Deep Encoder-Decoder Networks for Mapping Raw Images to Dynamic Movement Primitives

Rok Pahič[1,2], Andrej Gams[1], Aleš Ude[1,2], and Jun Morimoto[2]

***Abstract*— In this paper we propose a new approach for learning perception-action couplings. We show that by collecting a suitable set of raw images and the associated movement trajectories, a deep encoder-decoder network can be trained that takes raw images as input and outputs the corresponding dynamic movement primitives. We propose suitable cost functions for training the network and describe how to calculate their gradients to enable effective training by back-propagation. We tested the proposed approach both on a synthetic dataset and on a widely used MNIST database to generate handwriting movements from raw images of digits. The calculated movements were also applied for digit writing with a real robot.**

I. INTRODUCTION

Autonomous cognitive robots are expected to be able to perceive their environment and interact with the external world. This makes it necessary to provide them with a framework that combines perception and action. Many different approaches that combine perception and actions have been proposed in the literature, such as for example object-action complexes (OACs) [1], which is a grounded representation that binds objects, actions, and attributes in a causal model. The main feature of OACs is that they join perception-action space of an agent with its planing / reasoning capabilities. However, the learning of effective perception-action couplings remains a difficult problem.

In recent years deep neural networks have been applied to various fields with significant success [2], e. g. in visual recognition tasks and in natural language processing. The ability of deep neural networks to learn highly nonlinear transformations between the input and output data makes them a good candidate for learning direct perception-action couplings. Consequently, deep learning is a quickly growing research area in robotics, where it has been shown that it is suitable for learning end-to-end visuomotor policies that require close coordination between vision and control, such as screwing a cap onto a bottle [3]. Deep Bayesian convolutional neural networks have been applied to estimate the model's relocalization uncertainty and improve state of the art localization accuracy [4]. In yet another work, a deep convolutional autoencoder network that extracts images features and a fully connected deep time delay neural network that learns the dynamics of a robot task process have been proposed and tested in the context of object folding [5].

Published in: IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, pp. 5863-5868, 2018.

[1]Dept. of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia rok.pahic@ijs.si, andrej.gams@ijs.si, ales.ude@ijs.si

[2]ATR Computational Neuroscience Laboratories, Kyoto, Japan xmorimo@atr.jp

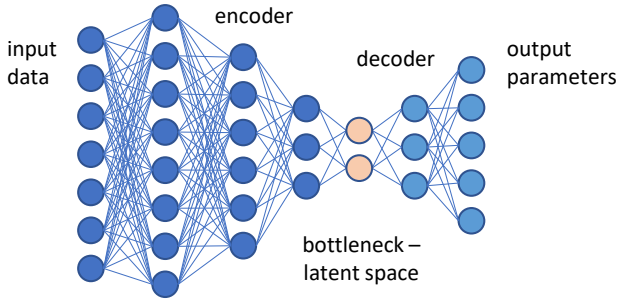

Fig. 1. An example encoder-decoder network architecture with five hidden layers

We are interested to apply deep neural networks in the context of imitation learning. Previous works have shown that autoencoders [6] or variational autoencoders [7] can be used to reduce the dimensionality of the movements obtained by human demonstration and effectively train dynamic movement primitives in a low-dimensional latent space. In these works, the training data were acquired either synthetically or by motion capture. Deep autoencoders have also been shown to significantly outperform other technologies, e. g. principal component analysis, in computing low dimensional latent spaces from raw character images [8]. A shift invariant autoencoder was developed [9] to extract a typical spatial subpattern independent of its relative position in an image.

Our focus is on learning direct mappings between images and actions. So instead of autoencoder networks, we are interested in encoder-decoder networks, where transformation and generalization of the incoming image data into the output robot actions occurs through a low-dimensional latent space. Encoder-decoder networks in combination with convolutional layers have proven to be useful in computer vision. One of the best known examples is SegNet deep convolutional encoder-decoder architecture for semantic pixel-wise segmentation [10]. A similar architecture have proven to be successful at object contour detection [11]. Convolutional neural networks have also been employed to generate task parametrized dynamic movement primitives [12].

