Generative Human Motion Modeling Using Deep Probabilistic Neural Networks

Thesis

submitted in partial fulfillment of the requirements for the degree

Master of Science

Graduate School of Neural Information Processing

Faculty of Science

Faculty of Medicine

Eberhard-Karls-Universität Tübingen

Presented by

Nima Ghorbani

from Tabriz, Iran

Tübingen, November 25, 2017

Thesis Advisor: Prof. Dr. Martin Giese
Section for Computational Sensorimotorics,
Department of Cognitive Neurology,
Hertie Institute for Clinical Brain Research,
Werner Reichardt Center for Integrative Neuroscience,
University Clinic Tübingen
Tübingen, Germany

Second Reader: Prof. Dr. Martin V. Butz
Computer Science and Psychology,
Cognitive Modeling,
University of Tübingen
Tübingen, Germany

- I affirm that I have written the dissertation myself and have not used any sources and aids other than those indicated.
- I affirm that I have not included data generated in one of my laboratory rotations and already presented in the respective laboratory report.

Date / Signature: November 25, 2017 Nima Ghorbani

Dedication

To the derives of my life: things that I don't know and my loved ones that in their comfort I dare to know more.

Acknowledgments

I appreciate working along the Computational Sensomotorics group members especially Albert Mukovskiy, Andrea Christensen, Enrico Chiovetto, Björn Müller, Jindrich Kodl, Leonid Fedorov, Mirjana Angelovska, Mohammad Hovaidi Ardestani, Nick Taubert, Nicolas Ludolph, Nitin Saini, and Tjeerd Dijkstra for our inspiring discussions and their friendly support. Tjeerd and Nick shared their lab equipment with me, which made this whole work possible. Mohammad helped me to bring the manuscript closer to a scientific writing for which I am grateful.

I am thankful to my advisor, Prof. Dr. Martin Giese, for providing me a place in his lab and his full support for every aspect of this work. My gratitude also goes to my second advisor, Prof. Dr. Martin Butz, for Âť HIS enthusiastic guidance and help for the final version of this manuscript.

I am thankful towards my Mom and Dad, Esmat and Younes, for their unconditioned love and support. Last but not least, I am grateful towards the love of my life, Parisa, for bearing with me through the past six months that I was often not enough there for her.

I hope the ideas and the spirit of this thesis would contribute to a world of peace and mutual respect.

Abstract

Deep Neural Networks (DNNs) have made significant contributions in the computer vision field. Among these, generative models of the natural images based on DNNs have demonstrated an unprecedented performance among the attempts in capturing the underlying generative process for these signals. Same as the images, human motion can be quantized via the Motion Capture (MoCap) techniques resulting in a structured, high-dimensional time-series signal, that is employable in training DNNs. Moreover, despite high-dimensionality of the human motion signal, it has been shown that its variations can, in fact, be described by a mapping from a set of relatively low number of parameters, or a motion manifold [Elgammal and Lee, 2008].

In this thesis, we present a Deep Generative Human Motion Model (DGHMM) that has a structured, low-dimensional latent variable manifold of human motion and is capable of producing novel realistic trajectories for joints of an articulated human skeleton. Following the work of Kingma and Welling [2013], we make use of DNNs in a probabilistic Autoencoder (AE) configuration to construct such a model. We pay particular attention to the organization of the data on the manifold while utilizing concepts already established in training deep generative models in computational vision field and apply them to the human motion time-series signal.

The result is a compact, continuous, and smooth manifold of the human motion signal that is capable of generating novel human joint trajectories. We propose and evaluate a set of architectural choices that make DNNs specifically suitable for this synthesis task. We investigate the practical manifold size and then use our curated labeled dataset of human motion clips to provide a visualization of the manifold.

Clusters of semantically similar motions appear on the manifold, which yield a 61.05% accuracy for the test set in a simple k-Nearest Neighbors (k-NN) classification task applied on the latent-code with 15 motion classes. Based on these observations we introduce a benchmark to test the usefulness of the manifold in a secondary retrieval task, which has practical use in providing a quantitative measure of similarity for different human motion contents. Additionally, we demonstrate the smoothness of the manifold by utilizing its ability in linearly blending the content and style of the given motions. Moreover, we show samples from the DGHMM that are rated by our human subjects on average above the purely artificial score. Finally, we investigate the effect of using different representations of 3D human motion data for generative modeling purposes and conclude that so far the

best performing choice would be the joint position representation[1].
To summarize, we have developed a framework for building various DNN models in AE configuration for human joint trajectory data. Additionally, we provide various tools in open source Python programming language[2] for the interested audience to conduct further experiments on our presented DGHMM. The results of this research have expected benefits in computational neuroscience, animation research, robotics, and interactive virtual reality where the feed for the full-body joint trajectories can be produced by a real-time generative model such as the one presented here.

[1] Joint positions are in local body coordinate system relative to the hips node as the root; c.f. Chapter 3.
[2] Actually for interactive Python environment, namely Jupyter Notebooks.

Contents

List of Figures

List of Algorithms

Acronyms

AAE Adversarial Autoencoder

AE Autoencoder

BN Batch Normalization

CMU Carnegie Mellon University

DAE Denoising Autoencoder

DGHMM Deep Generative Human Motion Model

DNN Deep Neural Network

DOF Degree of Freedom

FCN Fully-Connected Network

GP Gaussian Process

GPDM Gaussian Process Dynamical Model

GPLVM Gaussian Process Latent Variable Model

GPU Graphics Processing Unit

HMM Hidden Markov Model

k-NN k-Nearest Neighbors

KL Divergence Kullback-Leibler Divergence

MoCap Motion Capture

PCA Principle Component Analysis

RBM Restricted Boltzmann Machine

SGD Stochastic Gradient Descent

t-SNE t-Distributed Stochastic Neighbor Embedding

VAE Variational Autoencoder

VR Virtual Reality

Chapter 1

Introduction

A Generative Model is a distribution over the data, $P(X)$, where X usually has a high-dimensionality. For example, high number of correlated dimensions, namely pixels, compose an image and in this case, a generative model would try to capture some statistics of the data, e.g. correlation between pixels, so that the ones having closer features such as location and color would correspond to the same semantic group such as a human face. Then one can use such a generative model to produce faces that are similar to those in the training dataset but not a replica. Formally, if the original dataset is distributed according to $P_d(X)$, we would like to build a $P_g(X)$, that is very close to $P_d(X)$, which can also be sampled from.

Human motion, same as the digitized images, can be captured and processed as a quantized signal. The capturing is done via the Motion Capture (MoCap) techniques that commonly record the 3D Cartesian coordinates of some specially placed markers on the target body. After post-processing, each instance of the resulting time-series signal will contain some parametrized configuration of the key points[1] of the skeleton. Each instance of the configurations is called a pose, and pose parametrization can be done by at least three numbers representing either the joint positions or the joint angles[2]. Therefore, each data point in our dataset consists of over ten thousand features that are the variation of poses over time. Contrary to this high dimensional visible space, it is known that each such human motion data point is the result of a mapping from samples of a low-dimensional manifold that is embedded in a high dimensional motion space [Bowden, 2000; Elgammal and Lee, 2008]. To learn such a manifold, we would need a trainable model and an unsupervised technique for training it.

In this thesis, we use a Deep Neural Network (DNN) as our flexible model and train it in an Autoencoder (AE) configuration that provides a generative low-dimensional manifold of human motion signal. Validity of these choices are supported by three facts: first, existing experimental evidence demonstrate that deep layers of connected neurons are favored by the mother nature as general purpose function approximators [Hinton, 2007];

[1] e.g. Joints

[2] Angles between body segments

second, in practice construction of hierarchical, high level abstractions from data has shown beneficial effects in various tasks such as image and speech recognition [Graves et al., 2013; Krizhevsky et al., 2012]; Finally, carefully regularized AEs implemented with DNNs can learn a mapping from the visible data space to a low-dimensional code and vice versa, in a completely unsupervised way, while the low-dimensional code can also be interpreted as a manifold [Vincent et al., 2010]. These networks have shown competitive performance in representation learning from large amounts of data with no label information, which is in contrast to the majority of DNNs that are in use today that need expensive-to-obtain labeled data.

A recent work by Holden et al. [2016] benefits from a single layer Denoising Autoencoder (DAE) to create a motion manifold over large amounts of available MoCap data. As extending their work to generative human motion modeling, we use a different kind of AE, namely a Variational Autoencoder (VAE) [Kingma and Welling, 2013] that is based on probabilistic Bayesian Inference; a generative model in nature that enables us to directly sample from the manifold to get realistic, novel human motion data. VAEs learn a joint distribution over the data and the latent-code that is most probable given the observed data points and some explicit prior distribution over the latent space, e.g. a spherical multi-dimensional Normal distribution. This regularizes the manifold, which is in VAE setting the posterior distribution conditioned on a given observation by forcing it to be as close to a prior distribution as possible[3]. Therefore, we will have control on the coordinate system that is spanned by the manifold representation. By imposing a prior, the dimensions of the latent-code are forced to learn independent representations, which can be considered as an improvement over the blind manifold learning in Holden et al. [2015].

Our present research is an unprecedented attempt to model the distribution of naturalistic human joint trajectories across various motion styles and contents. By having a generative model of the human motion signal, we might be able to get a better understanding of the actual generative process that is happening in our nervous system; or in another use case, by having a conditional generative model we can control any humanistic skeleton with a high level, goal-directed command such as throwing a ball to a specific point, or catching the incoming ball, and it will automatically execute the command in a naturally plausible way without being programmed for that specific task.

Further advancements brought by our research to the field of Computational Sensomotorics are as following: (1) In contrast to Holden et al. [2015], in our deep network (9 layers) we don't use custom made layers, and rather we try to benefit from advances in computational vision field and use computations and procedures that are more carefully studied. (2) In addition to the joint position representation provided by Holden et al. [2015], we provide joint angle data representation across various human MoCap datasets that are further used to train a manifold for joint angle data. (3) We also extract labels

[3] In Kullback-Leibler Divergence (KL Divergence) sense.

from the description of motions that can further be used for visualization purposes or semi-supervised training of VAEs for human motion data [Kingma et al., 2014]. (4) We provide various visualizations that help to understand the organization of the data on the manifold further, and quantify performance of the latent-code in a secondary classification task. (5) We conduct a psychophysics experiment to assess the naturalness of the generated samples of our generative model.

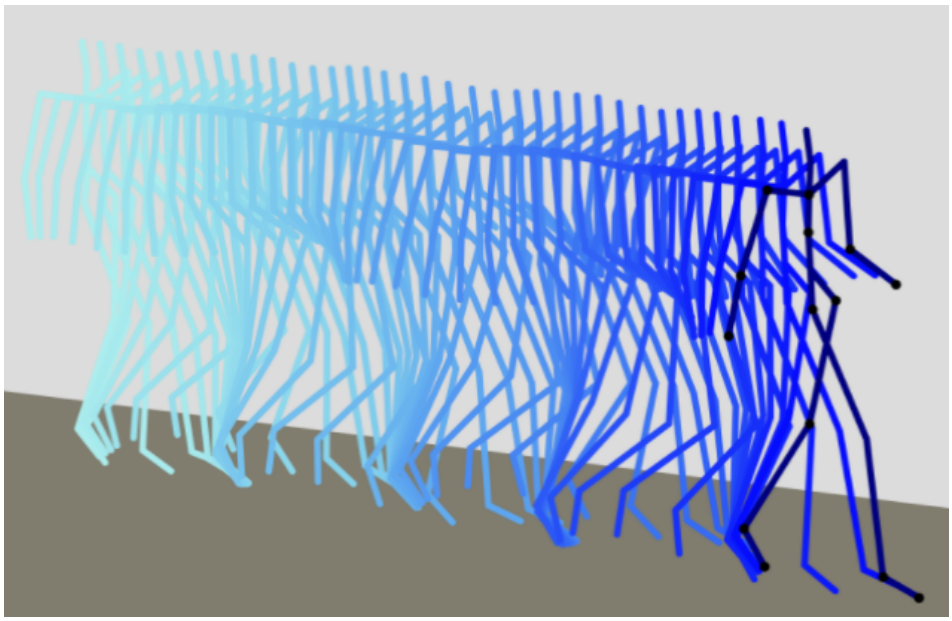
In the following sections of this chapter, we review the literature related to attempts in data-driven human motion modeling, especially those that try to make a motion manifold as a primary step. In Chapter 2, we explain the theoretical concepts behind the Deep Generative Human Motion Model (DGHMM) that is presented in this thesis. These include parametrization methods used for representing time-series motion data, basic DNN models, and various layers that are used in the architecture of our model. Furthermore, we will clarify specific models and training procedures that we use in our work, such as DAE, and VAE. In Chapter 3, the details about the training data, as well as the used hierarchical human skeleton will be explained. Moreover, we will describe the specific architecture of our generative model. In Chapter 4, we will present our generative model, and its low-dimensional manifold by various means: first, we clarify how we decided on the specific manifold size, then we apply different visualization techniques to study the organization of the training data on the manifold by encoding some labeled motion data, and subsequently apply dimensionality reduction on the latent-code to come up with a visualization of the manifold. Next, we study smoothness of the manifold in blending style and content of different motions. Finally, we show samples from our DGHMM that demonstrate the ability of its generative process. In Chapter 5, we discuss the possible improvements and future research directions of our work.

1.1 Human Motion Synthesis

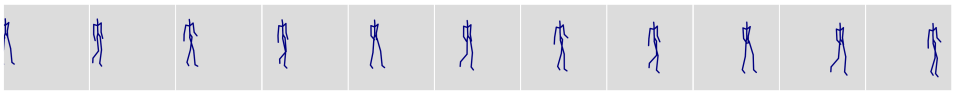
In capturing human motion, trajectories of the *key points* of the skeleton are recorded as a time-series of 3D Cartesian coordinates, where each frame has the pose information in that specific time point. Later these key points can be connected by lines to create simple stick figures (Fig. 1.1) or be used to derive a fully skinned avatar and replay a life-like 3D avatar. Rather than using recorded motion data, an animator can produce a behaving character motion by *keyframing*[4]. That means, setting the character pose at key time points within a software and then using the software to interpolate those frames through time to make the full trajectories. This method is slow and time-consuming. Therefore, one can use the existing MoCap data to substitute or complement the keyframing.

Here we widely categorize different methods for exploiting MoCap data for human motion synthesis into three categories: Direct methods, Physics-based methods, and

[4]The word *key* is used in two different meanings in animation production. One is used for describing the important nodes of the skeleton, that is recorded via MoCap techniques. In another case, it refers to specific, or key, time points, where an animator manually sets the pose setting of an articulated character.



(a)



(b)

Figure 1.1: Stick figure walk. By capturing the variation of 3D positions of the key points of the skeleton through the MoCap techniques, a life-like moving avatar can be reproduced. Here these key points are connected by lines to create the simple stick figures. (a) Rotoscopic plot where lighter colors show early time points of the motion clip and the darker colors show the later time points. (b) Stroboscopic plot of the same motion with shots at certain time points of the animation.

Modeling methods. In the following, we will review the literature on these categories, while highlighting their differences with our method.

1.1.1 Direct Methods

Direct methods are considered the most basic way to generate new motions based on the available MoCap data. Here, a transformation operation such as concatenation, blending, or interpolation of two or more motion sequences is commonly used to produce an output motion. Fundamental to all such methods is a rich dataset of motions that covers huge number of task-specific examples.

