Training of deep neural networks for the generation of dynamic movement primitives

Rok Pahič [a,c,*], Barry Ridge [a,b], Andrej Gams [a], Jun Morimoto [b], Aleš Ude [a,b,d]

[a] Humanoid and Cognitive Robotics Laboratory, Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
[b] ATR Computational Neuroscience Laboratories, 2-2-2 Hikaridai Seika-cho, Sorakugun, Kyoto 619-0288, Japan
[c] Jozef Stefan International Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia
[d] Faculty of Electrical Engineering, University of Ljubljana, Tržaška cesta 25, 1000 Ljubljana, Slovenia

ARTICLE INFO

Keywords:
Training of deep neural networks
Dynamic movement primitives
Robot skill learning

ABSTRACT

Dynamic movement primitives (DMPs) have proven to be an effective movement representation for motor skill learning. In this paper, we propose a new approach for training deep neural networks to synthesize dynamic movement primitives. The distinguishing property of our approach is that it can utilize a novel loss function that measures the physical distance between movement trajectories as opposed to measuring the distance between the parameters of DMPs that have no physical meaning. This was made possible by deriving differential equations that can be applied to compute the gradients of the proposed loss function, thus enabling an effective application of backpropagation to optimize the parameters of the underlying deep neural network. While the developed approach is applicable to any neural network architecture, it was evaluated on two different architectures based on encoder–decoder networks and convolutional neural networks. Our results show that the minimization of the proposed loss function leads to better results than when more conventional loss functions are used.

1. Introduction

The generic idea of encoding elementary movements using dynamical systems has become accepted in the motor skill learning community (Kober & Peters, 2010). This led to the development of effective movement representations such as *dynamic movement primitives* (DMPs) (Schaal, Mohajerian, & Ijspeert, 2007). DMPs can be viewed as nonlinear dynamical systems that represent elementary movements. They can be used as building blocks that can be sequenced and modulated in real time to generate more complex movements (Ijspeert, Nakanishi, Hoffmann, Pastor, & Schaal, 2013). Thus it is natural to apply one of the prime machine learning technologies, i.e. deep neural networks, for the learning of dynamic systems such as DMPs.

Deep neural networks have been applied successfully in many different application areas (LeCun, Bengio, & Hinton, 2015), e.g. for visual classification and natural language processing. Due to deep learning, visual object recognition has already reached the stage where computers are superior to humans at some specific tasks. Several factors contributed to these breakthroughs. They include the increased computational power and the possibility to utilize GPUs to speed up the training of deep neural network models. Another contributing factor was the development of appropriate deep neural network architectures, such as deep convolutional neural networks (CNNs) (Krizhevsky, Sutskever, & Hinton, 2012). Such network architectures contain fewer but properly organized parameters that can be trained more easily while retaining the expressive power for visual recognition tasks.

The universal approximation theorem (Hornik, Stinchcombe, & White, 1989) indicates that neural networks can provide the functionality needed to learn highly nonlinear mappings that link robot sensory data to DMPs. However, training of deep neural networks for robotic applications often leads to issues that have not yet been fully addressed by the machine learning community (Pierson & Gashler, 2017; Sünderhauf et al., 2018). There is need for better evaluation metrics (distance functions) which – when combined with appropriate representations – result in a more effective training process.

In this paper we focus on the problem of training deep neural networks that have DMP parameters as outputs. The main result of the paper is a general, physically meaningful evaluation metric for DMPs and a mathematical machinery that exploits the proposed metric to guide the training process. The proposed metric

* Corresponding author at: Humanoid and Cognitive Robotics Laboratory, Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia.
E-mail addresses: rok.pahic@ijs.si (R. Pahič), barry.ridge@ijs.si (B. Ridge), andrej.gams@ijs.si (A. Gams), xmorimo@atr.jp (J. Morimoto), ales.ude@ijs.si (A. Ude).

computes the physical distance between movements instead of calculating the distance between the parameters of DMPs, which have no physical meaning. We derive formulas for the calculation of gradients of such a metric with respect to DMP parameters. By making use of these gradients, we can apply state-of-the art optimization methods based on backpropagation to train the parameters of the desired neural network.

Other approaches that utilize deep neural networks to learn DMPs have been reported in the literature. For example, Chen et al. use autoencoders (Chen, Bayer, Urban, & van der Smagt, 2015) and variational autoencoders (Chen, Karl, & van der Smagt, 2016) to reduce the dimensionality of the movements obtained by human demonstration and effectively train DMPs in a low-dimensional latent space. Pervez, Mao, and Lee (2017) use a pre-trained CNN for finding task parameters from input images, while using another fully-connected neural network to learn the mapping from the clock signal and task parameters to the forcing term of a DMP. Similarly, in the work of Kim, Lee, and Kim (2018), hierarchical deep reinforcement learning is used to optimize the forcing term of a DMP for demonstrated trajectories. None of these approaches address the issue of a proper metric for networks that directly compute DMPs.

For evaluation purposes we applied the proposed approach to the problem of mapping raw images of digits to the corresponding robot handwriting trajectories, which are represented by DMPs. We developed two neural network architectures for this task by borrowing from the work on deep autoencoders, which were shown to be effective at computing low dimensional latent spaces from raw character images (Hinton & Salakhutdinov, 2006) and SegNet network (Badrinarayanan, Kendall, & Cipolla, 2017), in which pre-trained layers from a CNN were adapted to form a fully-convolutional encoder–decoder network for semantic pixel-wise segmentation. We extended our previously developed encoder–decoder network architecture (Pahič, Gams, Ude, & Morimoto, 2018) with convolutional layers to achieve more robust processing of input images. Here we stress, however, that the proposed evaluation metric and gradient calculation are applicable to any neural network architecture including recurrent neural networks and are not limited to the architectures tested in this paper.

The rest of the paper is organized as follows. We start by specifying the data needed for training neural networks that have DMPs as output. A review of the DMP representation for robot motion trajectories is provided in Appendix A. The core of the paper is in Section 3, where we propose two loss functions to train such networks and discuss how to compute their gradients to enable backpropagation. A detailed mathematical derivation of the gradient of the newly proposed loss function is provided in Appendix B. Neural network architectures applied to learn transformations that transform images of digits into DMPs are described in Section 4. The paper concludes with the experimental results and the discussion sections.

2. Datasets for training deep image-to-motion encoder–decoder networks

In order to facilitate the understanding of what follows, we first briefly describe the task used to evaluate the proposed methodology. Note, however, that the proposed methodology is applicable to any neural network architecture that has DMPs as outputs and can be trained by backpropagation.

Our experimental problem was to learn a deep neural network that transforms raw images of digits into robot writing trajectories. This is a highly nonlinear transformation that requires a lot of data if we are to capture variations in different handwriting styles. The training data are given as pairs of images and the associated writing trajectories. After training, the robot observes a digit, captures its image, and feeds the captured image to the trained neural network, which outputs the corresponding robot writing trajectory. We selected dynamic movement primitives (DMPs) to represent handwriting trajectories. See Appendix A for more details on DMPs. The robot finally replays the resulting DMP and writes the digit in the same style as in the observed image.

