TECHNISCHE
UNIVERSITÄT
MÜNCHEN

TU MÜNCHEN

Fakultät für Informatik

EFFICIENT MOVEMENT REPRESENTATION AND PREDICTION WITH MACHINE LEARNING

Nutan Chen

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ-Prof. Dr. Stephan Günnemann

Prüfer der Dissertation:

1. Univ-Prof. Dr. Patrick van der Smagt
2. Univ-Prof. Dr. Alin Albu-Schäffer

Die Dissertation wurde am 10.08.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 08.02.2018 angenommen.

ABSTRACT

Machine learning is a powerful tool for movement modeling. Two kinds of movement modeling include static representation and dynamic prediction. In this study, these two methods are used to model different kinds of movements, including fingertip, human body and robot, described as follows.

First of all, "static movement" representation of the human finger is shown. Estimating human fingertip forces is essential to understanding force distribution in grasping and manipulation. Human grasping behavior can then be used to develop force and impedance-based grasping and manipulation strategies for robotic hands. However, estimating human grip force naturally is only possible with objects attached by instruments or gloves with sensors, thus greatly limiting the type of objects used. In this thesis, an approach is described, which uses images of the human fingertip to reconstruct grip force and torque at the finger.

Dynamic prediction of human and robot movements using time-dependent unsupervised learning is described. High dimensional movements, as they are found in robotics or humanoids, make finding efficient movement predictions difficult. Typically, they are either used in configuration or Cartesian space, but both approaches do not generalize well. Methods are therefore explored, which embed dynamic movement primitives or reinforcement learning into time-dependent unsupervised learning.

The methods were evaluated on a large number of experiments, involving a range of applications. With these approaches, the results were significantly better than prior works with respect to accuracy and scope of use.

ZUSAMMENFASSUNG

Maschinelles Lernen ist ein leistungsfähiges Werkzeug für die Bewegungsmodellierung. Zwei Arten der Bewegungsmodellierung sind die statische Darstellung und die dynamische Vorhersage. In dieser Studie werden diese beiden Methoden verwendet, um verschiedene Arten von Bewegungen zu modellieren, einschließlich der von Fingerspitzen, menschlichen Körper und Roboter, wie es im Folgenden beschrieben.

Als erstes wird die "statische Bewegungsdarstellung" des menschlichen Fingers gezeigt. Die geschätzte Krafteinwirkung der menschlichen Fingerspitze ist entscheidend für das Verständnis von Kraftverteilung beim Greifen und Manipulation. Menschliches Greifverhalten kann dann dafür verwendet werden, um Kraft und Impedanz basierten Strategien des Greifen und die Manipulation für Händen der Roboter weiter zu entwickeln. Allerdings ist natürlich Abschätzen menschlichen nur mit Objekten angebracht durch Instrumente oder Handschuh mit Sensoren. Deswegen, die Anwendung der verschiedene Objekts wird stark begrenzt. In dieser Arbeit wird eine Methode beschrieben. Diese Methode rekonstruieret die Griffkraft und den Drehmoment am Finger von Menschen durch die Bilder der Fingerspitze.

Eine dynamische Vorhersage von Mensch-und Roboter-Bewegungen wird durch die verwendung von zeitabhängigem unüberwachtem Lernen beschrieben. Hohe Dimension Bewegungen, wie sie in Roboter oder Humanoiden zu finden sind, sind schwer für eine effiziente Bewegung prognose zu machen. Typischerweise werden sie entweder in der Konfiguration oder cartesianischen Raum verwendet, aber beide Ansätze lassen sich schwer zu generaliseren. Deshalb werden Methoden erforscht, die Dynamic Movement Primitives oder Reinforcement Learning in zeitabhängiges unüberwachtes Lernen einbetten.

Die Methoden werden, von einer großen Anzahl von Experimenten ausgewertet, die eine Vielzahl von Anwendungen enthalten. Mit diesen Ansätzen wurden deutlich besser Ergebnisse als in früheren Arbeiten erhalten, in Bezug auf Genauigkeit und Umfang der Nutzung.

ACKNOWLEDGEMENTS

This thesis contains the work I conducted at Biomimetic robotics and maching learning (BRML) over the past years.

I must thank my supervisor, Patrick van der Smagt, for endless inspirations, comments, and ideas. I learned from him the research methods and the technical knowledge.

I am also grateful to the lab mates, Agneta Gustus, Benedikt Staffler, Christian Osendorfer, Daniela Korhammer, Grady Jensen, Hannes Hoeppner, Justin Bayer, Markus Kuehne, Marvin Ludersdorfer, Max Karl, Max Soelch, Rachel Hornung and Sebastian Urban. Working with them over the past years has influenced my thinking. I also would like to thank other former and current lab members.

Besides BRML members, I would like to acknowledge other co-authors of publications, Benoni B.Edin, Goeran Westling, Herke van Hoof and Jan Peters. Their collaboration and suggestions enlightened me. Words cannot express how grateful I am to them.

This work was supported in part by the TACMAN project, EC Grant agreement no. 610967, within the FP7 framework programme.

CONTENTS

Part I

INTRODUCTION

1 OVERVIEW

The ability to efficiently represent and predict complex movements plays a crucial role for automatically observing and interpreting human or robot movements. The data from a human or a robot that has a large number of sensors may contain redundant and uncorrelated information. Because of this, movement representations and predictions from this data can be entangled and hide relevant factors, which can significantly influence the performance of the movement models. How can we disentangle this? To be able to deal with such data incrementally we introduce some local terminology: static vs. dynamic movement. With *static* movement we refer to data sets with i.i.d. (independently and identically distributed) data, where there is no temporal dependency in subsequent data. This is true when, e.g., comparing pictures randomly chosen from ImageNet, but not true when considering subsequent frames from a movie. The latter we describe with *dynamic* data, where there exists a temporal dependency in subsequent data points.

The thesis aims at obtaining: (1) efficient representation of "static movement" [1] from high-dimensional data, (2) prediction for dynamic movements from temporal data, using machine learning. To study the behavior of human/robot movements, new machine-learning methods for movement modeling are developed in this thesis.

Representation of "static movement" aims at mapping the sensory input to labels or latent spaces. The labels can be the motion classes, emotion classes and so on. Supervised learning such as Gaussian process (Rasmussen and Williams, 2006) and convolutional neural networks (CNN) (LeCun et al., 1998) can adequately represent the labeled "static movement". Unsupervised learning, such as Gaussian Process Latent Variable Model (GPLVM) (Lawrence, 2005) and autoencoder (Rumelhart et al., 1988; Bourlard and Kamp, 1988; Bengio, 2009), represents the data in a compact dimensional latent space, and is able to extract important information from the raw data. Thereafter, the latent values can be used for regression or classification.

1 In a "static movement", the data points are assumed to be independent and the time-dependent structure of the data is ignored.

In a dynamic movement, the independence assumption no longer holds and our models need to represent the series rather than single points, which requires time-series models such as hidden Markov Move (HMM) (Rabiner, 1989) and recurrent neural networks (RNN) (for an overview, see e.g., (Medsker and Jain, 1999)). Additionally, the task becomes challenging when the data is high-dimensional. Therefore, dynamic modeling methods in latent space such as Gaussian Process Dynamical Model (GPDM) (Wang et al., 2008) and Conditional Restricted Boltzmann Machine (CRBM) (Schölkopf et al., 2007) were explored, which provide an expression for both the observed data and its latent representation. A corresponding dynamic model is able to structure the latent space for temporal data, rather than encode the individual data patterns.

In the following parts of this chapter, a non-mathematical overview of machine learning methods for human/robot movement representation and prediction will be provided. The approaches and outline of this thesis will also be described.

1.1 RELATED WORK

1.1.1 *Machine learning*

A large part of animal cognition can be captured by correctly modeling the related data, which is what machine learning does. In contrast to a hand-coded system, learning from data using algorithms, machine learning aims at finding hidden insights and making prediction for unseen data. Machine learning has been central to artificial intelligence (AI) (Turing, 1950) and has been the foundation of a variety of application domains, such as speech recognition, image recognition, financial engineering, and so on.

A simple machine-learning algorithm such as linear regression is able to do document classification or email spam filtering. In some domains, computers outperform humans. For instance, AlphaGo of Google DeepMind, learned from human expert games, beat the world campion in Go in 2016 (Silver et al., 2016).

Since the last decade, deep learning has been greatly improved. Deep learning (LeCun et al., 2015) is a subfield of machine learning, which has architectures of multiple neural network layers with nonlinear processing. It learns multiple levels, constructing a hierarchy of concepts, of representations of data. Benefitting from large data sets and powerful computation, deep learning is able to tackle many more complicated tasks. For instance, He et al. (2016) built neural nets with a depth up to 152 layers and won the first place in several main categories of image recognition challenges in 2015. The above AlphaGo is another successful example of deep learning, in this case combined with reinforcement learning. Goodfellow et al. (2016) and Li and Dong (2014) present an overview on deep learning.

Other methods, including reinforcement learning (RL) and Gaussian process (GP), are promising approaches to improve machine learning. See e.g., (Sutton and Barto, 2012; Rasmussen and Williams, 2006) for more information on RL and GP, and e.g., (Murphy, 2012; Bishop, 2006) on other subfields of machine learning.

1.1.2 *Movement modeling and analysis*

Movement modeling is a crucial part of animation, physical rehabilitation, human-robot interaction, robot motion planning, and so on. The modeling processes consist of movement capture, pose modeling and movement modeling. Since data capture is time-consuming and expensive, the movement capture data is limited; therefore, generalization of novel movement and adaptation for unknown environment from limited existing captured data are important and remain challenging.

Robots are able to capture their motion by reading from their sensors, while human motion capture is more complicated. Human motion capture can be separated into optical systems and non-optical systems. Optical systems include marker-based method such as (Welch and Foxlin, 2002) and markerless method such as multicamera (Kanade and Narayanan, 2007), Kinect (Wilson, 2010; Chen et al., 2012), Leap motion (Weichert et al., 2013), and so on. Non-optical systems (e.g., CyberGlove, and Electromyography (EMG) (Vogel et al., 2011)) are able to detect more types of data such as inertial and force. Motion capture data, however, usually is not directly applied to use cases, since the manual modification of the data is very limited.

With the captured sensor data, pose modeling is then to represent the pose. The human/robot configuration can be constructed from the sensor readings. Pose modeling techniques include human body representations (O'Brien et al., 2000), facial representations (Igarashi et al., 2005), and mesh representations (Allen et al., 2003).

Physical modeling and statistical modeling are two main categories of dynamic movement modeling. The former is a traditional approach, which satisfies the principles of physics and generates natural movements. For instances, Popović and Witkin (1999) proposed muscle exertion, Fang and Pollard (2003) introduced joint angle acceleration, and Neff and Fiume (2002) presented muscle and spring in motion. A typical physical modeling approach is based on optimization algorithms. With pyshical constriants such as spacetime, kinematic and dynamics, optimization-based algorithms can generate realistic motions. Unfortunately, physical-based methods have shortcomings: 1) these methods are highly sensitive to the hand-crafted constraints; 2) high dimensional data significantly increases the modeling difficulty.

An alternative approach of movement modeling is based on statistical models. One example is motion graphs (Kovar et al., 2002). The motion graph extracts sub-motions from existing motions, and generates specification motions from the original motion and the transitions between motions. However, motion graph is restricted by the recorded motions. Alternative statistical models, machine learning-based approaches, dramatically improve the generalization capability of movement modeling using PCA, GPLVM and so on. In addition, machine learning is able to model the complexities of the real dynamics and generate realistic motions. See Section 1.1.3 for more information on machine learning-based movement modeling.

1.1.3 *Machine learning-based movement modeling*

Given a data set of movements, how can we efficiently represent the states of the movements and predict the next state based on previous and current states using machine learning? In the following, Gaussian Process (GP)-based and deep learning-based methods to solve these issues are mainly introduced.

Gaussian Process based methods

Gaussian Process (GP) (Rasmussen and Williams, 2006) based models are widely used for movement modeling. A GP is a stochastic process defined particularly by its mean and covariance functions. A basic assumption of GPs is that closely located inputs behave similarly. Accordingly, training inputs which are close to a test input have a similar target value as the test input. Gaussian Process Latent Variable Model (GPLVM) (Lawrence, 2005) is an unsupervised learning method that learns a low-dimensional representation of the data. In this method, a GP smoothly maps the latent data to the observation space. It optimizes a maximum a posteriori of the latent representation while estimating the hyper-parameters. When an out-of-sample data is mapped, the GPLVM optimizes the latent representation of the data. GPLVM is able to autonomously determine the dimensionality of the latent variables using Automatic Relevance Detection (ARD) kernel. It generalizes well even with a small set of training data. Titsias and Lawrence (2010) further developed GPLVM using variational Bayes by marginalizing of the latent variables. Both GP and GPLVM assume that data are independent, and they do not consider temporal continuity of data. They perform well on static data. Various dynamic models are then developed from GP and GPLVM.

Bitzer et al. (2008) used GPLVM to reduce the dimensions of "static movements". In the low-dimensional space, they then modeled the dynamic movements using dynamic movement primitive (DMP). Furthermore, Bitzer and Vijayakumar (2009) developed this method using simple sequence priors (SS-GPLVM), which generate movements that accurately follow the desired trajectory, e.g, the interpolated movements. In high dimensionality, DMP is possible to produce unrealistic movements. Instead, SS-GPLVM reduces the dimensions, and can thus reconstruct natural movements.

GPDM (Wang et al., 2008) extends GPLVM by mapping the data from observation space to a nonlinear dynamical system in the latent space. Using Markov chains, GPDM forces the latent space to be smooth. Besides mapping from the latent space to the observation space, GPDM maps the latent values at the previous time step to the latent values at the current time step. The parameters are marginalized out using GP priors for the two mappings. GPDM results in a higher probability of a density function to movements which are close to the training data. GPDM can be trained from a small data set. Applied to human motion, it is able to model 50-dimensional data in 3D latent space and fill in missing frames of a motion. Intention-driven Dynamics model (IDDM) (Wang et al., 2013) is an extension of GPDM to human-robot interaction. Human movement drives the dynamics in the low-dimensional space.

Darby et al. (2009) proposed hierarchical GPLVM (H-GPLVM). The model descends from a top-level node through the hierarchy to the leafs. As an example of human movement modeling, the leafs of the lowest level are the 3D joint angles of human. It is able to search for subtrees such as arms and legs, which are assumed to be independent from each other. For instance, the model explores a new movement of waving hands and walking, while only separate movements of walking and waving hands are provided by the training data.

A problem regarding the GPLVM family is the sensitivity to the initial guess. Additionally, different from parametric GP-based models which store all training data, the following deep-learning models are parametric and able to train on large datasets.

Deep-Learning based methods

A convolutional neural network (CNN) (LeCun et al., 1998) is a neural network architecture for regression and classification that is relatively robust to shifts, scales and distortions of the input data and can be trained efficiently on large data sets. Tompson et al. (2014) proposed a method consisting of a CNN and a Markov Random Field for human body pose recognition from videos. Another extension of CNNs is to extract features and then combine them with time series methods, for instance, recurrent neural networks (RNNs) (Fragkiadaki et al., 2015).

RNNs are a method that can model dynamic temporal movements. RNNs, which are widely used for tasks such as handwriting recognition and speech recognition, process a sequence of inputs using internal memory. As described in this section, an RNN can be developed to be more advanced by combining with representation learning such as VAEs and CNNs.

Building on the representational power of deep neural networks, Schölkopf et al. (2007) have obtained competitive results using CRBMs. RBMs are bidirectionally connected networks based on a probabilistic model and are simplified by restrictions on the networks. The CRBM is modified from the RBM with autoregressive connections. The CRBM is a generative model with the past n and m time steps of visible units to the current visible unit and the current hidden unit, respectively. The model is able to effectively learn different movements, and can smoothly transit between them. The authors also consider stacking several CRBMs to achieve a higher-level motion model. Taylor et al. (2010) further developed CRBMs to Implicit Mixture of Conditional Restricted Boltzmann Machines (imCRBM) for tackling multiple activities. Boulanger-Lewandowski et al. (2012) explored RNN-RBM, which is a probabilistic model based on RNNs. These methods, however, rely on binary latent variables. It is questionable whether they pose a reasonable prior for human or humanoid kinematics.

An alternative powerful deep-learning approach, the autoencoder (AE) (Vincent et al., 2008), is a neural network method to learn features from unlabeled data. AEs consist of encoder networks and decoder networks, which reconstruct the input data in the output layer. The number of neurons in the hidden layer is less than that of the input layer, which pushes the data through a bottleneck and forces it to extract the most relevant features. AEs allow for non-linear feature extraction. Denoising autoencoders (Vincent et al., 2008) are based on the basic AE with corruption of the input during the training process. This model is able to robustly reconstruct undestroyed input data from a partially corrupted one. Sparse autoencoders are imposed by adding an extra term of penalty on the hidden units to the cost function during training (Ng, 2011), or manually setting most of the smallest hidden units to zero (Makhzani and Frey, 2013). The latent space can therefore sparsely represent the inputs. Lange and Riedmiller (2010); Mattner et al. (2012) applied AE to feature generation which are then used for reinforcement learning for movements of visual data.

The Variational Autoencoder (VAE) (Kingma and Welling, 2014; Rezende et al., 2014) has obtained competitive results. It learns a low-dimensional latent space of high-dimensional data using statistical inference. Different from standard AEs as described above, VAEs have strong assumptions regarding to the distribution of the probabilistic latent variables.

The Autoencoders have no internal state, and therefore cannot represent temporal dependencies in the input data. There are several ways of dealing with this; one possibility is extending the VAE to be a recurrent neural network, e.g., Stochastic Recurrent Network (STORN) (Bayer and Osendorfer, 2014) or Variational Recurrent Neural Network (VRNN) (Chung et al., 2015). While having their merit in, e.g., anomaly detection (Sölch et al., 2016), their prediction capabilities are not as good as expected (Theis et al., 2016). Furthermore, it is not clear how a control signal can be included. A different, very promising approach, Deep Variational Bayes Filtering (DVBF) (Karl et al., 2017), is obtained by rolling a VAE out in time. However, DVBF requires extra input of control signal which is difficult to be captured for human movements. In addition, the capability of new movement generalization of DVBF is still not explored. Therefore, a further development of DVBF is studied in this thesis to solve these issues.

1.2 APPROACHES AND CONTRIBUTIONS

Machine learning, as described in the previous section, has been successfully applied to movement modeling. However, more issues need to be addressed to improve the algorithms and applications. To this purpose, two parts are proposed as follows.

The first part is "static movement" representation. Static models are presented that allow measuring finger contact force from fingernail images. Specifically, steady cameras are used to observe the nails of the fingers while in contact with an object, and the relationship between nail coloration as well as the deformation of the surrounding skin and force vector are learned. With such low frequency and smooth data, static methods are sufficient to represent the movements. Compared to previous studies, our methods are more robust for various environments, and augment the force estimation ability of image-based method to a larger force range.

The second part is dynamic movement prediction. To efficiently present the high dimensional sequence data in a low dimensional latent space, we investigate unsupervised learning with dynamics for movements. In particular, reinforcement learning or dynamic movement primitive are embedded into a time-dependent AE or

VAE. As a result, the algorithms can efficiently predict movements in latent space for high-dimensional human or robot movements.

The major contributions of this thesis are:

- the capability of the human finger movement modeling is dramatically improved;
- *time-dependent AEs/VAEs* are designed, which significantly improve the latent representation of time series data;
- a powerful generalization capability of human/robot movement modeling in the latent space is developed.

More details of the approaches are presented in each chapter.

1.3 FIRST PUBLISHED APPEARANCES AND OUTLINE OF THE THESIS

This thesis incorporates previous publications and is organized as follows.

In Part 2, human "static movement" representation will be shown, specifically, grip force detection from camera images. This chapter takes the work from

- Nutan Chen, Göran Westling, Benoni B. Edin, and Patrick van der Smagt. Estimating fingertip forces, torques, and local curvatures from fingernail images. *submitted*

In Part 3, dynamic prediction of human/robot movements in latent space will be presented. This chapter takes the work from

- [1] Herke van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2016b
- Nutan Chen, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Efficient movement representation by embedding dynamic movement primitives in deep autoencoders. In *International Conference on Humanoid Robots (HUMANOIDS)*, pages 434–440, 2015
- Nutan Chen, Maximilian Karl, and Patrick van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *International Conference on Humanoid Robots (HUMANOIDS)*, 2016b

In Part 4, the research will be concluded.

The following first-authored and co-authored publications are not used in this thesis:

- Nutan Chen, Alexej Klushyn, Alexandros Paraschos, Djalel Benbouzid, and Patrick van der Smagt. Active learning based on data uncertainty and model sensitivity. *submitted*, 2018b
- Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018a
- Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models based on learned skills. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Robot Learning*, 2017a
- Nutan Chen, Sebastian Urban, Justin Bayer, and Patrick van der Smagt. Measuring fingertip forces from camera images for random finger poses. In *International Conference on Intelligent Robots and Systems (IROS)*, 2015
- Nutan Chen, Sebastian Urban, Christian Osendorfer, Justin Bayer, and Patrick van der Smagt. Estimating finger grip force from an image of the hand using convolutional neural networks and Gaussian processes. In *International Conference on Robotics and Automation (ICRA)*, 2014
- Rachel Hornung, Nutan Chen, and Patrick van der Smagt. Multimodal motion modeling. *Handbook of Multimodal-Multisensor Interfaces (in print)*
- Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. In *International Conference on Learning Representations (ICLR)*, 2014
- Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models based on learned skills. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Workshop on Acting and Interacting in the Real World: Challenges in Robot Learning*, 2017b
- Nutan Chen, Maximilian Karl, and Patrick van der Smagt. Dynamic movement primitives in latent space of time-dependent

1 Since it is the work that we cooperated with IAS institute, TU Darmstadt, it is only partially shown in this thesis.

variational autoencoders. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Neurorobotics*, 2016a

- Herke van Hoof, Nutan Chen, Maximilian Karl, Tucker Hermans, Gerhard Neumann, Patrick van der Smagt, and Jan Peters. Learning robot in-hand manipulation with tactile features. *Robotics: Science and Systems (RSS) Workshop on Bootstrapping Manipulation Skills*, 2016a

Part II

BACKGROUND

2

FUNDAMENTALS

This chapter presents a review of some machine learning and robotics principles from an algorithmic point of view, which are basic necessities for the following chapters.

2.1 MACHINE LEARNING

2.1.1 *Neural Networks*

A neural network (Bishop, 2006) is a model based on linear combinations of parameterized nonlinear basis functions. Given a training dataset $\mathbf{x} := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ and corresponding target dataset $\mathbf{y}$, we write a neural network

$$\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1}$$

where $\theta = \{\mathbf{W}, \mathbf{b}\}$ are the parameters, and σ is the activation function or transfer function which works on each element of the vector. Modifying the architecture of a NN by taking the output of one layer as the input of another layer, the model can be extended to multiple layers. Fig. 1 shows a neural network with one hidden layer.

The parameters of the network are obtained by minimizing the loss function. It is possible to define an arbitrary loss function. For instance, with a regression problem, we can write a loss function

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i)\|. \tag{2}$$

The parameters of NNs are updated using the error-correction-learning rule through back-propagation (BP). BP starts at the output layer and recursively computes the local error for each neuron towards the first hidden layer. A gradient descent algorithm is a classical error-correction-learning,

$$\begin{aligned} \mathbf{w}(t+1) &= \mathbf{w}(t) - \eta g(\mathbf{w}(n)), \\ g(\mathbf{w}(n)) &= \nabla \mathcal{L}(\mathbf{w}(n)), \end{aligned} \tag{3}$$

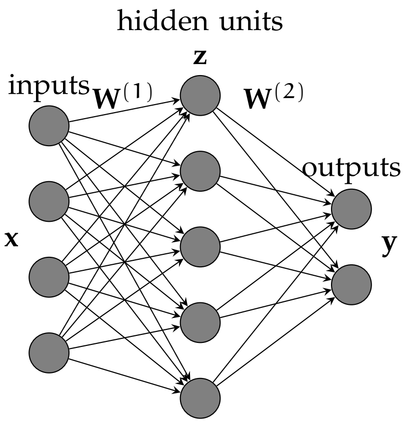


Figure 1: A feedforward neural network with a hidden layer and an output layer. $\mathbf{W}^{(i)}$ is the parameters of layer i. $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ denote the input, hidden and output variables respectively. The nodes represent the variables, and the links represent the weight and bias parameters.

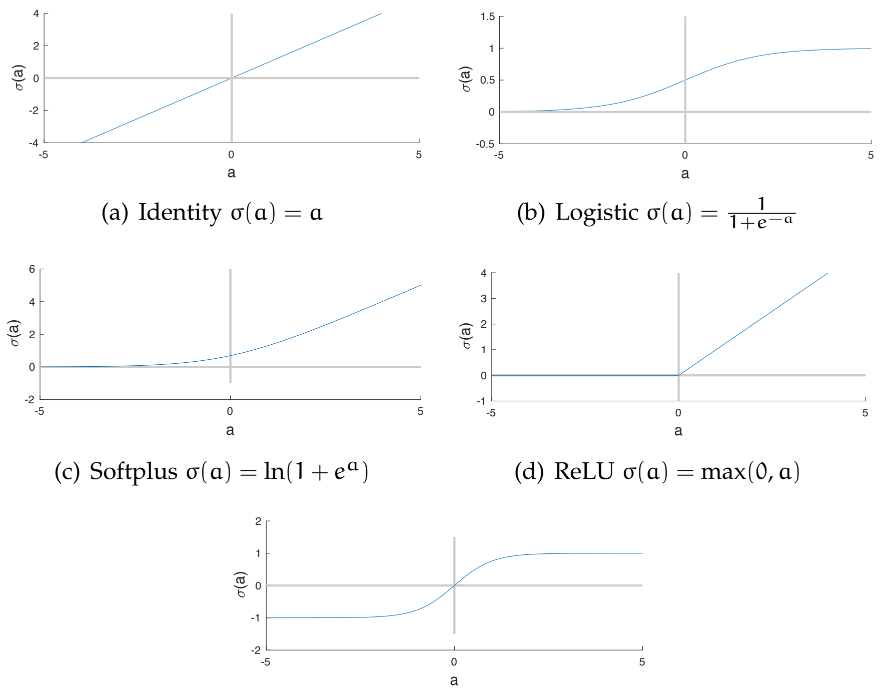
where t is the training iteration, and $\eta > 0$ denotes the step size or training rate. If the step size is too large the training probably oscillates or diverges, while too small step size might lead to a long convergence time. The direction of $-\nabla\mathcal{L}(\mathbf{w}(t))$ is the steepest descent direction. The chain rule is used for computing the derivatives of multiple layers.

Activation function. The activation function enables networks to be non-linear. Fig. 2 illustrates several commonly used activation functions. An identity function is a linear function. Softplus function is a smooth approximation of the ReLU. Compared to softplus function or other similar activation functions, Rectified linear units (ReLU) (Nair and Hinton, 2010) is faster and more effective for training deep networks with large datasets. A logistic function is similar to a tanh function with different ranges of the outputs.

Overfitting. A complex neural network with a large amount of weights and biases probably causes overfitting. The accuracy of predictive performance is reduced when overfitting occurs. Some approaches can reduce the effects of overfitting, e.g., data augmentation, noise adding, bagging, boosting, reduction of the feature number, early stopping, regularization, dropout and Bayesian approach with proper prior. We present regularization and dropout in this section, which are used in the following chapters.

Regularization of the parameters in error function is able to prevent overfitting

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i)\| + \lambda \mathbf{r}(\theta), \tag{4}$$



(a) Identity $\sigma(a) = a$

(b) Logistic $\sigma(a) = \frac{1}{1+e^{-a}}$

(c) Softplus $\sigma(a) = \ln(1 + e^{a})$

(d) ReLU $\sigma(a) = \max(0, a)$

(e) $\text{Tanh}(a) \equiv \frac{2}{1+e^{-2a}} - 1$

Figure 2: Activation funcions.

where λ denotes the scale of the regularization and $\mathbf{r}$ is the regularization term. The regularization can be a smoothness term with $\mathbf{r}(\theta) = \sum_{j=1}^{M} \|\theta_j\|_2$ or a sparsity term of the parameters with $\mathbf{r}(\theta) = \sum_{j=1}^{M} \|\theta_j\|_1$, where M is the parameter size.

Random Dropout (Srivastava et al., 2014) of the neurons during training process in the layers of neural networks prevent complex co-adaptations and reduce overfitting. For every neuron of the dropout layer, each of the input neuron of this layer is picked with a probability of p, and its value is set to 0, while the rest are unchanged during the training process. Thus, the chosen neurons are dropped. During testing process, all neurons are present without corruption and the weights of the dropout layers are scaled by multiplying $1 - p$. The weights on other layers and biases are unchanged.

Fast dropout (Wang and Manning, 2013) is more efficient to train a model by sampling from a Gaussian approximation (see Section 2.1.2.1).

2.1.2 *Probabilistic Neural Networks*

Deterministic neural networks might have difficulties at representing uncertainties. The uncertainties include the biased distribution between the training and test data, noisy data, model parameters and structure uncertainties (Ghahramani, 2015; Gal, 2016). The former two are the data uncertainties and the latter two are the model uncertainties. Probabilistic algorithms can calibrate the model and tackle model uncertainties. Additionally, a probabilistic model is able to provide confidence bounds for evaluating data uncertainties, particularly, whether a model is certain about its output; therefore, it can be used for data analysis and decision making. The probabilistic models are not restricted to neural networks, but are also applied to methods such as Gaussian process (see Section 2.1.6).

The Bayesian theory is the foundation of probabilistic models. In neural networks, a Bayesian inference estimates the model uncertainty through the distributions over the weights (Buntine and Weigend, 1991; MacKay, 1992; Hinton and Van Camp, 1993; Williams, 1997). Before having seen the data, we have the prior probability $p(\theta)$, where θ represents the parameters. The likelihood, $p(\mathcal{D}|\theta)$, is the probability of the data $\mathcal{D}$ given θ. Based on Bayes' rule, we obtain the posterior probability of θ given the data,

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{5}$$

We apply neural networks to this process; therefore, given the training data, the probability distributions of the weights are obtained. Subsequently, we estimate the confidence bounds through the prediction,

$$p(\mathbf{y}_\star|\mathbf{x}_\star, \mathcal{D}) = \int p(\mathbf{y}_\star|\mathbf{x}_\star, \theta)p(\theta|\mathcal{D})d\theta, \tag{6}$$

where the parameters are marginalized out. $\mathbf{x}_\star$ and $\mathbf{y}_\star$ are the test input and output respectively. With this approach, the regularization and model selection/comparison are autonomously performed. Further developments of this technique include variational inference (see Section 2.1.5.3), and countless recent models (Graves, 2011; Blundell et al., 2015; Hernández-Lobato and Adams, 2015; Depeweg et al., 2016).

Probabilistic neural networks are used for a large number of applications. The confidence estimation is useful for efficient data collection such as active learning and reinforcement learning. For instance, in active learning, the algorithm is able to query the information source by the confidence values.

We take fast dropout which is used in Chapter 3 as an example, since we do not have space to discuss all the models.

2.1.2.1 *Fast Dropout*

In a fast dropout (Wang and Manning, 2013), $z_i \sim \text{Bernoulli}(p_i)$ is sampled to determine whether the input x_i is dropped out, where p_i is the rate of not dropping out. The output y is derived by

$$a = \mathbf{w}^T \mathbf{D}_z \mathbf{x} \tag{7}$$

$$y = \sigma(a) \tag{8}$$

where $\mathbf{w}$ is a weight vector and $D_z = \text{diag}(z) \in \mathbb{R}^{m \times m}$. m is the data dimension.

The input of the output layer takes a random variable for every hidden unit. Under fast-dropout training, we can assume its input as a Gaussian distribution $\mathbf{X} \sim \mathcal{N}(x|\mu, s^2)$. For any hidden unit, the mean and variance of output y are ν and τ^2. Using sigmoid activation function σ, we have:

$$\nu = \int_{-\infty}^{\infty} \sigma(x)\mathcal{N}(x|\mu, s^2)dx \approx \sigma\left(\frac{\mu}{\sqrt{1+\pi s^2/8}}\right), \tag{9}$$

$$\tau^2 = \underset{X \sim \mathcal{N}(\mu, s^2)}{\text{Var}} [\sigma(X)] = E[\sigma(X)^2] - E[\sigma(X)]^2. \tag{10}$$

We draw samples of a Gaussian approximation for $a = \mathbf{w}^T \mathbf{D}_z \mathbf{x}$. The mean and variance of a can be obtained. We assume that $\mathbf{x}$ components are independent; therefore, the central limit theorem of Lyapunov condition is satisfied with $m \to \infty$. Consequently, a is approximately Gaussian.

The neural networks with fast dropout can be trained to update $\mathbf{w}$ through back-propagation.