A set of works use a graph-based search method to retrieve the sequence of motions that are needed to concatenate to achieve a task that has been defined for the character [Arikan and Forsyth, 2002; Kovar et al., 2002; Lee et al., 2002]. These methods, more like others in this category, suffer from the extensive relying on the content already in the dataset. Another set of works tries to blend-in precisely aligned, classified, and segmented motions from the dataset to generate an interactive character [Mukai and Kuriyama, 2005; Park et al., 2002; Rose et al., 1998]. Despite the efforts to automate the aligning step [Kovar

and Gleicher, 2004], these methods still require intense preprocessing, which makes the production process very time-consuming. Additionally, to address the blending artifacts, massive post-processing steps, including inverse kinematics, might be required, that add to the already complicated motion production procedure.

1.1.2 Physics-based Methods

Physics-based methods can be seen as the most comprehensive way to address the problem of understanding human motion by a physical-dynamical model, including masses, springs, and forces. Although possible [Liu et al., 2005], they are nearly unfeasible in general case of realistic human motion production.
In our work we use a model, that is trained only on a MoCap dataset, without imposing explicit physical constraints. Nevertheless, our generated motions follow realistic dynamics without any effort in constraining the motion, although if desired, constraining the trajectories is still possible through applying an optimization on the latent-code to minimize the constraint cost of the final output [Holden et al., 2016].

1.1.3 Modeling Methods

Statistical modeling of the human motion is made possible through the concept of *key points*[5]. These landmarks provide the means for a model to capture long range, nonlinear dependencies between them in the existing MoCap data, that is later used in various tasks such of generation of novel motions. The first effort in this direction incorporated the Hidden Markov Models (HMMs) [Brand and Hertzmann, 2000] while trying to separate the style and the content of motions within a motion dataset to learn more general features. However, HMMs with their simple, discrete states are inefficient for continuous human motion data. Piecewise linear dynamical models have been offered to address this problem [Bissacco, 2005; Li et al., 2002; Pavlovic et al., 2000], however, they require expensive approximations for inference and learning.
Another line of efforts concentrated in leveraging flexible Gaussian Processs (GPs) to simultaneously learn a low-dimensional embeddings in the motion data via a Gaussian Process Latent Variable Model (GPLVM) [Lawrence, 2004] and a dynamical model on top of these embeddings via a Gaussian Process Dynamical Model (GPDM) [Wang et al., 2005, 2008]. Taubert et al. [2013, 2014] have successfully applied them in real-time Virtual Reality (VR) setups where a human can interact with an online avatar for a handshake or high-five actions. These models are called lazy learning algorithms, because they do not record trained parameters while training, and they require the whole data from the training set to work in the test time. This is a big disadvantage that effectively cripples these models from being applied to large datasets. Additionally, these models cannot simultaneously synthesis different types of motion, effectively because in these models

[5]Assigning a point to the center of the joint in an articulated skeleton.

"data speaks for itself" [Rasmussen and Williams, 2005], and obviously different types of motion data naturally don't say the same thing.

Alternative approaches to the data-driven human motion modeling use neural networks. These methods scale very well to large datasets and benefit from efficient training algorithms [Rumelhart et al., 1986]. Taylor and Hinton [2009] use a Restricted Boltzmann Machine (RBM) to learn a probability distribution over each frame of the time-series motion data, and infer an intractable posterior of a latent variable model. Their model can generate a motion content with different styles via a single model, yet we want to model the vast amount of contents and styles at the same time. Moreover, these models use a time-series interpretation where generating each frame requires multiple previous frames. Holden et al. [2015, 2016] make another elegant attempt at model-based human motion generation. They use a feedforward shallow convolutional neural network to make a low-dimensional representation of human motion, and by combining this network with few other task-specific regressors, they can produce motions in which the character follows a specified task with realistic joint trajectories. Our approach can be considered as an extension to their work, with the aim of learning a true distribution on the human motion dataset with a deep probabilistic neural network model. Following Holden et al. [2015, 2016], we take a clip of motion as a single entity like an image and try to learn the temporal dynamics at once, and in different hierarchical levels, just like in image-based DNNs. This is as well a time-series approach, however we tend to use only feedforward networks without any recurrent element to avoid unnecessary complexities.

Chapter 2

Theoretical Foundations

In this chapter, we will cover the theory behind the Deep Generative Human Motion Model (DGHMM) that we have developed in this thesis. We begin by introducing the different parametrization methods of the time-series 3D human motion signal. Next, we present a fast review of the basics of Deep Neural Networks (DNNs) and the math behind the specific layers, as well as the two Autoencoder (AE) architectures with specific regularizations that we have extensively used.

2.1 Human Motion Data

To create a model of the human motion, first, we have to capture it as a quantized signal. This can be done with the Motion Capture (MoCap) techniques, that are either marker-based or marker-less. Most common marker-based optical techniques use cameras to precisely record 3D Cartesian positions of the markers that are placed at specific positions on the target body, as in Fig. 2.1. Later, marker positions are used to infer positions of the joints[1] or angles between body segments[2] at each time point of the capture session. The final motion clip will be a time-series signal, whose certain window will be a single data point $X \in \mathbb{R}^{T \times D}$ in our dataset. Here T would be the window size[3], and D will be the overall Degrees of Freedom (DOFs) for representing the skeleton pose configuration in each frame. In the following, we will review different ways to do this representation in 3D space, and the methods we chose for this thesis.

2.1.1 3D Motion Parametrization

Joint Positions is an intuitive way to represent poses of an articulated skeleton hierarchy in 3D Cartesian space with three numbers representing the distance of each joint from the normal Cartesian planes. We use joint positions in local body coordinate system where each joint's position is relative to a reference node, or root. Using joint position

[1] Center of the joints. Joint Positions.
[2] Joint Angles
[3] Number of frames in the clip. T divided by the sampling framerate shows the duration of the clip.

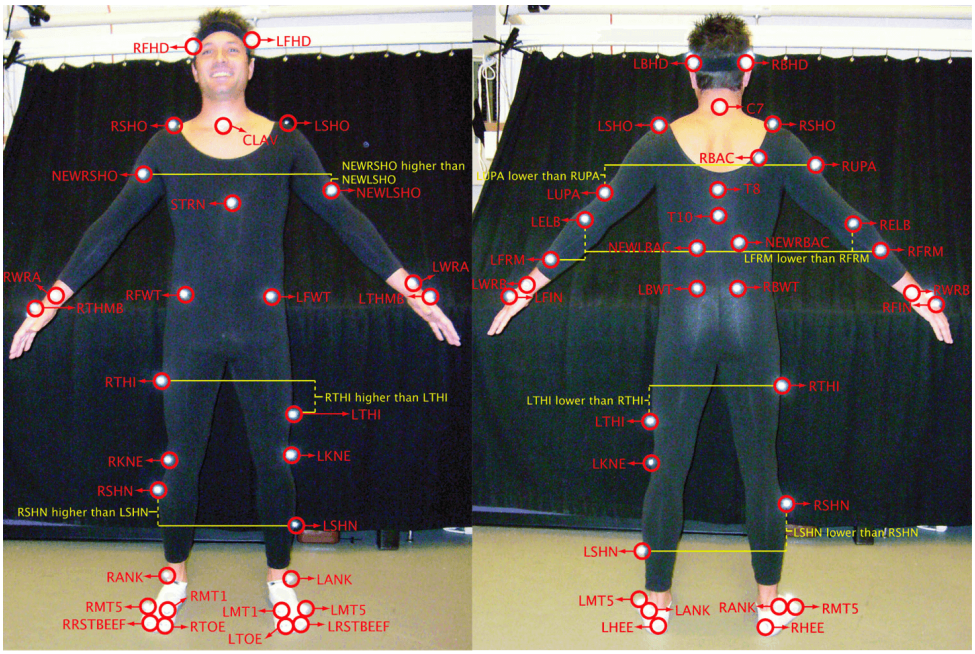


Figure 2.1: Example of marker placements for an optical-based MoCap setup. Marker-based optical MoCap techniques record precise positions of the markers placed on specific locations on the target body by using multiple fast infrared cameras. The image shows the marker placement used throughout recordings of the Carnegie Mellon University (CMU) MoCap Database. After the recording session, the marker positions are used to infer positions of the center of the joints or the angles between segments of the body. Image taken from `http://mocap.cs.cmu.edu`.

to represent full-body pose configuration has several advantages: distance between body parts that avoid a self collision is always implied in the data; also, task-specific constraints for guiding the generated full-body joint trajectories can be easily accomplished. As a disadvantage, joint positions are not commonly used to control the actuators of a robot directly; also constraints on the bone lengths and the joint angles could theoretically be violated.

Joint Angles is used to refer to parameterizing three DOF rotations of each joint from its initial pose relative to its previous node in the skeleton hierarchy to reach a certain orientation. To parameterize these rotations one can choose *Euler Angles*, which uses rotations around three mutually perpendicular axes to describe a three DOF rotation. Although this parametrization is intuitive, it suffers from Gimbal Lock, where one axis of rotation is lost when the two other axes become parallel with each other. Additionally, in this representation, a single orientation can be represented by the combination of many non-unique rotations, and the interpolation of orientations is not straightforward. To remedy all these problems one can use *Quaternions*, that parametrize angular dispositions with four numbers instead of three, hence the name. In this thesis, we chose not to use Quaternions, because then one DOF per joint would be added to the total DOF count, which is undesired. Instead we opt for the *Axis-Angle* parametrization and their direct relative, *Exponential Maps* [Grassia, 1998], that use three numbers for 3D rotation parametrization purpose.

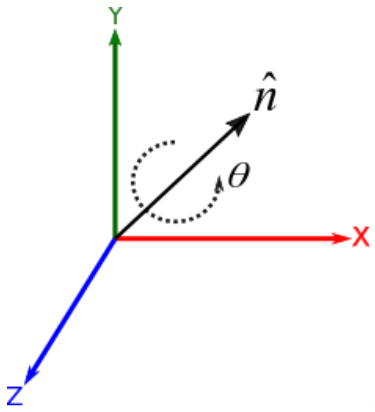


Figure 2.2: Angle-Axis parameterization of an orientation. $\hat{n}$ is the unit-length axis of rotation going through the origin and θ is the rotation angle about that axis.

The *Euler's rotation theorem*, states that any 3D orientation can be described by a single rotation around a specially defined axis. With the Euler angles representation, this axis is restricted to the cardinal axis, but in the axis-angle representation, one is free in choosing the axis of the rotation (Fig. 2.2). Axis-angle representation uses a unit-length axis of rotation, $\hat{n}$, and a three-component rotation angle θ to represent any orientation. As an alternative, one can multiply $\hat{n}$ by θ, since the unit length, to arrive at exponential map representation.

$$e = \hat{n} \times \theta$$

It is obvious that the axis of rotation and the rotation angle can both be deduced from e. This parametrization of rotations is the most compact form that also avoids all the problems of the Euler angle parameterization. Throughout this thesis, we use exponential maps representation of joint angles[4], therefore each joint will impose three DOFs. Using joint angles to represent full-body pose configuration has several advantages: joint angles can be directly fed to the robot actuators to control its movement; joint angles provide smoother trajectories since their dynamics are always constrained to biophysics of the actual body. On the other hand, defining goal-directed constraints for character motion with joint angles is not as straightforward as in joint positions case.

Having forward kinematics (joint positions) helps with fast constraining of the character motion to reach a defined goal[5]. Also, inverse kinematics is needed for Virtual Reality (VR) and robotics setups. Therefore, in this thesis, we provide both, independently, as data for training of the DGHMM for each representation. In case of using a regressor on top of the manifold to reach a defined goal, it would be best implemented by joint position representation as shown in Holden et al. [2016].

[4]Rotation of joints of the articulated, hierarchical skeleton.
[5]Such as precise end-effector positions

2.2 Generative Human Motion Manifold

Our aim is to learn a low-dimensional latent-code, or the Manifold, Z [6] from the human motion data X, along with the mappings from the data space to the latent space, $q(Z|X;\theta)$, and vice versa, $q(\hat{X}|Z;\phi)$. For this purpose, we use a *denoising criterion* [Vincent et al., 2010] as our unsupervised objective to train a DNN in AE configuration. We also intend to control the structure of the manifold and bring its distribution very close to a Normal distribution so that the decoder can be easily sampled from to create theoretically endless novel human motion trajectories. We do this by probabilistic interpretation of the AE, and matching the latent-code to a prior multi-dimensional Normal distribution [Kingma and Welling, 2013]. This effectively ensures that the manifold is compact and any point on the manifold corresponds with high probability to some valid full-body human joint trajectories.
In the next parts, we briefly review the math behind the DNNs and the important layers that we extensively use in our human motion model. Then, we briefly review two different AE types that have been mixed for this thesis. To communicate basic concepts, throughout this chapter we might give examples in a simple image-based dataset, the MNIST handwritten digit dataset, rather than the human motion dataset[7].

2.3 Neural Networks

Warren McCulloch and Walter Pitts proposed the first artificial neuron in their famous work [McCulloch and Pitts, 1943]. Their very first neuron model, called McCulloch-Pitts neuron, is still in use in today's complex networks. It is a weighted sum of some connections that is later passed through a nonlinearity function (Fig. 2.3)

$$y_k = \phi(\sum_{i=1}^{n} w_k x_i + b_k). \tag{2.1}$$

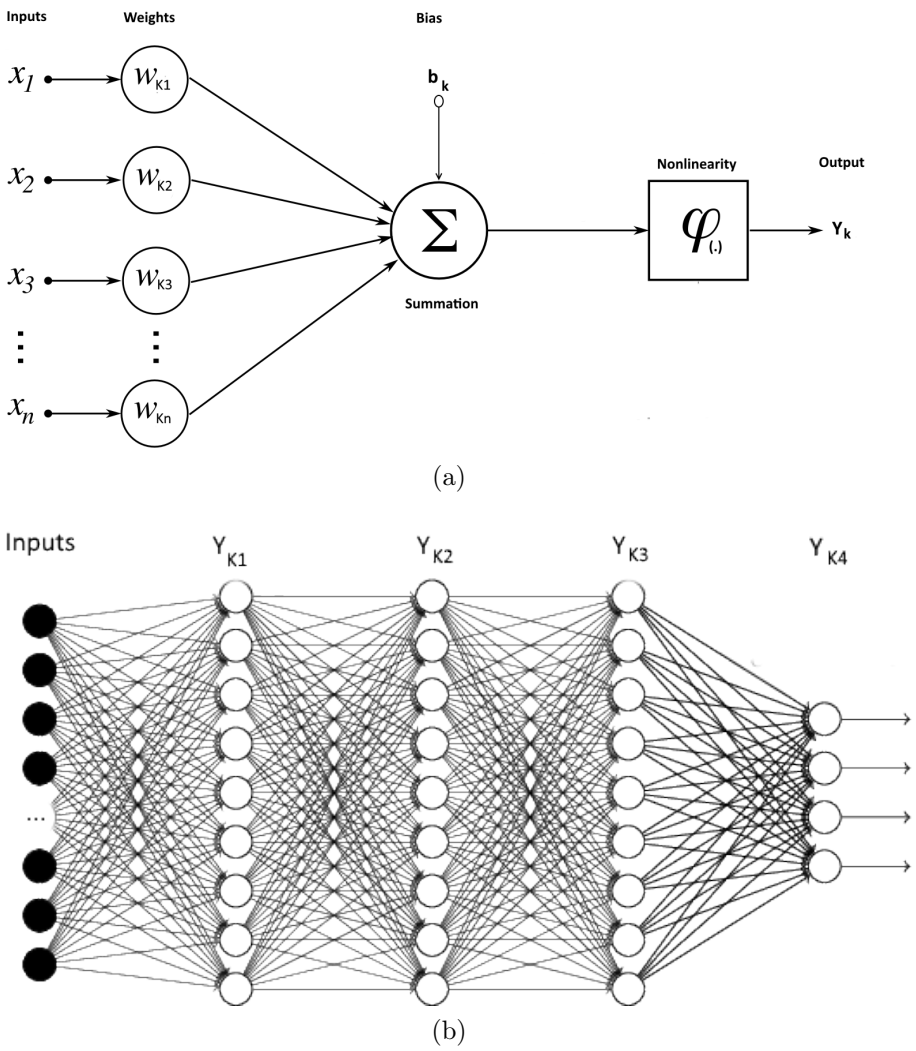
The common nonlinearity used to be a sigmoid function, but recently non-saturating functions such as Rectified Linear Unit (ReLU) has been shown to perform better [Nair and Hinton, 2010]. Therefore, each neuron in a fully connected layer of a neural network will be computed according to the following equation:

$$y_k = max(0, \sum_{i=1}^{n} w_k x_i + b_k). \tag{2.2}$$

The weights and biases of the networked neurons, which are adjustable parameters of the model [Rosenblatt, 1957], can be efficiently trained by descending the gradient of the negative loss via the backpropagation technique [Robbins and Monro, 1951; Rumelhart

[6] Latent-code, low-dimensional embedding, and Manifold are used interchangeably throughout this manuscript.