Unlike the above-mentioned encoder-decoder networks, which convert raw images into another pixel-wise image representation, in this work we investigate the conversion of raw images into movements associated with the images. As a practical example we consider the problem of reproducing handwritten digits, i. e. we propose to train an encoder-decoder network which calculates a movement that generated a handwritten digit directly from its image. Other

possible applications of our work are discussed in Section VI, Conclusions and future directions.

II. DEEP ENCODER-DECODER NETWORKS

As autoencoders have shown good performance for the calculation of low-dimensional latent space representations of human movements [6], [7] as well as for the reproduction of images of handwritten characters [8], and encoder-decoder networks have shown to be useful for converting raw images into different image representations, we designed our architecture for decoding movements from images as a fully-connected encoder-decoder architecture. A simple example encoder-decoder network is shown in Fig. 1.

Much larger networks than the one shown in Fig. 1 were used in our experiments, where typically an input layer consisted of 1600 neurons, there were 7 hidden layers with 1500, 1300, 1000, 600, 200, 20, and 35 neurons, respectively, and the output layer consisted of 55 neurons. As shown in [8], fully connected autoencoders work well for handwriting images with simple backgrounds, therefore we did not integrate convolutional layers into the proposed network architecture. But this is certainly an option, especially once images with more complicated backgrounds start being used as input data.

We use the following input-output couples for training of the above-described network:

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

where M is the number of training couples, $\mathbf{C}_j \in \mathbb{R}^{H \times W}$ are the input images of width W and height H, and $\mathbf{M}_j$ the corresponding movements associated with each image, i. e.

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

Here $\boldsymbol{y}_{i,j} \in \mathbb{R}^{\mathrm{d}}$ are the vectors describing the movement's degrees of freedom, e. g. Cartesian positions or joint angles, $t_{i,j} \in \mathbb{R}$ the measurement times for the j-th movement, and d is the number of degrees of freedom.

III. TRAJECTORY REPRESENTATION BY DYNAMIC MOVEMENT PRIMITIVES

As explained above, each image in the database is associated with the corresponding movement that created it. An effective movement representation for both learning and control are dynamic movement primitives (DMPs) developed by Ijspeert et al. [13]. Let's denote a time-dependent movement trajectory as $\boldsymbol{y}(t) \in \mathbb{R}^d$. A DMP specifying this trajectory is given by the following system of differential equations

$$\tau \dot{\boldsymbol{z}} = \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - \boldsymbol{z}) + \mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}_0)\mathbf{F}(x), \tag{3}$$

$$\tau \dot{\boldsymbol{y}} = \boldsymbol{z}, \tag{4}$$

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 following equation

$$\tau \dot{x} = -\alpha_x x. \tag{5}$$

The phase x is used instead of time to avoid explicit time dependency. It is fully defined by setting its initial value to $x(0) = 1$. Eq. system (3) – (5) constitutes a *dynamic movement primitive* (DMP). 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 (3) – (4) 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

$$\mathbf{F}(x) = \frac{\sum_{k=1}^{N} \boldsymbol{w}_k \Psi_k(x)}{\sum_{k=1}^{N} \Psi_k(x)} x, \tag{6}$$

$$\Psi_k(x) = \exp\left(-h_k (x - c_k)^2\right), \tag{7}$$

where c_k are the centers of Gaussians distributed along the phase of the trajectory, and h_k their widths. The role of $\mathbf{F}$ is to adapt the dynamics of (3) – (4) to the desired trajectory $\boldsymbol{y}(t)$, thus enabling the system to reproduce any smooth movement from the initial position $\boldsymbol{y}_0$ to the final configuration $\boldsymbol{g}$. This can be accomplished by computing the free parameters $\boldsymbol{w}_k \in \mathbb{R}^d$ using regression techniques. See [14] for more details.

