LEARNING OF DYNAMIC MOVEMENT PRIMITIVES WITH DEEP NEURAL NETWORKS

Rok Pahič

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Assoc. Prof. Dr. Aleš Ude, Jožef Stefan International Postgraduate School and Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Assoc. Prof. Dr. Andrej Gams, Chair, Jožef Stefan International Postgraduate School and Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Sašo Džeroski, Member, Jožef Stefan International Postgraduate School and Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Florentin Wörgötter, Member, Georg-August University Göttingen, Göttingen, Germany

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Rok Pahič

LEARNING OF DYNAMIC MOVEMENT PRIMITIVES WITH DEEP NEURAL NETWORKS

Doctoral Dissertation

UČENJE DINAMIČNIH GENERATORJEV GIBOV Z GLOBOKIMI NEVRONSKIMI MREŽAMI

Doktorska disertacija

Supervisor: Assoc. Prof. Dr. Aleš Ude

Ljubljana, Slovenia, March 2021

To my family and friends

Acknowledgments

I would like to thank my supervisor Prof. Dr. Aleš Ude for his guidance and help with my research work. This research work was also done in collaboration with Dr. Barry Ridge, Dr. Andrej Gams and Zvezdan Lončarević, I would like to thank them all for their help.

Next, I would like to thank my colleagues at the Department of Automatics, Biocybernetics and Robotics at the Jožef Stefan Institute. Dr. Miha Dežman for interesting research discussions, which enriched the working day with topics of mechanical engineering. Special thanks also to colleagues who were always prepared for immediate help with their robotics expertise: Mihael Simonič, Dr. Aljaž Kramberger, Dr. Miha Deniša, Dr. Bojan Nemec, and Dr. Leon Žlajpah.

I would also like to thank my parents Franc and Lidija for their support through-out my education.

This research has received funding from the Slovenian Research Agency (ARRS) young researcher grant PR-07602.

Abstract

Robots that are supposed to perform human-like tasks must possess appropriate skills to carry them out. In unstructured environments and for complex tasks, these skills are difficult to pre-program due to the complexity of the real world. It is therefore advantageous if robots have the ability to acquire the necessary skills by learning.

Dynamic movement primitives (DMPs) have proven to be an effective movement representation for motor skill learning. In this thesis, we propose a new approach for training deep neural networks to synthesize DMPs. 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 backpropagation to optimize the parameters of the underlying deep neural network.

The choice of an appropriate representation is important when reconstructing motion. A version of DMPs called arc-length dynamic movement primitive (AL-DMP) can separate the spatial from temporal aspects of motion and is more suitable for processing data that do not contain temporal information. We therefore extended our approach to neural networks that can synthesize spatial paths represented by AL-DMPs.

While the developed approaches are applicable to any neural network architecture, they were evaluated on two different architectures based on encoder–decoder networks and convolutional neural networks. Moreover, we proposed deep neural network architectures that support the processing of variable-size images and images with cluttered background. The developed approaches were applied for the reproduction of handwritten digits from single images. Our results show that the minimization of the proposed loss functions leads to better results than when more conventional loss functions are used. Our experiments also show that the network can be applied to input images of sizes that are different from the size of training images. Finally, the proposed approaches were successfully applied for reproducing real handwritten digits with a humanoid robot.

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 thesis, 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 (PCA).

Povzetek

Tako kot ljudje tudi roboti, ki naj bi opravljali posamezne naloge namesto človeka, potrebujejo ustrezne veščine za te naloge. V nestrukturiranem okolju in za zapletene naloge je zaradi zapletenosti te veščine težko sprogramirati vnaprej. Bolje je, da robotom omogočimo, da se sami naučijo potrebnih spretnosti.

Dinamični generatorji gibov (DGG) so se izkazali kot učinkovita predstavitev gibanja pri učenju gibalnih spretnosti. V tej disertaciji predlagamo nov pristop k učenju globokih nevronskih mrež za računanje DGG. Poglavitna novost našega pristopa je, da uporabljamo novo optimizacijsko funkcijo, ki meri fizično razdaljo med trajektorijami gibanja in ne le napako med parametri DGG, ki nimajo fizičnega pomena. Ta pristop smo omogočili z izpeljavo diferencialnih enačb za izračun gradientov predlagane optimizacijske funkcije, kar omogoča uporabo metode vzvratnega razširjanja napake za optimizacijo parametrov globokih nevronskih mrež.

Pri rekonstrukciji gibanja je pomembna izbira ustrezne oblike predstavitve gibanja. Različica DGG, imenovana dinamični elementarni gibi po naravnem parametru (NP-DGG), lahko loči prostorski in časovni vidik gibanja in je primernejša za obdelavo podatkov brez informacije o časovnem poteku. Zato smo predlagano metodologijo razširili iz DGG na nevronske mreže, ki sintetizirajo gibalne poti, predstavljene z NP-DGG.

Razviti pristopi so sicer uporabni za katero koli arhitekturo nevronske mreže, vendar smo jih v disertaciji preizkusili na dveh različnih arhitekturah, ki temeljita na kodirno-dekodirnih mrežah in konvolucijskih nevronskih mrežah. Predlagali smo tudi dodatne arhitekture globokih nevronskih mrež, ki podpirajo obdelavo slik spremenljive velikosti in slik z nestrukturiranim ozadjem. Razviti pristopi so bili uporabljeni za reprodukcijo ročno napisanih števk iz digitalnih slik. Naši rezultati kažejo, da uporaba predlaganih optimizacijskih funkcij daje boljše rezultate kot uporaba običajnih optimizacijskih funkcij, ki merijo napako med parametri. Iz naših rezultatov je tudi razvidno, da predlagana nevronska mreža omogoča uporabo vhodnih slik drugačne velikosti kot slike v podatkovni zbirki za učenje. Predlagane pristope smo na koncu uspešno uporabili za reprodukcijo realnih rokopisnih števk s humanoidnim robotom.

Tako kot ljudje lahko tudi roboti izboljšajo svoje sposobnosti z vadbo (učenjem), to je z večkratno izvedbo želene naloge in posodobitvijo parametrov z uporabo na novo zbranih podatkov. V tej disertaciji predlagamo učenje robotov z uporabo statističnega in spodbujevalnega učenja v latentnem prostoru želene naloge. Latentni prostor izračuna globoka nevronska mreža za avtokodiranje, pri čemer podatke za učenje mreže ustvarimo v simulaciji. Z eksperimenti smo pokazali, da je tako generirana latentna predstavitev uporabna tudi za učenje na realnem robotu.

Rezultati simulacij in eksperimentov z realnim robotom kažejo, da je mogoče z uporabo latentnega prostora gibalnih veščin doseči znatno zmanjšanje količine podatkov, potrebnih za učinkovito učenje z Gaussovo regresijo. Podobno lahko znatno zmanjšamo število epizod spodbujevalnega učenja. Prav tako je latentni prostor, ki temelji na avtokoderju, učinkovitejši kot latentni prostor, izračunan z metodo glavnih komponent.

Contents

List of Figures xv

List of Tables xvii

Abbreviations xix

Symbols xxi

1 Introduction 1
- 1.1 Review of the Relevant Scientific Field . . . 5
- 1.2 Thesis Purpose, Goals and Hypothesis . . . 7
- 1.3 Thesis Outline . . . 9

2 Deep Neural Networks for the Generation of Trajectories/Paths from Images 11
- 2.1 Trajectory/Path Representations . . . 12
  - 2.1.1 Dynamic movement primitives . . . 12
  - 2.1.2 Arc-length dynamic movement primitives (AL-DMPs . . . 13
  - 2.1.3 Normalized arc-length dynamic movement primitives . . . 14
  - 2.1.4 Estimation of normalized AL-DMPs . . . 15
  - 2.1.5 Execution of normalized AL-DMPs . . . 16
- 2.2 Datasets for Training Deep Image-to-Motion Encoder-Decoder Networks . . . 16
- 2.3 Loss Functions and Calculation of their Gradients . . . 17
  - 2.3.1 Loss functions for training neural networks with DMP parameters as output . . . 18
  - 2.3.2 Loss functions for training neural networks with normalized AL-DMPs as output . . . 21
- 2.4 Deep Neural Network Architectures . . . 24
  - 2.4.1 Fully-connected encoder-decoder architecture . . . 25
  - 2.4.2 Convolutional encoder-decoder architectures . . . 26
  - 2.4.3 Convolutional encoder-decoder architectures for variable size input images . . . 28
- 2.5 Experimental Evaluation . . . 30
  - 2.5.1 Datasets . . . 30
    - 2.5.1.1 Annotated MNIST (a-MNIST) . . . 31
    - 2.5.1.2 Synthetic MNIST (s-MNIST) . . . 31
    - 2.5.1.3 Synthetic MNIST with background . . . 35
  - 2.5.2 Neural network training . . . 37
    - 2.5.2.1 Number of training parameters and avoidance of overfitting 39
  - 2.5.3 Evaluation of results . . . 40
  - 2.5.4 Robustness of the proposed neural networks to noise . . . 41
  - 2.5.5 Performance of the proposed DMP-based loss functions . . . 43

2.5.6 Comparing performance of the proposed DMP-based and normalized AL-DMP-based loss functions 48
2.5.7 Using input images of different sizes 53
2.5.8 Experiment with real robot images 59

3 Robot Skill Learning in Latent Space of a Deep Autoencoder Neural Network 65
3.1 DMP Latent Space Representations 66
3.1.1 DMP parameter space 66
3.1.2 Autoencoder-based latent space 66
3.1.3 PCA-based latent space 67
3.2 Learning Algorithms 68
3.2.1 Reward-weighted policy learning with importance sampling 68
3.2.2 Gaussian Process Regression 69
3.3 Learning in AE- and PCA-Based Latent Spaces 69
3.3.1 Reinforcement learning in latent spaces 70
3.3.2 Gaussian Processes Regression in latent spaces 70
3.4 Experimental Setup 70
3.4.1 Generation of simulated ball throwing trajectories for training 71
3.4.2 Generating AE and PCA-based latent spaces 75
3.4.3 Dataset for statistical learning 76
3.5 Experimental Results 77
3.5.1 Reproduction error of AE- and PCA-based latent spaces 78
3.5.2 Reinforcement learning experiments 79
3.5.2.1 Dynamic simulation 79
3.5.2.2 Real robot experiments 80
3.5.3 Evaluation of statistical learning in latent spaces 81
3.5.3.1 Performance of GPR 81
3.5.3.2 Incremental dataset augmentation 83

4 Conclusions 87

References 91

Bibliography 99

Biography 103

List of Figures

Figure 2.1: Training of writing DMPs 18
Figure 2.2: Comparison of training with normalized AL-DMPs and DMPs parameters 22
Figure 2.3: IMEDNet (Image-to-Motion Encoder-Decoder Network) neural network 25
Figure 2.4: CIMEDNet (convolutional image-to-motion encoder-decoder network) architecture with only convolutional encoder 27
Figure 2.5: CIMEDNet architecture with fully-connected layers in the encoder part 27
Figure 2.6: CIMEDNet+ (convolutional image-to-motion encoder-decoder network + global polling layer) architecture 29
Figure 2.7: VIMEDNet (variable input to motion encoder-decoder network) architecture 29
Figure 2.8: Examples of annotated digit images from the a-MNIST dataset 31
Figure 2.9: Examples of digit images and writing trajectories from the s-MNIST dataset 32
Figure 2.10: Examples of digit images and writing trajectories from the s-MNIST-AWGN-19.0-SNR dataset 32
Figure 2.11: Examples of annotated digit images from the s-MNIST-AWGN-9.5-SNR dataset 33
Figure 2.12: Examples of annotated digit images from the s-MNIST-MB dataset . . 33
Figure 2.13: Examples of annotated digit images from the s-MNIST-RC-AWGN dataset 34
Figure 2.14: Examples of annotated digit images from the s-MNIST-GRAY dataset . 34
Figure 2.15: Examples of training images from the s-MNIST with a background dataset 36
Figure 2.16: Examples of training images from the s-MNIST-GRAY with a background dataset 37
Figure 2.17: Example results for two different neural networks IMEDNet and CIMEDNet presented on various different noise profiles of the s-MNIST data . 42
Figure 2.18: Convergence of training and validation errors for evaluation of DMP-based loss functions 45
Figure 2.19: Example reconstruction results for digits from the a-MNIST dataset for DMP-based loss function 46
Figure 2.20: Example reconstruction results for digits from the s-MNIST dataset for DMP-based loss function 47
Figure 2.21: Convergence of training and validation errors for comparing DMP-based and normalized AL-DMP-based loss functions 51
Figure 2.22: CIMEDNet reconstruction results for training with DMP trajectory and normalized AL-DMP path loss functions. 52
Figure 2.23: Visualization of writing trajectories generated by VIMEDNet trained with DMP trajectory loss function. 56
Figure 2.24: Visualization of writing trajectories generated by VIMEDNet trained with normalized AL-DMP path loss function. 57

Figure 2.25: Convergence of training and validation errors when processing input images of different sizes. 58
Figure 2.26: Comparison of the experimental setup for testing the CIMEDNet and VIMEDNet neural network architecture. 60
Figure 2.27: Writing digits with the humanoid robot TALOS. 62
Figure 2.28: Digits written by TALOS using DMPs that were output by CIMEDNet and VIMEDNet. 63
Figure 2.29: Digits written by TALOS using normalized AL-DMPs that were output by CIMEDNet and VIMEDNet. 64

Figure 3.1: Simple example AE network 66
Figure 3.2: MuJoCo dynamic simulation experimental setup 71
Figure 3.3: Mitsubishi PA-10 robot experimental setup for the evaluation of reinforcement learning and statistical learning in different spaces. 72
Figure 3.4: Simulated ball throwing trajectories dataset 73
Figure 3.5: Example joint trajectories and joint velocities 75
Figure 3.6: Illustration of the designed autoencoder structure with five hidden layers. 76
Figure 3.7: Example points in the resulting AE-based latent space 77
Figure 3.8: Query points for statistical generalization and targets for testing. 78
Figure 3.9: Convergence for reinforcement learning on simulated data 80
Figure 3.10: Number of rollouts to the first hit for reinforcement learning on simulated data 81
Figure 3.11: Convergence of reinforcement learning of throwing movements on a real robot 82
Figure 3.12: Number of rollouts to the first hit for reinforcement learning of throwing movements on a real robot 83
Figure 3.13: Contour plots showing the throwing error resulting from the throwing trajectories computed by GPR in different learning spaces 84
Figure 3.14: Improved performance of statistical learning by autonomous dataset augmentation procedure for experiment in the simulation. 86
Figure 3.15: Improved performance of statistical learning by autonomous dataset augmentation procedure on a real robot 86

List of Tables

Table 2.1: DMP reconstruction statistics comparing robustness of different neural network architectures to noise . . . 43
Table 2.2: DMP reconstruction statistics for DMPs computed by IMEDNet and CIMEDNet trained by DMP-based loss functions . . . 44
Table 2.3: Reconstruction statistics for neural networks with DMP and normalized AL-DMP parameters at the output when using different loss functions. . 49
Table 2.4: Reconstruction statistics for input images of different sizes for neural networks trained with DMPs . . . 53
Table 2.5: Reconstruction statistics for input images of different sizes for neural networks trained with DMPs and normalized AL-DMPs . . . 54

Table 3.1: Reproduction errors resulting from the projection onto the latent spaces computed by AE, PCA, and AE with linear activation functions . . . 79
Table 3.2: Average throwing error and its variance when applying GPR in three different learning spaces. . . . 82

Abbreviations

AE	...	Autoencoder
AL-DMP	...	Arc Length Dynamic Movement Primitive
CIMEDNet	...	Convolutional Image to Motion Encoder-Decoder Network
CNN	...	Convolutional Neural Network
DMP	...	Dynamic Movement Primitive
GPR	...	Gaussian Process Regression
IMEDNet	...	Image to Motion Encoder-Decoder Network
PoWER	...	Policy learning by Weighting Exploration with the Returns
RL	...	Reinforcement Learning
VIMEDNet	...	Variable Image to Motion Encoder-Decoder Network

Symbols

a	... scaling parameter for scaling temporal course of motion
$\mathbf{b}_{\mathrm{AE}}$	... AE biases
$\mathbf{C}_j$	... j-th input image of width W and height H
c_k	... k-th center of Gaussian distributed along the phase of the trajectory or the path
$\mathbf{D}$	... training data with pairs of images and the associated handwriting paths
D_{q}	... dimension of the query point space
d	... distance of the target
d_{AE}	... autoencoder latent space dimension
d_{DMP}	... number of DMP parameters (DMP space dimension)
d_{PCA}	... PCA latent space dimension
d_{m}	... movement trajectory space dimension
d_{T}	... desired target distance
$\mathbf{E}$	... training data with the DMP parameters of the robot throwing trajectories joint motion
$\mathrm{E_a}(\cdot)$	... normalized AL-DMP path loss function
$\mathrm{E_t}(\cdot)$	... DMP trajectory loss function
$\mathrm{E_p^{AL}}(\cdot)$	... normalized AL-DMP parameters loss function
$\mathrm{E_p^{DMP}}(\cdot)$	... DMP parameters loss function
$\mathrm{E_J}(\cdot)$	... reproduction error of joint trajectory
$f(\cdot)$	... scaling function for scaling temporal course of motion
$\mathbf{F}(x)$	... phase-dependent DMP or AL-DMP nonlinear forcing term
$\mathbf{F}_{\mathrm{dec}}(\cdot)$	... decoder mapping function
$\mathbf{F}_{\mathrm{enc}}(\cdot)$	... encoder mapping function
$\mathbf{F}_{\mathrm{fk}}(\cdot)$	... robot forward kinematics
$\mathbf{G}_{\mathrm{AE}}(\cdot)$	... GPR transformation function from a desired target to an AE-based latent space
$\mathbf{G}_{\mathrm{DMP}}(\cdot)$	... GPR transformation function from a desired target to a DMP space
$\mathbf{G}_{\mathrm{PCA}}(\cdot)$	... GPR transformation function from a desired target to a PCA-based latent space
g	... gravitational acceleration
$\boldsymbol{g}$	... final position on trajectory or path (DMP or AL-DMP parameter)
$g()$	... Gaussian process function
h	... height of the target
$\mathbf{h}_k^{\mathrm{AE}}$	... output of the network's k-th layer
h_k	... width of the k-th Gaussian distributed along the phase of the trajectory or the path
$\mathrm{in}(n, i)$	... function that selects the trial with the i-th highest reward from the trial set
$\boldsymbol{J}_r$	... robot Jacobian at release configuration
$\mathbf{K}(\cdot,\cdot)$	... joint covariance matrices
$k(\cdot,\cdot)$	... covariance function of a Gaussian process

L	...	spatial length of the complete path traversed by the trajectory (AL-DMP parameter)
M	...	number of basis functions for speed encoding
$\mathbf{M}_j$	...	corresponding writing movements associated with the j-th image
$\mathbf{M}_j^{f(t)}$	...	writing movements as a temporal sequence of positions on the trajectory
$\mathbf{M}_j^{f(s)}$	...	writing movements sampled as a function of normalized arc-length $s_{i,j}$
$\mathbf{M}_j^{DMP}$	...	DMPs output data of the neural network
$\mathbf{M}_j^{AL}$	...	normalized AL-DMP output data of the neural network
m	...	number of best trials
$m()$	...	mean function of a Gaussian process
N	...	number of basis functions
n_{DOF}	...	number of robot degrees of freedom
P	...	number of training pairs
p_a	...	one of the normalized AL-DMP parameters
p_h	...	one of the DMP parameters
$\boldsymbol{p}(t)$	...	motion of a free-flying ball
$\boldsymbol{p}_r$	...	ball position at release time
$\dot{\boldsymbol{p}}_r$	...	ball velocity at release time
p_x^r	...	x coordinate of motion at release time
p_y^r	...	y coordinate of motion at release time
$\mathbf{Q}$	...	set of measured query points
$\mathbf{Q}^*$	...	set of new query points for prediction
$\boldsymbol{q}$	...	GPR query point - throw hit
$\boldsymbol{q}^*$	...	new target of the throw
$\boldsymbol{q}_{\mathrm{T}}$	...	target
$\boldsymbol{q}_{\mathrm{d}}$	...	desired target of the throw
$R()$	...	terminal reward function
R_k	...	terminal reward received at the end of the k-th rollout (trial)
$\mathbf{S}$	...	PCA covariance matrix
$s(t)$	...	spatial length or normalized spatial length of trajectory as a function of time
$\widetilde{s}(t)$	...	normalized arc length
T	...	duration of the complete movement
t	...	time
t_{end}	...	time when the robot stops moving
t_r	...	time at which the robot releases the ball
v_i	...	i-th AL-DMP weight for speed encoding
v_0	...	absolute velocity at release time
$\mathbf{W}_{\mathrm{PCA}}$	...	PCA matrix
$\mathbf{W}_{\mathrm{opt}}$	...	normalization weight matrix
$\mathbf{W}_{\mathrm{AE}}$	...	AE weights
$\boldsymbol{w}_k$	...	k-th DMP or AL-DMP weight
x	...	the common phase in DMP or AL-DMP framework
$\widetilde{x}$	...	normalized phase
$\boldsymbol{y}$	...	movement trajectory (robot joint angles or Cartesian space)
$\widetilde{\boldsymbol{y}}$	...	normalized path
$\boldsymbol{y}_0$	...	initial position on the trajectory or path (DMP or AL-DMP parameter)
$\boldsymbol{y}_0'$	...	starting spatial velocities (AL-DMP parameter)
$\widetilde{\boldsymbol{y}}_0'$	...	normalized starting spatial velocities (AL-DMP parameter)
$\boldsymbol{y}_r$	...	robot joint configuration at release time

$\dot{\boldsymbol{y}}_r$	...	joint velocity at release time
$\boldsymbol{y}_j^{\mathrm{AE/PCA}}$	...	robot joint trajectory obtained by integrating the DMP computed from the AE- or PCA-based latent space parameters
$\boldsymbol{y}^{\mathrm{AL-DMP}}$	...	path generated by the normalized AL-DMP
$\boldsymbol{y}^{\mathrm{DMP}}$	...	trajectory generated by the DMP
$\dot{y}_{1,2,3}^{\max}$	...	maximum allowed joint 1,2,3 velocity
$\boldsymbol{z}$	...	scaled velocity of motion $\boldsymbol{y}$
α	...	angle at which the ball should hit the target
α_r	...	angle of motion at release time
$\alpha_x, \alpha_z, \beta_z$	...	DMP or AL-DMP constants, chosen so that the resulting dynamical system is critically damped
$\boldsymbol{\varepsilon}_n$	...	zero mean Gaussian exploration noise
ϵ	...	observation noise
$\boldsymbol{\zeta}^\star$	...	autoencoder parameters (weights and biases of neurons in the AE network)
$\boldsymbol{\Theta}_n$	...	set of all executed policy parameters
$\boldsymbol{\theta}_n$	...	control policy parameters
$\boldsymbol{\theta}_n^*$	...	control policy parameters for the n-th rollout
$\boldsymbol{\theta}_{n+1}$	...	updated control policy parameters
θ_k	...	one of the parameters describing the robot motion (DMP or latent space parameter) in k-th observation
$\boldsymbol{\theta}_k^*$	...	executed control policy parameters
$\boldsymbol{\theta}^{\mathrm{AE}}$	...	autoencoder latent space parameters
$\boldsymbol{\theta}^{\mathrm{DMP}}$	...	DMP parameters
$\tilde{\boldsymbol{\theta}}^{\mathrm{DMP}}$	...	DMP parameters calculated by autoencoder or PCA
$\bar{\boldsymbol{\theta}}^{\mathrm{DMP}}$	...	mean of the PCA training data
$\boldsymbol{\theta}^{\mathrm{PCA}}$	...	PCA latent space parameters
$\boldsymbol{\vartheta}$	...	input values of Gaussian process by Gaussian process regressions
$\boldsymbol{\vartheta}^*$	...	output values of Gaussian process by Gaussian process regressions
$\bar{\boldsymbol{\vartheta}}^*$	...	estimated values of Gaussian process by Gaussian process regressions
$\boldsymbol{\lambda}$	...	PCA eigenvalues
$\boldsymbol{\Sigma}$	...	zero mean Gaussian exploration noise variance
σ_n, σ_f, l_i	...	hyperparameters of the Gaussian process
τ	...	time constant (DMP parameter)
$\boldsymbol{\tau}_i$	...	i-th robot throw
$\Psi_k(x)$	...	k-th DMP or AL-DMP forcing term
$\langle\cdot\rangle_{w(\boldsymbol{\tau})}$	...	importance sampling

Chapter 1

Introduction

The number of robot applications has been steadily increasing in recent years. Besides in standard industrial environments, robots are now used also in medical care and household environments. With the variety of robot applications, the complexity and the variety of tasks that a robot needs to perform are also increasing. In many applications there is a need to adapt to the wear and tear that occurs as robots perform these tasks. To be effective in such applications, robots need to be able to quickly adapt their behavior to a target task, even if the current configuration of the task is different from previously known task configurations. Many of the new applications therefore cannot be programmed in a standard way. Thus, there is an increasing need for robots with learning capabilities.

Without learning and adaptation, the operation of robots in natural environments is not possible. Learning allows a robot to adapt its behavior with respect to changing tasks and environments and even act in situations that were not considered by a robot programmer. Robot learning is based on methods from machine learning, which is a part of artificial intelligence. In this thesis, we focus on deep neural network architectures for robot skill learning and learning of feature representations, methods to train deep neural networks, and the application of features computed by neural networks to improve robot skill learning, including imitation, reinforcement learning and statistical learning.

Imitation learning works in two steps. In the first step, a robot is provided with one or multiple demonstrations of the task. In the second step, the acquired data are used to compute new task parameters, which often includes generalization to new task situations. Good generalization is only possible if a sufficient amount of task demonstrations is available [1].

Success and performance of machine learning methods depends heavily on the selected feature representation. When feature extraction is done by hand, a skilled and experienced user is required. The goal of feature learning is to automatically compute optimal features from the data [2]. One approach to learning a complex robot task is to use elemental behaviors as building blocks for more complex behaviors. Learning algorithms can then focus on the selection, adaption, sequencing and co-activation of building blocks and modules [3].

Reinforcement learning is an unsupervised learning method, where a robot autonomously discovers optimal behaviors through trial-and-error interaction with an environment. Robot in a given state $\mathbf{s}$ uses policy π and chooses action $\mathbf{a}$. The application of the selected action results in a state change. The resulting behavior is evaluated with a reward function. Robots use reinforcement learning to compute optimal policies so that actions that result in the highest reward are selected in each state [4].

Supervised learning uses labeled data. For each input data point, there is a defined proper output value. The goal of learning is to find a model that computes outputs from

inputs with the least error. Examples of supervised learning are regression and classification problems.

All the above-mentioned learning methods had expanded their capabilities by incorporating methods from the relatively new field of deep learning, which has recently had a big impact in the field of artificial intelligence.

Artificial intelligence is a broad and evolving field. In the early days of artificial intelligence, the focus was on problems that can be formally described and are intellectually challenging for a human (e.g. chess playing). However, it turns out that problems that people solve intuitively and are hard to formalize (action understanding, natural language understanding) are more challenging. A good approach for such problems is to let machines learn from their experience and organize their knowledge in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. A graphical representation of such knowledge consists of many hierarchical layers, thus the name deep learning was coined.

With the increased computational power, deep learning methods that directly model the relationship between the available sensory input and the desired output vectors have become more popular. Deep learning methods have achieved major success in many different application areas [5], e. g. natural language processing and visual classification.

In the field of natural language processing, deep learning methods are successful at part-of-speech tagging, chunking, named entity recognition, and semantic role labeling [6]. Deep learning has been successfully applied for audio recognition. Feature representations learned from unlabeled data have shown good performance in multiple audio classification tasks [7] and can be used for acoustic modeling [8].

In face recognition, it is possible to train face detectors with unlabeled data [9]. Deep learning also achieves excellent performance in object recognition benchmarks [10]. Due to deep learning, visual object recognition has already reached the stage where computers are superior to humans at some tasks. These advances were enabled by the development of suitable deep neural network architectures, such as deep convolutional neural networks (CNNs) [11], where connectivity between neurons mimics the organization of neurons in human visual cortex [12]. This leads to fewer but properly organized parameters that can be trained more easily while retaining the expressive power for visual recognition tasks. The possibility to utilize GPUs to speed up the training of deep neural network models also made an important contribution.

When deep learning techniques are applied to either the model-free (be it value-based or policy-based) or model-based reinforcement learning approaches, the resulting approaches are classified as Deep Reinforcement Learning [13]. Deep reinforcement learning gained popularity after it was able to outperform human experts in Atari games [14] and the game of Go [15]. Due to the success of deep reinforcement learning in these areas, it is natural to apply these approaches also to robotic problems.

Robotics is a broad field with many specific issues that can be approached by deep learning. Deep learning can be used as a form of robotic world interpretation to generate a model of rigid body motion from raw point cloud data [16], where the generated model predicts changes to the environment based on specific actions. Robotic grasping is another complex problem involving perception, planning and control, which can all be addressed by deep learning algorithms. For example, deep learning can be applied to find optimal grasp on the basis of human generated data [17] or with trial-and-error experiments with the robot [18]. Traditional policy search methods often require hand-engineered components for perception, state estimation and low-level control. Levine et al. [19] developed a method to train all these components jointly and to map image observations directly to robot joint torques. The resulting policies are represented by a deep convolutional neural

network trained with a guided policy search method that converts policy search into supervised learning. With deep reinforcement learning, it is also possible to learn hand-eye coordination for robot grasping directly on real robots. Levine et al. [20] used a large-scale data collection on real robots to train a deep neural network representing the value function for robot grasping. However, large-scale data collection on real robots is expensive and potentially dangerous. This can be reduced by learning in simulation before transferring the learning process to the real robot, as was done by James et al. [21] for Deep Q-Learning and Pinto et al. [22] for the Asymmetric Actor Critic method. The simulation environment is usually different from the real world, and the real world environment can change constantly due to the unpredictable conditions in the real world. If the learned control policy is not robust enough, it cannot handle these changes. Instead of learning a robust deep neural network policy that is invariant to all possible environment changes, it is often better that the policy continues to adapt to the current new environment during deployment, as in [23] and [24].

Besides end-to-end deep learning, the ability of deep learning to deal with complex, high-dimensional world representations can be exploited to extend classical learning methods, e. g. to generate simpler world representations. Finn et al. [25] substitute difficult manual feature selection from visual input with spatial autoencoders that can learn a proper state representation directly from images. Chen et al. [26] developed an autoencoder that learns a representation of high-dimensional sensory data (vision or tactile), which is used as feedback for control. In the second phase, an optimal controller is found by reinforcement learning using low dimensional representation learned by the autoencoder.

In this thesis, we apply deep learning in two different learning frameworks. First we propose a deep encoder-decoder network to learn direct perception-action couplings between image input and motion output. In the second part we use a deep autoencoder network for learning feature representations, where the learned feature representation is used to improve the performance of robot skill learning.

The first theme of the thesis is how to apply deep neural networks to learn perception-action couplings. The universal approximation theorem [27] for neural networks shows that deep neural networks provide the functionality needed to learn highly nonlinear mappings between perception and action. Such mappings are the key to the development of autonomous robots that need to operate in real unstructured environments. By coupling perception and action, a robot can form higher-level concepts such as object-action complexes [28], which have been proposed to bind objects, actions, and the accompanying attributes in a causal model. Together with suitable representations, deep neural networks can contribute to the development of a new generation of cognitive robots.

The training of deep neural networks for robotic applications often leads to issues that have not yet been fully addressed by the machine learning community [29]. There is a need for better evaluation metrics which – when combined with appropriate representations – lead to a more effective training process. The aim of many current research papers in deep learning for robotics is to design powerful end-to-end learning processes that can map data from images derived from the perceptual systems of robots to their control inputs [30]. However, due to the large amount of data needed for this type of learning, end-to-end learning in robotics is often limited to simulation environments [31], [32] and the proposed approaches do not scale to real robots.

In our work we first investigate an effective evaluation metric (loss function) for one particular application domain: training of deep neural networks with dynamic movement primitives (DMPs) [33], [34] at the output. The proposed metric takes into account the physical distance between movements instead of measuring the distance between the parameters of DMPs, which have no physical meaning.

When directly reproducing motion from raw digital images, we often encounter the problem that a single image does not necessarily contain information about the time course of the desired movement. This problem can sometimes be addressed if the rest of the training data contain information about time. For example, if the training data consist of images of digits and the associated writing trajectories, the correct timing of movement can be reproduced because the writing trajectories associated with images contain the required information. However, this is not always the case and it is sometimes necessary to work without having information about the temporal component of motion. We have therefore extended our approach from standard dynamic movement primitives [34], which implicitly contain the information about the temporal course of motion, to arc-length dynamic movement primitives (AL-DMPs) [35], [36], which effectively separate the spatial and temporal component of motion and can therefore be used even when information about the timing of motion is not available. For this dissertation we did not use the exact original form of the AL-DMPs, but we adapted their definition to create a new version called normalized AL-DMPs. The new version allows a more efficient training of neural networks than the original AL-DMP formulation. Both DMPs and AL-DMPs have been extensively used in the scope of learning by demonstration. Other representations have also been utilized, e. g., polynomial splines, radial basis functions, Gaussian Mixture Models, probabilistic movement primitives, etc. See [37] for a review. However, DMPs and AL-DMPs have many advantages because they can be modulated and are robust to perturbations.

Another topic that we deal with in this part of the thesis is how to provide input images of different sizes to the developed neural network. The ability to use an arbitrary size of input data for a neural network does not only resolve the scaling and changing aspect ratio issues, but also opens the possibility to train a neural network faster on smaller input images and then use it to process bigger images. An example would be the learning of neural networks for processing images with cluttered, irrelevant background. The neural networks learn to suppress background information on smaller examples of input images and then use this knowledge when processing real input images at full size.

The second main topic of the thesis is how to improve robot skill learning by using low dimensional features computed by autoencoder neural networks. Learning complete actions and/or skills from scratch is in most cases not feasible because the search space is too large [38]. 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 [39]. However, unless the robot can generalize from the available human demonstrations, it is unlikely that the accumulated knowledge is directly applicable in all possible states of the real world [40]. Skill transfer from a human might also not achieve the same outcome when performed by a robot due to the correspondence problem [41], 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 should 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 the optimal skill instantiation for the current task description can be computed by statistical generalization of skill executions stored in the database [42]. If statistical generalization does not result in an appropriate action to fulfil the desired task, then the computed skill 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 [43]. 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 [44]. The use of parameterized policies, policy search methodologies [45] 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. The aforementioned dynamic movement primitive (DMP) representation [46] provides a low-dimensional representation of the action space. However, the dimensionality of the DMP parameter space is still rather high [40]. 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 [47]. Dimensionality reduction and regression have been combined in Locally Weighted Projection Regression (LWPR) [48], an algorithm that achieves nonlinear function approximation in high dimensional spaces even in the presence of redundant and irrelevant input dimensions. Another common method is the use of (deep) autoencoders [49], where the data is pushed through the layer with the smallest number of neurons – the latent space.

In this thesis, 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 is 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.

By acting in the real world and accumulating new knowledge, a robot can gradually improve its performance [50], which is an important step towards achieving the dream of lifelong robot learning [51]. We therefore also take under consideration another learning aspect, which is 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. In the thesis we evaluate if an autoencoder trained on a database of simulated data can be used to compute the latent space of real robot actions.

1.1 Review of the Relevant Scientific Field

Other approaches to utilize deep neural networks for learning of DMPs have been reported in the past. For example, Chen et al. used autoencoders [52] and variational autoencoders [53] to reduce the dimensionality of the movements obtained by human demonstration and effectively train dynamic movement primitives in a low-dimensional latent space. Pervez et al. [54] used 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 et al. [55] the authors used hierarchical deep reinforcement learning to optimize the forcing term of a DMP for demonstrated trajectories. In the field of imitation learning, Zhou et al. [56] proposed to apply mixture density network for the mapping from the task parameter query to the parameters describing the movement primitives distribution.

Many of the above-mentioned works are based on dimensionality reduction methods originally proposed by the computer vision community. This research started with the development of deep autoencoders, which were shown to be effective at computing low dimensional latent spaces from raw character images [57]. The MNIST training set [58], which consists of handwritten digits, is often used for testing the proposed networks and algorithms. To improve the previously reported results on image reproduction, Gregor et al. [59] proposed a recurrent neural network architecture called DRAW, which consists of a pair of recurrent neural networks: an encoder network that compresses the real images

presented during training, and a decoder network that reconstitutes images after receiving the compressed codes. Encoder-decoder networks in combination with convolutional layers have proven to be useful also in computer vision. A well-known example is SegNet [60], in which pre-trained convolutional layers from a convolutional neural network (CNN) were adapted to form a fully-convolutional encoder-decoder architecture for semantic pixel-wise segmentation. While these research results are not directly applicable to the problem of mapping images to robotic movements, they inform about the possible neural network architectures that can be used for this purpose.

Usually, when CNNs are used for supervised learning of perception-action couplings, they are used in combination with another neural network in two separately trainable parts. For example, in Yang et al. [61], a deep convolutional autoencoder is first used for finding camera image features and then, in combination with the recorded joint angles, sequences are formed for the learning of task dynamics with a time delay neural network. This approach produces the next step from the image of the current step while working in an online loop, whereas in our proposed method, by contrast, we use single images for generating entire trajectories.

To evaluate the proposed metrics and network architectures, we apply them to the problem of mapping raw images of digits to the robot handwriting trajectories. Although the focus of the thesis is not on the development of an optimal approach for robotic writing, we mention here some related approaches that aim at inferring writing movements from images. Ali [62] models individual brush strokes of calligraphic characters using Gaussian Mixture Models, combining the brush strokes using Gaussian Mixture Regression and reproducing brush stroke trajectories using DMPs. The reproduced stroke trajectories were iteratively refined using reinforcement learning for learning examples in the database, but each reproduction started from scratch with no generalization between different examples or to new unseen cases. A database approach to learning to draw characters was recently presented also by Kotani and Tellex [63]. Unlike our proposed method that focuses on end-to-end learning of DMPs that describe digits, this approach is based on inferring a sequence of commands for writing a character, where each command is represented as a shift in the x and y directions and a Boolean value which determines whether or not the pen makes contact with the table surface.