[7] This is a common practice in the field to start with small datasets and check the model validity.



(a)

(b)

Figure 2.3: Artificial Neural Network. (a) The building units of the sophisticated fully connected artificial neural networks are just affine transformations of an input passed through a nonlinear function. (b) While the weights and biases of the linear operation are trainable, a network of such neurons will turn into a general purpose function approximator. Images adapted from Haykin [2009].

et al., 1986]. We will use the batched version of the gradient descent, namely Stochastic Gradient Descent (SGD), where the gradient of the parameters is computed using only a fraction of the whole training dataset. For more information on this topic see Goodfellow et al. [2016].

Layers of neurons can be put in extended hierarchies to create a DNN. However, by layering fully-connected layers to make a deep Fully-Connected Network (FCN) the number of parameters grow very fast, which in turn makes the overfitting problem more severe, and also limits how deep we can make our model. Additionally, FCNs are oblivious about the coordinates of the input, either spatial or temporal. For example, a sample from the MNIST dataset is an image of spatial size of 28×28, $X \in [0:1]^{28\times 28}$. To use this data point as an input to a FCN, one should vectorize that image and therefore loose spatial information.

In Section 2.3.1, we will see how these problems can be addressed, for image-based data,

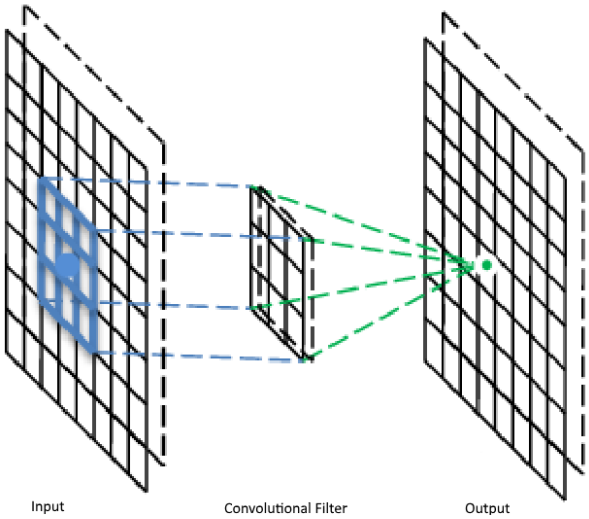


Figure 2.4: Convolution operation. Each channel of the input is convolved with its respective kernel window, that is strided over the input dimensions. Image adapted from [Guo et al., 2016].

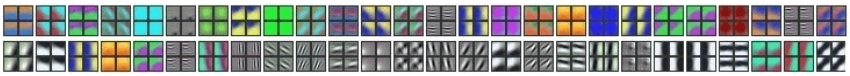
by using the idea of receptive fields in convolutional layers with weight sharing [LeCun et al., 1989]. Later we will use the repurposed spatial convolutional layers in the 1D temporal domain, where the height of the signal is one and width is the time axis. Then the channels will be the DOFs in our human skeleton.

2.3.1 Convolutions

Convolutional DNNs, unlike FCNs, benefit from the receptive fields that are biologically plausible [Hinton, 2007; Hubel and Wiesel, 1959]. The idea is that instead of vectorizing the input and multiplying it by adjustable weights, as seen in Eq. (2.2), we take a *window* of weights, also called filters or kernels, and slide it over the input dimensions, Fig. 2.4. This operation is effectively the cross-correlation of the input and the weights, which is more commonly called convolution in the neural networks research field. More specifically, if w is the window of $M \times N$ weights, and y^l are activations of layer l then:

$$y_{ij}^l = \phi\left(\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} w_{mn} \times y_{(i+m)(i+n)}^{l-1}\right) \tag{2.3}$$

Using convolutional layers instead of dense layers presented before has various advantages: First, one can work with arbitrarily sized inputs and yield arbitrarily sized outputs, since the dimensions of the kernels are independent of the input/output dimensions. Second, weight sharing significantly lowers the number of parameters introduced in each layer. This is done by retaining the weights for each channel of the output so that after training each weight will learn spatially invariant features that might happen anywhere in the input. Studies show that in the early layers of the network hierarchy, these are low-level



(a)

(b)

Figure 2.5: Visualization of the kernels in two layers of a convolutional DNN. (a) Filters of the first layer of a convolutional DNN trained on natural images. Here low-level features such as edges and colors are detected. (b) Filters in the 8th layer of the same network are more about higher level abstractions such as ships, chairs, etc. Image adapted from [Yosinski et al., 2015].

features such as edges or colors, and in the higher layers they are more abstract concepts such as faces, and objects, Fig. 2.5[8].

2.3.2 Transposed Convolutions

After striding the filters during the forward pass through a convolutional layer, the dimensionality of the output is subsampled. More specifically, for an input image of hight and width I, if the kernel dimensions are equal to F, with equal input paddings of P, after stride S along the height and width the output dimensions O will be:

$$O = \frac{I - F + 2 \times P}{S} + 1. \tag{2.4}$$

For example for an image of 28×28, and $F = 4, S = 2, P = 1$ then $O = 14$. This is also the same for 1-D temporal convolutions that we use in this thesis to obtain the low-dimensional manifold. To recover the lost information for the reconstruction objective, an upsampling mechanism is needed. For this purpose Holden et al. [2015, 2016] use a custom-made heuristic method, however we choose to use the better studied transposed convolutional layer, commonly known as deconvolution layer [Zeiler et al., 2010]. Hence the name, this layer is in nature a convolutional layer, whose input and output are swapped. Therefore, instead of downsampling after a strided convolution, the output will be upsampled.

2.3.3 Batch Normalization

Batch Normalization (BN) [Ioffe and Szegedy, 2015] is applied on the layer activations, which normalizes them by a running mean and variance. This has been shown to improve

[8]For more visualization of the kernels in different levels of the DNN hierarchy refer to Yosinski et al. [2015], especially see figure 5 of their paper.

convergence of the training process. BN has been reported to be an essential component in some generative models [Radford et al., 2015]. According to the law of total variance:

$$Var_{Total} = Var_{Signal} + Var_{Noise} \tag{2.5}$$

Therefore, one explanation for benefits of BN layer can be that it whitens the distribution of the activations and forces the training to be done on the actual signal variance, which expedites the convergence. We use BN on activation of all layers except the final layers in the encoder and the decoder.

2.4 Unsupervised Learning

Next, we will study a basic unsupervised training procedure suggested for DNNs that has been shown to be capable of learning *useful features* from the data when correctly regularized [Bengio et al., 2013; Vincent et al., 2010]. This architecture is called the AE, which we will use in an under complete mode with special regularizations to create a low-dimensional manifold of human motion.
An AE, as shown in Fig. 2.6, consists of an encoder, $f : X \to z$, that maps the data X to the latent-code z, and a decoder, $g : z \to \hat{X}$ that maps the latent-code to the reconstructed version of the input, $\hat{X}$. The training objective would be to bring $\hat{X}$ as close to X as possible by means of adjusting the parameters of the encoder and the decoder.
We will see how this architecture can be forced to learn useful features from the data, and how by a stochastic interpretation of the encoding/decoding mappings, one can turn a classical AE into a generative model.

2.4.1 Denoising Autoencoder

In order to motivate the AE network to learn useful features rather than an identity function, that would simply copy input to its output, one should regularize the model. One suggested regularization is called denoising criterion in which one provides the network with corrupted version of the input, and then the task of the network will be to fix this corruption by minimizing some distance measure between the output and the original input [Vincent et al., 2008, 2010]. More specifically the Denoising Autoencoder (DAE) loss function will be:

$$L_{DAE}(X, g(f(C(\tilde{X}|X), \theta), \phi)). \tag{2.6}$$

where C is the stochastic corrupting function, f is the encoder function, and g is the decoder. In all our implementations the reconstruction loss function will be the L^2 distance:

$$L_{DAE}(X; \theta, \phi) = ||X - g(f(\hat{X}; \theta); \phi)||_2 + ||\theta, \phi||_2. \tag{2.7}$$

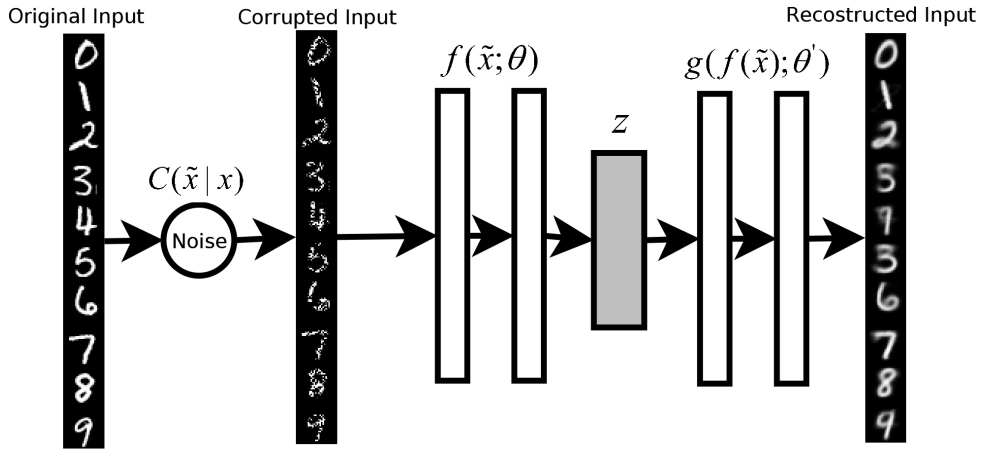


Figure 2.6: Denoising Autoencoder. The original input shows ten samples from the MNIST test set. The input is corrupted with 50% binomial noise, and then passed through a network with a final two dimensional latent-code, Z. The reconstructed image is shown on the right. The output of the network is compared to the original input, in the L^2 distance sense, and that error is used to train the model.

In Eq. (2.7) the second term is another regularization term to keep the weights low and regularized, and lowers the possibility of overfitting. Algorithm 1 shows the exact algorithm for training a DAE as in Fig. 2.6.

The denoising procedure forces the model to implicitly learn the data generating distri-

Algorithm 1 Denoising Autoencoder

$\theta, \phi \leftarrow$ Initial Parameters
repeat
  $X^M \leftarrow$ Random minibatch of M data points (drawn from the full dataset)
  $\hat{X} \leftarrow C(\hat{X}|X)$ ▷ Corrupt the input
  $grads \leftarrow \nabla_{\theta,\phi} L_{DAE}(\theta, \phi; X^M)$ ▷ Gradients of the loss Eq. (2.7)
  $\theta, \phi \leftarrow$Update parameters using *grads* (e.g. Adam)[Kingma and Ba, 2014]
until convergence of the parameters (θ, ϕ)
return θ, ϕ

bution [Alain and Bengio, 2012], and is one suggested way to learn useful representations from the data using a high capacity model [Bengio et al., 2013; Vincent et al., 2008]. Next we will present the manifold learning capability assigned to DAEs.

2.4.2 Manifold Learning Interpretation of Training DAEs

A **manifold** is a low-dimensional set of connected points that are embedded in a higher-dimensional data space[9]. Learning a manifold is to assume that a mapping from the data space to this low-dimensional embedding exists, and the important variations in the data space, as input to the mapping, occur only along the manifold representation, or the output of the mapping. For more on nonlinear manifold learning refer to [Narayanan

[9]Despite existing mathematical formalism for the term *manifold* we present a loose definition more appropriate to the machine learning field. For a more formal definition see Cayton [2005].

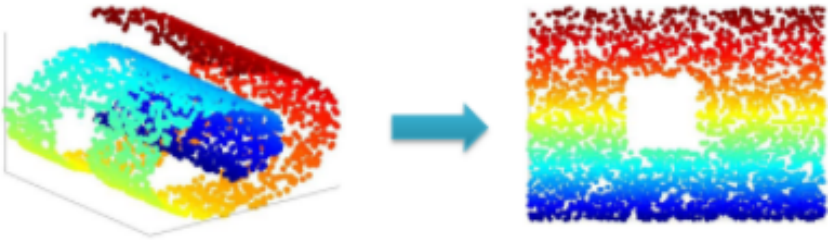


Figure 2.7: Nonlinear manifold learning. Image to the left shows a swiss roll, a $2D$ nonlinear manifold embedded in $\mathbb{R}^3$. Image to the right shows a $2D$ representation of the manifold learned by some nonlinear manifold learning algorithm. Image reproduced from [Zhou et al., 2013].

and Mitter, 2010; Roweis and Saul, 2000; Schölkopf et al., 1998; Tenenbaum et al., 2000; Weinberger et al., 2004].

An AE can learn a mapping to the low-dimensional representation of the data, and another mapping to recover the original data from this representation. It is important to note that the mappings are not from random inputs, rather inputs that are highly probable under the data generating distribution, $P_{data}(X)$. Moreover, in the denoising task any corrupted input $\tilde{X}$ is brought back to close vicinity of the manifold, Fig. 2.8. Therefore, a DAE can be seen as capable of learning nonlinear manifolds embedded in the data [Alain and Bengio, 2012; Bengio et al., 2013].

The assumption that human motion data lies along a low-dimensional manifold is due to two observations: First, by taking a uniform random number generator for each joints trajectory, we would never get structured samples such as human motions happening in everyday life; therefore, the distribution of realistic human motion data lies in a small subset of the total space of joint trajectories. Second, we can imagine neighborhoods for each motion that can be generated by varying, for example, only the speed of motion, or rotating the root node that causes the whole hierarchy to be rotated.

There has been a successful attempt by Holden et al. [2015] to use DAEs in creating a manifold of human motion from joint position data. In this thesis we make the first attempt, to our knowledge, to make a generative human motion model by extending their results by means of our DGHMM, which is a deep convolutional AE architecture that is able to generate novel motions by simply sampling from the internal manifold. We also study the possibility of creating such a manifold on joint angle data, which would make our model directly portable to robotics and VR setups.

