbol{y}(t)$. We use a fifth degree polynomial to represent the throwing motion. The coefficients of the polynomial are computed by setting the following values: $\dot{\boldsymbol{y}}(0) = \ddot{\boldsymbol{y}}(0) = 0$, $\boldsymbol{y}(t_r) = \boldsymbol{y}_r$, $\dot{\boldsymbol{y}}(t_r) = \dot{\boldsymbol{y}}_r$, $\ddot{\boldsymbol{y}}(t_r) = 0$ and $\dot{\boldsymbol{y}}(t_{end}) = 0$. We obtain

$$\boldsymbol{y}(t) = \left(\boldsymbol{y}_r - \frac{t_r(\frac{2}{5}(t_r/t_{end})^2 - \frac{14}{15}t_r/t_{end} + \frac{1}{2})}{(t_r/t_{end} - 1)^2}\dot{\boldsymbol{y}}_r\right)$$

$$+\frac{1-\frac{4}{3}t_r/t_{end}}{t_r^2(t_r/t_{end}-1)^2}\dot{\boldsymbol{y}}_r t^3 + \frac{2(t_r/t_{end})^2-1}{2t_r^3(t_r/t_{end}-1)^2}\dot{\boldsymbol{y}}_r t^4 + \frac{2-3t_r/t_{end}}{5t_{end}t_r^3(t_r/t_{end}-1)^2}\dot{\boldsymbol{y}}_r t^5. \quad \text{(B.6)}$$

Note that t_r (release time) and t_{end} (the time when the robot stops moving) still have not been determined. We compute them by requiring that at the beginning of motion, the ball holder is in a horizontal position, making sure that the ball does not fall from the holder. This is the case in our experimental setup if the sum of the three joint angles active in throwing, i.e. $\sum_{i=1}^{3} y_i$, is equal to 135 degrees. We formulate the following optimization problem

$$\arg\min_{t_r,\, t_{end}} \left\{ \left(3\pi/4 - \textstyle\sum_{i=1}^{3} y_i(0)\right)^2 \right\} \quad \text{subject to} \quad 0.3 \le t_{end},\ 0 < t_r < t_{end},\ \boldsymbol{y}^{\min} \le \boldsymbol{y}(0) \le \boldsymbol{y}^{\max}, \quad \text{(B.7)}$$

where

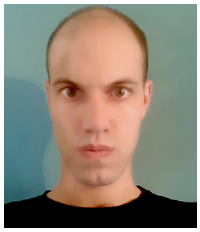
$$\boldsymbol{y}(0) = \boldsymbol{y}_r - \dot{\boldsymbol{y}}_r \frac{t_r(\frac{2}{5}(t_r/t_{end})^2 - \frac{14}{15}t_r/t_{end} + \frac{1}{2})}{(t_r/t_{end}-1)^2}. \quad \text{(B.8)}$$

The ball throwing trajectory is fully determined by solving (B.7).