One of the options for using arbitrary size input data is spatial pyramid pooling as proposed by He et al. [64]. They used a spatial pyramid pooling layer to convert the last convolutional layer to a fixed-length vector, which was then used as input to the fully connected part for classification. Another possibility is to use the region of interest pooling as in [65]. The convolutional neural network encodes the input image of arbitrary size in the feature map. From the feature map, regions of different positions, sizes, and ratios are sampled. In each region, max-pooling is used to create a vector of fixed size by quantizing the regions to the same number of subregions and using max-pooling for each region. Another approach is to apply a global average pooling layer [66], which takes averages over each feature map in the last convolutional layer to be used as inputs for each class in the final softmax layer. In contrast to this work, we propose a global maximal pooling layer, which takes the maximum value from each feature map in the last convolutional layer instead of the average value. We use the resulting vector, which forms the latent layer in our architecture, as input for the subsequent decoder layers that regress on the final output values representing the digit trajectory shape. Not only that our approach deals with a regression problem as opposed to classification, but the use of a global maximal pooling layer as opposed to an average pooling layer in this way allows the network to more easily filter out background noise and focus on the most relevant local features. In addition, although the models described in [66] could be trained with differently sized inputs, our

proposed architecture is capable of being trained with inputs of a particular size and then generalizing to use arbitrarily sized inputs at the execution time.

DMPs are often used as a motor representation in reinforcement learning (RL) [43], [44]. The dimensionality of learning DMP parameters in combination with tactile and visual feedback has been deemed too large in some practical applications [67], prompting RL in latent space of actuator redundancies. Latent spaces defined by autoencoders were applied for this purpose [67], [68]. Deep autoencoders and variational autoencoders have also been used to train movement primitives in a low-dimensional latent space [69], [70]. RL in the latent space of a deep autoencoder network greatly reduces the dimensionality. However, depending on the size of the latent space, it can also reduce the accuracy of the representation [70]. Another way to reduce the dimensionality of the search space is by constraining the parameter space with statistical generalization [40], [71]. 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 [72], [73].

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) [74] and Locally Weighted Regression (LWR) [42], [75] have been applied for this purpose. The application of GPR was extended to compliant DMPs [76], Cartesian DMPs [77], arc-length dynamic movement primitives (AL-DMPs) [78], and for the autonomous generation of the training database [50].

Task parameters were also applied for learning parameterized skills [79], where the authors propose to adapt the DMP weights of a single demonstration based on the task parameter. Parameterized skill memories [80] enable task-specific generalization from a low number of examples and are based on DMPs. Methods based on other trajectory representations were used as well. For example, the Mixture of Motor Primitives (MoMP) [81] 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. The approach of Calinon [82] is centered around task-specific Gaussian Mixture Models (TP-GMM). An extensive list of papers on learning of task-parameterized movements is provided in [82].

Statistical learning in latent spaces was analyzed in several papers. Zuo et al. [83] 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. [84] demonstrated that supervised dimensionality reduction architectures can provide improved generalization performance, which is also the key feature of our approach. Yoo et al. [85] reported an approach where GPR is applied in the latent space defined by VAE. However, generalization in [85] was applied to visual data, whereas we concentrate on latent space generalization to synthesize new robot trajectories.

1.2 Thesis Purpose, Goals and Hypothesis

The purpose of this thesis is to advance the field of robot learning with the application of deep neural networks. In particular, we focus on the development of new approaches to learning perception-action couplings with deep neural networks and on improving robot skill learning in latent spaces computed by deep autoencoder neural networks. To achieve these advances we have set six goals listed below, which represent our scientific contributions to this field of science. For each goal, we present a scientific argumentation and corresponding hypotheses.

For easier references in the thesis, we have listed each of our goals with the capital letter **G** and the corresponding number. Similarly, we mark each corresponding hypothesis with

the capital letter **H** and a number that corresponds to the goal number.

- **G1: Develop a suitable deep neural network architecture capable of generating a DMP-encoded motion trajectory at the neural network output from an image at the neural network input.**

  For the autonomous operation of robots, it is necessary to connect the robot's perceptual system with its actions. Robot actions usually involve motion defined as robot trajectories, hence it is advantageous if it is possible to compute robot trajectories directly from the perceptual input. One of the most informative senses is robot vision, but its complexity requires highly nonlinear mappings in order to connect vision with motor representations.

  Due to their high representational power, deep neural networks (DNNs) are highly successful in solving vision tasks such as for example object recognition. DNNs have recently been gaining traction also in robotics. They are suitable to directly connect perception, e. g. robot vision, to action, e. g. motion trajectories. Since motion trajectories depend on time, the number of samples that can be gathered from different trajectories vary with the duration of motion. It is therefore not possible to directly output a complete trajectory as a discrete sequence of samples if feedforward networks are used because the size of output layer in a neural network is fixed. On the other hand, we can use DMP parameter values as output as DMPs enable the generation of temporal trajectories with different duration.

  H1: A deep encoder-decoder network is able to learn perception-action couplings with images at the input and DMPs at the output.

- **G2: Improve the training of the proposed deep neural network architecture by developing a new loss function for calculating the physical error between the motion trajectories at the network output.**

  A loss function defined as the mean square error between the predicted and desired DMP parameters has no physical meaning. A better way is to calculate the distance between the actual predicted and desired trajectories in physical space, e. g. Cartesian or joint space. However, the training of neural networks using backpropagation algorithm requires the calculation of gradients of the loss function. This is not so trivial because a DMP is defined by a second-order dynamical system. The derivation of gradients for such loss functions has not yet been reported in the literature. By deriving the proper equations for the gradients of proposed loss function, we enable the use of a loss function with physical meaning, which should improve the prediction performance of neural networks having DMPs as output.

  H2: The application of a loss function that measures the physical error between motion trajectories gives better results than the loss function that calculates the DMP parameter error.

- **G3: Extend the proposed method including the loss function to networks that have AL-DMPs at the neural network output instead of standard DMPs.**

  AL-DMPs separate the spatial and temporal parts of the trajectory, which provides the ability to learn both parts separately in the proposed learning framework. This enables us to separate or prioritize the learning of spatial or temporal part of the trajectory in cases when the available data do not support the joint learning of spatial and temporal components of motion.

H3: Extending the same learning framework to AL-DMPs makes it possible to learn handwriting movements when information about the temporal evolution of handwriting motion is not available.

- **G4: Develop a neural network architecture capable of using input images of variable sizes and containing cluttered backgrounds.**

  Neural network architectures with the ability to use input images of variable size and containing cluttered backgrounds are essential components of an end-to-end learning system based on neural networks. They enable us to avoid applying classic computer vision algorithms to pre-process input images. Most existing solutions in the literature rely on input images of fixed sizes and are usually intended for classification applications.

  H4: We can extend the proposed deep neural network architectures to be applied to variable size input images with cluttered background.

- **G5: Improve the robot skill learning by projecting DMPs to low-dimensional latent spaces computed by autoencoder neural networks.**

  Robot learning algorithms such as reinforcement learning and statistical learning can be applied to adapt robot skills to the changes in the environment and other task changes. However, these algorithms suffer from the curse of dimensionality when applied to robots with many degrees of freedom and continuous states and actions. It is therefore often necessary to apply dimensionality reduction methods to make robot learning problems computationally tractable. We propose to apply deep autoencoder neural networks to reduce dimensionality when learning DMP parameters. To evaluate the effectiveness of the proposed methodology, we compare the proposed autoencoder-based latent space with the initial DMP parameter space and the PCA-based latent space.

  H5: Robot skill learning in the autoencoder-based latent space performs better than in the DMP parameter space or in the PCA-based latent space.

- **G6: Evaluate the possibility of using the autoencoder trained on the simulated dataset for learning skills with a real robot.**

  The training of autoencoders requires a relatively large training dataset. It is, however, time-consuming to gather large amounts of training data with a real robot. In addition, the risk of damaging the robot is high. These issues can be avoided by making use of simulation data for training. But since simulation does not describe the real world perfectly, the usability of models trained in simulation cannot be guaranteed for real-world applications. In this thesis, we show that dimensionality reduction through autoencoders trained in simulation is effective also for learning on a real robot.

  H6: We can use latent spaces computed by autoencoders trained on simulated data for learning skills with a real robot.

1.3 Thesis Outline

Here we outline the content of each chapter. The detailed structure of the chapters is presented at the beginning of each chapter.

In this chapter (Chapter 1), we have introduced the field of research, the research problems and provided an overview of the relevant scientific field (Sec. 1.1). We presented the purpose of the thesis with research goals and corresponding hypotheses (Sec. 1.2).

The first part of the work is presented in Chapter 2. This part deals with the use of a deep encoder-decoder network to directly learn perceptual-action couplings between an image input and a motion output. In this chapter, we address the first four goals of the thesis. We introduce the approach to generate a DMP-encoded motion trajectory at the output of the neural network from an image at the input of the neural network (**G1**) and a new DMP trajectory loss function together with its gradients to better train the neural network (**G2**). We extend the developed methods from the first two goals to a normalized AL-DMP path representation by developing a normalized AL-DMP path loss function and deriving its gradients (**G3**). To extend the usability of neural networks that directly couple image inputs and motion outputs, we have also extended the input capabilities of the proposed neural networks. We describe a new neural network architecture, which is able to use inputs of different sizes and can deal with complex backgrounds (**G4**). At the end of the chapter, we evaluate all four goal-correlated hypotheses in different experiments to reproduce handwritten images from a single image.

The second part of the thesis is presented in Chapter 3 where we address the remaining two goals of the thesis. This part deals with how to improve robot skill learning by using features computed by the autoencoder neural network (**G5**). The proposed approach is evaluated by experiments in which a robot learns how to throw a ball precisely at a target, either by reinforcement learning or by statistical generalization. Since the dataset for training of deep autoencoders is created in simulation, we have also addressed goal **G6** with the same experiments.

In Chapter 4, we discuss the results of the experiments and possible future work for both parts of the thesis. Based on the achieved results, we evaluate the acceptance of our hypotheses and the fulfillment of the set goals.

Chapter 2

Deep Neural Networks for the Generation of Trajectories/Paths from Images

This chapter describes the use of a deep encoder-decoder network to learn direct perception-action couplings between an image input and a motion output. As a motion output, we used two different motor representations (DMP and normalized AL-DMP) as defined in our goals **G1** and **G3**. As defined in **G2**, we introduced a new, better loss function that measures the actual physical difference between the resulting motion for both motor representations and derived equations of its gradient, which is necessary for learning with backpropagation. We have published the proposal for a new, better DMP loss function in [86] and [87], while the path reproduction with normalized AL-DMP has been published in [88]. We have also extended the input capabilities of our approach by designing a deep neural network capable of using inputs of different sizes and dealt with cluttered backgrounds according to our **G4** goal. This was part of the results reported in [88].

First we describe the methodology. To compute the motion output, we used two different motor representations (DMPs and normalized AL-DMPs), which are presented in Section 2.1. In the same section, we also introduce the original AL-DMPs, which are the basis for normalized AL-DMPs. We describe how the normalized AL-DMPs are estimated and executed. In Section 2.2, we specify the data that were used to train neural networks mapping image inputs to motion outputs. In Section 2.3, we introduce the new loss functions and derive the gradients of loss functions based on the integration of DMPs (Section 2.3.1) and normalized AL-DMPs (Section 2.3.2). The applied neural network architectures are presented in Section 2.4. We introduce the fully connected encoder-decoder neural network (Section 2.4.1), the encoder-decoder network with convolutional layers (Section 2.4.2), and our new convolutional architecture capable of using variable size input images (Section 2.4.3).

The methodology is followed by the experimental results in Section 2.5. As a task, we chose the reproduction of handwritten numbers from single images. First, we define the datasets used for training and testing neural networks in our experiments (Section 2.5.1), explain the training procedure (Section 2.5.2) and the evaluation of experimental results (Section 2.5.3). We have conducted a total of five different types of experiments. The first experiment in Section 2.5.4 served to select suitable neural network architectures and training procedures. In the second experiment (Section 2.5.5), we tested DMP-based loss functions and then compared them with loss functions based on normalized AL-DMPs in Section 2.5.6. In Section 2.5.7, we used all of the investigated loss functions in experiments with input images of different sizes. Finally, we also performed real robot experiments,

which are described in Section 2.5.8.

2.1 Trajectory/Path Representations

As it is standard in robotics, in the thesis we refer to a path if we are given a set of points that lead us from the starting point to the final point without specifying how quickly the robot should move. On the other hand, we refer to a trajectory if we are given a path together with a temporal schedule for how to get from the initial to the final point.

The sampling of human or robot motion is usually done in constant time steps, which means that the number of data points representing the trajectory depends on the duration of motion. Neural networks have a fixed size output. If we want to train feedforward neural network architectures with trajectories of different duration at the output, we have to encode the trajectories of variable duration into the constant number of parameters needed for fixed-size output of feedforward neural network.

The first trajectory representation that we used for encoding of trajectories into the fixed number of parameters is DMP. The next suitable candidate is AL-DMP. Its advantage is that it first generates the path from the trajectory by separating spatial and temporal aspects of motion and thus encodes spatial and temporal parts separately. We did not use classical AL-DMPs in our experiments. Instead we proposed a version called normalized AL-DMP which is better suited for training neural networks. We present the estimation of normalized AL-DMP parameters and the execution of robot motion specified by normalized AL-DMP.

2.1.1 Dynamic movement primitives

Let us denote a time-dependent movement trajectory as $\boldsymbol{y}(t) \in \mathbb{R}^{d_\mathrm{m}}$, where d_m specifies the dimension of space that contains the desired movement trajectory (robot joint angles of Cartesian space). A *dynamic movement primitive* (DMP) [34] specifying such a movement is defined by the following system of differential equations

$$\tau \dot{\boldsymbol{z}} = \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - \boldsymbol{z}) + \mathrm{diag}(\boldsymbol{g} - \boldsymbol{y}_0)\ \mathbf{F}(x), \tag{2.1}$$

$$\tau \dot{\boldsymbol{y}} = \boldsymbol{z}, \tag{2.2}$$

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

$$\tau \dot{x} = -\alpha_x x. \tag{2.3}$$

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 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 (2.1) – (2.2) 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

$$\mathbf{F}(x) = \frac{\sum_{k=1}^{N} \boldsymbol{w}_k \Psi_k(x)}{\sum_{k=1}^{N} \Psi_k(x)} x, \quad (2.4)$$

$$\Psi_k(x) = \exp\left(-h_k (x - c_k)^2\right), \quad (2.5)$$

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 (2.1) – (2.2) 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 $\boldsymbol{w}_k \in \mathbb{R}^{d_\mathrm{m}}$ using regression techniques. See [42] 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 output the other parameters of the differential equation system (2.1) – (2.3) to fully specify a DMP:

$$\{\boldsymbol{w}_k\}_{k=1}^{N},\ \tau,\ \boldsymbol{g},\ \boldsymbol{y}_0. \quad (2.6)$$

2.1.2 Arc-length dynamic movement primitives (AL-DMPs

The DMP movement representation introduced above already has many advantages for robot trajectory generation and control as DMPs can be learned, modulated, and are robust against perturbations. They can also be used to compare trajectories to each other. However, due to the implicit dependency on time because of time derivatives in Eqs. (2.1) – (2.3), they cannot the separate spatial and temporal aspect of motion. For this reason, Gašpar et al. [36] extended the DMP movement representation with arc-length parameterization of the trajectory and developed a new movement representation called arc-length dynamic movement primitive (AL-DMP). For a time-parameterized trajectory $\mathbf{y}(t)$, the spatial length of the trajectory (also called arc-length) up to time t is defined as

$$s(t) = \int_0^t \|\dot{\mathbf{y}}(u)\| du. \quad (2.7)$$

Given the duration T of the complete movement, the spatial length L of the complete path traversed by the trajectory is given by

$$L = s(T) = \int_0^T \|\dot{\mathbf{y}}(u)\| du, \quad (2.8)$$

The speed of movement is related to arc-length and can be calculated as the time derivative of s

$$\dot{s}(t) = \|\dot{\mathbf{y}}(t)\|. \quad (2.9)$$

Parameter s of any valid re-parameterization $\boldsymbol{y}(s(t))$ of time-dependent trajectory $\boldsymbol{y}(t)$ needs to be strictly increasing, i.e. $\dot{s}(t) > 0, \forall t$ [89]. Otherwise it is not possible to compute derivatives of the path $\boldsymbol{y}(s)$ with respect to parameter s. According to Eq. (2.9), the derivative of speed always fulfils condition $\dot{s}(t) \geq 0$. If there exist times t where the speed of motion equals zero, the trajectory needs to be subdivided into segments of non-zero speed, except possibly at the end of the trajectory. In this case, the first and second derivatives of $\boldsymbol{y}(s)$ with respect to arc-length parameter s at both ends of each trajectory segment ought to be computed as left and right derivatives, respectively. This is possible if the path $\boldsymbol{y}(s)$ is piecewise C^2 continuous.

The reparametrization of the trajectory with the arc-length parameter leads to a time-independent parametrization of the path traversed by the trajectory. To obtain arc-length dynamic movement primitives, we rewrite Eqs. (2.10) – (2.12) by computing the derivatives with respect to arc-length parameter instead of time. The following dynamical system is obtained:

$$L\boldsymbol{z}' = \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}) - \boldsymbol{z}) + \mathbf{F}(x), \tag{2.10}$$

$$L\boldsymbol{y}' = \boldsymbol{z}, \tag{2.11}$$

$$Lx' = -\alpha_x x. \tag{2.12}$$

Here, all derivatives are taken with respect to arc-length instead of time. Such derivatives are denoted by $'$. The time constant τ, the function of which in standard DMPs is to speed-up or slow-down the movement during its execution by the robot, is replaced by arc length $L > 0$, which can be calculated by integrating (2.8).

Since robots are controlled at constant time steps, the information about time needs to be added to AL-DMPs to fully specify a motion trajectory. This can be accomplished by specifying the speed of motion, that is by approximating the derivative of arc-length $\dot{s}$. For this purpose we write the speed of motion as a linear combination of radial basis functions along phase x

$$\dot{s}(x) = 1 + \frac{\sum_{i=1}^{M} v_i \Psi_i(x)}{\sum_{i=1}^{M} \Psi_i(x} x. \tag{2.13}$$

Since speed is encoded separately, it is not necessary that the number of basis functions M for the approximation of speed is the same as the number of basis functions N in the forcing term (2.4).

Just like in standard DMPs, parameters α_z, β_z, and α_x are usually also taken as constant in AL-DMPs and do not change when encoding different movements. Thus, to fully specify an AL-DMP, a neural network needs to output the following parameters

$$\{\boldsymbol{w}_k\}_{k=1}^{N},\ L,\ \boldsymbol{g},\ \boldsymbol{y}_0,\ \boldsymbol{y}_0' \tag{2.14}$$

If the speed also needs to be learned, then the network needs to output an additional set of weights $\{v_i\}_{i=1}^{M}$ for speed encoding. These parameters do not influence the spatial shape of the trajectory.

A network that outputs AL-DMPs instead of standard DMPs thus has a similar set of neurons at the output layer. If the speed of motion is not estimated, the only additional parameters compared to standard DMPs are the starting spatial derivatives $\boldsymbol{y}_0'$. The reason for this is that unlike in standard DMPs, where initial and final velocities are assumed to be zero, we cannot assume that the derivatives of $\boldsymbol{y}$ with respect to arc length would be equal to zero. This is because by definition, these derivatives have a unit norm, i.e. $\|\boldsymbol{y}'(s)\| = 1$. This can be easily proven by computing the derivative of $\boldsymbol{y}(s(t))$ with respect to t and taking into account Eq. (2.9)

$$\left\| \frac{d\boldsymbol{y}}{ds} \right\| = \left\| \frac{d\boldsymbol{y}/dt}{ds/dt} \right\| = \frac{\|\dot{\boldsymbol{y}}\|}{\|\dot{\boldsymbol{y}}\|} = 1.$$

2.1.3 Normalized arc-length dynamic movement primitives

Note that among $\{\boldsymbol{w}_k\}_{k=1}^{N}$, $\boldsymbol{g}$, L, $\boldsymbol{y}_0$, and $\boldsymbol{y}_0'$, L is the only parameter in the differential equation system (2.10) – (2.12) that affects all dimensions of the control variable $\boldsymbol{y}$. We

therefore reformulate system (2.10) – (2.12) to avoid using L. This can be achieved by re-parameterization with normalized arc length, i.e. $\widetilde{s} = s/L$, $ds/d\widetilde{s} = L$. Let us define $\widetilde{\boldsymbol{y}}(\widetilde{s}) = \boldsymbol{y}(\widetilde{s}L) = \boldsymbol{y}(s), \widetilde{\boldsymbol{z}}(\widetilde{s}) = \boldsymbol{z}(s)$, and $\widetilde{x}(\widetilde{s}) = x(s)$. This re-parameterization changes the spatial derivatives as follows:

$$\widetilde{\boldsymbol{y}}' = \frac{d\widetilde{\boldsymbol{y}}}{d\widetilde{s}} = \frac{d\boldsymbol{y}}{ds}\frac{ds}{d\widetilde{s}} = L\boldsymbol{y}'. \tag{2.15}$$

Similarly we obtain

$$\widetilde{\boldsymbol{z}}' = L\boldsymbol{z}, \ \widetilde{x}' = Lx'. \tag{2.16}$$

We can now rewrite the differential equation system (2.10) – (2.12)

$$\widetilde{\boldsymbol{z}}' = \alpha_z(\beta_z(\boldsymbol{g} - \widetilde{\boldsymbol{y}}) - \widetilde{\boldsymbol{z}}) + \mathbf{F}(\widetilde{x}), \tag{2.17}$$

$$\widetilde{\boldsymbol{y}}' = \widetilde{\boldsymbol{z}}, \tag{2.18}$$

$$\widetilde{x}' = -\alpha_x \widetilde{x}. \tag{2.19}$$

Note that the above equation system does not contain L and a neural network needs to output only the following parameters

$$\{\boldsymbol{w}_k\}_{k=1}^N, \boldsymbol{g}, \ \boldsymbol{y}_0, \ \boldsymbol{y}_0', \tag{2.20}$$

to fully specify a normalized AL-DMP. By excluding L with normalization, some parameters change value but the encoded spatial trajectory remains intact. At the same time, as we will see later in the dissertation, its exclusion leads to simplified calculations of gradients of criterion functions and greatly simplifies and consequently accelerates the training of the neural networks. The consequence of normalization is also that the norm of velocity with respect to arc length is no longer a unity vector, but has the norm equal to the arc length L

$$\|\widetilde{\boldsymbol{y}}'\| = \left\|\frac{d\widetilde{\boldsymbol{y}}}{d\widetilde{s}}\right\| = \left\|\frac{d\boldsymbol{y}}{ds}\frac{ds}{d\widetilde{s}}\right\| = L\left\|\boldsymbol{y}'\right\| = L. \tag{2.21}$$

For the sake of clarity, in the rest of this dissertation we omit the modifier $\widetilde{\ }$ when referring to the differential equation system (2.17) – (2.19) and its parameters. We call the resulting representation *normalized arc length dynamic movement primitive.*

2.1.4 Estimation of normalized AL-DMPs

Given a sequence of points $\{\boldsymbol{y}_i\}_{i=0}^n$, $\boldsymbol{y}_i \neq \boldsymbol{y}_{i-1}\ \forall i$, on the robot path, we estimate the arc length L as follows

$$L = \sum_{i=1}^{n} \|\boldsymbol{y}_i - \boldsymbol{y}_{i-1}\|. \tag{2.22}$$

We also compute the normalized arc length parameters s_i

$$s_i = s_{i-1} + \frac{1}{L}\|\boldsymbol{y}_i - \boldsymbol{y}_{i-1}\|, i = 1, \ldots, n, \ s_0 = 0. \tag{2.23}$$

Given the fact that according to Eq. (2.21) $\|\boldsymbol{y}'\| = L$, we can approximate the normalized arc length derivatives using the following formulas for all i, $i = 0, \ldots, n-1$:

$$\boldsymbol{y}_i' = L\frac{\boldsymbol{y}_{i+1} - \boldsymbol{y}_i}{\|\boldsymbol{y}_{i+1} - \boldsymbol{y}_i\|}, \tag{2.24}$$

$$\boldsymbol{y}_i'' = \frac{\boldsymbol{y}_{i+1}' - \boldsymbol{y}_i'}{s_{i+1} - s_i}. \tag{2.25}$$

From these data we can copy the initial position and velocity $\boldsymbol{y}_0$ and $\boldsymbol{y}'_0$ as well as the goal $\boldsymbol{g} = \boldsymbol{y}_n$. The weights $\boldsymbol{w}_k$ of the forcing term (2.4) can be estimated by rewriting Eq. (2.10) – (2.11) as a second-order differential equation for all i, $i = 0, \ldots, n$:

$$\boldsymbol{y}''_i - \alpha_z(\beta_z(\boldsymbol{g} - \boldsymbol{y}_i) - \boldsymbol{y}'_i) = \frac{\sum_{k=1}^{N} \boldsymbol{w}_k \Psi_k(x_i)}{\sum_{k=1}^{N} \Psi_k(x_i)} x_i. \tag{2.26}$$

In Eq. (2.26), the weights $\boldsymbol{w}_k$ appear linearly, thus given $\boldsymbol{y}_i$, $\boldsymbol{y}'_i$, $\boldsymbol{y}''_i$, and x_i, they can be computed by solving a linear least squares optimization problem.

2.1.5 Execution of normalized AL-DMPs

Since robots are controlled at constant time steps, the information about time needs to be added to normalized AL-DMPs to create an executable robot motion trajectory. The normalized arc length parameter s starts at 0 and reaches 1 at the end of the path. To define a schedule to perform a normalized AL-DMP with a robot, we specify the desired duration of motion T and compute a fifth-order polynomial $s(t)$ with boundary conditions $s(0) = \dot{s}(0) = \ddot{s}(0) = \dot{s}(T) = \ddot{s}(T) = 0,\ s(T) = 1$. Note that $\dot{s}(t) > 0$ for all $0 < t < T$.

To sample the path at the constant time step Δt, $t_{i+1} = t_i + \Delta t$, $t_0 = 0$, we first compute the normalized arc length integration step

$$\Delta s_i = s(t_i + \Delta t) - s(t_i). \tag{2.27}$$

We can then integrate the normalized AL-DMP with a variable integration step Δs_i to sample the corresponding time parametrized trajectory at constant time steps as required by robot controllers. The desired positions $\boldsymbol{y}$ are obtained directly by integrating (2.17) – (2.19) while the velocities $\dot{\boldsymbol{y}}$ and accelerations $\ddot{\boldsymbol{y}}$ (if needed) can be computed as follows

$$\dot{\boldsymbol{y}} = \frac{d}{dt}\boldsymbol{y}(s(t)) = \boldsymbol{y}'\dot{s}, \tag{2.28}$$

$$\ddot{\boldsymbol{y}} = \frac{d}{dt}(\boldsymbol{y}'(s(t))\,\dot{s}(t)) = \boldsymbol{y}''\dot{s}^2 + \boldsymbol{y}'\ddot{s}, \tag{2.29}$$

Note that the choice of schedule $s(t)$ does not change the spatial course of motion. Any continuously increasing time parametrization of s results in the same spatial path, but executed at different speeds. Compared to standard DMPs [34], the normalized AL-DMPs as defined by Eqs. (2.17) – (2.19) do not enable temporal and spatial scaling of the resulting motion. If these properties of DMPs are needed, we can sample the normalized AL-DMP with the given schedule $s(t)$ using the above procedure and re-encode the resulting motion using the standard DMP representation.

2.2 Datasets for Training Deep Image-to-Motion Encoder-Decoder Networks

In order to facilitate the understanding of what follows, we next briefly describe the task used to evaluate the proposed approaches and the data used for network training. Note, however, that the proposed loss functions and the calculation of their gradients are useful to implement back-propagation in any neural network that outputs DMP or normalized AL-DMP parameters, not just the ones considered in our experiments.

Our experimental problem is 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 to learn. 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 DMP or normalized AL-DMP. The robot then calculates the writing movement either from DMP or normalized AL-DMP, replays this movement and writes the digit in the same style as in the captured image.

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

$$\mathbf{D} = \{\mathbf{C}_j, \mathbf{M}_j\}_{j=1}^{P}, \tag{2.30}$$

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, and $\mathbf{M}_j$ the corresponding writing movements associated with each image. The most straightforward form to represent time-dependent trajectory data for execution on a robot or from a human recording is as a temporal sequence of positions on the trajectory

$$\mathbf{M}_j^{f(t)} = \{\boldsymbol{y}_{i,j}, t_{i,j}\}_{i=1}^{T_j}. \tag{2.31}$$

Here, $\boldsymbol{y}_{i,j} \in \mathbb{R}^{\mathrm{d_m}}$ are the robot configurations, e. g. Cartesian positions or joint angles, on the $j-$th trajectory at time $t_{i,j} \in \mathbb{R}$ and d_{m} is the number of degrees of freedom. With arc-length parameterization and L_j length normalization, we sample the data as a function of normalized arc-length $s_{i,j}$

$$\mathbf{M}_j^{f(s)} = \left\{\boldsymbol{y}_{i,j}, \boldsymbol{y}'_{i,j}, \boldsymbol{y}''_{i,j}, s_{i,j}\right\}_{i=1}^{n_j}, \tag{2.32}$$

where $\boldsymbol{y}_{i,j} \in \mathbb{R}^{\mathrm{d_m}}$ are now robot configurations and $\boldsymbol{y}'_{i,j} \in \mathbb{R}^{\mathrm{d_m}}, \boldsymbol{y}''_{i,j} \in \mathbb{R}^{\mathrm{d_m}}$ their derivatives at normalized arc-length $s_{i,j} \in \mathbb{R}$. We explicitly include arc-length derivatives in the training data here as these derivatives can be estimated in different ways, not necessarily from an arc-length dependent sequence of configurations. On the other hand, the estimation of time derivatives from the temporal data sequence in Eq. (2.31) is more straight-forward as time derivatives can be easily estimated directly from temporal sequences.

Note that the number of data points in datasets (2.31) and (2.32) 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 or normalized AL-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

$$\mathbf{M}_j^{DMP} = \left\{\{\boldsymbol{w}_{k,j}\}_{k=1}^{N},\ \tau_j,\ \boldsymbol{g}_j,\ \boldsymbol{y}_{0,j}\right\}, \tag{2.33}$$

while for normalized AL-DMP, the output parameters consist of

$$\mathbf{M}_j^{AL} = \left\{\{\boldsymbol{w}_{k,j}\}_{k=1}^{N},\ \boldsymbol{g}_j,\ \boldsymbol{y}_{0,j},\ \boldsymbol{y}'_{0,j}\right\}. \tag{2.34}$$

See Section 2.1 for more details about these parameters.

2.3 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 [90] is the method of choice to implement such an optimization process. Backprop-

agation requires the gradients of all functions that constitute a neural network, including the gradients of the loss function.

2.3.1 Loss functions for training neural networks with DMP parameters as output

The most common loss function that could be used for training of a neural network with DMP parameters as output is the mean squared error of DMP parameters, which is defined for the j-th training datapoint as follows:

$$\mathrm{E}_{\mathrm{p}}^{DMP}(j) = \frac{1}{2}\left(\sum_{k=1}^{N} \|\boldsymbol{w}_k - \boldsymbol{w}_{k,j}\|^2 + (\tau - \tau_j)^2 + \|\boldsymbol{g} - \boldsymbol{g}_j\|^2 + \|\boldsymbol{y}_0 - \boldsymbol{y}_{0,j}\|^2\right). \tag{2.35}$$

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 (2.33) at index j.

While this loss function provides for an easy implementation, it does not measure the real physical difference between the training movement and the movement calculated by the neural network, but rather the difference between the DMP parameters. A more natural loss function measures the difference between the trajectories generated by the output DMP, here denoted as $\boldsymbol{y}^{\mathrm{DMP}}$, and the training data $\boldsymbol{y}_{i,j}$ from Eq. (2.31). We define the following loss function

$$\mathrm{E}_{\mathrm{t}}(j) = \frac{1}{2T_j}\sum_{i=1}^{T_j} \|\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j}\|^2, \tag{2.36}$$

where $\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j})$ and $x_{i,j} = x(t_{i,j})$ are obtained by integrating the j-th output DMP as calculated by the neural network and T_j is the number of points on the j-th DMP. The process of training deep neural networks for mapping raw images to DMPs with this loss function is presented graphically in Figure 2.1.

To apply backpropagation, we need to be able to compute the gradients of the loss function. The gradients of error function (2.35) are trivial to compute as this is simply the

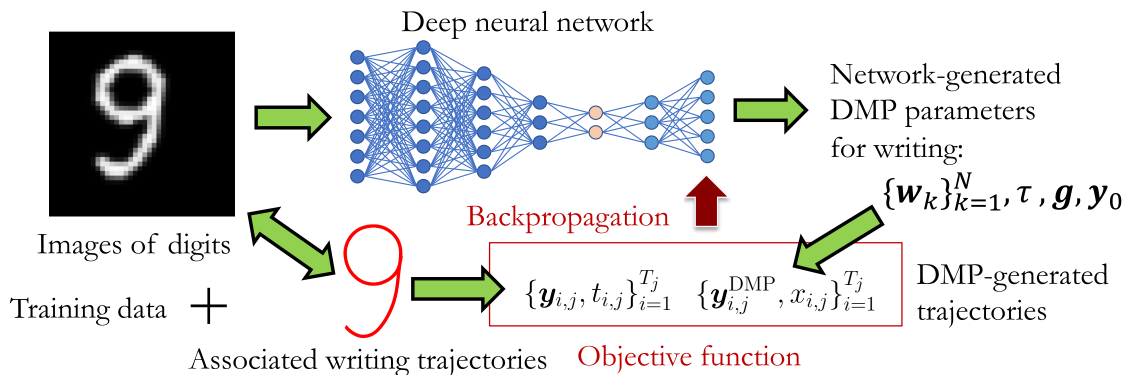


Figure 2.1: Training of writing DMPs with the loss function (2.36). Each input image is fed to the neural network that transforms it into DMP parameters. The output DMP parameters are used to generate a temporal sequence of points on the DMP trajectory, which are compared to the data points on the training 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.

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 (2.36) as $\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j})$ are calculated by integrating DMP equations (2.1) – (2.3).

Let $p_h,\ h = 1, \ldots, H$, be the DMP parameters specified in Eq. (2.6), 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

$$\frac{\partial \mathrm{E_t}(j)}{\partial p_h} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j}\right)^{\mathrm{T}} \frac{\partial \boldsymbol{y}^{\mathrm{DMP}}}{\partial p_h}(x_{i,j}), \tag{2.37}$$

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 (2.1) – (2.3). 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.

We next analyze the structure of partial derivatives $\partial \boldsymbol{y}^{\mathrm{DMP}} / \partial p_h$ that appear in Eq. (2.37). Analyzing differential equation system (2.1) – (2.3), it is clear that of all DMP parameters listed in Eq. (2.6), 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. (2.61) becomes

$$\frac{\partial \mathrm{E_t}(j)}{\partial p_h} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(y_l^{\mathrm{DMP}}(x_{i,j}) - y_{l,i,j}\right) \frac{\partial y_l^{\mathrm{DMP}}}{\partial p_h}(x_{i,j}),\ \forall p_h \neq \tau, \tag{2.38}$$

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

We start with the forcing term parameters $w_{l,k},\ k = 1, \ldots, N,\ l = 1, \ldots, d_m$. The partial derivatives $\partial y^{\mathrm{DMP}} / \partial w_{l,k}$ can be obtained by calculating the derivatives of Eqs. (2.1) and (2.2) with respect to $w_{l,k}$

$$\tau \frac{\partial \dot{z}_l}{\partial w_{l,k}} = \alpha_z(-\beta_z \frac{\partial y_l}{\partial w_{l,k}} - \frac{\partial z_l}{\partial w_{l,k}}) + (g_l - y_{l,0}) \frac{\psi_k(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{2.39}$$

$$\tau \frac{\partial \dot{y}_l}{\partial w_{l,k}} = \frac{\partial z_l}{\partial w_{l,k}}. \tag{2.40}$$

Because the following holds for continuously differentiable trajectories

$$\frac{d}{dt} \frac{\partial}{\partial w_{l,k}} z_l = \frac{\partial}{\partial w_{l,k}} \frac{d}{dt} z_l, \tag{2.41}$$

$$\frac{d}{dt} \frac{\partial}{\partial w_{l,k}} y_l = \frac{\partial}{\partial w_{l,k}} \frac{d}{dt} y_l, \tag{2.42}$$

we obtain the resulting differential equation system with respect to $\partial y_l / \partial w_{l,k}$ and $\partial z_l / \partial w_{l,k}$

$$\tau \frac{d}{dt} \frac{\partial z_l}{\partial w_{l,k}} = \alpha_z(-\beta_z \frac{\partial y_l}{\partial w_{l,k}} - \frac{\partial z_l}{\partial w_{l,k}}) + (g_l - y_{l,0}) \frac{\psi_k(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{2.43}$$

$$\tau \frac{d}{dt} \frac{\partial y_l}{\partial w_{l,k}} = \frac{\partial z_l}{\partial w_{l,k}}. \tag{2.44}$$

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 (2.43) – (2.44). 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

$$\frac{\partial y_l}{\partial w_{l,k}}(1) = \frac{\partial z_l}{\partial w_{l,k}}(1) = 0. \tag{2.45}$$

The partial derivatives with respect to parameters g_l, $y_{0,l}$, $l = 1, \ldots, d_m$, 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. (2.1) and (2.2) with respect to g_l and $y_{0,l}$. We obtain

$$\tau \frac{d}{dt}\frac{\partial z_l}{\partial g_l} = \alpha_z \left(\beta_z \left(1 - \frac{\partial y_l}{\partial g_l} \right) - \frac{\partial z_l}{\partial g_l} \right) + \frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{2.46}$$

$$\tau \frac{d}{dt}\frac{\partial y_l}{\partial g_l} = \frac{\partial z_l}{\partial g_l}, \tag{2.47}$$

and

$$\tau \frac{d}{dt}\frac{\partial z_l}{\partial y_{0,l}} = \alpha_z \left(-\beta_z \frac{\partial y_l}{\partial y_{0,l}} - \frac{\partial z_l}{\partial y_{0,l}} \right) - \frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{2.48}$$

(2.49)

$$\tau \frac{d}{dt}\frac{\partial y_l}{\partial y_{0,l}} = \frac{\partial z_l}{\partial y_{0,l}}. \tag{2.50}$$

The values of the above partial derivatives at phases $x_{i,j}$ can be calculated by respectively integrating the equation systems (2.46) – (2.47) and (2.48) – (2.50). The initial values are set as follows

$$\frac{\partial y_l}{\partial g_l}(1) = \frac{\partial z_l}{\partial g_l}(1) = 0, \tag{2.51}$$

$$\frac{\partial y_l}{\partial y_{0,l}}(1) = 1, \; \frac{\partial z_l}{\partial y_{0,l}}(1) = 0. \tag{2.52}$$

In equation (2.52), 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. (2.3). Instead of the simplified Eq. (2.38), we need to apply the full Eq. (2.61), i.e.

$$\frac{\partial \mathrm{E_t}(j)}{\partial \tau} = \frac{1}{T_j} \sum_{i=1}^{T_j} \left(\boldsymbol{y}^{\mathrm{DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j} \right)^{\mathrm{T}} \frac{\partial \boldsymbol{y}^{\mathrm{DMP}}}{\partial \tau}(x_{i,j}). \tag{2.53}$$

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. (2.1) and (2.2) with respect to τ

$$\tau \frac{d}{dt}\frac{\partial z_l}{\partial \tau} = \alpha_z \left(-\beta_z \frac{\partial y_l}{\partial \tau} - \frac{\partial z_l}{\partial \tau} \right) - \dot{z}_l + (g_l - y_{0,l}) \frac{\partial}{\partial \tau} \left(\frac{\sum_{n=1}^{N} w_{l,n} \Psi_n(x)}{\sum_{n=1}^{N} \Psi_n(x)} x \right), \tag{2.54}$$

$$\tau \frac{d}{dt}\frac{\partial y_l}{\partial \tau} = \frac{\partial z_l}{\partial \tau} - \dot{y}_l. \tag{2.55}$$

Since x depends on τ, we also need to compute

$$\frac{\partial}{\partial \tau}\left(\frac{\sum_{n=1}^{N} w_{l,n}\Psi_n(x)}{\sum_{n=1}^{N}\Psi_n(x)}x\right) =$$

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

$$\frac{\left(\sum_{n=1}^{N}\Psi_n'(x)\right)\left(\sum_{n=1}^{N} w_{l,n}\Psi_n(x)x\right)}{\left(\sum_{n=1}^{N}\Psi_n(x)\right)^2}\frac{\partial x}{\partial \tau}. \quad (2.56)$$

Finally, differential equation (2.3) 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

$$\tau\frac{\partial \dot{x}}{\partial \tau} = -\alpha_x\frac{\partial x}{\partial \tau} - \dot{x}. \quad (2.57)$$

Thus, to calculate $\partial \boldsymbol{y}^{\mathrm{DMP}}/\partial \tau$, we need to integrate $2d+1$ equations comprising differential equation system (2.54), (2.55), and (2.57) in $\partial y_l/\partial \tau$, $\partial z_l/\partial \tau$, and $\partial x/\partial \tau$, with initial values set to

$$\frac{\partial y_l}{\partial \tau}(1) = \frac{\partial z_l}{\partial \tau}(1) = \frac{\partial x}{\partial \tau}(1) = 0, \quad l = 1,\ldots,d_m. \quad (2.58)$$

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 (2.54), (2.55), (2.57) contains the values of $\dot{y}_l$, $\dot{z}_l$, x, and $\dot{x}$, thus the DMP differential equation system (2.1) – (2.3) 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. (2.53), one could consider τ as constant when optimizing the loss function (2.36) to calculate the rest of the DMP parameters, i. e. $\{\boldsymbol{w}_k\}_{k=1}^N$, $\boldsymbol{g}$, and $\boldsymbol{y}_0$. We could then estimate τ by a separate deep neural network.