2.1.3 *Convolutional Neural Networks*

A convolutional neural network (CNN) is an architecture for regression and classification that is relatively robust to shifts, scales and distortions of the input data and, especially, can be efficiently trained on large data sets (LeCun et al., 1998). CNNs are widely applied to images and videos (LeCun et al., 2010). A CNN is typically designed as multiple stages of convolutions and max-pooling and the top layers are usually ordinary multi-layer perceptrons. Max-pooling is a non-linear down-sampling method that decreases computational complexity. Fig. 3 shows the architecture of LeNet by LeCun et al. (1998), which is one of the pioneering CNNs. A first convolutional layer followed by a first max-pooling layer, another convolutional layer followed by a second max-pooling layer, and finally, two fully connected perceptron layers.

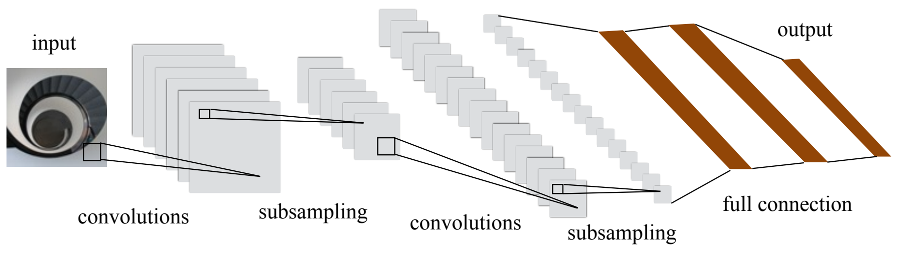


Figure 3: Architecture of convolutional neural networks. The brown quadrangles represent neurons.

In LeNet, The feature map h of the 2D input images is computed with convolution as

$$h(m, n) = \sum_{u=0}^{l_x} \sum_{v=0}^{l_y} w(u, v)\, g(u + m, v + n) + b, \tag{11}$$

where g is the input map, w the kernel weights, b the bias, (l_x, l_y) is the size of the filter, and (m, n) is the pixel position on the feature map.

The max-pooling activation is computed as

$$p(m, n) = \max_{i=1}^{r_1} \left(\max_{j=1}^{r_2} h(r_1 m + i, r_2 n + j) \right), \tag{12}$$

where (r_1, r_2) is the pooling size and p is the feature map in the max-pooling layer. The max-pooling layers take the output of convolutional layers as input, and reduce the resolution of the input.

The penultimate part is a fully-connected multi-level perceptron (MLP) with hidden units of which the final layer is linear with outputs.

We use the chain rule to backpropagate error gradients back into the network to minimize the loss function. For regression problem, we write the square error loss

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y} - \hat{\mathbf{y}}\|_2, \tag{13}$$

where $\mathbf{y}$ is the ground truth of the outputs. The model can be applied to classification problem by changing the loss function.

CNNs have successfully performed on many datasets. For instance, Goodfellow et al. (2014) beat the benchmarks of Street View House Numbers (SVHN) dataset in 2014; Krizhevsky et al. (2012) beat the benchmarks of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012; Simonyan and Zisserman (2014) beat the benchmarks of ILSVRC in 2014; He et al. (2016) beat the benchmarks of ILSVRC 2015.

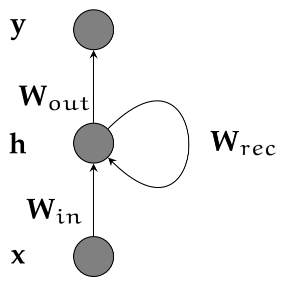


Figure 4: A recurrent neural network.

2.1.4 *Recurrent Neural Networks*

A recurrent neural network (RNN) is a neural network using the internal memory to capture and process arbitrary sequences of inputs. For every time step of a sequence, the output is predicted using the current input and previous inputs or latent units.

Given a sequence of input data $\mathbf{x}_t \in \mathbb{R}^l (t = 1, 2, \dots T)$ and corresponding targets $\mathbf{y}_t \in \mathbb{R}^m$ $(t = 1, 2, \dots T)$. $\hat{\mathbf{y}}_t \in \mathbb{R}^m$ $(t = 1, 2, \dots T)$ is the output of RNN which has hidden layers $\mathbf{h}_t \in \mathbb{R}^n$ $(t = 0, 1, \dots T)$. f_h and f_y are transfer functions, the constant T is the sequence length and l, m, n are the input, output and hidden dimensions at every time step (see Fig.4).

Taking the Elman network (Elman, 1990) with one hidden layer as an example, we have

$$\begin{aligned} \mathbf{h}_t &= f_h(\mathbf{x}_t\mathbf{W}_{\text{in}} + \mathbf{h}_{t-1}\mathbf{W}_{\text{rec}} + \mathbf{b}_h), \\ \hat{\mathbf{y}}_t &= f_y(\mathbf{h}_t\mathbf{W}_{\text{out}} + \mathbf{b}_y) \end{aligned} \tag{14}$$

where $\theta = \{\mathbf{W}_{\text{in}}, \mathbf{W}_{\text{out}}, \mathbf{W}_{\text{rec}}, \mathbf{b}_h, \mathbf{b}_y\}$ are the parameters. It can be extended to multiple hidden layer neural networks. There are other RNNs with different archtectures such as Jordan network (Jordan, 1997).

The gradients are calculated by Backpropagation through Time (BPTT) (Werbos, 1990). The parameters are obtained by minimizing the loss function

$$\mathcal{L}(\theta) = \sum_i \|\hat{\mathbf{y}}^{(i)} - \mathbf{y}^{(i)}\| \tag{15}$$

RNNs have difficulties learning long-sequence dependencies, because of the vanishing gradient problem (Hochreiter, 1991) of BPTT. There are some approaches to tackle the problem. Proper initialization or regularization of the parameters can reduce vanishing gradients. The derivative of the ReLU activation function is a constant, so that the ReLU has less vanishing gradient compared to tanh or sigmoid. Additionally, other architectures such as Long Short-Term Memory (LSTM)

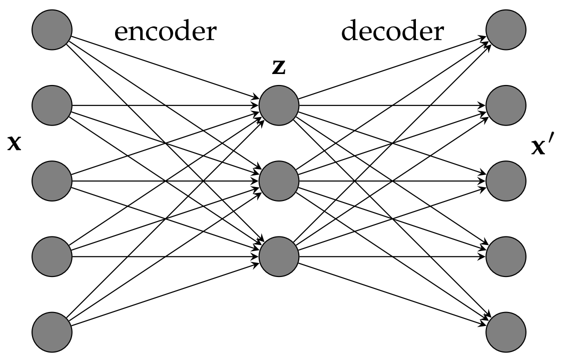


Figure 5: An autoencoder. The nodes represent the variables, and the links represent the weight and bias parameters.

(Hochreiter and Schmidhuber, 1997) and Gated Reccurent Unit (GRU) (Chung et al., 2014) can solve this problem.

2.1.5 *Autoencoders*

The idea of autoencoders is coarsely comparable to principal component analysis (PCA). Yet, the representational capabilities are much greater since autoencoders allow for non-linear feature extraction. Also, autoencoders are not limited to normally distributed data and do not assume perpendicularity of the "principal components".

An autoencoder or a Diabolo network (Rumelhart et al., 1988; Bourlard and Kamp, 1988; Bengio, 2009), which consists of encoder networks and decoder networks, is a neural network method to learn features from unlabeleld data. The encoder network takes $\mathbf{x} \in \mathbb{R}^d$ as input vector, and maps the input to a latent representation with multiple hidden layers. Every hidden layer computes a mapping $\mathbf{z} = \mathbf{h}_\theta(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, where $\theta = \{\mathbf{W}, \mathbf{b}\}$ are parameters, σ is the activation function, $\mathbf{z} \in \mathbb{R}^{d'}$ is the feature representation, and d' is the number of hidden neurons in that layer.

Subsequently, the feature representation is reconstructed back to a observation vector $\mathbf{x}' \in \mathbb{R}^d$ through the decoder networks with the same structure. Every hidden layer of decoder is $\mathbf{x}' = g_{\theta'}(\mathbf{y}) = \sigma(\mathbf{W}'\mathbf{y} + \mathbf{b}')$, where $\theta' = \{\mathbf{W}', \mathbf{b}'\}$, $\mathbf{b}'$ is a bias vector in the decoder layers. The weight matrix can be restricted to be equal to the transpose of the encoder weights $\mathbf{W}' = \mathbf{W}^T$ (Bengio, 2009). To seek the parameters θ and θ', the problem becomes:

$$\begin{aligned}\theta^\star, \theta'^\star &= \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\mathbf{x}^{(i)}, \mathbf{x}'^{(i)}) \\ &= \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\Big[\mathbf{x}^{(i)}, g_{\theta'}\big(h_\theta(\mathbf{x}^{(i)})\big)\Big], \end{aligned} \tag{16}$$

where n is the number of a training set, $\mathcal{L}$ is a loss function and, the squared error $\mathcal{L}(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^2$ is used to represent the reconstruction error in the observation data.

The number of neurons in the hidden layer is less than that of the input layer, which pushes the data through a bottleneck and forces it to extract the most relevant features. It also makes the trivial solution of the identity function at each neuron impossible.

2.1.5.1 *Denoising Autoencoder*

Denoising autoencoders (DAs, Vincent et al. (2008)) are based on the basic autoencoder with corruption of the input during the training process. The model is able to robustly reconstruct undestroyed input data from a partially corrupted one. We partially destroy the initial $\mathbf{x}$ to generate $\tilde{\mathbf{x}}$ as the input instead, and the DA reconstructs $\mathbf{x}$. For every input frame $\mathbf{x}_i$, each of the input neuron is dropped with a probability of p, and we have

$$\theta^\star, \theta'^\star = \arg\min_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left[\mathbf{x}^{(i)}, g_{\theta'}\left(h_\theta(\tilde{\mathbf{x}}^{(i)})\right)\right]. \tag{17}$$

The layers are fully connected except the removed neurons. During testing process, we take $\mathbf{x}$ as the input, and the input layer to the first hidden layer becomes $\mathbf{h}_\theta(\mathbf{x}) = \sigma[(1-p)\mathbf{W}\mathbf{x} + \mathbf{b}]$.

The basic principles of denoising and dropout are the same, which avoid over-fitting. Noise is applied to the input layer for denosing, while applied to all layers except the output layer for dropout.

2.1.5.2 *Sparse Autoencoder*

The autoencoder is able to discover meaningful structures in the latent space with the constraints of the sparsity (Ng, 2011; Nair and Hinton, 2009). For instance, sparsity can deactivate some hidden neurons for distinguishing the features of the input.

Sparsity encourages hidden units to be active only rarely by adding an extra term of penalty to the cost. Nair and Hinton (2009) presented a l_k norm penalty

$$\mathcal{L}_{sparse}(\mathbf{z}) = \frac{1}{n}\eta \sum_{i=1}^{n} \|\mathbf{z}^{(i)}\|_k, \tag{18}$$

where η is a penalty parameter, and the l_k norm with $k \geqslant 1$ is defined as $\|\mathbf{z}\|_k \triangleq (\sum_{i=1}^{d'} |z_i|^k)^{1/k}$. It results in a sparse solution for $\mathbf{z}$.

Alternative types of the penalty term also have reasonable results. For instance, Ng (2011) presented Kullback-Leibler (KL, see Section 2.1.5.3) divergence

$$\hat{\rho}_j = \frac{1}{n}\sum_{i=1}^{n} \mathbf{z}_j^{(i)}, \tag{19}$$

$$\mathcal{L}_{sparse}(\mathbf{z}) = \sum_{j}^{d'} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}, \tag{20}$$

where j is the index of the hidden neurons and ρ represents a sparsity parameter. A typical ρ is a small number close to zero. The average of each hidden neuron, $\hat{\rho}_j$, is enforced to ρ; therefore, most hidden units are close to zero.

2.1.5.3 *Variational autoencoder*

Variational inference. Variational inference (Bishop, 2006) is a method to approximate the intractable posterior distribution $p(\mathbf{z}|\mathbf{x})$ through a tractable approximate variational distribution $q_\phi(\mathbf{z})$. $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{z} \in \mathbb{R}^{d'}$ are the observed data and its corresponding latent representation, respectively. As the dissimilarity function, the Kullback-Leibler (KL) divergence between the approximate distribution q_ϕ and the target distribution p is minimized to obtain the variational parameter ϕ for the optimal approximate distribution q_ϕ. The marginal log-likelihood is written as

$$\begin{aligned}
\log p(\mathbf{x}) &= \log p(\mathbf{x},\mathbf{z}) - \log p(\mathbf{z}|\mathbf{x}) && (21)\\
&= [\log p(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z})] - \log\frac{p(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z})} && (22)\\
&= \int q_\phi(\mathbf{z})\log\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}d\mathbf{z} - \int q_\phi(\mathbf{z})\log\frac{p(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z})}d\mathbf{z} && (23)\\
&= \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})}\right] + \mathbb{KL}\big(q_\phi(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\big), && (24)
\end{aligned}$$

in which through the KL divergence definition we have:

$$\mathbb{KL}\big(q_\phi(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\big) = -\int q(\mathbf{z})\log\frac{p(\mathbf{z}|\mathbf{x})}{q(\mathbf{z})}d\mathbf{z}. \tag{25}$$

We compute the expectation of $\log p(\mathbf{x})$ when $\mathbf{z} \sim q(\mathbf{z})$ in (23). $\log p(\mathbf{x})$ is independent of q_ϕ, so that $\mathbb{E}_{q(\mathbf{z})} \log p(\mathbf{x}) = \log p(\mathbf{x})$. Additionally, because of the non-negative property of the KL and the independence, the first term of (24), the evidence lower bound (ELOB) $\mathcal{L}^{\text{bound}}(q)$, is maximized to minimize the KL divergence.

We have an alternative derivation,

$$\begin{aligned}\log p(\mathbf{x}) &= \log \int_{\mathbf{z}} p(\mathbf{x},\mathbf{z}) d\mathbf{z} && (26)\\ &= \log \int_{\mathbf{z}} p(\mathbf{x},\mathbf{z}) \frac{q_\phi(\mathbf{z})}{q_\phi(\mathbf{z})} d\mathbf{z} && (27)\\ &= \log \left(\mathbb{E}_{q_\phi(\mathbf{z})} \left[\frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})} \right] \right) && (28)\\ &\geqslant \mathbb{E}_{q_\phi(\mathbf{z})} \left[\log \frac{p(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z})} \right], && (29)\end{aligned}$$

where the derivation from (28) to (29) is based on Jensen's inequality.

Variational Autoencoder. The variational autoencoder (VAE, Kingma and Welling (2014); Rezende et al. (2014)) efficiently infers the unobserved latent variables of probabilistic generative models. The unobserved latent vectors $\mathbf{z}^{(i)}$ code the observed vectors $\mathbf{x}^{(i)}$ from the dataset. As prior distribution of the latent space variables, an isotropic Gaussian $p^\star(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ is used. For non-binary data, a standard choice for the decoder $p(\mathbf{x}|\mathbf{z})$ is a Gaussian, where

$$\begin{aligned}\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2)),\\ \boldsymbol{\mu} &= \mathbf{w}_2\mathbf{h} + \mathbf{b}_2,\\ \log \boldsymbol{\sigma}^2 &= \mathbf{w}_3\mathbf{h} + \mathbf{b}_3,\\ \mathbf{h} &= f(\mathbf{z}) = h(\mathbf{w}_1\mathbf{z} + \mathbf{b}_1), && (30)\end{aligned}$$

in which the parameters μ, σ are given by a multi-layer perceptron parametrized by $\mathbf{w}$ and $\mathbf{b}$ jointly represented by θ. h is the activation function. $\{\boldsymbol{\mu}^{\text{enc}}, \boldsymbol{\sigma}^{\text{enc}}\}$ and $\{\boldsymbol{\mu}^{\text{dec}}, \boldsymbol{\sigma}^{\text{dec}}\}$ represent $\{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$ for the decoder and encoder, respectively. f is a function of one layer neural network in (30), but can be extended to multiple layers.

We would like to find parameters θ that optimize the marginal likelihood $p_\theta(\mathbf{x}^{(i)})$. As this objective is intractable for (30), we re-write the marginal likelihood as

$$\begin{aligned}\log p_\theta(\mathbf{x}^{(i)}) &= \log \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta^\star(\mathbf{z}) d\mathbf{z}\\ &= \mathbb{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}). && (31)\end{aligned}$$

In this equation, a parametric approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to $p_\theta(\mathbf{z}|\mathbf{x})$ is used as this term relies on an intractable integral. The encoder $q_\phi(\mathbf{z}|\mathbf{x})$ has a similar structure as (30), but $\mathbf{z}$ and $\mathbf{x}$ are swapped and the weights and biases are in a different set of parameters ϕ. In (31), $\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$ is a lower bound on the marginal likelihood

$$\begin{aligned}\mathcal{L}(\theta, \phi; x^{(i)}) =& \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})]\\ &- \mathbb{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z})). && (32)\end{aligned}$$

The first term can be interpreted as a reconstruction cost, which is approximated by sampling from $q_\phi(\mathbf{z}|\mathbf{x})$. The KL-divergence term $\mathbb{KL}$ quantifies the loss of information when the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ is used instead of $p_\theta(\mathbf{z})$.

The lower bound is optimized by stochastic backpropagation. As the reconstruction term is estimated through sampling, we compute the gradient through the sampling process of $q_\phi(\mathbf{z}|\mathbf{x})$ with the reparametrization trick $\mathbf{z} = y(\phi, \epsilon)$, where y is a function of ϕ with noise ϵ, as in (Kingma and Welling, 2014).

2.1.6 *Gaussian Process*

Gaussian Process Regression. A Gaussian Process (GP) (Rasmussen and Williams, 2006) is a stochastic process given by its mean $m(\mathbf{x})$ and covariance $k(\mathbf{x}, \mathbf{x}')$,

$$m(\mathbf{x}) = \mathbf{E}[f(\mathbf{x})], \tag{33}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{E}\Big[\big(f(\mathbf{x}) - m(\mathbf{x})\big)\big(f(\mathbf{x}') - m(\mathbf{x}')\big)\Big]. \tag{34}$$

Assuming the GP has a zero mean function, the squared exponential covariance (SE) is derived as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\Big(-\frac{1}{2l^2}\|\mathbf{x} - \mathbf{x}'\|_2\Big). \tag{35}$$

The length-scale l and the signal variance σ_f^2 are the hyper-parameters. Points that have distances to each other smaller than l can be considered to have similar values.

The inputs of training points are $\mathbf{X} := (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)^T$. $\mathbf{x}_i$ is a 1D vector. In addition, $\mathbf{y} := (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N)^T$ is the corresponding target. In realistic situations, the function values are unknown; therefore, we the noisy observations is written as

$$\mathbf{y} = f(\mathbf{x}) + \epsilon, \text{ with } \epsilon \sim \mathcal{N}(0, \sigma_n^2), \tag{36}$$

where σ_n is the noise variance hyper-parameter. Thus, the joint distribution of the training target y and test output $\mathbf{f^*}$ under the prior is

$$\begin{bmatrix} y \\ \mathbf{f^*} \end{bmatrix} = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K}_y & \mathbf{K}^* \\ \mathbf{K}^{*T} & \mathbf{K}^{**} \end{bmatrix}\right) \tag{37}$$

and the target value $\mathbf{f}^*$ for $\mathbf{x}^*$ is distributed as

$$p(\mathbf{f}^*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}^*|\mathbf{E}[\mathbf{f}^*], \mathrm{Var}[\mathbf{f}^*]), \tag{38}$$

$$\mathbf{E}[\mathbf{f}^*] = \mathbf{K}^{*T}(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, \tag{39}$$

$$\mathrm{Var}[\mathbf{f}^*] = \mathbf{K}^{**} - \mathbf{K}^{*T}(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}^*, \tag{40}$$

where $K_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$, $K^*_i = k(\mathbf{x_i}, \mathbf{x}^*)$, $\mathbf{I}$ is the identity matrix.

We maximize the log likelihood function

$$\begin{aligned}\log p(\mathbf{y}|X) = & -\frac{1}{2}\mathbf{y}^T(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{y} \\ & -\frac{1}{2}\log\left|\mathbf{K}+\sigma_n^2\mathbf{I}\right| - \frac{n}{2}\log 2\pi,\end{aligned} \tag{41}$$

and consequently obtain the optimal values for the hyper parameters, $\theta = l, \sigma_n, \sigma_f$, using the training set. $\frac{1}{2}\mathbf{y}^T(\mathbf{K}+\sigma_n^2\mathbf{I})^{-1}\mathbf{y}$ is a data fit term, $\frac{1}{2}\log\left|\mathbf{K}+\sigma_n^2\mathbf{I}\right|$ is a complexity penalty term and $\frac{n}{2}\log 2\pi$ is a constant term.

θ can be learned by gradient ascent techniques

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|\mathbf{X}) = \frac{1}{2}\mathbf{y}^T\mathbf{K}_y^{-1}\frac{\partial\mathbf{K}_y}{\partial\theta_j}\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\mathrm{tr}(\mathbf{K}_y^{-1}\frac{\partial\mathbf{K}_y}{\partial\theta_j}), \tag{42}$$

where $\mathbf{K}+\sigma_n^2\mathbf{I} \triangleq \mathbf{K}_y$.

Fully Independent Training Conditional Approximation. A critical issue with GP methods is that large computations are required: $O(N^3)$ for training and $O(N^2)$ for per testing case, where N is the number of training samples. To reduce the computational costs, one of the solutions is the fully independent training conditional approximation (FITC) (a.k.a. sparse pseudo-input GP (Snelson and Ghahramani, 2006)). The inducing points are a small amount of inputs M that summarize a large number of inputs N. By using inducing points we reduced the training and testing cost to $O(NM^2)$ and $O(M^2)$, respectively. FITC is implemented by randomly selecting a subset of the training data as the inducing points: $\overline{\mathbf{X}} := (\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \ldots, \overline{\mathbf{x}}_m)^T$. A more efficient likelihood approximation is then given by

$$\begin{aligned}p(\mathbf{y}|\mathbf{f}) &\simeq q(\mathbf{y}|\mathbf{u}) \\ &= \mathcal{N}(\mathbf{K}_{f,u}\mathbf{K}_{u,u}^{-1}u, \mathrm{diag}[\mathbf{K}_{f,f} - \mathbf{Q}_{f,f} + \sigma_{\mathrm{noise}}^2\mathbf{I}]),\end{aligned} \tag{43}$$

where $\mathbf{u}$ is the corresponding latent values of $\overline{\mathbf{X}}$, $\mathbf{f} = \{f_n\}_{n=1}^N$ are latent values based on $\mathbf{x}_n \in \mathbf{X}$, the covariance function $K_{f,f}$ is the Gram matrix of all pairs $(\mathbf{x}_i, \mathbf{x}_j)$, and diag[*] is a diagonal matrix, and $\mathbf{Q}_{f,f} \doteq \mathbf{K}_{f,u}$.

2.1.7 *Reinforcement learning*

Reinforcement learning (RL) (Sutton and Barto, 2012) maps situations to actions and consequently maximize reward signals, where actions

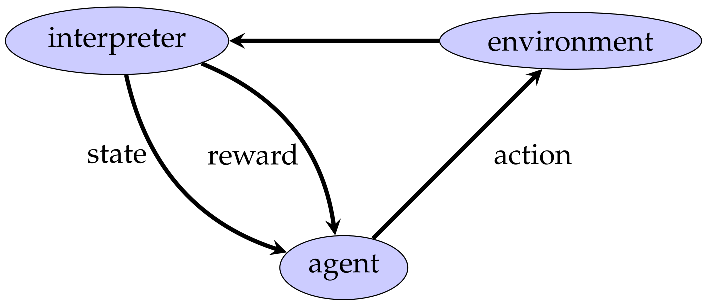


Figure 6: A basic reinforcement learning scenario.

affect both of the immediate and the future rewards. The learner is not specifically told which actions ought to take, but instead to try out which actions lead to the most cumulative rewards.

The learning problem of RL is formulated as a Markov Decision Process (MDP). The basic reinforcement learning system consists of an agent and environment states $\mathbf{s}$, an action $\mathbf{a}$ of the agent, a policy π from states to actions, a reward signal $\mathcal{R}$, and a value function $\mathbf{V}$. A model of the environment is an option, which categories RL into model-free and model-based methods. Model-free methods are simpler but require more samples. In the contrary, model-based methods are sample efficient, but it is required to estimate the model of the environment; therefore, they are sensitive to the accurate of the model.

At time t, the agent (e.g., a robot) obtains an observation including the reward $\mathcal{R}_t$. An action a_t is selected from the set of actions. With the interaction of the agent and the environment, the state s_{t+1} and the reward $\mathcal{R}_{t+1}$ of the next time step with the transition is detected (see Fig. 6). The action can be selected by exploration on uncharted territories and exploitation based on known areas.

Policy search. Policy search, a subfield of reinforcement learning, is a promising approach for solving MDP. It searches for parameters for a given policy parametrization. In this section we focus on a particular policy search method, non-parametric relative entropy policy search (NP-REPS) (van Hoof et al., 2015b,a), which is used in Chapter 4. See e.g., (Deisenroth et al., 2013) for more information on policy search.

Policy representation. Policy representations are used for policy search. Three policy representation types are commonly used including linear, radial basis functions (RBF) networks and dynamic movement primitives (DMPs) (Deisenroth et al., 2013). The former two are time independent representations $\pi_\theta(\mathbf{x})$, while a DMP is a time dependent representation $\pi_\theta(\mathbf{x}, t)$, where $\mathbf{x}$ is the state and θ is the policy parameter.

A linear or RBF policy is written as

$$\pi_\theta(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}), \tag{44}$$

where ϕ is a basis function vector for linear representation and $\phi_i(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x} - \mu_i)^T \mathbf{D}_i(\mathbf{x} - \mu_i))$ for RBF representations. $\mathbf{D}_i$ is a diagonal matrix. $\mathbf{w}$, μ and $\mathbf{D}$ are parameters.

The DMP is a spring-damper system and directly controls the acceleration of the robot joints

$$\pi_\theta(\mathbf{x}_t, t) = \ddot{\mathbf{y}} = \frac{1}{\tau}\alpha_z(\beta_z(\mathbf{g} - \mathbf{y}_t) - \dot{\mathbf{y}}_t) + \frac{1}{\tau}\mathbf{f}_t. \tag{45}$$

See the annotations of (45) and more details on DMPs in (53) of Section 2.2.1. The development of DMPs in Chapter 5 and 6 can be used for policy representations.

2.1.7.1 *Non-parametric REPS*

Traditional policy search might have the problems of optimization bias and overfitting, and consequently lead to oscillations and divergence. Relative entropy policy search (REPS) (Peters et al., 2010) is a method to bound the divergence between state-action distributions. Benefiting from the boundary, REPS requires relatively few samples and works well on real robots.

In a state $\mathbf{s}$, an action $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$ is selected by an agent from a policy π. It is assumed that $\mathbf{s} \in \mathcal{S} = \mathbb{R}^{D_s}$ and $\mathbf{a} \in \mathcal{A} = \mathbb{R}^{D_a}$. With a transition $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'} = p(\mathbf{s}'|\mathbf{a}, \mathbf{s})$, we have $\int_\mathcal{S}\int_\mathcal{A} \mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})d\mathbf{a}\,d\mathbf{s} = \mu_\pi(\mathbf{s}')$, where $\mu_\pi(\mathbf{s})$ is a state stationary distribution under policy. We maximize the average reward $J(\pi) = \int_\mathcal{S}\int_\mathcal{A} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}^{\mathbf{a}}_{\mathbf{s}}d\mathbf{a}\,d\mathbf{s}$ by choosing a policy, where $p_\pi(\mathbf{s}, \mathbf{a}) = \mu_\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})$ is the joint state-action distribution.

REPS can be written as an optimization problem

$$\max_{\pi,\mu_\pi} \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{R}^{\mathbf{a}}_{\mathbf{s}}\mathbf{dads}, \tag{46}$$

$$\text{s. t.} \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathbf{dads} = 1, \tag{47}$$

$$\forall s' \iint_{\mathcal{S}\times\mathcal{A}} \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}\mathbf{dads} = \mu_\pi(\mathbf{s}'), \tag{48}$$

$$\mathbb{KL}(\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})||q(\mathbf{s}, \mathbf{a})) \leqslant \epsilon. \tag{49}$$

(46) presents that the expected average reward is maximized by the joint state-action distribution. (47) is a constraint of $\pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s})$ being a probability distribution with the sum of the distribution being one. (48) constrains the the system dynamic $\mathcal{P}^{\mathbf{a}}_{\mathbf{ss}'}$. (49) is a bound on the KL divergence between the proposed state-action distribution and sampling distribution q that ensures smooth policy updates. The

Lagrangian optimization solves the optimization problem (van Hoof et al., 2015b)

$$p_\pi(\mathbf{s},\mathbf{a}) = \pi(\mathbf{a}|\mathbf{s})\mu_\pi(\mathbf{s}) \propto q(\mathbf{s},\mathbf{a}) \exp\left(\frac{\delta(\mathbf{s},\mathbf{a},V)}{\eta}\right) \tag{50}$$

where $V(\mathbf{s})$ and η represent Lagrangian multipliers. $V(\mathbf{s})$ is a value function; therefore, δ is the Bellman error

$$\delta(\mathbf{s},\mathbf{a},V) = \mathcal{R}_\mathbf{s}^\mathbf{a} + \mathbb{E}_{\mathbf{s}'}[V(\mathbf{s}')|\mathbf{s},\mathbf{a}] - V(\mathbf{s}). \tag{51}$$

It is assumed that $V = \sum_{\tilde{\mathbf{s}}\in\tilde{\mathcal{S}}} \alpha_{\tilde{\mathbf{s}}} k(\tilde{\mathbf{s}},\cdot)$, where $\tilde{\mathcal{S}}$ are sampled states, α are embedding strengths, and V is interpreted as a reproducing kernel Hilbert space (RKHS).

We estimate η and α by minimizing the dual function

$$g(\eta,\alpha) = \eta\epsilon + \eta\log\left(\sum_{i=1}^{n}\frac{1}{n}\exp\left(\frac{\delta(\mathbf{s}_i,\mathbf{a}_i,\alpha)}{\eta}\right)\right), \tag{52}$$

where $(\mathbf{s}_i,\mathbf{a}_i) \sim q(\mathbf{s},\mathbf{a})$.

We approximate δ, since the transition distribution is not given. van Hoof et al. (2015b) stated the approximation using kernel methods.

From previous policies, we have the reference distribution q. The initial samples are generated from an initial wide, uniformed exploration policy $\tilde{\pi}_0$. The policy converges to a local optimal policy, because the variance of polices is reduced for every iteration. We set λ and bandwidths of k_a through a cross-validation to minimize the discrepancy between true and predicted embedding strengths, and set bandwidths of k_s through a cross-validation to minimize the mean squared Bellman error.

$\tilde{\pi}(\mathbf{a}|\mathbf{s},\theta) = \theta^T\phi(s)$ is used as the policies where $\phi(s)$ are the features. We give a Gaussian prior over the parameters θ, and estimate a generalizing policy by conditioning on the sampled actions. Because the actions were drawn from previous distribution $q(\mathbf{a},\mathbf{s})$ instead of $p_\pi(\mathbf{a},\mathbf{s})$, importance weights w_i are used to compute a posterior

$$p(\theta|\mathbf{a}_1,\mathbf{s}_1,\dots,\mathbf{a}_n,\mathbf{s}_n) \propto p(\theta)\prod_{i=1}^{n}\tilde{\pi}(\mathbf{a}|\mathbf{s},,,)^{w_i},$$

where $(\mathbf{s}_i,\mathbf{a}_i) \sim q(\mathbf{a},\mathbf{s})$ and $w_i = p_\pi(\mathbf{s}_i,\mathbf{a}_i)/q(\mathbf{s}_i,\mathbf{a}_i) = \exp(\delta(\mathbf{s}_i,\mathbf{a}_i,V^*)/\eta^*)$. We obtain hyper-parameters by maximizing the marginal likelihood with cross-validation.

2.2 ROBOT LEARNING

Robot learning is a comprehensive field of machine learning and robotics. Pre-programmed robots obtain specific skills, but cannot adapt these skills for unknown environment. In contrast, the robot learning models learn the sequences of sensory inputs, temporarily memorize the data, and subsequently adapt for its realistic environments. The learning can be self-exploration or supervision from a teacher (e.g., imitation learning).

Robot learning is widely used for many robot tasks, for instance, grasping (Lenz et al., 2015), locomotion, human robot interaction (Maeda et al., 2016).

Dynamic movement primitives (DMPs) are one of the classical robot learning algorithms. Since we do not have sufficient space to discuss all algorithms, we focus on DMPs which are used in Chapter 5 and 6.