α_z, β_z, and α_x are usually constants that do not change between movements. Thus the neural network needs to learn the other parameters of differential equation system (3) – (5) to fully specify a DMP:

$$\{\boldsymbol{w}_k\}_{k=1}^{N},\ \tau,\ \boldsymbol{g},\ \boldsymbol{y}_0. \tag{8}$$

IV. COST FUNCTION AND ITS GRADIENT

A simple cost function for evaluating the output of the encoder-decoder neural network would be to convert the example movements $\mathbf{M}_j$ into DMPs and define the cost function for the j-th DMP as follows

$$\mathrm{E}_p(j) = \frac{1}{2}\left(\sum_{k=1}^{N} \|\boldsymbol{w}_k - \boldsymbol{w}_{k,j}\|^2 + (\tau - \tau_j)^2 + \|\boldsymbol{g} - \boldsymbol{g}_j\|^2 + \|\boldsymbol{y}_0 - \boldsymbol{y}_{0,j}\|^2\right). \tag{9}$$

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 calculated from the training data (1). While this cost function provides for an easy implementation, it does not measure directly the difference between the training movement and the movement calculated by the neural network, but rather the difference between the DMP parameters. If we want to directly measure the difference between the trajectories defined by DMP parameters calculated by a neural network, here denoted as $\boldsymbol{y}^{\mathrm{DMP}}$, and the training data $\boldsymbol{y}_{i,j}$ from Eq. (2), we need to define the following cost function

$$\mathrm{E}_t(j) = \frac{1}{2T_j} \sum_{i=1}^{T_j} \|\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j}\|^2, \tag{10}$$

where $\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j})$ and $x_{i,j} = x(t_{i,j})$ are obtained by integrating the DMP calculated by the neural network.

A. Calculating the gradient of the cost function

Backpropagation [15] is the method of choice for learning deep neural networks. Backpropagation requires the gradients of the error function to be available. The gradients of error function (9) are trivial to compute as this is simply the Euclidean distance between the training DMP parameters and the parameters computed by the neural network. But it is more difficult to compute the gradients of (10) as $\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j})$ are calculated by integrating DMP equations (3) – (5).

The partial derivative of the cost function $\mathrm{E}_t(j)$ with respect to $w_{l,k},\ k = 1, \ldots, N,\ l = 1, \ldots, d$, are calculated as follows

$$\frac{\partial \mathrm{E}_t(j)}{\partial w_{l,k}} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(y_l^{\mathrm{DMP}}(x_{i,j}) - y_{l,i,j} \right) \frac{\partial y_l^{\mathrm{DMP}}}{\partial w_{l,k}}(x_{i,j}), \tag{11}$$

where $\partial y_l^{\mathrm{DMP}} / \partial w_{l,k}$ is the partial derivative of y_l^{DMP} with respect to $w_{l,k}$. These derivatives can be obtained by calculating the derivatives of Eq. (3) and (4) with respect to $w_{l,k}$

$$\begin{aligned} \tau \frac{\partial \dot{z}_l}{\partial w_{l,k}} &= \alpha_z(-\beta_z \frac{\partial y_l}{\partial w_{l,k}} - \frac{\partial z_l}{\partial w_{l,k}}) + \\ &\quad (g_l - y_{l,0}) \frac{\psi_k(x)}{\sum_{n=1}^N \Psi_n(x)} x, \end{aligned} \tag{12}$$

$$\tau \frac{\partial \dot{y}_l}{\partial w_{l,k}} = \frac{\partial z_l}{\partial w_{l,k}}. \tag{13}$$

and integrating the resulting differential equation system in $\partial y_l / \partial w_{l,k}$ and $\partial z_l / \partial w_{l,k}$. This can be done because the following holds for continuously differentiable trajectories

$$\begin{aligned} \frac{d}{dt} \frac{\partial}{\partial w_{l,k}} z_l &= \frac{\partial}{\partial w_{l,k}} \frac{d}{dt} z_l, \\ \frac{d}{dt} \frac{\partial}{\partial w_{l,k}} y_l &= \frac{\partial}{\partial w_{l,k}} \frac{d}{dt} y_l. \end{aligned}$$