In this experimental setting, the input and output data pairs have the following structure:

$$\mathbf{D} = \left\{\mathbf{C}_j, \mathbf{M}_j\right\}_{j=1}^{P}, \tag{1}$$

where P is the number of training pairs, $\mathbf{C}_j \in \mathbb{R}^{H\times W}$ are the input images of width W and height H, while $\mathbf{M}_j$ are the corresponding writing movements associated with each image. Note that apart from the application of convolutional neural networks in our experiments, all other derivations in this paper are independent of the type of input data. Hence our approach is applicable to other sensory inputs. A time-dependent trajectory data can be represented as a temporal sequence of configurations on the trajectory

$$\mathbf{M}_j = \left\{\boldsymbol{y}_{i,j}, t_{i,j}\right\}_{i=1}^{T_j}. \tag{2}$$

Here $\boldsymbol{y}_{i,j} \in \mathbb{R}^d$ are the robot configurations, e.g. Cartesian positions or joint angles, on the jth trajectory in the ith sample at time $t_{i,j} \in \mathbb{R}$ and d is the number of degrees of freedom. In our handwriting experiments, trajectories were specified as 2-D planar movements ($d = 2$) carried out at a fixed orientation.

The number of data points in dataset (2) is not constant and varies with respect to the selected sampling step and overall time/length of the movement trajectory. Thus it is not possible for a neural network to directly output the sequence of data points on the trajectory. Instead, the proposed neural networks output the parameters of DMPs, where the resulting output movements are encoded by a constant number of parameters. In the case of DMPs, the output data of the neural network is given by

$$\mathbf{M}_j^{DMP} = \left\{\{\boldsymbol{w}_k\}_{k=1}^{N},\ \tau,\ \boldsymbol{g},\ \boldsymbol{y}_0\right\}. \tag{3}$$

See Appendix A to understand the meaning of these parameters.

The process of training deep neural networks for mapping images to handwriting DMPs is presented graphically in Fig. 1. Technical details about the generation of the training data are explained in Section 5.1.

3. Loss functions and calculation of their gradients

The training of neural networks is usually realized by estimating the parameters of a given network that minimize a pre-specified loss function. For supervised learning, the loss function usually measures the difference between the desired and actual outputs. Backpropagation (Rumelhart, Hinton, & Williams, 1986) is the method of choice to implement such an optimization process. Backpropagation requires the gradients of all functions that constitute a neural network, including the gradients of the loss function.

The most common loss function that can be used for training a neural network with DMP parameters on the output is the mean squared error of DMP parameters, which – for the jth training datapoint – is defined as follows:

$$\mathrm{E}_p(j) = \frac{1}{2}\left(\sum_{k=1}^{N}\|\boldsymbol{w}_k - \boldsymbol{w}_{k,j}\|^2 + (\tau - \tau_j)^2 + \|\boldsymbol{g} - \boldsymbol{g}_j\|^2 + \|\boldsymbol{y}_0 - \boldsymbol{y}_{0,j}\|^2\right). \tag{4}$$

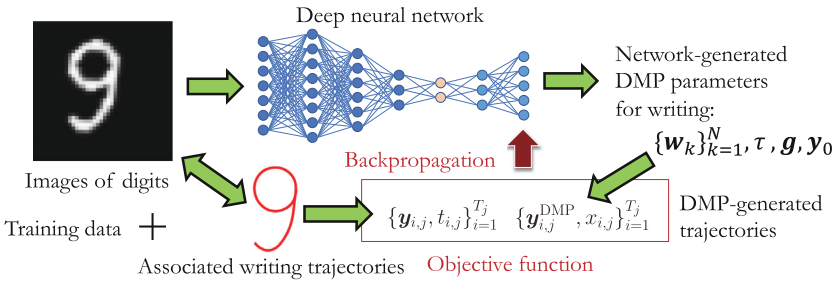

Fig. 1. Training of writing DMPs. Each input image is fed to the neural network that transforms the input image into DMP parameters (3). The output DMP parameters are used to generate a temporal sequence of points on the DMP trajectory, which are compared to the data points on the training trajectory associated with the input image. The loss function and its gradients are then computed to optimize the parameters of the deep neural network by backpropagation.

Here $\{\{\boldsymbol{w}_k\}_{k=1}^N, \tau, \boldsymbol{g}, \boldsymbol{y}_0\}$ denotes the output of the neural network and $\{\{\boldsymbol{w}_{k,j}\}_{k=1}^N, \tau_j, \boldsymbol{g}_j, \boldsymbol{y}_{0,j}\}$ the DMP parameters estimated from the training data (3) at index j. See Appendix A for more details about DMPs and their calculation.

While this loss function provides for an easy implementation, it does not measure the physical distance between the training movement and the movement calculated by the neural network, but rather the distance between the DMP parameters. This distance has no physical meaning. A more natural loss function would measure the distance between the training trajectory data $\boldsymbol{y}_{i,j}$ from Eq. (2) and the points on the trajectory generated by the output DMP, here denoted as $\boldsymbol{y}^{\mathrm{DMP}}$. Hence we define the following loss function

$$\mathrm{E}_t(j) = \frac{1}{2T_j} \sum_{i=1}^{T_j} \|\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j}\|^2, \tag{5}$$

where $\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j})$ and $x_{i,j} = x(t_{i,j})$ are obtained by integrating the jth output DMP as calculated by the neural network and T_j is the number of points on the jth trajectory.

To apply backpropagation, we need to be able to compute the gradients of the loss function. The gradients of loss function (4) are trivial to compute as (4) is simply the Euclidean distance between the training DMP parameters and the DMP parameters computed by the neural network. But it is more difficult to compute the gradients of (5) as $\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j})$ are calculated by integrating DMP equations (7)–(9).

Let p_h, $h = 1, \ldots, H$, be the DMP parameters specified in Eq. (12), which are computed as output of the neural network. Then the partial derivatives of $\mathrm{E}_t(j)$ with respect to each DMP parameter can be calculated as follows

$$\frac{\partial \mathrm{E}_t(j)}{\partial p_h} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j}\right)^{\mathrm{T}} \frac{\partial \boldsymbol{y}^{\mathrm{DMP}}}{\partial p_h}(x_{i,j}), \tag{6}$$

where $\partial \boldsymbol{y}^{\mathrm{DMP}} / \partial p_h$ is the partial derivative of $\boldsymbol{y}^{\mathrm{DMP}}$ with respect to the DMP parameter p_h. The difficulty lies in the computation of $\partial \boldsymbol{y}^{\mathrm{DMP}} / \partial p_h$ because $\boldsymbol{y}^{\mathrm{DMP}}$ is computed by integrating differential equation system (7)–(9). It turns out, however, that each of the partial derivatives $\partial \boldsymbol{y}^{\mathrm{DMP}} / \partial p_h$ can also be computed by integrating a system of differential equations with proper boundary conditions. These differential equation systems are similar to the initial DMP equations.

A detailed mathematical derivation of partial derivatives with respect to each DMP parameter is provided in Appendix B.

4. Neural network architectures for reproduction of writing dmps

To evaluate the effectiveness of the proposed loss functions, we considered deep neural networks that transform raw images of digits into the corresponding robot writing trajectories. In such a setting, the pixels of an image containing a digit are used as input to a deep neural network, which is trained to compute the corresponding DMP parameters of writing trajectories that reproduce the observed digit.

We developed two different types of encoder–decoder network architectures to implement this transformation process:

- The first is a fully-connected image-to-motion encoder–decoder network architecture (here abbreviated as IMEDNet) shown in Fig. 2. This neural network architecture was inspired by the work of (Hinton & Salakhutdinov, 2006).
- The second is a CNN-based architecture, i.e. a convolutional image-to-motion encoder–decoder network (CIMEDNet) shown in Fig. 3. CIMEDNet uses convolutional layers followed by some additional fully-connected layers in the encoder, whereas the decoder is again constructed of fully-connected layers only. The resulting encoder part of the network has a somewhat similar structure to the LeNet-5 architecture (LeCun, Bottou, Bengio, & Haffner, 1998). The fully connected encoder–decoder part is taken over from IMEDNet with the first three layers removed.