2.3.2 Loss functions for training neural networks with normalized AL-DMPs as output

Similarly to the DMP loss function in Eq. (2.35) we can define mean squared error for the j-th normalized AL-DMP parameters:

$$\mathrm{E_p^{\mathit{AL}}}(j) = \frac{1}{2}\left(\sum_{k=1}^{N}\|\boldsymbol{w}_k - \boldsymbol{w}_{k,j}\|^2 + \|\boldsymbol{g} - \boldsymbol{g}_j\|^2 + \|\boldsymbol{y}_0 - \boldsymbol{y}_{0,j}\|^2 + \|\boldsymbol{y}_0' - \boldsymbol{y}_{0,j}'\|^2\right). \quad (2.59)$$

where $\{\{\boldsymbol{w}_k\}_{k=1}^N,\ \boldsymbol{g},\ \boldsymbol{y}_0\ \ \boldsymbol{y}_0'\}$ denotes the output of the neural network and $\{\{\boldsymbol{w}_{k,j}\}_{k=1}^N,\ \boldsymbol{g}_j,\ \boldsymbol{y}_{0,j}\ \ \boldsymbol{y}_{0,j}'\}$ the normalized AL-DMP parameters from the training data in Eq. (2.34). As in the case of DMPs, this loss function formulation does not directly measure the difference between the training movement and the movement calculated by the neural network, but rather the difference between the normalized AL-DMP parameters. For direct measurement of the difference between the trajectories defined by normalized AL-DMP parameters calculated by the neural network, here denoted as $\boldsymbol{y}^{\mathrm{AL-DMP}}$, and the

training data $\boldsymbol{y}_{i,j}$ from Eq. (2.32), we need to define the following loss function

$$\mathrm{E_a}(j) = \frac{1}{2n_j} \sum_{i=1}^{n_j} \|\boldsymbol{y}^{\mathrm{AL-DMP}}(x_{i,j}) - \boldsymbol{y}_{i,j}\|^2, \tag{2.60}$$

where $\boldsymbol{y}^{\mathrm{AL-DMP}}(x_{i,j})$ and $x_{i,j} = x(t_{i,j})$ are obtained by integrating the AL-DMP calculated by the neural network. In Figure 2.2, the process of training deep neural networks for mapping raw images to normalized AL-DMPs with this loss function is graphically compared to the process of training deep neural networks for mapping raw images to DMPs with loss function (2.36).

It is more difficult to compute the gradients of (2.60) with respect to the normalized AL-DMP parameters than of (2.59) because $\boldsymbol{y}(x_{i,j})$ are calculated by integrating differential equations (2.17) – (2.19). Below we denote by p_a, $a = 1, \ldots, A$, any of the normalized AL-DMP parameters specified in Eq. (2.34). The partial derivatives of $\mathrm{E_p}(j)$ with respect to each normalized AL-DMP parameter p_a can be calculated as follows

$$\frac{\partial \mathrm{E_a}(j)}{\partial p_a} = \frac{1}{n_j} \sum_{i=1}^{n_j} (\boldsymbol{y}(x_{i,j}) - \boldsymbol{y}_{i,j})^{\mathrm{T}} \frac{\partial \boldsymbol{y}}{\partial p_a}(x_{i,j}), \tag{2.61}$$

where $\partial \boldsymbol{y}/\partial p_a$ is the partial derivative of $\boldsymbol{y}$ with respect to the normalized AL-DMP parameter p_a.

Next, we show how to compute the partial derivatives $\partial \boldsymbol{y}/\partial p_a$. Using notation $\boldsymbol{y} = [y_1, \ldots, y_d]^{\mathrm{T}}$, we observe that for any parameter p_a and any continuously differentiable

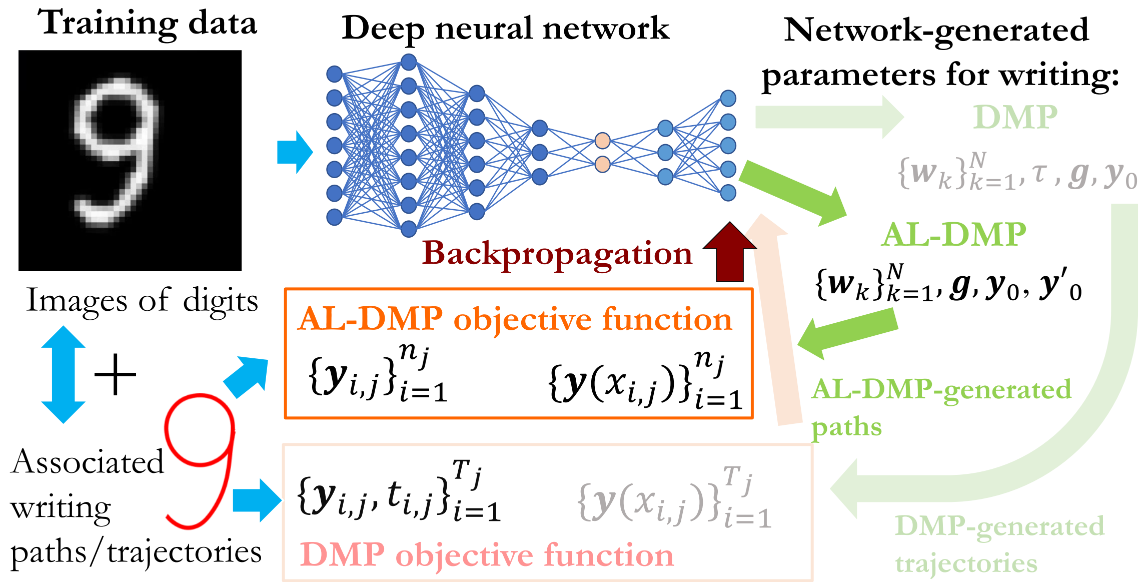


Figure 2.2: Comparison of training with normalized AL-DMPs (bold) and DMPs (shaded) parameters. An input image is fed to a deep neural network, which transforms the image into normalized AL-DMP (DMP) parameters. The output normalized AL-DMP (DMP) is then used to generate a sequence of points along the path (trajectory). During training, the generated points are compared with the data points along the training path (as a function of arc-length) / trajectory (as a function of time). The parameters of the deep neural network are estimated by backpropagation.

path, the following holds true for partial derivatives $\partial y_l/\partial p_a$, $\partial z_l/\partial p_a$, $l = 1, \ldots, d_m$,

$$\frac{d}{ds}\frac{\partial}{\partial p_a} z_l = \frac{\partial}{\partial p_a}\frac{d}{ds} z_l = \frac{\partial z_l'}{\partial p_a},$$

$$\frac{d}{ds}\frac{\partial}{\partial p_a} y_l = \frac{\partial}{\partial p_a}\frac{d}{ds} y_l = \frac{\partial y_l'}{\partial p_a}.$$

Thus $\partial y_l/\partial w_{l,k}$ and $\partial z_l/\partial w_{l,k}$ can be obtained by calculating the derivatives of (2.17) and (2.18) with respect to $w_{l,k}$

$$\frac{d}{ds}\frac{\partial z_l}{\partial w_{l,k}} = \alpha_z(-\beta_z \frac{\partial y_l}{\partial w_{l,k}} - \frac{\partial z_l}{\partial w_{l,k}}) + \frac{\psi_k(x)}{\sum_{n=1}^{N} \Psi_n(x)} x, \tag{2.62}$$

$$\frac{d}{ds}\frac{\partial y_l}{\partial w_{l,k}} = \frac{\partial z_l}{\partial w_{l,k}}, \tag{2.63}$$

and integrating the resulting differential equation system of $\partial y_l/\partial w_{l,k}$ and $\partial z_l/\partial w_{l,k}$ with initial values

$$\frac{\partial y_l}{\partial w_{l,k}}(1) = \frac{\partial z_l}{\partial w_{l,k}}(1) = 0. \tag{2.64}$$

The partial derivatives with respect to the parameters g_l, $y_{0,l}$, $y_{0,l}'$, $l = 1, \ldots, d_m$, are obtained analogously, i. e. by calculating the partial derivatives of Eqs. (2.17) and (2.18) with respect to g_l, $y_{0,l}$ and $y_{0,l}'$. This results in

$$\frac{d}{ds}\frac{\partial z_l}{\partial g_l} = \alpha_z \left(\beta_z \left(1 - \frac{\partial y_l}{\partial g_l}\right) - \frac{\partial z_l}{\partial g_l}\right), \tag{2.65}$$

$$\frac{d}{ds}\frac{\partial y_l}{\partial g_l} = \frac{\partial z_l}{\partial g_l}, \tag{2.66}$$

and

$$\frac{d}{ds}\frac{\partial z_l}{\partial y_{0,l}} = \alpha_z \left(-\beta_z \frac{\partial y_l}{\partial y_{0,l}} - \frac{\partial z_l}{\partial y_{0,l}}\right), \tag{2.67}$$

$$\frac{d}{ds}\frac{\partial y_l}{\partial y_{0,l}} = \frac{\partial z_l}{\partial y_{0,l}}, \tag{2.68}$$

and

$$\frac{d}{ds}\frac{\partial z_l}{\partial y_{0,l}'} = \alpha_z \left(-\beta_z \frac{\partial y_l}{\partial y_{0,l}'} - \frac{\partial z_l}{\partial y_{0,l}'}\right), \tag{2.69}$$

$$\frac{d}{ds}\frac{\partial y_l}{\partial y_{0,l}'} = \frac{\partial z_l}{\partial y_{0,l}'}. \tag{2.70}$$

The values of the above partial derivatives at phases $x_{i,j}$ can be calculated by respectively integrating the equation systems (2.46) – (2.47), (2.48) – (2.50) and (2.69) – (2.70), where the initial values are set as follows

$$\frac{\partial y_l}{\partial g_l}(1) = \frac{\partial z_l}{\partial g_l}(1) = 0, \tag{2.71}$$

$$\frac{\partial y_l}{\partial y_{0,l}}(1) = 1, \ \frac{\partial z_l}{\partial y_{0,l}}(1) = 0, \tag{2.72}$$

$$\frac{\partial y_l}{\partial \tilde{y}'_{0,l}}(1) = 0, \ \frac{\partial \tilde{z}_l}{\partial \tilde{y}'_{0,l}}(1) = 1. \tag{2.73}$$

In equation (2.72), we took into account that y_l is initially set to $y_{0,l}$, and in equation (2.73), that y'_l is initially set to $y'_{0,l}$.

If we used differential equation system (2.10) – (2.12) instead of (2.17) – (2.19), we would also need to calculate the partial derivatives with respect to parameter L. It turns out that since L affects all dimensions of control variable $\boldsymbol{y}$, the calculation of partial derivatives with respect to L is similar to the calculation of partial derivatives with respect to τ in equation system (2.54) - (2.58), thus much more complicated than for other AL-DMP variables. Within the AL-DMP formulation, the temporal part and the spatial part of the trajectory are separated. The spatial part is encoded in the parameters that we used for neural network training. By excluding L from parameter sets with normalization, some parameters change value but the encoded spatial trajectory remains intact. At the same time, its exclusion greatly simplifies and consequently accelerates the training of neural networks.

2.4 Deep Neural Network Architectures

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 or normalized AL-DMP parameters of writing trajectories that reproduce the observed digit.

We used different types of encoder-decoder neural network architectures to carry out these experiments. The deep neural network architecture that we developed and tested as first is a fully-connected image-to-motion encoder-decoder network architecture (IMEDNet). A fully-connected neural network is easy to implement, and its simple design with the possibility of a large number of trainable parameters ensures a big representational power. All this is useful for initial testing and proof-of-concept.

A smaller number of trainable parameters accelerates the training of the neural network and can also result in better generalization. Therefore, in the next step we developed two different variants of a CNN-based architecture, i.e. a convolutional image-to-motion encoder-decoder network (CIMEDNet). The first of these variants uses pre-trained convolutional layers in the encoder part of the network exclusively, followed by fully-connected layers in the decoder part. The other variant uses pre-trained convolutional layers followed by some additional fully-connected layers in the encoder, whereas the decoder is again constructed of fully-connected layers only.

For real images, neural networks should be able to process input data with background. All three network architectures mentioned above have fixed-size inputs, hence they are only applicable if the input images have a fixed size. They can only be used on the same size they have been trained on. The variety of image sizes from robot cameras is too large to allow successful training of neural networks on all possible input images with cluttered background. We can overcome this issue with a network architecture capable of processing variable input sizes. Such a network architecture allows training on fixed-size input images and can then be applied to process robot camera images of different sizes.

Thus in the next development step, we upgraded our convolutional neural network architecture with a global maximal pooling layer to create a new network architecture

called CIMEDNet+ that is capable of processing input images of different sizes. In the final development step, we added additional layers to the CIMEDNet+ architecture to create a more powerful variable-size input image-to-motion encoder-decoder network (VIMEDNet).

2.4.1 Fully-connected encoder-decoder architecture

The first of the applied deep neural network architectures is the fully-connected image-to-motion encoder-decoder network architecture (IMEDNet) shown in Fig. 2.3. Its application was inspired by the work of Hinton and Salakhutdinov [57]. It has an input layer consisting of 1600 neurons (for an image size of 40×40 pixels) and was applied to reproduce 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. (2.4), 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). Alternatively, the output consists of 56 neurons that correspond to the parameters of a two-dimensional normalized AL-DMP (2×25 neurons for the weights of the two forcing terms, 2 neurons each to specify the initial derivative $\boldsymbol{y}_0'$, the initial position $\boldsymbol{y}_0$ and the end position $\boldsymbol{g}$ on the trajectory). 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 around 6.38 million. This configuration of neurons forms an asymmetrical encoder-decoder architecture. The bottleneck layer consists of 20 neurons, which can be used to define the latent space for writing trajectories.

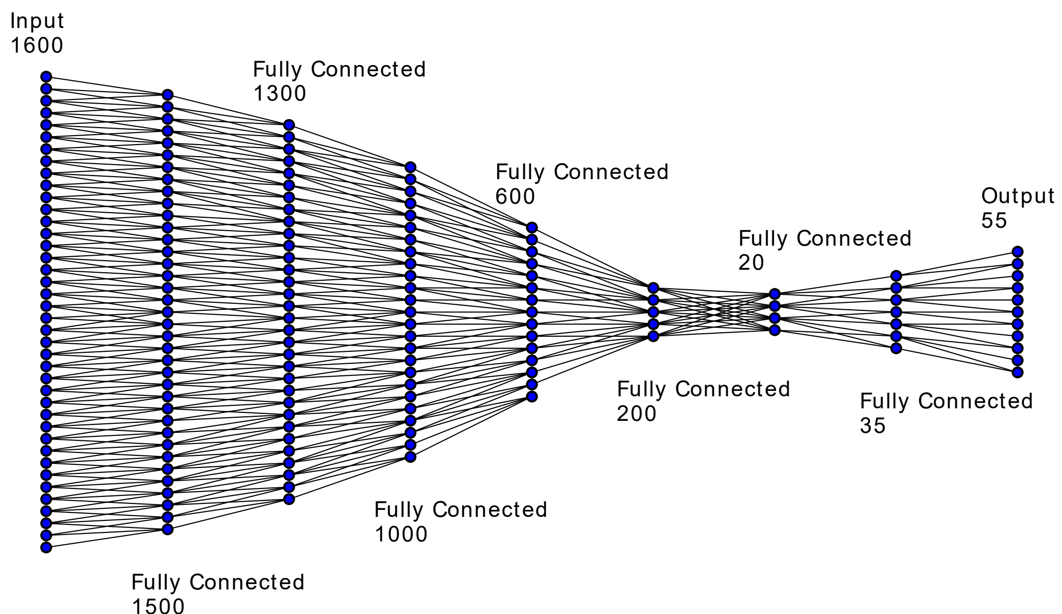


Figure 2.3: IMEDNet (Image-to-Motion Encoder-Decoder Network) neural network with seven fully-connected hidden layers (note that the ratio between the number of actual neurons and the number of depicted neurons is not the same in all layers).

2.4.2 Convolutional encoder-decoder architectures

Convolutional neural networks (CNN) are specialized for processing data in a grid-like form. They use mathematical operation convolution to compute feature maps from input images. Convolution is defined by a small number of neurons called kernel. It adds to neural network's important properties in terms of image processing. By making kernels smaller than input images, the number of parameters that need to be learned is significantly reduced. The application of the same parameters to process different parts of input data is called parameter sharing. This form of parameter sharing also provides for invariance to translation.

Each layer of CNN typically consists of three stages. The first layer performs several convolutions in parallel to produce a set of linear activations. The second stage is called detection where the computed activations are run through a nonlinear activation function. In the last stage, a pooling function modifies the input to the next layer. The pooling function replaces the outputs with a summary statistic of all nearby outputs. This helps make the network invariant to small translations of the input.

The purpose of creating a CNN-based architecture, i.e. a convolutional image-to-motion encoder-decoder network (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 reduced performance. For example, in the first version of CIMEDNet, where the encoder part consisted of convolutional layers only as shown in Fig. 2.4 and the number of trainable parameters was around 23.000, the network performance was lower compared to IMEDNet in our initial experiments.

Good results were obtained by the proposed architecture in Fig. 2.5, which still contains significantly less parameters than IMEDNet while preserving sufficient representational power to map images to handwriting DMPs or normalized AL-DMPs. We used this version of CIMEDNet in all other experiments. It consists of an encoder that contains 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, fully-connected layers of size 600, 200, 20, 35 and the output layer of size 55 or 56, matching the number of DMP or normalized AL-DMP parameters. The number of all trainable parameters in this network is around 721.000.

The encoder part of the proposed network has a somewhat similar structure as the LeNet-5 architecture [91]. The fully connected encoder-decoder part is taken over from IMEDNet with the first three layers removed.

In the experiments described in Section 2.5.4, we tested the robustness of the proposed networks to noise. We also tested the training with frozen convolutional layers for both of the proposed CIMEDNet architectures. When training with frozen convolutional layers, these layers were pretrained as a part of a CNN classifier that provides class names as output. We used the original MNIST dataset to train the classifier. This classifier 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 layer of size 320, a fully-connected layer of size 50 and the output layer of size 10, matching the number of digits. After training the classifier, the fully-connected layers are removed, while the convolutional layers are retained and used to form the first layers of the encoder in our proposed CIMEDNet architectures.

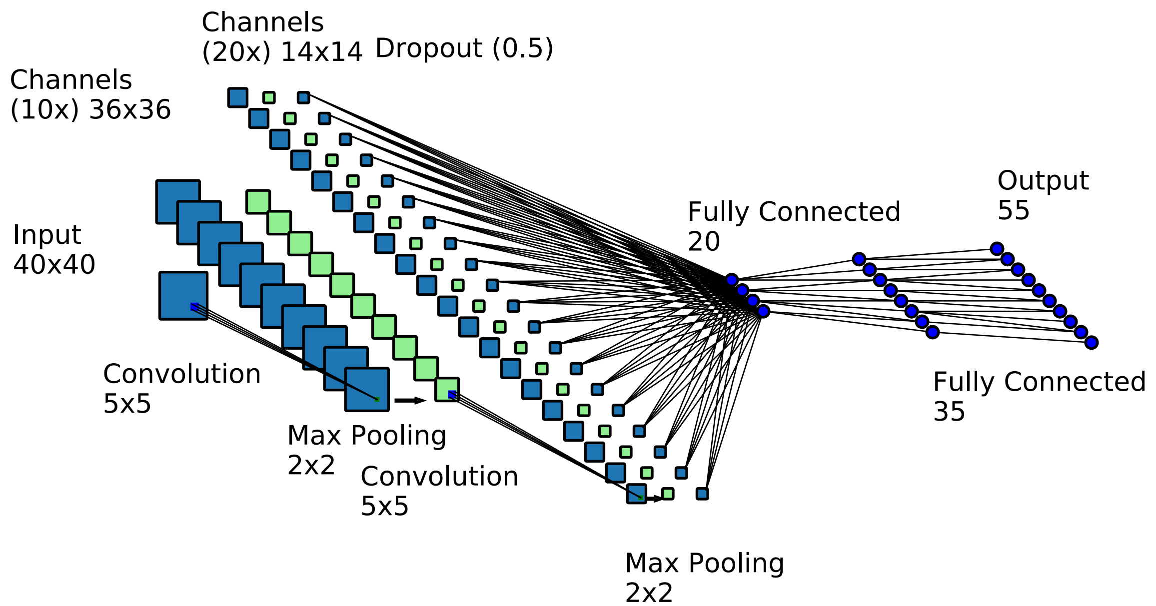


Figure 2.4: CIMEDNet (convolutional image-to-motion encoder-decoder network) architecture with an encoder consisting of two convolutional layers and a decoder consisting of two fully-connected layers.

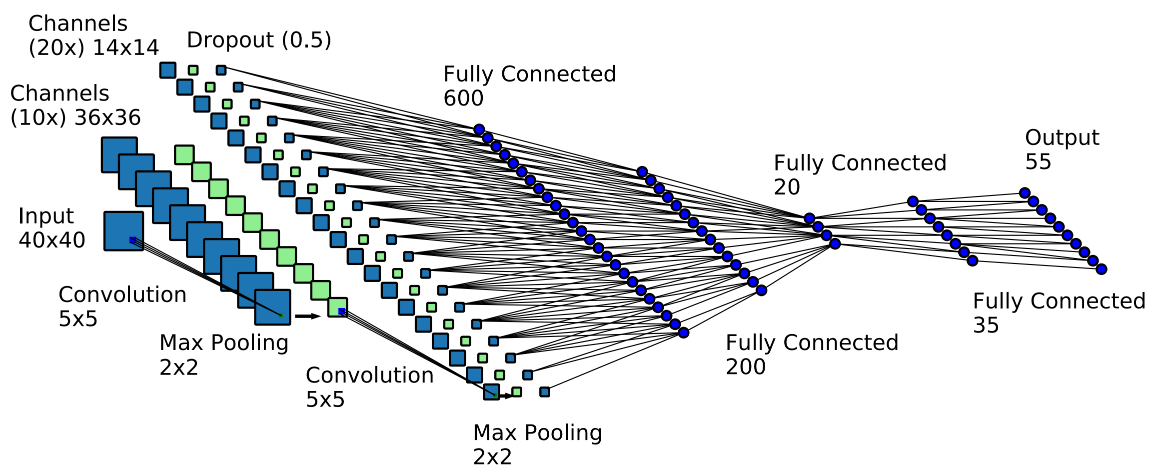


Figure 2.5: CIMEDNet architecture with an encoder consisting of two convolutional layers followed by three fully-connected layers, and a decoder consisting of two fully-connected layers.

2.4.3 Convolutional encoder-decoder architectures for variable size input images

The CIMEDNet architectures described above take images of fixed size as input. Here, we propose two network architectures that enable the processing of variable size images with a significant portion of cluttered background. The ability to process input images of variable sizes is achieved by introducing a max-pooling layer that transforms input data into a vector of fixed size just before the decoder part of the network.

The neural network architectures described above are effective if the input image contains only the sheet of paper with a digit written on it. They cannot, however, deal with images that contain a highly textured background outside of the said sheet of paper. To reproduce digits with a robot, we thus still need standard machine vision algorithms to find and extract the sheet of paper from the acquired camera image. To generate the inputs for both networks in our experiments, the observed images of digits were cropped and resized to a fixed-size input (see also Section 2.5.8). To alleviate the need for standard machine vision algorithms, we need neural networks that can deal with a significant amount of textured background not related to the images of digits.

To achieve invariance to varying backgrounds, we have created a new dataset of images of digits that contain a significant portion of background outside of the sheet of paper on which digits are written (see Section 2.5.1.3). For this new dataset, we first experimented with the CIMEDNet architecture. The results in Section 2.5.7 show that the performance of CIMEDNet decreases drastically already at an image size of 120x120 pixels. Such a neural network cannot be trained to the size of images obtained from a real robot. The application of CNNs like in CIMEDNet instead of fully connected neural networks like in IMEDNet already helps to extend the applicability of the trained neural networks to larger inputs. Fewer network parameters in CNNs at the same input size make training faster, less memory demanding and enable better generalization. But even for CNNs it still applies that the number of trainable parameters increases with the increasing input size. As a result, training slows down, generalization performance decreases, and especially when using GPUs, our databases can exceed the available GPU memory.

In many cases, the relevant information is only a local part of the input, and CNN must learn how to extract this information. As in the case of digit writing, the neural network has to use information from the part of the input image with the sheet of paper, everything else is just background that the neural network has to ignore. If the neural network can learn this on the small inputs with paper sheets in front the background, the neural network will generalize to larger inputs with much more background in the image without additional training. It is often useful to be able to process images of different sizes because the trained images might come from different sources and rescaling to a fixed size can lead to the loss of information. However, we still make the assumption that the relevant details are always in the same size range.

Classical CNN architectures like the proposed CIMEDNet cannot directly transfer network weights between architecture versions with varying size inputs. The problem does not lie in the convolutional layers themselves because the number of parameters within a general convolution layer depends only on the kernel size, the number of input channels, and the number of output channels. The part that prevents the application of CNNs to varying size images is the transition from the convolutional part to the fully connected part of the architecture. The input in the first fully connected layer is the vectorized output from the last convolutional layer. In the convolutional layers, the size of the output channels depends on the size of the input channels, and this dependence passes through all convolutional layers to the last convolutional output. Thus the size of the input to the fully connected part depends on the input image size of the neural network. To solve this

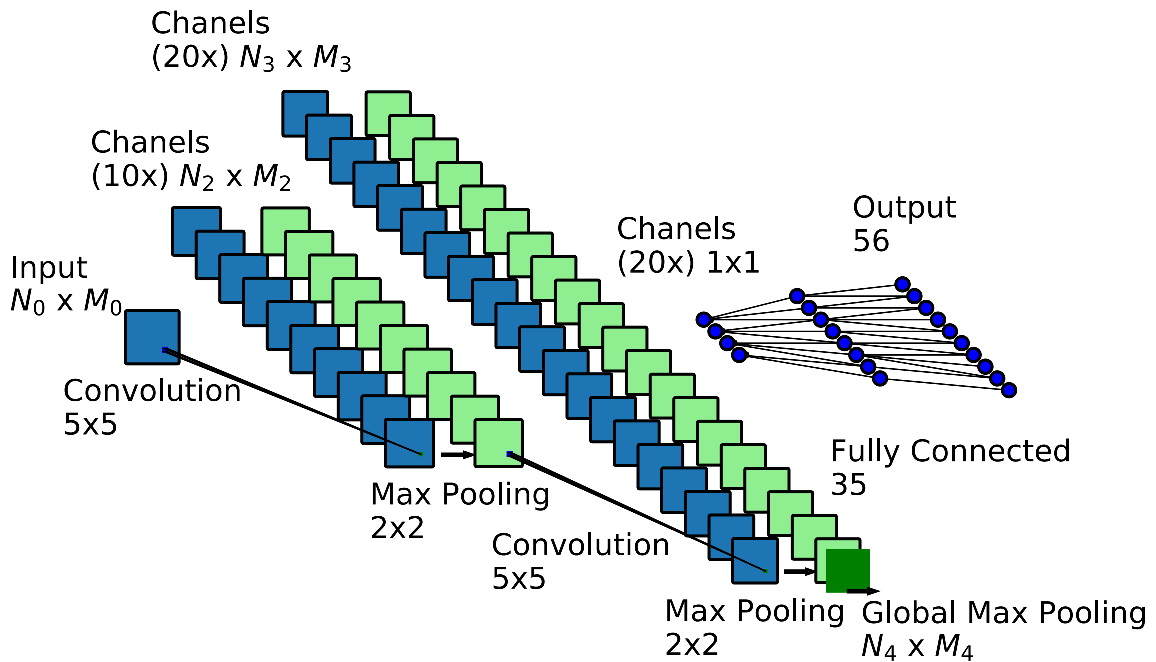


Figure 2.6: CIMEDNet+ (convolutional image-to-motion encoder-decoder network + global polling layer) architecture with an encoder consisting of two convolutional layers followed by a global max pooling layer, and a decoder consisting of two fully-connected layers. Note that the size of channels in the convolutional part of the network is not fixed. Thus the network can take images of any size as input.

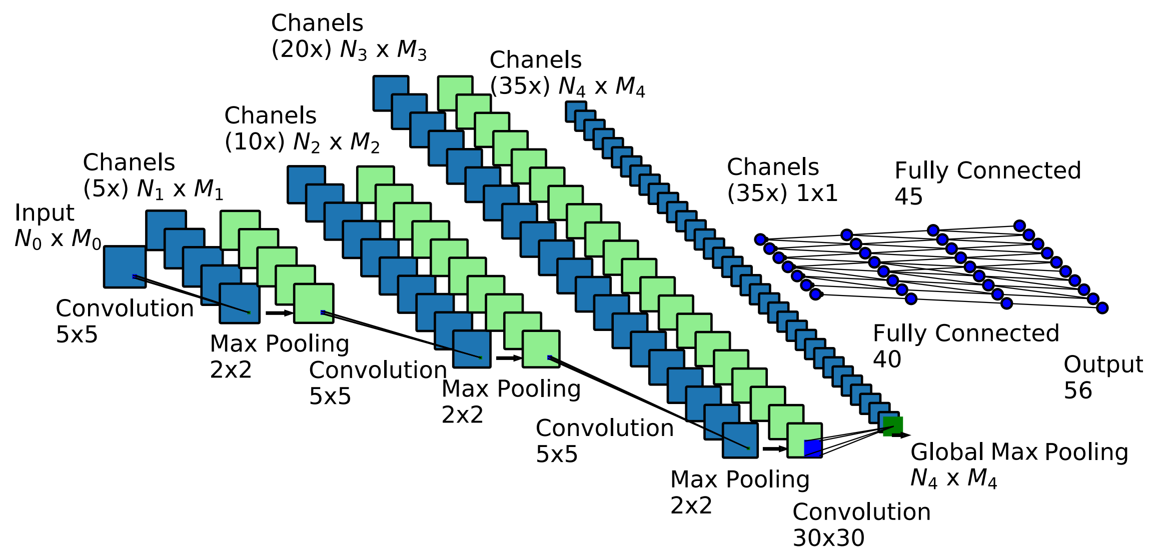


Figure 2.7: VIMEDNet (variable input to motion encoder-decoder network) architecture with an encoder consisting of four convolutional layers followed by a global max pooling layer, and a decoder consisting of three fully-connected layers. Note that the size of channels in the convolutional part of the network is not fixed. Thus the network can take images of any size as input.

problem, we introduced a 2D max pooling layer between the last convolutional layer and the first fully connected layer. The kernel size of the maximum pooling layer corresponds to the size of the last convolutional channels, thus it always generates output channels of size one and serves as the global maximum pooling layer. In this architecture, the only part that changes with the size of the input image is the kernel size of the maximum pooling layer. Since the kernel of the max pooling layer has no trainable parameters,such an architecture can be used with inputs of different sizes than the training inputs.

We developed a network architecture CIMEDNet+ based on the initial version of CIMEDNet (shown in Fig. 2.4). The ability to process input images of varying sizes is brought into CIMEDNet+ by adding a global maximum pooling layer instead of fully connected layers in the encoder part of the network (see Fig. 2.6). In CIMEDNet+, the output from the global maximum pooling layer also represents the latent space. The global maximal pooling layer chooses the maximum value from each channel and computes the bottleneck layer neurons from these values. This way we obtain a fixed number of neurons in the bottleneck layer because the number of channels is constant.

Our experiments with cluttered background show (see Sec. 2.5.7) that the performance of CIMEDNet+ is not as good as the performance of the original CIMEDNet architecture, which is probably due to the reduced number of parameters in CIMEDNet+ compared to CIMEDNet. We therefore inserted two additional convolutional and max pooling layers into the encoder part of the network and one additional fully connected layer into the decoder part (see Fig. 2.7). We also increased the number of neurons in the bottleneck layer from 20 to 35. The resulting architecture is called VIMEDNet (variable image to motion encoder-decoder network). Our experiments have shown (see Section 2.5.8 for details) that the proposed architecture is powerful enough to transform robot real camera images of digits with a significant amount of background into writing trajectories.

2.5 Experimental Evaluation

The main goal of our experiments was to evaluate the proposed loss functions and deep neural networks for learning DMPs and normalized AL-DMPs. For testing purposes we applied the developed networks and criterion functions to the problem of reproducing handwritten digits. For the development of convolutional neural networks, we first tested the robustness of the proposed neural networks to noise. Then we focused on the effectiveness of different movement representations and loss functions for the reproduction of handwritten digits. We compared the performance of the proposed approach when standard DMPs and normalized AL-DMPs are used as a movement representation. In the third set of experiments, we evaluated the performance of CIMEDNet, CIMEDNet+, and VIMEDNet networks on input images of different sizes. Finally, we tested the network performance when real images taken by a humanoid robot are used as input.

2.5.1 Datasets

Here we explain the acquisition of datasets that were used in our experiments for training and testing. We have created three different groups of datasets. In the first group, we hand-annotated trajectories for images with human-written digits from the MNIST dataset. Since hand annotation is time-consuming and requires a lot of human work for large datasets, we created a second group of datasets with computer-generated writing trajectories and the corresponding synthetic images of digits. To evaluate the ability of different neural network architectures to process images with cluttered background, we created the third group of datasets. This group was created by pasting images from the

second group datasets onto different backgrounds.

2.5.1.1 Annotated MNIST (a-MNIST)

To evaluate our methodology, we need pairs of input and output data. Because until now, no suitable dataset exist with pairs of images and corresponding motions, we first created our own dataset by annotating images of digits with the corresponding handwriting motions.

The first dataset was created based on the well-known MNIST dataset [58], which consists of a training set of 60,000 samples and a test set of 10,000 samples of handwritten digit images of size 28×28 grayscale pixels, 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. Some examples of annotated images are shown in Fig. 2.8. For training, we generated 11700 samples of digit 3 and 11700 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, $\pm 8°$ for rotations, $\pm 10\%$ 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.

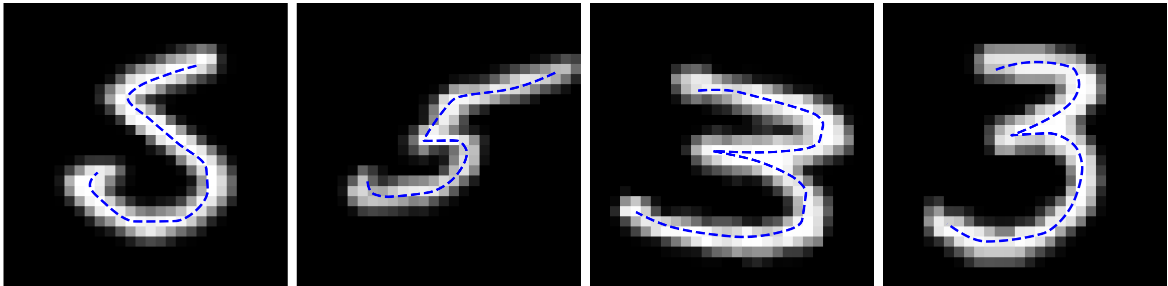


Figure 2.8: Examples of annotated digit images from the a-MNIST dataset. The blue dashed line represents the writing trajectory corresponding to the digit in the image.