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 (12) – (13). 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

$$\frac{\partial y_l}{\partial w_{l,k}}(1) = \frac{\partial z_l}{\partial w_{l,k}}(1) = 0. \tag{14}$$

The partial derivatives with respect to the parameters $g_l,\ y_{0,l},\ l = 1, \ldots, d$, are obtained analogously. We start by computing

$$\frac{\partial \mathrm{E}_t(j)}{\partial g_l} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(y_l^{\mathrm{DMP}}(x_{i,j}) - y_{l,i,j} \right) \frac{\partial y_l^{\mathrm{DMP}}}{\partial g_l}(x_{i,j}), \tag{15}$$

$$\frac{\partial \mathrm{E}_t(j)}{\partial y_{0,l}} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(y_l^{\mathrm{DMP}}(x_{i,j}) - y_{l,i,j} \right) \frac{\partial y_l^{\mathrm{DMP}}}{\partial y_{0,l}}(x_{i,j}). \tag{16}$$

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 Eq. (3) and (4) with respect to g_l and $y_{0,l}$, which results in

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

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

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

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

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

In equation (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. (5). We start by computing

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

To compute the partial derivatives $\partial y_l^{\mathrm{DMP}} / \partial \tau$ at phases $x_{i,j}$, we first calculate the partial derivatives of Eq. (3) and (4) with respect to τ

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

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

Since x depends on τ, we also need to compute

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

Finally, differential equation (5) 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

$$\tau\frac{\partial\dot{x}}{\partial\tau} = -\alpha_x\frac{\partial x}{\partial\tau} - \dot{x}. \tag{26}$$

Thus, to calculate $\partial\boldsymbol{y}^{\mathrm{DMP}}/\partial\tau$, we need to integrate $2d+1$ equations comprising differential equation system (24) – (26) in $\partial y_l/\partial\tau$, $\partial z_l/\partial\tau$, and $\partial x/\partial\tau$, with initial values set to

$$\frac{\partial y_l}{\partial\tau}(1) = \frac{\partial z_l}{\partial\tau}(1) = \frac{\partial x}{\partial\tau}(1) = 0, \quad l = 1,\ldots,d. \tag{27}$$

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

Note that the differential equation system (24) – (26) contains the values of $\dot{y}_l$, $\dot{z}_l$, x, and $\dot{x}$, thus the DMP differential equation system (3) – (5) must be integrated simultaneously to have all the necessary parameters available. If one wanted to avoid the rather complicated calculation of partial derivative $\partial\mathrm{E}_t(j)/\partial\tau$ from Eq. (23), one could estimate τ in a separate deep neural network and consider τ as constant when optimizing criterion function (10) to calculate the rest of the DMP parameters, i. e. $\{\boldsymbol{w}_k\}_{k=1}^N$, $\boldsymbol{g}$, and $\boldsymbol{y}_0$. In this case the neural network would have one less neuron in the output layer.

V. EXPERIMENTAL RESULTS

In the first set of experiments we used a database that consisted of 40×40 images of synthetically written digits and the associated two-dimensional handwriting movements. For a DMP representation of the movement trajectory we chose 25 radial-basis functions for every dimension. The weights of these basis functions form together with the common time constant (1 parameter) and the start and the goal values of a planar movement (2×2 parameters), the set of 55 DMP parameters computed as the output of our neural network. The database was generated using a computer program that produces handwritten digits from a combination of straight lines and elliptic arcs. When generating these geometric elements, we varied the parameters such as length and angle of a straight line or length of minor and major axis, angle between axes, and ellipse center of an elliptic arc. These parameters varied according to a uniform distribution. From these trajectories, grayscale images were generated with the predefined width. The resulting images were processed with a Gaussian filter and some moderate salt-and-pepper noise was added to the foreground pixels. Finally, both the generated trajectories and the resulting images were transformed using affine transformations composed of translation, rotation, scaling, and shearing. These parameters were again taken from a uniform distribution.