The purpose of creating CIMEDNet architecture was to reduce the number of network parameters in order to achieve faster training and better generalization. However, if we simplify the model too much, we lose the representational power of the neural network, which can lead to a reduced performance. For example, when the encoder part consisted of convolutional layers only, the network performance was lower compared to IMEDNet in our experiments. The best results were obtained by the here proposed architecture, which still contains significantly less parameters than IMEDNet while preserving sufficient representational power to map images to handwriting DMPs.

To generate the inputs for both networks, the observed images of digits were cropped and resized to a fixed-size input (see also Section 5.3.2). By applying standard image processing methods we ensured that only images of digits without any background outside of the paper on which they were written were passed to the networks as input.

5. Experimental evaluation

Our experiments focused on the evaluation of performance of different loss functions for the generation of writing trajectories. The suitability of CIMEDNet architecture for processing of real digit images taken by a humanoid robot was also evaluated.

5.1. Datasets

Here we explain the acquisition of datasets that were used in our experiments for training and testing.

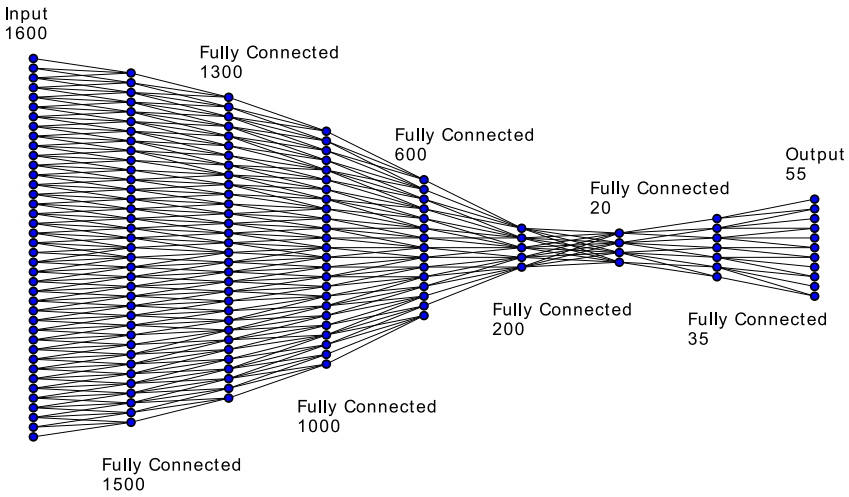

Fig. 2. IMEDNet (Image-to-Motion Encoder–Decoder Network) neural network with an input layer consisting of 1600 neurons (for an image size of 40 × 40 pixels) for the reproduction of 2-D writing trajectories from raw images. The output layer contains 55 neurons that correspond to the parameters of a two-dimensional DMP (2 × 25 neurons for the weights of the two forcing terms defined in Eq. (10), 2 neurons each to specify the beginning $\boldsymbol{y}_0$ and the end $\boldsymbol{g}$ of the trajectory, respectively, and 1 neuron for the joint time constant τ). There are 7 hidden fully-connected layers with 1500, 1300, 1000, 600, 200, 20, and 35 neurons, respectively. The number of all trainable parameters in the network is 6 381 335. This configuration of neurons forms an asymmetrical encoder–decoder architecture (note that a ratio between the number of actual neurons and the number of drawn neurons is not the same in all layers). The bottleneck layer consists of 20 neurons, which can be used to define the latent space for writing trajectories.

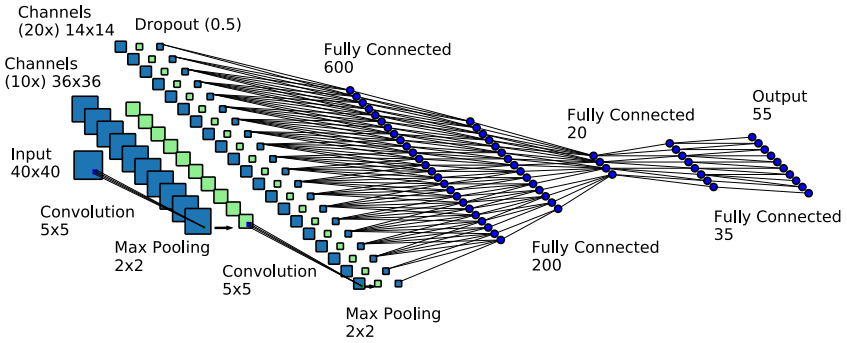

Fig. 3. The CIMEDNet (Convolutional Image-to-Motion Encoder–Decoder Network) architecture with an encoder consisting of two convolutional layers followed by three fully-connected layers, and a decoder consisting of two fully-connected layers. The network takes as input a 40 × 40 grayscale image, followed by a convolutional layer with 5 × 5 kernel size and 10 feature maps, a 2 × 2 max pooling layer, a convolutional layer with 5 × 5 kernel size and 20 feature maps, a 2 × 2 max pooling layer, a 0.5 dropout layer, a fully-connected layers of size 600, 200, 20, 35 and the output layer of size 55, matching the number of DMP parameters. The number of all trainable parameters in this network is 720 955.

5.1.1. Annotated MNIST (a-MNIST)

To evaluate our methodology, we needed pairs of input and output data. Because there is no suitable dataset with pairs of images and corresponding motions, we first created our own dataset by annotating images with motions.

The first dataset was created based on the well-known MNIST dataset (LeCun, Cortes, & Burges, 2019), which consists of images of digits but contains no writing trajectories. Using touch interface, we annotated 1170 images of digit 3 and 1170 images of digit 5 from this dataset with the corresponding handwriting movements. For training, we generated 11 700 samples of digit 3 and 11 700 samples of digit 5 in total by applying affine transformations to the original images as well as to the manually added handwriting movements. Values for affine transformations were ±3 pixels for translations, ±8° for rotations, ±10% for scalings and ±0.1 for shear values. The resulting dataset is called annotated MNIST dataset (a-MNIST) and consists of the original and transformed images from the MNIST database, supplemented with the corresponding handwriting trajectories.

The extension of the dataset by applying affine transformations was necessary because otherwise the dataset would contain only 2340 annotated examples, which is too little for reliable training of deep neural networks that map images to DMPs. Training with such a small dataset would lead to a poor generalization performance.

5.1.2. Synthetic MNIST (s-MNIST)

The a-MNIST dataset was gathered by hand-annotating images of digits with handwriting trajectories. This is a time consuming process. In order to provide more data for a thorough and controlled evaluation, we developed a synthetic method to generate 40 × 40 images of digits and the associated two-dimensional writing trajectory movements.

The synthetic trajectory data were generated using a combination of straight lines and elliptic arcs. When generating these geometric elements, we varied the parameters such as lengths, angles, and minor and major axes and center of elliptic arcs. We varied these parameters according to a uniform distribution. From these trajectories, binary images were generated with the predefined width of the image curve. The resulting images were processed with a Gaussian filter. Finally, both the generated trajectories and the resulting images were transformed using affine transformations composed of translation, rotation, scaling, and shearing. Values for affine transformations were the same as for

a-MNIST and were taken from a uniform distribution. Using the above procedure, we generated pairs of synthetic digit images and the associated handwriting trajectories. We call this dataset synthetic MNIST (s-MNIST), as it simulates the real MNIST dataset but does not contain real images from MNIST. For our experiments we generated 2000 pairs of images and trajectories for each digit, for a total of 20 000 samples.

5.1.3. Suitability of data for neural network training

Our data generation procedures make sure that there is a unique mapping between digit images and handwriting trajectories. To ensure this, the writing of every digit – either simulated or generated with a touch interface – was always started from the same starting point. We also did not vary the speed of handwriting movements. Thus the training data do not contain any digits that would be written in two different ways.