2.2.1 *Dynamic movement primitives*

Model. DMPs are generally trained from a demonstration of a trajectory in joint space of a robot or Cartesian space of a robot end effector, which can then subsequently be reproduced and adjusted (Ijspeert et al., 2013). A DMP is a point attractor using a second-order dynamic system:

$$\tau \ddot{\mathbf{y}} = \alpha_z(\beta_z(\mathbf{g}-\mathbf{y})-\dot{\mathbf{y}})+\mathbf{f}, \tag{53}$$

where τ is a time constant and α_z is a damping constant and $\alpha_z\beta_z$ is a spring constant. Typically, $\mathbf{y}$ is the position in joint or Cartesian space. The difference term $(\mathbf{g}-\mathbf{y})$ attracts the trajectory to the goal position $\mathbf{g}$, and we set the final frame of the demonstration as the goal.

Following Ijspeert et al. (2013), we can choose the basic forcing term $\mathbf{f}$ as a linear combination of basis functions Ψ_i:

$$\mathbf{f}(t) = \frac{\sum_{i=1}^{N} \Psi_i(t)\mathbf{w}_i}{\sum_{i=1}^{N} \Psi_i(t)}, \tag{54}$$

to create a linear time-variant dynamical system. To decouple this from the dynamics of the data, the time t is replaced using a first-order linear dynamic system

$$\tau \dot{s} = -\alpha_s s, \tag{55}$$

where α_s is a constant coefficient. Thus the state s converges monotonically to zero.

By differing the basis functions Ψ we can let DMPs be discrete or rhythmic dynamical systems. In the discrete systems, Ψ are written as (Ijspeert et al., 2013)

$$\Psi_i(s) = \exp\left[-\frac{1}{2\sigma_i^2}(s-c_i)^2\right] \tag{56}$$

where c_i and σ_i describe the width and centers of the basis functions, respectively.

In the rhythmic dynamical systems, we can write Ψ as

$$\Psi_i(s) = \exp(h_i(\cos(s - c_i) - 1)), \tag{57}$$

$$\text{with} \quad \tau\dot{s} = 1, \tag{58}$$

where $s \in [0, 2\pi]$, and h are parameters.

Learning. During training, with the demonstration $\mathbf{y}^{\text{demo}}$ and its computed derivatives, the target values of the forcing term results in,

$$\mathbf{f}^{\text{target}} = \tau^2\ddot{\mathbf{y}}^{\text{demo}} - \alpha_z\left(\beta_z(\mathbf{g} - \mathbf{y}^{\text{demo}}) - \tau\dot{\mathbf{y}}^{\text{demo}}\right). \tag{59}$$

Given $\{(\mathbf{f}_1, \mathbf{f}_1^{\text{target}}), \dots, (\mathbf{f}_n, \mathbf{f}_n^{\text{target}})\}$ of length n, locally weighted regression (LWR) (Atkeson et al., 1997; Ijspeert et al., 2013) is a method that is commonly used to obtain $\mathbf{w}_i$ for each kernel function Ψ_i. An LWR minimizes the locally weighted quadratic error criterion,

$$J_i = \sum_{t=1}^{n} \Psi_i(t)(f_t^{target} - \mathbf{w}_i\xi_t)^2, \tag{60}$$

where $\xi(t)$ is an extra term which can be multiplied to the right side of (54). For instance, we set $\xi_t = r$ for rhythmic system, where r represents an amplitude signal. The parameters are computed by

$$\mathbf{w}_i = \frac{\zeta^T \Gamma_i \mathbf{f}^{\text{target}}}{\zeta^T \Gamma_i \zeta}, \tag{61}$$

where

$$\zeta = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \dots \\ \xi_n \end{bmatrix} \quad \mathbf{f}^{target} = \begin{bmatrix} \mathbf{f}_1^{\text{target}} \\ \mathbf{f}_2^{\text{target}} \\ \dots \\ \mathbf{f}_n^{\text{target}} \end{bmatrix}$$
$$\Gamma_i = \begin{bmatrix} \Psi_i(1) & & & 0 \\ & \Psi_i(2) & & \\ & & \dots & \\ 0 & & & \Psi_i(n) \end{bmatrix}. \tag{62}$$

The training process is not limited to LWR. An optimizer can be used to find the parameters by minimizing the discrepancy between $\mathbf{f}$ and $\mathbf{f}^{\text{target}}$

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\text{argmin}} \sum_{t=1}^{n} \mathcal{L}\left[\mathbf{f}_t, \mathbf{f}_t^{\text{target}}\right], \tag{63}$$

where $\mathcal{L}$ is a loss function.

Part III

"STATIC MOVEMENT" REPRESENTATION

3 MEASURING FINGERTIP FORCES FROM CAMERA IMAGES

In this chapter, we consider a "static movement" representation problem, in which we are given a visual data set of human fingernails from a regression task. The goal is to learn a mapping from images to the corresponding labels using a training data set, which can then be used with novel images. Specifically, if the task is to measure the finger contact forces, we develop a model where we feed in an image represented by pixels and obtain an output of finger contact forces that the image represents.

Although the data in this chapter is made up of sequences of movements, it was recorded at a low frequency; therefore, the data can be considered as static. The frequency must be low with respect to the dynamics of underlying process (blood dispersion). Three time-independent algorithms are used, c.q., Gaussian process (GP), Convolutional Neural Networks (CNN) and Neural Networks with Fast Dropout (NN-FD), compared with a time-dependent algorithm, Recurrent Neural Networks with Fast Dropout (RNN-FD). The results further show that the performance of dynamic models yield to that of static models. Hence, it verifies the i.i.d. assumption.

3.1 INTRODUCTION

When studying the use of the human hand in grasping and manipulation (Johansson and Flanagan, 2009; Panarese and Edin, 2011), one of the most important aspects is how the finger force is controlled between themselves and the handled objects. These force vectors describe how we interact with an object, even more than the positions of the fingers themselves. Furthermore, it is these force vectors that are being accurately controlled by our neural system to optimize for grip stability and minimum intervention.

At the same time, *measuring* these forces is prohibitively difficult. Gloves with force sensors at the fingers destroy the “natural” feel and interaction, falsifying experimental data. Attaching measurements in-

Figure 7: Estimation of the finger force and torque from videos.

struments to objects is the obvious alternative, but leads to restrict experiments with only few objects, and is restricted to predefined positions on the object.

Following up on the seminal work by Mascaro *et al.* (see, e.g., (Sun et al., 2009; Grieve et al., 2009)), we use steady color cameras to observe the nails of the fingers while in contact with an object (Fig. 7), and learn the relationship between nail coloration and force vector.

Moving away from a constrained lab setting (e.g., a finger brace (Grieve et al., 2013)) with perfect conditions and comfortable restrictions, we extend this method to a more universal environment. In this study, with variables of contact surface, object weight and recording time, we find our approach performs accurately, and apply it to finger grasping analysis.

3.2 METHODS

3.2.1 *Setup*

Hardware setup. The recording setup consists of two stationary cameras and two force/torque sensors (see Fig. 3.8(a)). The cameras record the distal phalanges of the index finger and thumb, illuminated with diffuse light, as well as calibration markers on the respective nails. Video data is captured by two POINTGREY cameras at approx. 24 fps at a resolution of 1280×980 pixels. At the same time, the ATI Nano-17 six-axis force/torque sensors, located under the index finger and thumb, measure ground truth forces and torques at 100 Hz.

The whole setup is depicted in Fig. 8,9. In Fig. 3.8(a), camera 1 captures the images of marker 1, LED 1 and the index finger, while camera 2 captures the images of marker 2, LED 2 and the thumb. Marker 3 is on the F/T sensors. In Fig. 3.8(b), the contact surfaces can be easily changed. Niobium magnets guarantee the distance between the two contact surfaces to be 49.5 mm. The object that is held, shown in Fig. 3.8(c), is set up so that it adjusts to (1) change the weight of the object; (2) change the shape of the contact surfaces; (3) change the material and friction of the contact surfaces. In Fig. 9, the 11 curvature types of the surfaces can represent most kinds of commonly grasped objects. The surface on the sensor of the thumb is flat sandpaper.

Data synchronization. The image data and force/torque data are synchronized using LEDs. The LEDs are connected to the F/T sensor directly; therefore, LED signals and force data are synchronized beforehand. LEDs have signals of 1 and 0, representing on and off, respectively. To increase the accuracy, two LEDs are used for each camera, and only one LED is on or off. The sequences of LED signals collected from images and forces are scaled to the same frequency, and the cross-correlation of the sequences is calculated. Consequently, based on the cross-correlation, the force data is shifted to synchronize the image data.

3.2.2 *Image Alignment*

Image alignment was developed to reduce the variance caused by the finger orientation and location in the visual data. Before this, however, the mean shift algorithm (Comaniciu et al., 2000) was used to track the finger in the video stream. The fingernail and its surrounding skin were then segmented from the background based on the edge, and the fingernail geometry centers were shifted to the same position in the images. To segment the finger image robustly, we transferred the image from RGB to HSV, and changed hue and saturation to distinguish the finger from the background and then transferred the segmented image back to RGB. With this approach it was possible to segment a fingernail from a background color very close to skin color.

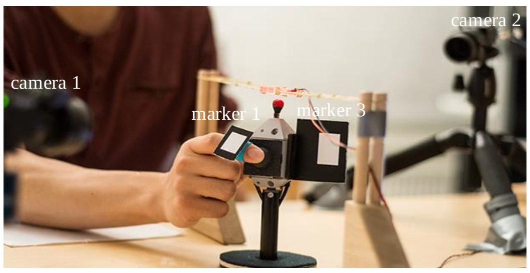
In fingertip RGB images, red and blue channels varied little with contact forces. Earlier we demonstrated alignment with convolutional neural networks (Chen et al., 2014) but the method we now propose achieves a high quality without reconstruction of a 3D finger model. In short, alignment transformations were generated using the blue channel of the image (Fig. 10), using the following method.

Given a reference image **R**, every subsequent image **J** was aligned using non-rigid image alignment (Myronenko and Song, 2010). In short, assume that the two images have the following intensity relationship:

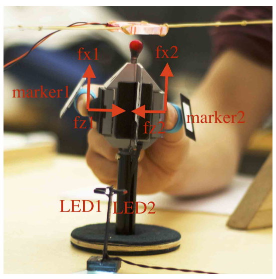
$$\mathbf{R} = \mathbf{J}(T) + \mathbf{v} + z \tag{64}$$

where $\mathbf{v}$ is an intensity correction field, $z \sim \mathcal{N}(0, \sigma^2)$ is zero-mean Gaussian noise, and T is the geometric transformation that registers J onto I. To estimate $\mathbf{v}$ and T, we minimized the objective function

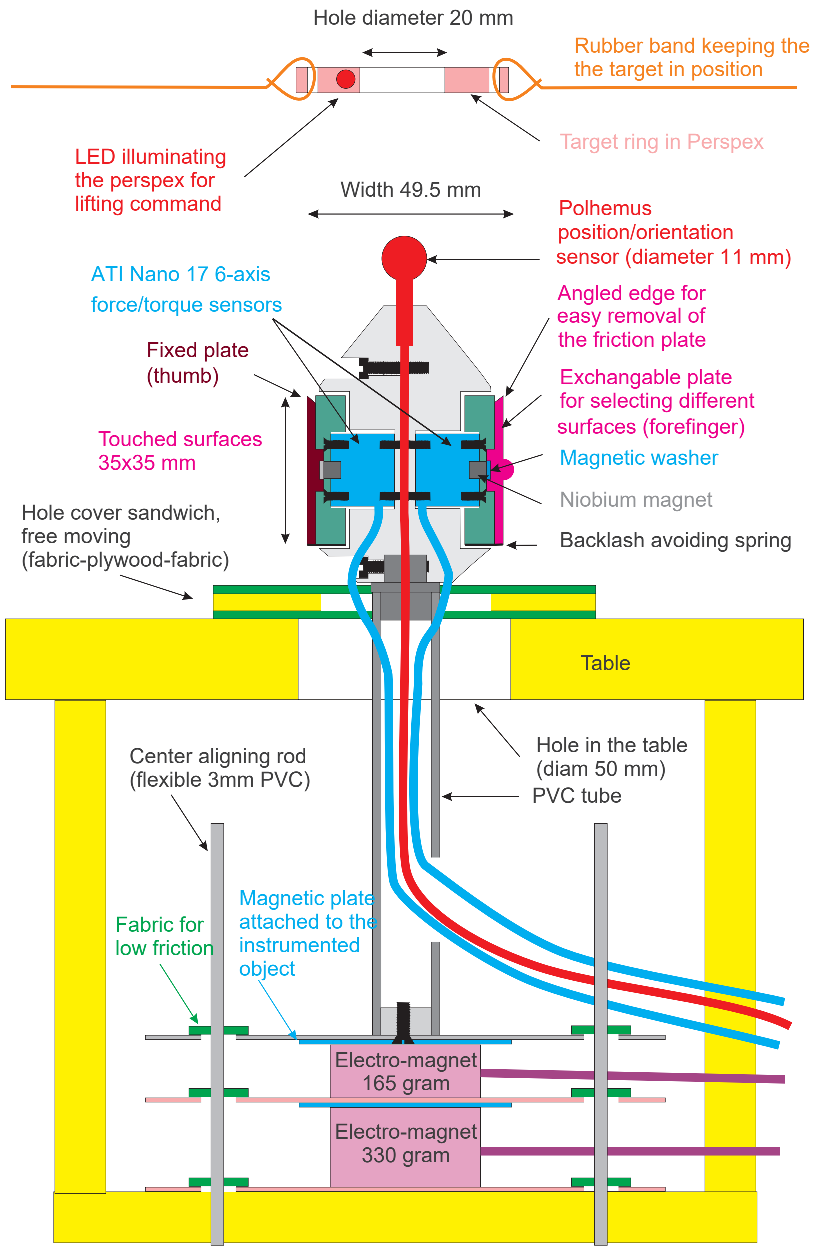
$$E(\mathbf{v}, T) = D(\mathbf{v}, T) + w\|\mathbf{P}\mathbf{v}\|^2 \tag{65}$$



(a) Recording system.



(b) Recording system (Cont'd)


Hole diameter 20 mm
Rubber band keeping the the target in position
Target ring in Perspex
LED illuminating the perspex for lifting command
Width 49.5 mm
Polhemus position/orientation sensor (diameter 11 mm)
ATI Nano 17 6-axis force/torque sensors
Angled edge for easy removal of the friction plate
Fixed plate (thumb)
Exchangable plate for selecting different surfaces (forefinger)
Touched surfaces 35x35 mm
Magnetic washer
Niobium magnet
Hole cover sandwich, free moving (fabric-plywood-fabric)
Backlash avoiding spring
Table
Center aligning rod (flexible 3mm PVC)
Hole in the table (diam 50 mm)
PVC tube
Fabric for low friction
Magnetic plate attached to the instrumented object
Electro-magnet 165 gram
Electro-magnet 330 gram

(c) The grasped object. The object above the table is visible for the subjects.

Figure 8: Setup. (b) F refers to the forces of the fingertips, where x and z represent lift and grip directions, respectively.

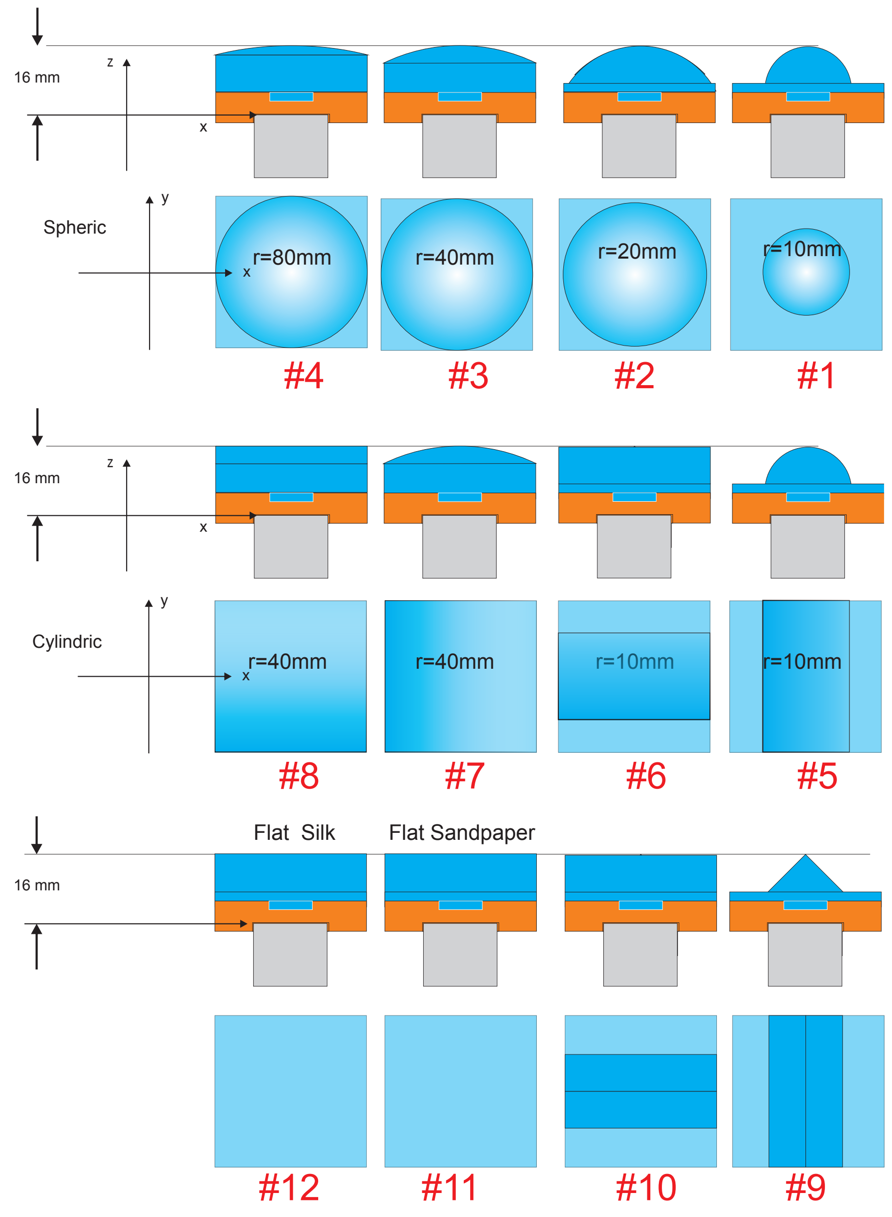


Figure 9: Contact surfaces for the index finger. Odd rows are the front view, and even rows are the top view. # 1 to # 11 are sandpaper and # 12 is silk. r represents the radius of the surface. # 5 to # 8 are flat in the x or y direction; therefore, r only represents the radius of the non-flat direction.

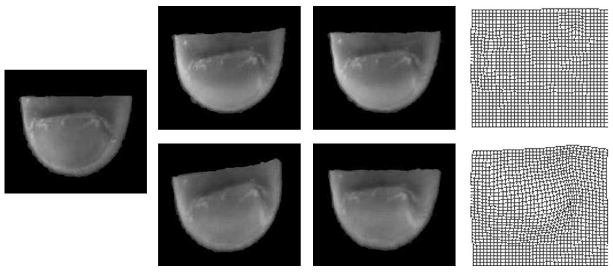


Figure 10: Image alignment. The columns from left to right are the reference image, the images before alignment, the aligned images and the mesh transformation. The finger images are in the green channel. The more deformation the image (row 2, column 2) has before alignment, the more mesh transformation (row 2, column 4) it is.

where

$$D(\mathbf{v}, T) = \|\mathbf{R} - \mathbf{J}(T) - \mathbf{v}\|^2 \tag{66}$$

is a measure of the similarity of $\mathbf{R}$ and $\mathbf{J}$, and $\|\mathbf{P}\mathbf{v}\|^2$ is a regularization term that penalizes some properties of $\mathbf{v}$, e.g., unsmoothness (the scalar w thus parameterizes the trade-off between the data fitness and regularization).

We model the transformation T by using the free-form deformation transformation with three hierarchical levels of B-spline control points. We update the transformation parameters via gradient-descent optimization.

3.2.3 *Predictors*

The fingernail and surrounding skin color distribution and deformation reflected the changes of contact force. Several variants of predictors were developed to construct the mappings from the finger images to the fingertip contact force/torque and contact surface curvatures. Below we describe in detail the four prediction methods: Gaussian Process (GP) regression and Convolutional Neural Networks (CNN), Neural Networks (NN) and Recurrent Neural Networks (RNN). GP and CNN performed best out of the four methods. While the GP produced slightly better results than the CNN, the CNN was considerably faster. The time-independent models (GP, CNN and NN) over-performed the time-dependent model (RNN), which coincides the i.i.d. assumption of the data.

Gaussian Process Regression. In our implementation of GP (see Section 2.1.6), the inputs $\mathbf{X} := (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)^T$ were aligned images reshaped to 1D vectors. The associated targets $\mathbf{y} := (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N)^T$ were the measured forces and torques and the curvature of the surface in contact with the fingertip. In a simple case without loss of

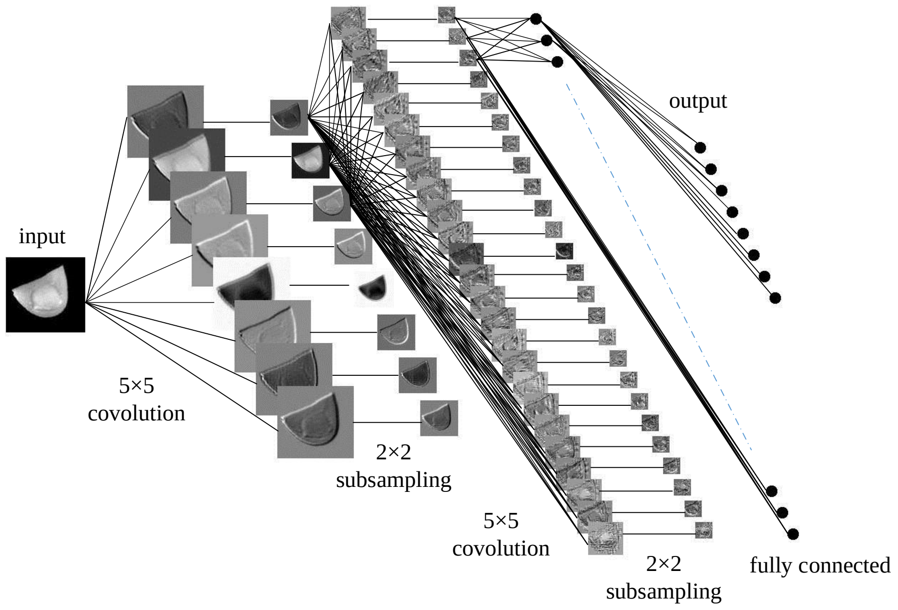


Figure 11: Architecture of convolutional neural networks for fingernail images.

generality, we assumed the GP has a zero mean function, and used the squared exponential (SE) covariance.

The GP predictor was able to train one model for all participants but then the data set increased, of course, to tens of thousands of images. To avoid the issue of large computational costs, we implemented the fully independent training conditional approximation (FITC).

Convolutional Neural Networks. Our implementation of CNNs contained six layers (Fig. 11): a first convolutional layer followed by a first max-pooling layer, another convolutional layer followed by a second max-pooling layer, and finally, two fully connected perceptron layers. Both convolutional layers featured a 5×5 sized filter, the first layer employed 8 kernels, the second layer 25 kernels. We used square error loss. See Section 2.1.3 for more information on CNNs.

Neural Networks with Fast Dropout. We have an alternative method, multilayer perceptron (MLP), mapping the image vectors to the targets. To reduce over-fitting during training process, fast dropout is used (see Section 2.1.2.1).

Recurrent Neural Networks with Fast Dropout. Since picking up and placing is a sequence movement, a recurrent neural network is employed to process the sequence data. To ensure that the i.i.d. assumption of the data set is correct, we compare the RNN with time-independent models.

RNN with fast dropout (RNN-FD) (Bayer et al., 2014) can be straightforwardly implemented as FD applied to NN. See Section 2.1.4 and 2.1.2.1 for information on RNNs and FDs.

3.2.4 *Calibration and postprocessing*

Torque calibration. The output of the F/T sensors consists of the forces and torques applied on each sensor, but these are not exactly equal to the fingertip forces and torques because of the shift of the contact position and the rotation of the finger with respect to the sensor. Therefore, we calibrate the data to correct the fingertip forces and torques. The forces and torques are calibrated separately. After training a force/torque predictor by the calibrated training and validation data, calibration of the testing data is only for the purpose of checking the accuracy of the results. When the system is used "in production", the calibration process is not required. Our approach to compute fingertip force and torque does not depend on knowing the axis of the fingertip nor on knowledge of the spatial orientation of the manipulated object.

Torque calibration is detected by means of finger contact positions. We have the mapping from the force f to the torque τ:

$$\tau_x = f_z y - f_y z,\ \tau_y = -f_z x + f_x z,\ \tau_z = -f_x y + f_y x, \tag{67}$$

where $\{x, y, z\}$ is the contact position with respect to the sensor coordinate.

Given the force and torque, there are infinitely many possibilities for the contact positions computed from (67). We calculate approximately the contact positions: first, using x and y to estimate z' based on the shape of the surfaces, ad second, estimating x' and y' by

$$x' = \frac{\tau_y - f_x z'}{-f_z} \text{ and } y' = \frac{\tau_x + f_y z'}{f_z}, \tag{68}$$

where $\{x', y', z'\}$ is the updated contact position.

The calibrated torques $\{\tau'_x, \tau'_y, \tau'_z\}$ are then updated:

$$\begin{aligned} \tau'_x &= \tau_x - (f_z y' - f_y z'), \\ \tau'_y &= \tau_y - (-f_z x' + f_x z'), \\ \tau'_z &= \tau_z - (-f_x y' + f_y x'). \end{aligned} \tag{69}$$

We repeat the above steps until the torques converge. z is initialized to 0.

Force calibration. Due to the finger rotation during grasping, the forces on the fingertip are rotated with respect to the sensor coordinate. The mapping from the finger image to the force is surjection for the rotation with respect to the x and y axes, so that we only calibrate f_x and f_y, which are rotated with respect to the z axis. We focus on force rotation calibration using two printed markers (see Fig. **??**) for each finger.

First of all, a rectangular marker is designed such that the pose (position and orientation) of the rectangle can be detected by a 2D camera using HALCON (MVTec). The four line segments of the rectangle border are detected, and the corresponding intersections are taken as corners of the rectangle. Once the internal camera parameters and the rectangle size in space and the detected corners are known, the rectangle pose in the camera space can be initially estimated. After that, a nonlinear optimization approach updates the final pose through minimization of the cost function, which is the geometrical distance between the detected borders and the back projection of the space rectangle onto the image. Basically, we can compute where the rectangle is in space.

Given the coordinate of a rectangle marker, the difference in the positions of two markers can be used to estimate the orientation θ of the finger with respect to the marker on the sensor. We select a frame as the reference frame and estimate the angle θ_r. Thus, f_x and f_y are calibrated by rotating $(\theta - \theta_r)$ with respect to the z axis.

Marker Calibration. In the case of recording the data at different times, the marker location on the finger can be changed. Since the marker on the finger is pasted without calibration, simple pre-processing is required for the marker orientation calibration. For this, we record a small data set and rotate the ground truth of f_x and f_y with respect to the z axis in $[-20, 20]$ degrees with one-degree steps. Subsequently, we choose the angle with the minimum error as the marker orientation of the testing data and match it to the same orientation in the training data set.

Postprocessing. To reduce the prediction noise, we smoothed the outputs from the predictors. Weighted linear least squares and a 2nd-degree polynomial model were used for local regression. In addition, it assigned lower weight to outliers in the regression to smooth the data robustly.

3.3 EXPERIMENTS AND RESULTS

3.3.1 *Data*

Kinematic and kinetic data were acquired with an experimental setup used previously (Luciw et al., 2014). In short, five healthy, right-handed participants (ages of 19~65; one female) were asked to repeatedly grasp and lift an instrumented object using their thumb and index finger. A light-emitting diode (LED) mounted into a translucent Perspex rectangle above the object signalled the start of a trial and when the object should be replaced on the support table.

The unpredictable weight of the object varied between 165, 330 and 660 g. The contact surfaces at the two digits could easily be changed between any of 12 surfaces all but one covered with sandpaper: 4 spherical convex surfaces (c=12.5, 25, 50 or 100 m^{-1}), 4 cylindrical convex surfaces (c_1 and c_2=0, 25 or 100), 2 surfaces with triangular surfaces, and 2 flat surfaces (one with sandpaper and one with silk). There were thus 36 weight $\times$ surface combinations. All participants repeated each combination in unpredictable orders 5 times, i.e., they all perform a total of 180 grasp-and-lift trials.

To measure the contact force/torque, the subjects pinched the object between their thumb and index finger. Each of the two contact surfaces were coupled to a six-axis force/torque sensor (ATI F/T 17). The series of trials divided into segments by detecting the forces for the starting point and releasing point. The light intensity at the nails was 120 lux at the index finger and 150 lux at the thumb.

As an extension of (Urban et al., 2013; Grieve et al., 2010), which restricted the norm force range to $[0, 10]$ N and the shear force to $[-2.5, 2.5]$ N, our experiments allowed the subjects to pick the object up with natural norm force up to 15.5 N. Specifically, the smaller friction (silk surface) requires the larger norm force.

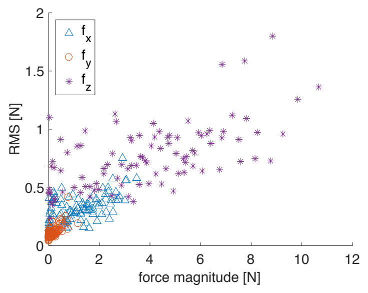
The testing samples are selected from time-continuous data. This increases the prediction difficulty compared to (Grieve et al., 2013), where the testing data was randomly selected from the whole data set.

We use root-mean-square error (RMSE) to evaluate the results. The unit of force and torque are N and Nmm, respectively, except when it is specifically noted.

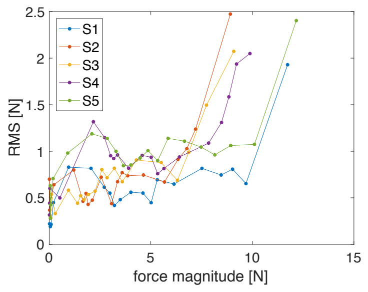
3.3.2 *Force/torque Prediction*

One of five series is selected as testing data set for each kind of surface and object weight for Gaussian Process. Therefore, approximately 20% is testing data and 80% is training data. In addition, to avoid over-fitting for CNN, NN-FD and RNN-FD, 25% of the GP training data is selected as validation data and 75% is training data while the testing data set is the same as that of GP.

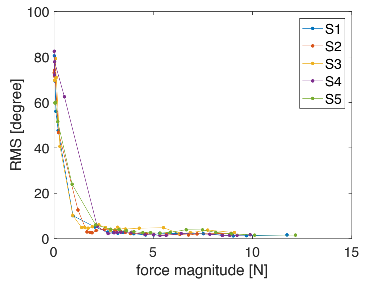
Although each subject has a slight different range of force and torque, the data set is approximately in the range of f_x $[-0.9, 4]$ N,



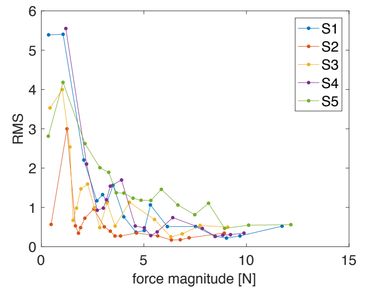
(a) RMS error in force components f_x, f_y, f_z over five subjects. The percentiles are computed separately in f_x, f_y and f_z.



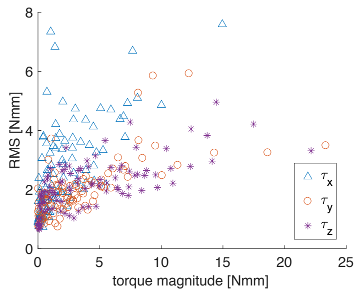
(b) RMS error in predicted force length magnitude.



(c) RMS error in angle between predicted and surface normal (f_z). With increasing the force, the error decreases.