In this synthetic experiment, for each digit from zero to nine 2000 couples of images and the corresponding trajectories were generated. Thus the training database consisted of 20000 image - handwriting trajectory couples, together constituting dataset (1). Some example images and the associated trajectories that were used for evaluation are shown in Fig. 4. The data from Fig. 4 were not part of the training database. Typically about 25000 training updates were performed before the network learned to approximate the training data well and acquired good generalization properties. It is clear that the trajectories generated by the output DMPs (shown in red) are close to the original trajectories used to generate the images (shown in blue), although the DMPs were generated by the proposed encoder-decoder neural network using grayscale images only as input. Note that the same network synthesizes the handwriting movements for all ten digits; it is not necessary to train a separate network for classification. Evaluation database was generated in the same way as the training database, with the same size of 2000 examples for each out of ten digits. The statistics of reconstruction quality is shown in Tab. I. It is clear that by using criterion function (10), we can obtain significantly better results than with criterion function (9), although this second cost function is also able to reproduce the digits, albeit with lower quality. As one could expect, the third row in Tab. I show that the errors are smaller if the network is specifically trained for one digit only.

We were also able to use the DMPs computed by the proposed encoder-decoder network for writing all 10 digits with a Kuka LightWeight Robot arm (LWR). For writing with a pen, we used admittance control with a specified contact force in direction of the pen tip while following a

TABLE I

DMP RECONSTRUCTION STATISTICS. THE RESULTS ARE IN PIXELS.

	Mean	**Std. deviation**
Synthetic data, criterion function (9), 10 digits	0.40	0.84
Synthetic data, criterion function (10), 10 digits	0.23	0.56
Synthetic data, criterion function (9), digit 2 only	0.11	0.08
MNIST database, criterion function (9), 2 digits	1.23	2.50
MNIST database, criterion function (10), 2 digits	0.85	2.28

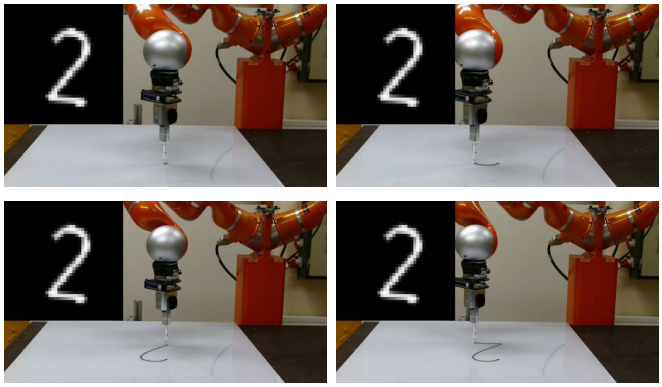

Fig. 2. Writing digit 2 with a robot. The movement is calculated from the image shown in upper left corner. See also the video that accompanies this paper.