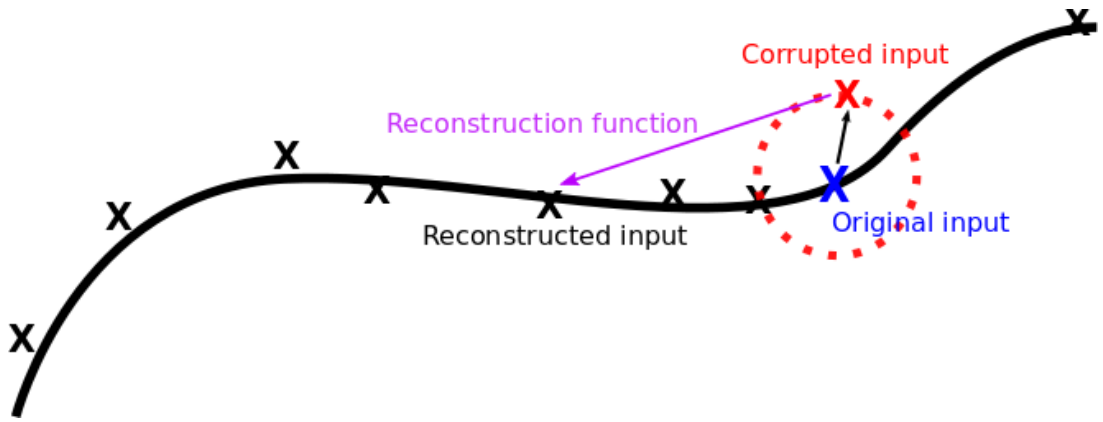


Figure 2.8: Manifold learning interpretation of training DAEs. Suppose that a low-dimensional manifold (the solid line) exists for the human motion data (black cross). Then during the training process the applied noise moves the data away from this manifold (red cross). The task of the DAE would be to bring the point back near the low-dimensional embedding. The interactions between the input corrupting process and the denoising procedure forces the model to be more sensitive to variations along the manifold while ignoring orthogonal ones. Image adapted from [Vincent et al., 2010].

2.5 Variational Autoencoder

Variational Autoencoders (VAEs) are probabilistic interpretation of AEs that are capable of learning the data generating distribution, as well as a latent variable representation of the data. They have been successfully applied in learning generative process of complex datasets, such as handwritten digits [Kingma and Welling, 2013], faces [Kingma and Welling, 2013; Kulkarni et al., 2015], physical scene models [Kulkarni et al., 2015], text sequences [Bowman et al., 2015] and others [Gregor et al., 2015; Maaløe et al., 2016]. In this section we review the math behind this generative model[10].

Take $P(X)$ as data generating distribution, and $P(z)$ as the distribution of the latent variable in an AE setting. Then by marginalizing z from the $P(X, z)$ we can get the data distribution:

$$P(X) = \int P(X, z)dz = \int P(X|z)p(z)dz. \tag{2.8}$$

In VAE setting, $P(z)$ is inferred from $P(z|X)$. That is as mentioned in Section 2.4.2 we make the latent-code likely under the data distribution. Then the idea would be to approximate the true posterior, $P(z|X)$, with a simpler distribution $Q(z|X)$, e.g. a Gaussian, and bring $Q(z|X)$ closer to $P(z|X)$ by means of Kullback-Leibler Divergence (KL Divergence). Therefore:

$$D_{KL}[Q(z|X)|P(z|X)] = E[\log \frac{Q(z|X)}{P(z|X)}] = E[\log Q(z|X) - \log P(z|X)]. \tag{2.9}$$

[10] All the math presented here are from the original paper Kingma and Welling [2013] with some steps added for clarity.

Additionally, we can also make $P(X|z)$ and $P(z)$ appear:

$$D_{KL}[Q(z|X)|P(z|X)] = E[\log Q(z|X) - \log \frac{P(X|z)P(z)}{P(X)}] \\ = E[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X). \quad (2.10)$$

$$D_{KL}[Q(z|X)|P(z|X)] - \log P(X) = E[\log Q(z|X) - \log P(X|z) - \log P(z)]. \quad (2.11)$$

And:

$$\log P(X) - D_{KL}[Q(z|X)|P(z|X)] = E[\log P(X|z) - (\log Q(z|X) - \log P(z))] \\ = E[\log P(X|z)] - E[\log Q(z|X) - \log P(z)] = E[\log P(X|z)] - D_{KL}[Q(z|X)|P(z)]. \quad (2.12)$$

This would be the VAE optimization object:

$$\log P(X) - D_{KL}[Q(z|X)|P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)|P(z)]. \quad (2.13)$$

Here $Q(z|X)$ will be our encoder that gives the latent-code given the data point, and $P(X|z)$ is the decoder that gives the reconstructed input given the latent-code, which can also be a sample from the prior distribution, $P(z)$.
The right hand side of the Eq. (2.13) has two parts: $E[\log P(X|z)]$ is the reconstruction loss given the latent-code; and the $D_{KL}[Q(z|X)|P(z)]$ is trying to bring the latent-code distribution as close as possible to $P(z)$. For simplicity, originally the authors of Kingma and Welling [2013] took $P(z) \sim N(0, I)$. This way if $Q(z|X)$ is a Gaussian as well, then $D_{KL}[Q(z|X)|P(z)]$ can be computed in closed form. In fact they took $Q(z|X) \sim N(\mu(X), \Sigma(X))$, therefore:

$$D_{KL}[N(\mu(X), \Sigma(X))\|N(0, I)] \\ = \frac{1}{2}(\text{tr}(\Sigma(X)) + \mu(X)^T\mu(X) - k - \log \det(\Sigma(X))) \\ = \frac{1}{2}\sum_k(\Sigma(X) + \mu^2(X) - 1 - \log \Sigma(X)). \quad (2.14)$$

The final loss function that we use is:

$$L_{VAE}(X; \theta, \phi) = ||X - g(f(X_c; \theta); \phi)||_2 - \frac{\alpha}{2}\sum_k(\Sigma(X) + \mu^2(X) - 1 - log\Sigma(X)) + ||\theta, \phi||_2. \quad (2.15)$$

The problem is that, it is not possible to use SGD while a stochastic sampling is present on the computation graph. Then how to compute z from the outputs of encoder[11]? The original authors of Kingma and Welling [2013] have introduced the *re-parametrization*

[11] The Mean and the Variance

Figure 2.9: Learned manifold of the faces by a VAE. A 2D manifold of the faces learned with a VAE and Algorithm 2. Each face is a sample from the generative model. Notice two independent factors of variations, namely face rotations and smiling, that are smoothly encoded along different dimensions of the manifold. Image adapted from [Kingma and Welling, 2013].

trick to deal with this issue. That is to take the stochastic sampler out of the graph and using the following formula instead:

$$z = \mu(X) + \Sigma^{\frac{1}{2}}(X) \odot \epsilon. \tag{2.16}$$

where $\epsilon \sim N(0, 1)$.

Fig. 2.9 shows a manifold of faces learned by a VAE from the Frey Face dataset[12], trained

Algorithm 2 Variational Autoencoder [Kingma and Welling, 2013]

$\theta, \phi \leftarrow$ Initial Parameters
repeat
  $X^M \leftarrow$ Random minibatch of M data points (drawn from the full dataset)
  $\mu, \Sigma \leftarrow$ Pass the input X through the encoder
  $\epsilon \leftarrow$ Random samples from the noise distribution
  $z \leftarrow \mu + \Sigma^{\frac{1}{2}}(X) \odot \epsilon$ ▷ Eq. (2.16)
  $\hat{X} \leftarrow$ Pass the latent-code z through the decoder
  $grads \leftarrow \nabla_{\theta,\phi} L_{VAE}(\theta, \phi; X^M)$ ▷ Gradients of the loss function, Eq. (2.15)
  $\theta, \phi \leftarrow$Update parameters using $grads$ (e.g. Adam)[Kingma and Ba, 2014]
until the ending critetrion is met
return θ, ϕ

with Algorithm 2.

In the objective of a VAE, the KL Divergence is essentially regularizing the latent-code

[12] http://www.cs.nyu.edu/~roweis/data.html

to make its distribution as close to a Gaussian as possible. Thus in a trained model, a sample from a Gaussian, can suffice to generate novel samples that are most likely similar to the samples in the training dataset.

The math published for the VAE only assumes a Gaussian prior while any other distribution would make the KL Divergence intractable. An alternative would be to use the Adversarial Autoencoders (AAEs) [Makhzani et al., 2015] that are very flexible in shaping the distribution of the latent-code to any arbitrary prior. That is done by replacing the KL Divergence loss in the VAE objective with an adversarial training procedure [Goodfellow et al., 2014] that tries to force the encoder to produces samples $q(z)$, that are similar to samples from the prior $P(z)$. We also made experiments in this direction, yet since we opted to use a Gaussian as our prior distribution on the latent-code, both accomplished similar results. On the other hand, training an AAE is much slower than the VAE because it includes multiple training loops that interfere with training the whole network on one Graphics Processing Unit (GPU).

Chapter 3

Methods

In this chapter, we clarify the procedures regarding the preparation of the data used for training the Deep Generative Human Motion Model (DGHMM), namely data formats, preprocessing steps and the output of the network. Then we explain the specific architecture of our model, as well as the objective function that we optimized the parameters of the model for. Finally, we delineate the exact training procedures, such as the implementation framework and the hardware used.

3.1 Data for Training the DGHMM

For training the DGHMM we benefit from three major, freely available human Motion Capture (MoCap) datasets, namely the Carnegie Mellon University (CMU) dataset[1], HDM05 [Müller et al., 2007], Berkeley MHAD [Ofli et al., 2013], as well as a small dataset by Holden et al. [2016]. Different recorded motions from different subjects and recording sessions are subsampled to 60 frames-per-second and retargeted to a standardized character with a simplified set of joints of 21 nodes, Fig. 3.1. The retargeting is done in python with the scripts released by Holden et al. [2015, 2016].
No commonly used, extensive preprocessing step, e.g. classifying and aligning, is applied to the motion clips. Any motion clip in our dataset is obtained by slicing a long source clip into windowed slices of T frames. Following Holden et al. [2015], we chose $T = 240$ frames and every slice overlaps the preceding clip by half, that is by 120 frames. If a single slice length is much smaller than the desired window length, then the motion represented within the slice is reversed in time and appended to the slice until the slice length gets within the desired range. If the length was still small, then frames from the beginning and the end of the slice are repeated until the desired length is achieved. In the end, any data point in our dataset will be a time-series signal, $X \in \mathbb{R}^{\mathrm{T}\times\mathrm{D}}$, where D is the overall number of Degrees of Freedom (DOFs) for the standard skeleton.
We seek to investigate the possibility of using the joint position or alternatively the joint angle representation of skeleton motion for training the DGHMM. The **joint positions**

[1] http://mocap.cs.cmu.edu/

include the joint trajectories as positions in the 3D Cartesian space. Following Holden et al. [2015], the global translations in the XZ-plane, and the global rotations around the Y-axis are removed, and instead, the velocities in the respective planes are appended to the motion clips as new DOFs[2]. The **joint angles** representation include the rotation of different body segments relative to their parent node in the skeleton hierarchy to reach the desired orientation from a zeroed initial setting. As clarified in Section 2.1.1 the angles are represented by the exponential map parametrization which yields the same numbers of DOFs for joint angle data type.

Using the descriptions accompanying the motion files obtained from the respective cited sources we assign a *loose label* to each extracted clip. The labels are not precise, since the recording session might be long, and the description might not always be applicable to the whole clip. For example several extracted clips from one clip of the CMU dataset might be labeled as *walk* following the description of the original clip, yet in fact, these clips might also include some turns, backward walking, jumps, sidewalking, etc. These labels are only used for later visualization purposes and are not fed to the network at any stage.

The input to the network are batches of motion clips, $M \times 1 \times T \times D$, where M is the batch size, T is the window size of the clip, and D is the total DOFs that describe the motion. In all the implementations $M = 256$, $T = 240$, and D depends on the chosen data type. For joint-positions data type $D = 66$, including a total of 60 DOFs for 21 joints, and 3 DOFs for the root velocities. For the joint angles data type $D = 69$, including 66 DOFs for 21 joints, 3 DOFs for the root positions, and 3 DOFs for the root velocities. Each channel of the training data is normalized by its respective mean and variance before being fed to the network. At the test time, the training mean and variance is used to normalize the input to the network and unnormalize the output from the network. In total, 20965 motion clips of $T = 240$, equal to roughly 23 hours with 15 labels were extracted: *cartwheel, dance, hop-both-legs, hop-one-leg, jump, jumping-jacks, kick, march, misc, punch, run, throw, walk, walk-on-place, waving-two-hands*. Ten percent of the motions within each class are randomly set aside for our test dataset, which makes the total count of its elements 2105 motion clips. We augment our training dataset by appending the time-flipped training motions, which doubles the training dataset size. The total training dataset size is 18860, and after augmentation, it reaches 37720 motion clips.

[2] 3 DOFs

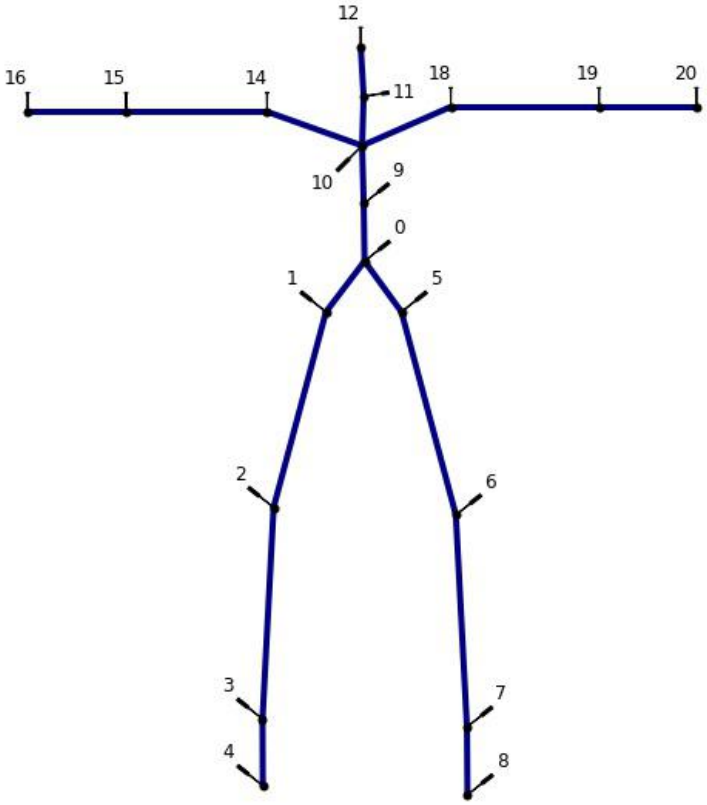


Figure 3.1: The standard skeleton in T-Pose. This is the standard human skeleton hierarchy that has been used for retargeting the human MoCap datasets. Joint names of the articulated skeleton are: *0: Hips (root), 1: LeftUpLeg, 2: LeftLeg, 3: LeftFoot, 4: LeftToeBase, 5: RightUpLeg, 6: RightLeg, 7: RightFoot, 8: RightToeBase, 9: Spine, 10: Spine1, 11: Neck1, 12: Head, 13: LeftShoulder, 14: LeftArm, 15: LeftForeArm, 16: LeftHand, 17: RightShoulder, 18: RightArm, 19: RightForeArm, 20: RightHand.*

3.2 Motion Visualization

Visualization of the human skeleton motion is done via stick figures produced in python scripts, Fig. 3.1. Different types of visualization techniques are used: **Full animated motions** in video files. For this, we used enhanced versions of the scripts originally obtained from Holden et al. [2015]. The enhancement makes the code suitable for visualization of the joint position and the joint angle representations, as well as making the Jupyter Notebook animations possible. In addition to the moving character movies, **Rotoscopic** and **Stroboscopic** plots, Fig. 1.1, that are shots at different time points of the animation, are presented to make the paper-printed visualizations of the animations possible. Moreover, plots of the **single joint trajectories** for precise inspection purposes will be provided.