2.5.1.2 Synthetic MNIST (s-MNIST)

The size of a-MNIST dataset is limited by the necessity of hand-annotating images of digits with handwriting trajectories, which 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 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 centers of elliptic arcs. We varied these parameters according to the 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-described procedure, several datasets of pairs of synthetic digit images and the associated handwriting trajectories were generated both with and without different types of noise:

- **s-MNIST:** 2000 pairs of images and trajectories without any added noise were generated for each digit (see Fig. 2.9), for a total of 20000 samples,

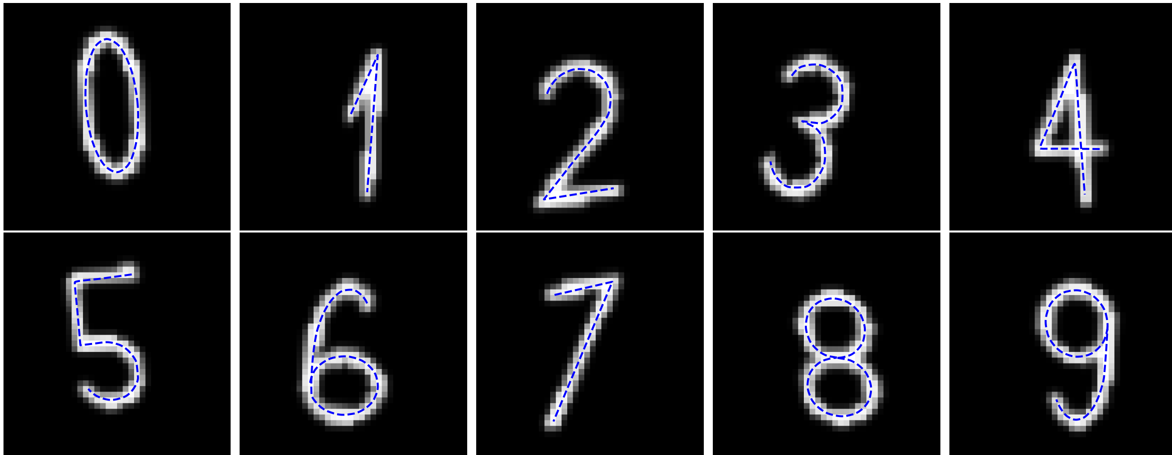


Figure 2.9: Examples of digit images and writing trajectories from the s-MNIST dataset. The blue dotted line represents the trajectory corresponding to the image.

- **s-MNIST-AWGN-19.0-SNR:** 300 samples per digit / 3000 total samples, using additive white Gaussian noise with the signal-to-noise ratio of 19.0 (see Fig. 2.10),

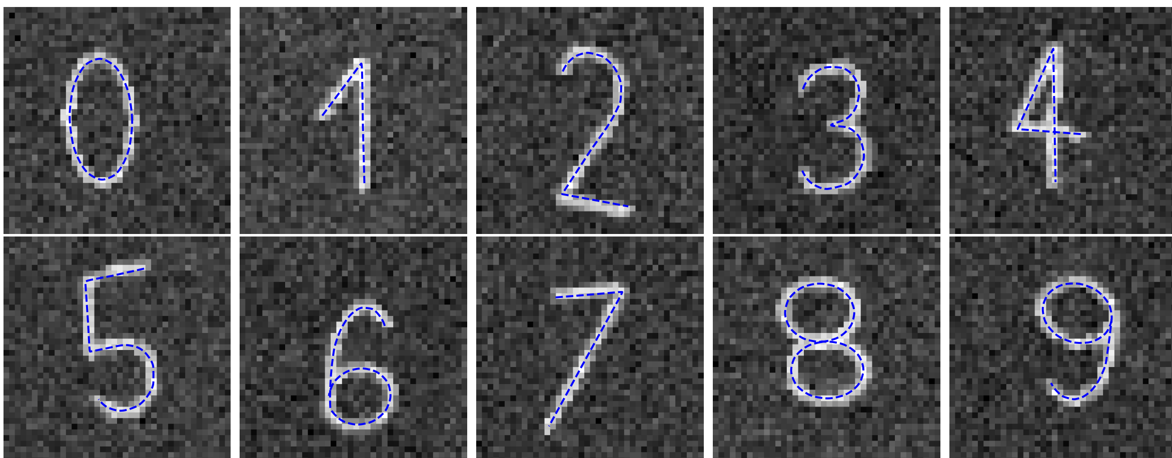


Figure 2.10: Examples of digit images and writing trajectories from the s-MNIST-AWGN-19.0-SNR dataset. The blue dashed line represents the trajectory corresponding to the image.

- **s-MNIST-AWGN-9.5-SNR:** 300 samples per digit / 3000 total samples, using additive white Gaussian noise with the signal-to-noise ratio of 9.5 (see Fig. 2.11),
- **s-MNIST-MB:** 300 samples per digit / 3000 total samples, using a motion blur

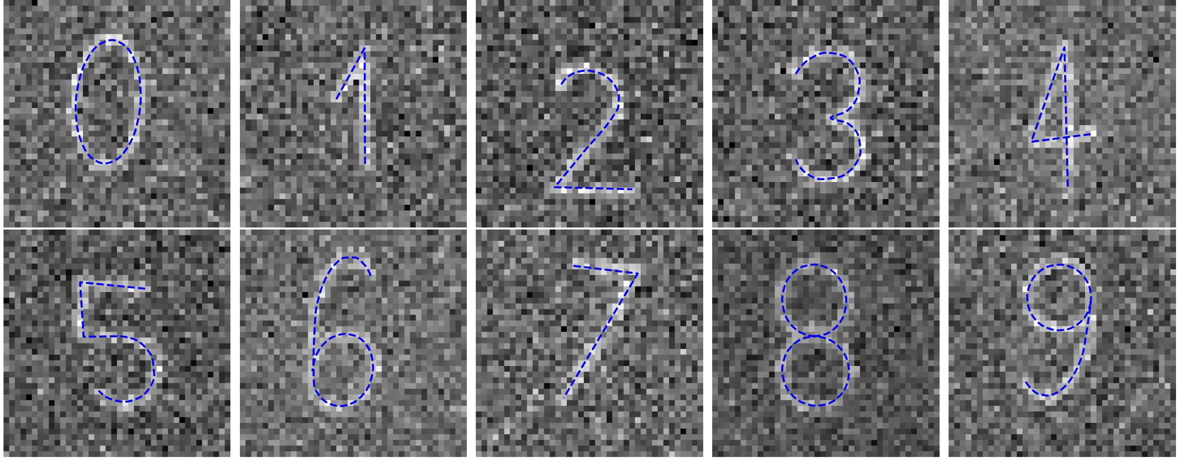


Figure 2.11: Examples of annotated digit images from the s-MNIST-AWGN-9.5-SNR dataset. The blue dashed line represents the trajectory corresponding to the image.

filter emulating a linear motion of the camera of 5 pixels and a 15 degree motion in the counterclockwise direction (see Fig. 2.12),

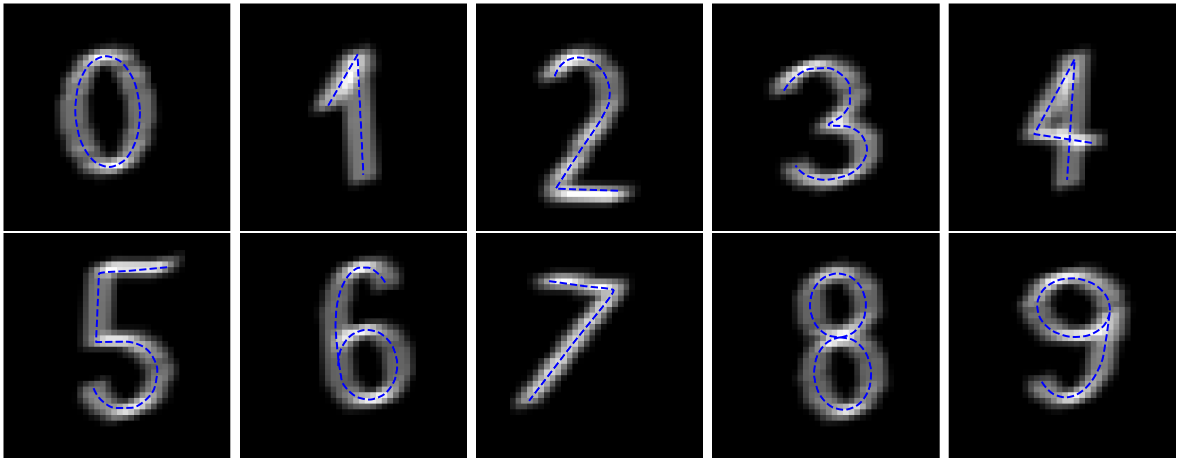


Figure 2.12: Examples of annotated digit images from the s-MNIST-MB dataset. The blue dashed line represents the trajectory corresponding to the image.

- **s-MNIST-RC-AWGN:** 300 samples per digit / 3000 total samples, using a contrast range scaled down to half as well as additive white Gaussian noise with the signal-to-noise ratio of 9.5 (see Fig. 2.13),

We call these datasets synthetic MNIST (s-MNIST), as they simulate the real MNIST dataset but do not contain real images from MNIST. The above noise profiles are the same as in the noisy MNIST dataset (n-MNIST) [92], which is a publicly available set of three sub-datasets that standardize noise distortion of the original MNIST dataset. The s-MNIST dataset of 20000 examples without noise was used as a training set for experiments in Section 2.5.4 and Section 2.5.5 and as a source of images with digits to create datasets with background in Section 2.5.1.3. All other s-MNIST datasets with noise and only 3000 examples were used only for testing in Section 2.5.4 when we evaluated the neural network architectures.

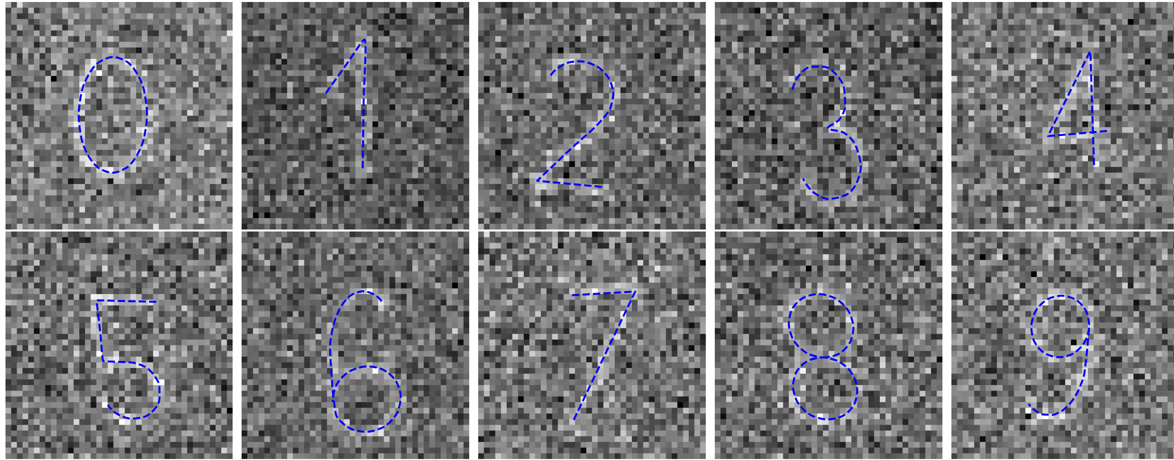


Figure 2.13: Examples of annotated digit images from the s-MNIST-RC-AWGN dataset. The blue dashed line represents the trajectory corresponding to the image.

The s-MNIST dataset has the constant width of the curves that form the images of digits and constant gray levels, with the digits curve always white and a sheet of paper in the background always black. In order to use a deep neural network trained with such a dataset on real robot images (see Section 3.5), we need computer vision algorithms in which we rescale the color scheme and width to match the training dataset. We do not use image processing algorithms when using variable input size CNNs on real robot images that contain a textured background. This variation must be processed in the network itself. The neural network must be trained on a dataset that contains this variation just like the following dataset:

- **s-MNIST-GRAY:** 2000 samples per digit / 20000 total samples, with varied width of curves forming the images of digits and the background computed with varying grayscale (see Fig. 2.14),

Figure 2.14: Examples of annotated digit images from the s-MNIST-GRAY dataset. The blue dashed line represents the trajectory corresponding to the image.

Since sharp corners result in discontinuous velocities, AL-DMPs cannot represent such paths. Within standard DMPs and real physical movements, sharp corners are handled

by reducing the movement velocity, but AL-DMPs cannot do this because the derivatives with respect to the arc length parameter always have a unit norm. For our normalized AL-DMP experiments, we therefore used only the digits without sharp corners: 0, 6, 8 and 9. See Chapter 4 for the discussion on how to deal with other digits using AL-DMP representation.

To account for variability in the temporal execution of motion typical for humans, we created a dataset which consists of the same spatial trajectories (paths) as s-MNIST, but with nonlinearly scaled temporal course of motion. At the beginning of Section 2.1.5, we defined the normalized arc length as a function of time by a fifth order polynomial $s(t)$. This parametrization was also used to generate s-MNIST datasets. For this new dataset, we scaled $s(t)$ by a quadratic function

$$f(s(t)) = as(t)^2 + (1-a)s(t) \tag{2.74}$$

with randomly selected parameter a, where $-1 \leq a \leq 1$. The resulting parametrization is still strictly increasing but is nonlinearly transformed in time. While such a transformation does not change the weights $\boldsymbol{w}$ of normalized AL-DMPs, it does change the weights of standard DMPs. The important difference between s-MNIS and temporally scaled s-MNIST is that in the former, the selection of duration of motion does not affect the weights of DMPs, whereas in the latter it does. This is problematic for learning with DMPs because it is not possible to learn random choices of timing. This affects the performance of networks outputting DMPs. Similarly as for s-MNIST, we also created four digits 0,6,8,9 dataset to train neural networks suitable for processing real images captured by a robot.

For experiments with normalized AL-DMPs, we created altogether three additional datasets:

- **s-MNIST(0,6,8,9):** 5000 pairs of images and trajectories without any added noise were generated for digits 0, 6, 8, 9 for a total of 20000 samples,
- **s-MNIST(0,6,8,9)-VTS:** 5000 pairs of images and trajectories temporally scaled for digits 0, 6, 8, 9 for a total of 20000 samples,
- **s-MNIST(0,6,8,9)-GRAY:** 5000 pairs of grayscale images created with the varied width of curves and the corresponding trajectories for digits 0, 6, 8, 9 for a total of 20000 samples.

2.5.1.3 Synthetic MNIST with background

To train neural networks for generating writing trajectories from digits contained within images of more natural scenes, we prepared a database with digit images from different s-MNIST datasets pasted on different images of textured backgrounds. The digit images of size 40×40 pixels from different s-MNIST datasets were pasted at random positions on the background images of size 80×80 pixels.

We set the size of training images to 80×80 because this way we ensure that a portion of the image that is not covered by the 40×40 image of the digit is always present. Thus the network is trained to deal with the portions of images that are not covered by the images of digits.

To generate the synthetic MNIST with a background dataset for neural network training, we used images from the s-MNIST dataset, meaning we had 20000 different examples of digit shapes. The source of our background images was the LSUN-dataset [93], which is often used for scene classification. From this dataset, we took 300 images belonging to the scene 'Classroom' and resized them to 80×80 resolution. The size difference between

the size of the training images and the size of the digit images provides 40×40 possible different digit image positions in the training image. The resulting images were filtered with a Gaussian filter (see Fig. 2.15).

The training examples were prepared separately for each training epoch. For each digit shape from 14000 different shapes in the 70% training portion of the s-MIST dataset, we randomly selected a new background from 300 different backgrounds and inserted the digit shapes at a new random location in the area of 40×40 pixels. Even though there are only 14000 different outputs in the dataset that correspond to digit shapes, then there are $14000 \times 300 \times 40 \times 40$ for a total of 6720 million possible inputs.

Figure 2.15: Examples of training images from the s-MNIST with a background dataset.

The above-described dataset was used in our synthetic experiments. To apply the trained neural networks to real images captured by a robot, we generated a modified version of this dataset and re-trained the network. To generate the modified version, we used s-MNIST-GRAY images where the width of the curves forming the digits varied and the images of digits were variably grayscales instead of binary. The resulting 40×40 images were then pasted onto the "Classroom" scene of the LSUN-dataset images rescaled to 80×80 resolution and filtered as described above. Examples from the resulting dataset are shown in Fig. 2.16.

We also experimented with images containing background on neural networks trained with normalized AL-DMPs. Since the normalized AL-DMPs cannot represent digits with sharp corners, we generated the s-MNIST dataset with a background that has the s-MNIST(0,6,8,9) dataset for the source of the digit shape. To evaluate the use of the DMP trajectory loss function and the normalized AL-DMP path loss function also with variable input neural network architectures, we generated an additional dataset with a background onto which we pasted digits with variably scaled trajectories from the s-MNIST(0,6,8,9)-VTS dataset.

In order to train neural networks with normalized AL-DMPs that are suitable for processing real images captured by a robot, we generated the dataset with a background based on s-MNIST(0,6,8,9)-GREY. In this dataset with simulated images, the width of the curves forming the images of the digits varied, and the simulated images of the digits were computed as grayscale rather than binary images to correspond to the real images of the robot camera. The digit images are pasted onto the background in the same way as in other S-MNIST datasets with a background.

For experiments with input images containing a background, we have created a total of 6 different s-MNIST with a background datasets:

- **s-MNIST with background:** 2000 trajectories from the s-MNIST dataset for each digit, altogether 20000 digit shapes for output. 2000 images of digits from the s-MNIST dataset for each digit, altogether 20000 images of digits randomly pasted

Figure 2.16: Examples of training images from the s-MNIST-GRAY with a background dataset.

at 40x40 different locations in 300 different background LSUN 'classroom' images for altogether 9600 million possible input examples.

- **s-MNIST-GRAY with background:** 2000 trajectories from the s-MNIST-GRAY dataset for each digit, altogether 20000 digit shapes for output. 2000 images of digits from the s-MNIST-GRAY dataset for each digit, altogether 20000 digit images randomly pasted at 40x40 different locations in 300 different background LSUN 'classroom' images for altogether 9600 million possible input examples.
- **s-MNIST(0,6,8,9) with background:** 5000 trajectories from the s-MNIST dataset for digit 0, 6, 8, 9, altogether 20000 digit shapes for output. 2000 images of digits from the s-MNIST dataset for each digit, altogether 20000 digit images randomly pasted at 40x40 different locations in 300 different background LSUN 'classroom' images for altogether 9600 million possible input examples.
- **s-MNIST(0,6,8,9)-VTS with background:** 5000 variable time-scaled trajectories from the s-MNIST dataset for digit 0, 6, 8, 9, altogether 20000 digit shapes for output. 2000 images of digits from the s-MNIST dataset for each digit, altogether 20000 digit images randomly pasted at 40x40 different locations in 300 different background LSUN 'classroom' images for altogether 9600 million possible input examples.
- **s-MNIST(0,6,8,9)-GRAY with background:** 5000 trajectories from the s-MNIST-GRAY dataset for digit 0, 6, 8, 9, altogether 20000 digit shapes for output. 2000 images of digits from the s-MNIST-GRAY dataset for each digit, altogether 20000 digit images randomly pasted at 40x40 different locations in 300 different background LSUN 'classroom' images for altogether 9600 million possible input examples.

2.5.2 Neural network training

We used the PyTorch platform [94] in order to implement the proposed networks (Sec. 2.4). 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.

To train the MNIST CNN classifier described in Section 2.4.2, we used a stochastic gradient descent optimizer, a negative log-likelihood objective function, a batch size of

64, a learning rate of 0.01 with momentum 0.5, and trained for 10 epochs. This achieved a 98% accuracy, which we deemed sufficient for our purposes in extracting the trained convolutional layers for use in the encoder-decoder networks. In the case of the CIMEDNet architecture, we also experimented with either freezing the convolutional layer weights or training the entire network end-to-end. The results for these different training regimes are cataloged in Table 2.1.

For training the encoder-decoder networks, we used the Adam optimizer [95] 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 with DMPs, the DMP representation of handwriting trajectories comprised 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 (2.6) that represent each handwriting motion.

In all experiments with normalized AL-DMPs, the normalized AL-DMP representation of handwriting trajectories comprised 25 radial basis functions for every dimension of a two-dimensional writing trajectory. The weights of these basis functions form together with the start and end points of a planar movement and the initial derivatives (3×2 parameters) the full set of 56 DMP parameters (2.20) that represent each handwriting path.

When training with the DMP trajectory loss function defined in Eq. (2.36) or training with the normalized AL-DMP path loss function defined in Eq. (2.60), the trajectories do not need to be transformed into DMPs or normalized AL-DMPs. However, to compute these loss functions, we needed to conduct temporal integration of the output parameters for each training example. In addition, the gradients of the loss function must be computed by integrating the differential equations provided in Sec. 2.3. The most computationally expensive part in these calculations is the calculation of the weighted sum of radial basis functions. As this sum is the same across all differential equations, it needs to be calculated only once at each sampling step. Thus we add $\mathcal{O}(n)$ computational complexity, where n is the average number of time/spatial integration steps, to the initial computational complexity of training neural network directly with DMP or normalized AL-DMP parameters.

For more efficient training, we pretrained a neural network using the DMP parameters loss function (2.35) or the normalized AL-DMP parameters loss function (2.59) and then used the newly proposed loss functions for additional training. When using the DMP or AL-DMP parameters loss function, 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 Sec. 2.3. However, the training trajectories must be transformed to DMPs or normalized AL-DMPs to compute these loss functions.

The application of DMP trajectory and normalized AL-DMP path loss functions within the PyTorch framework requires the implementation of new custom PyTorch *autograd* functions, which should implement forward propagation and backpropagation for our custom-designed loss function. The resulting DMP *autograd* function computes the DMP trajectories by integrating differential equation system (2.1) – (2.3) in the forward propagation, while for backpropagation the loss function gradients are computed by integrating differential equations described in Sec. 2.3.1. The resulting normalized AL-DMP *autograd* function computes the normalized AL-DMP paths by integrating differential equation system (2.17) – (2.19) in the forward propagation, while for backpropagation the loss function

gradients are computed by integrating differential equations described in Sec. 2.3.2. We use the Euler integration method in both cases of forward and backward propagation, 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.

2.5.2.1 Number of training parameters and avoidance of overfitting

The number of parameters in our fully connected IMEDNet neural network for DMP output is rather high, as it contains 6381335 free parameters, all of which have to be calculated during the training. A high number of parameters slows down the training of the neural network and could lead to overfitting of the model. CNN architectures with their parameter sharing usually achieve comparable performance with significantly fewer parameters. Our first CNN proposal was the CIMEDNet version without fully connected layers in the encoder part. This architecture with its 27615 free parameters has drastically fewer parameters than IMEDNet, but does not achieve the performance of IMEDNet (Sec. 2.5.4). However, the CIMEDNet version with fully connected layers in the encoder part can match the IMEDNet performance and still has almost nine times fewer parameters with its 720955 free parameters.

The training data without background typically consisted of 20000 data points, of which 17000 were used for optimization and the rest for testing. In our experiments, the dimension of the neural network DMP output was 55 as there were 55 DMP parameters. Thus we obtained 17000*55 = 935000 equations. This number is higher than the number of parameters in CIMEDNet (720955) but still less than the number of parameters in IMEDNet (6381335). 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 426955, which is much less than the number of equations that are considered during training (935000). For CIMEDNet without fully connected layers in the encoder part, the number of free parameters is even reduced to 17815. 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. 2.18 shows that CIMEDNet converged significantly faster.

When we experimented with inputs of different sizes (Sec. 2.5.7), we first used the variable input architecture CIMEDNet+ with only 8015 free parameters. This architecture was not able to achieve the desired performance, so we proposed the VIMEDNet architecture. The ViMEDNet architecture with its 642260 free parameters is close to the number of parameters in the CIMEDNet architecture for the 40×40 pixel input size. In these experiments we generated training inputs separately for each training period. We pasted 14000 different digit shapes at locations in the area of 40×40 pixels in 300 different background images. This results in $14000 \times 300 \times 40 \times 40 = 6720$ million possible input images. Each input example provides 55 equations that together overconstrain the optimization problem. Nevertheless, we used the validation stop as a control measure for the overfitting.

In the same experiment, we also compared the CIMEDNet architectures trained for different input sizes. Note that compared to CIMEDNet+ and VIMEDNet, CIMEDNet cannot be trained on one size and used on another, but we have to generate and train a neural network for each input size separately, which also changes the number of free parameters depending on the input size. We have trained CIMEDNet for input sizes of 60×60 pixels with 8880815 free parameters, 80×80 pixels with 26640815 free parameters, and 120×120 pixels with 90980815 free parameters. The number of free parameters grows exponentially with the increasing input size. However, due to the large amount of data we provide with the generation of new combinations for each training epoch, the optimization problem was still overconstrained in all cases.

When we use normalized AL-DMPs as output for training the neural network, its output size is 56 instead of 55. Connecting an additional output neuron in the deep neural network adds additional free parameters. Since the output neuron is connected with a weight to each neuron in the previous layer, the number of additional free parameters is equal to the number of neurons in the previous layer plus one bias parameter. In the case of IMEDNet and CIMEDNet, the model with the additional output neuron gains 36 additional free parameters (35 weights and 1 bias) and in the case of VIMEDNet, the model gains 46 additional free parameters (45 weights and 1 bias). At the same time, this one additional output neuron provides one additional equation per training example, giving a total of 56 equations per training example for the entire model. With 16000 training examples, the additional output neuron added 16000 equations to the number of total equations considered during training. When we compare training with normalized AL-DMP loss functions to training with DMP loss functions, we added 16000 additional equations for training with normalized AL-DMPs and only 36 or 46 additional free parameters in the model. This ensures that training with normalized AL-DMP loss functions is even more overconstrained than training with DMP loss functions.

2.5.3 Evaluation of results

We used dynamic time warping [96] to compare the distance between DMP or normalized AL-DMP paths, which were generated by the neural network, and the testing paths. 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 DMP trajectory loss function (2.36) and normalized AL-DMP path loss function (2.60) because dynamic time warping does not result in a differentiable loss function.

Each testing path is sampled at equidistant points, with the sampling step Δs equal to approximately 0.2 pixels, where the exact sampling step is computed differently for each path so that the path endpoints are always included. The constant sampling step is fine because our data are synthetic and therefore do not contain noise.

In some experiments, we compared the DTW result for different parameters with statistical tests. We used a two-sample t-test in Section 2.5.5 and ANOVA with multiple comparisons in Section 2.5.6. A two-sample t-test was conducted 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, 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.

A common statistical test to check whether the mean values of the results of different experiments are equal or not is the analysis of variance (ANOVA). However, ANOVA only checks whether the mean values are the same for all experiments or not. It provides no further information about which individual pairs of test results mean values are different. In order to find statistically significant differences between the mean values of DTW results for different experiments, we applied two-way ANOVA based on the multiple comparisons of experimental mean values with Tukey's honestly significant difference criterion. We used MATLAB functions from [97] to compute the described statistical tests.

2.5.4 Robustness of the proposed neural networks to noise

It is not trivial to design a good neural network architecture that is suitable for a certain application. Small and simple neural networks might not have sufficient representational power for the problem that needs to be solved, whereas large and complex neural networks are slow to train and might have problems with vanishing gradients. This can lead to bad generalization results.

In this section, we present a study [98] performed to identify a suitable neural network design for realizing a mapping of digit images to the corresponding handwriting trajectories. We compared a fully connected neural network IMEDNet (Sec. 2.4.2) and two different convolutional architectures of CIMEDNet (Sec. 2.4.1). Each of the CIMEDNet architectures were trained with two separate training regimes in which the convolutional layer weights were either frozen or the models were trained end-to-end, respectively. When using the frozen convolutional layers, we pretrained these layers in the CNN digit classifier.

In the experiment we tested generalization results of neural networks on the same type of data. We also evaluated the robustness of the proposed networks to noise when generating standard DMPs. Noise robustness is one specially desired ability for the neural networks that are used in real-world experiments (Sec. 2.5.8). In this set of experiments, the simple DMP parameters loss function (2.35) was used. After training on the noiseless s-MNIST dataset, each of the models was tested on all five of the noiseless and noisy s-MNIST datasets. The quantitative results are presented in Tab. 2.1, while qualitative results for the selected samples are presented in Fig. 2.17.

As can be seen in Tab. 2.1, the CIMEDNet model that is trained end-to-end significantly outperforms the IMEDNet model on both the noiseless s-MNIST dataset and on most of the noisy s-MNIST datasets. Only for the dataset with motion blur noise are the mean values of the IMEDNet and CIMEDNet (end-to-end) results almost equal, nevertheless the variance of the CIMEDNet results is still smaller. We believe that this may be due to the fact that motion blur can significantly distort the overall object shape and edge profiles. Given that convolutional neural networks function by exploiting hierarchies of image filters, which are often heavily represented by edge detectors, this may impact on their effectiveness in such circumstances. The CIMEDNet that was trained with frozen convolutional layers also fared well, beating the IMEDNet model on the same noisy datasets despite not scoring as well on the noiseless dataset. This indicates that the feature detectors in the convolutional layers allow for more robust generalization whereas fully-connected layers are more inclined to overfit.

The qualitative results in Fig. 2.17 are also interesting. Here, only results for the IMEDNet model and the best-performing CIMEDNet model that was trained end-to-end are shown for comparison on the selected samples from the noiseless s-MNIST dataset and three of the noisy datasets: s-MNIST-AWGN-19.0-SNR, s-MNISTAWGN-9.5-SNR and s-MNIST-MB. The trained neural networks often perform surprisingly well given that they were not trained or fine-tuned on the noisy data. Both neural networks produce

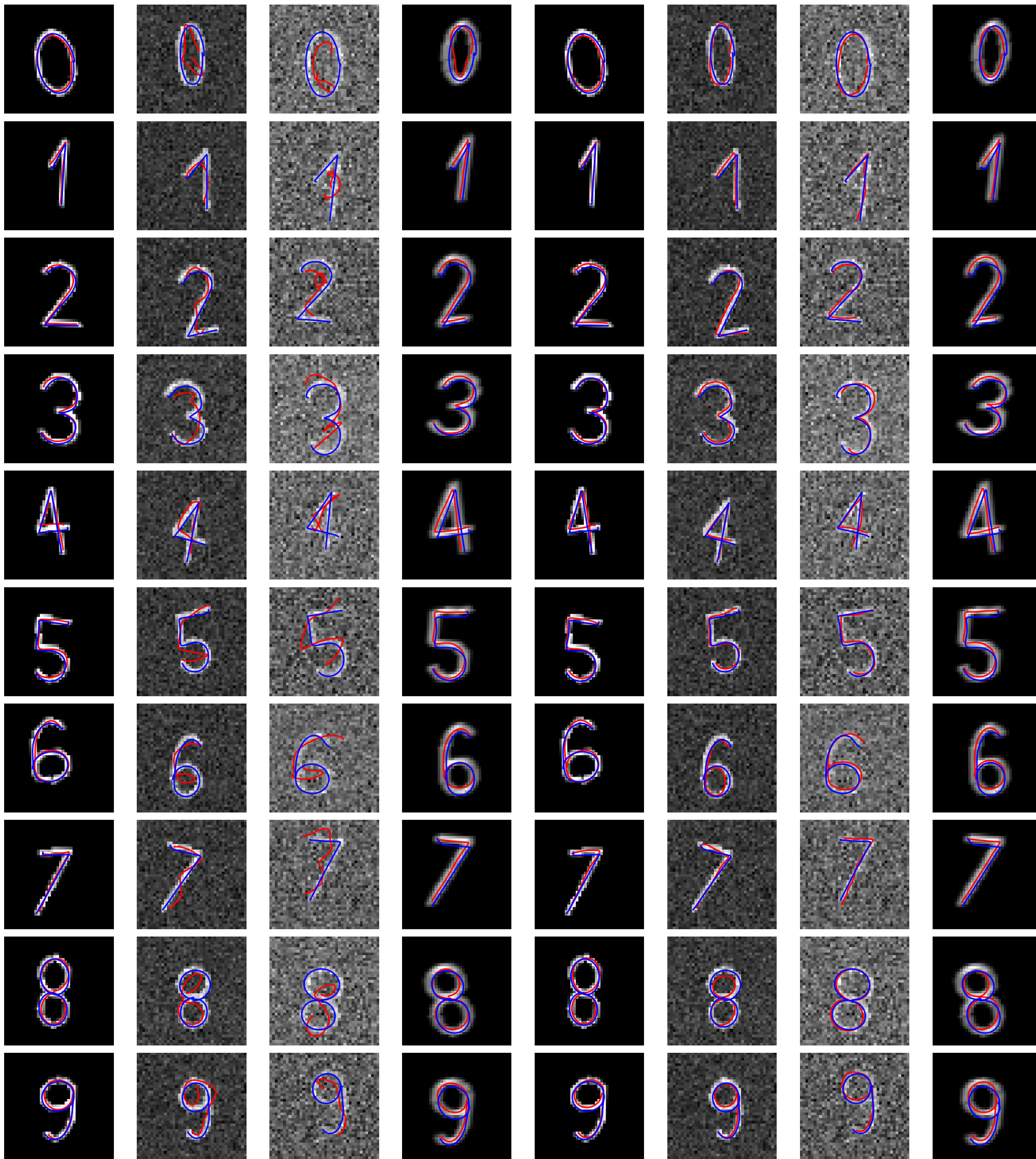


Figure 2.17: Example results for two different neural networks: IMEDNet in the four leftmost columns and CIMEDNet in the four rightmost columns, presented on various different noise profiles of the s-MNIST data. Columns 1 and 5 show results for each of the two models on s-MNIST without any noise, columns 2 and 6 show results for AWGN noise with a signal-to-noise ratio of 19.0, columns 3 and 7 show results for AWGN noise with a signal-to-noise ratio of 9.5 and columns 4 and 8 show the results for motion blur noise. The original trajectories are shown in blue, while the DMP trajectories calculated by the networks are shown in red. Test inputs in matching dataset columns are identical for comparison. None of the presented samples or noisy datasets were used for training.

Table 2.1: DMP reconstruction statistics for comparing robustness of different neural network architectures to noise. The results are in pixels. The best result for each dataset is highlighted in boldface.

	s-MNIST	**s-MNIST AWGN 19.0-SNR**	**s-MNIST AWGN 9.5-SNR**	**s-MNIST MB**	**s-MNIST RC-AWGN**
IMEDNet (End-to-End)	0.215 ± 0.083	0.559 ± 0.198	1.660 ± 0.597	**0.354 ± 0.146**	2.323 ± 0.765
CIMEDNet - fully conv. enc. (Frozen Conv.)	0.354 ± 0.201	0.605 ± 0.332	1.789 ± 0.921	0.465 ± 0.281	2.320 ± 0.947
CIMEDNet - fully conv. enc. (End-to-End)	0.554 ± 0.420	0.861 ± 0.532	2.046 ± 0.937	0.668 ± 0.488	2.956 ± 1.072
CIMEDNet (Frozen Conv.)	0.262 ± 0.103	0.541 ± 0.200	1.476 ± 0.546	0.473 ± 0.245	2.185 ± 0.756
CIMEDNet (End-to-End)	**0.194 ± 0.076**	**0.358 ± 0.139**	**1.017 ± 0.447**	**0.357 ± 0.124**	**1.935 ± 0.657**

highly legible writing trajectories that closely match the actual trajectories in the case of the s-MNIST-MB dataset. The CIMEDNet model is demonstrably superior to IMEDNet in many cases with the s-MNIST-AWGN-19.0-SNR and s-MNIST-AWGN-9.5-SNR data, producing much more legible results and demonstrating the robustness of the convolutional layers in dealing with even extremely high noise levels.

From the experimental results we have concluded that for the following experiments, two network designs are best suited, IMEDNet and CIMEDNet with fully connected layers in the encoder part, which is trained end-to-end.

2.5.5 Performance of the proposed DMP-based loss functions

First we compare DMP parameters loss function (2.35) and DMP trajectory loss function (2.36), which were both used for training of neural networks that output DMP parameters. The first objective function directly compares the DMP parameters, which have no physical meaning, whereas the other objective function compares the distance between the training trajectories and the DMP-generated trajectories. In these experiments, we used the IMEDNet and the second version of CIMEDNet architectures. They were chosen because of their robustness to noise as demonstrated in Section 2.5.4. 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 the s-MNIST dataset, we used 1700 training pairs of images and writing trajectories for each digit, which altogether makes 17000 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, 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. 2.18, which depicts the convergence of training and validation errors. The graphs show that IMEDNet needs at least twice as many epochs for both loss functions and for both data sets until convergence.

Table 2.2: DMP reconstruction statistics for DMPs computed by IMEDNet and CIMEDNet trained by loss functions (2.35) and (2.36) 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**
IMEDNet		
DMP parameters loss function (2.35)	0.215 ± 0.084	0.322 ± 0.137
DMP trajectory loss function (2.36)	0.134 ± 0.045	0.231 ± 0.104
t-test comparison results	$t(5998) = 46.83$, $p < 0.001$	$t(5998) = 29.12$, $p < 0.001$
CIMEDNet		
DMP parameters loss function (2.35)	0.194 ± 0.076	0.389 ± 0.183
DMP trajectory loss function (2.36)	0.131 ± 0.063	0.319 ± 0.142
t-test comparison results	$t(5998) = 35.37$, $p < 0.001$	$t(5998) = 17.82$, $p < 0.001$

The statistics of reconstruction quality are shown in Tab. 2.2. The results obtained with both datasets and both network architectures confirmed that by using the physically meaningful DMP trajectory loss function (2.36), we obtain significantly better results than with the simpler DMP parameters loss function (2.35). 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 errors 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. 2.19 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. 2.20.