5.2. Neural network training

We used the PyTorch platform (Paszke et al., 2017) in order to implement the proposed networks. Training was performed on a 16 core workstation with two graphics cards (NIVIDIA GTX 1080Ti GPUs). The PyTorch training algorithms allow us to select various loss functions and parameters. For training the encoder–decoder networks, we used the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 0.0005. The batch size was set to 128 for weight updates. In order to avoid learning plateaus, the optimizer parameters were reset to the initial values every 500 epochs. We halted the training if the best validation loss was unchanged after 60 epochs. We retained the neural network parameters with the best validation result.

In all experiments, the DMP representation of handwriting trajectories was comprised of 25 radial-basis functions for every dimension of the two dimensional writing trajectory. The weights of these basis functions form together with the joint time constant (1 parameter) and the start and end points of a planar movement (2×2 parameters) the full set of 55 DMP parameters (12) that represent each handwriting motion.

When training with the loss function defined in Eq. (5), the trajectories do not need to be transformed into DMPs. However, to compute this loss function, we needed to conduct temporal integration of the DMPs computed by a neural network for each training example. In addition, the gradients of the loss function (5) must be computed by integrating the differential equations provided in Appendix B. Even though we implemented this integration on the GPU cores, the computational complexity is higher compared to training directly with DMP parameters.

For more efficient training, we pretrained a neural network using criterion function (4) and then modified the loss function for additional training. When using loss function (4), the gradient calculation is simpler and can be performed using the built-in PyTorch functionalities. In this case, the gradients can be calculated without integrating the differential equations provided in Appendix B. However, the training trajectories must be transformed to DMPs to compute loss function (4).

The application of loss function (5) within the PyTorch framework requires the implementation of a new custom PyTorch *autograd* function, which should implement forward propagation and backpropagation for our custom-designed loss function. The resulting *autograd* function computes the DMP trajectories by integrating differential equation system (7)–(9) in the forward propagation, while for backpropagation the loss function gradients are computed by integrating differential equations described in Appendix B. We use Euler integration method in both cases, but more advanced Runge–Kutta methods could also be applied. We implemented two variants; firstly in Python for CPU utilization and secondly in C to exploit CUDA library for GPU utilization.

5.2.1. Number of training parameters and avoidance of overfitting

The number of parameters in our neural networks is rather high. IMEDNet contains 6 381 335 and CIMEDNet 720 955 free parameters, which must all be computed during training. The training data typically consist of 20 000 data points, of which 17 000 were used for optimization and the rest for testing. In our experiments, the dimension of the neural network output was 55 as there were 55 DMP parameters. Thus we obtained 17 000*55 = 935 000 equations. This number is higher than the number of parameters in CIMEDNet (720 955) but still less than the number of parameters in IMEDNet (6 381 335). Thus the CIMEDNet optimization problem is overconstrained, which is good to prevent overfitting.

To avoid overfitting, especially in the case of IMEDNet, we applied a standard technique that results in early stopping of the optimization process. This technique relies on a criterion that measures the performance of the neural network in each validation step. In the case of CIMEDNet, we also introduced a 0,5 dropout layer, which is located at the boundary between the convolutional and the fully connected part. This reduces the number of active parameters at each optimization step to only 426 955, which is much less than the number of equations that are considered during training (935 000). Our results show that we achieve a reasonable degree of generalization both with CIMEDNet and IMEDNet as they were able to replicate a-MNIST handwriting trajectories that were not in the training dataset. Thus the fact that the optimization problem that needs to be solved for IMEDNet is underconstrained did not drastically influence the performance of IMEDNet. However, Fig. 4 shows that CIMEDNet converged significantly faster.

5.3. Evaluation of loss functions

We used dynamic time warping (Sakoe & Chiba, 1978) to compare the distance between DMP trajectories, which were generated by the neural network, and the testing trajectories. Dynamic time warping computes the minimum spatial (image) distance between trajectories that are being compared. With dynamic time warping we estimate the spatial overlap between the two trajectories regardless of their parameterization, which is the most relevant measure when comparing writing trajectories. Note that we could not use dynamic time warping to implement loss functions (4) and (5) as dynamic time warping does not result in a differentiable loss function.

We conducted a two-sample t-test to compute if the difference between the mean values of the distance between trajectories (computed by dynamic time warping) for the two different loss functions is statistically significant. The reported t-value is the ratio of the observed difference and the size of the variability in the data. The higher the absolute t-value is, the lower the p-value, where the p-value denotes the probability that results have the same mean. In our experiments, p-value denotes the probability that the difference between the mean values of the distance between trajectories is not statistically significant.

5.3.1. Performance of the proposed loss functions

First we compare loss functions (4) and (5) that were both used for training of neural networks that output DMP parameters. We trained the four possible combinations of neural network architectures and loss functions on both s-MNIST and a-MNIST dataset.

In the case of s-MNIST dataset, we used 1700 training pairs of images and writing trajectories for each digit, which altogether makes 17 000 training pairs. For testing we used 300 pairs of images and writing trajectories per digit, altogether 3000 for the whole test dataset. For training with the a-MNIST dataset,

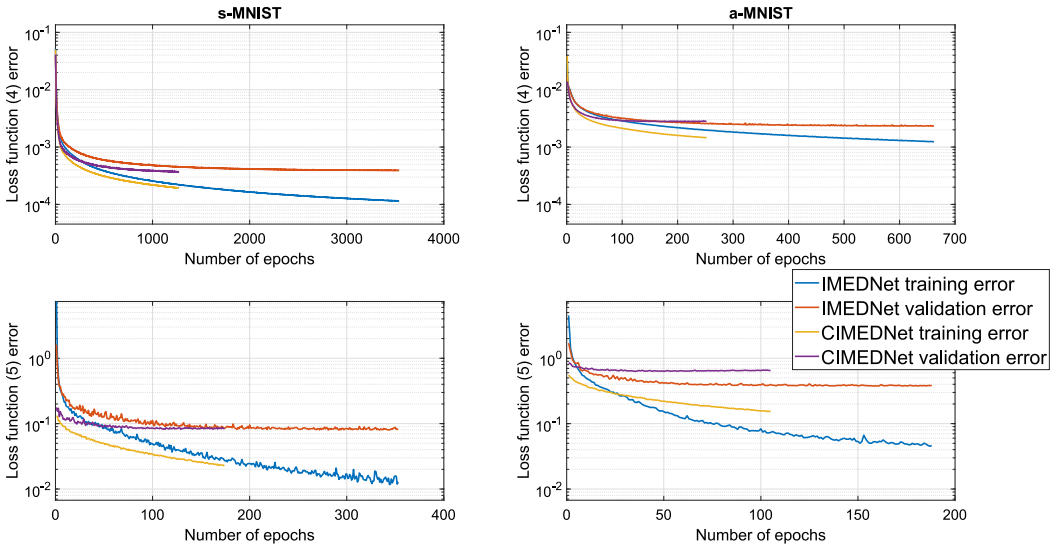

Fig. 4. Convergence of training and validation errors obtained while computing the parameters of our neural networks. We present results for IMEDNet and CIMEDNet, both applied to s-MNIST and a-MNIST dataset. In all cases the training was stopped when the validation error did not drop for 60 consecutive epochs.