References

[1] J. Peters, J. Kober, K. Muelling, O. Kroemer, G. Neumann, Towards robot skill learning: From simple skills to table tennis, in: European Conference on Machine Learning (ECML), 2013, pp. 627–631.
[2] T. Petrič, A. Gams, L. Colasanto, A.J. Ijspeert, A. Ude, Accelerated sensorimotor learning of compliant movement primitives, IEEE Trans. Robot. 34 (6) (2018) 1636–1642.
[3] S. Thrun, T.M. Mitchell, Lifelong robot learning, Robot. Auton. Syst. 15 (1) (1995) 25–46.
[4] S. Schaal, Is imitation learning the route to humanoid robots? Trends Cogn. Sci. 3 (6) (1999) 233–242.
[5] R. Dillmann, Teaching and learning of robot tasks via observation of human performance, Robot. Auton. Syst. 47 (2–3) (2004) 109–116.
[6] B. Nemec, R. Vuga, A. Ude, Efficient sensorimotor learning from multiple demonstrations, Adv. Robot. 27 (13) (2013) 1023–1031.
[7] A. Alissandrakis, C.L. Nehaniv, K. Dautenhahn, Solving the correspondence problem between dissimilarly embodied robotic arms using the ALICE imitation mechanism, in: Second International Symposium on Imitation in Animals & Artifacts (AISB), 2003, pp. 79–92.
[8] A. Ude, A. Gams, T. Asfour, J. Morimoto, Task-specific generalization of discrete and periodic dynamic movement primitives, IEEE Trans. Robot. 26 (5) (2010) 800–815.
[9] J. Kober, D. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.
[10] J. Kober, J. Peters, Policy search for motor primitives in robotics, Mach. Learn. 84 (1–2) (2011) 171–203.
[11] M.P. Deisenroth, G. Neumann, J. Peters, A survey on policy search for robotics, Found. Trends Robot. 2 (1–2) (2013) 388–403.
[12] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, S. Schaal, Dynamical movement primitives: Learning attractor models for motor behaviors, Neural Comput. 25 (2) (2013) 328–373.
[13] I. Jolliffe, J. Cadima, Principal component analysis: A review and recent developments, Phil. Trans. R. Soc. A 374 (2016) 20150202.
[14] S. Vijayakumar, S. Schaal, Locally weighted projection regression: Incremental real time learning in high dimensional space, in: Seventeenth International Conference on Machine Learning (ICML), Morgan Kaufmann, San Francisco, CA, 2000, pp. 1079–1086.
[15] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[16] R. Pahič, A. Gams, A. Ude, J. Morimoto, Deep encoder-decoder networks for mapping raw images to dynamic movement primitives, in: IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 2018, pp. 5863–5868.
[17] K.S. Luck, G. Neumann, E. Berger, J. Peters, H.B. Amor, Latent space policy search for robotics, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Chicago, IL, 2014, pp. 1434–1440.
[18] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, J. Peters, Stable reinforcement learning with autoencoders for tactile and visual data, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Deajeon, Korea, 2016, pp. 3928–3934.
[19] N. Chen, J. Bayer, S. Urban, P. van der Smagt, Efficient movement representation by embedding Dynamic Movement Primitives in deep autoencoders, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Seoul, Korea, 2015, pp. 434–440.
[20] N. Chen, M. Karl, P. van der Smagt, Dynamic movement primitives in latent space of time-dependent variational autoencoders, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Cancun, Mexico, 2016, pp. 629–636.
[21] D.M. Wolpert, J. Diedrichsen, J.R. Flanagan, Principles of sensorimotor learning, Nat. Rev. Neurosci. 12 (12) (2011) 739–751.
[22] B. Nemec, D. Forte, R. Vuga, M. Tamošiūnaitė, F. Wörgötter, A. Ude, Applying statistical generalization to determine search direction for reinforcement learning of movement primitives, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Osaka, Japan, 2012.
[23] A. Colomé, C. Torras, Dimensionality reduction and motion coordination in learning trajectories with Dynamic Movement Primitives, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, 2014, pp. 1414–1420.
[24] D. Forte, A. Gams, J. Morimoto, A. Ude, On-line motion synthesis and adaptation using a trajectory database, Robot. Auton. Syst. 60 (10) (2012) 1327–1339.
[25] T. Matsubara, S.-H. Hyon, J. Morimoto, Learning parametric dynamic movement primitives from multiple demonstrations, Neural Netw. 24 (5) (2011) 493–500.
[26] R. Pahič, B. Ridge, A. Gams, J. Morimoto, A. Ude, Training of deep neural networks for the generation of dynamic movement primitives, Neural Netw. 127 (2020) 121–131.
[27] M. Deniša, A. Gams, A. Ude, T. Petrič, Learning compliant movement primitives through demonstration and statistical generalization, IEEE/ASME Trans. Mechatronics 21 (5) (2016) 2581–2594.
[28] A. Kramberger, A. Gams, B. Nemec, D. Chrysostomou, O. Madsen, A. Ude, Generalization of orientation trajectories and force-torque profiles for robotic assembly, Robot. Auton. Syst. 98 (2017) 333–346.
[29] T. Gašpar, B. Nemec, J. Morimoto, A. Ude, Skill learning and action recognition by arc-length dynamic movement primitives, Robot. Auton. Syst. 100 (2018) 225–235.
[30] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, O. Sigaud, Learning compact parameterized skills with a single regression, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Atlanta, GA, 2013, pp. 417–422.
[31] R. Reinhart, J. Steil, Efficient policy search with a parameterized skill memory, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, 2014, pp. 1400–1407.
[32] K. Muelling, J. Kober, O. Kroemer, J. Peters, Learning to select and generalize striking movements in robot table tennis, Int. J. Robot. Res. 32 (3) (2013) 263–279.
[33] Y. Zhou, J. Gao, T. Asfour, Movement primitive learning and generalization: Using mixture density networks, IEEE Robot. Autom. Mag. 27 (2) (2020) 2–12.
[34] S. Calinon, A tutorial on task-parameterized movement learning and retrieval, Intell. Serv. Robot. 9 (1) (2015) 1–29.
[35] Y. Zuo, G. Avraham, T. Drummond, Traversing latent space using decision ferns, 2018, arXiv:1812.02636.
[36] L. Le, A. Patterson, M. White, Supervised autoencoders: Improving generalization performance with unsupervised regularizers, in: Advances in Neural Information Processing Systems 31, Curran Associates, 2018, pp. 107–117.
[37] Y. Yoo, S. Yun, H.J. Chang, Y. Demiris, J.Y. Choi, Variational autoencoded regression: High dimensional regression of visual data on complex manifold, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, Hawaii, 2017, pp. 2943–2952.
[38] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[39] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning), The MIT Press, 2005.
[40] E. Todorov, T. Erez, Y. Tassa, MuJoCo: A physics engine for model-based control, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal, 2012, pp. 5026–5033.
[41] B. Nemec, R. Vuga, A. Ude, Exploiting previous experience to constrain robot sensorimotor learning, in: IEEE-RAS International Conference on Humanoid Robots (Humanoids), Bled, Slovenia, 2011, pp. 727–732.
[42] R. Pahič, Throwing trajectories dataset, 2019, https://github.com/abr-ijs/Throwing-trajectories-dataset.
[43] M. Ranzato, Y.-L. Boureau, S. Chopra, Y. LeCun, A unified energy-based framework for unsupervised learning, in: M. Meila, X. Shen (Eds.), Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, San Juan, Puerto Rico, 2007, pp. 371–379.
[44] H. Kamyshanska, R. Memisevic, The potential energy of an autoencoder, IEEE Trans. Pattern Anal. Mach. Intell. 37 (6) (2015) 1261–1273.