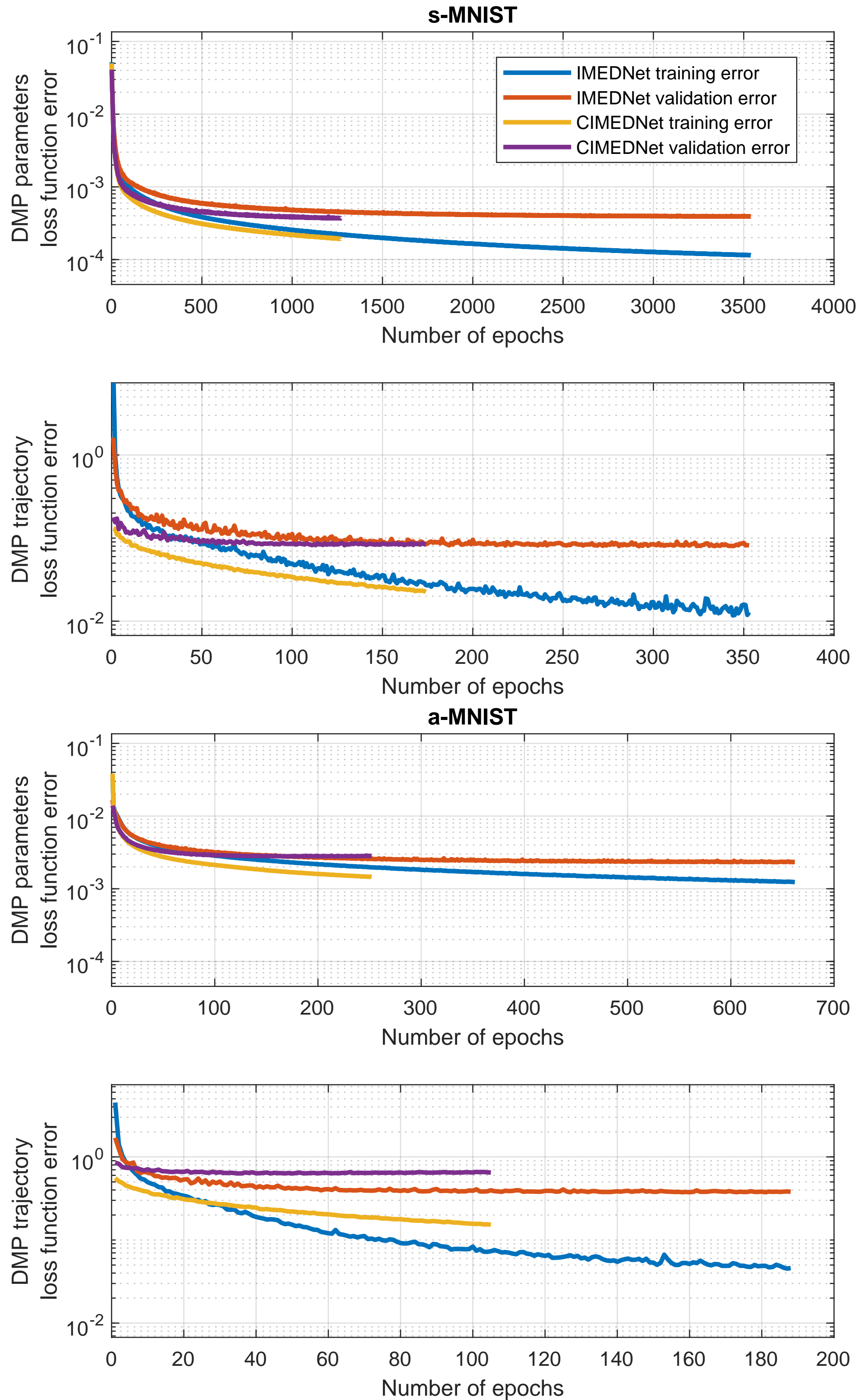


Figure 2.18: Convergence of training and validation errors for evaluation of DMP-based loss functions. We present results for IMEDNet and CIMEDNet, both applied to the s-MNIST and the a-MNIST dataset. In all cases the training was stopped when the validation error has not dropped for 60 consecutive epochs.

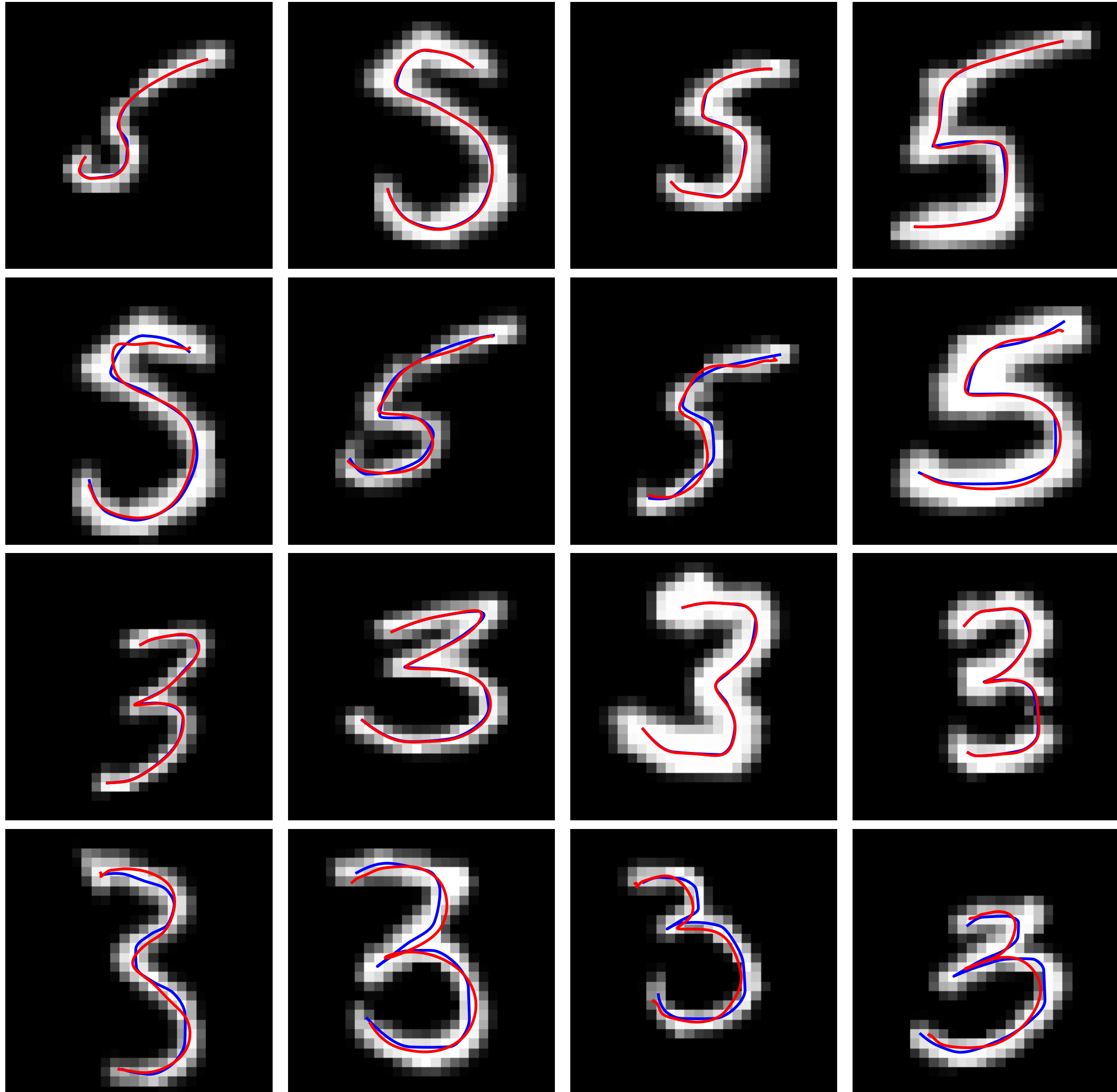


Figure 2.19: Example reconstruction results for digits from the a-MNIST dataset. The manually generated and transformed trajectories are shown in blue, while the DMP trajectories calculated by the CIMEDNet are shown in red. These data were used only for testing, not for training. Hence these results demonstrate the generalization performance of the proposed encoder-decoder network.

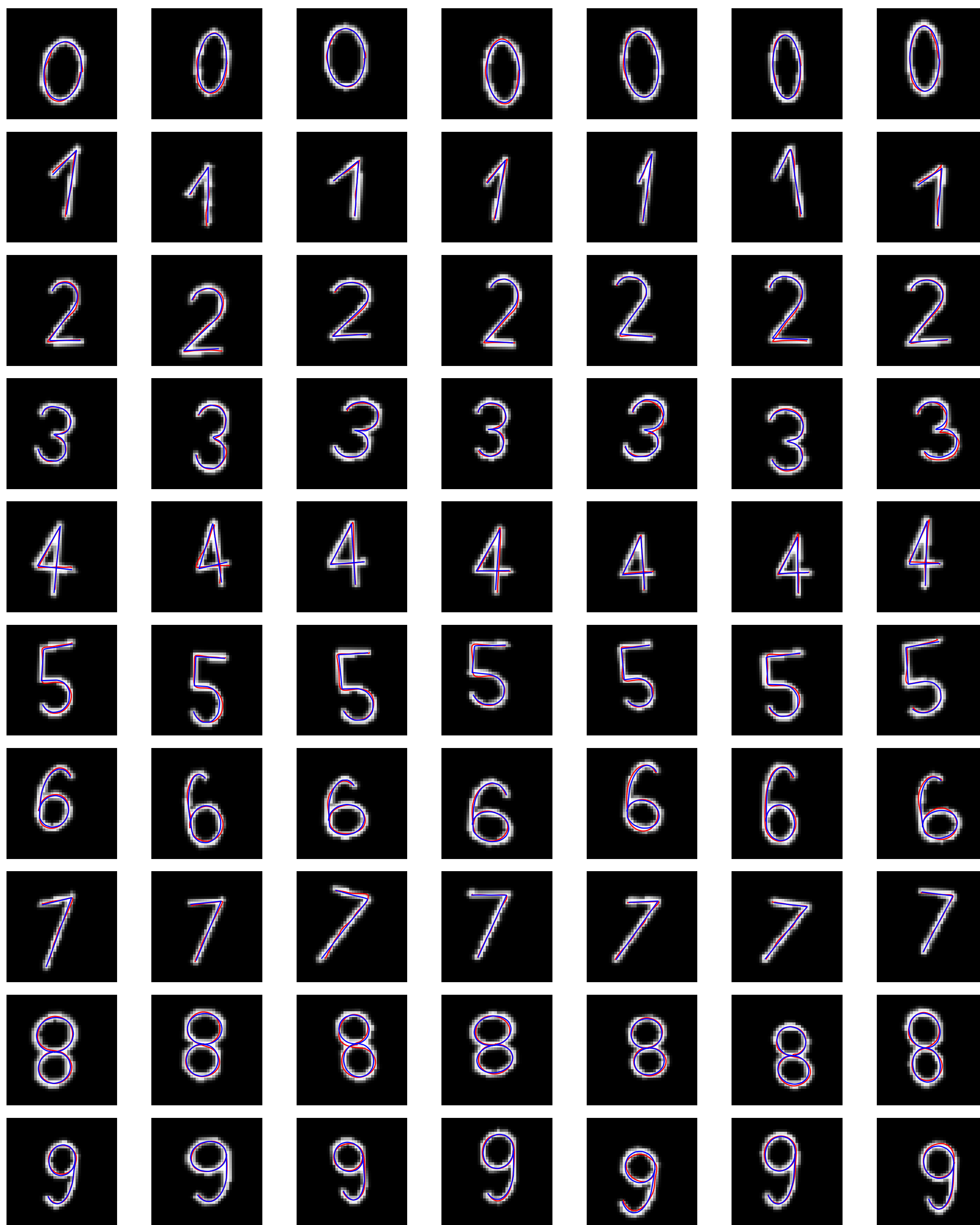


Figure 2.20: Example synthetic data showing the images of digits and the corresponding handwriting trajectories. The original trajectories are shown in blue, while the DMP trajectories calculated by CIMEDNet are shown in red. CIMEDNet is able to reconstruct handwriting trajectories well even though these images and trajectories were not used for training.

2.5.6 Comparing performance of the proposed DMP-based and normalized AL-DMP-based loss functions

Next we compare the effectiveness of two different motion representations, DMPs and normalized AL-DMPs, for the generation of writing trajectories with deep neural networks. We used 4 different loss functions to train the neural networks; DMP parameters loss function (2.35) and DMP trajectory loss function (2.36) for networks generating DMP parameters and normalized AL-DMP parameters loss function (2.59) and normalized AL-DMP path loss function (2.60) for networks generating normalized AL-DMP parameters as output.

For evaluation, we used the s-MNIST(0,6,8,9) dataset, this is a dataset limited to 4 digits: 0, 6, 8, and 9. The reason for this is that arc-length DMPs cannot represent trajectories with sharp corners because sharp corners cause discontinuities in arc-length derivatives. While this limitation could be overcome by segmenting digits with sharp corners into parts without sharp corners, this would not make a significant contribution to the understanding of the effectiveness of both representations.

Although CIMEDNet contains significantly less parameters than IMEDNet, the performance of both network architectures was similar when using the same representation and loss function. Note, however, that CIMEDNet can be trained significantly faster than IMEDNet.

The results are shown in Tab. 2.3. These results show the distance of the writing trajectories generated by DMPs and AL-DMPs, which were computed by the respective neural networks, to the writing trajectories from the testing data using the dynamic time warping algorithm. For the statistical test we used ANOVA with multiple comparisons.

The best results were obtained when using normalized AL-DMP path loss function (2.60) and the second best results with DMP trajectory loss function (2.60). The reason for this is the same as in the previous experiment for the DMP representation. Unlike normalized AL-DMPs parameters loss function (2.59), which measures the distance between the normalized AL-DMP parameters that have no physical meaning, normalized AL-DMP path loss function (2.60) measures the real spatial distance between the digits. The performance of normalized AL-DMP is better because the timing of motion does not affect the result. Consequently, in the normalized AL-DMP representation, the points are evenly distributed along the path, whereas in the DMP representation, the trajectory points are spatially more distant and less numerous in parts with high velocity. Thus the precision of approximation is lower with DMP-based neural networks in high velocity parts of trajectories because there are less sampling points.

While the performance of estimation with normalized AL-DMP parameters loss function (2.59) and its DMP equivalent DMP parameters loss function (2.35) was worse for both network architectures, the DMP representation outperformed the normalized AL-DMP representation in this case. We believe that this reversal of performance is due to the additional parameters of normalized AL-DMPs, i. e. the starting velocities, which have a different scale than position parameters and must be normalized. Since there are more parameters in normalized AL-DMPs, such scaling problems negatively affect the performance of networks with AL-DMP outputs compared to the networks with DMP outputs.

The above results already show the advantages of AL-DMPs in combination with path loss function (2.60) over standard DMPs. But the most important advantage of AL-DMPs is their independence from the temporal schedules. When the proposed neural networks are trained using data with inconsistent temporal schedules, e. g. temporally scaled s-MNIST(0,6,8,9)-VTS data, DMP-based neural networks are no longer able to learn handwritten digits precisely, as evident from Table 2.3. On the other hand, the results of normalized AL-DMPs are not affected by temporal inconsistencies at all. The results are by

Table 2.3: Reconstruction statistics for neural networks with DMP and normalized AL-DMP parameters at the output when using different loss functions. The results are in pixels. Result pairs that are not statistically significantly different ($p > 0.001$) are marked with * or **.

	s-MNIST(0,6,8,9)	**s-MNIST(0,6,8,9)-VTS**
IMEDNet		
DMP parameters loss function (2.35)	0.116 ± 0.028 *	/
DMP trajectory loss function (2.36)	0.088 ± 0.016 **	2.279 ± 0.486
Normalized AL-DMP parameters loss function (2.59)	0.264 ± 0.074	/
Normalized AL-DMP path loss function (2.60)	0.059 ± 0.014	0.059 ± 0.014
CIMEDNet		
DMP parameters loss function (2.35)	0.116 ± 0.029 *	/
DMP trajectory loss function (2.36)	0.091 ± 0.020 **	2.220 ± 0.516
Normalized AL-DMP parameters loss function (2.59)	0.182 ± 0.087	/
Normalized AL-DMP path loss function (2.60)	0.066 ± 0.017	0.066 ± 0.017

an order of magnitude better with normalized AL-DMPs than with DMP-based networks in this case.

The progress of the training process is illustrated in Fig. 2.21, which shows the convergence of training and validation errors. The training process on the s-MNIST(0,6,8,9)-VTS dataset for loss functions based on normalized AL-DMPs is not shown because it is the same as for s-MNIST(0,6,8,9). If we look at the error of DMP trajectory loss functions for both neural networks trained on s-MNIST(0,6,8,9)-VTS, we can already suspect the poor generalization shown in Table 2.3 from the difference and step divergence between the validation and training results.

In the previous experiment, we have already trained IMEDNet and CIMEDNet with the DMP parameters loss function and the DMP trajectory loss function on the s-MNIST dataset with all ten digits. By comparing these results (Tab. 2.2) with the results of this experiment, in which we trained both neural networks and loss functions on the s-MNIST(0,6,8,9) dataset with only four digits, we can confirm the expected trend that a simpler dataset with only four digits and more examples per digit gives better training results.

Fig. 2.22 presents the CIMEDNet reconstruction results for training with the DMP trajectory loss function and the normalized AL DMP path loss function. The DMP trajectories (blue crosses) or normalized AL-DM paths (red x-s) generated by CIMEDNet are plotted with points to show the difference between their spatial distributions. Uniformly time-distributed points of DMP trajectories have a high spatial density near the start and end point of the trajectory where velocity is low. In this area, the differences between the original trajectories (green dotted line) and the DMP trajectories are small because the high number of points on these trajectory sections gives priority to these sections during

training. This has a negative effect on the high-speed sections of the trajectories. Due to the lack of points, these sections are less important for optimization during training. These sections show larger deviations between the original trajectories and the trajectories generated by the neural network. This problem becomes even more evident with the results of our previous DMP trajectory experiment presented in Fig. 2.20, where the trajectories usually have a really good precision at the beginning and the end of the trajectory, but the largest deviation is in the middle of the trajectories. The normalized AL-DMP path in Fig. 2.22 has spatially evenly distributed points and a more constant shape deviation over the entire trajectory, which lowers the average error of the entire trajectory. This advantage of the normalized AL-DMP framework over the DMP framework is also noticeable when comparing results in Fig. 2.23 and 2.24 in Sec. 2.5.7.

DMP trajectories trained on s-MNIST(0,6,8,9)-TS (magenta circles) still have good predictions of start and end points because these DMP parameters are not dependent on temporal course of motion. Where the DMP trajectory loss function fails is the prediction of DMP weights. DMP weights encode the shape of the trajectory. Even if the general shape of the digit is still recognizable, the accuracy is far worse than in any other experiment. In contrast to this, in the case of CIMEDNet training with normalized AL-DMP path loss function using s-MNIST(0,6,8,9)-TS dataset, the temporal scaling does not affect the spatial weights and the results are consistent with previous results of training with s-MNIST(0,6,8,9) (red x-s).

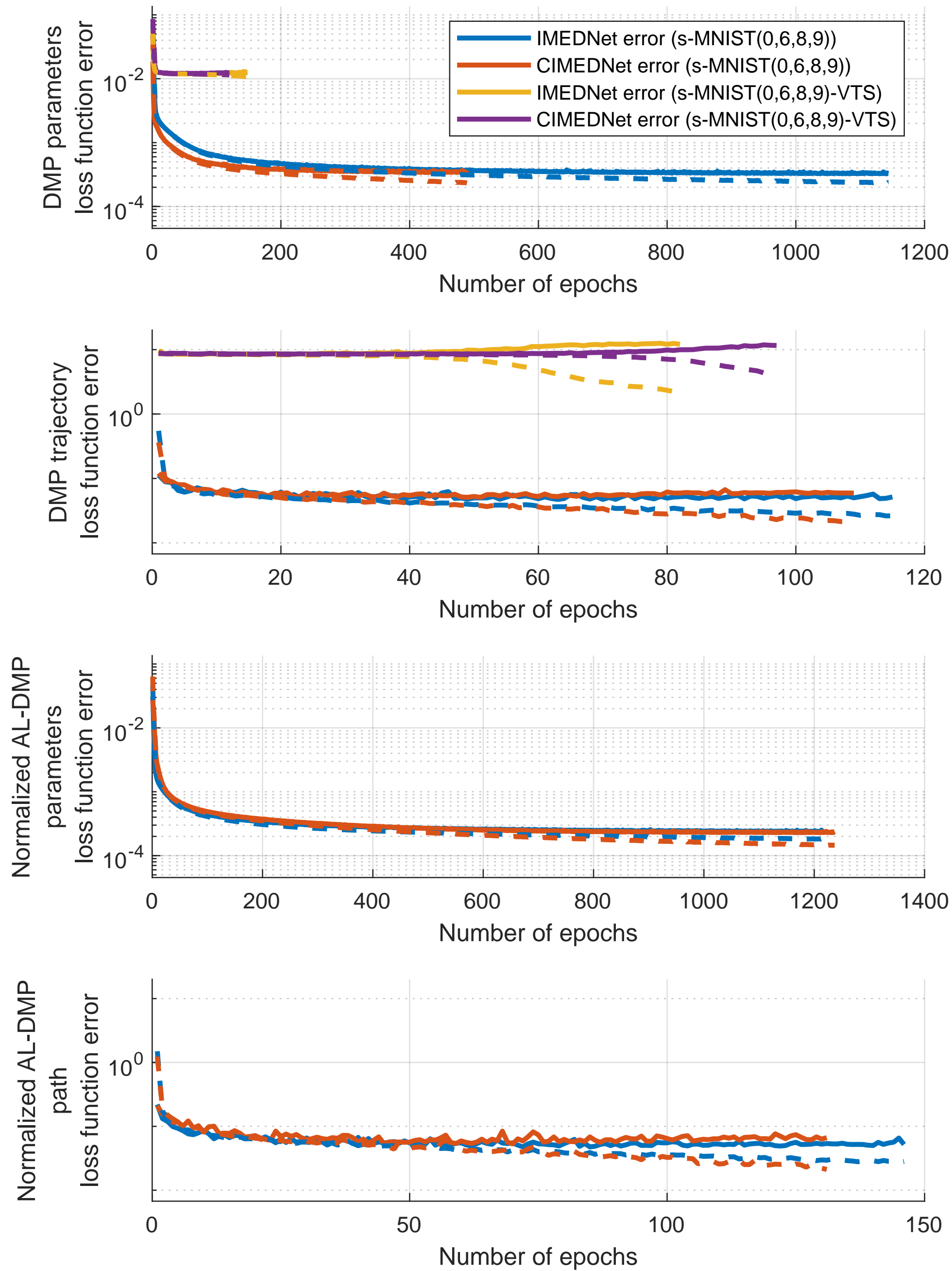


Figure 2.21: Convergence of training errors (solid lines) and validation errors (dashed lines) for comparing DMP-based and normalized AL-DMP-based loss functions. We present results for IMEDNet and CIMEDNet, both trained on datasets s-MNIST(0,6,8,9) and s-MNIST(0,6,8,9)-VTS with loss functions based on DMPs and normalized AL-DMPs. In all cases, the training was stopped when the validation error did not drop for 60 consecutive epochs.

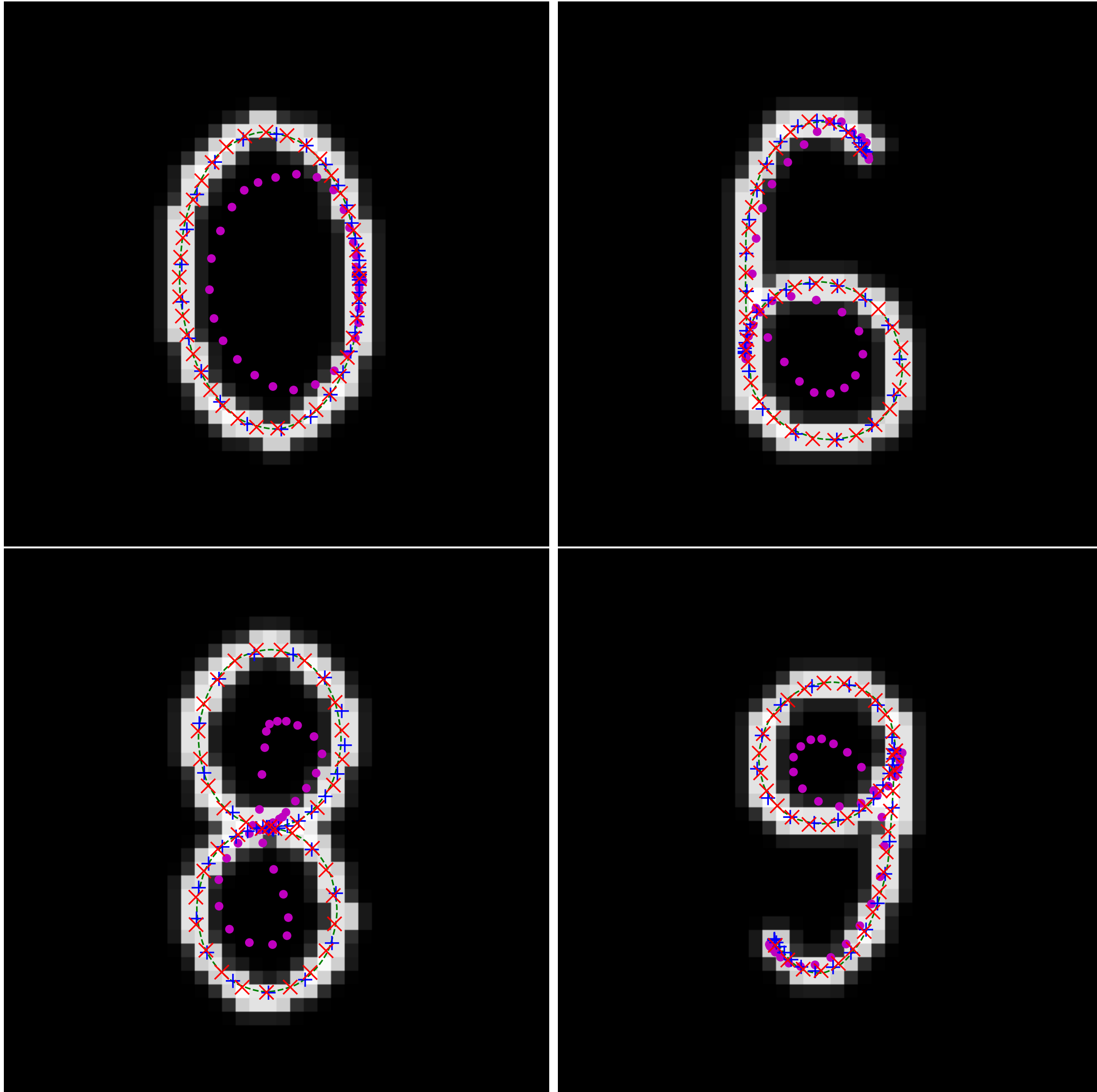


Figure 2.22: CIMEDNet reconstruction results for training with DMP trajectory and normalized AL-DMP path loss functions. The original trajectories for digits 0, 6, 8, 9 are shown in green dashed line, while the points on the normalized AL-DMP paths generated by the CIMEDNet trained with loss function (2.60) and s-MNIST(0,6,8,9) are shown with red x. These data were used only for testing, not for training. CIMEDNet trained with DMP trajectory loss function (2.36) generated DMP trajectories presented with blue crosses when trained on s-MNIST(0,6,8,9) and DMP trajectories presented with magenta circles when trained on s-MNIST(0,6,8,9)-TS.

2.5.7 Using input images of different sizes

In the next experiment, we tested the performance of different neural network architectures when processing different size digit images with a background. For training, we used DMP trajectory loss function (2.36) when the network's outputs were DMP parameters and normalized AL-DMP path loss function (2.60) when the network's outputs were normalized AL-DMP parameters. We used the s-MNIST datasets with a background for training (see Section 2.5.1.3). These datasets contained background images of a fixed size (80×80) with digit images of 40×40 pixels pasted on the background at random locations. Thus, a significant portion (75 %) of each digit image contained a background. In the first experiment, we used only DMPs for training. Therefore, we could use a dataset *S-MNIST with background* in which the pasted digit images of 40×40 pixels contained all ten digits. In the second experiment, in which we also compared training with DMPs and training with normalized AL DMPs on neural networks with variable input sizes, we were limited to digits 0, 6, 8, 9 because of AL-DMPs. In this case, we used datasets *S-MNIST(0,6,8,9) with background* and *S-MNIST(0,6,8,9)-VTS with background.* The background images used for training were different from the ones used for testing.

Results in Tab. 2.4 for training with DMP trajectory loss function (2.36) show that the performance of VIMEDNet is for an order of magnitude better than the CIMEDNet+. As is also evident from Fig. 2.23, the VIMEDNet is able to successfully reproduce good quality writing movements for all tested image sizes, even if it was trained only on the 80×80 pixel images.

For comparison, we also trained and tested the initial CIMEDNet architecture on the datasets with different image sizes and backgrounds. As the CIMEDNet cannot deal with variable size images, we had to train a different neural network for each image size. The results in Tab. 2.4 show that the performance of CIMEDNet is good for small inputs. However, as the input image size increases, the performance drops drastically because of the transition layer between convolution and fully connected layers in the encoder. The number of parameters in this layer depends quadratically on the input size, which makes training more difficult with larger inputs. The increasing number of parameters also slows down training and requires more GPU memory during training.

We repeated the same experiment for all network architectures. Only this time, in addition to training with the DMP trajectory loss function (2.36), we also trained with the normalized AL-DMP path loss function (2.60), either on *S-MNIST(0,6,8,9) with background* or *S-MNIST(0,6,8,9)-VTS with background* datasets. The results of the experiment

Table 2.4: Reconstruction statistics for input images of different sizes for neural networks trained with DMPs and *S-MNIST with background.* The size of the test set is 3000 and the results are in pixels.

	CIMEDNet+	**VIMEDNet**	**CIMEDNet**
60×60	2.530 ± 1.414	0.319 ± 0.109	0.494 ± 0.189
80×80	2.528 ± 1.410	0.287 ± 0.092	4.004 ± 1.996
120×120	2.529 ± 1.408	0.277 ± 0.079	4.930 ± 1.119
200×200	2.540 ± 1.414	0.277 ± 0.087	/
360×360	2.533 ± 1.412	0.280 ± 0.112	/
680×680	2.535 ± 1.409	0.281 ± 0.108	/

are displayed in Tab. 2.5. Since the variable time scaling of the trajectory does not influence the results from training with the normalized AL-DMP path loss function, we present the results for both data sets together in the first part of the table. The results for training with the DMP trajectory loss function are presented separately in the second and third part.

Results for training with both loss functions on *S-MNIST(0,6,8,9) with background* confirm the result of the experiment with the DMP trajectory loss function from Tab. 2.4. CIMEDNet+ and VIMEDNet, trained on 80×80 pixel input images, are able to reproduce trajectories computed from input images of different sizes. As expected, the general results in Tab. 2.5 are better than in Tab. 2.4 as a consequence of the simpler training dataset. CIMEDNet, trained separately for each input size, is well suited for small output sizes, but its performance decreases significantly as the input size starts growing. The VIMEDNet has a better performance than the CIMEDNet+, but the difference between them when using normalized AL-DMP path loss function is smaller than in the experiment with DMPs trajectories. Nevertheless, the VIMEDNet performance slightly degrades as the difference between the training and test image size increases. This could be the consequence of generating training backgrounds by resizing the original images, thus dropping the resolution and reducing the amount of detail contained in the training images.

If we further compare the results of training with the normalized AL-DMP path loss function and the results of training with the DMP trajectory loss function, we can see

Table 2.5: Reconstruction statistics for input images of different sizes for neural networks trained with normalized AL-DMPs. The size of the test set is 3000 and the results are in pixels.

		CIMEDNet+	**VIMEDNet**	**CIMEDNet**
Normalized AL-DMP loss function S-MNIST(0,6,8,9)-VTS with background	60×60	0.223 ± 0.086	0.146 ± 0.049	0.247 ± 0.093
	80×80	0.222 ± 0.086	0.138 ± 0.043	0.419 ± 0.164
	120×120	0.223 ± 0.087	0.135 ± 0.041	2.919 ± 2.239
	200×200	0.225 ± 0.100	0.136 ± 0.043	/
	360×360	0.223 ± 0.084	0.138 ± 0.050	/
	680×680	0.225 ± 0.088	0.139 ± 0.049	/
DMP trajectory loss function S-MNIST(0,6,8,9) with background	60×60	1.561 ± 0.720	0.149 ± 0.077	0.477 ± 0.198
	80×80	1.559 ± 0.718	0.136 ± 0.074	0.709 ± 0.261
	120×120	1.556 ± 0.720	0.134 ± 0.061	5.640 ± 1.042
	200×200	1.568 ± 0.736	0.139 ± 0.089	/
	360×360	1.556 ± 0.717	0.166 ± 0.162	/
	680×680	1.553 ± 0.714	0.199 ± 0.215	/
DMP trajectory loss function S-MNIST(0,6,8,9)-VTS with background	60×60	3.086 ± 1.005	2.398 ± 0.554	2.223 ± 0.472
	80×80	3.140 ± 1.027	2.347 ± 0.496	4.613 ± 1.761
	120×120	3.189 ± 1.046	2.326 ± 0.483	5.271 ± 1.018
	200×200	3.260 ± 1.067	2.325 ± 0.481	/
	360×360	3.233 ± 1.041	2.324 ± 0.475	/
	680×680	3.159 ± 1.044	2.317 ± 0.474	/

that the results of CIMEDNet+ and CIMEDNet are better for normalized AL-DMPs. This is the consequence of the already mentioned advantages of the normalized AL-DMP path loss function. For VIMEDNet, the results for both loss functions in the range of 60×60 to 200×200 pixels are almost the same. Nevertheless, VIMEDNet trained with the normalized AL-DMP loss function has a better generalization performance between different input sizes and is able to keep relatively good performance also to input sizes of 680×680.

When comparing the results of training neural networks with variable size input images with DMP trajectory loss function or normalized AL-DMP path loss function on *s-MNIST(0,6,8,9)-VTS* in Tab. 2.5, we again confirm the advantages of AL-DMPs in combination with path loss function (2.60). Their independence from the temporal schedules enables training with inconsistent temporal schedules when DMP-based neural networks are no longer able to learn handwritten digits precisely. The results are again by an order of magnitude better with normalized AL-DMPs than with DMP-based networks.

As evident from Fig. 2.24, the VIMEDNet trained with the normalized AL-DMP path loss function is able to successfully generate writing trajectories for all tested image sizes. When comparing AL-DMP paths with the DMP-generated paths in Fig. 2.23, we can again notice the problem of the DMP trajectory loss function, which is more optimized at the start and the end of the trajectory, while in the middle of the trajectory, the deviation from the original shape is larger. The deviations of AL-DMP path from the original path are more evenly distributed throughout the whole digit shape.

The progress of the training process is shown in Fig. 2.25, where the convergence of training and validation errors are presented. The VIMEDNet has the fastest and most persistent convergence. If we look at the training progress of CIMEDNet architectures with fixed-size inputs, we can see that with the increasing size of input images, the learning ability of CIMEDNet decreases to the level that it can learn almost nothing. This is the consequence of the increasing size of the transition layers between the convolutional and the fully connected part of the neural network. Not only does the number of parameters increase, but also the area of the paper with a digit becomes much smaller than the image size. This means that for each example, the relevant information that flows through the large transition layer only passes through a few local neurons. Compared to CIMEDNet with 60×60 input, in CIMEDNet with 120×120 input, the individual neurons in the transition layers activates in less training examples. The same is true for the relevant gradient update during learning. This slows down or even effectively stops the learning process.

Figure 2.23: Visualization of writing trajectories generated by VIMEDNet trained with DMP trajectory loss function. The network can process images of different sizes at inputs. In this experiment, 40×40 digit images were pasted onto $60 \times 60, 80 \times 80, 120 \times 120, 200 \times 200, 360 \times 360$ and 680×680 pixel background images. The neural network was trained with 80×80 pixel images.

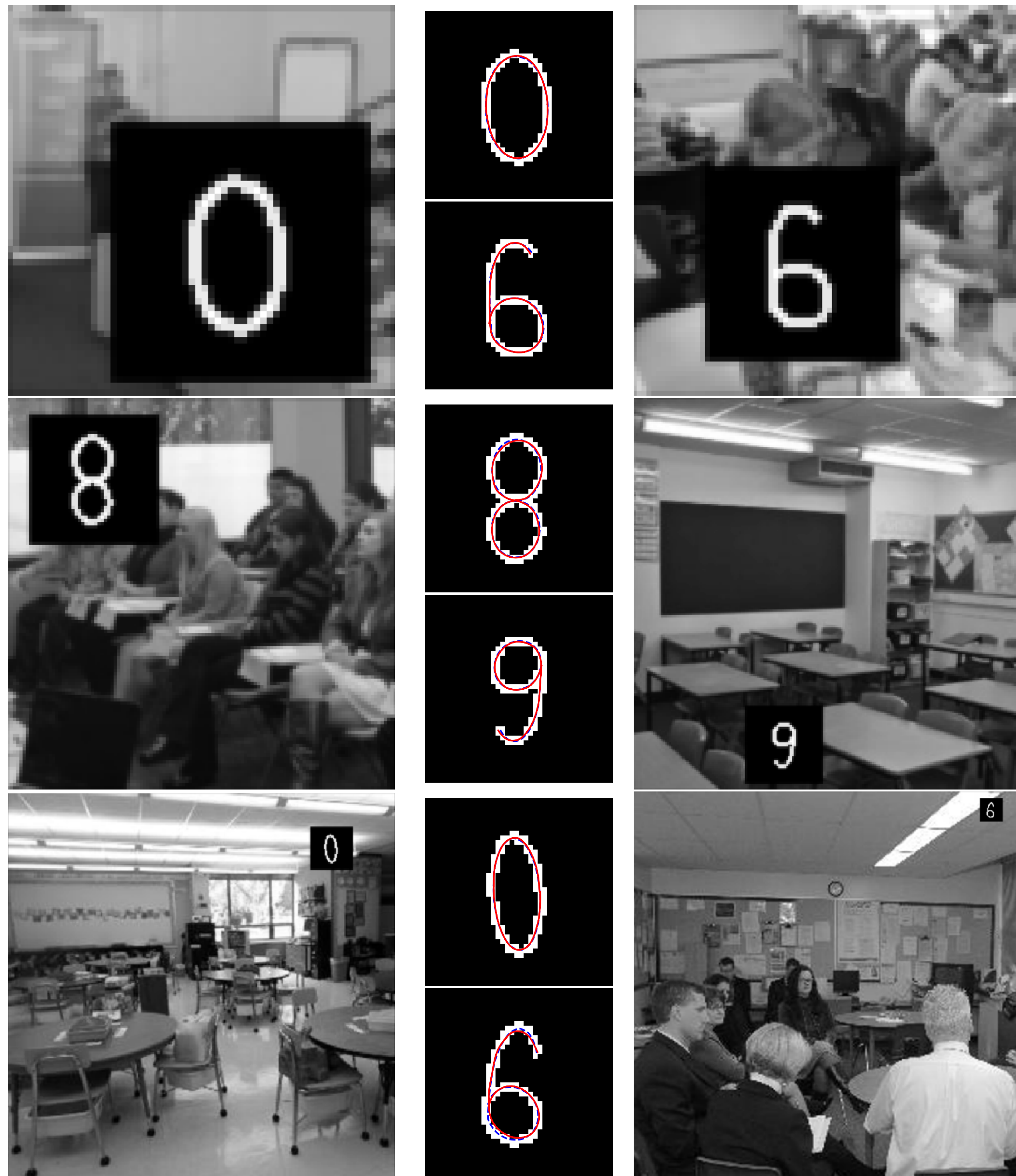


Figure 2.24: Visualization of writing trajectories generated by VIMEDNet trained with normalized AL-DMP path loss function. In this experiment, 40×40 digit images were pasted onto $60 \times 60, 80 \times 80, 120 \times 120, 200 \times 200, 360 \times 360$ and 680×680 pixel background images. The neural network was trained with 80×80 pixel images.