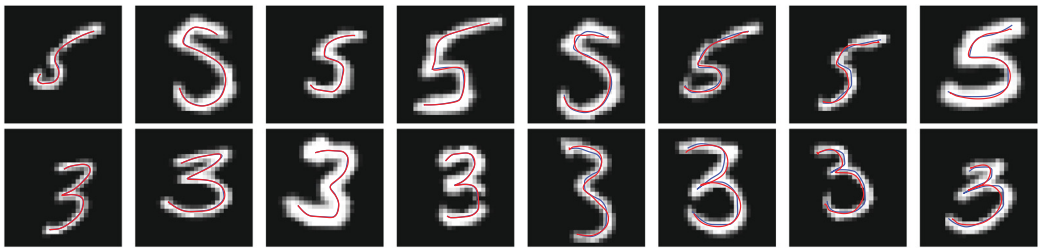

Fig. 5. Example reconstruction results for digits from the a-MNIST dataset. The manually generated and transformed trajectories are shown in blue, while the DMP trajectories calculated by the CIMEDNet are shown in red. These data were used only for testing, not for training. Hence these results demonstrate the generalization performance of the proposed encoder–decoder network. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

we used 8190 pairs of images and writing trajectories per digit, whereas for testing 3000 pairs were used.

The progress of the training process is shown in Fig. 4, which depicts the convergence of training, validation and test errors. The graphs show that IMEDNet needs at least twice as many epochs for both loss functions and for both datasets until convergence.

The statistics of reconstruction quality are shown in Table 1. The results obtained with both datasets and both network architectures confirmed that by using the physically meaningful loss function (5), we obtain significantly better results than with the simpler loss function (4). Results of the t-test for both datasets are shown in separate rows of the table.

Note that errors for the a-MNIST dataset are larger than for the s-MNIST dataset. This is the consequence of there being more complex shapes present inside the a-MNIST dataset. Even though the a-MNIST dataset contains just two different digits compared to the ten digits inside the s-MNIST dataset, the a-MNIST dataset is not only more complex in terms of trajectory shape but also contains digits with different line width compared to constant width lines in the s-MNIST dataset. Variation of the line width lowers the usefulness of each trained convolutional kernel on different examples, which led to better performance of IMEDNet compared to CIMEDNet for the a-MNIST dataset.

The analysis of results for the a-MNIST dataset presented in Fig. 5 shows that the CIMEDNet network can compute a good approximation of the handwriting motion, even if they are qualitatively not as good as the results for the s-MNIST dataset presented in Fig. 7.

Table 1
DMP reconstruction statistics for DMPs computed by IMEDNet and CIMEDNet trained by loss functions (4) and (5) using s-MNIST and a-MNIST dataset, respectively. Dynamic time warping was used to compute the pixel distance between the DMP-generated and test trajectories. The bottom row shows the results of t-test for both loss functions, which proves that the difference is statistically significant. These results were calculated using test samples that were not used for training.

	s-MNIST	a-MNIST
IMEDNetloss function (4)	0.215 ± 0.084	0.322 ± 0.137
IMEDNetloss function (5)	0.134 ± 0.045	0.231 ± 0.104
t-test comparison results	$t(5998) = 46.83$, $p < 0.001$	$t(5998) = 29.12$, $p < 0.001$
CIMEDNetloss function (4)	0.194 ± 0.076	0.389 ± 0.183
CIMEDNetloss function (5)	0.131 ± 0.063	0.319 ± 0.142
t-test comparison results	$t(5998) = 35.37$, $p < 0.001$	$t(5998) = 17.82$, $p < 0.001$

5.3.2. Experiment with a humanoid robot

We also evaluated the performance of CIMEDNet network on real input images that were taken by the TALOS humanoid robot (Stasse et al., 2017). The writing trajectories computed by the respective neural networks were used to generate the handwriting movements on TALOS. This experiment demonstrates end-to-end reproduction of writing trajectories from real images.

The parameters of our neural networks are computed off-line and do not need to be estimated during the real robot operation. The robot takes an image of a digit and feeds it to the pretrained

(a) (b) (c)

Fig. 6. Experimental setup for testing the CIMEDNet network architecture with real images. A sheet of paper with a handwritten digit is pasted onto a board in front of the robot (a). The red frame plotted around the sheet of paper indicates the part of the robot image that was extracted (b) and used as input to the neural network (c) . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

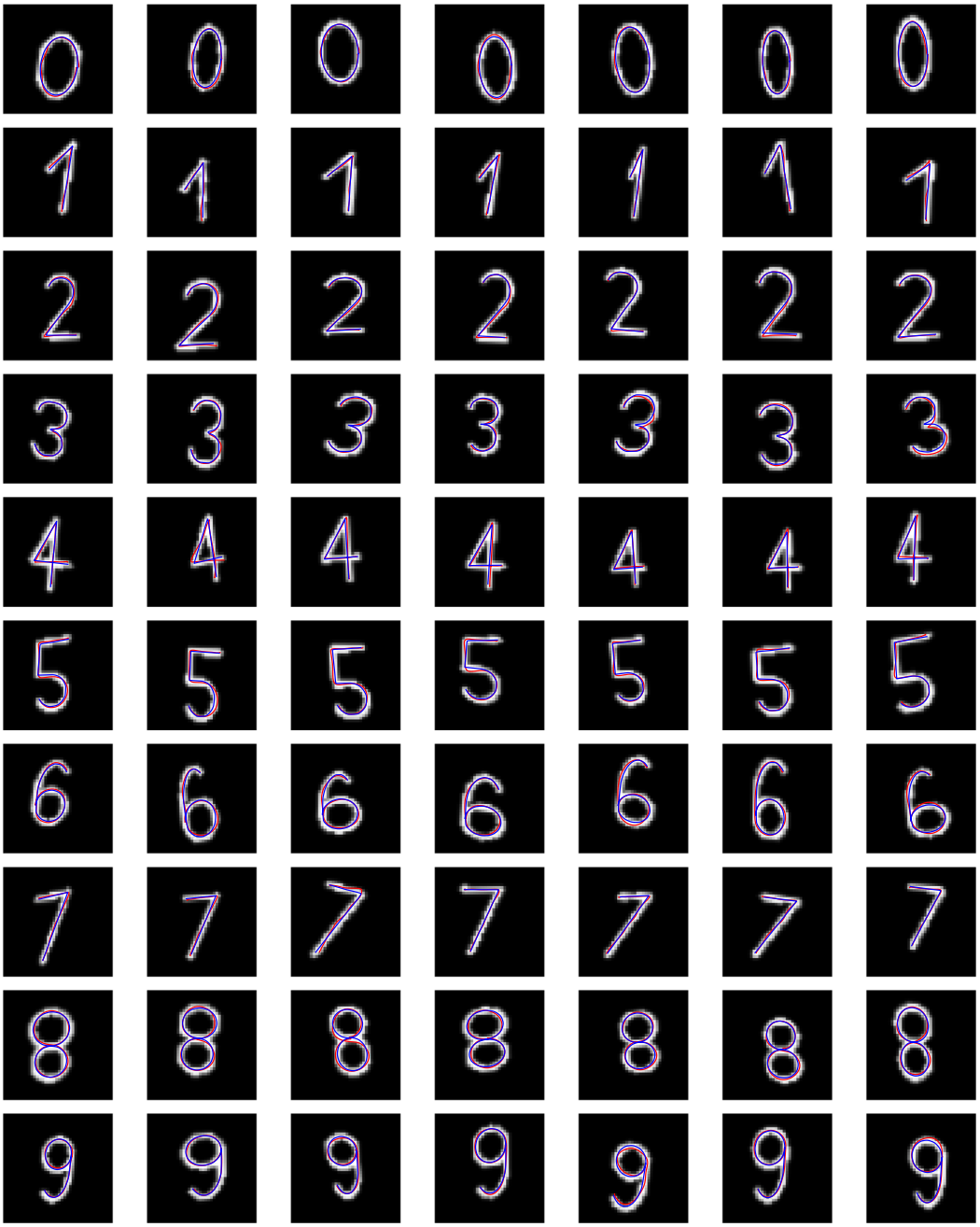

Fig. 7. Example synthetic data showing the images of digits and the corresponding handwriting trajectories. The original trajectories are shown in blue, while the DMP trajectories calculated by CIMEDNet are shown in red. CIMEDNet is able to reconstruct handwriting trajectories well even though these images and trajectories were not used for training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