3.3 Network Architecture

The DGHMM is essentially a Variational Autoencoder (VAE) with probabilistic interpretation of the encoder and the decoder, plus a regularization on the latent-code space, Section 2.5. This regularization is in the form of a Kullback-Leibler Divergence (KL Divergence) between a normal distribution and the distribution of the latent-code. Fig. 3.2 shows the overall visualization of the architecture of the network[3]. The encoder includes a series of strided 1D temporal convolutions, Section 2.3.1, that compress the temporal dimensions of the input motion clips from the original 240 to 30 and two dense layers, Eq. (2.2), that throw temporal information away and enable computation of the non-temporal latent mean and variance required for the Variational Autoencoder (VAE) model. The decoder is symmetric to the encoder in that it has a dense layer to expand the dimensions for the next transposed convolutions, Section 2.3.2, which upsample their input from previous layers and recover the temporal information. Each convolution subsamples the temporal dimension by the factor two, and each transposed convolution upsamples the input also by the factor two. In total, the final network has 8 layers and the latent-code vector size is 100 dimensions. A comprehensive explanation of the reason behind the choice of the manifold size is given in Section 4.1. For further discussion on other hyperparameter choices, refer to Appendix B. Activation of each layer is normalized with Batch Normalization (BN) technique, presented in Section 2.3.3.

3.3.1 Optimization Objective

The main unsupervised objective is to reconstruct the input after passing it through the network. To keep the network from overfitting, or learning the identity function[4], three different regularization components are included: First, regularization of the weights to keep them small and sparse. Second, the corrupting process of the input is considered a regularization which enforces the network to learn *useful features* [Vincent et al., 2010]. Third, the KL Divergence of the latent-code from a Normal distribution which forces the encoder to create a code that is very close to samples drawn from a Normal distribution. The total loss of the network is given in Eq. (2.15). Similar to Lüdi et al. [2017][5], we introduce a variable α to weight the reconstruction versus the regularization, that is by using a different $\alpha \in [0, 1]$ value to weight the objective towards better reconstruction or more regularization of the latent-code. We start from α zero in initial steps, to enforce learning a better reconstruction, and gradually increase α value towards one, to regularize the latent-code, and to push its distribution closers towards a Normal distribution. Especially when having small latent-code size, this gradual training is

[3]For an in-depth visualization of the DGHMM architecture refer to Appendix A.

[4]Thus just copying the input to the output

[5]This paper was published during final stages of our work, and while sharing some similarities, it is only intended for the single task of walking and therefore can be seen as a special case of our results. Nevertheless, they also use the procedure with varying α, correctly suggested to better separate dimensions on the manifold, where the original idea is referred to Rezende et al. [2014].

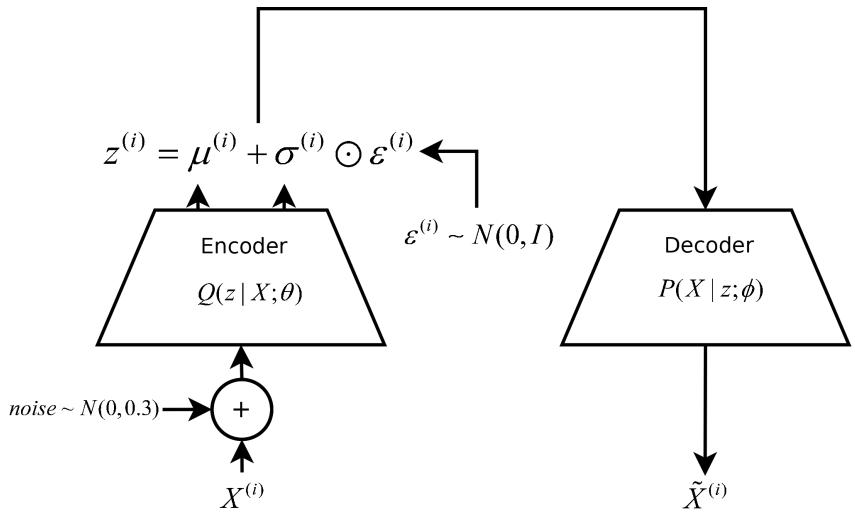


Figure 3.2: The Deep Generative Human Motion Model (DGHMM). The proposed Deep Generative Human Motion Model (DGHMM) is essentially a VAE with an additive Gaussian noise applied to the input. The input noise is to ensure that the network learns useful features in different levels of its hierarchy via this secondary regularization criterion. The output of the encoder has lower temporal dimensions than the input, which is later flattened by a dense layer to produce the inferred mean and variance. Since the reparametrization trick, Eq. (2.16), a sample from a Normal distribution is added to the encoder outputs and next it is fed to the decoder as the latent-code. The decoder uses this code to reconstruct the input. After the training, the generator (decoder) can be given samples from a Normal distribution and produce realistic human motions that are very similar to the training dataset but not exactly.

observed to be important in having a good variety of the final samples of the generative model. During the gradual training, the model without BN struggles to converge, since the KL loss grows very large, and even diverges, at first epochs when the *alpha* is set to zero and the model parameters are optimized to reach the best reconstruction only. When BN is applied, the divergence of KL loss at first steps of the training with $\alpha = 0$ is not sever and it recovers at later epochs with increasing α values.

3.4 Training

The training of the network is done with Adam [Kingma and Ba, 2014], on a single NVIDIA GeForce 980Ti graphics card with 6GB memory. All the implementations are done in python and the Tensorflow framework [Abadi et al., 2016]. During the training, the dataset is shuffled for each epoch and the data for each batch is selected randomly with no preference given to any motion class [Bengio, 2012]. The training is done until a fixed amount of iterations is reached and the test set reconstruction loss has reached its plateau. Then we choose the model snapshot with the lowest test set reconstruction error, for which the samples from the generative model are visually appealing. Since we had no quantitative measure to assess the perceptual quality of the generated samples we

had to manually check for that. It is important to note that contrary to Holden et al. [2015] we don't follow a layerwise training procedure and the final model is trained in one stage and end-to-end.

Chapter 4

Results

In this chapter, we present the results regarding our attempt to build a Deep Generative Human Motion Model (DGHMM).

4.1 Manifold Size

To find the optimum size of the latent-code vector, D, we investigate the effects on reconstruction power of the model by changing D. For this purpose, we fix the network architecture and all the related hyperparameters and only change the latent-code dimensions by the desired length. Then we train the network for a fixed number of epochs[1] and report the best reconstruction loss achieved. Since the parameters of the model are randomly initialized we repeat the experiment and report the average of the individual best results, Fig. 4.1. The best convincing choice regarding the good reconstruction loss given the total parameters of the model is D=100. Another alternative to reconstruction loss would have been to do a quantitative analysis of the perceptual quality of the samples of the generative model, however to our knowledge such a measure is not yet well-established [Theis et al., 2015]; meanwhile the former is easily achievable. Additionally, it is intuitive to think that an efficient code should be able to include as much information as to initiate reconstruction of the original input.

[1] Number of the epochs were chosen large enough to ensure convergence of the models.

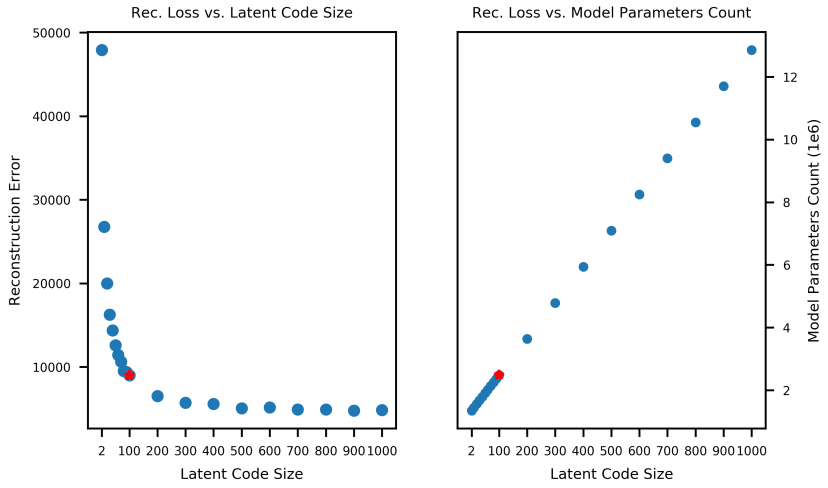


Figure 4.1: Size of the latent-code versus the reconstruction loss. The left plot shows the averaged best reconstruction loss achieved for five repetitions each for 400 epochs. The right plot shows a linear relationship between the size of the latent-code, D, and the count of the parameters of the model. The red point is D=100, which is a convincing choice regarding the good reconstruction loss given the total parameters of the model. Also, it is confirmed that this latent-code size is capable of generating trajectories in the final generative model.

4.2 Manifold Visualization

To visualize the organization of the data on the manifold of the DGHMM we can pass the data through the encoder and visualize the latent-code. Our proposed human motion manifold is a 100 dimensional vector, that obviously cannot be directly visualized in 2D or 3D space. To get a low-dimensional projection of the latent-code we first try the Principle Component Analysis (PCA) method [Jolliffe, 2014], even though we expect it to be ineffective due to the highly nonlinear structure of the manifold. In this case, PCA requires a high number of principle components, which are much more than what we need for visualization purposes. Fig. 4.2 shows the amount of variance in the data retained in case of a linear projection by PCA, which is inline with our hypothesis, and PCA proved to be indeed inadequate for visualizing this highly nonlinear manifold.

Next, we try the t-Distributed Stochastic Neighbor Embedding (t-SNE) method which is especially suitable for visualization of the high dimensional nonlinear data. Fig. 4.3 shows the t-SNE applied on the latent-code computed for the entire training dataset[2],[3]. In the accompanied DVD media, we provide an interactive version of Fig. 4.3 where each point is clickable and retrieves the corresponding motion video file. Two points should be noted in this visualization of the manifold: (1) Points in the top region correspond to motions including more of upper limb movements, and points in the bottom region show motions that include movement of both limbs. Motions in the center show a smooth

[2]Except the misc, and walk classes.

[3]The reason to choose the training dataset was simply because more samples make the point of the visualization more clear.

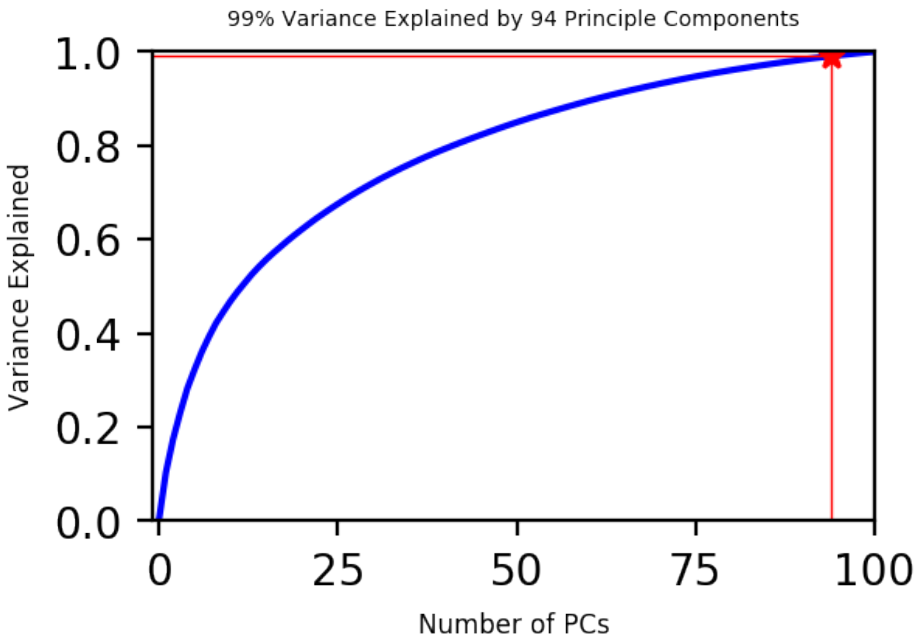


Figure 4.2: PCA applied on the latent-code. Variation of the elements in the vector of the Latent-Code, or the Manifold, is highly nonlinear, and a direct linear projection such as the PCA would be inadequate for visualization purposes. The figure shows PCA applied on the latent-code of the encoded test set. To keep 99% of the variance, 94 principle components or dimensions of the latent-code are needed, which doesn't fulfill our purpose for a low-dimensional visualization.

transition between the two extremes. (2) Although the labels are not precise, still clusters of semantically close motions do appear. We apply a simple k-Nearest Neighbors (k-NN) classification task on the latent-code which yields a 61.05% accuracy for the test set[4]. These observations might be effected by the internal mechanisms of the t-SNE method, therefore we investigate it more in Section 4.3.

[4]In a second experiment we trained the DGHMM with class-specific batches of data, so that each batch has elements of a single class whenever enough data is present, or it will sequentially go over other classes until the batch is full. This change is done to get a better cluster separation, which is also confirmed by the results. The k-NN classification yields 67.65% test set accuracy which is much higher than the shuffle mode that previously achieved 61.05% test set accuracy.