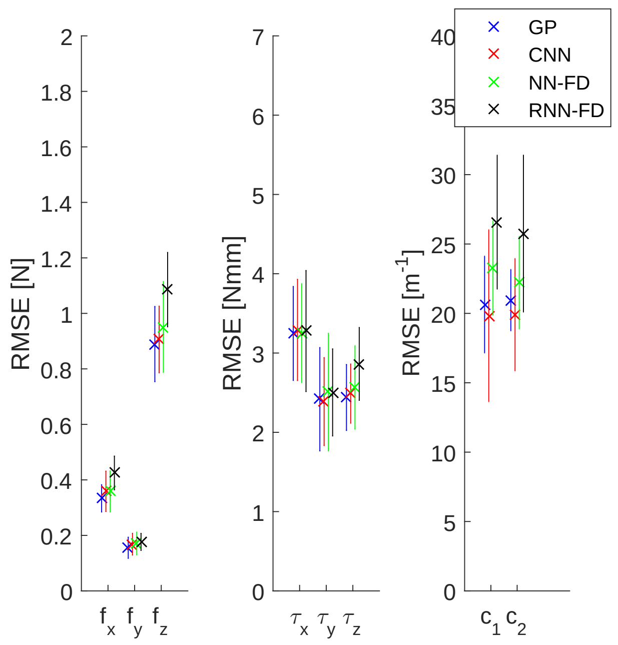


(d) RMS error in Normal (grip):Tangential (load) force ratio. The ratio is possibly extremely large when the tangential force is close to zero. Therefore, the data with tangential force less than 0.2 is excluded.

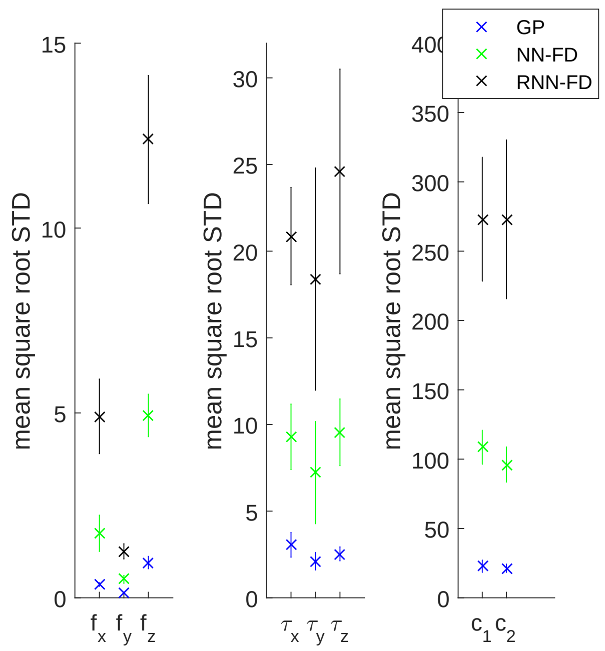


(e) RMS error in torque components τ_x, τ_y, τ_z. The percentiles are computed separately in τ_x, τ_y and τ_z.

Figure 12: Index fingers of 5 subjects using GPs. Each subject has one predictor model. One of five series is selected as testing data set for each kind of surface and object weight. (a)-(e): Each point represents a percentile of the values in the predicted f or τ for the percentages in the interval from 0 to 100 with five-percentage steps. The horizontal and the vertical axes are the mean values and the RMS errors of the percentiles respectively.



(a) The output error.



(b) The output standard deviation.

Figure 13: Predictor comparison for index fingers of 5 subjects. One of five series is selected as testing data set for each kind of surface and object weight. Each subject has one predictor model.

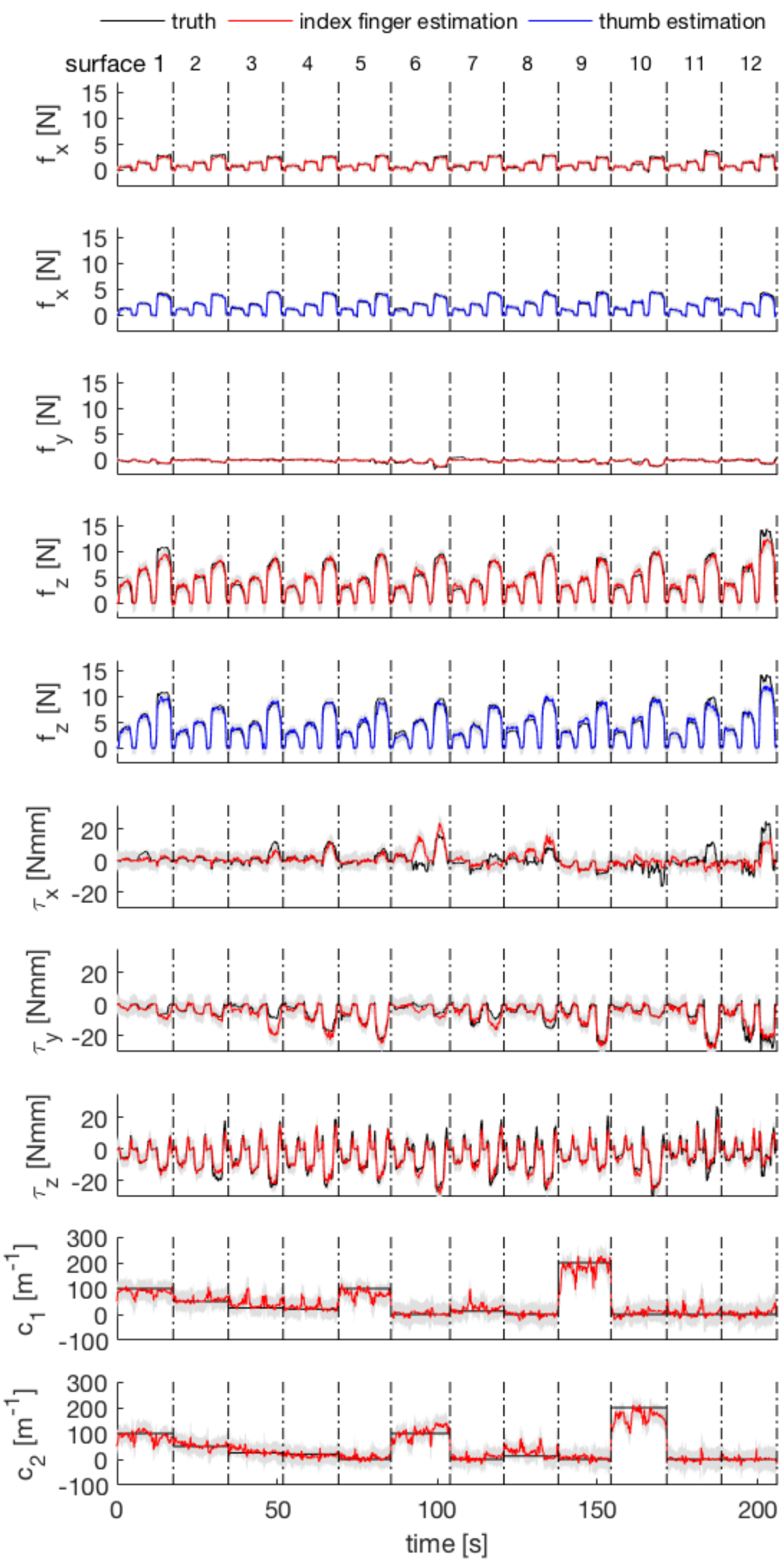


Figure 14: Prediction for S1 using GP. The result of force/torque prediction is shown. The grey areas are the 95% confidence interval. There are 36 trials. In each surface type the grasping is ordered by three types of weights. For observation, the time between two grasping is set to 0, and the grasping is separated by dashed lines instead.

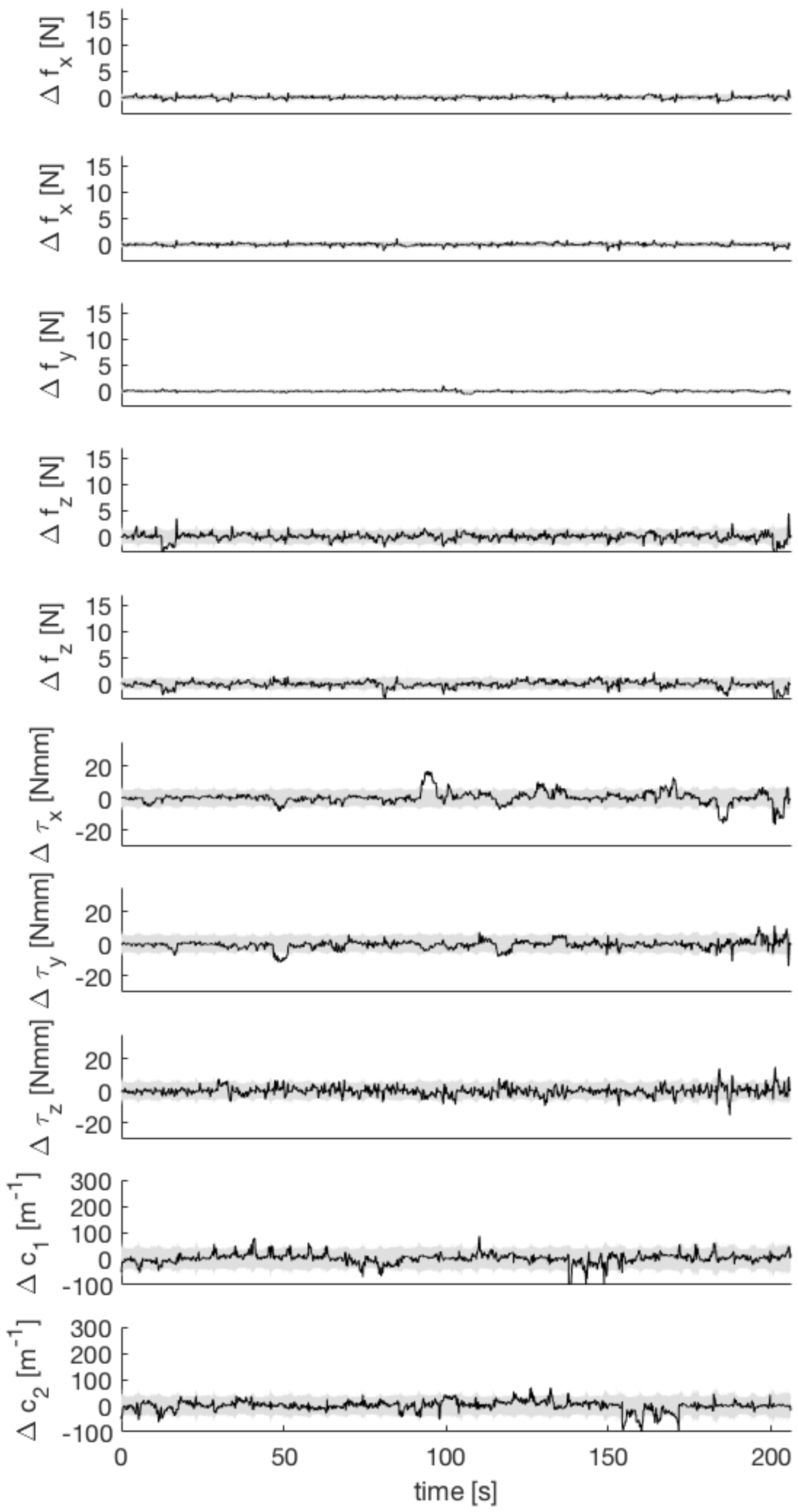


Figure 15: Prediction for S_1 using GP. The error of force/torque prediction is shown. The grey areas are the 95% confidence interval. There are 36 trials.

f_y $[-2, 0.8]\,\mathrm{N}$, f_z $[0, 15]\,\mathrm{N}$, τ_x $[-22, 26]\,\mathrm{Nmm}$, τ_y $[-30, 14]\,\mathrm{Nmm}$, τ_z $[-35, 27]\,\mathrm{Nmm}$, c_1 $[0, 200]\,\mathrm{m}^{-1}$ and c_2 $[0, 200]\,\mathrm{m}^{-1}$.

The predicted results of the index fingers of five subjects with four predictors are illustrated in Fig. 13. The accuracy of GP is slightly higher than that of CNN. On the other hand, CNN is able to efficiently process large data sets, especially image data, so that it is considerably faster than GP. Because GP and CNN are more accurate than NN-FD and RNN-FD, the following experiments are only based on GP and CNN. More information of the index fingers of 5 subjects using GP is shown in Fig. 12.

To perform further manipulation applications, multiple-finger prediction is crucial. Fig. 16,17 show that the GP accurately predicts the forces, torques and surface curvatures of the thumbs.

Furthermore, Fig. 14,15 show an example of the prediction of S_1 and Fig. 24 illustrates further details of three trials of picking up and replacing an object. It is able to predict norm forces of higher than 10 N, although when it is close to 15 N it is not as accurate as lower force predictions. The x axes of the fingers are almost vertical, which makes the sum of the f_x of the index finger and thumb is approximately equal to the object gravity during the holding process. f_z of the index finger and thumb are approximately equal to each other. At the low force frames, the fingernail images of different surfaces are very similar to each other, especially when the fingers have no contact to the surfaces, which causes that the surface curvature prediction c_1 and c_2 is not as accurate as that of the high-force frames.

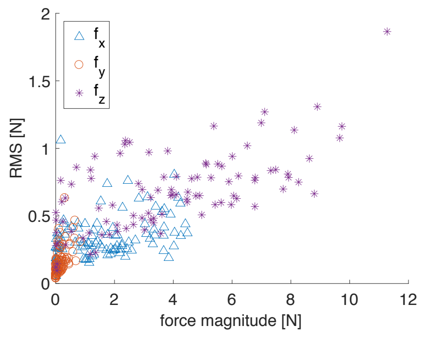
The current data processing method runs at approximately 30 Hz for an image resolution of 111×105 pixels, using an off-the-shelf GPU processor. Speedup is possible, e.g., by reducing the image resolution or using advanced hardwares.

3.3.3 *Surface Cross validation*

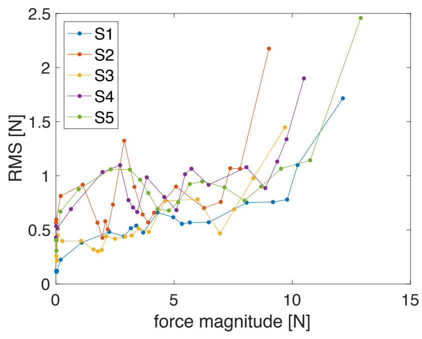
Our approach is capable of predicting the force and torque by surface cross validation (see Fig. 18, 19). The testing data set is surface 3, which is never shown in the training or validation data set.

3.3.4 *Predictor for all subjects*

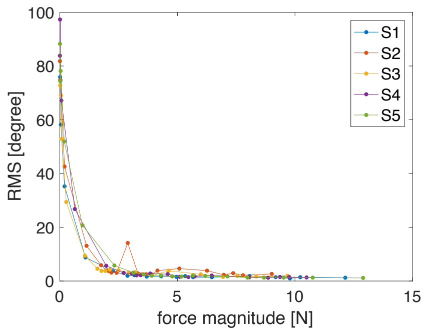
In this experiment, the training, validation and testing data sets are the same as in Section 3.3.2, but all subjects are combined in one model. To handle the large dataset size issue of GP, we randomly reduce the training data set to 30% and use GP-FITC with 17% inducing points chosen from the training data. On the other hand, CNN



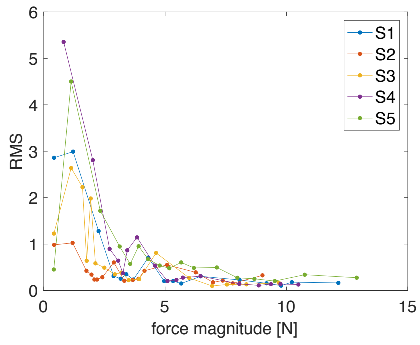
(a) RMS error in force components f_x, f_y, f_z.



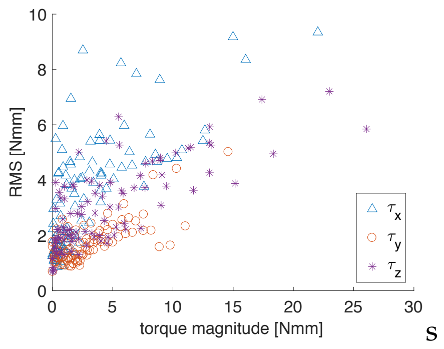
(b) RMS error in predicted force length magnitude.



(c) RMS error in angle between predicted and surface normal (f_z).

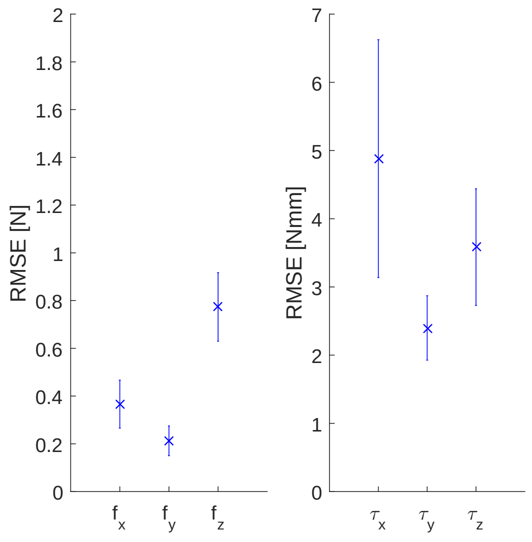


(d) RMS error in Normal:Tangential force ratio.

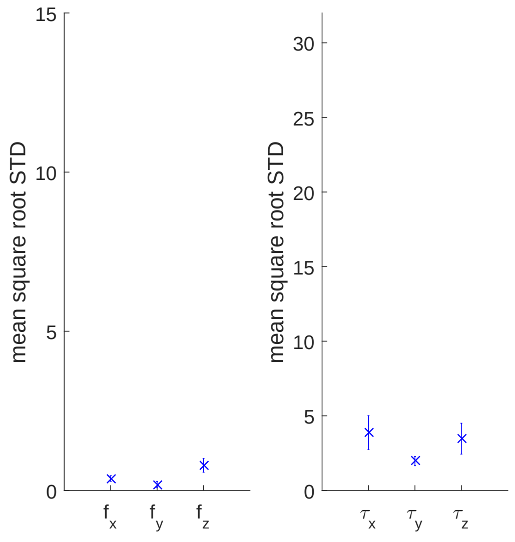


(e) RMS error in torque components τ_x, τ_y, τ_z.

Figure 16: Prediction of the thumbs of 5 subjects using GP. One of five series is selected as testing data set for each kind surface of the corresponding index finger and object weight. More annotations can be seen in Fig. 13.



(a) The output error.

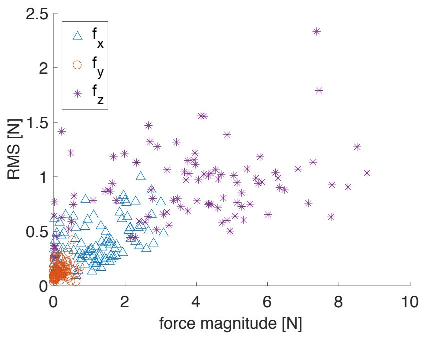


(b) The output standard deviation.

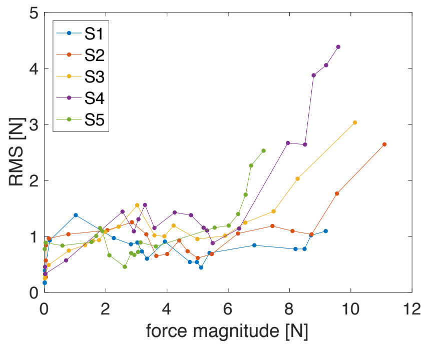
Figure 17: Prediction of the thumbs of 5 subjects using GP.

does not reduce any training data. As illustrated in Fig. 20,21, CNN performs more accurately than GP.

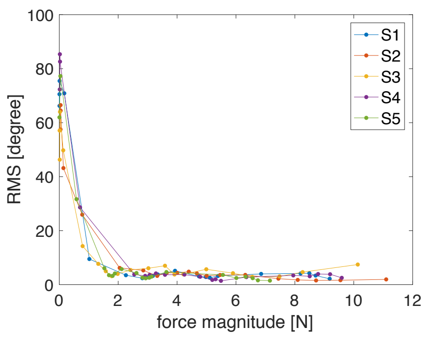
GP is more accurate than other models in simple conditions, e.g., training each subject separately with small data sets. On the other hand, CNN is more robust and faster when dealing with large data with more variables.



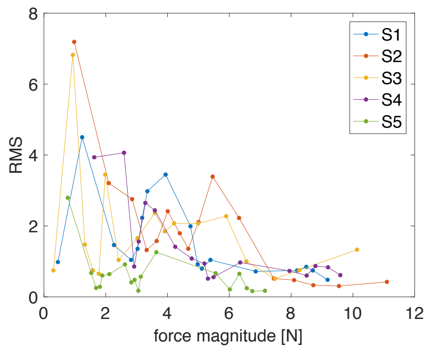
(a) RMS error in force components f_x, f_y, f_z.



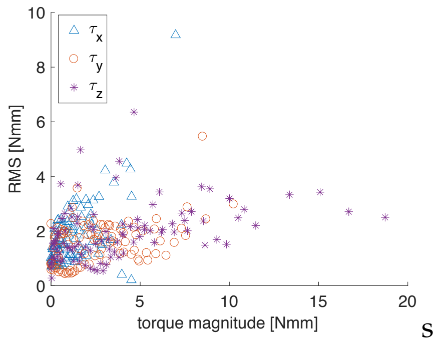
(b) RMS error in predicted force length magnitude.



(c) RMS error in angle between predicted and surface normal (f_z).

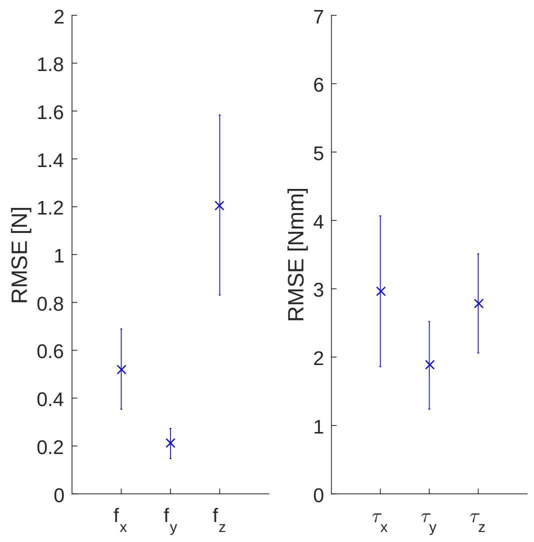


(d) RMS error in Normal:Tangential force ratio.

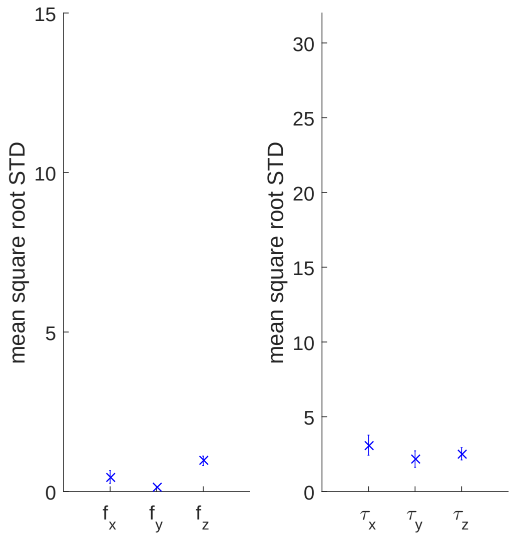


(e) RMS error in torque components τ_x, τ_y, τ_z.

Figure 18: Prediction of 5 subjects for surface cross validation. The testing data set is surface 3. More annotations can be seen in Fig. 13.



(a) The output error.

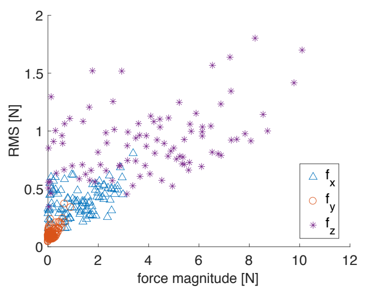


(b) The output standard deviation.

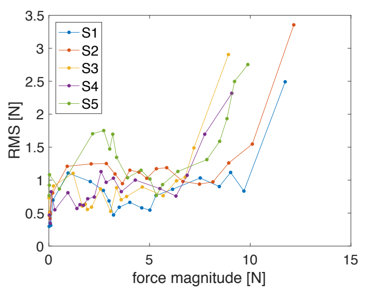
Figure 19: Prediction of 5 subjects for surface cross validation using GP.

3.3.5 *Time cross validation*

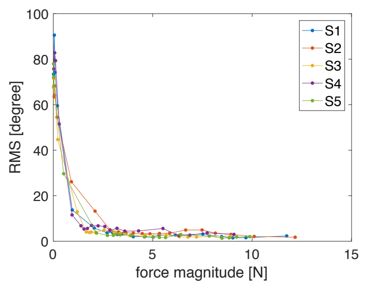
Variables such as finger temperature and lighting vary across time. In this experiment, the interval of the training and testing data recorded for S3 is 1 week. Table 1 and Fig. 22 illustrate the results of the GP prediction.



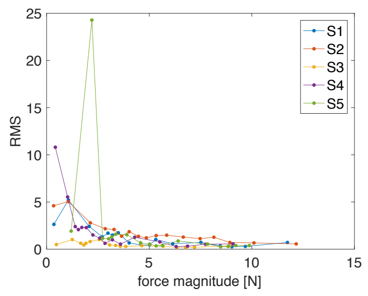
(a) RMS error in force components f_x, f_y, f_z.



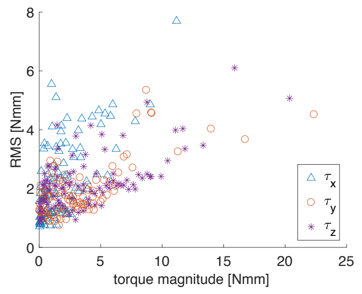
(b) RMS error in predicted force length magnitude.



(c) RMS error in angle between predicted and surface normal (F_Z).

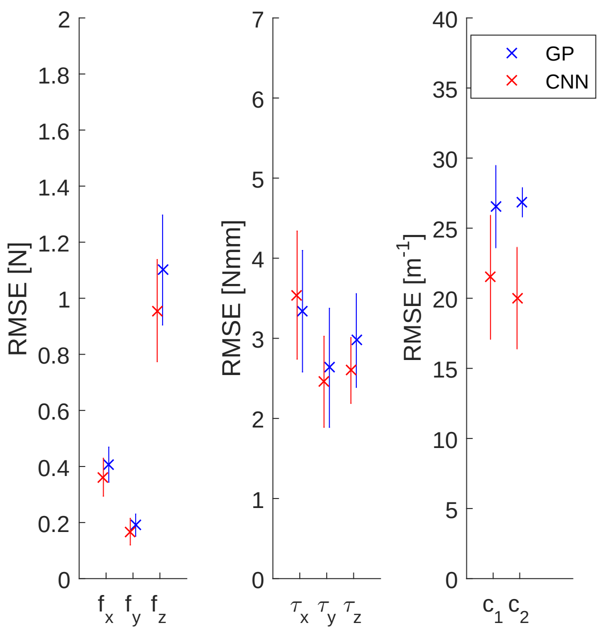


(d) RMS error in Normal:Tangential force ratio.

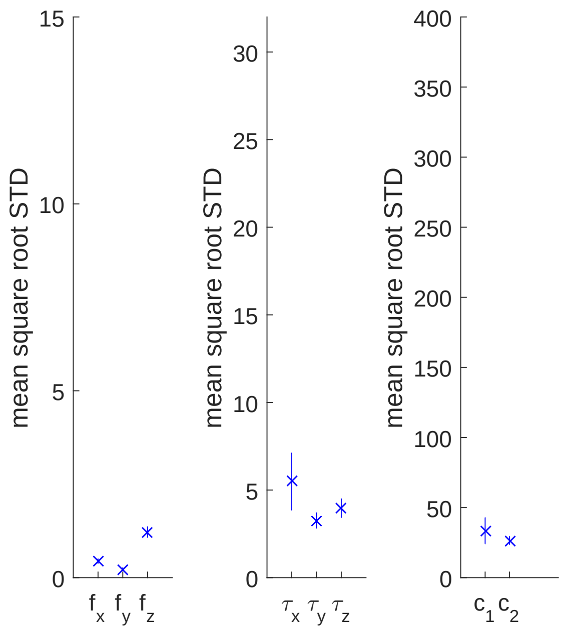


(e) RMS error in torque components τ_x, τ_y, τ_z. For observation, we use the absolute value of the torque.

Figure 20: Prediction of 5 subjects. One model is used for all 5 subjects. One of five series is selected as testing data set for each kind of surface and object weight. More annotations can be seen in Fig. 13.

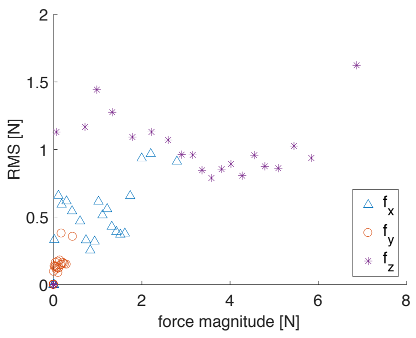


(a) The output error.

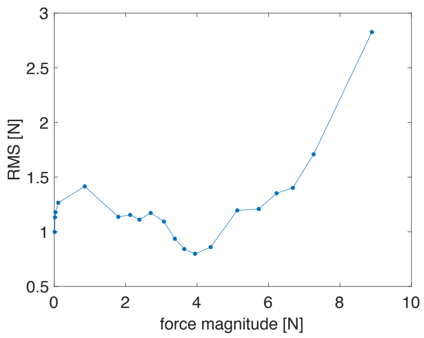


(b) The mean standard deviation output by GP.

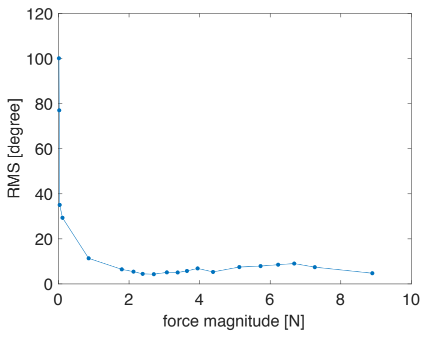
Figure 21: Prediction of 5 subjects. One model is used for all 5 subjects.



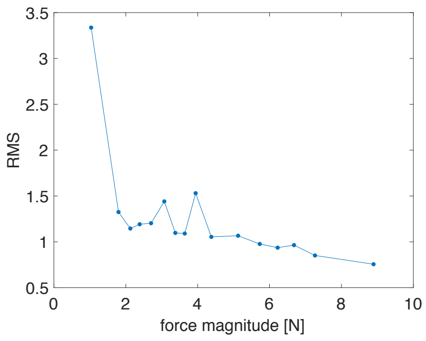
(a) RMS error in force components f_x, f_y, f_z.



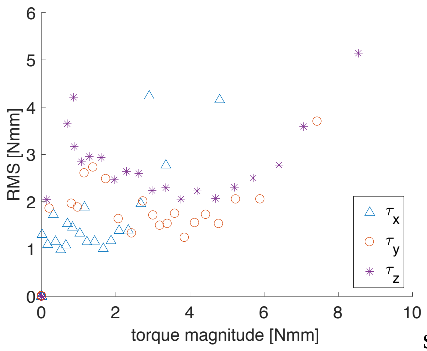
(b) RMS error in predicted force length magnitude.



(c) RMS error in angle between predicted and surface normal (f_z).



(d) RMS error in Normal:Tangential force ratio.



s

(e) RMS error in torque components τ_x, τ_y, τ_z.

Figure 22: Prediction of S_5 for times cross validation. The interval of the training and testing data recorded is 1 week. More annotations can be seen in Fig. 13.

Table 1: Time cross validation error. The interval of the training and testing data recorded is 1 week. RMSE of force and torque estimation by GP, as well as the mean of the estimated standard deviation (in brackets) given by the GP are listed below.

		x	y	z
	f	0.778(0.500)	0.197(0.179)	1.33(1.46)
S_1	τ	3.86(4.16)	2.81(2.86)	3.80(3.29)

3.3.6 *Subject cross validation*

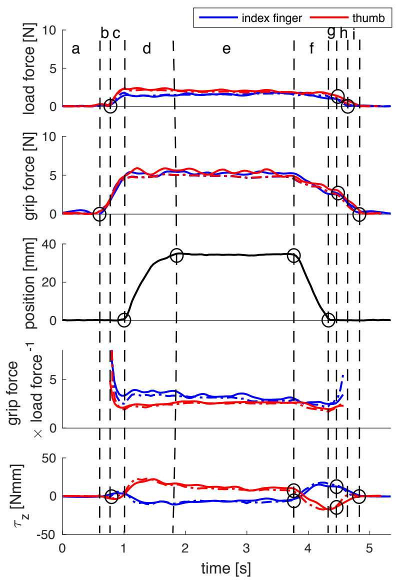
In this experiment, we take S_3 as the testing dataset, and the other four subjects as the training and/or the validation data sets. CNN can predict whether the force increases or decreases, but cannot accurately estimate the quantity of the forces, while GP only has constant outputs. CNN functions relatively well regardless of to changes to lighting, rotation and shift of the images. In contrast, GP is sensitive to these variables and has difficulty if these variables are never shown in the training data. In addition, the fingernail color distribution varies from subject to subject, which is the main reason for the inaccurate prediction in both methods.