Rok Pahič received the M.Sc. degree in mechanical engineering from the University of Maribor, Slovenia. As of 2016 he started his Ph.D. study at Jožef Stefan International Postgraduate School and works at the Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia. His research work focuses mainly on deep learning in robotics.



Zvezdan Lončarević received B.Sc. degree in Mechatronics from the Faculty of Technical Science, University of Novi Sad, Serbia in 2017 and M.Sc degree in Information and Communication Technologies from Jožef Stefan International Postgraduate School, Ljubljana, Slovenia in 2019. As of 2019 he started his Ph.D. study at Jožef Stefan International Postgraduate School and works at the Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia. His research work focuses mainly on generation of robotic trajectories through optimization and reinforcement learning.

Andrej Gams is currently a senior research associate at Dept. of Automatics, Biocybernetics, and Robotics at Jožef Stefan Institute, and an assistant professor at the Jožef Stefan International Postgraduate School. He received his diploma degree and his Ph.D. in robotics from the University of Ljubljana, Slovenia. He was a postdoctoral researcher with SCIEX NMS-CH fellowship at the Biorobotics Laboratory, EPFL, Switzerland, and a visiting researcher at the ATR Computational Neuroscience Laboratories in Japan. His research interests are at the intersection of humanoid and applicative industrial robotics, specifically learning by imitation, learning for manipulation, compliance and adaptation of robotic motion.

Aleš Ude received the Diploma degree in applied mathematics from the University of Ljubljana, Slovenia, in 1990, and the Ph.D. degree from the Faculty of Informatics, University of Karlsruhe, Germany, in 1995. Currently he is a research councillor at Jožef Stefan Institute in Ljubljana, Slovenia, where he heads the Department of Automatics, Biocybernetics, and Robotics. He is also a visiting researcher at ATR Computational Neuroscience Laboratories, Kyoto, Japan. His research interests include robot learning, humanoid robotics and reconfigurable robot systems.