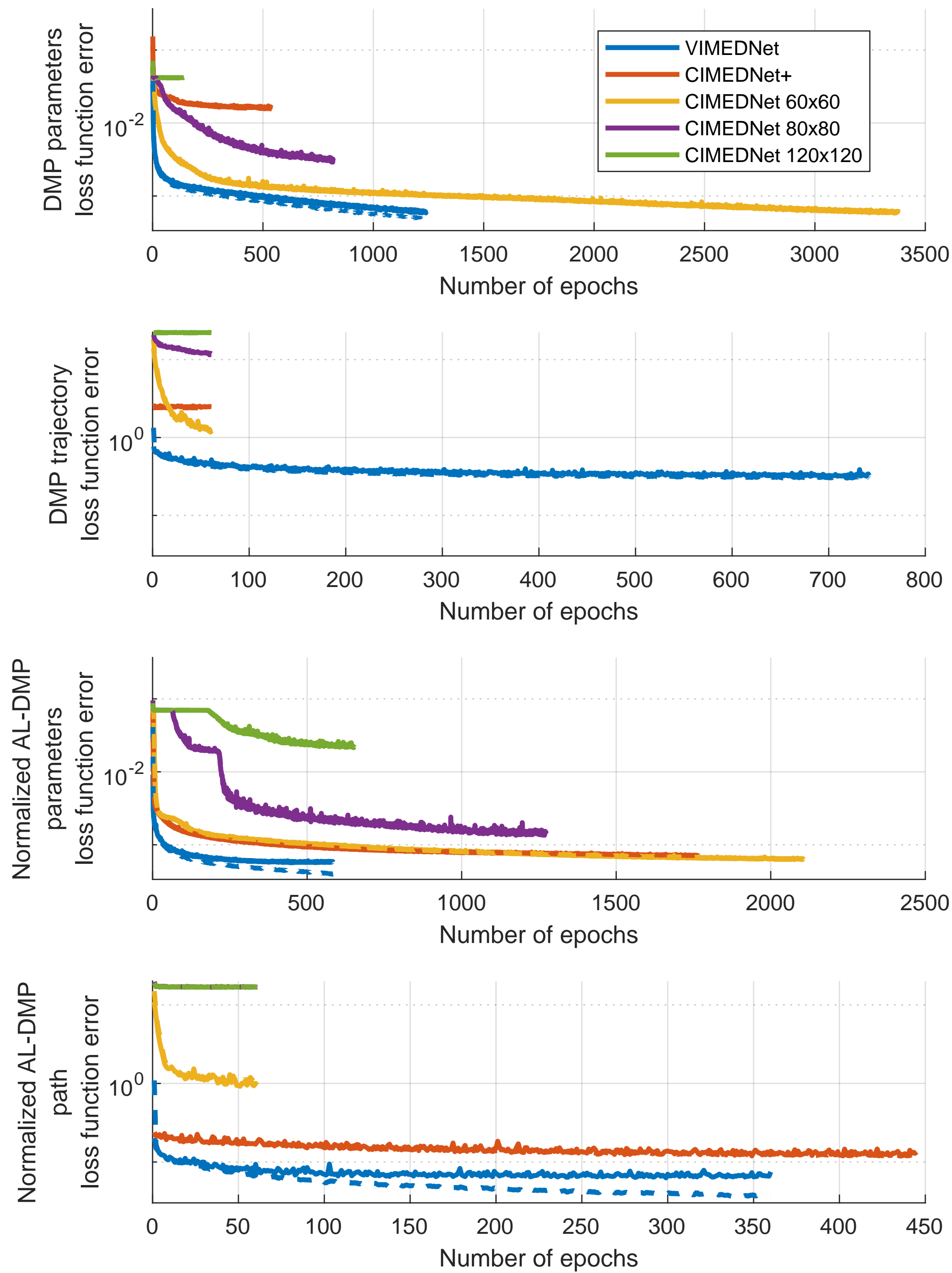


Figure 2.25: Convergence of training errors (solid lines) and validation errors (dashed lines) when processing input images of different sizes. The results for training five different neural network architectures on s-MNIST(0,6,8,9) with a background dataset are shown. In all cases, the training was stopped when the validation error stopped dropping for 60 consecutive epochs.

2.5.8 Experiment with real robot images

The goal of this experiment was to show to the robot real handwritten digits on a piece of paper. The robot should reproduce the digit by writing it on the prepared sheet of paper. For each neural network, we performed two sets of experiments. In the first experiment, we trained CIMEDNet and VIMEDNet with the DMP loss function (2.36), and in the second experiment, we trained both neural networks with the normalized AL-DMP loss function (2.60).

We evaluated the performance of CIMEDNet and VIMEDNet on real input images that were taken by the TALOS humanoid robot [99]. TALOS is a human-size humanoid robot developed by PAL Robotics, with 32 degrees of freedom, equipped with several sensors including RGB-D camera, IMU, force-torque sensors on its wrists and ankles and torque sensors inside its joints. It runs the Robot Operating System (ROS) [100] and comes with the robot dynamic simulation Gazebo [101], which offers initial testing before using the real robot. In these experiments, the images were taken by TALOS's on-board camera. Using real images as input, both neural network architectures generated DMPs or normalized AL-DMPs as output (see Fig. 2.26). The writing trajectories computed by the respective neural network were used to generate the handwriting movements on TALOS.

When we used neural networks with DMP parameters as output, the generated trajectories already contain the temporal information that is necessary to execute the generated trajectories on the robot. In the case of neural nets with normalized AL-DMP parameters, the generated path represents only the spatial part of the trajectory. To execute it on the robot, we added the temporal part of the robot motion as described in Section 2.1.5.

VIMEDNet can reproduce digits from camera images of any size. However, the size of subimages containing the digits must be similar as in the training data. On the other hand, with CIMEDNet, both the size of input images and the size of subimages containing the digits must be similar as in the training data. This is a severe limitation of CIMEDNet because one cannot simply rescale camera images to a smaller size. By doing so, the digit size becomes too small and the digit is no longer recognizable. To overcome this issue, we generated inputs for CIMEDNet by processing real camera images with standard computer vision algorithms in order to detect the area containing the digit in the acquired image. The extracted subimage was then resized to the image size for which CIMEDNet had been trained (40×40 pixels).

For experiments with CIMEDNet (left column in Fig. 2.26), we pasted a piece of paper with a handwritten digit on a whiteboard in front of the robot. We used images of digits pasted on the whiteboard where segmentation is easier than with more complex backgrounds. The acquired camera images were converted to grayscale images. Canny edge detector was then applied. Next we searched through extracted edges to detect closed contours with the shape of a parallelogram and having the appropriate size. The content of the detected parallelogram was then converted to a square shape using affine transformation and resized to 40×40 pixel images required for the CIMEDNet input. The resulting image was then further processed to reduce the influence of lighting. Next, the image was thresholded to obtain a binary image. In the end, we dilated the image with the kernel of size 2 to closely match the width of the pen used by the human writer to the width of the digits in the training dataset.

The processed images were used as input to CIMEDNet, which was trained to return DMP or normalized AL-DMP parameters of the associated writing trajectories. The handwriting trajectories 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 constant height and orientation of the hand (shown in Fig. 2.27). The trajectory was executed with position

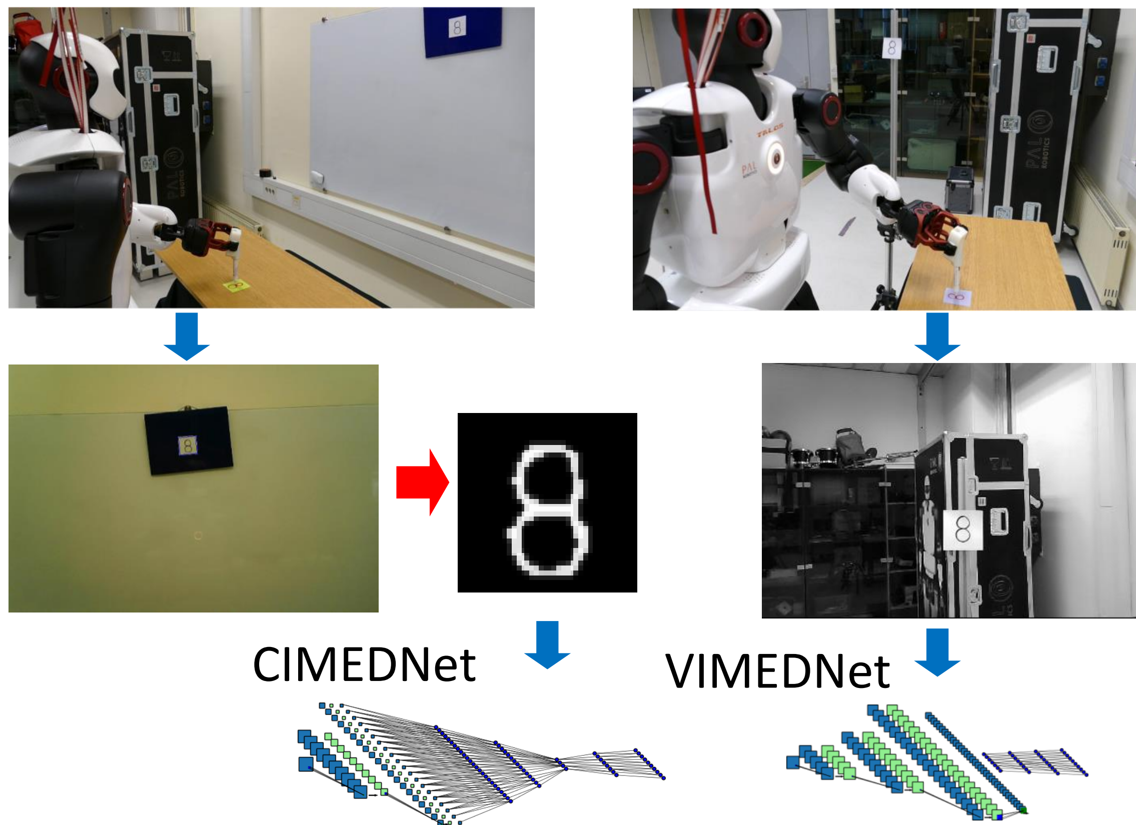


Figure 2.26: Comparison of the experimental setup for testing the CIMEDNet (left) and VIMEDNet (right) neural network architecture. For CIMEDNet, a paper sheet with a handwritten digit is pasted onto a board in front of the robot, the input to the neural network is extracted from the robot camera image using computer vision algorithms (red arrow) and the resized extracted image is used as input. For VIMEDNet, a sheet of paper with a handwritten digit is attached to the rod on the robot's left side, the acquired robot camera image is resized and then used directly as input.

control and the desired force of the pen on the paper was ensured with a spring behind the pen in the holder.

Both CIMEDNet architectures used in the experiment with real robot images were taken directly from the previous experiments: CIMEDNet trained with s-MNIST and loss function (2.36) from the experiment in Section 2.5.5 and CIMEDNet trained with s-MNIST (0,6,8,9) and loss function (2.60) from the experiment in Sec. 2.5.6.

To evaluate the performance of VIMEDNet on real-world data, we attached the handwritten digit to a rod placed on the left side of the robot, as shown in the right column of Fig. 2.26. The VIMEDNet used in this experiment was trained with data described at the end of Sec. 2.5.1.3, i.e. modified s-MNIST with a background dataset. For training with the DMP loss function (2.36), we used modified s-MNIST with all ten digits (*s-MNIST-GRAY with background*), and for the normalized AL DMP loss function (2.60), we used the dataset with only digits 0, 6, 8 and 9 (*s-MNIST(0,6,8,9)-GRAY with background*).

The current version of VIMEDNet cannot deal with scaled images of digits. This means that while the size of the input image is arbitrary, the size of the subimage containing the sheet of paper with a digit must be approximately 40×40 pixels. This is because the

network is trained on 80×80 pixel images with the digit subimage size of 40×40 pixels. We made sure that this was the case in images used in our experiments. Trajectories generated by the neural network were executed in the same way as in the first experiment.

The robot-written digits generated by both neural networks trained on DMPs are shown in Fig. 2.28. Comparing the human-written digits with digits written by a robot using DMPs computed by the CIMEDNet neural network, we observe that the CIMEDNet network is capable of good handwriting reproduction. If we compare the handwritten digits (white papers) with robot-written digits (yellow paper), we can see that CIMEDNet is capable of good trajectory reproduction especially with straight lines as in digits 1, 4, 7 and parts of 2 and 5. Angles and lengths of straight lines are reproduced with good precision. Results are a little worse for curves, especially if we look at digit 0. Still, the starting points and general shapes of curves can be reproduced with good precision for digit 6. The problems with curve reproduction occur due to the synthetic dataset used for training. Human-like straight lines are easily produced in synthetic dataset generation, but this is not the case for human-written curves. In synthetic dataset, we used just elliptic curves for curves generation, but the elliptic curves are often not a good representation for human-written curves.

On grey sheets of paper, we can see the results of writing trajectories output by VIMEDNet. VIMEDNet was also able to reproduce the handwritten digits, but the reproduction quality is significantly worse. VIMEDNet generally recognizes the digit, but the precision of trajectory reproduction is worse by an order of magnitude compared to CIMEDNet. This is because CIMEDNet received segmented digits and thus had better data to work with. As shown in Section 2.5.7, VIMEDNet would outperform CIMEDNet if given the same data.

The robot-written digits generated in the experiment with normalized AL-DMPs are shown in Fig. 2.29. Normalized AL-DMPs are only able to write smooth trajectories, so we used only the examples of human-written digits 0, 6, 8 and 9. If we compare the human-written digits with digits written by a robot using normalized AL-DMPs calculated by CIMEDNet, we find that CIMEDNet is able to reproduce the handwritten digits well. If we compare the robot-written digits generated by CIMEDNet outputting DMPs (in Fig. 2.28) and the robot-written digits generated by CIMEDNet outputting AL-DMPs (in Fig. 2.29), we can observe better results for AL-DMPs. The reasons for this are the better training performance already demonstrated in Section 2.5.6 and also the fact that the training dataset for normalized AL-DMPs contains only 4 different digit forms, which allows better specialization of the neural network.

When comparing the human-written digits with digits written by a robot using normalized AL-DMPs computed by VIMEDNet, we observe that VIMEDNet was also able to reproduce the handwritten digits, but the reproduction quality is not as good as when using the CIMEDNet network. Still, the reproduction of human-written digits 0, 6, 8 and 9 is better than with VIMEDNet trained with DMP framework (in Fig. 2.28).

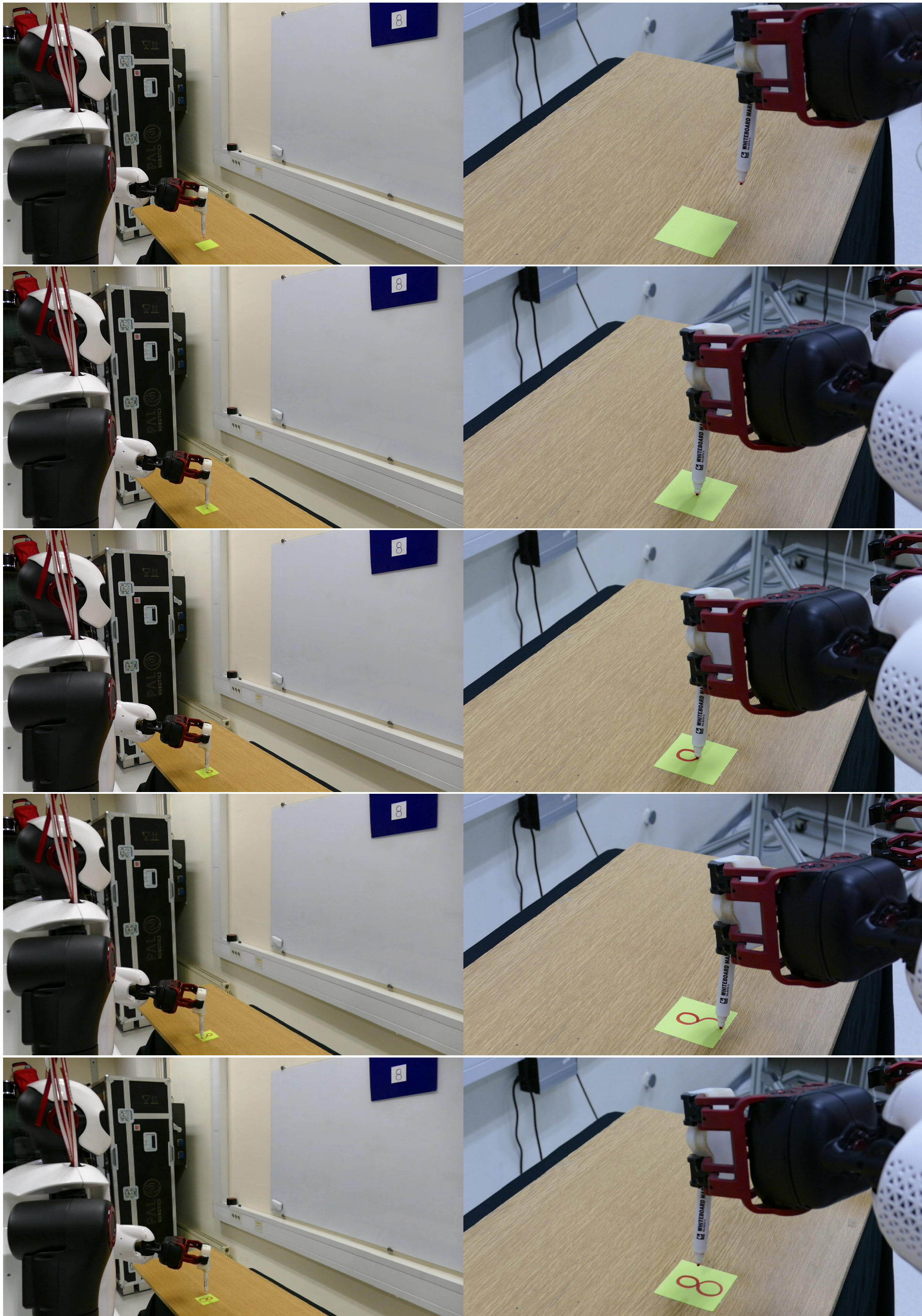


Figure 2.27: Writing digits with the humanoid robot TALOS. The robot holds the writing pen in its hand. The starting position of the pen tip is 4 cm above the lower right corner of the paper sheet. For each writing trajectory, the robot first moves to the starting point, lowers the pen onto the paper and performs the DMP or normalized AL-DMP trajectory generated by a neural network in the horizontal plane, with constant height and orientation of the hand. After execution, the robot lifts the pen and returns to the starting position at the lower right corner of the paper.

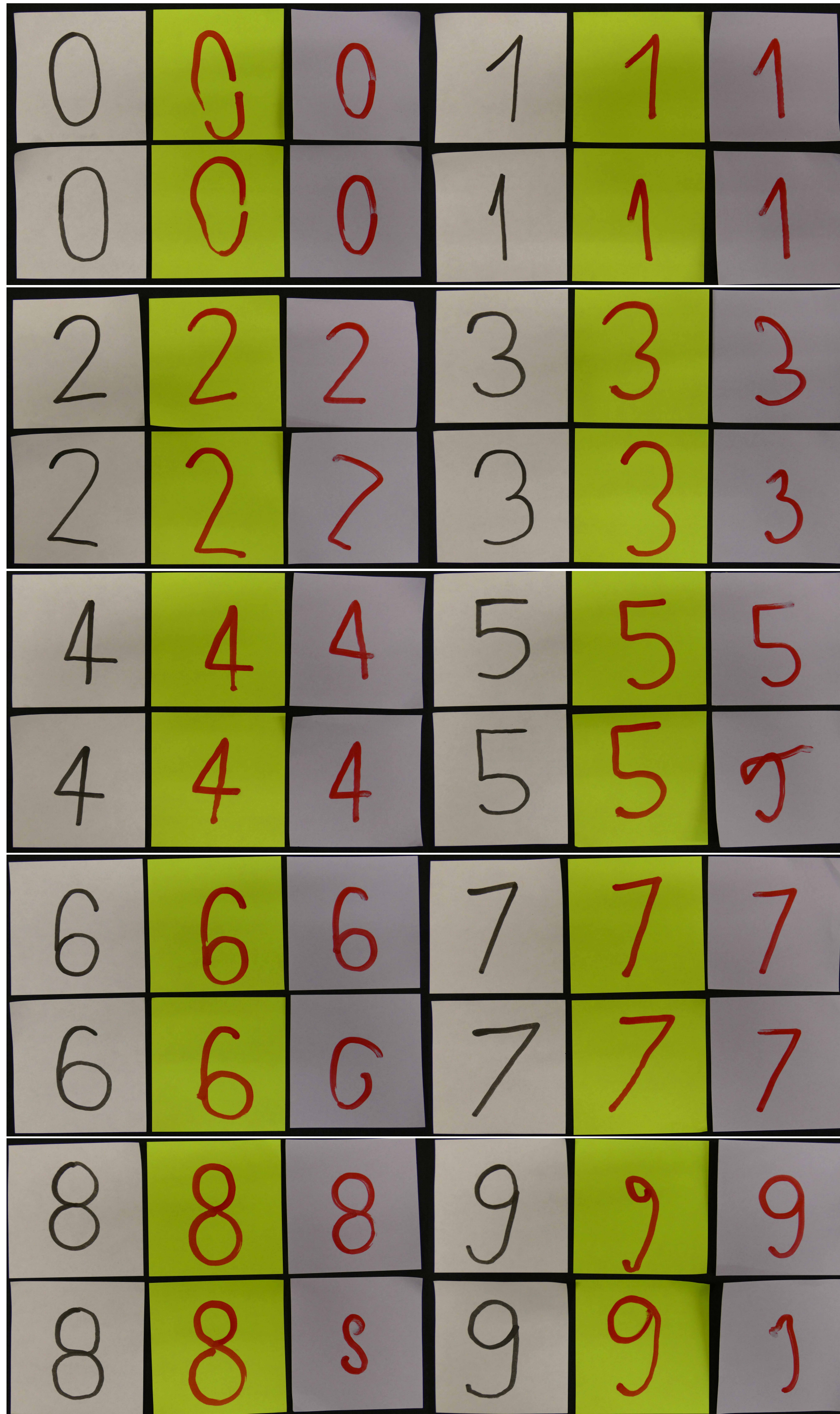


Figure 2.28: Digits written by TALOS using DMPs that were output by CIMEDNet and VIMEDNet. Red digits on yellow paper were generated using CIMEDNet, while red digits on grey paper were generated using VIMEDNet. Digits in black on white paper were handwritten by a human.

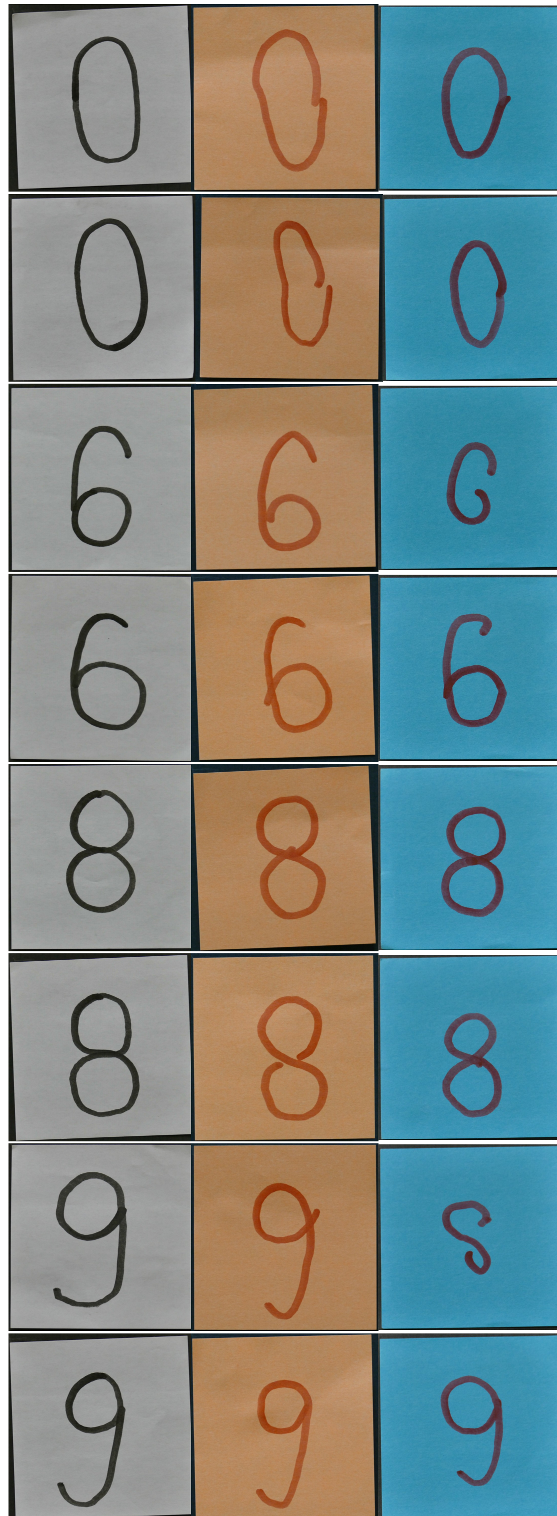


Figure 2.29: Reproduction of handwritten digits with TALOS. The left column shows the input images written by a human in black. The middle column shows digits written by TALOS in red on orange paper using normalized AL-DMPs that were output by CIMEDNet. The right column shows digits written by TALOS in red on blue paper using normalized AL-DMPs that were output by VIMEDNet.

Chapter 3

Robot Skill Learning in Latent Space of a Deep Autoencoder Neural Network

This chapter describes the use of a deep autoencoder network for feature representations learning, where the learned feature representation is used for improving the performance of robot skill learning, as specified in goal **G5**. The second topic of this chapter is the thesis goal **G6**, i.e. evaluating the possibility of using the autoencoder trained on the simulated dataset for learning skills with a real robot. The results presented in this chapter were published in [102].

First, in Section 3.1, we present three different representations of motion trajectories that we have used for learning robot skills. The first representation are the DMPs (Section 3.1.1), which parameters were used as a basis for the creation of the AE latent space (Section 3.1.2) and the PCA latent space (Section 3.1.3). Further, in Section 3.2, we introduced two learning algorithms that we later use to test skill learning with the proposed representations. The first learning algorithm is used for testing reinforcement learning and is a variant of the PoWER method called Reward Weighted Policy Learning with Importance Samples (Section 3.2.1). The second algorithm is Gauss Process Regression (GPR) (Section 3.2.2), which is used to test statistical generalization. In Section 3.3, we present further details on the implementation of reinforcement learning (Section 3.3.1) and statistical generalization (Section 3.3.2) with latent space representations.

We evaluated our approach with an experiment in which a robot throws a ball at a target. The experimental setup is presented in Section 3.4, consisting of the procedure for creating a simulated training dataset for generating latent space representations (Section 3.4.1), details on the parameters for generating AE and PCA latent space (Section 3.4.2), and the procedure for creating a dataset required for learning with statistical generalization (Section 3.4.3).

In Section 3.5, we present the results of the experiments. First we present the experiment in which we compare the data reproduction error of AE and PCA. Reinforcement learning in the DMP parameter space, PCA-based and AE-based latent space is presented in Section 3.5.2. In these experiments, we evaluated the stability and convergence of learning with the dynamic simulation of the robot (Section 3.5.2.1) and with the real robot experiment (Section 3.5.2.2). We evaluated the statistical generalization with different representations in Section 3.5.3. This includes the evaluation of generalization performance (Section 3.5.3.1) and the incremental collection of data for generalization (Section 3.5.3.2).

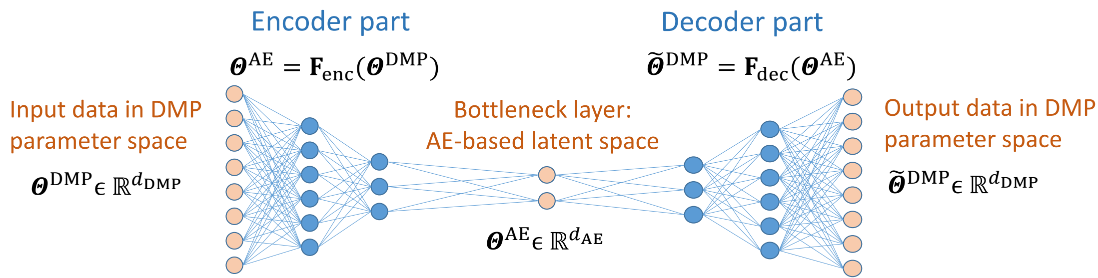


Figure 3.1: Simple example AE network: the encoder part $\mathbf{F}_{\mathrm{enc}} : \mathbb{R}^{d_{\mathrm{DMP}}} \mapsto \mathbb{R}^{d_{\mathrm{AE}}}$ of the network is to the left of the bottleneck layer (with bottleneck neurons as output) and the decoder part $\mathbf{F}_{\mathrm{dec}} : \mathbb{R}^{d_{\mathrm{AE}}} \mapsto \mathbb{R}^{d_{\mathrm{DMP}}}$ to the right of the bottleneck layer (with bottleneck neurons as input).

3.1 DMP Latent Space Representations

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

3.1.1 DMP parameter space

Dynamic Movement Primitives (DMPs) introduced in Section 2.1.1 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 [46]. 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 more robust than learning in the full, usually high-dimensional parameter space.

In a DMP representation, the motion of each robot degree of freedom is represented by a dynamic system. The parameters of this dynamic system include the weight vector $\boldsymbol{\omega} \in \mathbb{R}^N$, the final position on the trajectory $g \in \mathbb{R}$, the starting position $y_0 \in \mathbb{R}$ and the time constant $\tau \in \mathbb{R}$. A short summary of the DMP representation and a more detailed parameter description has already been provided in Section 2.1.1. For robots with more degrees of freedom n_{DOF}, there are separate parameters for the weight vector and for the initial (y_0) and final (g) position for each degree of freedom. On the other hand, there is only one common time constant τ. The number of weights N must be selected by a user so that the desired skill can be encoded with the required accuracy. Thus the full DMP parameter space has $d_{\mathrm{DMP}} = (N + 2)n_{\mathrm{DOF}} + 1$ parameters.

3.1.2 Autoencoder-based latent space

A deep autoencoder (AE) is a type of neural network often applied for dimensionality reduction [69]. 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 recreate 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. 3.1). 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 m 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

$$\boldsymbol{\zeta}^{\star} \quad = \quad \arg\min_{\boldsymbol{\zeta}} \frac{1}{m} \sum_{i=1}^{m} \left\| \boldsymbol{\theta}_i^{\text{DMP}} - \mathbf{F}_{\text{dec}} \left(\mathbf{F}_{\text{enc}} \left(\boldsymbol{\theta}_i^{\text{DMP}} \right) \right) \right\|^2, \tag{3.1}$$

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,

$$\boldsymbol{\theta}^{\text{AE}} = \mathbf{F}_{\text{enc}} \left(\boldsymbol{\theta}^{\text{DMP}} \right) \in \mathbb{R}^{d_{\text{AE}}}. \tag{3.2}$$

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

$$\tilde{\boldsymbol{\theta}}^{\text{DMP}} = \mathbf{F}_{\text{dec}} \left(\boldsymbol{\theta}^{\text{AE}} \right) \in \mathbb{R}^{d_{\text{DMP}}}. \tag{3.3}$$

3.1.3 PCA-based latent space

Principal Component Analysis (PCA) [103] 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. An important 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 eingenvalues of the data covariance matrix

$$\boldsymbol{S} = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}_i^{\text{DMP}} - \bar{\boldsymbol{\theta}}^{\text{DMP}} \right) \left(\boldsymbol{\theta}_i^{\text{DMP}} - \bar{\boldsymbol{\theta}}^{\text{DMP}} \right)^{\text{T}}, \tag{3.4}$$

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

$$\boldsymbol{\theta}^{\text{PCA}} = \boldsymbol{W}_{\text{PCA}}^{\text{T}} \left(\boldsymbol{\theta}^{\text{DMP}} - \bar{\boldsymbol{\theta}}^{\text{DMP}} \right). \tag{3.5}$$

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

$$\tilde{\boldsymbol{\theta}}^{\text{DMP}} = \boldsymbol{W}_{\text{PCA}} \boldsymbol{\theta}^{\text{PCA}} + \bar{\boldsymbol{\theta}}^{\text{DMP}}. \tag{3.6}$$

3.2 Learning Algorithms

We compared the proposed representations by implementing different learning methods in the AE-based latent space, in the PCA-based latent space, and in full DMP parameter space. For the first comparison, we applied Reward-Weighted Policy Learning with Importance Sampling reinforcement learning method. 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.

For the second comparison, we applied the Gaussian Process Regression (GPR) method. GPR has been selected because it had been demonstrated [104] that it outperforms other statistical learning methods in difficult learning problems such as estimating the inverse dynamics of a seven-degrees-of-freedom robot arm. The two methods are summarized below.

3.2.1 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 [44]. 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

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \frac{\langle (\boldsymbol{\Theta}_n - \boldsymbol{\theta}_n) R(\boldsymbol{\Theta}_n) \rangle_{w(\boldsymbol{\tau})}}{\langle R(\boldsymbol{\Theta}_n) \rangle_{w(\boldsymbol{\tau})}}, \tag{3.7}$$

where $\boldsymbol{\Theta}_n = \{\boldsymbol{\theta}_k^*\}_{k=1}^n$ denotes the set of all policy parameters $\boldsymbol{\theta}_k^*$ executed until the n-th 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, the role of which 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 (3.7) can be applied to estimate the weights of the DMP forcing term. It is equivalent to

$$\boldsymbol{\theta}_{n+1} = \frac{\sum_{i=1}^m R_{\mathrm{in}(n,i)} \boldsymbol{\theta}^*_{\mathrm{in}(n,i)}}{\sum_{i=1}^m R_{\mathrm{in}(n,i)}}, \tag{3.8}$$

where function $\mathrm{in}(n, i)$ selects the trial with the i-th 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 $\boldsymbol{\theta}_n$

$$\boldsymbol{\theta}_n^* = \boldsymbol{\theta}_n + \boldsymbol{\varepsilon}_n. \tag{3.9}$$

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 (3.7) and equivalently (3.8) 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.

3.2.2 Gaussian Process Regression

Gaussian Process Regression (GPR) is a nonparametric, Bayesian approach to regression. A Gaussian process is defined as

$$g(\mathbf{q}) \sim \mathcal{GP}\left(m(\mathbf{q}), k(\mathbf{q}, \mathbf{q}')\right), \tag{3.10}$$

where $m(\mathbf{q}) = \mathbb{E}(g(\mathbf{q}))$ is the mean function and $k(\mathbf{q}, \mathbf{q}') = \mathbb{E}((g(\mathbf{q}) - m(\mathbf{q}))(g(\mathbf{q}') - m(\mathbf{q}')))$ the covariance function of the process. Let us assume that we have a set of noisy observations $\{\mathbf{q}_k, \theta_k\}_{k=1}^m$, $\theta_k = g(\mathbf{q}_k) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$, where $\mathcal{N}$ denotes the Gaussian distribution. In our experiments, θ_k is one of the parameters describing the motion (one of the DMP or latent space parameters), $\mathbf{q}$ is the desired target of the throw, and g is the unknown nonlinear function mapping query points to the parameters of the throwing motion. Subtracting the mean from the training data, we can assume that $m(\mathbf{q}) = 0$. If we are given a set of m_2 new query points $\mathbf{Q}^* = \{\mathbf{q}_k^*\}_{k=1}^{m_2}$, then the joint distribution of all outputs is given as [104]

$$\begin{bmatrix} \boldsymbol{\vartheta} \\ \boldsymbol{\vartheta}^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{Q}, \mathbf{Q}) + \sigma_n^2 \mathbf{I} & \mathbf{K}(\mathbf{Q}, \mathbf{Q}^*) \\ \mathbf{K}(\mathbf{Q}^*, \mathbf{Q}) & \mathbf{K}(\mathbf{Q}^*, \mathbf{Q}^*) \end{bmatrix}\right), \tag{3.11}$$

where $\mathbf{Q} = \{\mathbf{q}_k\}_{k=1}^m$, $\mathbf{Q}^*$, $\boldsymbol{\vartheta} = [\theta_1, \ldots, \theta_m]^{\mathrm{T}}$, $\boldsymbol{\vartheta}^*$ respectively combine all inputs and outputs and $\mathbf{K}(\cdot, \cdot)$ are the joint covariance matrices calculated according to the model (3.10). Based on the joint distribution (3.11), the expected value $\bar{\boldsymbol{\vartheta}}^* \in \mathbb{R}^{m_2}$ can be calculated as [104]

$$\bar{\boldsymbol{\vartheta}}^* = \mathbb{E}(\boldsymbol{\vartheta}^* | \mathbf{Q}, \boldsymbol{\vartheta}, \mathbf{Q}^*) = \mathbf{K}(\mathbf{Q}^*, \mathbf{Q})[\mathbf{K}(\mathbf{Q}, \mathbf{Q}) + \sigma_n^2 \mathbf{I}]^{-1} \boldsymbol{\vartheta}, \tag{3.12}$$

with the following estimate for the covariance of the prediction

$$\operatorname{cov}(\boldsymbol{\vartheta}^*) = \mathbf{K}(\mathbf{Q}^*, \mathbf{Q}^*) - \mathbf{K}(\mathbf{Q}^*, \mathbf{Q})[\mathbf{K}(\mathbf{Q}, \mathbf{Q}) + \sigma_n^2 \mathbf{I}]^{-1} \mathbf{K}(\mathbf{Q}, \mathbf{Q}^*).$$

One commonly used covariance function is

$$k(\mathbf{q}, \mathbf{q}') = \sigma_f^2 \sum_{i=1}^{D_q} \exp\left(-\frac{1}{2} \frac{(q_i - q_i')^2}{l_i^2}\right), \tag{3.13}$$

which results in a Bayesian regression model with an infinite number of basis functions. D_q denotes the dimension of the query point space. σ_n^2, σ_f^2 and l_i are the hyperparameters of the Gaussian process that need to be estimated in the training phase. See [104] for more details.

3.3 Learning in AE- and PCA-Based Latent Spaces

The main aim of this chapter 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 Reward-Weighted Policy Learning with Importance Sampling and Gaussian Process Regression 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.