3.3.7 *Human grasping analysis*

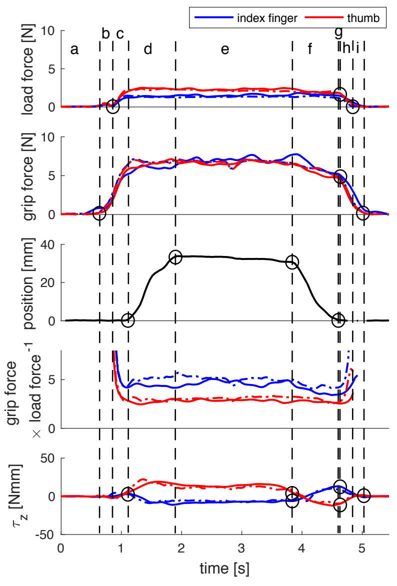
The approach can be used in settings such as robot teleoperation, force-based control and so on. As an example of the applications, we analyze human grasping in (Johansson and Flanagan, 2009) using the result of our predictor.

For observation, Fig. 24 only shows f_x, f_z and τ_z, which are most significant for picking up and replacing in our experiments. It is evaluated using S_1 grasping surface 3. From the figure, grasping can be seen in six steps: reaching, loading, lifting, holding, replacing and unloading. Other surfaces have similar plots for the steps.

Consider the index finger as an example, the loading starts from the critical point of forces between zero and nonzero. Lifting is from f_x peak to torque peak. Holding keeps both forces and torques stable. Replacing is from torque peak until f_x peak and unloading is from the lowest point of torque to zero forces.

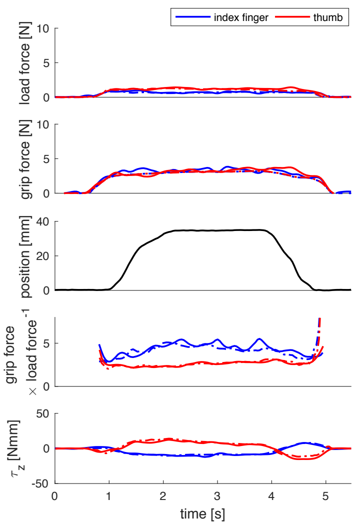


(a) Grasping the object with flat sandpaper surface.

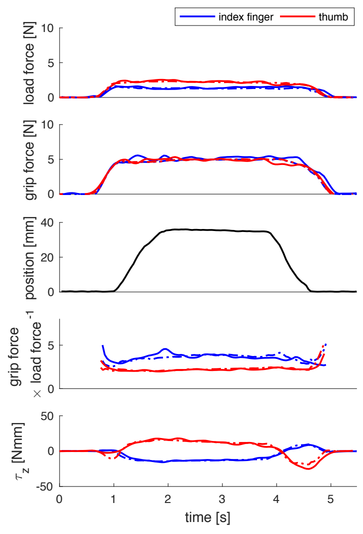


(b) Grasping the object with flat silk surface.

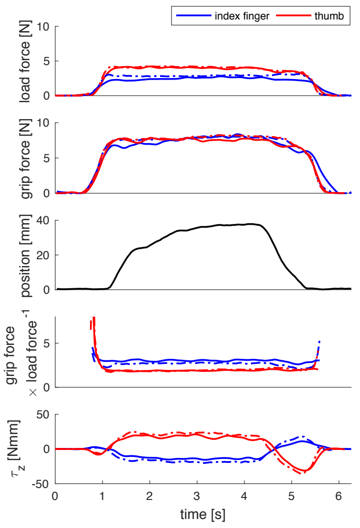
Figure 23: Grasping analysis of S_1 with 330 g weight. The dashed lines are the ground truth and the solid lines are the prediction. The load force is the force length magnitude of f_x and f_y and the grip force is f_z. The phases of the trials (Johansson and Flanagan, 2009) include: a - reaching phase; b - preload phase; c - loading phase; d - transitional phase; e - static phase; f - replacement phase; g - delay; h - pre-unload phase; i - unloading phase. h and i phases are possible integrated into one phase, which depends on the placing behaviour of the subjects. With the decreasing of the friction, the ratio (grip force × load force^{-1}) increases to avoid slipping. Subfigures in the third row are the position of the object.



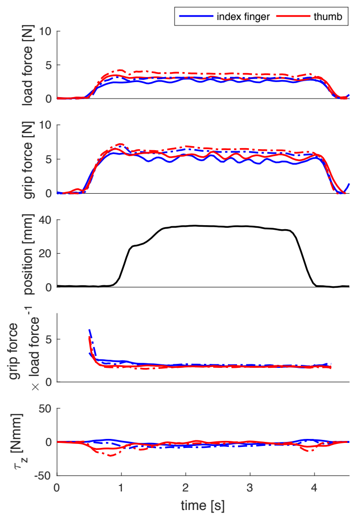
(a) Grasping of S1 the object with 165 g weight.



(b) Grasping of S1 the object with 330 g weight.



(c) Grasping of S1 the object with surface 660 g weight.



(d) Grasping of S2 the object with 660 g weight.

Figure 24: Grasping analysis of different subjects with surface 2 and different object weights. The sum of the load force of the thumb and index finger are approximately equal to the object gravity during the holding process.. The difference of the load force of the thumb and the index finger is caused by the inclination of the object. The grip forces of the thumb and the index finger are approximately equal to each other. See more annotations in Fig. 23.

3.4 CONCLUSIONS

This chapter has presented a method measuring finger contact force, torque, and surface curvature using the deformation and color distribution of the fingernail and its surrounding skin. Compared with prior studies, we remove a constrained lab setting with perfect conditions and restrictions. We then capture fingernail images when subjects make contact with various surfaces. After conducting non-rigid finger image alignments, we train force/torque and contact surface prediction models using the aligned images. The results show that the models accurately predict the force/torque used for picking up and replacing an object for multiple fingers and multiple people. The models were evaluated across surfaces and variables (e.g., finger temperature and lighting).

We compared four machine learning methods on their ability to predict force, torque and contact surface. The time-independent models is more accurate than the time-dependent model; therefore, the i.i.d. assumption of the data set is verified.

Throughout this chapter, we have studied models of "static movement" representations. Sequential data is commonly studied in movement modeling as well, which lead us to turn in next chapters to time-dependent models such as reinforcement learning and dynamic movement primitives.

Part IV

DYNAMIC MOVEMENT PREDICTION

4

STABLE REINFORCEMENT LEARNING WITH AUTOENCODERS FOR TACTILE AND VISUAL DATA

As seen in the previous chapter, "static movement" representation was explored for a data set with an i.i.d. assumption. In many common scenarios, movements do not have this assumption. Particular such data sets are sequential data including time-series data (e.g., speech and video) and ordered data (e.g., genes). In sequential data sets, the recent observations are more informative than the current observation in predicting the next value. In this and next chapters, we focus on cases with time-series data.

A time-series model, reinforcement learning, is now discussed to enable robots to learn new skills. However, it is non-trivial to take high-dimensional observation data, such as sensor reading of visual and tactile feedback, into consideration. In addition, since it is not straightforward to learn temporally stable representations, unsupervised deep learning, specifically, autoencoders, is modified to represent the temporal movements. Subsequently, dynamic modeling are used to predict control policies from reinforcement learning.

4.1 INTRODUCTION

To minimize the human engineering effort when teaching a robot new skills, robots should be able to learn through trial and error as formalized in the reinforcement learning framework. Reaching, grasping, and manipulation skills rely on high-dimensional sensor inputs: visual feedback is crucial in locating objects (Lampe and Riedmiller, 2013), whereas tactile feedback is critical for robustness to perturbations and inaccuracies (Lampe and Riedmiller, 2013; Pastor et al., 2012). Such high-dimensional sensory inputs pose a major challenge to reinforcement learning (RL) algorithms.

In robotics tasks, RL methods have addressed this challenge by relying on human-designed features to represent policies or value functions. For example, Lampe and Riedmiller (2013) used the center of mass extracted from visual images. A popular policy parametriza-

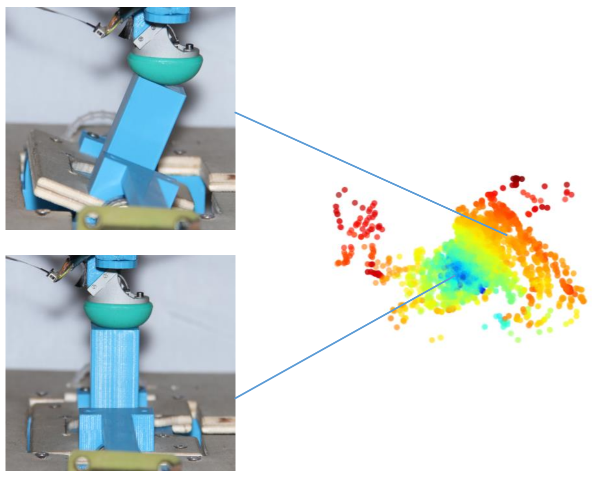


Figure 25: A 5-DoF robot with 228-dimension tactile sensor data learns to manipulate in a learned three-dimensional latent space. The right figure shows the latent representation of the tactile data. In the left figure, different robot states yield the depicted points in latent-space.

tion uses dynamic motor primitives to generate trajectories to be tracked (Kroemer et al., 2015; Chebotar et al., 2014; Kroemer et al., 2010; Kalakrishnan et al., 2011; Pastor et al., 2011; Kober et al., 2008). Feedback can be integrated using e.g. task-specific controllers (Kalakrishnan et al., 2011), anomaly detectors (Pastor et al., 2011), trajectories in sensor space (Chebotar et al., 2014), or by adding attractors (Kober et al., 2008). Designing such features or feedback terms is non-trivial in complex and high-dimensional sensory domains. These task-specific features are strongly dependent on manual tuning. We propose to instead learn policies in a widely applicable form from high-dimensional sensory inputs.

In previous work (van Hoof et al., 2015b,a), a RL method was proposed that uses non-parametric policies, which are widely applicable and do not rely on engineered features. This method is based on distances between input points. However, the distances are possible to be less meaningful, since there might be large displacement of the data caused by noises.

Neural network, c.q., deep learning-based methods have been used for RL. As described in Chapter 1, autoencoders are a promising latent representation method. We therefore reduce the dimensions using autoencoders for RL. A related study of our work is (Lange and Riedmiller, 2010; Mattner et al., 2012). In our work, we will explore a more advanced method, variational autoencoder (VAE) with dynamics, to represent the high dimensional data for RL. For instance, Fig. 25 illustrates a robot manipulating a pendulum in an autoencoded latent space.

We apply the novel method successfully to two tasks—a visual pendulum swing-up task and a robot manipulation with tactile data feedback task.

4.2 POLICY SEARCH WITH LEARNED REPRESENTATIONS

We consider our approach with two steps: representation learning from deep autoencoders and non-parametric relative entropy policy search in the learned latent space.

4.2.1 *Learning Representations using Autoencoders*

An autoencoder (Rumelhart et al., 1988; Bourlard and Kamp, 1988; Bengio, 2009) is a deep neural network that transforms the robot states $\mathbf{x} \in \mathbb{R}^d$ to a latent representation $\mathbf{z} \in \mathbb{R}^{d'}$ using an encoder. Subsequently, the latent representation is mapped back through a decoder (see Section 2.1.5). In this chapter, we used denoising autoencoders (DAs, see Section 2.1.5.1) and variational autoencoders (VAEs, see Section 2.1.5.3), a probabilistic model of data $\mathbf{x}$ and latent variables $\mathbf{z}$.

4.2.1.1 *VAE with Dynamics*

We modified the VAE to take the transition dynamics into account, which we expect to yield better representations based on performance of similar networks proposed by Watter et al. (2015); Krishnan et al. (2015). A linear layer is added between the encoder and the decoder (Fig. 26). It predicts the next latent state $\tilde{\mathbf{z}}_{t+1}$ from the latent state $\mathbf{z}_t$ and action $\mathbf{a}_t$

$$\tilde{\mathbf{z}}_{t+1} = [\mathbf{z}_t^T \ \ \mathbf{a}_t^T]\mathbf{w} + \mathbf{b}, \tag{70}$$

where $\mathbf{w}$ and $\mathbf{b}$ are parameters represented by θ_a. After that, the model reconstructs the next state $\mathbf{x}_{t+1}$ from $\tilde{\mathbf{z}}_{t+1}$. The transition layer is chosen to be linear to enforce control-affine dynamics, which are convenient for control tasks. The modified lower bound is defined as

$$\begin{aligned}\mathcal{L}(\theta, \theta_a, \phi; \mathbf{x}_t, \mathbf{a}, \mathbf{x}_{t+1}) =& \mathrm{E}_{q_\phi(\mathbf{z}|\mathbf{x}_t)}[\log p_\theta(\mathbf{x}_{t+1}|\mathbf{z}, \mathbf{a})] \\ & - \mathbb{KL}(q_\phi(\mathbf{z}|\mathbf{x}_t) \| p_\theta(\mathbf{z})),\end{aligned} \tag{71}$$

taking the dynamics into account.

4.2.1.2 *DA with Dynamics*

Similar to the VAE with dynamics, the denoising autoencoder with dynamics takes transitions into account and has an additional layer

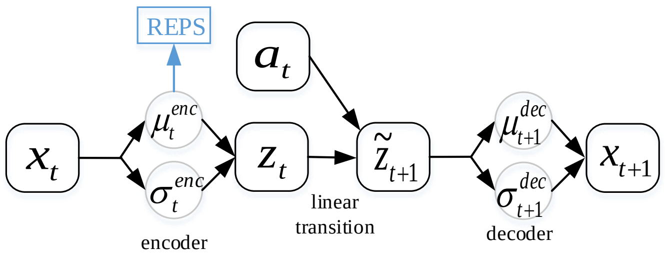


Figure 26: Illustration of the VAE with dynamics. The network is trained such that the decoder output is close to the next state x_{t+1}. After training, μ_t^{enc} is used as input to the reinforcement learning algorithms, which generates data used to update the network in turn. μ and σ represent the mean and standard deviation respectively.

as described in (70), written as $\tilde{\mathbf{z}}_{t+1} = h_{\theta_a}(\mathbf{z}_t, \mathbf{a})$, in the latent space. The weights are updated by optimizing

$$[\theta^\star, \theta'^\star, \theta_a^\star] = \underset{[\theta,\theta',\theta_a]}{\operatorname{argmin}} \sum_{t=1}^{n-1} \frac{\mathcal{L}[\mathbf{x}_{t+1}, g_{\theta'}(h_{\theta_a}(f_\theta(\tilde{\mathbf{x}}_t), \mathbf{a}))]}{n-1},$$

where, f and g are the decoder and encoder respectively.

4.2.1.3 *Latent Space Updates with On-Policy Samples*

To represent data most relevant for the current policy, we consider a variant where we update the autoencoder using samples from the most recent policies. We can either use data from the most recent iterations only, or, when little data is available, we can give recent data a higher weight in the loss function. We will specify the data used for re-training the latent space for each experiment individually.

4.2.1.4 *Non-Parametric Policy Updates*

To update the control policy based on the learned representation, we use the non-parametric relative entropy policy search (NP-REPS, see Section 2.1.7.1) algorithm introduced in (van Hoof et al., 2015b). This algorithm aims to maximize the expected rewards obtained by the policy while bounding the relative entropy between successive policies. This bound controls the greediness of the algorithm, and thus prevents overfitting to small batches of data as well as oscillations or divergence in the learning process.

4.3 EXPERIMENTAL SET-UP AND RESULTS

In this section, we experimentally study the performance of our proposed algorithms using high-dimensional sensory input. We first perform a simulated pendulum swing-up experiment with image data, and then perform a 5 DOF robot manipulation task with tactile data.

4.3.1 *Experimental Set-Ups*

To explore environment, the reinforcement learning agent is initialized using 30 roll-outs for the simulated pendulum swing-up task and 45 roll-outs for the real-robot task with random exploratory policy. The learned RL model and the policy of the agent are updated after every iteration. To reduce computations, only the last three iterations are used for updating. We set the KL bound ϵ of REPS to 0.5.

SIMULATED VISUAL PENDULUM SWING-UP In this experiment, we simulate a pendulum that has to be swung up and balanced at the upright position, based on visual input. We reproduce the set-up of (van Hoof et al., 2015b), where the pendulum has length $l = 0.5\,\mathrm{m}$, mass $m = 10\,\mathrm{kg}$, and friction coefficient $k = 0.25\,\mathrm{Ns}$. The action a is a torque applied at the pivot, with a maximum of 30 Nm such that the pendulum cannot be swung up from the downward position directly.

After each time step of $0.05\,\mathrm{s}$, the agent receives a reward according to $r(s, a) = -10\theta^2 - 0.1\dot{\theta}^2 - 10^{-3}a^2$. A rather high level of additive noise is applied to the controls (with a standard deviation of $4.5\,\mathrm{Nm}$). We ended the roll-out after each time-step with a probability of 0.02, resulting in an effective discount factor $\gamma = 0.98$. For this experiment, we used 10 roll-outs per iteration.

The agent only has access to images of the pendulum. We render an image of the pendulum in its current state, as well as a difference image between visual representations of the pendulum in its current and previous state (to provide a notion of angular velocity). We blur the image and difference image with a Gaussian filter to enhance the spatial generalizability of the kernel. This filter has a standard deviation of 20% of the image width. Both images are resized to 20×20 pixels and concatenated into a vector with 800 features. We choose squared-exponential kernels for all variables.

The denoising autoencoder (DA) has one input layer, five hidden layers and one output layer. The number of neurons per layer are 800, 120, 50, d', 50, 120 and 800 respectively, where d' is the number of latent dimensions. We set the corruption level to 0.2, which means 20% of the image is corrupted in every training image. The DA with dynamics uses the same parameters but uses one more layer, as explained in Sec. 4.2.1.2. For the VAE, we set the number of neurons per layer to 800, 512, d', 512, and 800, respectively. All network parameters where chosen to minimize the reconstruction error. The VAE with

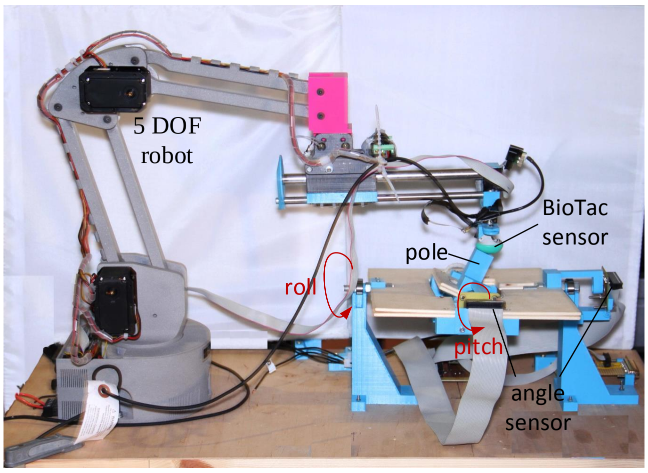


Figure 27: Set-up of the tactile control experiment. The five-DoF robot touches a blue pole that can rotate about two axes using the green BioTac fingertip sensor. The pitch and roll of the pole are measured to provide a feedback signal, but not used in the control policy.

dynamics has the same architecture, except the layer from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}$, as shown in Fig. 26. The encoder networks in both VAE and VAE with dynamics used the square as the transfer function for the σ output. We furthermore evaluate a VAE that is retrained on the three most recent iterations with each policy update.

With this set-up, we want to answer two questions. First, we want to investigate which type of autoencoder is most suitable to find features for the reinforcement learning task. Secondly, we want to validate whether updating the autoencoder to represent the states visited by the most recent policy improves learning of the task.

REAL-ROBOT TACTILE CONTROL EXPERIMENT In this experiment, a 5 degree of freedom robot manipulates a pole through a SynTouch BioTac tactile sensor on the end-effector (see Fig. 27). The pole is on a platform which is able to rotate in roll (α) and pitch (β), measured in degrees. Two angle sensors are mounted on the platform to measure the pole angles. The initial position of the pole is a $5°$ rotation from the central position with a uniformly random angle. The task for the robot is to move the pole to the stable center position where α and β are zero degree by manipulating the top of the pole. The reward function defining this upright pole position as a goal is $r(s, a) = 60(\exp(-(\alpha^2 + \beta^2)/60) - 1)$. The reset probability is set to 0.05 (equivalent to $\gamma = 0.95$). In this experiment, we use 15 roll-outs per iteration.

The input to the neural network is a 12-time step window of the 19 electrodes of the BioTac sensor, yielding 228-dimensional input data.

We used a hidden layer with 512 neurons and a feature layer with three neurons, chosen based on reconstruction error. In this experiment, the σ values of the decoder networks are constant, independent of $\mathbf{z}$. Actions consist of an increment in forward-backward and left-right position. The desired position is kept constant during the 33 ms time window. The control frequency is 2.8 kHz.

As the data size recorded on the real robot is relatively small, sampled data from all previous iterations is used to retrain the autoencoder. Recent data is more relevant than older data, and therefore errors on recent data were given a bigger weight in the loss function (triple weight for the most recent iteration, and double for the iteration before that).

4.3.2 *Results of the Visual Pendulum Swing-up Experiments*

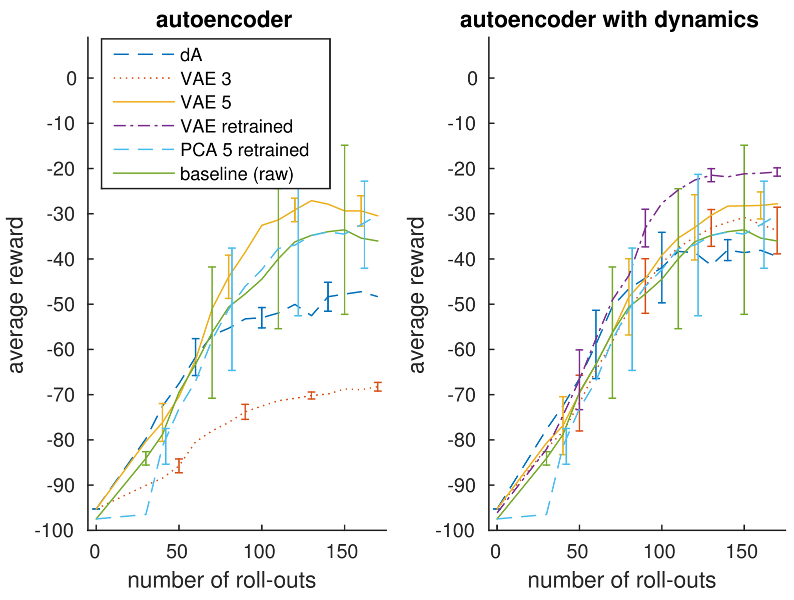


Figure 28: Comparison of learning curves for the pendulum task using representations learned with different types of autoencoders. Shown are results of denoising autoencoders (DA) with 10 latent variables, and of variational autoencoders (VAE) with three or five latent variables. For the best category of encoders, 'VAE 5', we also compare to a version that is retrained with every policy update. Error bars show the sample standard deviation. Averages are calculated over five independent runs, using a single learned feature space per encoder type. The 'retrained' encoder is retrained independently for each run. Rollouts contained 50 time steps, on average.

In the first evaluation on the simulated visual pendulum swing-up task, we compared different types of autoencoders and inspect the effect of re-training the encoder on the most recently sampled data. For the evaluation, we use five independent trials for each feature representation. The results of this experiment are shown in Fig. 28. Compared to the denoising autoencoder, most variational autoencoders provide representations that yield higher average rewards when used with reinforcement learning compared to using the raw sensor data, a truncated PCA with five components, or the denoising autoencoder in our tasks. An exception is the variational autoencoder with three latent features without additional dynamics. The learned representation for this type of encoder is presented in Fig. 29. Without the dynamics information, points which are regularly spaced in the original space are not so in the learned feature space. The encoder that forces the dynamics to be linear learned a more regular feature space. In the other cases, the encoders with dynamics also tended to yield slightly better representations.

Figure 28 also shows the performance based on features that are updated as the policy improves. These updates should intuitively ensure that the variational autoencoder focuses on representing the states visited by the current policy. In our experiment, updating the autoencoder this way improves the learning progress. As training the encoder is a computationally expensive procedure, we only performed this evaluation for the most promising encoder type.

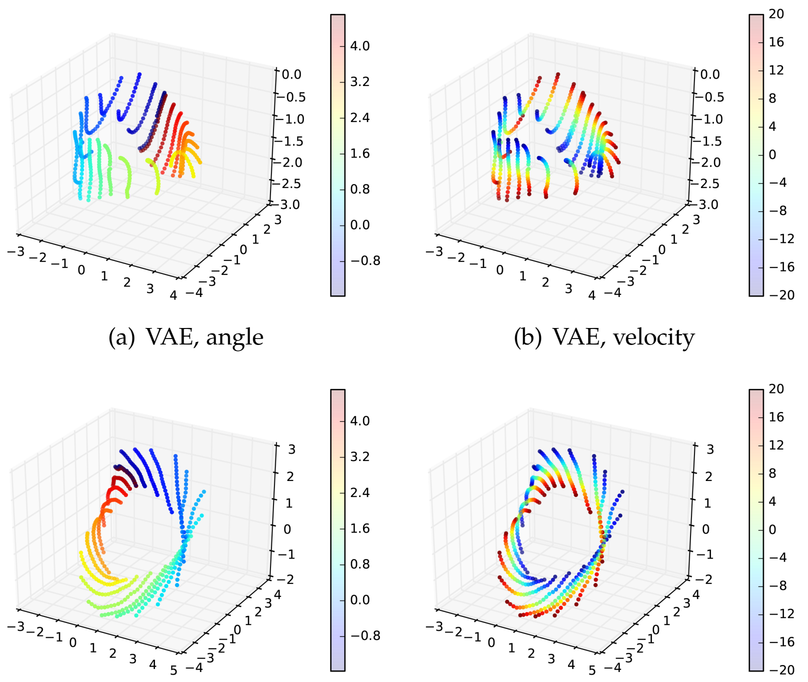
4.3.3 *Results of the Tactile Manipulation Robot Experiment*

The visual pendulum experiment indicates that the representation learning of retrained VAE with dynamics performed best. This representation was thus used for the robot manipulation task. Fig. 30 shows the learned representation of the tactile sensor. With this representation, the average reward of five independent trials increased from -49.5 to -13.8 in 165 roll-outs, and thereafter it achieved -9.9 using exploiting policy. However, the average reward is -52.0 in 165 roll-outs with the raw tactile data.

4.4 DISCUSSION AND CONCLUSION

In this study, we have presented a reinforcement learning algorithm that learns non-linear policies in a low-dimensional latent space of an autoencoder with high-dimensional sensory inputs.

In a simulated visual pendulum swing-up experiment, variational autoencoders obtained significantly better results than other dimension reduction methods including denoising autoencoder and PCA. We developed variational autoencoders to be time-dependent, which force the data to have a more meaningful latent space, rather than en-



(a) VAE, angle (b) VAE, velocity

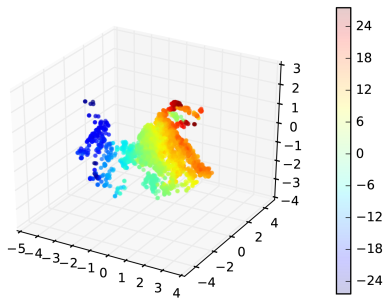
(c) VAE+dynamics, angle (d) VAE+dynamics, velocity

Figure 29: Visualization of the learned feature spaces by the VAE with 3 latent features. The graph shows the three-dimensional feature space, represented by the x, y and z axis. The inputs are visual data with a grid over the state space—the position and the velocity of the pendulum. The color encodes the pendulum angle (left column) or velocity (right column). The latent space was learned using the variational autoencoders (top row), or the modified encoders with system dynamics (bottom row). The angle and velocity are not directly available to the learning algorithm, but are used to calculate the learner's reward. All learned structures reflect the periodic nature of the angle variable.

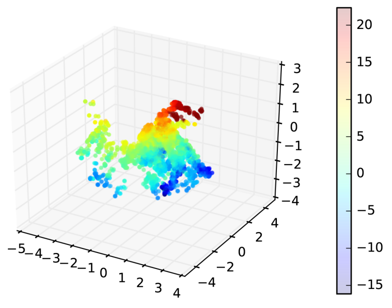
code individual input patterns. The modified VAE tended to improve reinforcement learning performance. Re-training the encoders on the state distribution induced by the policy markedly improved performance. In this case, the encoder objective incurs the biggest loss for states that are most relevant to the policy, and therefore, such states are likely to be represented especially well in the learned latent space.

In a real-robot experiment, a manipulation task was performed with high-dimensional tactile data. The latent space encoded the contact information, specifically, the roll and pitch of a handled pole, from tactile sensor into a low latent space. In this latent representation, the robot successfully learned policy to stabilize the pole.

In this chapter, we developed a novel model of the latent representation using the current state and action at time t and the next state at $t+1$. In the next two chapters, we will exploit the sequential patterns



(a) roll



(b) pitch

Figure 30: Latent space for tactile sensor. The axes represent three-dimensional latent values, the samples are colored according to roll and pitch of the platform. The visualization shows that in the learned latent space, the pitch and roll components are perpendicular to each other.

with long sequences which are probably more informative than that of two time steps. In addition, the dynamic movement model and the latent representation are trained separately in this chapter. We therefore will discuss whether seamless integration of these two types of models affects the patterns of the latent space.

5 EFFICIENT MOVEMENT REPRESENTATION BY EMBEDDING DYNAMIC MOVEMENT PRIMITIVES IN DEEP AUTOENCODERS

In Chapter 4, it is found that a model that combines a reinforcement learning setting and deep autoencoders, can build meaningful latent spaces and predict movements. The latent representation was learned by integrating the current and next states and the current controller into autoencoders. In order to develop a more powerful latent representation, it is necessary to exploit long sequence movements into unsupervised learning models, which is the main goal of this chapter.

In addition, besides reinforcement learning, an alternative algorithm, Dynamic Movement Primitives (DMP), is also a remarkable model for human and robot movements. Following the research presented in Chapter 4, a method is presented that embeds DMP into a deep autoencoder for human movement prediction. The embedded architecture then allows the DMP and the autoencoder to be trained as a unit, which forces the latent space to be more meaningful.

To further improve the generalization ability of the model, sparsities are added to clearly distinguish different movements in the latent space, which are used for novel motion generation. Additionally, the model is able to reconstruct missing data and change the goal attractor.

5.1 INTRODUCTION

The compact representation of movement sequences is a key element in predicting or analyzing such movement. For voluntary limb movement, simple polynomial approximations may suffice as these can represent minimal acceleration change over time (Flash and Hogan, 1985). However, for more complex movements, involving interaction or whole-body motion, such simple models do not suffice; indeed, a closed-form representation can usually not be found.

Dynamic movement primitives (DMP) were developed to represent movement for programming by demonstration (Ijspeert et al., 2002).

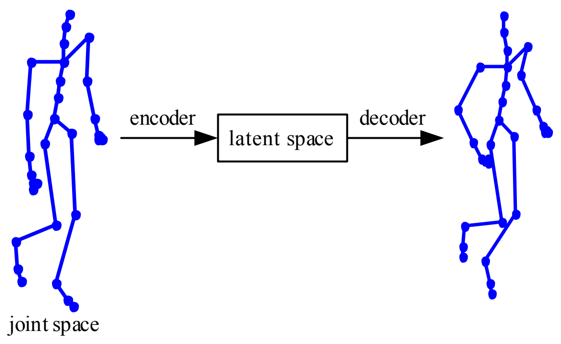


Figure 31: AE-DMP: Describing movement modified in latent space.

They represent nonlinear dynamic systems by second-order differential equations, if the parameters are correspondingly tuned. Additional parameters allow their variation with respect to their state space, thus varying, for instance, speed of motion. In their original interpretation, DMPs are learned in joint space of the robot or human. This has the disadvantage that adaptation of the DMP parameters has no clear relationship to the task space, and the DMPs do not generalize well. Also, setting the DMPs up in task space (Pastor et al., 2009) is problematic for high-dimensional systems, e.g. for humanoid or human movement. Instead we were inspired by the approach in (Bitzer and Vijayakumar, 2009) by representing the movement in a physically not defined, latent, space. In that paper, the authors propose to embed DMPs in Gaussian Process Latent Variable Models (GPLVM) by (a) having the GPLVMs learn a latent space of the movement, and (b) applying DMPs in that latent space.