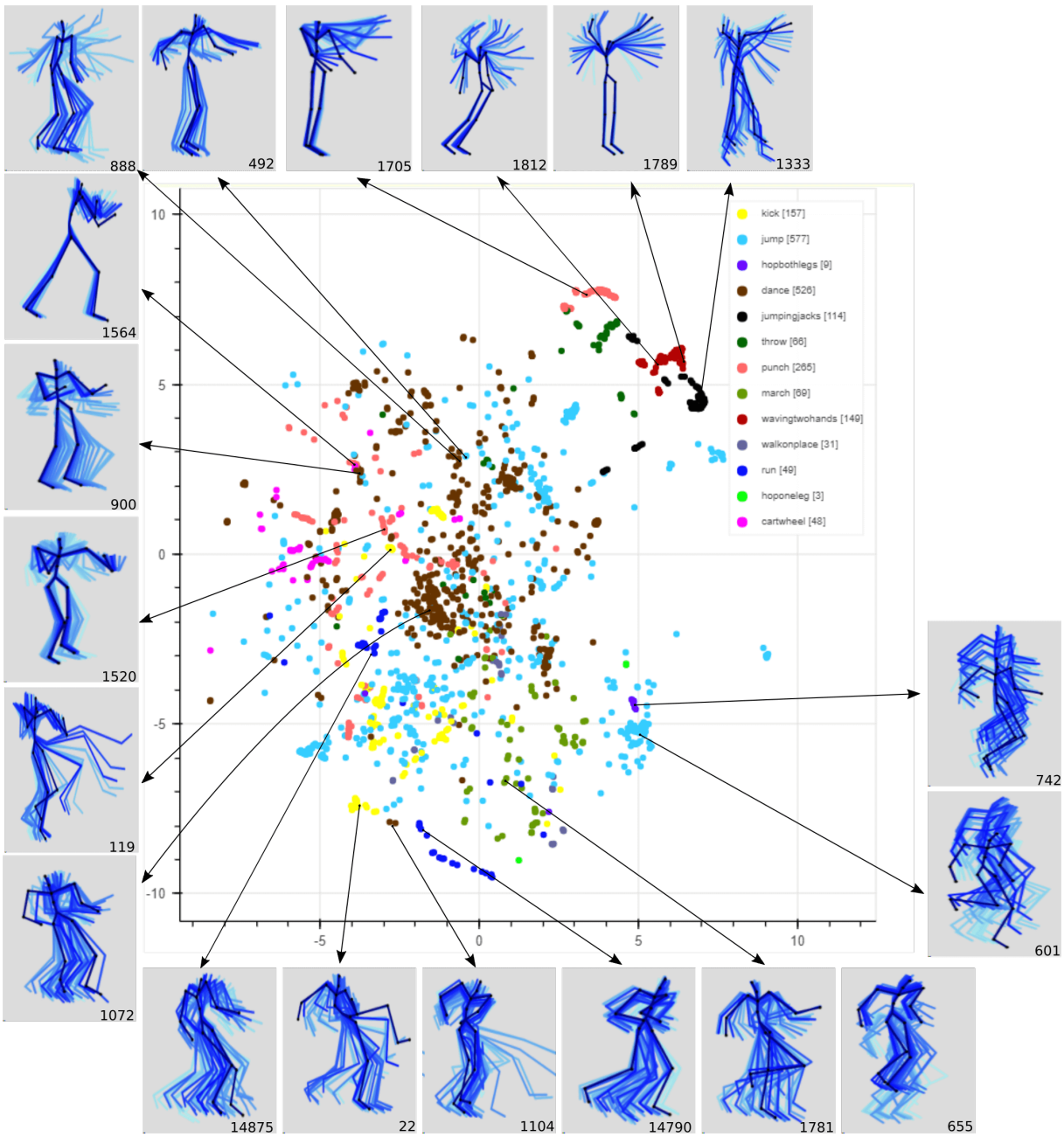
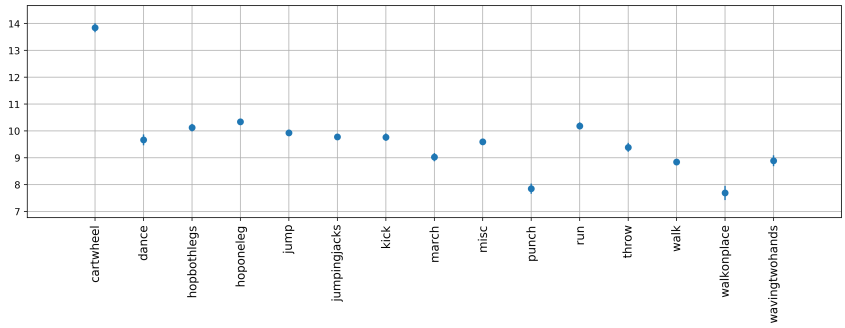


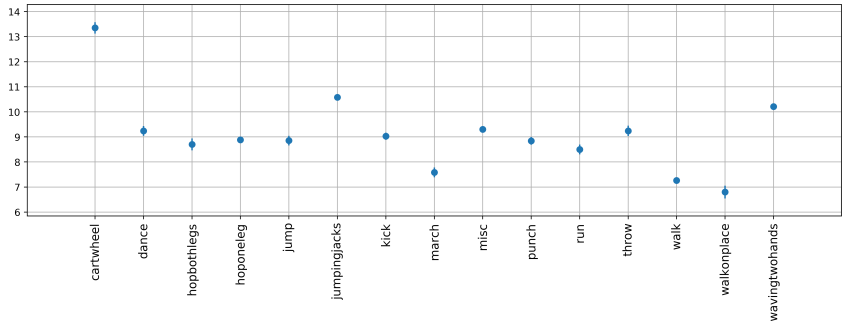
Figure 4.3: t-SNE applied on the latent-code. The coordinates of the points are computed by first encoding the data points in the training set, by passing them through the encoder, and then doing a nonlinear dimensionality reduction on this code. Each color represents the class of the original clip from which that data point is extracted from. The class misc is removed from the visualization since the definite class of its member motions is unknown. Additionally, the class walk is also not included since it includes various styles and contents which are making the visualization of the other more precise class labels difficult. The number in the lower corner of each rotoscopic figure can be used to fetch the video for that figure from the accompanying DVD media.

4.3 Retrieval Task Benchmark

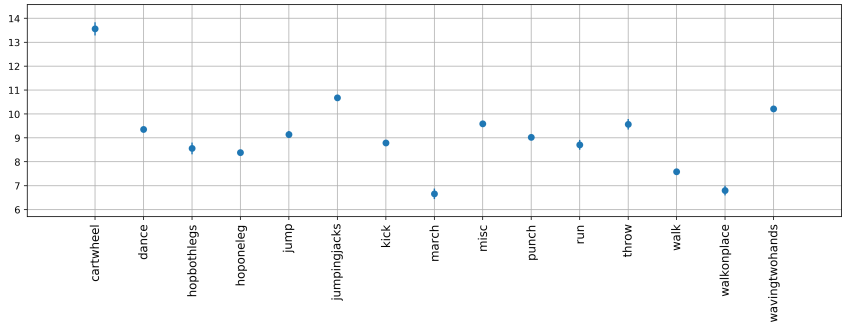
We aim to learn a low-dimensional embedding within the human motion time-series signal and use this manifold later in the motion generation task. Our training objective, namely reconstruction error, and the Kullback-Leibler Divergence (KL Divergence) is not a direct criterion for the usefulness of the features learned by the model and the structure of the manifold. For this reason, we seek a secondary task, that would be beneficial in benchmarking our manifold. For this purpose, we introduce the **Motion Retrieval Benchmark**: Latent-code of two motion data points of the same class shall be closer in L2 distance than the latent-code of the motions from completely different classes. This hypothesis is a direct result of the observation made in the Section 4.2. There it was observed that similar motions tend to be closer to each other in a low-dimensional visualization of the manifold, Fig. 4.3. Fig. 4.4 shows that this is, in fact, the case and the L2 distance of a specific class versus all the other classes shows a meaningful difference.



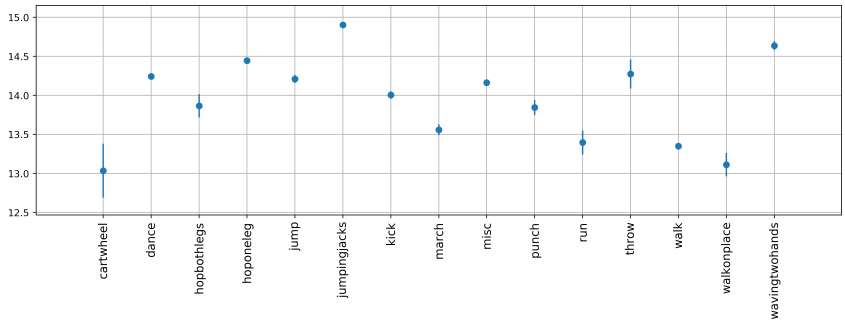
(a)



(b)



(c)



(d)

Figure 4.4: The retrieval task benchmark. Data points from the test set, corresponding to the same motion class, should have the closer L2 distance in the manifold space. Here we see the effectiveness of our manifold in this task for motions in various classes. Each subfigure is showing the L2 distance of a specific class versus all the other classes; note small distance for that specific class with itself: (a) punch, (b) walk, (c) march, (d) cartwheel action. Small distance with some other classes might be due to some intrinsic similarities between data points (motions) within those classes. The error bars show the variance of the computed measure across all the data points of that class.

4.4 Recovering the Encoded Trajectories

In a forward run of the Autoencoder (AE), the original trajectories of some motion data are encoded by the encoder to create a low-dimensional latent-code. The latent-code can then be decoded by the decoder to recover the original motion. Fig. 4.6 shows a sample motion that is first encoded and subsequently reconstructed by the decoder. Fig. 4.5 shows an stroboscopic view of an original motion and its reconstructed version after passing through the network. Fig. 4.6 shows the trajectories in detail. It should be noted that the training objective, Eq. (2.15), is not only to minimize the reconstruction error but to match the distribution of the latent-code to a Normal distribution. Therefore, the reconstructions might not be as perfect as the results achieved by a Denoising Autoencoder (DAE) as in Holden et al. [2015].

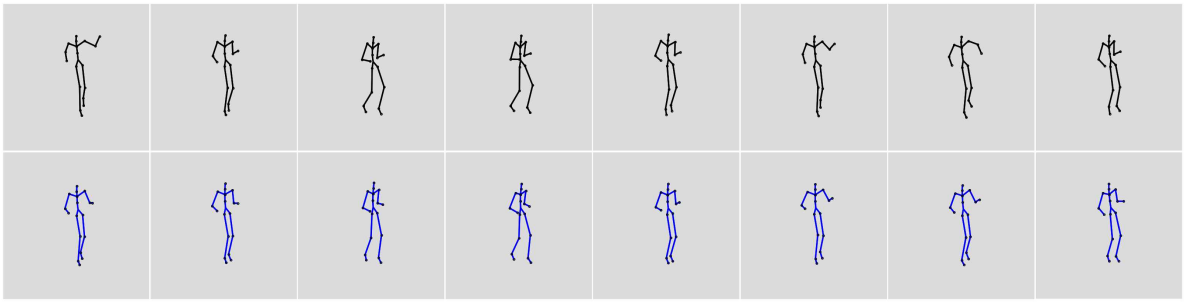


Figure 4.5: Human Motion reconstruction from the latent-code. We encode the original motion, a punch motion shown by the black character, to compute the latent-code value and subsequently decode that code to recover the original motion, shown by the blue character. For detailed trajectories of the joint refer to Fig. 4.6.

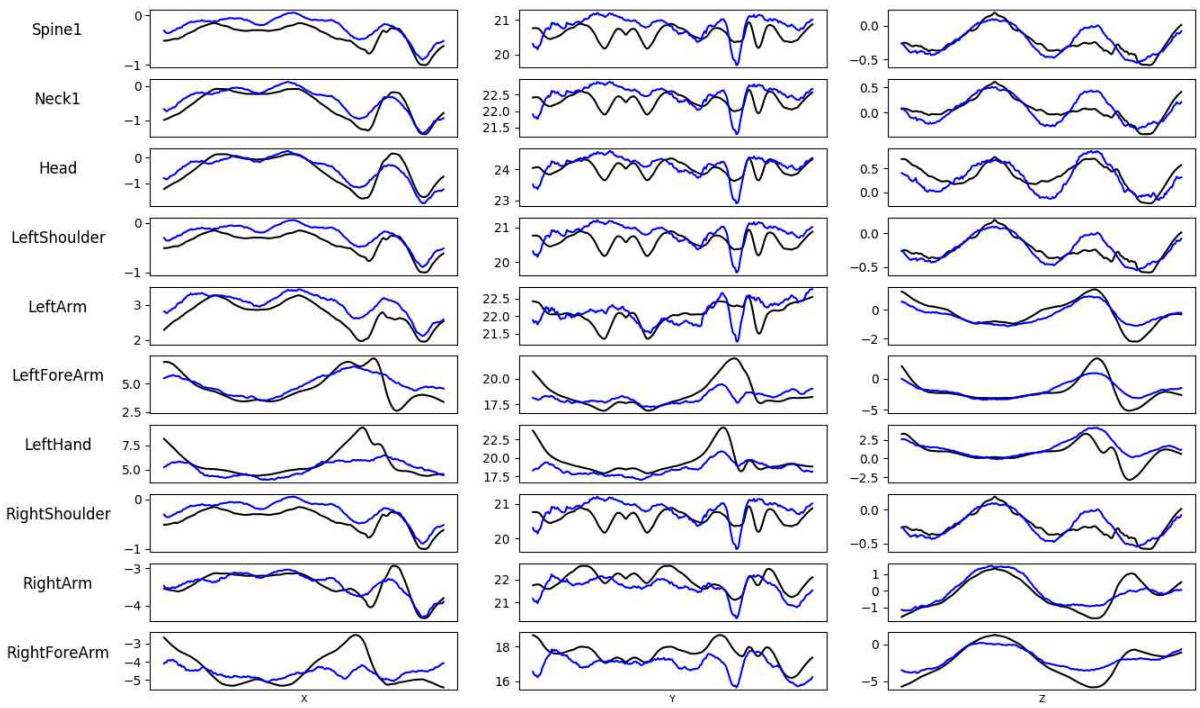


Figure 4.6: Recovering the Encoded Trajectories. The detailed trajectories for some joints corresponding to the motion in Fig. 4.5. The black line shows the original trajectories, and the blue line shows the reconstructed version. Since the best reconstruction is not the only objective of the training of the model, Eq. (2.15), the reconstructions are not as perfect as a DAE.

4.5 Human Motion Blending on the Manifold

First reported by [Holden et al., 2015] on a DAE, linear interpolation of different motions is also possible on the latent-code of our DGHMM. For this purpose, we first encode two different data points X_1, and X_2 by doing a forward pass through the model's encoder. Then we use a mixture coefficient "α" to do a weighted sum of the corresponding latent-codes Z_1, and Z_2, and get the target code Z_{target}. More specifically:

$$Z_{target} = (1 - \alpha)Z_1 + \alpha Z_2 \tag{4.1}$$

After we pass the Z_{target} through the decoder we get a motion *in-between* two original motions. By varying the mixture coefficient, we can linearly choose the amount of style and content from different motions. Fig. 4.7 shows interpolation of two motions across different content classes. Additionally, Fig. 4.8 shows the interpolation of motions within the same class while having different styles. We observe that in both cases by varying the coefficient, α one can smoothly transfer from one motion to the other.