3.3.1 Reinforcement learning in latent spaces

RL in both the DMP parameter space and latent spaces estimates the movement parameters using Eq. (3.8) – (3.9). The only difference when RL is implemented in the latent space is that the estimated parameters $\boldsymbol{\theta}_n^{\mathrm{AE}}$ and $\boldsymbol{\theta}_n^{\mathrm{PCA}}$ 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

$$\tilde{\boldsymbol{\theta}}_n^{\mathrm{DMP}} = \mathbf{F}_{\mathrm{dec}}\left(\boldsymbol{\theta}_n^{\mathrm{AE}}\right). \tag{3.14}$$

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

$$\tilde{\boldsymbol{\theta}}_n^{\mathrm{DMP}} = \boldsymbol{W}_{\mathrm{PCA}}\boldsymbol{\theta}_n^{\mathrm{PCA}} + \bar{\boldsymbol{\theta}}^{\mathrm{DMP}}. \tag{3.15}$$

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.

3.3.2 Gaussian Processes Regression in latent spaces

For GPR 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 $\boldsymbol{q}_{\mathrm{d}}$ predicts the corresponding control policy $\boldsymbol{\theta}_{\mathrm{d}}^{\mathrm{DMP}}$ when learning takes place in full DMP parameter space or $\boldsymbol{\theta}_{\mathrm{d}}^{\mathrm{AE}}$ and $\boldsymbol{\theta}_{\mathrm{d}}^{\mathrm{PCA}}$ when learning takes place in AE- and PCA-based latent space, respectively. Thus, the following transformation functions are computed by GPR:

$$\mathbf{G}_{\mathrm{DMP}}(\{\boldsymbol{\theta}_i^{\mathrm{DMP}}, \boldsymbol{q}_i\}_{i=1}^m) : \boldsymbol{q}_{\mathrm{d}} \mapsto \boldsymbol{\theta}_{\mathrm{d}}^{\mathrm{DMP}}, \tag{3.16}$$

$$\mathbf{G}_{\mathrm{AE}}(\{\boldsymbol{\theta}_i^{\mathrm{AE}}, \boldsymbol{q}_i\}_{i=1}^m) : \boldsymbol{q}_{\mathrm{d}} \mapsto \boldsymbol{\theta}_{\mathrm{d}}^{\mathrm{AE}}, \tag{3.17}$$

$$\mathbf{G}_{\mathrm{PCA}}(\{\boldsymbol{\theta}_i^{\mathrm{PCA}}, \boldsymbol{q}_i\}_{i=1}^m) : \boldsymbol{q}_{\mathrm{d}} \mapsto \boldsymbol{\theta}_{\mathrm{d}}^{\mathrm{PCA}}. \tag{3.18}$$

After computing the AE- and PCA-based latent space parameters using Eq. (3.17) and (3.18), 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. Eq. (3.14) and (3.15).

3.4 Experimental Setup

The experiments 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 sagittal plane, were used to realize ball throwing. The simulation setup is depicted in Fig. 3.2, while the real-world setup is shown in Fig. 3.3. We used MuJoCo [105] 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 was placed onto the holder, but was 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 becomes

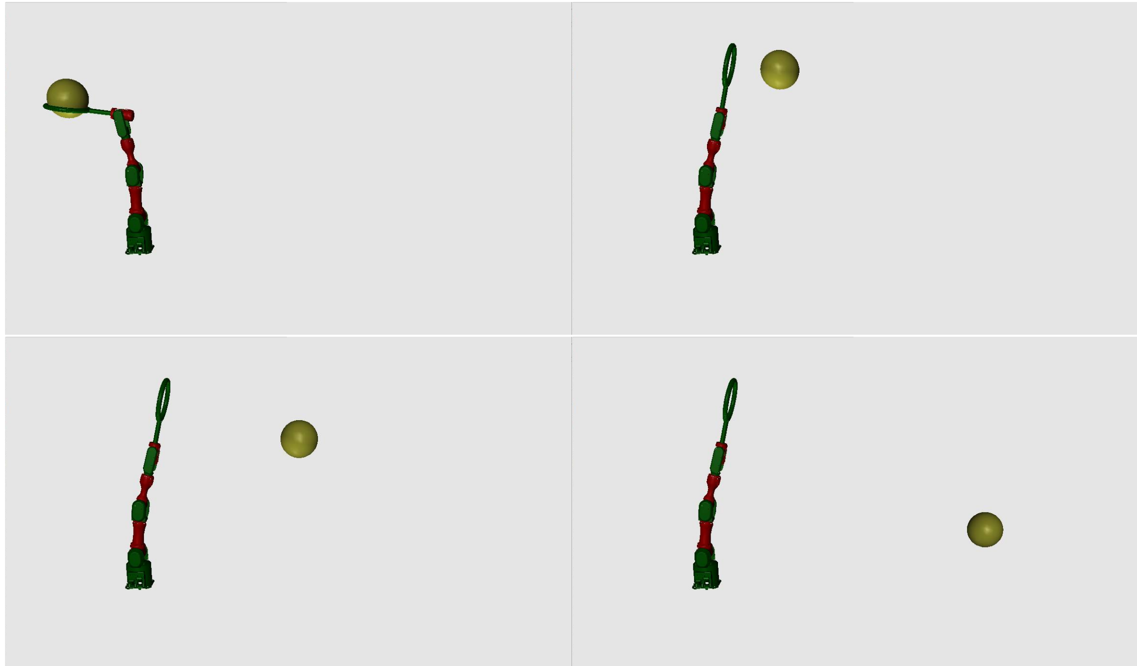


Figure 3.2: MuJoCo dynamic simulation experimental setup for evaluation of reinforcement learning and statistical learning in different spaces.

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 [42] and reinforcement learning [106]. 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.

3.4.1 Generation of simulated ball throwing trajectories for training

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

$$\mathbf{E} = \left\{\boldsymbol{\theta}_j^{\mathrm{DMP}}\right\}_{j=1}^{P}, \tag{3.19}$$

which consists of P robot throwing trajectories represented by the DMP parameters $\boldsymbol{\theta}_j^{\mathrm{DMP}}$. These parameters describe the joint motion of the three degrees freedom relevant for throwing. The example throwing trajectories are used for training of deep autoencoder neural networks or to compute PCA parameters. The trajectories were generated in kinematical simulation, where we assumed that the ball is released at maximal joint velocities.

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

$$\boldsymbol{p}(t) = \begin{bmatrix} p_x^r + (t - t_r) v_0 \cos(\alpha_r) \\ p_y^r + (t - t_r) v_0 \sin(\alpha_r) - \frac{1}{2}(t - t_r)^2 g \end{bmatrix}, \tag{3.20}$$

where t_r denotes the time at which the robot releases the ball, $\boldsymbol{p}_r = [p_x^r, p_y^r]^{\mathrm{T}}$ is 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. (3.20) can

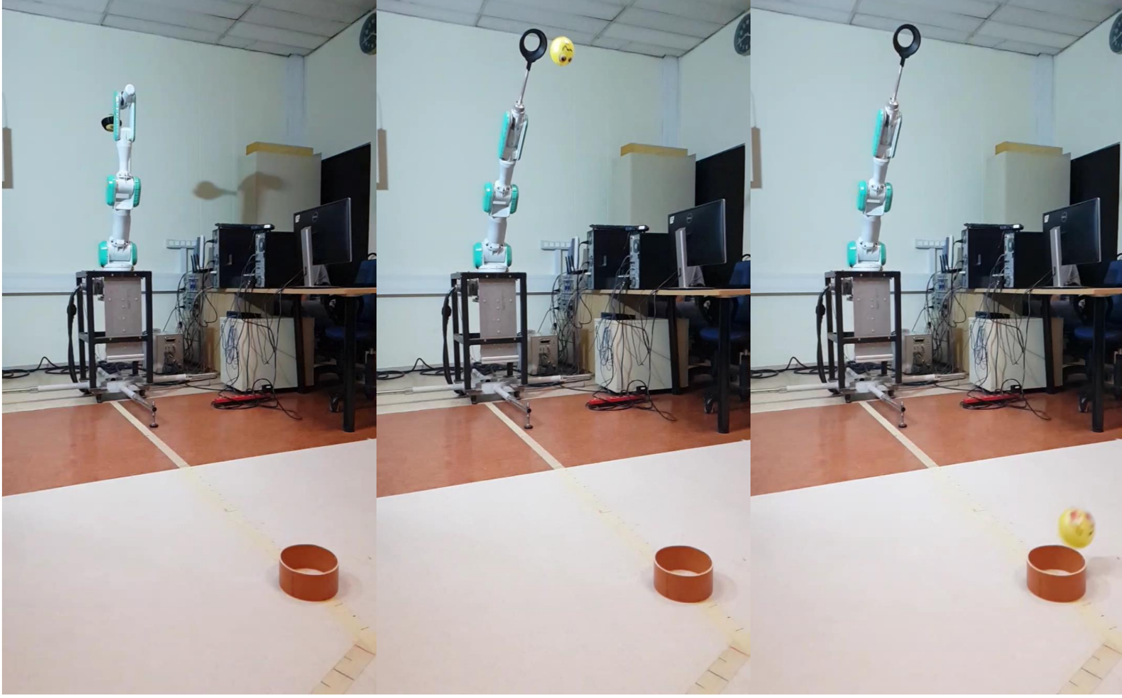


Figure 3.3: Mitsubishi PA-10 robot experimental setup for evaluation of reinforcement learning and statistical learning in different spaces. The PA-10 robot is shown in its initial posture (left), after the release of the ball (center), and when the ball lands (right).

be re-written as a parabola in 2-D plane

$$p_y(t) - p_y^r = \tan(\alpha_r)(p_x(t) - p_x^r) - \frac{1}{2}\frac{g}{v_0^2 \cos^2(\alpha_r)}(p_x(t) - p_x^r)^2. \tag{3.21}$$

All values are given in the robot base coordinate system.

Formula (3.21) was used in [42] to compute the angle

$$\alpha_r(\boldsymbol{p}_r, \alpha) = \arctan\left(2\frac{h - p_y^r}{d - p_x^r} - \tan(\alpha)\right) \tag{3.22}$$

and absolute velocity

$$v_0(\boldsymbol{p}_r, \alpha) = \sqrt{-\frac{g(d - p_x^r)^2}{2(\tan(\alpha)(d - p_x^r) - (h - p_y^r))\cos^2(\alpha_r(\boldsymbol{p}_r, \alpha))}} \tag{3.23}$$

of the robotic throwing movement at release time from the given release position $\boldsymbol{p}_r$, the final target position $\boldsymbol{q}_\mathrm{T} = [d, h]^\mathrm{T}$, 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 [42] 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 $\mathbf{F}_\mathrm{fk}$ 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 release position, $\boldsymbol{p}_r = \mathbf{F}_\mathrm{fk}(\boldsymbol{y}_r)$. We formulate the following optimization problem

$$\begin{aligned} &\arg\min_{\boldsymbol{y}_r,\alpha} \left\{ \|\boldsymbol{W}_{\text{opt}}\dot{\boldsymbol{y}}_r\|^2 = \left\| \boldsymbol{W}_{\text{opt}}\boldsymbol{J}_r^+ v_0\left(\mathbf{F}_{\text{fk}}(\boldsymbol{y}_r),\alpha\right) \begin{bmatrix} \cos(\alpha_r(\mathbf{F}_{\text{fk}}(\boldsymbol{y}_r),\alpha)) \\ \sin(\alpha_r(\mathbf{F}_{\text{fk}}(\boldsymbol{y}_r),\alpha)) \end{bmatrix} \right\|^2 \right\}, \\ &\text{subject to} \\ &-90^\circ \leq \alpha \leq -35^\circ,\ \boldsymbol{y}^{\min} \leq \boldsymbol{y}_r \leq \boldsymbol{y}^{\max}, \end{aligned} \tag{3.24}$$

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. (3.22) and (3.23), 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 (3.22) – (3.23). The resulting ball trajectories are shown in Fig. 3.4 as green dotted lines.

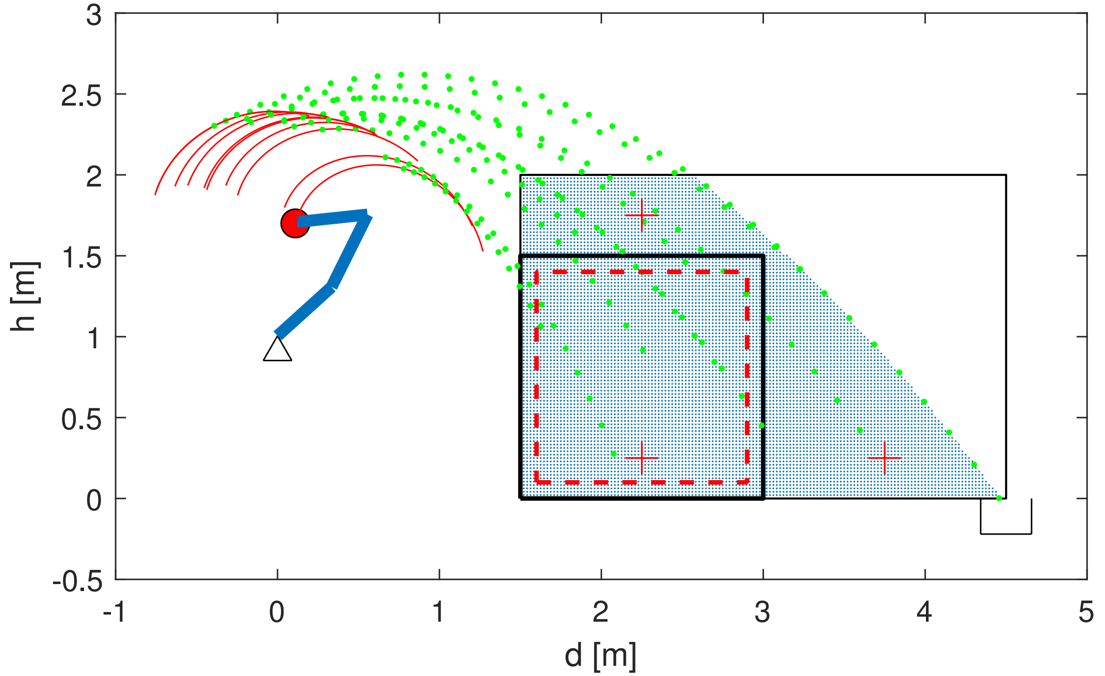


Figure 3.4: Simulated ball throwing trajectories dataset. Using kinematic simulation, we created a dataset of trajectories for targets in a rectangle of $3m \times 2m$ (the black rectangle). The targets for which executable robot trajectories could be generated are marked with small blue points. A subset of trajectories associated with targets in the black square was used to train GPR, while the 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. Some examples of the computed end-effector trajectories for ball throwing are presented in red with the corresponding ball flight trajectories in green.

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

$$\begin{aligned} \boldsymbol{y}(t) &= \left(\boldsymbol{y}_r - \frac{t_r(\frac{2}{5}(t_r/t_{end})^2 - \frac{14}{15}t_r/t_{end} + \frac{1}{2})}{(t_r/t_{end} - 1)^2}\dot{\boldsymbol{y}}_r\right) + \frac{1 - \frac{4}{3}t_r/t_{end}}{t_r^2(t_r/t_{end} - 1)^2}\dot{\boldsymbol{y}}_r t^3 + \\ & \frac{2(t_r/t_{end})^2 - 1}{2t_r^3(t_r/t_{end} - 1)^2}\dot{\boldsymbol{y}}_r t^4 + \frac{2 - 3t_r/t_{end}}{5t_{end}t_r^3(t_r/t_{end} - 1)^2}\dot{\boldsymbol{y}}_r t^5. \end{aligned} \tag{3.25}$$

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

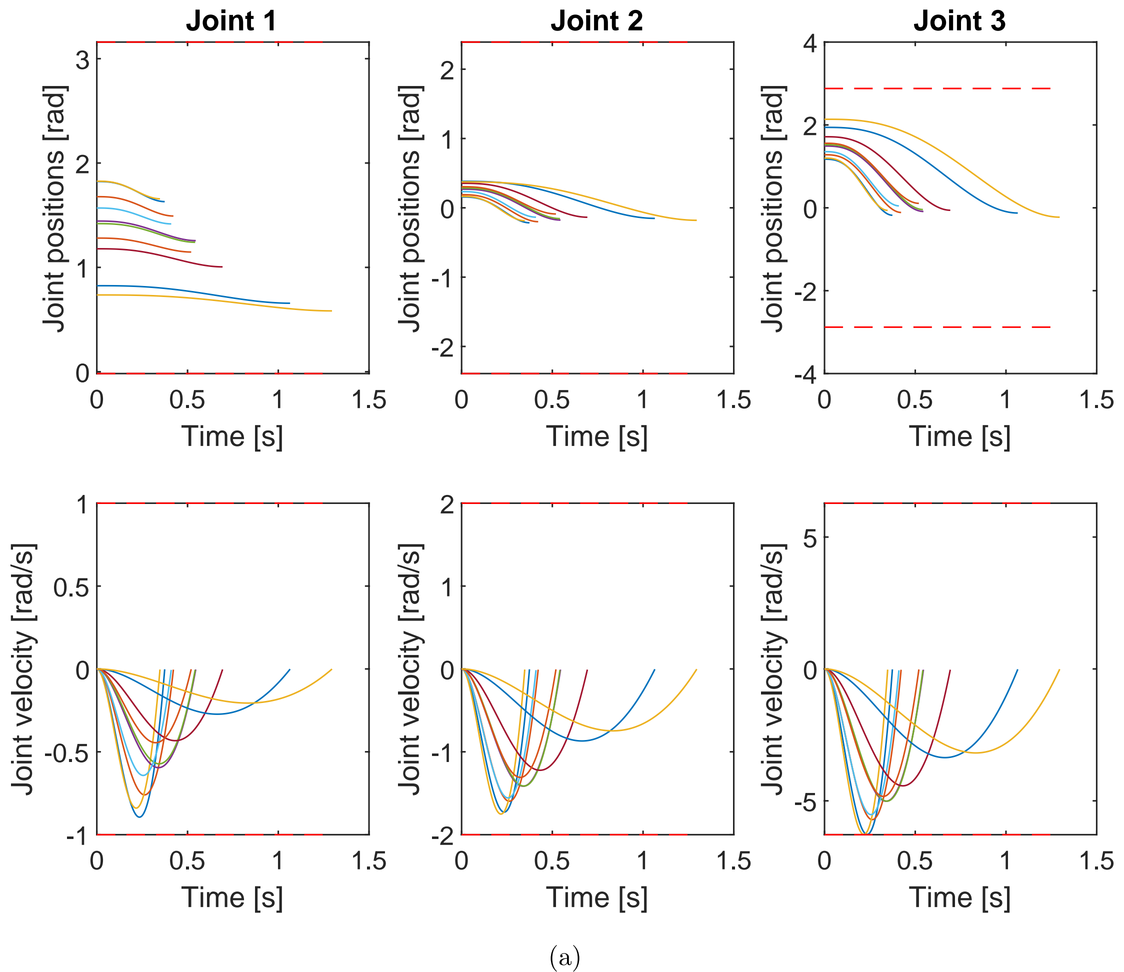
$$\begin{aligned} & \arg\min_{t_r, t_{end}} \left\{ \left(3\pi/4 - \textstyle\sum_{i=1}^{3} y_i(0)\right)^2 \right\} \\ & \text{subject to} \\ & 0.3 \le t_{end},\ 0 < t_r < t_{end},\ \boldsymbol{y}^{\min} \le \boldsymbol{y}(0) \le \boldsymbol{y}^{\max}, \end{aligned} \tag{3.26}$$

where

$$\boldsymbol{y}(0) = \boldsymbol{y}_r - \dot{\boldsymbol{y}}_r \frac{t_r(\frac{2}{5}(t_r/t_{end})^2 - \frac{14}{15}t_r/t_{end} + \frac{1}{2})}{(t_r/t_{end} - 1)^2}. \tag{3.27}$$

The ball throwing trajectory is fully determined by solving (3.26). The resulting ball throwing trajectories are shown in Fig. 3.4 as red lines and corresponding robot joint trajectories in Fig. 3.5.

A database for a target grid with the distances in the range from 1.5 to 4.5 meters 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 violating the real robot joint and joint velocity limits. This way we gathered 9824 throwing trajectory examples (see also Fig. 3.4). The simulated trajectories were encoded into the DMP parameters with $N = 20$ DMP for each DOF. We obtain 60 weights and together with 3 starting points, 3 goals and 1 common time constant, this adds up to 67-dimensional DMP parameters $\boldsymbol{\theta}^{DMP}$. The data are available for download at [107].



(a)

Figure 3.5: Example joint trajectories and joint velocities corresponding to these trajectories. Dashed red lines mark the boundary values for joint positions and velocities.

3.4.2 Generating AE and PCA-based latent spaces

We used the autoencoder or PCA to reduce the dimensionality of the 67-dimensional DMP parameter space $\boldsymbol{\theta}^{\mathrm{DMP}}$ and create a corresponding low-dimensional latent space.

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 DMP parameters passed through the autoencoder started to drop significantly. The resulting AE network comprised 5 hidden layers with 15, 10, 3, 10, and 15 neurons, as shown in Fig. 3.6. For the activation function of each hidden layer, we used $\mathbf{h}_k^{\mathrm{AE}} = \tanh(\mathbf{W}_{\mathrm{AE}}\mathbf{h}_{k-1}^{\mathrm{AE}} + \mathbf{b}_{\mathrm{AE}})$, where $\boldsymbol{\zeta}^\star = \{\mathbf{W}_{\mathrm{AE}},\ \mathbf{b}_{\mathrm{AE}}\}$ are the AE parameters, $\mathbf{h}_k^{\mathrm{AE}}$ denotes the output of the k-th layer, and $\mathbf{h}_{k-1}^{\mathrm{AE}}$ is the output of the previous layer that serves as input to the neurons of the k-th layer. The activation function of the output layer was linear. The input and output layer size was defined with the chosen number of DMP parameters, which was 67 as explained in Section 3.4.1. For autoencoder training, we used 70% of 9824 trajectories contained in dataset (3.19). The remaining 15% were used for validation and 15% for testing.

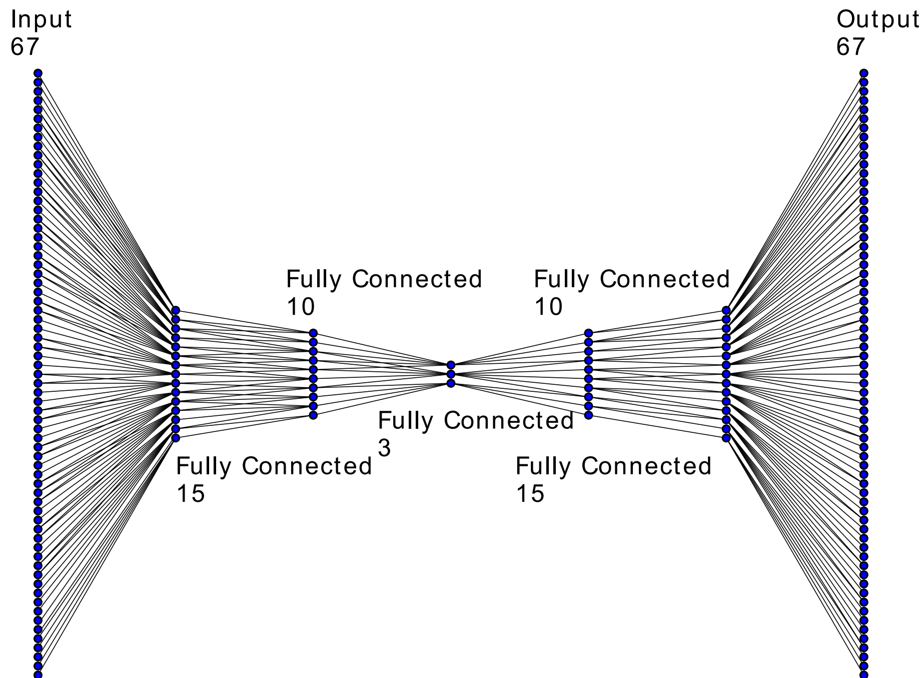


Figure 3.6: Illustration of the designed autoencoder structure with five hidden layers. The number of neurons per layer is 67 for input and output layers, and 15, 10, 3, 10, 15 for hidden layers.

The three-dimensional AE-based latent space can be used to visualize the data. Figure 3.7 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 (3.19), we also conducted the PCA analysis and obtained the following largest eigenvalues: $\boldsymbol{\lambda} = [1445980, 5769, 3623, 1058, 19, 1, ...]$. 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.

3.4.3 Dataset for statistical learning

Just like the computation of latent spaces, statistical learning needs training data too. As discussed in Section 3.3.2, 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). The data in the black square in Fig. 3.4 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 (Sec. 3.1.1) and then transformed into a 3 dimensional AE- and PCA-based latent space representation (see Section 3.1). These trajectory representations were then used to generate executable robot trajectories in dynamic simulation, where the new landing distances and heights were computed. Note

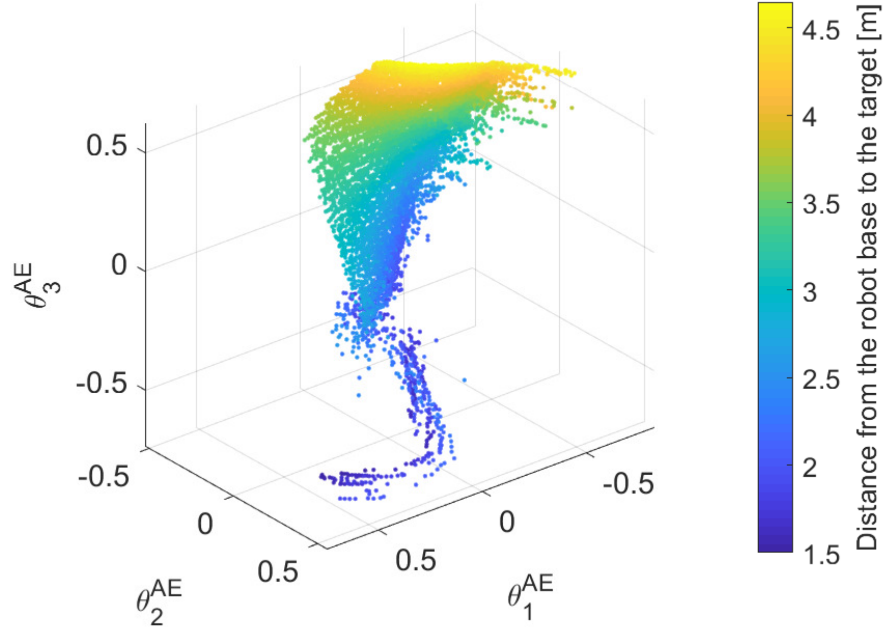


Figure 3.7: The plot shows some example points in the resulting AE-based latent space, computed by projecting the training data in Fig. 3.4 to the latent space. The color of points represents the distance from the base of the robot to the target of throwing.

that these were slightly different for each representation as the mapping to latent spaces and DMP integration result in slightly different movements (see Fig. 3.8).

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

$$\mathbf{q} = [d, h]^{\mathrm{T}}, \tag{3.28}$$

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

$$\mathcal{G}^{\mathrm{DMP}} = \{\boldsymbol{\theta}_j^{\mathrm{DMP}}, \mathbf{q}_j^{\mathrm{DMP}}\}_{j=1}^{R}, \tag{3.29}$$

$$\mathcal{G}^{\mathrm{AE}} = \{\boldsymbol{\theta}_j^{\mathrm{AE}}, \mathbf{q}_j^{\mathrm{AE}}\}_{j=1}^{R}, \tag{3.30}$$

$$\mathcal{G}^{\mathrm{PCA}} = \{\boldsymbol{\theta}_j^{\mathrm{PCA}}, \mathbf{q}_j^{\mathrm{PCA}}\}_{j=1}^{R}. \tag{3.31}$$

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.

3.5 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.

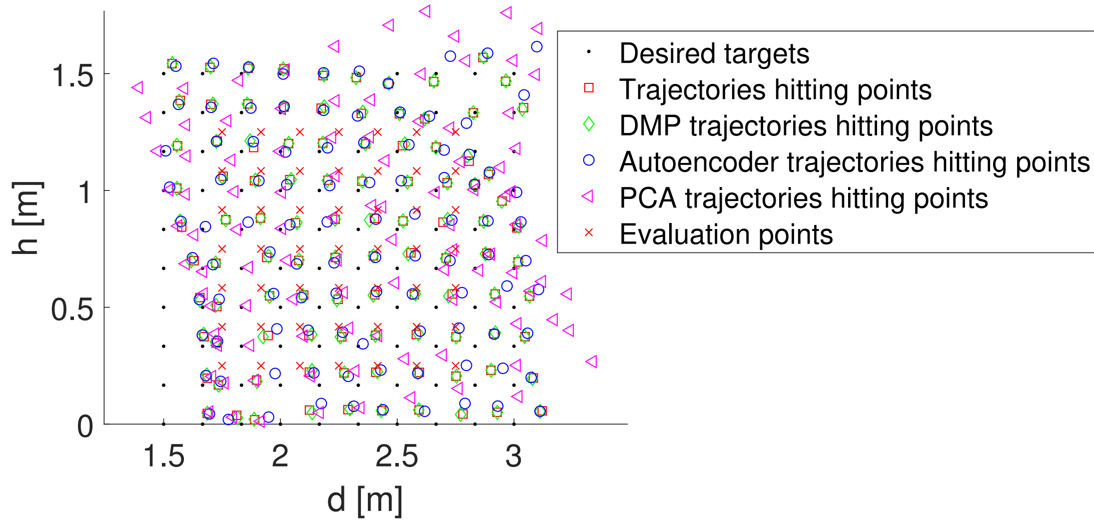


Figure 3.8: Query points for statistical generalization and targets for testing.

3.5.1 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 of how well each method has learned 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. (3.2) or (3.5) to respectively project the original DMPs onto the AE- and PCA-based latent spaces, followed by an application of Eq. (3.3) or (3.6) to map the latent space representations back to the DMP representation. The reproduction error for the j-th joint trajectory can thus be computed as follows

$$E_{\mathrm{J}}(j) = \frac{1}{T_j} \sum_{i=1}^{T_j} \|\mathbf{y}_j^{\mathrm{AE/PCA}}(x_{i,j}) - \mathbf{y}_j^{\mathrm{DMP}}(x_{i,j})\|, \tag{3.32}$$

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 j-th 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 j-th DMP without latent space projection. T_j denotes the number of points for the j-th DMP. A subset of the data described in Sec. 3.4.2 was used for testing (1474 examples), while the rest of the data was used to train AEs and compute PCA.

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 Tab. 3.1 show that the average reproduction error is smaller with AEs than with PCA. This is consistent with other results presented later in Sec. 3.5.2 and 3.5.3. In general, a better reproductive performance can be expected when using AE-

Table 3.1: 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 (3.32) 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

based latent spaces compared to PCA-based latent spaces. This is due to the ability of the AEs to perform nonlinear approximations [49]. Our results in Tab. 3.1 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.

3.5.2 Reinforcement learning experiments

The main focus of our RL experiments was to evaluate the stability and speed of convergence for robot learning to precisely hit the target.

3.5.2.1 Dynamic simulation

Reinforcement learning, more specifically reward-weighted policy learning with importance sampling, for each trajectory representation, was first tested in dynamic simulation with throwing at three different targets (see Fig. 3.4). For each target we carried out the reinforcement learning process (Eq. 3.8) 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 $\boldsymbol{q}_{\mathrm{T}}$ and the point on the throwing trajectory $\boldsymbol{q}_i$. The measured distance was used to compute the terminal reward for reinforcement learning

$$R(\tau_i) = \exp(-\|\boldsymbol{q}_{\mathrm{T}} - \boldsymbol{q}_i\|^2). \tag{3.33}$$

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. 3.9, 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. The results are shown in the bar graph in Fig. 3.10.

It is clear from these graphs that RL in latent spaces converges significantly faster than RL in the 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

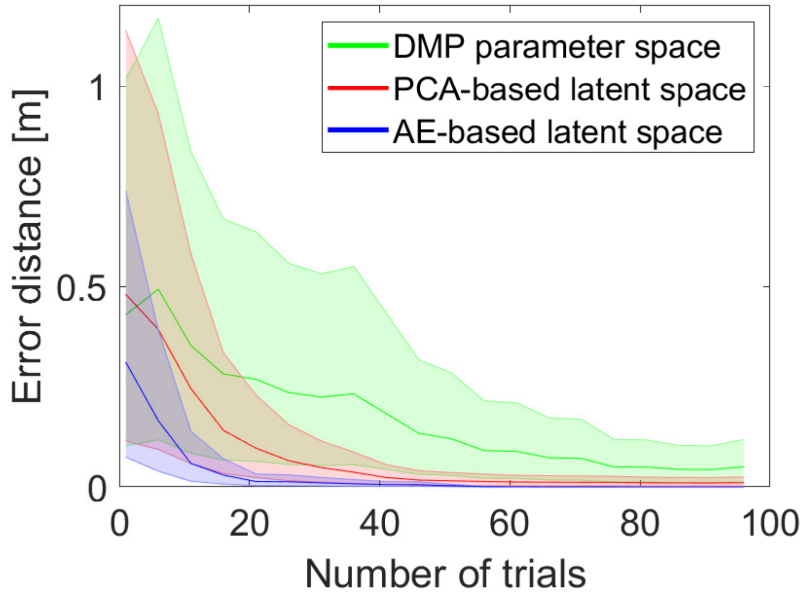


Figure 3.9: Convergence of reinforcement learning on simulated data. The green, red and blue lines show the average error distance after the specified number of trials for DMP representation, PCA-based and AE-based latent space representation. The shaded area is the 95% confidence interval for the exponential distribution of the results.

is also more stable as can be observed from the size of the 95% confidence interval. The reason for this is that RL in the full parameter space has a considerable chance to produce throws in which the robot loses the ball before making the actual throw. On the other hand, 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.

3.5.2.2 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 distance d^{T} and the ball landing distance d_i. The terminal reward for reinforcement learning was thus computed as

$$R(\tau_i) = \exp(-(d^{\mathrm{T}} - d_i)^2). \tag{3.34}$$

The size of the basket and the ball was 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 (Eq. 3.8) for each of the two targets 5 times, i. e., a total of 10, and obtained results shown in Fig. 3.11 and Fig. 3.12.

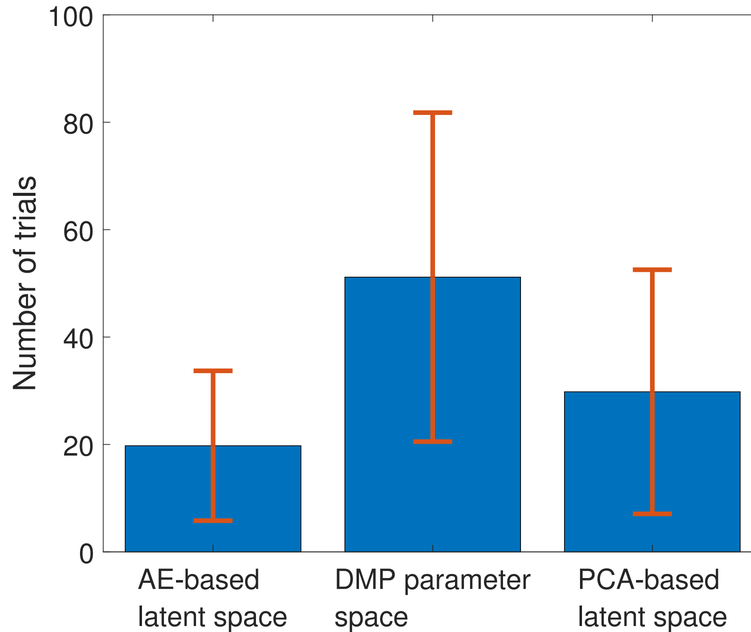


Figure 3.10: Number of rollouts to the first hit for reinforcement learning on simulated data. The bars show the average number of rollouts to the first hit for reinforcement learning with different trajectory representations. A throw was counted as a hit if the computed distance was less than 2 cm. Error bars show the variance of the results.

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.

3.5.3 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 Section 3.2.2) to exploit previous knowledge when throwing at new targets. In this section, we evaluate the accuracy of GPR for different skill representations.

3.5.3.1 Performance of GPR

In simulation we tested the performance of Gaussian process regression (GPR). The data within the black square in Fig. 3.4 was used for training and the data within the red dashed square for testing. The data at the edge of the training area 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. 3.8). For each trajectory representation, we used

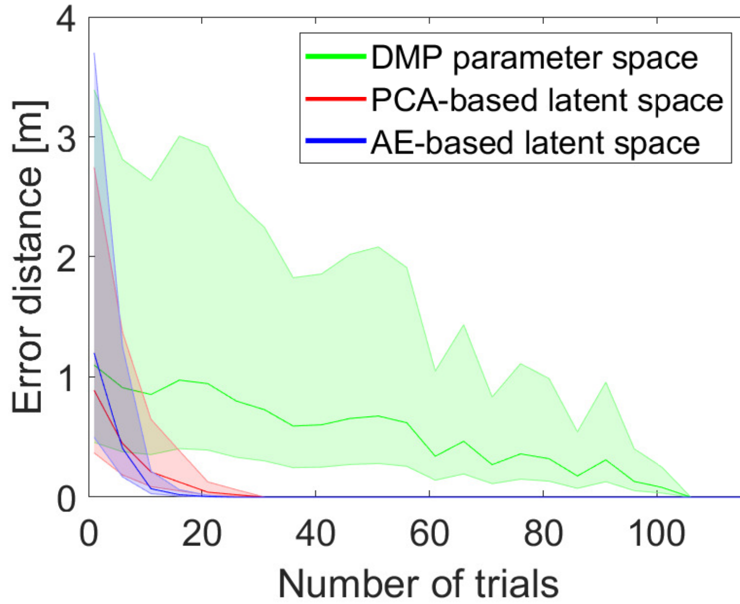


Figure 3.11: Convergence of reinforcement learning of throwing movements on a real robot. The green, red and blue lines show the average error distance after the specified number of trials for DMP representation, PCA-based and AE-based latent space representation. The shaded area is the 95% confidence interval for the exponential distribution of the results.

the corresponding dataset defined in (3.29) – (3.31) to calculate GPR mappings (3.16) – (3.18). 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 Tab. 3.2, where the average error and its variance for each representation are shown. In Figure 3.13, the throwing errors are presented with a contour plot.

The results in Tab. 3.2 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. 3.13, we can observe uniformly low errors for GPR in the AE-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.