Alternative approaches, building on the representational power of deep neural networks, have obtained competitive results. In (Taylor et al., 2007; Sutskever et al., 2009; Boulanger-Lewandowski et al., 2012) connectionist models were trained to represent human movement; these methods rely on binary latent variables of which it is questionable whether they pose a reasonable prior for human or humanoid kinematics. Bayer and Osendorfer (2014) propose a recurrent architecture with continuous latent variables, but their approach does not exploit prior knowledge on human motion.

In the approach we take in this study we extend deep neural networks in the form of autoencoders (Vincent et al., 2008) with DMPs. The autoencoder is used to learn a nonlinear dimension reduction of its data, leading to latent representations of these data in a layer of hidden units (as exemplarily illustrated in Fig. 31). However, rather than having these hidden units represent general data, we restrict them to be DMPs which can optimally represent dynamic movement,

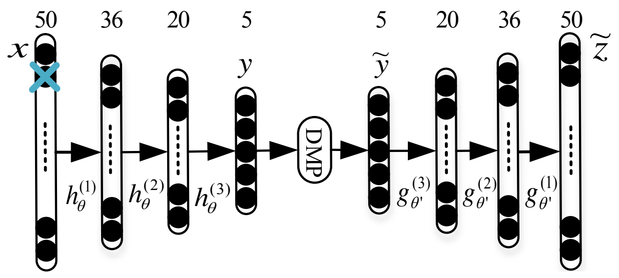


Figure 32: Architecture of AE-DMP. $\times$ represents the joints removed by denoising. Numbers indicate hidden layer sizes.

and generalize to similar movements of the system that is represented. In effect, we constrain the latent representation to represent the dynamics of the investigated system.

In the sequel we will formally define our approach and demonstrate its use on human kinematic data from the CMU Motion Capture Database.

5.2 AUTOENCODED DYNAMIC MOVEMENT PRIMITIVES

In this section, our new model of deep autoencoders, and specifically, denoising autoencoders (see Section 2.1.5.1), fitted with dynamic movement primitives (see Section 2.2.1) will be presented. In order to extract the features, a deep autoencoder (Rumelhart et al., 1988; Bourlard and Kamp, 1988; Bengio, 2009) is used to reduce the dimensionality from instantaneous high-dimensional joint information, while DMPs are able to make use of the feature data for a movement coded in time.

5.2.1 *Autoencoded Dynamic Movement Primitive*

Our model embeds DMP into denoising autoencoder seamlessly by taking DMP as one of the hidden layer as shown in Fig. 32. The number of hidden layers can be increased to form deeper architectures.

The parameters $\mathbf{w}$ can be trained using various machine learning methods such as locally weighted regression (LWR) (Schaal and Atkeson, 1998) and Gaussian Mixture Models (GMM) (Calinon et al., 2012). However, in order to fit DMP into autoencoder, the learning of $\mathbf{w}$ is accomplished using neural networks.

Therefore, given a sequence of data, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, in the observation space, the minimizer of AE-DMP is rewritten from (17) and (63) as

$$\mathbf{w}^\star, \theta^\star, \theta'^\star = \arg\min_{\mathbf{w},\theta,\theta'} \sum_{i=1}^{m} \left\{ \mathcal{L}(\mathbf{x}_i, \tilde{\mathbf{z}}_i) + \lambda \mathcal{L}\left[\mathbf{f}_i, \mathbf{f}_i^{target}\right] \right\}, \tag{72}$$

where λ is a penalty parameter, and different from (17) $\tilde{\mathbf{z}}$ is the reconstruction of the human movement in joint space based on the output of the DMP component $\tilde{\mathbf{y}}$. The second term of the the penalty $\mathcal{L}(\mathbf{f}_i, \mathbf{f}_i^{target})$ enables the DMP output to follow the demonstration accurately in the feature space. In addition, without $\mathcal{L}(\mathbf{f}_i, \mathbf{f}_i^{target})$, the neural networks take Ψ as the input and the encoder layers are missing during the training process. The weights are updated for a whole sequence batch. We have one $\mathbf{w}$ for one demonstration, because DMP is a model trained by one demonstration.

We pre-train the model by denoising autoencoder and DMP separately, then obtain the final weights from (72). Thus, the model autonomously learns movements in the feature space without hand-engineered representations.

5.2.2 *Sparse AE-DMP*

When the data contains multiple motions, AE-DMP is able to switch the motions or interpolate novel motions between existing ones. Sparse AE-DMP is an effective approach to achieve this with reasonable explanation in the hidden units. Adding an extra term of l_1 norm penalty to the loss function, sparsity (see Section 2.1.5.2) enforces hidden units to be rarely active (Nair and Hinton, 2009)

$$\mathcal{L}(\mathbf{y}) = \eta \sum_{i=1}^{m} \|\mathbf{y}_i\|_1, \tag{73}$$

where η is a penalty parameter. We then have a sparse solution for $\mathbf{y}$. For various movements, sparsity is designed to deactivate a part of hidden neurons for distinguishing the movements, so l_1 norm which can yield extreme small values of $\mathbf{y}$ is chosen. On the other hand, l_2 cannot deactivate the neurons and the derivation of pseudo-norm l_0 is not smooth.

Thus, for multiple movements, sparsity encodes one movement into only a few hidden units while other movements into other units. To switch or interpolate existing movements, we can simply deactivate/activate or interpolate these hidden units. This approach can be used as a motion primitive library for further development such as movement segmentation and recognition.

5.3 EXPERIMENTS

5.3.1 *Human Motion Data*

We use the CMU Graphics Lab Motion Capture Database to evaluate AE-DMP. That database is created by tracking 41 markers on human subjects during walking, jogging, etc. using a Vicon optical tracking system. The data are preprocessed using Vicon Bodybuilder to render limb joint angles, leading to a 62-dimensional feature vector. Of these, 6 values are almost constant (shoulder and finger movement, having an insignificant variance $< 10^{-26}$) and are ignored. Another set of 6 values represent the origin frame, and can be removed after all data are represented with respect to this frame. The resulting feature vector used in this study is 50-dimensional. For the purpose of this paper we evaluated results on 5 subjects (viz. Subject 35, 49, 74, 120 and 143) with various movements. Subject 49, 74, 120 and 143 are randomly selected, while subject 35 is stable, which is widely used for algorithm evaluation (Schölkopf et al., 2007; Bitzer et al., 2008).

Before training to the AE-DMP, the data is normalized in every dimension to zero mean and range in $[-1, 1]$. Visualization is done by means of the software described in (Urtasun et al., 2008).

In the following experiments, we use the mean square error (MSE $\pm$ standard deviation) to numerically evaluate the model. All results are reported in radians.

5.3.2 *Features in the Hidden Neurons*

To illustrate what movement the hidden neurons in AE-DMP encode, we train the model using a walking motion in this experiment. 21 sequences are trained and every sequence is segmented or scaled to 70 frames. After that, additional results of 5 subjects with 1 sequence for every movement are evaluated. We use the autoencoder to reduce the input dimensionality from 50 to 5. The used autoencoder consists of one input layer, 3 encoder layers with 36, 20 and 5 neurons respectively, one dynamic movement primitive layer, and 3 decoder layers plus output layer (see Fig. 32). The number of neurons in every layer is chosen for minimization of the reconstruction error for autoencoder. In the DMPs we use 20 weighting kernels Ψ of the primitive. Following Ijspeert et al. (2013) we set $\beta_z = 0.25\alpha_z$ to obtain critical damping. β_z can be tuned to vary the movement between smoothness and accuracy of following the given trajectory.

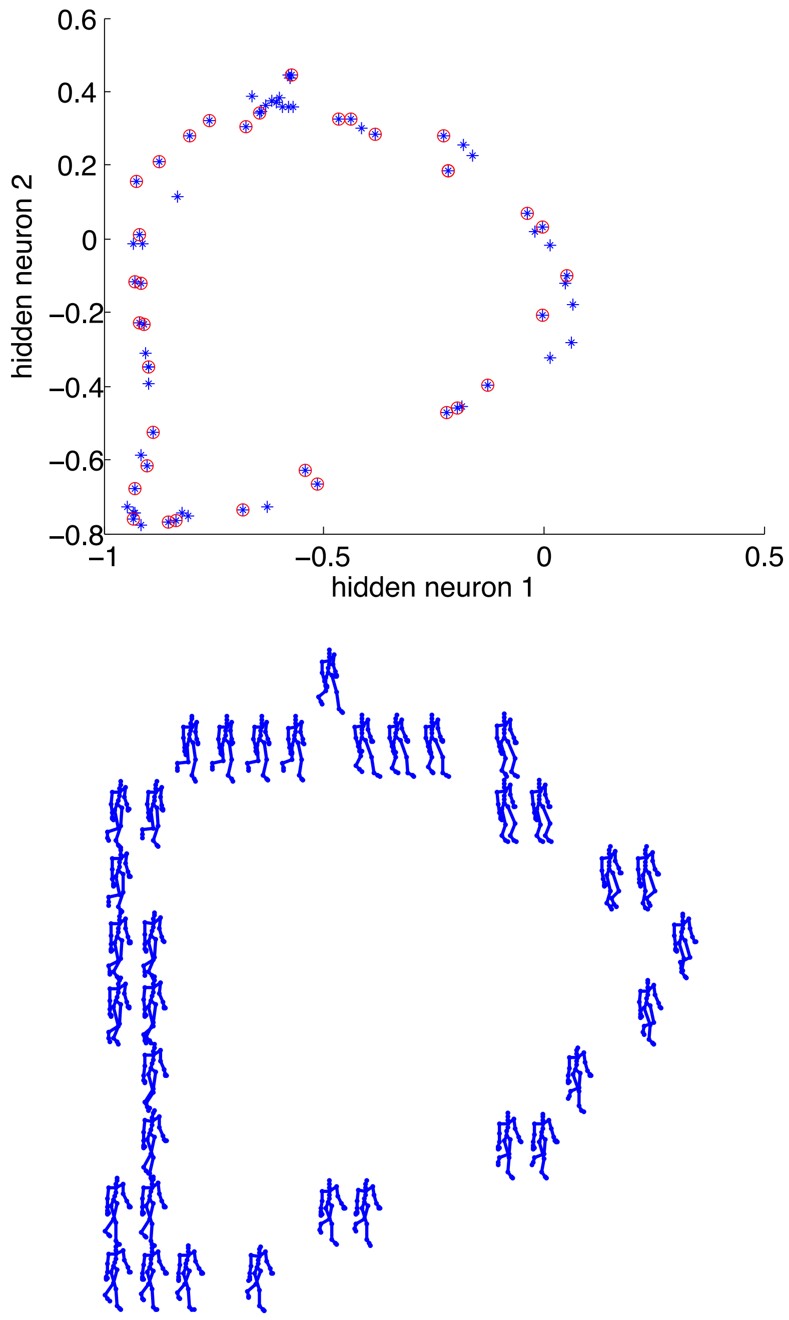
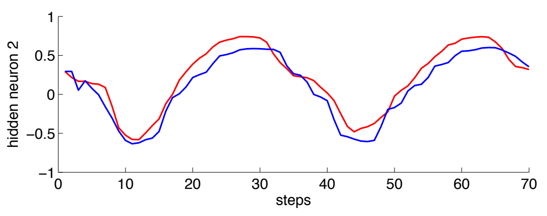
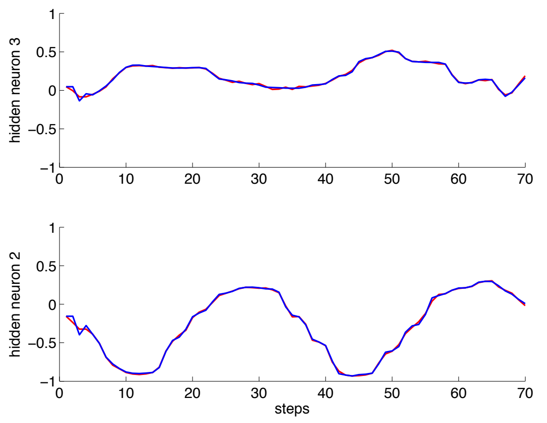


Figure 33: Data in hidden neurons of integrated AE-DMP. The blue stars in the upper figure are a sequence data in the two hidden neurons, and the red circles in the upper figure are selected to be plotted in the lower figure in the joint space. For visualization, the selected points probably are shifted to a grid in the lower figure. The front leg of the human model smoothly changes from the left leg to be the right leg in the lower figure from the top to down. Additionally, a walking movement can be seen in a counterclockwise direction in the lower figure.



(a): no regularization



(b): with regularization

Figure 34: AE-DMP trajectory in the feature space. The red line, extracted from the autoencoder, is the ground truth of the DMP, and the blue line is the DMP prediction of the trajectory. In the top figure, the DMP prediction without regularization is given. For the bottom two figures, the regularization term $\mathbf{f}$ is added.

Fig. 33 shows the data distribution in two of the five hidden neurons after the DMP component. The distribution is plotted in those two which have the two largest variances.

A movement of walking in feature space is illustrated in Fig. 34 for two hidden neurons; other hidden neurons render similar images. Hidden neuron 2 has the largest variance among all of the hidden neurons while hidden neuron 3 has the smallest. The DMP component in the AE-DMP is able to model the motions in the hidden space. The mean square error of the experiment in the joint space is $3.8 \cdot 10^{-4} \pm 1.2 \cdot 10^{-3}$ rad.

Fig. 34 demonstrates the effect of the regularization term $L(\mathbf{f}_i, \mathbf{f}_i^{target})$. In Fig. 34(a), the autoencoded input to the DMP is given (red line) as well as the output of the DMP, without using this regularization term. In addition, Fig. 34 shows the improved prediction when adding $L(\mathbf{f}_i, \mathbf{f}_i^{target})$.

A major strength of our method is caused by integrating the dimensionality reduction with the DMP-based movement representation. Other approaches, such as (Bitzer and Vijayakumar, 2009), separate these steps by first doing a dimensionality reduction and then representing the resulting latent space in DMPs. To demonstrate the advantage of our integrated approach we represented the 5 subjects with different movements (a) in our AE-DMP model, and (b) by first running an Autoencoder-based dimensionality reduction, and then using DMPs in the resulting latent space. The results are numerically reported in Table 2. The same movement results are shown in Fig. 36 with more details in every joint. For simple, smooth movement movements (in particular, balancing), the non-integrated model compares to the integrated AE-DMP. However, for movements, and esp. jerk movements such as kicking and complicated movement such as tai chi, show a considerable improvement of AE-DMP over the non-integrated model. In AE-DMP, the constraints that the DMP impose on the latent representation force the autoencoder to smoothen the hidden values, leading to much improved results. For instances, Fig. 35 illustrates the jerk trajectory of kicking in the hidden space using non-integrated model, but the trajectory is smooth by the integrated model.

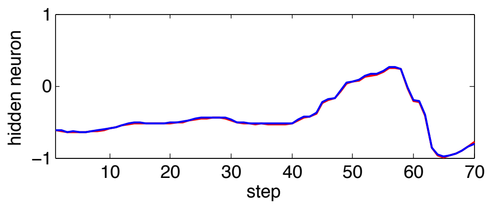
Compared with DMP and state of the art PCA dimensionality reduction for DMP, AE-DMP is more accurate as shown in Table 2. Similar as non-integrated method, both DMP and PCA-DMP have difficulty on jerk movement, and PCA-DMP lose some information of the movements during dimensionality reduction. For comparison, all the three models use the same parameters for DMP and PCA reduces dimensions to 5.

5.3.3 *New Motion Generation*

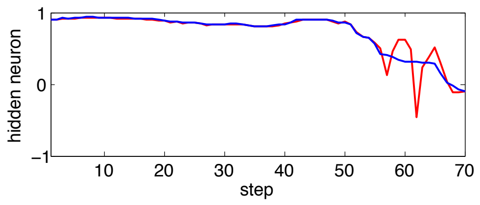
In this experiment, the walking movement and jogging movement are trained in one model. The training data includes 21 sequences of walking and 9 sequences of jogging.

Using sparse AE-DMP, there are 3 types of hidden values for multiple movements as shown in Fig. 37. By activating, deactivating or interpolating though hidden neuron 2, new movements are generated. As an example, Fig. 38 illustrates new movement generation from walking by activating the hidden neuron 2. The generated movement follows the walking demo in the terms of rhythm, while only the gesture becomes to jogging; therefore, the generated movement is slow jogging which has never shown in the training data. The interpolation of neuron 2 based on the walking movement has new motions between the middle and bottom of Fig. 38. Because we do not have ground truth of the new motion, it is not numerically evaluated.

The basic AE-DMP without sparsity can also generate or interpolate new motions from existing movements, but it is relative complicated because the hidden neurons which distinguish the motions may have



(a): Integrated AE-DMP. Other 4 trajectories of hidden neurons are as smooth as the illustrated trajectory.



(b): Non-integrated AE-DMP. 2 out of other 4 hidden neurons have jerk trajectories as the illustrated trajectory.

Figure 35: AE-DMP trajectory of kicking movement in the feature space. The red line, extracted from the autoencoder, is the ground truth of the DMP, and the blue line is the DMP prediction of the trajectory.

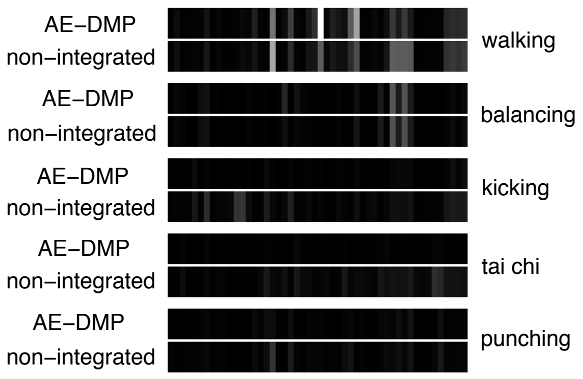


Figure 36: R-squared accuracy ($\sqrt{R^2}$). The R-squared value is indicated by a gray value, with black for 1 and white for 0. The horizontal axis represents 50 joints.

intersection. Compared with the basic AE-DMP, it simplifies the problem to single hidden neuron modification using sparsity.

Table 2: Error of the movement modeling, averaged over all joints over a whole movement. The error is MSE ± SD of the joint angle in radians. The AE-DMP is superior in all cases, but in particular for highly dynamic movements.

RMSE

movement	walking	balance	kicking	tai chi	punching
subject #	35	49	74	120	143
AE-DMP	$3.3 \cdot 10^{-4}$	$2.8 \cdot 10^{-4}$	$2.3 \cdot 10^{-4}$	$3.4 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$
non-integrated	$5.1 \cdot 10^{-4}$	$2.9 \cdot 10^{-4}$	$21 \cdot 10^{-4}$	$25 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$
DMP	$5.6 \cdot 10^{-4}$	$2.7 \cdot 10^{-4}$	$20 \cdot 10^{-4}$	$26 \cdot 10^{-4}$	$8.4 \cdot 10^{-4}$
PCA-DMP	$11 \cdot 10^{-4}$	$6.0 \cdot 10^{-4}$	$45 \cdot 10^{-4}$	$38 \cdot 10^{-4}$	$17 \cdot 10^{-4}$

SD

movement	walking	balance	kicking	tai chi	punching
subject #	35	49	74	120	143
AE-DMP	$1.2 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$	$0.93 \cdot 10^{-3}$
non-integrated	$2.4 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$	$30 \cdot 10^{-3}$	$12 \cdot 10^{-3}$	$1.7 \cdot 10^{-3}$
DMP	$2.5 \cdot 10^{-3}$	$1.8 \cdot 10^{-3}$	$27 \cdot 10^{-3}$	$13 \cdot 10^{-3}$	$4.4 \cdot 10^{-3}$
PCA-DMP	$4.0 \cdot 10^{-3}$	$2.4 \cdot 10^{-3}$	$28 \cdot 10^{-3}$	$14 \cdot 10^{-3}$	$57 \cdot 10^{-3}$

5.3.4 *Reconstruction for Missing Joints*

To illustrate the validity of our method, we demonstrate missing value imputation of missing joints and missing sub sequences. The DMP component is not used for joint reconstruction—this will be part of the next section. Denoising autoencoders have been shown to be generative models (Bengio et al., 2013): it turns out that corrupting an input and feeding it through the denoising autoencoder performs a step in a Markov chain, of which the equilibrium distribution is equal to the data generating distribution. This enables a principled missing value imputation method.

Let $\mathbf{x}_J$ denote the observed parts of the input $\mathbf{x}$; conversely, let $\mathbf{x}_{\bar{J}}$ denote the non-observed parts of the input. To find a reasonable imputation $\hat{\mathbf{x}}$, we can thus pass $\mathbf{x}_J \cup \mathbf{x}_{\bar{J}}$ through the autoencoder to obtain $\mathbf{x}_J^{(1)}$ and $\mathbf{x}_{\bar{J}}^{(1)}$. Here, the former is not corrupted, necessitating an adaption of the weights by a factor of $1-p$. $\mathbf{x}_J^{(1)}$ is henceforth discarded, and $\mathbf{x}_J \cup \mathbf{x}_{\bar{J}}^{(1)}$ passed through the network. This process is repeated to

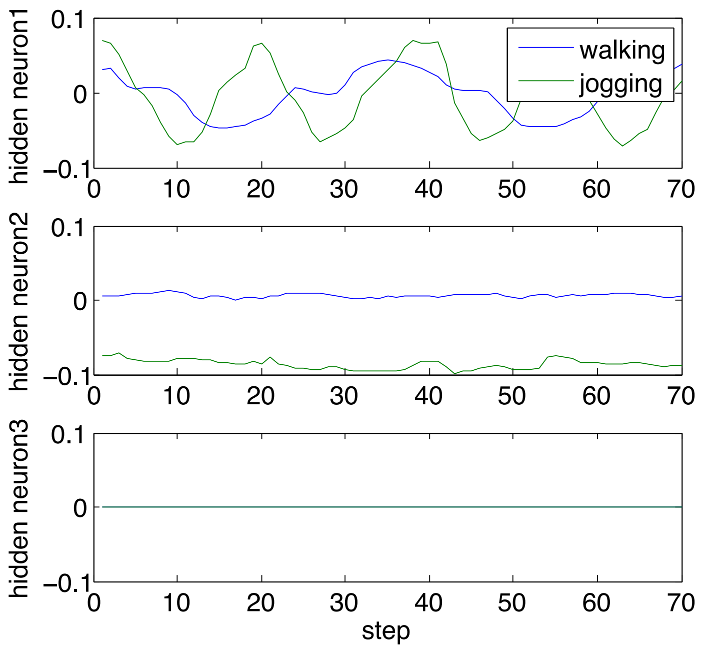


Figure 37: Feature representation of sparse AE-DMP. There are 3 kinds of hidden neurons for multiple movements. Both of walking and jogging are active as shown in the top figure, which encodes the information such as frequency of the movement; jogging is active while walking is not active as shown in the middle figure, which encodes the gesture of the movement; both of walking and jogging are not active as shown in the bottom figure.

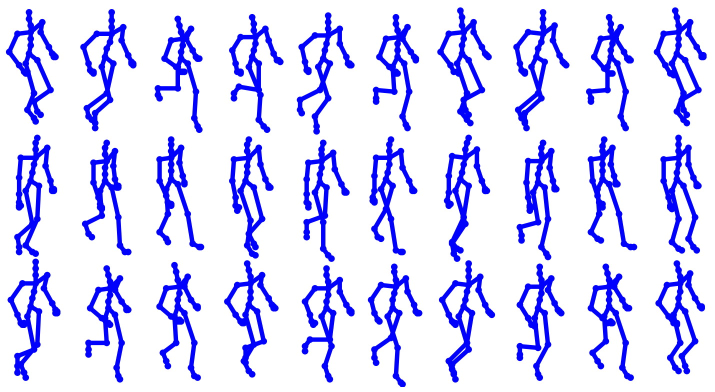


Figure 38: Novel motion generation. The top two rows are the training data of jogging and walking. The bottom row shows jogging generation from walking. It is a time sequence from left to right and plotted every 6 steps.

give $\mathbf{x}_{\bar{J}}^{(k)}$, which will be an unbiased sample of $p(\mathbf{x}_{\bar{J}}|\mathbf{x}_J)$. We then use $\hat{\mathbf{x}} = \mathbf{x}_J \cup \mathbf{x}_{\bar{J}}^{(k)}$ as the datum with imputed values. For our experiments, the chain converged after $k = 4$ steps. See Fig. 39 for an illustration.

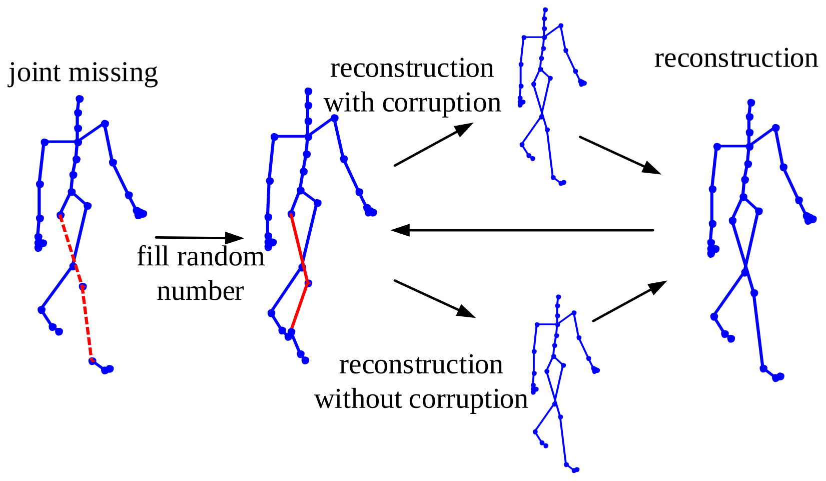

joint missing
fill random number
reconstruction with corruption
reconstruction without corruption
reconstruction

Figure 39: Missing joint reconstruction process. The red dashed lines are the ground truth of the missing joints with the corresponding limbs and the red solid lines are missing joints with corresponding limbs during reconstruction process.

In the experiments, we removed four right leg joints or four left arm joints out of fifty joints to generate two time sequences of testing data. Notably, a denoising autoencoder can reconstruct the missing joints in one frame without the time sequence information. The reconstruction error in joint space is shown in Table 3.

5.3.5 *Reconstruction for Missing Section*

We train the model as in Section 5.3.2, and the testing data has a demo of the first 10 frames while the last 60 frames are missing. Fig. 40 shows the reconstruction for the hip pitch and knee pitch of the right leg while other joints have similar plot. The error is $8.1 \cdot 10^{-4} \pm 3.4 \cdot 10^{-3}$ rad.

5.3.6 *Changing Goal Attractor*

AE-DMP maintains the generalization properties of DMP. Taking punching movement as an instance, Fig. 41 shows generalization with changing the attractor landscape. All of the start points are the same, while the testing movements are generalized with the shifted goals. The result is shown in two largest variance neurons.

In this punching movement, it does not require scaling properties or invariance properties. For other movements such as writing which

Table 3: Result of reconstruction for missing joints in joint space. The error is MSE ± SD of the joint angle in radians.

joint name	hip pitch	hip yaw	hip roll	knee pitch
MSE	$1.5{\cdot}10^{-3}$	$1.1{\cdot}10^{-3}$	$0.33{\cdot}10^{-3}$	$5.0{\cdot}10^{-3}$
SD	$2.0{\cdot}10^{-3}$	$1.3{\cdot}10^{-3}$	$0.50{\cdot}10^{-3}$	$8.5{\cdot}10^{-3}$

joint name	shoulder pitch	shoulder yaw	shoulder roll	elbow pitch
MSE	$0.74{\cdot}10^{-3}$	$0.73{\cdot}10^{-3}$	$0.38{\cdot}10^{-3}$	$2.7{\cdot}10^{-3}$
SD	$0.91{\cdot}10^{-3}$	$0.97{\cdot}10^{-3}$	$0.048{\cdot}10^{-3}$	$3.6{\cdot}10^{-3}$

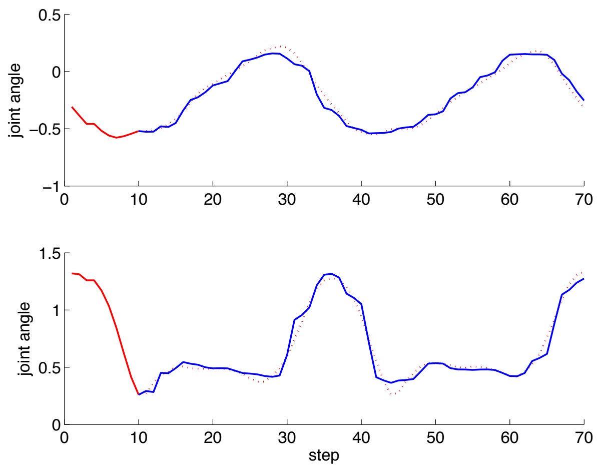


Figure 40: Filling data for missing section. The top and bottom figures are the hip pitch and knee pitch of the right leg respectively. The red solid line is the demo motion of testing data, the red dotted line is the truth of the missing section, and the blue solid line is the data filled using AE-DMP.

need invariance properties, the right side of (54) can be easily multiply by $(\mathbf{g} - \mathbf{y}_0)$ (Ijspeert et al., 2013).

5.4 CONCLUSIONS

In their standard formulation, DMPs suffer from suboptimal generalization—when used in configuration space—or are a victim of the curse of dimensionality—when used in task space. This is especially true for high-dimensional data sets, e.g. while representing

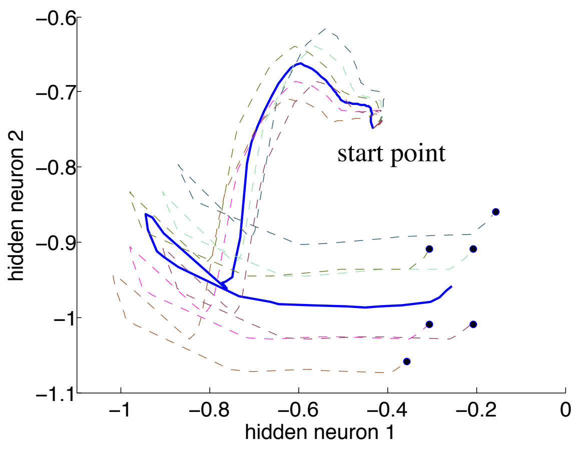


Figure 41: Changing goal attractor of punching movement in latent space. The solid line is the demonstration, the dashed line is the generalization movements with various goal attractors, and the black circles are the shifted goals.

the movement of human(oid) limbs. We present AE-DMP, a model that integrates DMPs with autoencoders to perform dimensionality reduction while optimally representing movement in latent feature space. The integration is shown to create a movement feature space in which DMPs can efficiently code and generalize movement.

To elaborate the connections and difference of various types of movements in feature space, we develop AE-DMP using sparsity. Sparse AE-DMP (SAE-DMP) can generate new movements which are not in the training data set by simply switching on/off or interpolating one hidden neuron. The new movement combines features of the existing two. For instance, slow jogging is generated based on walking and switching the movement type to jogging.

Experiments on a public motion capture database demonstrate that the model can accurately fill in corrupted data. The autoencoder facilitates the reconstruction of missing joints, while the DMP represents the dynamics to reconstruct missing movement sections.

However, generalization of a large amount of movement in a shared latent space still remains challenging. In subsequent work, the aim is to solve this issue by integrating a probabilistic model, variational autoencoder (VAE), with DMPs. Since VAE maximize the probabilistic independence in latent space, it will lead to a representation where the DMPs represent statistically independent parts of the movement, thus maximizing the generalizability.

6 DYNAMIC MOVEMENT PRIMITIVES IN LATENT SPACE OF TIME-DEPENDENT VARIATIONAL AUTOENCODERS

As can be seen in Chapter 5, the AE-DMP model benefits from representation of autoencoders and prediction of DMP. In the following work, we discuss modeling from the perspective of a probabilistic approach. DMPs are integrated with a more powerful representation tool, time-dependent variational autoencoders (VAE), to obtain a probabilistic version of VA-DMP. The model allows the use of DMP dynamics for sampling the latent variables in the VAE, thus steer the movement.

A shared latent representation for multiple movement types is then exploited, which is not implemented in the previous chapter. Also, the model has excellent generalization abilities such as switching between movements and changing goals. Additionally, when it reproduces the movement, optimal movements are generated.

6.1 INTRODUCTION

In earlier work Gaussian Process Latent Variable Models (GPLVM) (Lawrence and Moore, 2007; Bitzer and Vijayakumar, 2009) and denoising autoencoder (see Chapter 5) have been investigated for movement representation in latent space using DMP. These approaches efficiently reduced the data representation dimensions and enabled new movements generated by simply changing parameters in the latent space. However, these methods are only used to represent one, or at most two, different movement types; for more different movements, different models have to be trained. ProMP can combine and switch between different movements, and its dimension reduction model, dimension reduction ProMP (DR-ProMP) (Colome et al., 2014), was investigated for high-dimensional movements based on Expectation Maximization (EM). However, ProMP-based methods blend the movement via points which the trajectories go through, and require multiple demonstrations for each movement type to construct a sufficient probabilistic model.