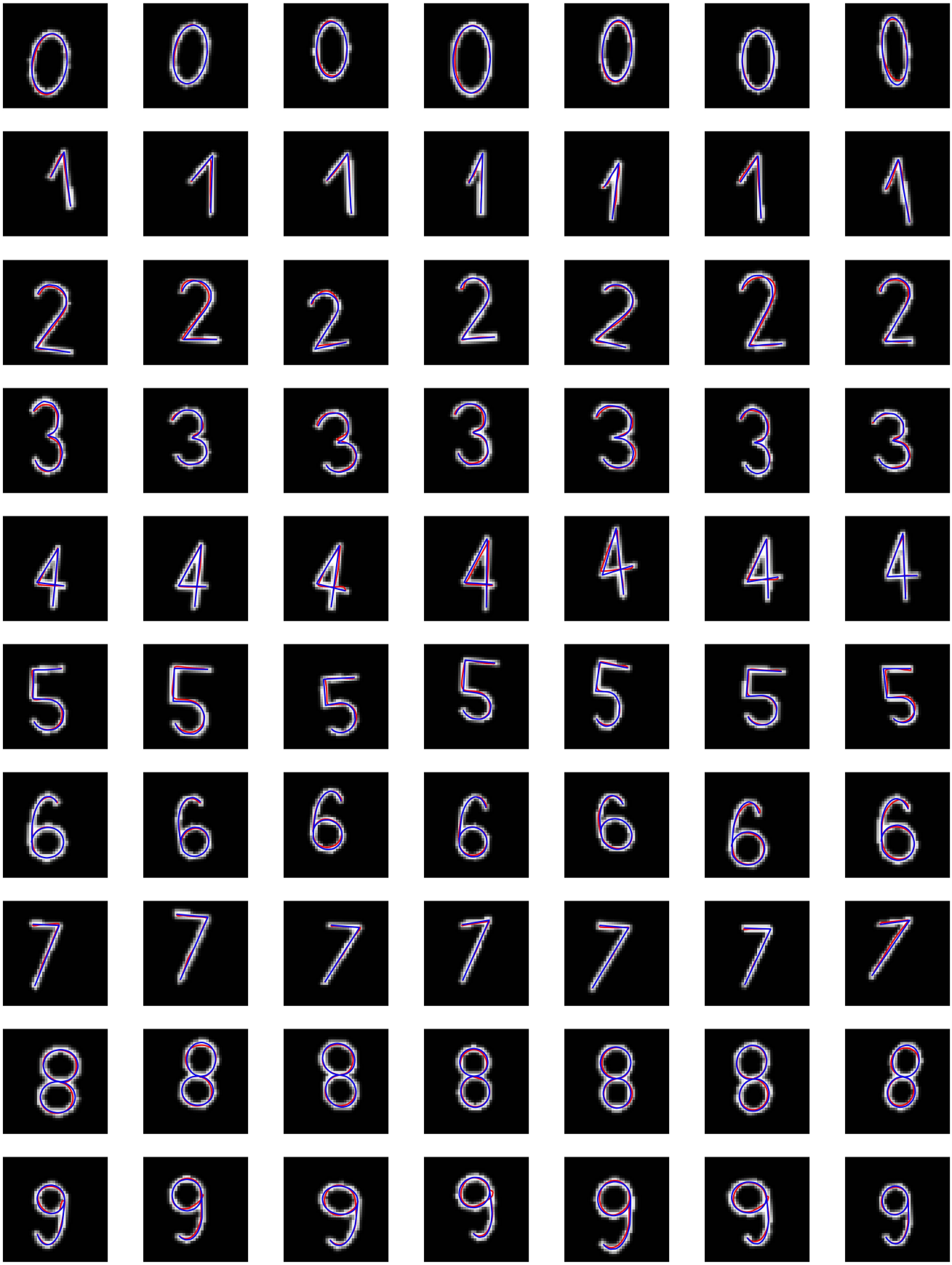

Fig. 4. Example synthetic data showing the images of digits and the associated handwriting trajectories used to generate them. The original trajectories are shown in blue, while the DMP trajectories calculated by a neural network are shown in red. Our neural network is able to reconstruct handwriting trajectories well although these images and trajectories were not used for training.

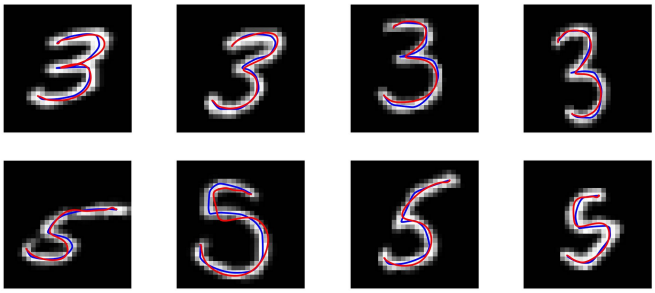

Fig. 3. The results of reconstruction for digits from MNIST dataset. The manually generated and transformed trajectories are shown in blue, while the DMP trajectories calculated by a neural network are shown in red. These data were used only for testing, not for training. Hence these results show the generalization performance of the proposed encoder-decoder network.

trajectory specified by the output DMP. A few screenshots from this experiments are shown in Fig. 2.