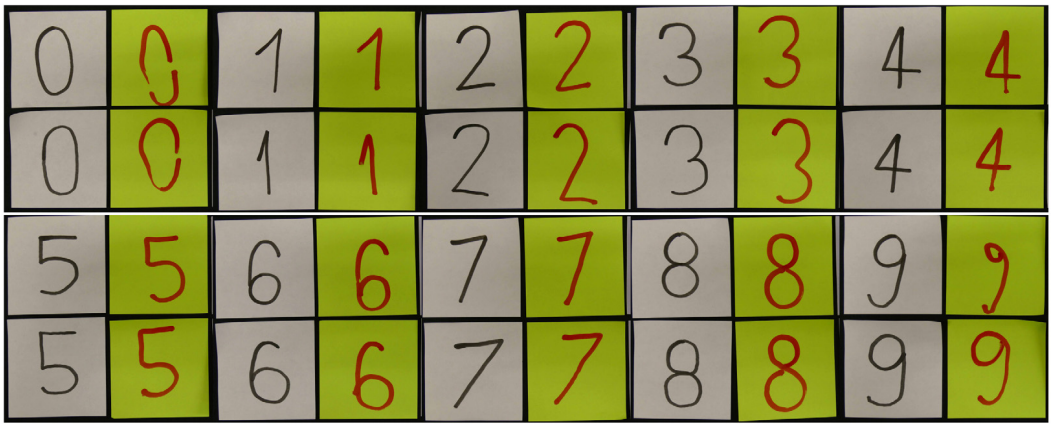

Fig. 8. Digits written by TALOS using DMPs that were output by the CIMEDNet neural network. The robotic writing of each digit was tested on two example handwritten digits. Digits in black on white paper were handwritten by a human and red digits on yellow paper were generated using the CIMEDNet neural network.

neural network, which computes the DMP parameters. The output DMPs are then used to generate the robot's handwriting movements. No gradient calculation is needed during real robot operation.

In the experiment, we pasted a piece of paper with a handwritten digit on a whiteboard in front of the robot, as shown in Fig. 6a. Standard computer vision algorithms were applied to detect the piece of paper in the images acquired by TALOS. The extracted subimage was resized to 40 × 40 pixels, equal to the image size for which the CIMEDNet was trained. Images like the one shown in Fig. 6b were used as input to the CIMEDNet neural network, which was trained to return DMP parameters of the associated writing trajectory. Loss function (5) and the PyTorch parameters as described in Section 5.3.1 were used for training. The handwriting trajectories generated by the DMPs, which were computed by the CIMEDNet, were specified in image coordinates (pixels). We transformed them into robot base coordinates, taking into account the actual paper size. The transformed trajectories were executed with a robot arm in the horizontal plane, with a constant height and orientation of the hand.

The robot-written digits generated in robotic experiments are shown in Fig. 8. Comparing the human-written digits with the digits written by Talos, we observe that the CIMEDNet is capable of good handwriting reproduction. The results are somewhat worse for digits with curves, e.g. when reproducing digit 0. The reason for this is that synthetic data was used for training, which is not fully representative for all human writing styles, especially for curved digits.

6. Discussion and future work

We proposed a new approach for training deep neural networks that compute DMP parameters on the output. Our results demonstrate that the performance of such neural networks can be significantly improved by specifying an appropriate loss function that measures real physical distance between the DMP-generated trajectories and the training trajectories as opposed to using simple Euclidean distance between the DMP parameters, which has no physical meaning. We derived the formulas for the computation of gradients of the newly proposed loss function, which is necessary for the application of backpropagation. The proposed methodology is applicable to any neural network architecture that can be trained by backpropagation, not just the ones tested in our experiments. Our experimental results show that deep neural networks can be effective at transforming sensory data to DMP parameters. More specifically, end-to-end conversion of digit images to DMPs has been achieved.

End-to-end learning of visuomotor policies has been addressed also by other researchers (Levine, Finn, Darrell, & Abbeel, 2016). It is by no means clear that it is always a good idea to directly map visual percepts to trajectories or even motor commands. For example, in some cases it might be beneficial to decompose the learning problem into smaller subproblems and organize learning in a hierarchical fashion. Another issue is to decide which neural network architecture is the most suitable for the given learning problem (fully connected networks, CNNs, RNNs, LSTMs, GANs, etc. and combinations thereof). Our paper does not address these larger learning issues. What it does address is that if one of the selected network architectures computes DMPs as output parameters to synthesize the desired movements, then its parameters can be estimated with the loss function and the backpropagation algorithm proposed in this paper. Our experiments show that training with the proposed loss function is superior to using standard loss functions that deep neural network libraries provide (which simply compare the DMP parameters from the training set to the DMP parameters computed by a neural network).

There are many possible extensions of our work. In the context of handwriting, other authors have shown that recurrent neural networks (RNNs) can be applied for the generation of handwritten digit images (Goudar & Buonomano, 2018). It is indeed possible to construct a recurrent neural network that outputs handwriting DMPs and train it with the proposed methodology. By applying RNNs, we could deal with issues such as different starting points when generating handwritten digits. Such approaches are currently under development in our lab. Another possible extension is to use a different representation to encode handwriting movements. For example, with arc-length DMPs (Gašpar, Nemec, Morimoto, & Ude, 2018) we can separate the temporal and spatial course of trajectories. This is beneficial if handwriting trajectories are performed at different speeds because in this case we can only reproduce the spatial course of movement from a single image as speed information is not available.

There are many tasks besides handwriting to which the proposed approach can be applied. Currently we are working on the extension to human–robot collaboration tasks. The idea is to observe human actions and use the resulting image sequence as input to a recurrent neural network that outputs the collaborative robot movement as DMP. This way we hope to achieve an effective human–robot collaboration. As explained above, the proposed methodology is general and can be used for any type of neural network that returns DMP parameters as output, including recurrent neural networks such as LSTM networks (Hochreiter &

Schmidhuber, 1997). The application of our approach to LSTM-based neural networks, which can take image sequences as input and compute DMP parameters, is an important topic of our current research.

One of the problems we encountered in our experiments with real data is that variations in simulated digit trajectories and images do not cover all variations arising in real images. Such issues can be addressed by mixing real and simulated data for training, but the generation of real training data is often expensive as it requires the gathering of human handwriting trajectories. It might therefore make sense to exploit either generative models, such as generative adversarial networks (GANs) (Goodfellow et al., 2014) or image style transfer (Gatys, Ecker, & Bethge, 2016) in order to first convert the input image into the style of the images from the original domain. This could be approached in the form of a two-step pipeline procedure, i.e. style conversion followed by prediction, or indeed, the style transfer properties of these approaches might be integrated into an entirely new network architecture that could be trained end-to-end.

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.

Acknowledgments

This work has received funding from the program group P2-0076 Automation, biocybernetics and robotics and the young researcher grant PR-07602, both funded by the Slovenian Research Agency, EU's Horizon 2020 grant QU4LITY (GA no. 825030); JSPS, Japan KAKENHI JP16H06565; JST, Japan-Mirai Program JPMJMI18B8; NEDO, Japan; the Commissioned Research of NICT; and the NICT Japan Trust (International research cooperation program).