In Chapter 4, we have shown that, in a reinforcement learning setting, variational autoencoders (VAE) can build more meaningful latent spaces than autoencoders or PCA. Following that line of thought, here we propose to use a VAE rolled out in time to reduce the dimensions for DMPs. To this end, we use a technique called *Deep Variational Bayes Filtering* (DVBF) (Karl et al., 2017). Our approach is an incremental advancement of DVBF and DMP by exploring DVBF to learn movements from multi-demonstrations, as well as benefiting from the latent representation of time-dependent VAE. Integration of DMPs with DVBF increases the constraints of the latent space, and therefore forces the distribution of the movements in the latent space to be meaningful. In our approach we train a model with a shared latent space for various similar as well as different movements. In contrast to ProMP, our method switches the movement completely, and does so by representing different movements in different “parts” of the latent space.

6.2 METHOD

Our approach consists of a modified DVBF which represents the high-dimensional data in a latent space and DMPs (see Section 2.2.1) which learns movements from demonstrations in the latent space. We focus on the discrete case, but an extension to rhythmic dynamical systems is straightforward by modifying the basis functions

6.2.1 *DMP in latent space*

The VAE, as formulized in the Section 2.1.5.3, has no internal state, and therefore cannot represent temporal dependencies in the input data. There are several ways of dealing with this; one possibility is extending the VAE to be a recurrent neural network (Bayer and Osendorfer, 2014; Chung et al., 2015). While having their merit in, e.g., anomaly detection (Sölch et al., 2016), their prediction capabilities are not as good as expected (Theis et al., 2016). Furthermore, it is not clear how a control signal can be included.

A different, very promising approach is obtained by rolling a VAE out in time. This approach, called *Deep Variational Bayes Filtering* (DVBF) was published in (Karl et al., 2017). In this paper we use DVBF to embed DMPs in VAE (see Fig. 42). By using this approach, we can directly learn all parameters—including the DMP parameters—using back-propagation.

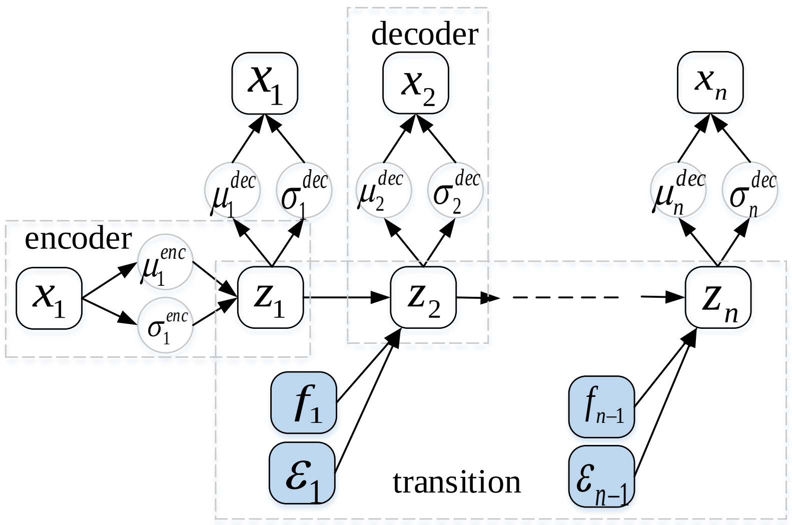


Figure 42: VAE-DMP information flow for the generative movement.

6.2.1.1 *Problem Formulation*

Given a movement $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where n is the length of the movement, we want to model the movement in latent space $\mathbf{z}_{1:n} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_n\}$. In order to embed DMP into DVBF, (53) is formulated for the transition function from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}$ in the latent space as

$$\begin{aligned}\tau\ddot{\mathbf{z}}_{t+1} &= \alpha\big(\beta(\mathbf{z}^{\text{goal}} - \mathbf{z}_t) - \dot{\mathbf{z}}_t\big) + \mathbf{f}_t + \epsilon \\ \dot{\mathbf{z}}_{t+1} &= \ddot{\mathbf{z}}_{t+1}dt + \dot{\mathbf{z}}_t \\ \mathbf{z}_{t+1} &= \dot{\mathbf{z}}_{t+1}dt + \mathbf{z}_t \end{aligned} \tag{74}$$

where dt is the step duration, $\mathbf{z}_t$ is the movement in latent space at step t, $\mathbf{z}^{\text{goal}}$ is the goal in the latent space corresponding to the final frame of movement in the joint space $\mathbf{x}_n$, $\mathbf{f}_t \in \mathbb{R}^{d'}$ is modified from (54) by changing the dimension to fit the latent space, $\epsilon = \hat{\epsilon}\mathbf{w}_\epsilon$, $\hat{\epsilon} \sim \mathcal{N}(0, \Sigma_\epsilon)$ is the system noise, and $\mathbf{w}_\epsilon \in \mathbb{R}^{d' \times d'}$ is the weight for the noise, so that the scale of the noise is learned autonomously.

Eq. (74) can be simplified as

$$\begin{bmatrix}\mathbf{z}_{t+1} \\ \dot{\mathbf{z}}_{t+1}\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{z}_t \\ \dot{\mathbf{z}}_t\end{bmatrix} + \mathbf{b}, \tag{75}$$

where the transition matrix is computed as

$$\mathbf{A} = \begin{bmatrix}-dt\alpha\beta\frac{1}{\tau} & -dt\alpha\frac{1}{\tau} + 1 \\ -\alpha\beta\frac{1}{\tau} & -\alpha\frac{1}{\tau}\end{bmatrix} dt + I, \tag{76}$$

and the control input is defined as

$$\mathbf{b} = \begin{bmatrix}dt \\ 1\end{bmatrix}(\alpha\beta\mathbf{z}^{\text{goal}} + \mathbf{f}_t + \hat{\epsilon})dt\frac{1}{\tau}. \tag{77}$$

I is an identity matrix.

Instead of generating $\mathbf{f}^{\text{target}}$ as (59) and (63) to find its parameters, we embed the DMP transition into DVBF and train its parameters through backpropagation.

6.2.1.2 *Inference model*

The encoder network is only used for the initial and goal states. We take a diagonal Gaussian distribution with mean μ_t and covariance Σ_t. The encoding process is $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mu_t, \text{diag}(\sigma_t^2))$.

The mean and covariance are encoded by a neural network

$$\begin{aligned}\mu_t^{\text{enc}} &= \mathbf{w}_\mu^{\text{enc}} h_\phi(\mathbf{x}_t) + \mathbf{b}_\mu^{\text{enc}},\\ \left(\sigma_t^{\text{enc}}\right)^2 &= \left(\mathbf{w}_\sigma^{\text{enc}} h_\phi(\mathbf{x}_t) + \mathbf{b}_\sigma^{\text{enc}}\right)^2,\end{aligned} \tag{78}$$

where, h_ϕ denotes an activation function. $\mathbf{w}_\mu^{\text{enc}}$, $\mathbf{w}_\sigma^{\text{enc}}$, $\mathbf{b}_\mu^{\text{enc}}$ and $\mathbf{b}_\sigma^{\text{enc}}$ are parameters. Based on a deep neural network, we parameterize the mean and variance of a Gaussian distribution.

We can get another representation of a latent space of the initial and goal frames by a transition function $\mathbf{z}_t^\star = g(\mathbf{z}_t)$. The transition function g can be, e.g., a multilayer perceptron (MLP). In the sequel, we will use $\mathbf{z}^\star$ in lieu of $\mathbf{z}$. The transition forces the Gaussian-shaped latent space to be replaced by any other kind of shapes.

During training, ϵ_{t+1} is predicted from $\mathbf{z}_t$ and $\mathbf{x}_{t+1}$, while ϵ is sampled from the prior without $\mathbf{z}_t$ or $\mathbf{x}_{t+1}$ for generating movement after training. ϵ is diagonal Gaussian distribution

$$\epsilon_{t+1} \sim p_\gamma(\epsilon_{t+1}|\mathbf{x}_{t+1}, \mathbf{z}_t) = \mathcal{N}(\mu_t^{\text{noise}}, \Sigma_t^{\text{noise}}), \tag{79}$$

where γ are the weights of neural network. The mean μ_t^{noise} and covariance Σ_t^{noise} are encoded by a neural network.

6.2.1.3 *Generative model*

We segment a movement into sub-sequence with length of l. The generative model includes decoders with $l = 1$

$$q_\phi(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mu_t^{\text{enc}}, \Sigma_t^{\text{enc}}), \tag{80}$$

and with $l > 1$

$$\hat{q}_{\hat{\phi}}(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{f}_t, \epsilon_t). \tag{81}$$

Consequently, the generated observation $\hat{\mathbf{x}}$ is

$$\hat{\mathbf{x}}_t \sim p_\theta(\mathbf{x}_t|\mathbf{z}_t) = \mathcal{N}(\mu_t^{\text{dec}}, \Sigma_t^{\text{dec}}). \tag{82}$$

(81) is the transition from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}$, which is described in the next subsection.

We take a Gaussian distribution with mean and constant covariance. Reconstruction ("decoding") of $\mathbf{x}$ is by another neural network,

$$\begin{aligned} \mu_t^{\mathrm{dec}} &= \mathbf{w}_\mu^{\mathrm{dec}} h_\theta(\mathbf{z}_t) + \mathbf{b}_\mu^{\mathrm{dec}}, \\ \sigma_t^{\mathrm{dec}^2} &= \mathbf{b}_\sigma^{\mathrm{dec}^2} \end{aligned} \tag{83}$$

where h_θ is the activation function. $\mathbf{w}_\mu^{\mathrm{dec}}$, $\mathbf{w}_\sigma^{\mathrm{dec}}$, $\mathbf{b}_\mu^{\mathrm{dec}}$ and $\mathbf{b}_\sigma^{\mathrm{dec}}$ are parameters.

Reconstruction through $\mathbf{z}$ from both (80) and (81) enforces the $\mathbf{z}_{t+1}$ from the latent transition to approximate the $\mathbf{z}_{t+1}$ from the encoder. Therefore, the training model contains both $\mathbf{z}$'s of subsequence length of 1, which allows $\mathbf{x}_t$ of every time step to be reconstructed from $\mathbf{z}_t$ of (80), and subsequence length of n.

6.2.1.4 *Transition model*

In the latent space, we predict the local transformation parameters of $\mathbf{f}$ and ϵ (see Fig. 43). The transition is described as

$$\mathbf{z}_{t+1} = f(\mathbf{z}_t, \mathbf{f}_t, \epsilon_t), \tag{84}$$

where f is a function of (75). ϵ is sample-specific noise. With meaningful transition priors, it avoids overfitting and has meaningful manifolds in the latent space.

The initial $\dot{\mathbf{z}}$ of the starting of the sequence can be predicted directly from the encoder with $\mathbf{x}_{1:m}$ as the input and $\{\mathbf{z}, \dot{\mathbf{z}}\}$ as the output, where $1 < m \leqslant n$. Besides, another approach initializes $\dot{\mathbf{z}}_0$ through the first step $\mathbf{z}_1$ and its previous step $\mathbf{z}_0$ using (74).

6.2.2 *Learning*

The learning process is through stochastic gradient variational Bayes.

Based on DVBF, the lower bound is rewritten as,

$$\begin{aligned} &\mathcal{L}(\mathbf{x}_{1:n}, \theta, \phi | \mathbf{f}_{1:n}) \\ &= \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x}_{1:n}|\mathbf{z}_{1:n}) - \log q_\phi(\epsilon_{1:n}|\mathbf{x}_{1:n}) + \log p(\epsilon_{1:n})\right] \quad (85) \\ &= \mathbb{E}_{q_\phi}\left[\log p_\theta(\mathbf{x}_{1:n}|\mathbf{z}_{1:n})\right] - \mathbb{KL}(q_\phi(\epsilon_{1:n}|\mathbf{x}_{1:n}) \| p(\epsilon_{1:n})), \quad (86) \end{aligned}$$

where $\epsilon_{1:n} = \{\epsilon_1, \epsilon_2, ..., \epsilon_n\}$ and $\mathbf{f}_{1:n} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_n\}$. $\mathbf{f}$ is encoded into $\mathbf{z}$ in (85).

During the training process, ϵ can act as a shortcut to encode all of the dynamical information; as a result, $\mathbf{f}$ is not learning meaningful val-

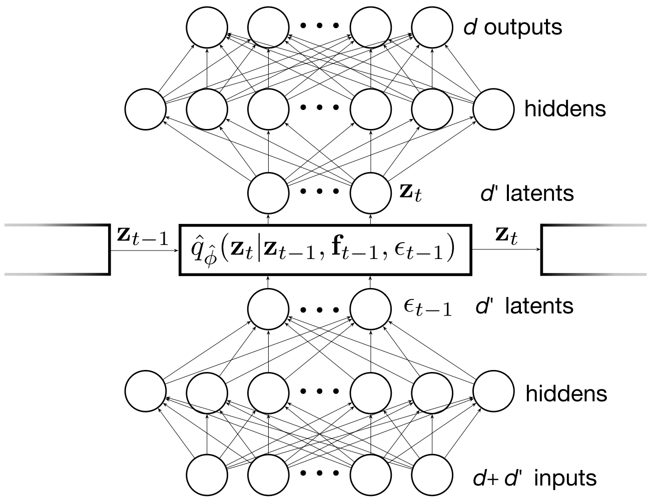


Figure 43: Neural network structure of the time-dependent VAE in time step t. The network structure is similar to a regular autoencoder with the addition of the DMP transition function in the center. The nodes ϵ_{t-1} are stochastic with mean and variance created by the neural network. The autoencoder takes $\mathbf{z}_{t-1}$ together with $\mathbf{x}_t$ as input.

ues. The annealing schedule improves the training of **f** and smooths out these local minima (Mandt et al., 2016). (85) is written as

$$\begin{aligned}\mathcal{L}(\mathbf{x}_{1:n}, \theta, \phi|\mathbf{f}_{1:n}) =& \mathbb{E}_{\mathbf{q}_\phi}\big[c_{t_a} \log p_\theta(\mathbf{x}_{1:n}|\mathbf{z}_{1:n}) \\ &- \log q_\phi(\epsilon_{1:n}|\mathbf{x}_{1:n}) + c_{t_a} \log p(\epsilon_{1:n})\big],\end{aligned} \tag{87}$$

where $c_{t_a} = \max(1, 0.01 + t_a/T_a)$ and t_a increases linearly from 0 after every training epoch until c_{t_a} equals 1.

Having only a small number of demonstrations increases the difficulty of training, since Monte Carlo estimation is used in (84) as in (Kingma and Welling, 2014). Additionally, a long sequence demonstration increases the training time and weakens the structure of the latent space at the start of the sequence, because of vanishing and exploding gradients. Therefore, we split a sequence up into overlapping subsequences to increase the batch size and shorten the sequence. On the contrary, the model has difficulties in recognizing the dynamics of the movement with too short subsequences, and consequently, the latent space might be unstructured. Thus, as a hyper-parameter, the length of the subsequences, l_{sub}, can be searched during training.

6.2.3 *Multi-demonstration model*

We extend VAE-DMP by training a single, shared latent space with multiple motion sequences. With multiple motion sequences (Wilson et al., 2007), the model (a) is not limited by the number of sequences, and (b) is more adaptable to a new movements, e.g., movement switching or goal changing.

Instead of learning the weights $\mathbf{w}$ for each type of movement individually we use a neural network which recognizes the correct weights from the observations. That way we get a continuous set of weights where the mapping between movement type and weights is trained by backpropagation. This neural network has the following shape:

$$\mathbf{w}(\mathbf{x}_{1:n}) = g_2(g_1(\mathbf{x}_1), g_1(\mathbf{x}_2), g_1(\mathbf{x}_3), \dots, g_1(\mathbf{x}_n)) \tag{88}$$

where g_2 and g_1 are each fully connected neural networks. We will call this whole network MLP_w.

6.2.4 *Movement switching*

The first generalization of the model is switching the movements after training by simply switching the weight $\mathbf{w}$ of $\mathbf{f}$ and the goal $\mathbf{z}_g$. We have $\eta^i \in [0, 1]$ for movement i. The weight and goal of the new movements at t step are

$$\mathbf{w}_t^{\star} = \sum_i \left(\frac{\eta_t^i \mathbf{w}_t^i}{\sum_i \eta_t^i} \right), \quad \mathbf{z}_{gt}^{\star} = \sum_i \left(\frac{\eta_t^i \mathbf{z}_{gt}^i}{\sum_i \eta_t^i} \right). \tag{89}$$

Switching from movement A to movement B, η^A is changed from 1 to 0, and η^B from 0 to 1. Given the starting point and the duration of switching, η^A and η^B can be estimated.

Goal changing enables the latent value to transfer from movement A to movement B, while the force term enables the movement to follow the demonstration trajectory B after switching. If we only change the goal but not the force term, the movement also switches from A to B; however, the trajectory is quite random after switching.

6.2.5 *Goal changing*

The flexible properties of DMPs are retained in VAE-DMP. For instance, it is able to reach new set points by simply changing the goal after training, while keeping the invariant properties. Following Ijspeert et al. (2013), to keep the invariant properties, an extra term of $(\mathbf{g} - \mathbf{y_0})$ is multiplied to the right side of (54). In addition, it can be multiplied by $(\mathbf{g} - \mathbf{y_0})/(|\mathbf{g}_{\text{fit}} - \mathbf{y}_{0,\text{fit}}| + \eta)$ instead to avoid coinciding $\mathbf{y}_0$ and $\mathbf{g}$, where fit represents the training data and η is a very small positive constant.

Other reformulations of DMP, e.g. to do obstacle avoidance (Hoffmann et al., 2009), can be implemented in VAE-DMP straightforwardly, but here we only focus on goal changing.

6.3 EXPERIMENTS

Experiments with two data sets are performed to evaluate VAE-DMP. The first data set contains optical tracking data of human movements. The second data set is obtained from 6-DoF robot arm simulations.

6.3.1 *High-dimensional human movement*

The data set for the first set of experiments is the CMU Graphics Lab Motion Capture Database, which is a part of the *KIT Whole-Body Human Motion Database*[1]. These data are downsampled and preprocessed as described in (Chen et al., 2015), and the resulting data consist of multiple movements, each of which is coded in 70 time steps of 50-dimensional vectors. The data is normalized to zero mean for every joint. Since we focus on learning relative movements, the body center is fixed. We split the movements up into subsequences with $l_{sub} = 10$ time steps.

The DMPs are set to critical control by setting $\beta = \alpha/4$ (Ijspeert et al., 2013). The VAE architecture is 5 layers with d inputs, 200 hidden neurons, d' latents, 200 hidden neurons, d outputs, where rectifier and identity activations are used for the hidden layers and the output layer, respectively. The structure of the VAE is shown in Fig. 43. The neural network g_2 inside the MLP$_w$ network has $70 \times h$ inputs and $50 \times d'$ outputs. The g_1 network takes 50 inputs and outputs h. Where h is 10 for the humanoid experiments and 5 for the robot experiment. Both g_1 and g_2 have no hidden layers. g_1 uses softmax as activation function and g_2 uses the identity activation function. The structure and sizes of the MLP$_w$ network can be seen in Fig. 44. The hyper-parameters of DMP and VAE are chosen based on the reconstruction error and the training time by grid search.

6.3.1.1 *Learning different movements*

In this task we train multiple demonstrations of walking, kicking, taichi and punching from 5 subjects (viz. subject 35, 74, 49, 120, and 143), all in one and the same model, and compare the results when we train each movement type to a single, specialised model (called "VAE-DMP single"). We also compare the results to our previous "AE-DMP single" (Chen et al., 2015), in which we train one movement per autoencoder. Fig. 45 shows the pose reconstruction error of our previous work of AE-DMP single with 5D latent space, VAE-DMP with 2D, 3D, 5D and 7D latent space, and VAE-DMP single with 5D latent space. For VAE-DMP single we used 120 rather than 200 hidden

1 https://motion-database.humanoids.kit.edu

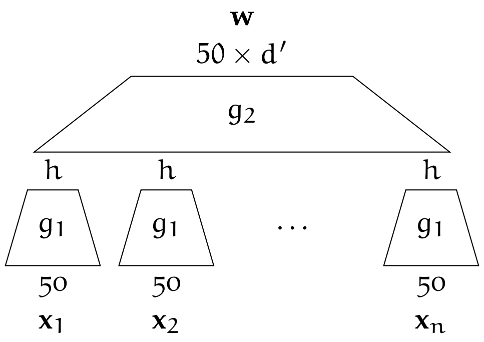


Figure 44: Structure of MLP$_w$ with input and output sizes. The g_1 networks take the 50-dimensional input $\mathbf{x}_t$ and has an output with h dimensions. g_2 has $70 \times h$ inputs and $50 \times d'$ outputs. h is 10 for the humanoid experiments and 5 for the robot experiments. d' is the number of latent dimensions.

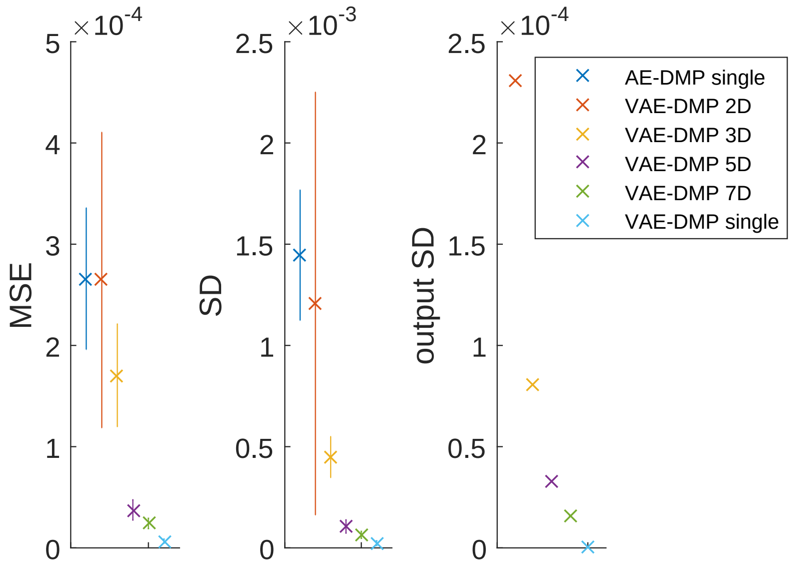


Figure 45: Error of the reconstruction of the movement modeling. The error is MSE ± SD of every single subject averaged over all joints over a whole movement in radians. The result is evaluated using Mean square error (MSE ± standard deviation). The left and middle figures show MSE and SD averaged over 5 subjects. The right figure is the mean output SD (directly predicted from the encoder) of the VAE-DMP reconstruction. It is a constant value for every joint during the whole movement, so that output SD of VAE-DMP 2D, 3D, 5D, and 7D do not have variance.

units in each hidden layer. Noticeable is the improvement over our previous model, AE-DMP single. When sufficient latent variables are chosen (in this case, 5 or 7), the error is much lower, even for the approach where all poses are represented in one model. The lowest reconstruction error is obtained in VAE-DMP single, with 7D VAE-DMP a close second.

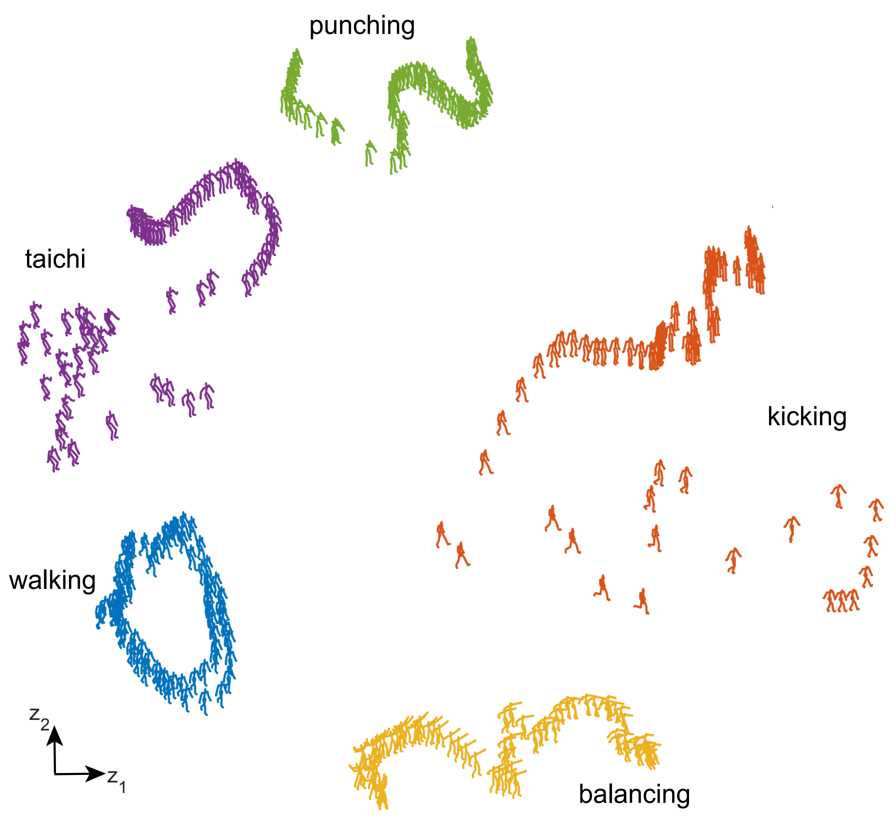
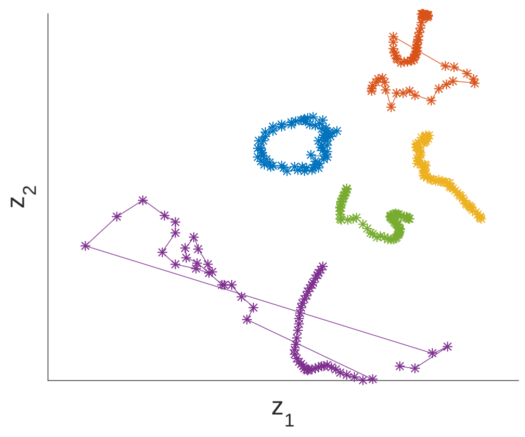


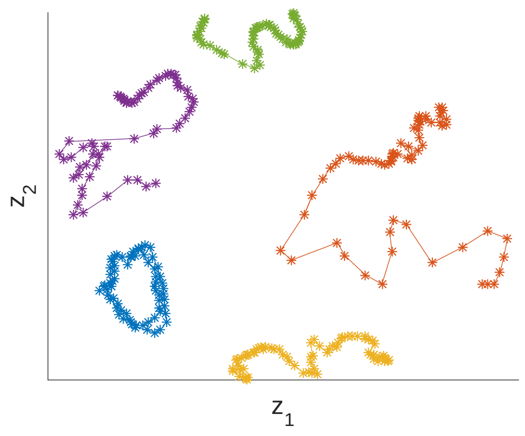
Figure 46: Movement distribution in 2D latent space. z_1 and z_2 are two latent dimensions, and every body pose in the joint space is generated from its corresponding latent state.

Although the reconstruction accuracy of VAE-DMP with 2D latent space yields to that with 7D latent space, we can more easily plot the former. Fig. 46 therefore illustrates the distribution of five movements in the latent space of VAE-DMP 2D. The patterns of the various generative movements can be seen. The walking is periodic, while punching is a single line in latent space. Kicking is a large-range movement, so that it has large range in the latent space, while balancing only has relative slight movement, and the range in the latent space is small.

The latent space of VAE-DMP is more meaningful than that of VAE (see Fig. 47). In the VAE latent space, a sequence of movement may spread far with different spacings for complex movements such as taichi, since it does not encode time information into the latent space. Accordingly, big gaps between two time steps may cause difficulties for DMPs. In addition, a large geometry distance in joint space may result in a small distance in the VAE latent space such as kicking. With correctly encoding the geometry distance from the joint space, VAE-DMP can improve the multi-demonstration model ability compared with VAE.



(a) VAE



(b) VAE-DMP

Figure 47: Trajectories of five human movements in the 2D latent space using VAE and VAE-DMP. The movements are colored the same as Fig. 46

Training with additional 4 walking movements from other subjects (viz. Subject 86, 91, 114 and 139), we have the results shown in Fig. 48; in this case, a VAE-DMP 3D is used. The five walking movements distribute in a cluster, disjunct from the other movements. The leg movements cluster approximately on the positive of the z_3 axis, while the arm movements cluster on the negative of the z_3 axis.

6.3.1.2 *Latent space smoothness*

The latent space of a VAE-DMP codes the learned movements in a compact, multivariate Gaussian space. As seen in Fig. 46, the demonstration trajectories do not span the whole latent space. But what movements are coded between the demonstrated trajectories? To verify that no discontinuities or “large jumps” occur when sampling a trajectory in latent space, we evaluate dx/dz over the whole latent

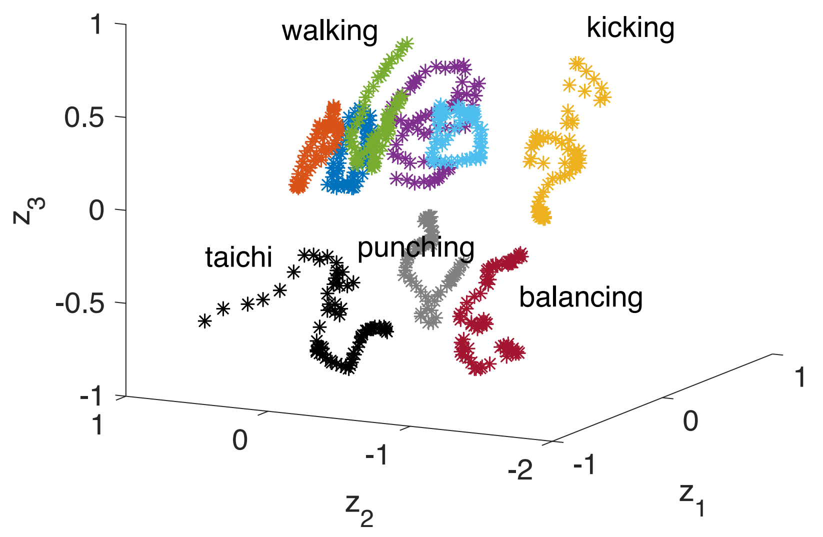


Figure 48: Nine movements represented in the 3D latent space of a single VAE-DMP.

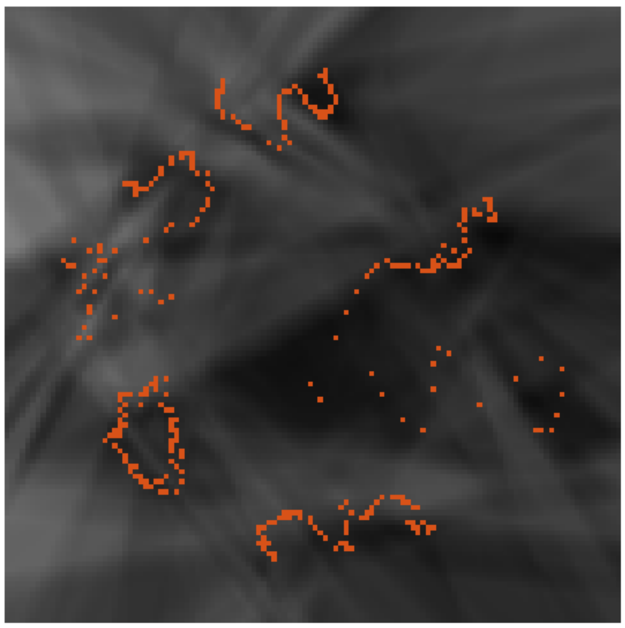


Figure 49: Smoothness of the latent space. The plot represents values of dx/dz, which vary between 4.7 and 7.4 for a 2D VAE-DMP. Higher values—indicated by lighter areas—mean that a step in latent space corresponds to a larger body movement in x. The colored points correspond to the demonstrated movements, as in Fig. 46. The data are evaluated at 120×120 grid points in the 2D latent space.

space of a VAE-DMP. The result for a VAE-DMP 2D is shown in Fig. 49. In the whole latent space, dx/dz varies between 4.7 and 7.4 and is indeed very smooth. Similar smooth latent spaces are seen in VAE-DMP 3D and VAE-DMP 5D.