In the final experiment we tested our approach on a standard MNIST dataset [16], which is widely used as a benchmark for testing classification algorithms in handwritten digit recognition systems. Using a touch interface we annotated 1170 digits 3 and 1170 digits 5 from this dataset with the corresponding handwriting movements.

For training we generated altogether 11700 digits 3 and 11700 digits 5 by applying affine transformations to the original images and the manually added handwriting movements. The results are shown in Tab. I. While the errors are by an order of magnitude larger than with synthetic data, the proposed deep encoder-decoder network is still able to find a good approximation of the handwriting motion. This can be confirmed by analyzing Fig. 3. The main reason for larger errors is that the MNIST database digits vary significantly more than the synthetically generated digits, thus more data is needed before the network can learn the digits well.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

We presented a new approach to convert greyscale images into the associated dynamic movement primitives using deep encoder-decoder neural networks. Our experimental results show that with the proposed approach, handwriting movements for digits can effectively be reconstructed from raw images of digits.

There are quite a few possibilities to extend the proposed approach. Our plan for the immediate future work is to explore the application of convolutional neural networks for the encoder part of the proposed architecture. This will enable more effective processing of real images, which should be beneficial especially for images with cluttered background. In connection to this we will investigate how pretrained convolutional neural networks can be exploited for faster training of the proposed encoder-decoder networks. Another possible extension is movement reproduction from image sequences, which requires an effective dimensionality reduction to be feasible.

While in this paper we focused on the reproduction of handwriting movements, our work has many potential applications beyond this task. For example, in the future we plan to associate assistive robot actions with pointing gestures observed by a robot. The image or image sequence containing the pointing gesture will in this case be fed as input to the encoder-decoder network and the output will be the robot action associated with the gesture. This way an effective human-robot interaction system could be created.

Acknowledgement: This work has received funding from the EU's Horizon 2020 RIA AUTOWARE (GA no. 723909); the Slovenian Research Agency under GA no. J2-7360; JSPS KAKENHI JP16H06565; NEDO; AMED SRPBS; ImPACT Program of Council for Science, Technology, and Innovation (Cabinet Office, Government of Japan); and the Commissioned Research of NICT.

REFERENCES

[1] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrčen, A. Agostini, and R. Dillmann, "Object-Action Complexes: Grounded abstractions of sensory-motor processes," *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 740–757, 2011.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, 2016.

[4] A. Kendall and R. Cipolla, "Modelling uncertainty in deep learning for camera relocalization," in *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, 2016, pp. 4762–4769.

[5] P.-C. Yang, K. Sasaki, K. Suzuki, K. Kase, S. Sugano, and T. Ogata, "Repeatable Folding Task by Humanoid Robot Worker Using Deep Learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 397–403, 2016.

[6] N. Chen, J. Bayer, S. Urban, and P. van der Smagt, "Efficient movement representation by embedding Dynamic Movement Primitives in deep autoencoders," in *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, Seoul, Korea, 2015, pp. 434–440.

[7] N. Chen, M. Karl, and P. van der Smagt, "Dynamic movement primitives in latent space of time-dependent variational autoencoders," in *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, Cancun, Mexico, 2016, pp. 629–636.

[8] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[9] T. Matsuo, H. Fukuhara, and N. Shimada, "Transform Invariant Auto-encoder," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, 2017, pp. 2359–2364.

[10] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for scene segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016, doi: 10.1109/TPAMI.2016.2644615.

[11] J. Yang, B. Price, S. Cohen, H. Lee, and M.-H. Yang, "Object Contour Detection with a Fully Convolutional Encoder-Decoder Network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, Nevada, 2016, pp. 193–202.

[12] A. Pervez, Y. Mao, and D. Lee, "Learning Deep Movement Primitives using Convolutional Neural Networks," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Birmingham, UK, 2017, pp. 191–197.

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

[14] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[16] Y. LeCun, C. Cortes, and C. J. Burges, "The MNIST database of handwritten digits," http://yann.lecun.com/exdb/mnist/, accessed: 2017-09-15.