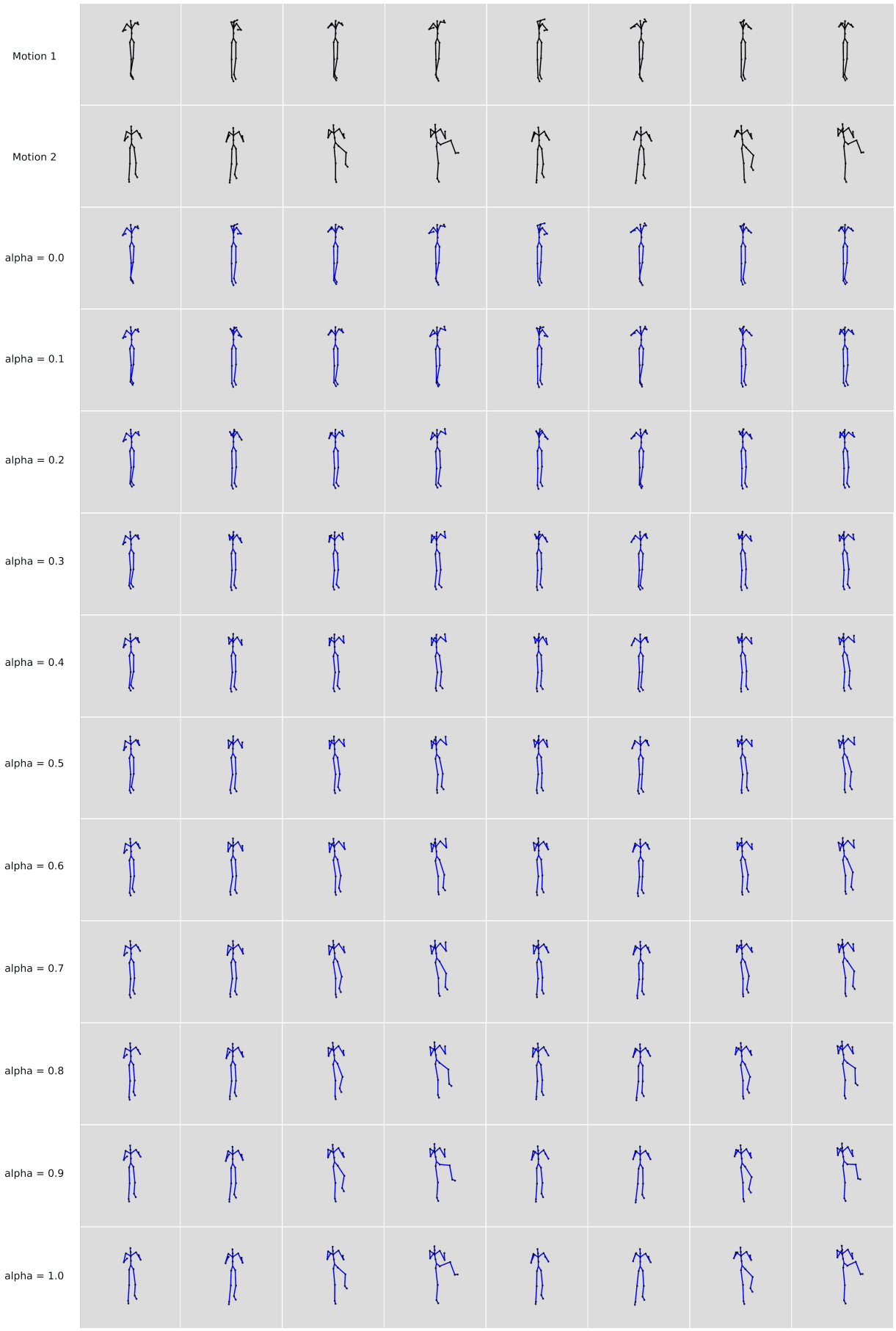


Figure 4.7: Human motion content interpolation on the manifold.

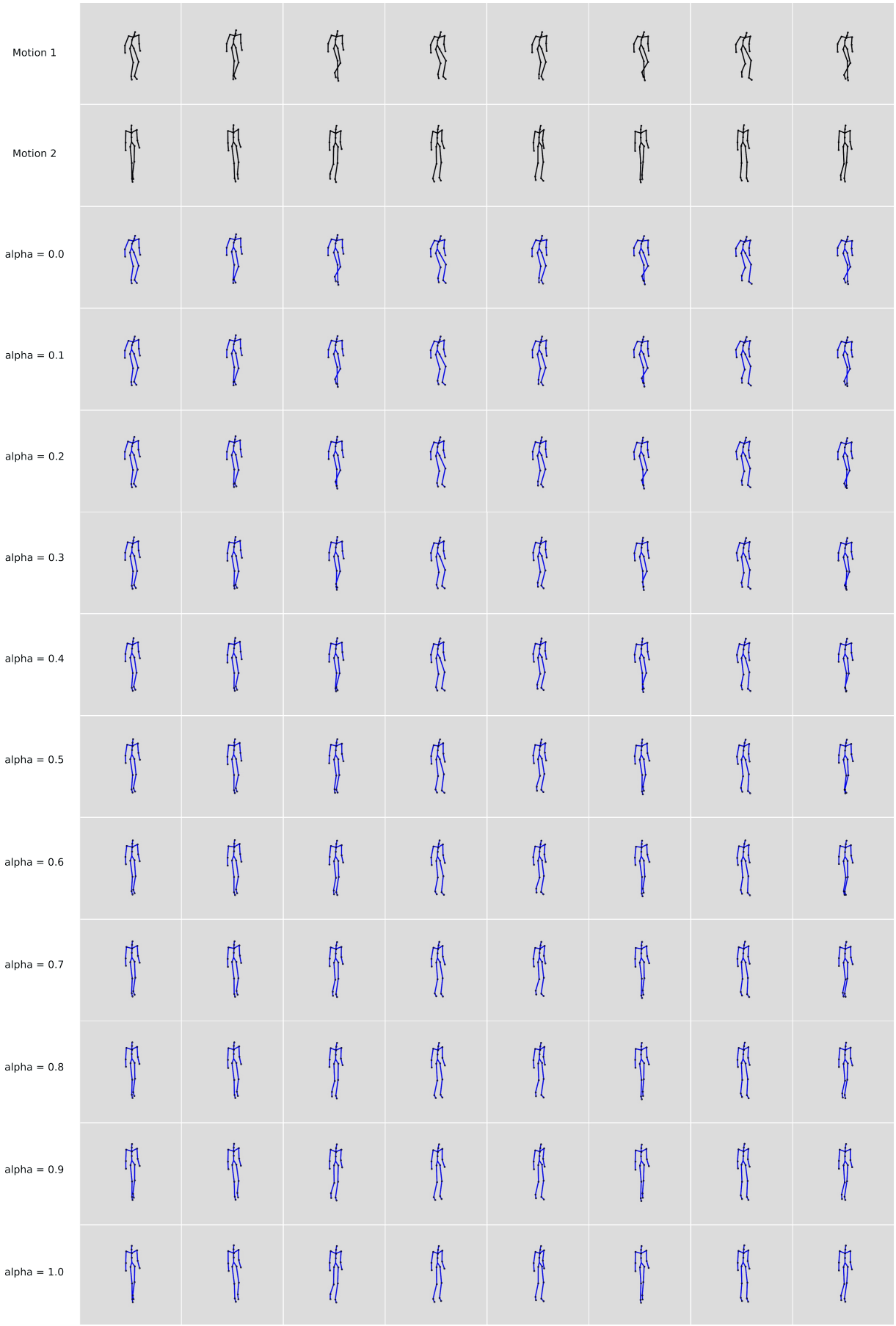


Figure 4.8: Human Motion style interpolation on the manifold.

4.6 Samples from the Generative Model

Sampling from the DGHMM is easily done by first sampling from a 100 dimensional normal distribution and then passing this sample through the decoder to get a novel motion that is close to the training data points but not exactly. Fig. 4.9 shows some samples from a fully trained model. More samples can be found in the accompanying DVD media.

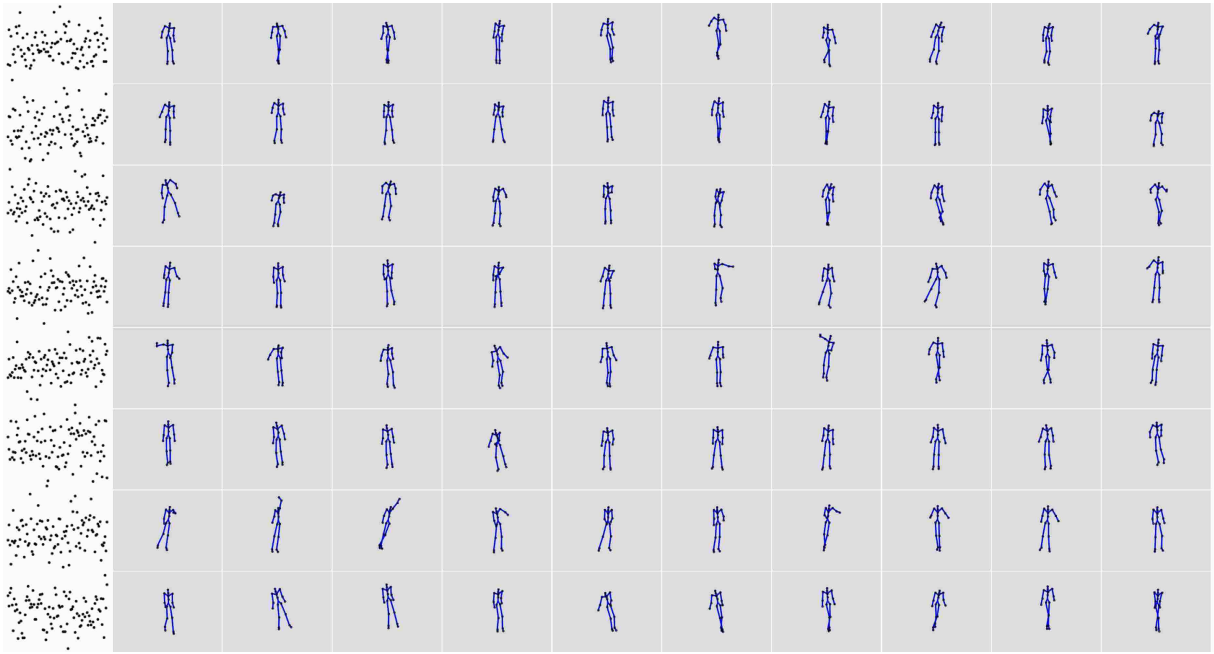


Figure 4.9: Samples from the DGHMM. Black dots qualitatively show the noise samples from the 100 dimensional Normal distribution that is used as a code for the decoder of the AE. The stick figures to the right show stroboscopic display of the animation created by the generative model.

4.7 Naturalness of the Generated Human Motions

To asses the naturalness of the randomly generated samples of the model we conducted a psychophysical experiment, in which participants had to rate the perceived naturalness of the human motions that they observed. Ten subjects[5], naive with respect to the purpose of the study, participated in the experiment. Stimuli were videos of human skeleton stick figures (Fig. 1.1) performing various random actions. Any other details in the scene, such as lighting or the surface plane, were removed to avoid other visual cues other than the actual character movement. All the motions were visualized with python scripts, extended from the published code by Holden et al. [2015]. Presentation of the stimulu and the process of the experiment were handled by our custom programmed python code. In the accompanying DVD media, the videos used in the experiment are included.
The full motions set included 15 samples from each of the following categories: generated

[5]All subjects were adults, older than 25 years old.

random samples from the model, randomly selected motion capture clips from the miscellaneous class of the the test set[6], and linearly blended motions[7] from randomly selected pairs of the previous category. The naive blending of the previously recorded motions was considered as our baseline in producing human motion. A single clip repeated three times in random order throughout the experiment. Each stimulus was repeated twice before the participant was asked to rate naturalness of the observed human motion. The ratings were done on a Likert scale ranging from 1(most likely artificial) to 5(most likely natural). The data was first statistically analyzed by One-Way ANOVA to check for categorical differences, and subsequently the positive difference was attributed to specific categories using post-hoc Tukey's test. Fig. 4.10 shows the results of the experiment.
The ANOVA test suggests the existence of significant statistical difference among the category means ($p=0.0007$). The results of the post-hoc test rejects the null hypothesis at the significance level of $p< 0.01$ for the pairs of "generated motions-original motions", and the "generated motions-blended motions".
In conclusion, these results indicate that the random samples driven from the DGHMM are not perceived as natural as the other categories, even though the joint covariations and the human skeleton hierarchy are abided in the generated samples. A deeper discussion on these results is given in Chapter 5.

[6]Motions in the miscellaneous have no defined category of motion, hence the name.

[7]Blending is done frame-by-frame where each frame represented joint positions in 3D cartesian space. Note that transformations on that XZ-plane as well as the rotations around the Y-axis had been already removed, Section 3.1.

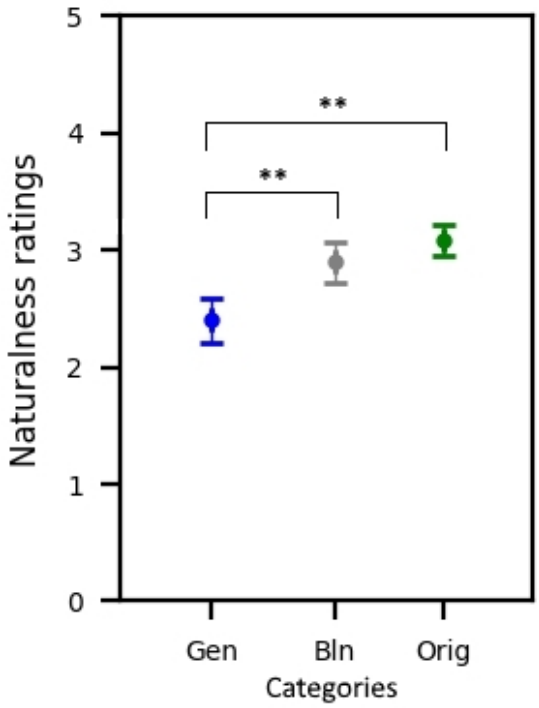


Figure 4.10: Naturalness of the samples from the DGHMM. The category of the motions are as following: "Gen" motions are generated randomly as samples from the DGHMM; "Bln" motions are naive blends of pairwise motions selected from the motions in the next category; "Orig" motions are randomly selected motions from our test dataset including original motions recorded from the human subjects. The results of this psychophysics analysis suggest that the random samples from the DGHMM are not as naturally perceived as the members of the other two categories. Error bars show the standard errors, with the asterisks indicating the significance level at which the post-hoc analysis rejected the null hypothesis for that pair of categories (** $p < 0.01$, Tukey's Test).

4.8 Manifold of the Joint Angles

As stated before, one of the achievements of this thesis is to provide an alternative representation to the joint position representation provided by the Holden et al. [2015]. That is the joint angle representation where the rotation of angles between the body segments is parametrized by the Exponential Map parametrization [Grassia, 1998]. By this way, essentially the total count of the Degrees of Freedom (DOFs) is kept similar to the joint position representation and all the training procedures and networks can be kept the same. Fig. 4.11 shows some samples from our joint angle generative model. It is obvious that the model performance is tremendously reduced and some inner workings should be altered to make the model work in this mode. One cue point would be the problems regarding the jumps in angle values represented by exponential maps which can be resolved by heuristic unwrapping of the angle values before presenting to the network. Another potential improvement direction would include angle velocities instead of absolute values. Using the joint angle representation for the training of the DGHMM is still open

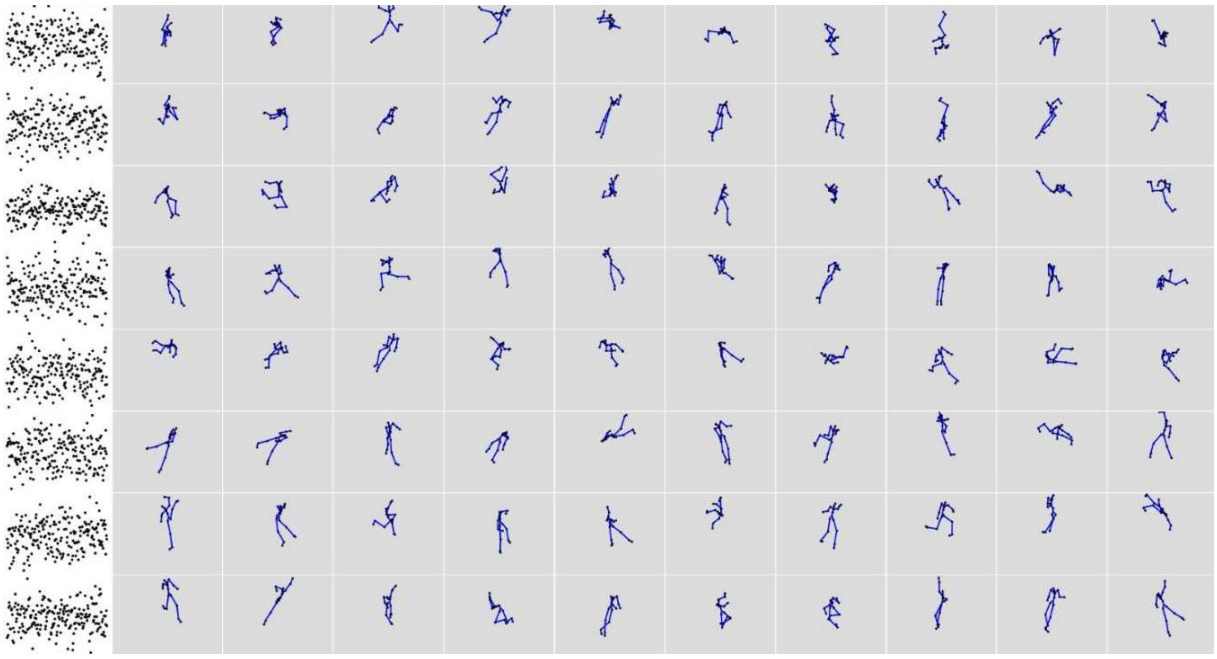


Figure 4.11: Samples from the joint angle generative model. Here the latent-code is 200 dimensional and the black dots qualitatively show the noise samples from the 200 dimensional Normal distribution that is used as the code for the decoder of the Variational Autoencoder (VAE). Compare the generated motions to those in Fig. 4.9.

for future research.