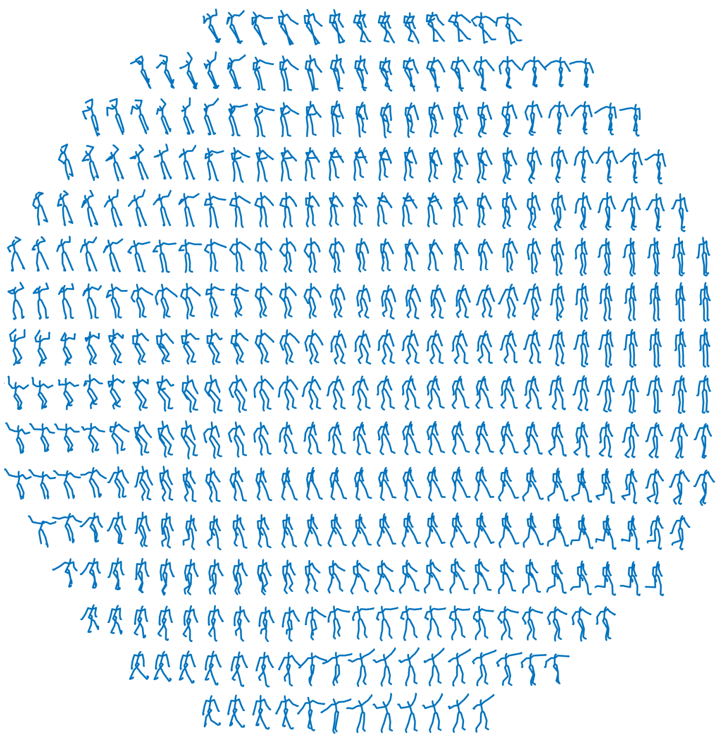
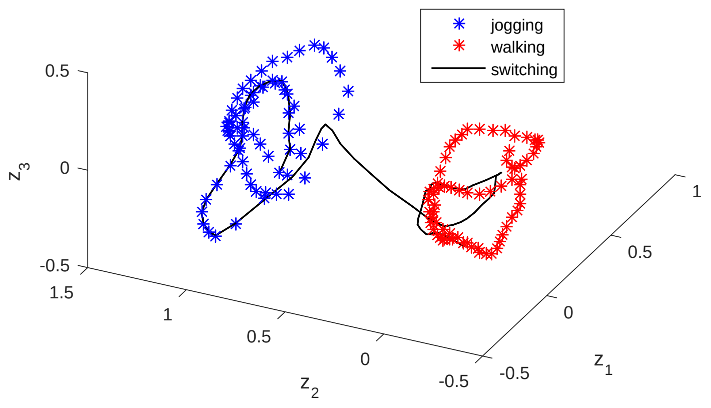


Figure 50: Body postures in the 2D latent space, in the space spanned by the multivariate Gaussian. The area covering a variance of $\sigma = 1$ is plotted.

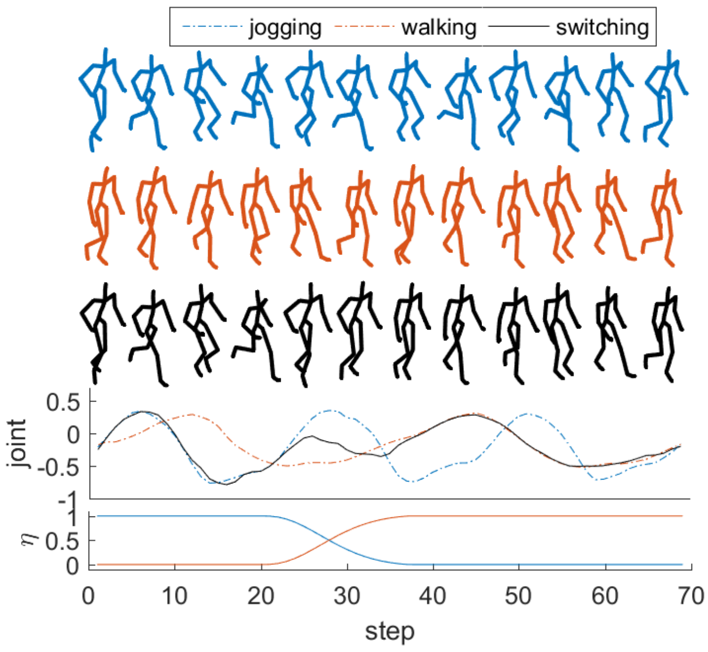
The joint space movements generated in 2D latent space is illustrated in Fig. 50. As $\mathbf{z}$ represents a multivariate normal $(\mu = 0, \sigma = 1)$ distribution, we can immediately compute the probability of a certain movement from its $\mathbf{z}$. As the plot shows, the movements between the demonstrated movements are smoothly interpolated. But also movements beyond that represent viable positions. Only when we move far away from the mean, starting between 2σ and 3σ does the latent space represent nonsensical postures. Of course, we can increase the training data set to obtain a larger confidence interval.

6.3.1.3 *Movement switching*

A model is trained by multiple demonstrations of a walking and jogging. $i = 1$ and $i = 2$ represent jogging and walking in η^i, respectively. The movements are switched from step 20 until step 38 (see Fig. 51). The starting and the duration of the switching can be changed. The model has a 5D latent space, and the largest two variance of the latent space are plotted. After it adapts to the walking from jogging, it follows the demonstration trajectory of walking. The generated latent values are not necessary to reproduce the latent values of demonstration precisely, while the reconstruction ability is more significant.



(a) Movement switching in latent space.



(b) Movement switching observed in state space. The figures of top three rows are plotted by every 6 time steps. We choose left hip for representing the joint, since it exactly follows the walking and jogging rhythm.

Figure 51: Movement switching from jogging to walking.

6.3.2 *Robot simulation for goal changing*

In this experiment we simulate a 6-DoF KUKA robot using the Robotic Toolbox (Corke, 2011). In the data set, the length of a demonstration is 76 steps. The subsequence length is 10 time steps. For the representation of the movement in the VAE, we use 5 layers with d, 100, $d' = 2$, 100, d. The input dimension of g_1 is d and the output dimension h is 5. g_2 takes $n \times h$ inputs and has $50 \times d'$ outputs.

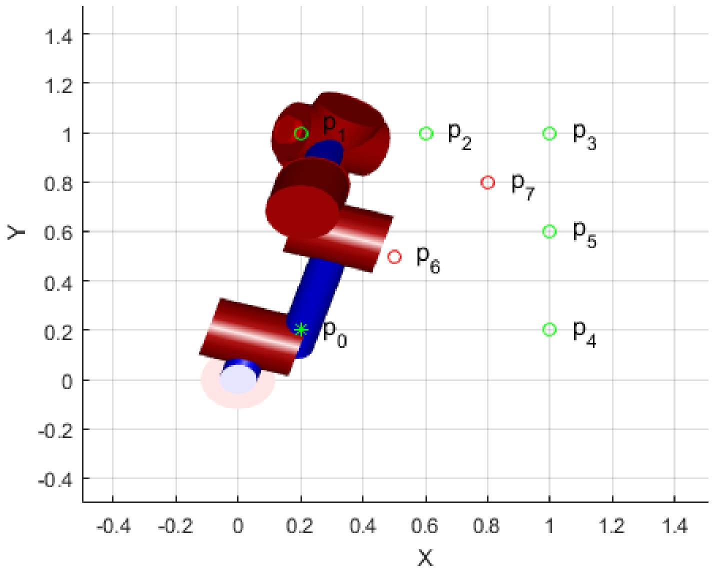


Figure 52: Robot data set.

The demonstration $\mathbf{x}^i$ is generated by moving the end-effector linearly from a starting point $\mathbf{p}_0$ to subsequent points $\mathbf{p}_i$, $i \in \{1, 2, \ldots, 5\}$ in Cartesian space. $\mathbf{x}^1$ is the demonstration for DMP, while $\{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^5\}$ are the demonstrations for VAE-DMP (see Fig. 52). $\mathbf{p}_6$ and $\mathbf{p}_7$ are the results of changing the goal to $[x = 0.5, y = 0.5]$ and $[x = 0.8, y = 0.8]$, respectively using VAE-DMP with given the force $\mathbf{f}^1$. All movements were planar, keeping z constant at 0.3.

For method comparison, we use the same version of DMP for both VAE-DMP and DMP. As described in Sec. 6.2.5, to keep the invariant properties (Ijspeert et al., 2013) for changing the goal, both forcing terms $\mathbf{f}$ in VAE-DMP and DMP are multiplied by a scaling term, specifically, $(\mathbf{z}^{\text{goal}} - \mathbf{z}_0)$ for VAE-DMP cq. $(\mathbf{g} - \mathbf{y_o})$ for DMP.

Fig. 53 shows the results of both DMP and VAE-DMP when changing the goal to [0.8, 0.8]. The optimal trajectory is the movement of the new goal with the end-effector moving linearly from $\mathbf{p}_0$. The optimal trajectory is not shown in the training data set but only its final frame is given as the new goal. It can be seen that VAE-DMP is able to generate a movement which is close to the optimal trajectory for the goal changing. However, DMP can only keep invariance of the demonstration for every joint independently, the effect of which is most clearly seen in joints 2 and 4. In this case, the robot joints are highly possible to be out of range. In contrast, VAE-DMP correlates the joints. In this experiment, VAE-DMP learns the invariance of the end-effector instead of every single joint. DMP requires manual selection of the important feature (e.g., joint angles of a robot or Cartesian space of the end effector) to learn, but VAE-DMP is able to learn the optimal trajectory autonomously and reproduces natural movements.

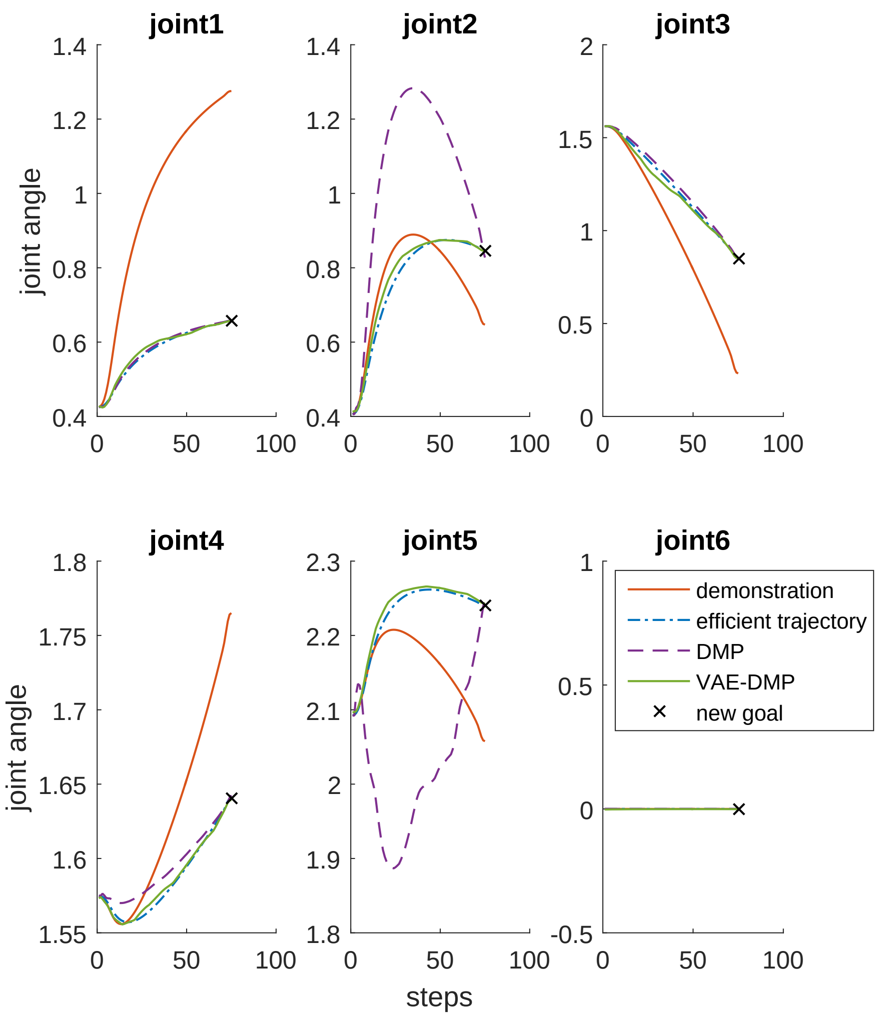


Figure 53: Results of goal changing.

In joint 5, the start and end angle are almost the same in the demonstration. This makes generalization with DMP difficult, as the force term is almost zero. In VAE-DMP, the joints are correlated in the latent space, so that the beginning and ending values are not the same.

6.4 CONCLUSIONS

In this chapter, we presented a novel approach to embed DMPs into a time-dependent variational autoencoder. Using a recently published approach called *Deep Variational Bayes Filtering*, we can embed DMPs in the latent space of a variational autoencoder, while simultaneously representing a large range of different movements in one single DVBF network. Thus we can also create smooth transitions between different movements. While DMPs can only independently keep the invariance per single joint, our approach allows the model to unsupervisedly learn the invariance features of the trajectory.

Part V

CONCLUSION

7 CONCLUSIONS AND OUTLOOK

This thesis has presented GP-based and deep-learning based approaches to address movement representation and time series prediction issues. Some crucial challenges encountered when applying the existing work regarding to both of the applications and the machine learning methods are: (1) how to robustly represent the movements without strictly restricting environments; (2) how to solve the computation issues for high dimensional large data; (3) how to learn the features from high dimensional observation space; (4) how to efficiently represent large amount of movements. In this thesis, algorithms were developed to address these issues as follows.

First of all, an approach was developed, using the deformation and color distribution of the fingernail and its surrounding skin, to estimate the fingertip forces, torques and contact surface curvatures for various objects, including the shape and material of the contact surfaces and the weight of the objects. In addition, compared with previous single finger estimation in an experimental environment, we extend the approach to multiple finger force estimation, which can be used for applications such as human grasping analysis. In experiments, the performance of dynamic approaches yielded to that of static approaches with regard to the experiments that involved finger force detection, because of the low frequency of the image data. The low frequency allows the data to be smooth, which does not require time series methods such as RNN.

Secondly, designing controllers that take high-dimensional feedback, such as tactile and visual data, into account is non-trivial. Therefore, robots should be able to learn tactile skills through trial and error by using reinforcement learning algorithms. The input domain for such tasks, however, might include strongly correlated or non-relevant dimensions, making it hard to specify a suitable metric on such domains. Auto-encoders specialize in finding compact representations, where defining such a metric is likely to be easier. Therefore, a reinforcement learning algorithm was developed, that can learn non-linear policies in continuous state spaces, which leverage representations learned using auto-encoders. The static variational autoencoder was modified to reproduce the system dynamics, rather than encode individual input patterns, which tended to improve reinforcement learning performance. Re-training the auto-encoders on the state distribution induced by the policy markedly improved performance.

Thirdly, DMP has been shown to be a powerful method of representing movements, but does not generalize well when used in configuration or task space. To solve this problem we developed a model called autoencoded dynamic movement primitive (AEDMP) which uses deep autoencoders to find a representation of movement in a latent space, in which DMP can optimally generalize new movements. The architecture embeds DMP into such an autoencoder and allows the whole model to be trained as a unit. To further improve the model for multiple movements, sparsity was added for the feature layer neurons; therefore, various movements can be observed clearly in the latent space. After training, the model finds a single hidden neuron from the sparsity that can efficiently generate new movements. Experiments clearly demonstrate the efficiency of missing data imputation using 50-dimensional human movement data.

Finally, an alternative method was explored, that embeds DMPs into the latent space of a time-dependent variational autoencoder framework. The method enables the representation of high dimensional movements in a low-dimensional latent space. Experimental results show that this framework has excellent generalization in the latent space, e.g., switching between movements or changing goals. It represents a large range of different movements in one single time-dependent network.

BIBLIOGRAPHY

B Allen, B Curless, and Z Popovic. The space of human body shapes: Reconstruction and parameterization from range scans. In *SIGGRAPH*. Addison-Wesley, 2003. (Cited on page 5.)

Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997. (Cited on page 32.)

Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Advances in Variational Inference*, 2014. (Cited on pages 9, 74, and 88.)

Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. In *International Conference on Learning Representations (ICLR)*, 2014. (Cited on page 42.)

Yoshua Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009. ISSN 1935-8237. (Cited on pages 3, 22, 65, and 75.)

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems (NIPS)*, pages 899–907, 2013. (Cited on page 82.)

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738. (Cited on pages 5, 15, and 24.)

Sebastian Bitzer and Sethu Vijayakumar. Latent spaces for dynamic movement primitives. In *International Conference on Humanoid Robots (HUMANOIDS)*, 2009. (Cited on pages 7, 74, 79, and 87.)

Sebastian Bitzer, Ioannis Havoutis, and Sethu Vijayakumar. Synthesising novel movements through latent space modulatoin of scalable control policies. In *International Conference on Simulation of Adaptive Behaviour (SAB)*, 2008. (Cited on pages 7 and 77.)

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning (ICML)*, pages 1613–1622, 2015. (Cited on page 18.)

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *International Conference on Machine Learning (ICML)*, pages 1159–1166, 2012. (Cited on pages 8 and 74.)

Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4):291–294, 1988. (Cited on pages 3, 22, 65, and 75.)

Wray L Buntine and Andreas S Weigend. Bayesian back-propagation. *Complex systems*, 5(6):603–643, 1991. (Cited on page 18.)

S. Calinon, Z. Li, T. Alizadeh, N. G. Tsagarakis, and D. G. Caldwell. Statistical dynamical systems for skills acquisition in humanoids. In *International Conference on Humanoid Robots (Humanoids)*, Osaka, Japan, 2012. (Cited on page 75.)

Y. Chebotar, O. Kroemer, and J. Peters. Learning robot tactile sensing for object manipulation. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 3368–3375, Sept 2014. (Cited on page 64.)

Nutan Chen, Göran Westling, Benoni B. Edin, and Patrick van der Smagt. Estimating fingertip forces, torques, and local curvatures from fingernail images. *submitted*.

Nutan Chen, Chee-Meng Chew, Keng Peng Tee, and Boon Siew Han. Human-aided robotic grasping. In *International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 75–80, 2012. (Cited on page 5.)

Nutan Chen, Sebastian Urban, Christian Osendorfer, Justin Bayer, and Patrick van der Smagt. Estimating finger grip force from an image of the hand using convolutional neural networks and Gaussian processes. In *International Conference on Robotics and Automation (ICRA)*, 2014. (Cited on page 37.)

Nutan Chen, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. Efficient movement representation by embedding dynamic movement primitives in deep autoencoders. In *International Conference on Humanoid Robots (HUMANOIDS)*, pages 434–440, 2015. (Cited on page 94.)

Nutan Chen, Sebastian Urban, Justin Bayer, and Patrick van der Smagt. Measuring fingertip forces from camera images for random finger poses. In *International Conference on Intelligent Robots and Systems (IROS)*, 2015.

Nutan Chen, Maximilian Karl, and Patrick van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Neurorobotics*, 2016a.

Nutan Chen, Maximilian Karl, and Patrick van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *International Conference on Humanoid Robots (HUMANOIDS)*, 2016b.

Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models based on learned skills. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Robot Learning*, 2017a.

Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models based on learned skills. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Workshop on Acting and Interacting in the Real World: Challenges in Robot Learning*, 2017b.

Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick van der Smagt. Metrics for deep generative models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018a.

Nutan Chen, Alexej Klushyn, Alexandros Paraschos, Djalel Benbouzid, and Patrick van der Smagt. Active learning based on data uncertainty and model sensitivity. *submitted*, 2018b.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. (Cited on page 22.)

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems (NIPS)*, pages 2980–2988, 2015. (Cited on pages 9 and 88.)

A. Colome, G. Neumann, J. Peters, and C. Torras. Dimensionality reduction for probabilistic movement primitives. In *International Conference on Humanoid Robots (HUMANOIDS)*, 2014. (Cited on page 87.)

Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Real-time tracking of non-rigid objects using mean shift*, pages 142–149, 2000. (Cited on page 37.)

Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, 2011. ISBN 978-3-642-20143-1. (Cited on page 100.)

John Darby, Baihua Li, Nicholas Costen, David J. Fleet, and Neil D. Lawrence. Backing off: Hierarchical decomposition of activity for 3d novel pose recovery. In *British Machine Vision Conference*, pages 1–11, 2009. (Cited on page 7.)

Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2 (1–2):1–142, 2013. (Cited on page 28.)

Stefan Depeweg, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *CoRR*, 2016. (Cited on page 18.)

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2): 179–211, 1990. (Cited on page 21.)

Anthony C Fang and Nancy S Pollard. Efficient synthesis of physically valid human motion. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 417–426, 2003. (Cited on page 5.)

Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *The journal of Neuroscience*, 5(7):1688–1703, 1985. (Cited on page 73.)

Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *International Conference on Computer Vision*, pages 4346–4354, 2015. (Cited on page 8.)

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, PhD thesis, University of Cambridge, 2016. (Cited on page 18.)

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015. (Cited on page 18.)

Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *International Conference on Learning Representations (ICLR)*, 2014. (Cited on page 20.)

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. (Cited on page 5.)

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011. (Cited on page 18.)

T. Grieve, Yu Sun, J.M. Hollerbach, and S.A. Mascaro. 3-d force control on the human fingerpad using a magnetic levitation device for fingernail imaging calibration. In *Joint EuroHaptics conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2009. Third Joint*, pages 411–416, 2009. (Cited on page 36.)

T. Grieve, L. Lincoln, Yu Sun, J.M. Hollerbach, and S.A. Mascaro. 3d force prediction using fingernail imaging with automated calibration. In *IEEE Haptics Symposium*, pages 113–120, 2010. (Cited on page 44.)

Thomas R. Grieve, John M. Hollerbach, and Stephen A. Mascaro. Force prediction by fingernail imaging using active appearance models. In *World Haptics Conference (WHC)*, pages 181–186, 2013. (Cited on pages 36 and 44.)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. (Cited on pages 4 and 20.)

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *ICML*, pages 1861–1869, 2015. (Cited on page 18.)

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM, 1993. (Cited on page 18.)

Sepp Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen*. PhD thesis, diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen, 1991. (Cited on page 21.)

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (Cited on page 22.)

Heiko Hoffmann, Peter Pastor, Dae-Hyung Park, and Stefan Schaal. Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In *International Conference on Robotics and Automation (ICRA)*, pages 2587–2592, 2009. (Cited on page 93.)

Rachel Hornung, Nutan Chen, and Patrick van der Smagt. Multimodal motion modeling. *Handbook of Multimodal-Multisensor Interfaces (in print)*.

Takeo Igarashi, Tomer Moscovich, and John F Hughes. As-rigid-as-possible shape manipulation. In *ACM transactions on Graphics (TOG)*, volume 24, pages 1134–1141, 2005. (Cited on page 5.)

Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1398–1403, 2002. (Cited on page 73.)

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Comput.*, 25(2):328–373, 2013. (Cited on pages 31, 32, 77, 85, 93, 94, and 101.)

Roland S Johansson and J Randall Flanagan. Sensorimotor control of manipulation. In *Encyclopedia of Neuroscience*, number 8 in Encyclopedia of Neuroscience, pages 593–604. Elsevier, 8 edition, 2009. (Cited on pages 35, 57, and 58.)

Michael I Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997. (Cited on page 21.)

M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal. Learning force control policies for compliant manipulation. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2011. (Cited on page 64.)

Takeo Kanade and PJ Narayanan. Virtualized reality: perspectives on 4d digitization of dynamic events. *IEEE Computer Graphics and Applications*, 27(3), 2007. (Cited on page 5.)

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational Bayes filters: Unsupervised learning of state space models from raw data. *International Conference on Learning Representations (ICLR)*, 2017. (Cited on pages 9 and 88.)

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *International Conference on Learning Representations (ICLR)*, 2014. (Cited on pages 9, 25, 26, and 92.)

Jens Kober, Betty Mohler, and Jan Peters. Learning perceptual coupling for motor primitives. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 834–839, 2008. (Cited on page 64.)

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *ACM transactions on graphics (TOG)*, volume 21, pages 473–482, 2002. (Cited on page 6.)

R. G. Krishnan, U. Shalit, and D. Sontag. Deep Kalman filters. In *Advances in Neural Information Processing Systems (NIPS) Workshop on Advances in Variational Inference*, 2015. (Cited on page 65.)

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. (Cited on page 20.)

O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robot. and Auton. Syst.*, (9):1105–1116, 2010. (Cited on page 64.)

O. Kroemer, C. Daniel, G Neumann, H. van Hoof, and J. Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015. (Cited on page 64.)

T. Lampe and M. Riedmiller. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2013. (Cited on page 63.)

Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010. (Cited on pages 8 and 64.)

Neil Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *J. Mach. Learn. Res.*, 6:1783–1816, December 2005. ISSN 1532-4435. (Cited on pages 3 and 6.)

Neil D Lawrence and Andrew J Moore. Hierarchical gaussian process latent variable models. In *Proceedings of the 24th international conference on Machine learning*, pages 481–488. ACM, 2007. (Cited on page 87.)

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proc IEEE*, volume 86, pages 2278–2324, 1998. (Cited on pages 3, 7, and 19.)

Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 253–256, 2010. (Cited on page 19.)

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. (Cited on page 4.)

Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015. (Cited on page 31.)

Deng Li and Yu Dong. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014. (Cited on page 5.)

Matt D Luciw, Ewa Jarocka, and Benoni B Edin. Multi-channel EEG recordings during 3,936 grasp and lift trials with varying weight and friction. *Scientific Data*, 1:1440047, 2014. (Cited on page 43.)

David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992. (Cited on page 18.)

Guilherme J Maeda, Gerhard Neumann, Marco Ewerton, Rudolf Lioutikov, Oliver Kroemer, and Jan Peters. Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks. *Autonomous Robots*, pages 1–20, 2016. (Cited on page 31.)

Alireza Makhzani and Brendan J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013. (Cited on page 8.)

Stephan Mandt, James McInerney, Farhan Abrol, Rajesh Ranganath, and David Blei. Variational tempering. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 704–712, 2016. (Cited on page 92.)

J. Mattner, S. Lange, and M. Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In *International Conference on Neural Information Processing (NIPS)*, pages 126–133, 2012. (Cited on pages 8 and 64.)

Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC Press, 1999. (Cited on page 4.)

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012. (Cited on page 5.)

Andriy Myronenko and Xubo B. Song. Intensity-based image registration by minimizing residual complexity. *IEEE Trans. Med. Imaging*, 29(11):1882–1891, 2010. (Cited on page 37.)

Vinod Nair and Geoffrey E. Hinton. 3d object recognition with deep belief nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1339–1347. Curran Associates, Inc., 2009. (Cited on pages 23 and 76.)

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *international conference on machine learning (ICML)*, pages 807–814, 2010. (Cited on page 16.)

Michael Neff and Eugene Fiume. Modeling tension and relaxation for computer animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 81–88. ACM, 2002. (Cited on page 5.)

Andrew Ng. Sparse autoencoder. *CS294A Lecture notes, Stanford University*, 2011. (Cited on pages 8, 23, and 24.)

James F. O'Brien, Robert E. Bodenheimer, Gabriel J. Brostow, and Jessica K. Hodgins. Automatic joint parameter estimation from magnetic motion capture data. In *Proceedings of Graphics Interface 2000*, pages 53–60, May 2000. (Cited on page 5.)

Alessandro Panarese and Benoni B Edin. Human ability to discriminate direction of three-dimensional force stimuli applied to the finger pad. *Journal of Neurophysiology*, 105(2):541–547, 2011. (Cited on page 35.)

P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. Skill learning and task outcome prediction for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. (Cited on page 64.)

P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal. Towards associative skill memories. In *IEEE-RAS International Conference on Humanoid Robotics (HUMANOIDS)*, 2012. (Cited on page 63.)

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *International Conference on Robotics and Automation (ICRA)*, pages 763–768. IEEE, 2009. (Cited on page 74.)

J. Peters, K. Muelling, and Y. Altun. Relative entropy policy search. In *National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*, pages 1607–1612, 2010. (Cited on page 29.)

Zoran Popović and Andrew Witkin. Physically based motion transformation. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM Press/Addison-Wesley Publishing Co., 1999. (Cited on page 5.)

Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. (Cited on page 4.)

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. (Cited on pages 3, 5, 6, and 26.)

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. pages 1278–1286, 2014. (Cited on pages 9 and 25.)

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988. (Cited on pages 3, 22, 65, and 75.)

Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Comput.*, 10 (8):2047–2084, November 1998. ISSN 0899-7667. doi: 10.1162/089976698300016963. (Cited on page 75.)

B. Schölkopf, J. Platt, and T. Hofmann. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1345–1352, 2007. (Cited on pages 4, 8, and 77.)

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. (Cited on page 4.)

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. (Cited on page 20.)

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, 2006. (Cited on page 27.)

Maximilian Sölch, Justin Bayer, Marvin Ludersdorfer, and Patrick van der Smagt. Variational inference for on-line anomaly detection in high-dimensional time series. *International Conference on Machine Learning (ICML) Workshop*, 2016. (Cited on pages 9 and 88.)

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. (Cited on page 17.)

Yu Sun, J.M. Hollerbach, and S.A. Mascaro. Estimation of fingertip force direction with computer vision. *IEEE Tr Robotics*, 25(6):1356–1369, 2009. ISSN 1552-3098. doi: 10.1109/TRO.2009.2032954. (Cited on page 36.)

Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted Boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1601–1608, 2009. (Cited on page 74.)

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT Press Cambridge, 2012. (Cited on pages 5 and 27.)

Graham W. Taylor, Geoffrey E. Hinton, and Sam T. Roweis. Modeling human motion using binary latent variables. In B. Schölkopf, J.C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1345–1352. MIT Press, 2007. (Cited on page 74.)

Graham W. Taylor, Leonid Sigal, David J. Fleet, and Geoffrey E. Hinton. Dynamical binary latent variable models for 3d human pose tracking. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition*, pages 631–638, 2010. (Cited on page 8.)

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *International Conference on Learning Representations (ICLR)*, 2016. (Cited on pages 9 and 88.)

Michalis K. Titsias and Neil D. Lawrence. Bayesian Gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 844–851, 2010. (Cited on page 6.)

Jonathan Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. *CoRR*, abs/1406.2984, 2014. (Cited on page 7.)

Alan M Turing. Computing machinery and intelligence. *Mind*, 59 (236):433–460, 1950. (Cited on page 4.)

Sebastian Urban, Justin Bayer, Christian Osendorfer, Goran Westling, Benoni B. Edin, and Patrick van der Smagt. Computing grip force and torque from finger nail images using Gaussian processes. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2013. (Cited on page 44.)

Raquel Urtasun, David J. Fleet, Andreas Geiger, Jovan Popovic, Trevor Darrell, and Neil D. Lawrence. Topologically-constrained latent variable models. In *International Conference on Machine Learning (ICML)*, volume 307, pages 1080–1087, 2008. (Cited on page 77.)

H. van Hoof, T. Hermans, G. Neumann, and J. Peters. Learning robot in-hand manipulation with tactile features. In *International Conference on Humanoid Robots (HUMANOIDS)*, 2015a. (Cited on pages 28 and 64.)

H. van Hoof, J. Peters, and G. Neumann. Learning of non-parametric control policies with high-dimensional state features. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015b. (Cited on pages 28, 30, 64, 66, and 67.)

Herke van Hoof, Nutan Chen, Maximilian Karl, Tucker Hermans, Gerhard Neumann, Patrick van der Smagt, and Jan Peters. Learning robot in-hand manipulation with tactile features. *Robotics: Science and Systems (RSS) Workshop on Bootstrapping Manipulation Skills*, 2016a.

Herke van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2016b.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, pages 1096–1103, 2008. (Cited on pages 8, 23, and 74.)

Jörn Vogel, Claudio Castellini, and Patrick van der Smagt. EMG-based teleoperation and manipulation with the DLR LWR-III. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 672–678, 2011. (Cited on page 5.)

J.M. Wang, D.J. Fleet, and A. Hertzman. Gaussian process dynamical models for human motion. *Transactions on Pattern Recognition and Machine Intelligence*, 30(2):283–298, 2008. (Cited on pages 4 and 7.)

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28, pages 118–126, May 2013. (Cited on pages 17 and 19.)

Z. Wang, K. Muelling, M. P. Deisenroth, H. Ben Amor, D. Vogt, B. Schoelkopf, and J. Peters. Probabilistic movement modeling for intention inference in human-robot interaction. *International Journal of Robotics Research*, (7):841–858, 2013. (Cited on page 7.)

M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2728–2736, 2015. (Cited on page 65.)

Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013. (Cited on page 5.)

Greg Welch and Eric Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Computer graphics and Applications*, 22(6): 24–38, 2002. (Cited on page 5.)

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (Cited on page 21.)

Christopher KI Williams. Computing with infinite networks. *Advances in neural information processing systems*, pages 295–301, 1997. (Cited on page 18.)

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical Bayesian approach. In *International conference on Machine learning (ICML)*, pages 1015–1022. ACM, 2007. (Cited on page 92.)

Jeffrey L Wilson. Microsoft kinect for xbox 360. *PC Mag. Com*, 2010. (Cited on page 5.)