Table 3.2: 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. The results show 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

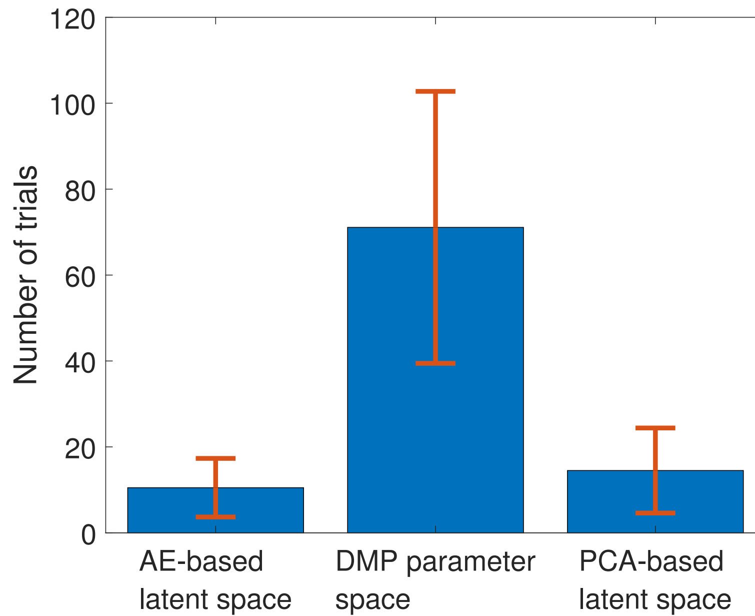


Figure 3.12: Number of rollouts to the first hit for reinforcement learning of throwing movements on a real robot. The bars show the average number of rollouts to the first hit for reinforcement learning with different trajectory representations. Error bars show the variance of the results.

3.5.3.2 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 procedure in Algorithm 3.1. 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. 3.4. 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 datasets became larger, the error of throws decreased for all three skill representations, as shown in Fig. 3.14 and in Fig. 3.15.

The simulation results shown in Fig. 3.14 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.

On the real robot, we performed a similar experiment for targets distributed along the line at the distance from 2m to 4m. We first performed 2 throws that 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. 3.15. 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

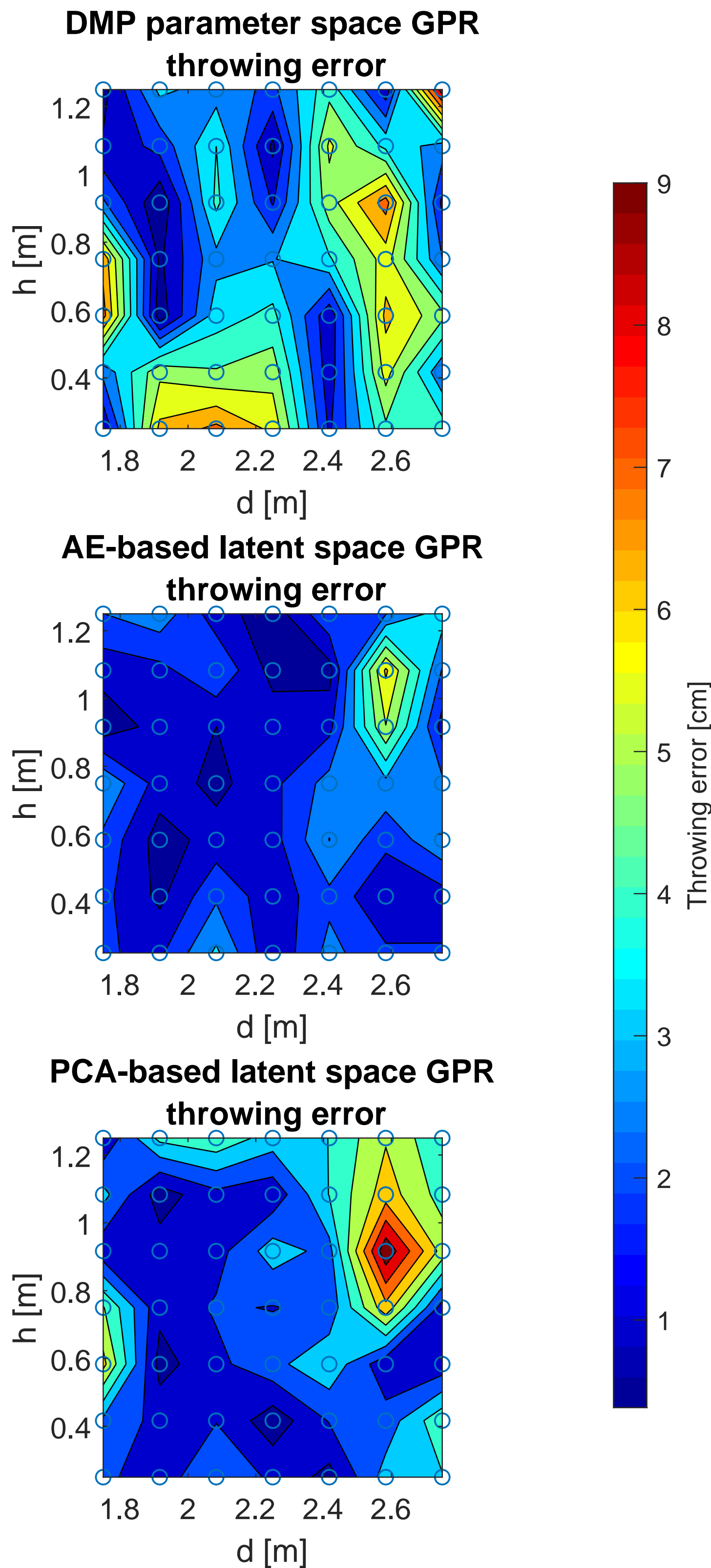


Figure 3.13: Contour plots showing the throwing error resulting from the throwing trajectories computed by GPR in different learning spaces. Circles represent different targets on a 7×7 target grid. Errors are in centimeters and represent the distance between the desired target and the closest point on the ball trajectory.

Algorithm 3.1: Incremental dataset augmentation.

Data: training set $\{\boldsymbol{\theta}_k, \boldsymbol{q}_k\}_{k=1}^{m}$, number of throws R
Result: distance d for each throw, GPR parameters computed from $\{\boldsymbol{\theta}_k, \boldsymbol{q}_k\}_{k=1}^{R}$

1 compute GPR parameters using the initial training set $\{\boldsymbol{\theta}_k, \boldsymbol{q}_k\}_{k=1}^{m}$;
2 **while** $m \leq R$ **do**
3   generate a random target position $\boldsymbol{q}_T$ within the selected training area;
4   compute the corresponding throwing trajectory $\boldsymbol{\theta}_{m+1}$ using GPR in the appropriate parameter space;
5   execute the throwing trajectory in dynamic simulation or with a real robot;
6   measure the actual ball landing position $\boldsymbol{q}_{m+1}$ and compute the distance between the actual landing position and the desired target position $\boldsymbol{q}_T$;
7   augment the training dataset $\{\boldsymbol{\theta}_k, \boldsymbol{q}_k\}_{k=1}^{m+1}$ and compute the new GPR parameter;
8   increase m;
end

a small statistical sample and simplification of the experimental task to one dimension.

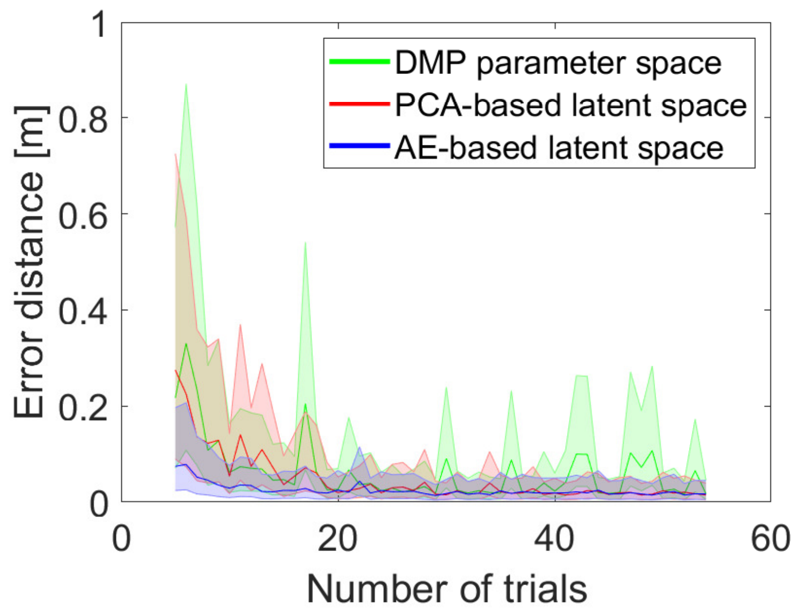


Figure 3.14: Improved performance of statistical learning by autonomous dataset augmentation procedure for experiment in the simulation. The green, red, and blue lines show the average error of throwing for different trajectory representations as the function of the number of points added to the training set used to compute GPR. The shaded areas show 95% confidence interval for the exponential distribution of the results.

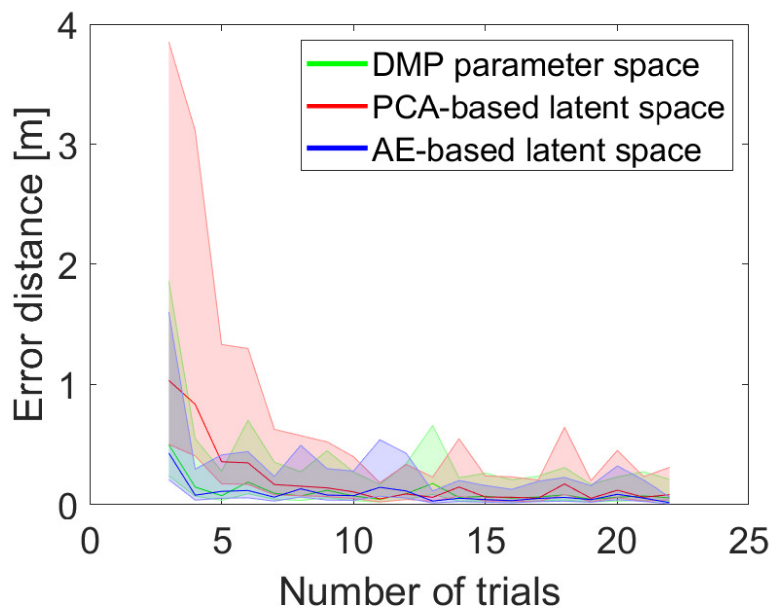


Figure 3.15: Improved performance of statistical learning by autonomous dataset augmentation procedure on a real robot. The green, red, and blue lines show the average error of throwing for different trajectory representations as the function of the number of points added to the training set used to compute GPR. The shaded areas show 95% confidence interval for the exponential distribution of the results.

Chapter 4

Conclusions

All of the initially specified hypotheses and goals were successfully addressed in this thesis. With this we have archived the following new scientific contributions:

- We developed a new image-to-motion deep neural network architecture that can generate a DMP-encoded motion trajectory at the neural network output from an image at the neural network input.
- We improved the training of the proposed network architecture by developing a loss function that calculates the real physical error between the motion trajectories at the network output and deriving its derivatives for backpropagation.
- We extended the proposed methodology, including the loss function, to networks that have AL-DMPs at the neural network output instead of standard DMPs.
- We extended the proposed image-to-motion neural network so that it became possible to process input images of variable sizes and containing cluttered backgrounds.
- We improved and accelerated robot skill learning by projecting DMPs to low-dimensional latent spaces computed by autoencoder deep neural networks.
- We have shown that autoencoders trained on simulated datasets can be used for learning skills with a real robot.

In the following we explain in more detail how these new scientific contributions were achieved.

In the second chapter, we developed a deep neural network architecture capable of generating a DMP-encoded motion trajectory from an input image, as specified in the first thesis goal **G1**. We also proposed a new approach for training deep neural networks that compute DMP parameters, as specified in the thesis goal **G2**. Our results (Section 2.5.5) demonstrate that the performance of such neural networks can be significantly improved by specifying an appropriate loss function that measures a real physical distance between the DMP-generated trajectories and the training trajectories as opposed to using the simple Euclidean distance between the DMP parameters, which has no physical meaning. This way we confirmed hypothesis ***H2***. We derived the formulas for the computation of gradients of the newly proposed loss function, which is necessary to apply 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, as we hypothesized in ***H1***.

In the same chapter, we also extended our approach for training deep neural networks that return DMP parameters as output, to training deep neural networks with normalized AL-DMP parameters at the output, corresponding with goal **G3**. For this purpose, we derived the gradients of the loss function based on normalized AL-DMPs, which is necessary for the application of the backpropagation algorithm for neural network training. We confirmed experimentally (Section 2.5.6) that the performance of neural networks with normalized AL-DMP parameters at the output can be improved by the proposed loss function, which measures the real physical distances between the normalized AL-DMP generated paths and training paths. This way we confirmed hypothesis ***H3***. Furthermore, we showed that neural networks with normalized AL-DMPs at the output can significantly outperform the networks that output DMP parameters, by an order of magnitude in cases where training data consists of trajectories with inconsistent temporal course of motion.

Compared to DMPs, the AL-DMPs and the normalized AL-DMPs are limited to paths with no sharp corners. This is because a real robot must stop if it is to change its direction of motion abruptly. This is handled in standard DMPs by reducing the speed of motion, but speed is separate in AL-DMPs. This AL-DMP limitation could be overcome by segmenting such paths into parts without sharp corners and describing each part with a separate AL-DMP in the training data. On the other hand, DMPs offer no solution to deal with trajectories of varying speed, which often occur in handwriting. Thus the AL-DMP representation is still advantageous. The currently applied neural network architectures offer no direct option to generate different numbers of AL-DMPs on the neural network output. A possible solution is to first train a classification neural network that identifies the observed digit and then apply a neural network trained for each digit. For example, the neural network trained for digit five could output three AL-DMP segments as there are two sharp corners in digit five. Another possibility is to use an architecture similar to the one proposed by He et al. [65], where a variable number of parts from the input feature map can be used for prediction. Such an architecture would enable us to deal with multiple digits in the input image or generate a variable number of handwriting movements to avoid sharp corners.

We achieved our thesis goal **G4** with the development of the new VIMEDNet architecture that can process input images of different sizes and outputs either DMP or normalized AL-DMP parameters. We proved this with experiments in Section 2.5.7 and confirmed the associated hypothesis ***H4***. The proposed architecture can also handle noise and images with a significant portion of background pixels that do not belong to the observed digit. VIMEDNet can be trained on small size images and then directly applied to larger input images. This approach enables more efficient training of neural networks. Besides evaluation experiments with synthetic data, we also performed some real robot writing experiments where we show that VIMEDNet can reproduce the observed digits. In this experiment, we assumed that the orientation and size of the digit in the input image is constant. This limitation can be solved by two approaches. Either we can create a dataset that contains these variations, or we extend the architecture even further and deal with this by integrating the approach of Ridge et al. [108] into VIMEDNet. The architecture proposed in [108] can handle variations in digit size and orientation but is limited to fixed-size input images if not extended by VIMEDNet.

There are many possible extensions of our work. The first of these is the possibility to use a neural network that also outputs AL-DMP velocity weights, allowing temporal and spatial aspects of motion to be learned simultaneously but separately. Another possibility is to learn normalized AL-DMP parameters for each degree of freedom separately. In the normalization, we omit the common parameter L, which would require the values of all other AL-DMP parameters for its gradient computation, as in the case of the DMP

parameter τ in (2.54) – (2.57). For normalized AL-DMPs, the computation of each path degree of freedom depends only on the parameters of that degree, which allows us to use a separate neural network for each path degree of freedom. In the context of handwriting, other authors have shown that recurrent neural networks (RNNs) can be applied for the generation of handwritten digit images [109]. It is indeed possible to construct a recurrent neural network that outputs handwriting DMPs or normalized AL-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. However, RNNs are usually more difficult to train than the proposed feedforward neural network architectures.

Besides handwriting, the proposed approach can be applied also to other tasks, e.g. 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 or normalized AL-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 or normalized AL-DMP parameters as output, including recurrent neural networks such as LSTM networks [110].

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) [111] or image style transfer [112] 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.

In the third chapter, we showed how to improve the robot skill learning by reducing the learning dimensions. We accomplished this by computing latent spaces defined by autoencoder neural networks as specified in goal **G5**. We demonstrated the advantages of motor learning in low-dimensional latent spaces compared to learning in the full motor parameter space, e. g. DMP parameter space. We showed that both reinforcement and statistical learning can be more effective in latent spaces. 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. These results confirm hypothesis ***H5***.

Even though the data for computing latent spaces in all experiments mentioned above were generated in simulation, the computed latent spaces were successfully used also to increase the performance of learning on the real robot. In this way, we simultaneously addressed goal **G6** and confirmed corresponding hypothesis ***H6***.

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. One

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 [113]. We believe that by implementing this approach for our AEs, e.g. by making use of denoising AEs or regularization of latent space [114], we can create even better AE latent spaces for motor learning. This issue will be addressed in 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 Section 3.4.3. 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.

Statistical learning provides a task policy for a particular environment. When the environment changes (different ball, different air drag, etc.), we either need to acquire new query points and learn the mapping from scratch, or we can add the new values describing the environment to the training dataset. In both cases, we can adapt to the new environment or even generalize to changes in the environment better and faster if only a small number of data-points is needed for generalization. The AE-based trajectory representation also offers good opportunities for future research.

Another venue for future 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 demonstrated in [70]. The latent space of the variational autoencoder could be used in the same way as our current AE-based latent space.

References

[1] S. Schaal, "Is Imitation Learning the Route to Humanoid Robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[2] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[3] G. Neumann, C. Daniel, A. Paraschos, A. Kupcsik, and J. Peters, "Learning modular policies for robotics.," *Frontiers in computational neuroscience*, vol. 8, no. June, p. 62, 2014.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, p. 0 278 364 913 495 721, 2013.

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

[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, Mar. 2011.

[7] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks.," *Nips*, pp. 1–9, 2009.

[8] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic Modeling Using Deep Belief Networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, Jan. 2012.

[9] Q. V. Le, "Building high-level features using large scale unsupervised learning," *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*, pp. 8595–8598, 2013.

[10] K. Sohn, D. Y. Jung, H. Lee, and A. O. Hero, "Efficient learning of sparse, distributed, convolutional feature representations for object recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2011, pp. 2643–2650.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, Lake Tahoe, Nevada, 2012, pp. 1097–1105.

[12] I. Kuzovkin, R. Vicente, M. Petton, J.-P. Lachaux, M. Baciu, P. Kahane, S. Rheims, J. R. Vidal, and J. Aru, "Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex," *Communications Biology*, vol. 1, no. 107, pp. 1–12, 2018.

[13] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *CoRR*, vol. abs/1811.12560, 2018. arXiv: 1811.12560.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[16] A. Byravan and D. Fox, "SE3-Nets: Learning Rigid Body Motion using Deep Neural Networks," p. 8, 2016. eprint: `1606.02378`.

[17] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[18] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, Sep. 2016, pp. 3406–3413.

[19] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 2016. eprint: `1504.00702`.

[20] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," *arXiv*, pp. 1–1, 2016. eprint: `1603.02199`.

[21] S. James and E. Johns, "3D Simulation for Robot Arm Control with Deep Q-Learning," 2016. eprint: `1609.03759`.

[22] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *CoRR*, vol. abs/1710.06542, 2017. arXiv: `1710.06542`.

[23] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *CoRR*, vol. abs/1806.10293, 2018. arXiv: `1806.10293`.

[24] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman, *Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning*, 2020. arXiv: `2004.10190`.

[25] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, IEEE, May 2016, pp. 512–519. eprint: `1509.06113`.

[26] N. Chen and J. Peters, "Stable Reinforcement Learning with Autoencoders for Tactile and Visual Data Stable Reinforcement Learning with Autoencoders for Tactile and Visual Data," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2016.

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

[28] 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.

[29] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[30] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[31] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 1071–1079.

[32] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, New York, 2016, pp. 2850–2869.

[33] S. Schaal, P. Mohajerian, and A. J. Ijspeert, "Dynamics systems vs. optimal control — a unifying view," in *Computational Neuroscience: Theoretical Insights into Brain Function*, Elsevier, 2007, pp. 425–445.

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

[35] A. Ude, R. Vuga, B. Nemec, and J. Morimoto, "Trajectory representation by nonlinear scaling of dynamic movement primitives," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, 2016, pp. 4728–4735.

[36] T. Gašpar, B. Nemec, J. Morimoto, and A. Ude, "Skill learning and action recognition by arc-length dynamic movement primitives," *Robotics and Autonomous Systems*, vol. 100, pp. 225–235, 2018.

[37] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 297–330, 2020.

[38] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[39] R. Dillmann, "Teaching and learning of robot tasks via observation of human performance," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 109–116, 2004.

[40] B. Nemec, R. Vuga, and A. Ude, "Efficient sensorimotor learning from multiple demonstrations," *Advanced Robotics*, vol. 27, no. 13, pp. 1023–1031, 2013.

[41] A. Alissandrakis, C. L. Nehaniv, and 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.

[42] 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.

[43] J. Kober, D. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, no. 11, pp. 1238–1274, 2013.

[44] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, no. 1-2, pp. 171–203, 2011.

[45] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, pp. 388–403, 2013.

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

[47] I. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, p. 20 150 202, Apr. 2016.

[48] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: Incremental real time learning in high dimensional space," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 1079–1086.

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

[50] T. Petrič, A. Gams, L. Colasanto, A. J. Ijspeert, and A. Ude, "Accelerated sensorimotor learning of compliant movement primitives," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1636–1642, Dec. 2018.

[51] S. Thrun and T. M. Mitchell, "Lifelong robot learning," *Robotics and Autonomous Systems*, vol. 15, no. 1, pp. 25–46, 1995, The Biology and Technology of Intelligent Autonomous Agents.

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

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

[54] A. Pervez, Y. Mao, and D. Lee, "Learning deep movement primitives using convolutional neural networks," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017, pp. 191–197.

[55] W. Kim, C. Lee, and H. J. Kim, "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 (IROS)*, 2018, pp. 3117–3123.

[56] Y. Zhou, J. Gao, and T. Asfour, "Movement primitive learning and generalization: Using mixture density networks," *IEEE Robotics & Automation Magazine*, pp. 2–12, 2020.

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

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

[59] K. Gregor, I. Danihelka, A. Graves, D. Jimenez Rezende, and D. Wierstra, "DRAW: A Recurrent Neural Network For Image Generation," in *International Conference on Machine Learning (ICML)*, Lille, France, 2015, pp. 1462–1471.

[60] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[61] 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, 2017.

[62] O. Ali, "Robotic calligraphy: Learning from character images," Master's thesis, TU Dortmund and TU Munich, 2015.

[63] A. Kotani and S. Tellex, "Teaching robots to draw," in *IEEE International Conference on Robotics and Automation ICRA*, Montreal, Canada, 2019, pp. 4797–4803.

[64] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *arXiv e-prints*, arXiv:1406.4729, arXiv:1406.4729, 2014.

[65] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *IEEE Int. Conf. on Computer Vision (ICCCV)*, 2017, pp. 2980–2988.

[66] M. Lin, Q. Chen, and S. Yan, "Network In Network," *arXiv e-prints*, arXiv:1312.4400, arXiv:1312.4400, 2013.

[67] K. S. Luck, G. Neumann, E. Berger, J. Peters, and H. B. Amor, "Latent space policy search for robotics," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Sep. 2014, pp. 1434–1440.

[68] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and 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.

[69] 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)*, Nov. 2015, pp. 434–440.

[70] N. Chen, M. Karl, and 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.

[71] D. M. Wolpert, J. Diedrichsen, and J. R. Flanagan, "Principles of sensorimotor learning," *Nature Reviews Neuroscience*, vol. 12, no. 12, pp. 739–751, 2011.

[72] B. Nemec, D. Forte, R. Vuga, M. Tamošiūnaitė, F. Wörgötter, and A. Ude, "Applying statistical generalization to determine search direction for reinforcement learning of movement primitives," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, 2012, pp. 65–70.

[73] A. Colomé and C. Torras, "Dimensionality reduction and motion coordination in learning trajectories with dynamic movement primitives," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1414–1420.

[74] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, 2012.

[75] T. Matsubara, S.-H. Hyon, and J. Morimoto, "Learning parametric dynamic movement primitives from multiple demonstrations," *Neural Networks*, vol. 24, no. 5, pp. 493–500, 2011.

[76] M. Deniša, A. Gams, A. Ude, and T. Petrič, "Learning compliant movement primitives through demonstration and statistical generalization," *IEEE/ASME Transactions on Mechatronics*, vol. 21, no. 5, pp. 2581–2594, 2016.

[77] A. Kramberger, A. Gams, B. Nemec, D. Chrysostomou, O. Madsen, and A. Ude, "Generalization of orientation trajectories and force-torque profiles for robotic assembly," *Robotics and Autonomous Systems*, vol. 98, pp. 333–346, 2017.

[78] T. Gašpar, B. Nemec, J. Morimoto, and A. Ude, "Skill learning and action recognition by arc-length dynamic movement primitives," *Robotics and Autonomous Systems*, vol. 100, 2017.

[79] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2013, pp. 417–422.

[80] R. Reinhart and J. Steil, "Efficient policy search with a parameterized skill memory," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014, pp. 1400–1407.

[81] K. Muelling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[82] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2015.

[83] Y. Zuo, G. Avraham, and T. Drummond, *Traversing latent space using decision ferns*, 2018.

[84] L. Le, A. Patterson, and M. White, "Supervised autoencoders: Improving generalization performance with unsupervised regularizers," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 107–117.

[85] Y. Yoo, S. Yun, H. J. Chang, Y. Demiris, and J. Y. Choi, "Variational autoencoded regression: High dimensional regression of visual data on complex manifold," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 2943–2952.

[86] R. Pahič, A. Gams, A. Ude, and 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.

[87] R. Pahič, B. Ridge, A. Gams, J. Morimoto, and A. Ude, "Training of deep neural networks for the generation of dynamic movement primitives," *Neural Networks*, vol. 127, pp. 121–131, 2020.

[88] R. Pahič, A. Gams, and A. Ude, "Reconstructing spatial aspects of motion by image-to-path deep neural networks," *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 255–262, 2021.

[89] W. Suleiman, "On Time Parameterization of a Robot Path," *IFAC Papers OnLine*, vol. 48, no. 3, pp. 52–57, 2015.

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

[91] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[92] S. Basu, M. Karki, S. Ganguly, R. DiBiano, S. Mukhopadhyay, S. Gayaka, R. Kannan, and R. Nemani, "Learning Sparse Feature Representations Using Probabilistic Quadtrees and Deep Belief Nets," en, *Neural Processing Letters*, vol. 45, no. 3, pp. 855–867, 2017.

[93] F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[94] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS 2017 Autodiff Workshop: The future of gradient-based machine learning software and techniques*, 2017.

[95] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations (ICLR)*, San Diego, 2015.

[96] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[97] *Matlab and statistics toolbox 2019a*, The MathWorks, Inc., Natick, Massachusetts, United States.

[98] B. Ridge, R. Pahič, A. Ude, and J. Morimoto, "Convolutional encoder-decoder networks for robust image-to-motion prediction," in *Advances in Service and Industrial Robotics*, K. Berns and D. Görges, Eds., Cham: Springer International Publishing, 2020, pp. 514–523.

[99] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Soueres, N. Mansard, F. Lamiraux, J.-P. Laumond, L. Marchionni, H. Tome, and F. Ferro, "TALOS: A new humanoid research platform targeted for industrial applications," in *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, Birmingham, UK, 2017, pp. 689–695.

[100] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: An open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[101] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, Sep. 2004, 2149–2154 vol.3.

[102] R. Pahič, Z. Lončarević, A. Gams, and A. Ude, "Robot skill learning in latent space of a deep autoencoder neural network," *Robotics and Autonomous Systems*, vol. 135, 103690, 2021.

[103] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[104] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[105] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 5026–5033.

[106] B. Nemec, R. Vuga, and A. Ude, "Exploiting previous experience to constrain robot sensorimotor learning," in *11th IEEE-RAS International Conference on Humanoid Robots*, Oct. 2011, pp. 727–732.

[107] R. Pahič, *Throwing trajectories dataset*, `https://github.com/abr-ijs/Throwing-trajectories-dataset`, 2019.

[108] B. Ridge, R. Pahič, A. Ude, and J. Morimoto, "Learning to write anywhere with spatial transformer image-to-motion encoder-decoder networks," in *IEEE International Conference on Robotics and Automation ICRA*, Montreal, Canada, 2019, pp. 4797–4803.

[109] V. Goudar and D. V. Buonomano, "Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks," *eLife*, vol. 7, no. e31134, 2018.

[110] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[111] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2672–2680.

[112] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, Nevada: IEEE, 2016, pp. 2414–2423.

[113] M. Ranzato, Y.-L. Boureau, S. Chopra, and Y. LeCun, "A unified energy-based framework for unsupervised learning," in *Artificial Intelligence and Statistics*, M. Meila and X. Shen, Eds., ser. Proceedings of Machine Learning Research, San Juan, Puerto Rico, 2007, pp. 371–379.

[114] H. Kamyshanska and R. Memisevic, "The potential energy of an autoencoder," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6, pp. 1261–1273, 2015.

Bibliography

Publications Related to the Thesis

Journal Articles

R. Pahič, B. Ridge, A. Gams, J. Morimoto, and A. Ude, "Training of deep neural networks for the generation of dynamic movement primitives," *Neural Networks*, vol. 127, pp. 121–131, 2020.

R. Pahič, Z. Lončarević, A. Gams, and A. Ude, "Robot skill learning in latent space of a deep autoencoder neural network," *Robotics and Autonomous Systems*, vol. 135, 103690, 2021.

R. Pahič, A. Gams, and A. Ude, "Reconstructing spatial aspects of motion by image-to-path deep neural networks," *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 255–262, 2021.

Conference Paper

R. Pahič, A. Gams, A. Ude, and 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.

B. Ridge, R. Pahič, A. Ude, and J. Morimoto, "Convolutional encoder-decoder networks for robust image-to-motion prediction," in *Advances in Service and Industrial Robotics*, K. Berns and D. Görges, Eds., Cham: Springer International Publishing, 2020, pp. 514–523.

R. Pahič, Z. Lončarević, A. Ude, B. Nemec, and A. Gams, "User feedback in latent space robotic skill learning," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, 2018, pp. 270–276.

Other Publications

Journal Articles

Z. Lončarević, R. Pahič, A. Ude, and A. Gams, "Generalization-based acquisition of training data for motor primitive learning by neural networks," *Applied Sciences*, vol. 11, no. 3, pp. 1013-1-1013-17, 2021.

Conference Paper

B. Ridge, R. Pahič, A. Ude, and J. Morimoto, "Learning to write anywhere with spatial transformer image-to-motion encoder-decoder networks," in *IEEE International Conference on Robotics and Automation ICRA*, Montreal, Canada, 2019, pp. 4797–4803.

Z. Lončarević, R. Pahič, M. Simonič, A. Ude, and A. Gams, “Generalization based database acquisition for robot learning in reduced space,” in *Advances in service and industrial robotics : results of RAAD : proceedings of the 29th International Conference on Robotics in Alpe-Adria-Danube Region (RAAD 2020)*, S. Zeghloul, M. Amine Labiri, and J. S. Sandoval Arevalo, Eds., ser. Service and industrial robotics, vol. 84, Cham: Springer, 2020, pp. 496–504.

Z. Lončarević, R. Pahič, and A. Gams, “Learning of the velocity profile for quality inspection motion using power,” in *Zbornik devetindvajsete mednarodne Elektrotehniške in računalniške konference ERK 2020 = Proceedings of the Twenty-ninth International Electrotechnical and Computer Science Conference ERK 2020*, A. Žemva and A. Trost, Eds., ser. Zbornik ... Elektrotehniške in računalniške konference (Online), 29, Ljubljana: Slovenska sekcija IEEE, = Slovenian Section IEEE, 2020, pp. 156–159.

Z. Lončarević, R. Pahič, M. Simonič, A. Ude, and A. Gams, “Reduction of trajectory encoding data usng a deep auroencoder network : Robotic throwing,” in *Advances in service and industrial robotics : proceedings of the 28th International Conference on Robotics in Alpe-Adria-Danube Region (RAAD 2019)*, K. Berns and D. Görges, Eds., ser. Advances in intelligent systems and computing (Print), vol. 980, Cham: Springer, 2019, pp. 86–94.

Z. Lončarević, R. Pahič, M. Simonič, A. Ude, and A. Gams, “Learning of robotic throwing at a target using qualitative learning reward,” in *Proceedings*, B. Dumnić, Ed., Novi Sad: University, Faculty of Technical Sciences, 2019, p. 6.

B. Ridge and R. Pahič, “Learning robotic handwriting with convolutionalimage-to-motion encoder-decoder networks,” in *Robotika : zbornik 22. Mednarodne multikonference Informacijska družba - IS 2019, 11. oktober 2019 : zvezek G = Robotics : proceedings of the 22nd International Multiconference Information Society - IS 2019,11 October, 2019, Ljubljana, Slovenia : volume G*, A. Gams and A. Ude, Eds., ser. Informacijska družba, Ljubljana: Institut "Jožef Stefan", 2019, pp. 23–26.

Z. Lončarević, R. Pahič, G. Papa, and A. Gams, “Experimental evaluation of deep-learning applied on pendulum balancing,” in *Zbornik osemindvajsete mednarodne Elektrotehniške in računalniške konference ERK 2019 = Proceedings of the Twenty-eighth International Electrotechnical and Computer Science Conference ERK 2019*, A. Žemva and A. Trost, Eds., ser. Zbornik ... Elektrotehniške in računalniške konference (Online), 28, Ljubljana: Društvo Slovenska sekcija IEEE, 2019, pp. 219–222.

Z. Lončarević, R. Pahič, A. Ude, B. Nemec, and A. Gams, “Replacing reward function with user feedback,” in *Zbornik sedemindvajsete mednarodne Elektrotehniške in računalniške konference ERK 2018 = Proceedings of the Twenty-seventh International Electrotechnical and Computer Science Conference ERK 2018*, A. Žemva and A. Trost, Eds., ser. Zbornik ... Elektrotehniške in računalniške konference (Online), 27, Ljubljana: Društvo Slovenska sekcija IEEE, 2018, pp. 139–142.

R. Pahič, V. Tič, and D. Lovrec, “Test stand for determining the performance characteristics of hydraulic directional control valves,” in *Conference proceedings*, D. Lovrec and V. Tič, Eds., Maribor: University of Maribor Press, 2017, pp. 271–280.

M. Cevzar, R. Pahič, T. Petrič, R. Goljat, A. Ude, and J. Babič, “Comparison of classification methods for hip exoskeleton actuator control,” in *Zbornik šestindvajsete mednarodne Elektrotehniške in računalniške konference ERK 2017 = Proceedings of the Twenty-sixth International Electrotechnical and Computer Science Conference ERK 2017*, A. Žemva and A. Trost, Eds., ser. Zbornik ... Elektrotehniške in računalniške konference ERK ..., 26, Ljubljana: IEEE, Slovenska sekcija IEEE, 2017, pp. 229–232.

R. Pahič, B. Ridge, and A. Ude, "Deep encoder-decoder networkfor learning to write digits with a humanoid robot," in *Knjiga povzetkov : science of the future how to stay up-tod-date with your research! = Book of abstracts*, M. Topole, T. Turk Dermastia, M. Dežman, B. Škrlj, A. Jurov, K. Bačnik, J. Masten, Ž. Marinko, P. Jovičević Klug, R. Pahič, I. Rybkin, J. Černilogar, and A. Kikaj, Eds., Ljubljana: Mednarodna podiplomska šola Jožefa Stefana, = Jožef Stefan International Postgraduate School, Inštitut Jožef Stefan, = Jožef Stefan Institute, 2019, pp. 41–42.

Z. Lončarević, R. Pahič, M. Simonič, and A. Gams, "Human intuitive reward systems for reinforcement learning of robotic actions in latent space of deep auto encoder network," in *Knjiga povzetkov : science of the future how to stay up-tod-date with your research! = Book of abstracts*, M. Topole, T. Turk Dermastia, M. Dežman, B. Škrlj, A. Jurov, K. Bačnik, J. Masten, Ž. Marinko, P. Jovičević Klug, R. Pahič, I. Rybkin, J. Černilogar, and A. Kikaj, Eds., Ljubljana: Mednarodna podiplomska šola Jožefa Stefana, = Jožef Stefan International Postgraduate School, Inštitut Jožef Stefan, = Jožef Stefan Institute, 2019, p. 45.

R. Pahič and A. Ude, "Statistical generalization of robot movement in autoencoder latent space," in *Zbornik = Proceedings*, M. Pavlin, J. Kralj, B. Škrlj, A. Pecman, T. Gornik, D. Levovnik, M. Topole, T. Turk Dermastia, M. Bergant, and A. Jurov, Eds., Ljubljana: Mednarodna podiplomska šola Jožefa Stefana, = Jožef Stefan International Postgraduate School, Inštitut Jožef Stefan, = Jožef Stefan Institute, 2017, p. 41.

Biography

Education

1. 9. 2014 – 12. 9. 2016: Master degree: MAG.INŽ.STR.(UN), University of Maribor, Faculty of Mechanical Engineering
1. 9. 2012 – 3. 11. 2016: Undergraduate degree: DIPL.INŽ.MEH.(UN), University of Maribor, Faculty of Mechanical Engineering
1. 9. 2011 – 11. 9. 2014: Undergraduate degree: DIPL.INŽ.STR.(UN), University of Maribor, Faculty of Mechanical Engineering

Conferences

June 2019: Attended with a contribution (article and oral presentation) the *"28th International Conference on Robotics in Alpe-Adria-Danube Region, RAAD 2019"* in Kaiserslautern, Germany
April 2019: Co-organized and attended with a contribution (abstract, poster, elevator pitch) the *"11th Jožef Stefan International Post-graduate School Students Conference"* in Planica, Slovenia
November 2018: Attended with a contribution (article and oral presentation) the *"IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids 2018)"* in Beijing, China
May 2018: Attended with a contribution (article and interactive presentation) the *"2018 IEEE International Conference on Robotics and Automation"* in Brisbane, Australia
April 2017: Attended with a contribution (extended abstract, poster, elevator pitch) the *"9th Jožef Stefan International Post-graduate School Students Conference"* in Ljubljana, Slovenia

Research visits

August 2017 – September 2017: Research visit at the Advanced Telecommunications Research Institute International, Department of Brain Robot Interface, Kyoto, Japan.

Memberships

October 2018 – September 2020 *Jožef Stefan International Postgraduate School* student-council member and the student-council representative of the "Information and Communication Technologies" study program
February 2018 – Today IEEE Graduate Student Member