Appendix A. Dynamic movement primitives

Let us denote a time-dependent movement trajectory as $\boldsymbol{y}(t) \in \mathbb{R}^d$, where d specifies the dimension of space that contains the desired trajectory (robot joint angle space or Cartesian space). A *dynamic movement primitive* (DMP) (Ijspeert et al., 2013) specifying such a movement is defined by the following system of differential equations

$$\tau \dot{\boldsymbol{z}} = \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - \boldsymbol{z}) + \mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}_0)\mathbf{F}(x), \tag{7}$$

$$\tau \dot{\boldsymbol{y}} = \boldsymbol{z}, \tag{8}$$

where $\boldsymbol{y}_0 \in \mathbb{R}^d$ is the initial position on the trajectory, $\boldsymbol{g} \in \mathbb{R}^d$ the final position on the trajectory, $\mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}_0) \in \mathbb{R}^{d\times d}$ a diagonal matrix with components of vector $\boldsymbol{g} - \boldsymbol{y}_0$ on the diagonal, $\mathbf{F}(x) \in \mathbb{R}^d$ a nonlinear forcing term, $\boldsymbol{z} \in \mathbb{R}^d$ a scaled velocity of motion, and $x \in \mathbb{R}$ the phase defined by the equation

$$\tau \dot{x} = -\alpha_x x. \tag{9}$$

The phase x is used instead of time to avoid explicit time dependency. It is fully defined by setting its initial value to $x(0) = 1$. If the parameters $\tau, \alpha_x, \alpha_z, \beta_z \in \mathbb{R}$ are defined appropriately, e.g. $\tau, \alpha_x > 0$ and $\alpha_z = 4\beta_z > 0$, then the linear part of equation system (7)–(8) becomes critically damped and $\boldsymbol{y}$, $\boldsymbol{z}$ monotonically converge to a unique attractor point at $\boldsymbol{y} = \boldsymbol{g}, \boldsymbol{z} = 0$. The forcing term $\mathbf{F}(x)$ is usually defined as a linear combination of radial basis functions

$$\mathbf{F}(x) = \frac{\sum_{k=1}^N \boldsymbol{w}_k \Psi_k(x)}{\sum_{k=1}^N \Psi_k(x)} x, \tag{10}$$

$$\Psi_k(x) = \exp\left(-h_k\,(x - c_k)^2\right), \tag{11}$$

where c_k are the centers of Gaussians distributed along the phase of the trajectory, and h_k their widths. The values of h_k are typically fixed, as are the values of c_k, which are usually distributed along the phase in a uniform manner. The role of $\mathbf{F}$ is to adapt the dynamics of (7)–(8) to the desired trajectory $\boldsymbol{y}(t)$, thus enabling the system to reproduce any smooth movement from the initial configuration $\boldsymbol{y}_0$ to the final configuration $\boldsymbol{g}$ on the trajectory. This can be accomplished by computing the free parameters $\boldsymbol{w}_k \in \mathbb{R}^d$ using regression techniques. See Ude, Gams, Asfour, and Morimoto (2010) for formulas to compute $\boldsymbol{w}_k$. The free parameter τ is usually set to the duration of motion.

α_z, β_z, and α_x are usually constants that do not change between movements. Thus the neural network needs to output the other parameters of the differential equation system (7)–(9) to fully specify a DMP:

$$\{\boldsymbol{w}_k\}_{k=1}^N,\ \tau,\ \boldsymbol{g},\ \boldsymbol{y}_0. \tag{12}$$

Appendix B. Derivation of the proposed loss function gradients

We first analyze the structure of partial derivatives $\partial \boldsymbol{y}^{\mathrm{DMP}}/\partial p_h$ that appear in Eq. (6). Recall that p_h is one of the DMP parameters. Analyzing differential equation system (7)–(9), it is clear that of all DMP parameters listed in Eq. (12), only τ affects multiple dimensions of $\boldsymbol{y}^{\mathrm{DMP}}$. All other DMP parameters affect only one dimension of $\boldsymbol{y}^{\mathrm{DMP}}$. Thus $\forall p_h \neq \tau$, the partial derivatives are different from zero only for the dimension of $\boldsymbol{y}^{\mathrm{DMP}}$ that is affected by this parameter. Thus Eq. (6) becomes

$$\frac{\partial \mathrm{E}_t(j)}{\partial p_h} = \frac{1}{T_j}\sum_{i=1}^{T_j}\left(y_l^{\mathrm{DMP}}(x_{i,j}) - y_{l,i,j}\right)\frac{\partial y_l^{\mathrm{DMP}}}{\partial p_h}(x_{i,j}),\ \forall p_h \neq \tau, \tag{13}$$

where l is the dimension of $\boldsymbol{y}^{\mathrm{DMP}}$ affected by the parameter p_h. In the following we derive the partial derivatives of y_l^{DMP} with respect to all DMP parameters.

We start with the forcing term parameters $w_{l,k}$, $k = 1, \ldots, N$, $l = 1, \ldots, d$. The partial derivatives $\partial y^{\mathrm{DMP}}/\partial w_{l,k}$ can be obtained by calculating the derivatives of Eqs. (7) and (8) with respect to $w_{l,k}$

$$\tau \frac{\partial \dot{z}_l}{\partial w_{l,k}} = \alpha_z(-\beta_z \frac{\partial y_l}{\partial w_{l,k}} - \frac{\partial z_l}{\partial w_{l,k}}) + (g_l - y_{l,0})\frac{\psi_k(x)}{\sum_{n=1}^N \Psi_n(x)}x, \tag{14}$$

$$\tau \frac{\partial \dot{y}_l}{\partial w_{l,k}} = \frac{\partial z_l}{\partial w_{l,k}}. \tag{15}$$

and integrating the resulting differential equation system in $\partial y_l/\partial w_{l,k}$ and $\partial z_l/\partial w_{l,k}$. Eqs. (14)–(15) can be derived because the following holds for continuously differentiable trajectories

$$\frac{d}{dt}\frac{\partial}{\partial w_{l,k}} z_l = \frac{\partial}{\partial w_{l,k}}\frac{d}{dt} z_l,$$

$$\frac{d}{dt}\frac{\partial}{\partial w_{l,k}} y_l = \frac{\partial}{\partial w_{l,k}}\frac{d}{dt} y_l.$$

Just like the DMP values $y_l^{\mathrm{DMP}}(x_{i,j})$, which we obtain through numerical integration, we can calculate the values $(\partial y_l/\partial w_{l,k})(x_{i,j})$ by integrating the differential equation system (14)–(15). For this purpose we need to know the initial values of $\partial y_l/\partial w_{l,k}$ and $\partial z_l/\partial w_{l,k}$ at $x(t_{1,j}) = x(0) = 1$. Since the initial position $y_l(1)$ on the trajectory does not depend on $w_{l,k}$, we can set

$$\frac{\partial y_l}{\partial w_{l,k}}(1) = \frac{\partial z_l}{\partial w_{l,k}}(1) = 0. \tag{16}$$

The partial derivatives with respect to the parameters g_l, $y_{0,l}$, $l = 1, \ldots, d$, are obtained analogously. Just like in the case of

partial derivatives with respect to $w_{l,k}$, the partial derivatives with respect to g_l and $y_{0,l}$ are obtained by calculating the partial derivatives of Eqs. (7) and (8) with respect to g_l and $y_{0,l}$. We obtain

$$\tau \frac{\partial \dot{z}_l}{\partial g_l} = \alpha_z \left(\beta_z \left(1 - \frac{\partial y_l}{\partial g_l} \right) - \frac{\partial z_l}{\partial g_l} \right) + \frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{17}$$

$$\tau \frac{\partial \dot{y}_l}{\partial g_l} = \frac{\partial z_l}{\partial g_l}. \tag{18}$$

and

$$\tau \frac{\partial \dot{z}_l}{\partial y_{0,l}} = \alpha_z \left(-\beta_z \frac{\partial y_l}{\partial y_{0,l}} - \frac{\partial z_l}{\partial y_{0,l}} \right) - \frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{19}$$

$$\tau \frac{\partial \dot{y}_l}{\partial y_{0,l}} = \frac{\partial z_l}{\partial y_{0,l}}. \tag{20}$$

The values of the above partial derivatives at phases $x_{i,j}$ can be calculated by respectively integrating the equation systems (17)–(18) and (19)–(20). The initial values are set as follows

$$\frac{\partial y_l}{\partial g_l}(1) = \frac{\partial z_l}{\partial g_l}(1) = 0, \tag{21}$$

$$\frac{\partial y_l}{\partial y_{0,l}}(1) = 1, \quad \frac{\partial z_l}{\partial y_{0,l}}(1) = 0. \tag{22}$$

In Eq. (22) we took into account that y_l is initially set to $y_{0,l}$.

The calculation of partial derivatives with respect to τ is somewhat more complicated because unlike previously considered parameters, τ affects all the degrees of freedom and also phase x through Eq. (9). Instead of the simplified Eq. (13), we need to apply the full Eq. (6), i.e.

$$\frac{\partial \mathrm{E}_t(j)}{\partial \tau} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j} \right)^{\mathrm{T}} \frac{\partial \boldsymbol{y}^{\mathrm{DMP}}}{\partial \tau}(x_{i,j}). \tag{23}$$

To compute the partial derivatives $\partial y_l^{\mathrm{DMP}} / \partial \tau$ at phases $x_{i,j}$, we first calculate the partial derivatives of Eqs. (7) and (8) with respect to τ

$$\tau \frac{\partial \dot{z}_l}{\partial \tau} = \alpha_z \left(-\beta_z \frac{\partial y_l}{\partial \tau} - \frac{\partial z_l}{\partial \tau} \right) - \dot{z}_l + (g_l - y_{0,l}) \frac{\partial}{\partial \tau} \left(\frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x \right), \tag{24}$$

$$\tau \frac{\partial \dot{y}_l}{\partial \tau} = \frac{\partial z_l}{\partial \tau} - \dot{y}_l. \tag{25}$$

Since x depends on τ, we also need to compute

$$\frac{\partial}{\partial \tau} \left(\frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x \right) = \frac{\left(\sum_{n=1}^{N} w_{l,n} (\Psi_n'(x) x + \Psi_n(x)) \right) \left(\sum_{n=1}^{N} \Psi_n(x) \right)}{\left(\sum_{n=1}^{N} \Psi_n(x) \right)^2} \frac{\partial x}{\partial \tau} - \frac{\left(\sum_{n=1}^{N} \Psi_n'(x) \right) \left(\sum_{n=1}^{N} w_{l,n} \Psi_n(x) x \right)}{\left(\sum_{n=1}^{N} \Psi_n(x) \right)^2} \frac{\partial x}{\partial \tau}. \tag{26}$$

Finally, differential equation (9) needs to be differentiated with respect to τ to compute the partial derivative $\partial x / \partial \tau$, which appears in the equation above. We obtain

$$\tau \frac{\partial \dot{x}}{\partial \tau} = -\alpha_x \frac{\partial x}{\partial \tau} - \dot{x}. \tag{27}$$

Thus, to calculate $\partial \boldsymbol{y}^{\mathrm{DMP}} / \partial \tau$, we need to integrate $2d+1$ equations comprising differential equation system (24), (25), and (27) in $\partial y_l / \partial \tau$, $\partial z_l / \partial \tau$, and $\partial x / \partial \tau$, with initial values set to

$$\frac{\partial y_l}{\partial \tau}(1) = \frac{\partial z_l}{\partial \tau}(1) = \frac{\partial x}{\partial \tau}(1) = 0, \quad l = 1, \ldots, d. \tag{28}$$

The initialization above is because the initial positions on the trajectory and the initial value of the phase do not depend on τ.

Note that differential equation system (24), (25), (27) contains the values of $\dot{y}_l$, $\dot{z}_l$, x, and $\dot{x}$, thus the DMP differential equation system (7)–(9) must be integrated simultaneously to have all the necessary quantities available. If one wanted to avoid the rather complicated calculation of partial derivative $\partial \mathrm{E}_t(j) / \partial \tau$ as specified by Eq. (23), one could estimate τ in a separate deep neural network and consider τ as constant when optimizing the loss function (5) to calculate the rest of the DMP parameters, i.e. $\{\boldsymbol{w}_k\}_{k=1}^{N}$, $\boldsymbol{g}$, and $\boldsymbol{y}_0$. If this is done, the resulting neural network has one neuron less on the output layer.

References

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A deep convolutional encoder-Decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495.

Chen, N., Bayer, J., Urban, S., & van der Smagt, P. (2015). Efficient movement representation by embedding dynamic movement primitives in deep autoencoders. In *2015 IEEE-RAS 15th international conference on humanoid robots (Humanoids)* (pp. 434–440).

Chen, N., Karl, M., & van der Smagt, P. (2016). Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *2016 IEEE-RAS 16th international conference on humanoid robots (Humanoids)* (pp. 629–636).

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In *IEEE conference on computer vision and pattern recognition* (pp. 2414–2423). Las Vegas, Nevada: IEEE.

Gašpar, T., Nemec, B., Morimoto, J., & Ude, A. (2018). Skill learning and action recognition by arc-length dynamic movement primitives. *Robotics and Autonomous Systems*, *100*, 225–235.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Goudar, V., & Buonomano, D. V. (2018). Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks. *eLife*, *7*(e31134).

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, *25*(2), 328–373.

Kim, W., Lee, C., & Kim, H. J. (2018). Learning and generalization of dynamic movement primitives by hierarchical deep reinforcement learning from demonstration. In *2018 IEEE/RSJ international conference on intelligent robots and systems* (pp. 3117–3123).

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd international conference for learning representations*, San Diego.

Kober, J., & Peters, J. (2010). Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, *17*(2), 55–62.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Cortes, C., & Burges, C. (2019). The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/ (accessed on September 18th, 2019).

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, *17*(1), 1334–1373.

Pahič, R., Gams, A., Ude, A., & Morimoto, J. (2018). Deep encoder-Decoder networks for mapping raw images to dynamic movement primitives. In *IEEE international conference on robotics and automation*, Brisbane, Australia (pp. 5863–5868).

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS 2017 autodiff workshop: The future of gradient-based machine learning software and techniques*.

Pervez, A., Mao, Y., & Lee, D. (2017). Learning deep movement primitives using convolutional neural networks. In *2017 IEEE-RAS 17th international conference on humanoid robotics (Humanoids)* (pp. 191–197).

Pierson, H. A., & Gashler, M. S. (2017). Deep learning in robotics: a review of recent research. *Advanced Robotics*, *31*(16), 821–835.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *26*(1), 43–49.

Schaal, S., Mohajerian, P., & Ijspeert, A. J. (2007). Dynamics systems vs. optimal control — a unifying view. In *Computational neuroscience: Theoretical insights into brain function* (pp. 425–445). Elsevier.

Stasse, O., Flayols, T., Budhiraja, R., Giraud-Esclasse, K., Carpentier, J., Mirabel, J., et al. (2017). TALOS: A new humanoid research platform targeted for industrial applications. In *IEEE-RAS 17th international conference on humanoid robotics (Humanoids)*, Birmingham, UK (pp. 689–695).

Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., et al. (2018). The limits and potentials of deep learning for robotics. *International Journal of Robotics Research*, *37*(4–5), 405–420.

Ude, A., Gams, A., Asfour, T., & Morimoto, J. (2010). Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, *26*(5), 800–815.