Chapter 5

Discussion

To understand how humans move the way they do, one logical way would be to try to artificially reproduce their motion[1]. The goal of this thesis is to make an attempt in this direction; that is to make a generative model of the trajectories naturally produced by the joints of an articulated human skeleton, or the human motion signal[2].

Our Deep Generative Human Motion Model (DGHMM) demonstrates a novel application of feed-forward Deep Neural Networks (DNNs) in Variational Autoencoder (VAE) configuration [Kingma and Welling, 2013] for learning a generative distribution over human motion time-series signal across various styles and contents. A sample from the DGHMM, $P_g(X)$, gives a time-series of plausible pose configurations of a human skeleton within a certain duration. We benefited from the idea of temporal convolutions for human motion data developed by Holden et al. [2015], by which they trained a Denoising Autoencoder (DAE) to learn a blind low-dimensional representation of human motion signal[3]. Similar to their results, our model is also capable of learning a low-dimensional representation of the human motion time-series signal, interchangeably called a motion manifold. Additionally, we put extra attention on the structure of the manifold by further regularizing its distribution to be as close as possible to a multi-dimensional Normal distribution. This regularization is important for making the model a generative one so that it can generate novel random motions by feeding the decoder a sample from a white noise generator.

A single data point for training the DGHMM is a window of human motion data, and the temporal convolutions are the same as the commonly known spatial convolutions in the computational vision field. These choices enable us to directly benefit from the massive recent advances in vision-based DNNs for development of our model. For example, we use strided convolutions for the temporal downsampling and the transposed convolutions for the temporal upsampling. Even the VAE idea is originally applied on the image data [Kingma and Welling, 2013], and we extend it to the natural human motion signal.

[1] "What I cannot create, I do not understand." - Richard Feynman

[2] To understand what we refer to as "human motion signal" refer to Chapter 1.

[3] The low-dimensional representation produced by the network in Holden et al. [2015] still had 5120 dimensions. Our low-dimensional representation has only 100 dimensions.

The main results are achieved on the joint position representation, however, we also present an attempt on the joint angle data as a stepping stone for future works. Additionally, we provide labels for the motions in the dataset by extracting them from the description given in the original source. By using these labels we do a visualization of the learned nonlinear manifold, that demonstrates the formation of clusters of semantically similar motions on the manifold. A simple k-Nearest Neighbors (k-NN) classification task on the latent-code yields a 61.05% accuracy for the test set with 15 motion classes, which is above the chance level. Based on these observations we also investigate the performance of a similarity measure on the elements of the test set, which confirms the effectiveness of our model in unsupervised clustering of the human Motion Capture (MoCap) data. The measure is called the retrieval task benchmark, and it assumes that motions with similar contents shall have closer L2 distance on the manifold, than the ones with completely different contents. As indicated by the results in Section 4.3 this assumption is suggested to be correct.
Finally, we show that semantic manipulation of the human motion signal is possible on the latent-code of the model. In Section 4.5 we demonstrate this capability of the manifold for smooth blending of content and style of different recorded motions. This observation is line with our manifold assumption of the latent-code; in other words, by passing the motions through the encoder, the mapped points will be on a nonlinear manifold which lies on the local planes of the most important variations of the original data. Therefore, when we simply interpolate between the latent-code of completely different motions the final code will represent a motion that involves an interpolation between most salient parts of the motions and therefore the blending is valid and smooth.

5.1 Future Work

Our DGHMM is useful in various fields such as Virtual Reality (VR) setups, robotics, and active human body prosthesis to name a few. In a VR setup the control subsystem does not need to compute the constrained whole body trajectories of the avatar, but only some high-level intention signal such as the end effector positions for a kick or a punch, and the rest will be handled automatically by the presented DGHMM. In this direction, we are able to extend the results of the Holden et al. [2016] to an interactive motion, especially useful for rehabilitation tasks. The motion is to do throwing and catching a ball, while character's full body joint trajectory is produced in response to a section of the trajectory of the ball. Following Holden et al. [2016] this network has two parts: a manifold part, that is trained on lots of available human MoCap data with the DAE procedure, and a regressor part for regressing from the task parameters that has been trained on an order of magnitude smaller dataset from in-house recorded motions of throwing and catching along with the trajectory of the ball. This model is fully convolutional, therefore it can produce any throwing and catching motion with the desired length and no need for any

post-processing step or re-training of the model. Converting this model into a generative one that fits in the context of the present work requires more effort; therefore, it is not included in the formal results.

Another beneficial future attempt would be to include inverse kinematics and produce joint angles which then makes our model directly portable to robotic platforms. Our preliminary results in Section 4.8 suggest that the joint angle data are more sensitive to noise in the produced data of the generative model, and perfect results still require more experiments.

Another possibility for future research could be to use an adversarial training procedure instead of the Kullback-Leibler Divergence (KL Divergence) to match the distribution of the latent-code to a known prior. This choice gives more flexibility for choosing the prior distribution[4], while matching the whole distribution of the latent-code to the whole prior distribution and not just the first two moments of the distribution [Makhzani et al., 2015]. We also did experiments in this direction, however the results did not show much improvement given the extra work of training three networks in the Adversarial Autoencoder (AAE) mode instead of our two networks[5]. Later implementations might seek benefits by training AAE networks instead of VAEs.

5.2 Naturalness of the Generated Samples

To estimate the performance of our generative model we checked for the quality of the samples, classification performance of the k-NN on the latent-code[6], and the reconstruction error of the model given the latent-code. However, these measures do not directly assess the naturalness of the samples driven from our generative model. In this sense, our research much like other attempts to generate natural signals can benefit from a long-wanted quantitative criterion to estimate the perceptual quality of the generated samples [Theis et al., 2015]. Meanwhile, we benchmark the naturalness of the samples of our model against the best available judge for naturalness, that is the human perception. For this purpose, in Section 4.7 we conducted a psychophysics experiment, comparing the score of naturalness given by the human subjects to samples from three motion categories; namely, random samples from the model, original human MoCap data, and as our baseline randomly naive blending of pairs of motions from the previous category.

The results presented in Section 4.7, demonstrate that even though random samples from the DGHMM are rated on average above the purely artificial score[7], still they are perceived significantly less natural than the other two motion categories namely the originally recorded motions and randomly blended ones. This observation can be justified by two reasons: first, the 100 dimensional space of the manifold is still large and provides

[4]Since we do not need to know the exact formulation of the prior as long as we can sample from it.

[5]The Encoder and the Decoder in an Autoencoder network.

[6]Section 4.3

[7]Rating was done on a Likert score range from 1(very artificial looking) to 5(very natural looking), while our model achieved mean score of 2.40.

much freedom to produce motions that might be outside of purely natural looking human motion. This issue can be addressed by providing other means to regularize the latent-code, for example by providing foot contact labels to avoid the foot sliding effect seen in the generated human motions[8]. Second, our training dataset is generated from huge amounts of MoCap data gathered from various datasets, for which we did not have the manpower to prune for problematic elements. We estimate, based on visually inspecting handful of samples, that around five percent of the whole training dataset has some form of corrupted motion, where the corruption might have been in the original recordings or originated in any stage of the dataset preparation procedure such as the skeleton retargeting stage. Regarding the time limit of our project and the already achieved results we avoided any further investment in cleaning the dataset from these problematic samples. Nevertheless, we expect that with a cleaner training dataset, the perceptual quality of the samples from the model should in principle increase.

We believe that since the modeling of human motion over these amounts of data contents is still at its preliminary stages, even though the results are rated above purely unnatural motion, still further advances in this field can potentially bring the quality of samples closer to naturalistic human joint movements.

5.3 Conclusion

We presented a Deep Generative Human Motion Model (DGHMM) capable of producing human motion time-series signal in the joint position representation. The low-dimensional nonlinear manifold of the human motion signal learned by this model is especially useful for various tasks including generating novel motions from random vectors, unsupervised classification of motions, and smooth blending of motions across different contents or styles.

While our model is specifically implemented to handle time-series human motion data, yet the proposed model and the ideas behind it, are potentially applicable to any non-motion high-dimensional continuous time-series signal. This work demonstrates the ability of deep non-recurrent neural networks in handling time-series signals that are usually done with recurrent neural networks.

This thesis demonstrates that deep networks of artificial neurons that are adequately regularized are capable of generating human motions from white noise, that look similar to motions produced by our nervous system. The primitives in our model, which are the time-invariant convolutional kernels learned by the backpropagation method, are mixed in time to generate the final full-body joint trajectories. Future attempts in adopting the presented work in robotics and VR setups could be beneficial.

[8] Foot sliding of the animated skeleton was reported by the participants to be the major clue that they used to detect the artificially generated motions.

Bibliography

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*, Mar. 2016. 25

G. Alain and Y. Bengio. What Regularized Auto-Encoders Learn from the Data Generating Distribution. *arXiv:1211.4246 [cs, stat]*, Nov. 2012. 15, 16

O. Arikan and D. A. Forsyth. Interactive Motion Generation from Examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 483–490, New York, NY, USA, 2002. ISBN 978-1-58113-521-3. doi: 10.1145/566570.566606. 4

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv:1206.5533 [cs]*, June 2012. 25

Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized Denoising Auto-Encoders as Generative Models. *arXiv:1305.6663 [cs]*, May 2013. 14, 15, 16

A. Bissacco. Modeling and Learning Contact Dynamics in Human Motion. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 421–428, Washington, DC, USA, 2005. ISBN 978-0-7695-2372-9. doi: 10.1109/CVPR.2005.225. 5

R. Bowden. *Learning Statistical Models of Human Motion*. 2000. 1

S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating Sentences from a Continuous Space. *arXiv:1511.06349 [cs]*, Nov. 2015. 17

M. Brand and A. Hertzmann. Style Machines. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 183–192, New York, NY, USA, 2000. ISBN 978-1-58113-208-3. doi: 10.1145/344779.344865. 5

L. Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, pages 1–17, 2005. 15

A. Elgammal and C. S. Lee. *The Role of Manifold Learning in Human Motion Analysis*. 2008. iii, 1

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *PMLR*, pages 249–256, Mar. 2010. 53

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. 2016. http://www.deeplearningbook.org. 11

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. 20

F. S. Grassia. Practical Parameterization of Rotations Using the Exponential Map. *Journal of Graphics Tools*, 3(3):29–48, Jan. 1998. ISSN 1086-7651. doi: 10.1080/10867651.1998.10487493. 8, 39

A. Graves, A.-r. Mohamed, and G. Hinton. Speech Recognition with Deep Recurrent Neural Networks. *arXiv:1303.5778 [cs]*, Mar. 2013. 2

K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. DRAW: A Recurrent Neural Network For Image Generation. *arXiv:1502.04623 [cs]*, Feb. 2015. 17

Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187(Supplement C):27–48, Apr. 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2015.09.116. 12

S. S. Haykin. *Neural Networks and Learning Machines*. Upper Saddle River, 2009. ISBN 978-0-13-147139-9. OCLC: 865163557. 11

K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, Feb. 2015a. 53

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015b. 53

G. E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11(10):428–434, Oct. 2007. ISSN 1364-6613. doi: 10.1016/j.tics.2007.09.004. 1, 12

D. Holden, J. Saito, T. Komura, and T. Joyce. Learning Motion Manifolds with Convolutional Autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, SA '15, pages 18:1–18:4, New York, NY, USA, 2015. ISBN 978-1-4503-3930-8. doi: 10.1145/2820903.2820918. 2, 6, 13, 16, 21, 22, 23, 26, 33, 34, 37, 39, 41, 56, 57

D. Holden, J. Saito, and T. Komura. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.*, 35(4):138:1–138:11, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925975. 2, 5, 6, 9, 13, 21, 42, 56, 57

D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148:574–591, Oct. 1959. ISSN 0022-3751. 12

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, Feb. 2015. 13

I. Jolliffe. Principal Component Analysis. In *Wiley StatsRef: Statistics Reference Online*. 2014. ISBN 978-1-118-44511-2. doi: 10.1002/9781118445112.stat06472. 28

D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Dec. 2014. 15, 19, 25, 53

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, Dec. 2013. iii, 2, 10, 17, 18, 19, 41

D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling. Semi-Supervised Learning with Deep Generative Models. *arXiv:1406.5298 [cs, stat]*, June 2014. 3

L. Kovar and M. Gleicher. Automated Extraction and Parameterization of Motions in Large Data Sets. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 559–568, New York, NY, USA, 2004. doi: 10.1145/1186562.1015760. 4

L. Kovar, M. Gleicher, and F. Pighin. Motion Graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ISBN 978-1-58113-521-3. doi: 10.1145/566570.566605. 4

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012. 2, 56

T. D. Kulkarni, W. Whitney, P. Kohli, and J. B. Tenenbaum. Deep Convolutional Inverse Graphics Network. *arXiv:1503.03167 [cs]*, Mar. 2015. 17

N. Lawrence. Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data. In *In NIPS*, 2004. 5

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.*, 1(4):541–551, Dec. 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. 12

J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive Control of Avatars Animated with Human Motion Data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 491–500, New York, NY, USA, 2002. ISBN 978-1-58113-521-3. doi: 10.1145/566570.566607. 4

Y. Li, T. Wang, and H.-Y. Shum. Motion Texture: A Two-level Statistical Model for Character Motion Synthesis. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 465–472, New York, NY, USA, 2002. ISBN 978-1-58113-521-3. doi: 10.1145/566570.566604. 5

C. K. Liu, A. Hertzmann, and Z. Popović. Learning Physics-based Motion Style with Nonlinear Inverse Optimization. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1071–1081, New York, NY, USA, 2005. doi: 10.1145/1186822.1073314. 5

M. Lüdi, M. Guay, B. Mcwilliams, and R. W Sumner. Automatically Learning an Intuitive Animation Interface from a Collection of Human Motion (PDF Download Available). 2017. 24

L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. Auxiliary Deep Generative Models. *arXiv:1602.05473 [cs, stat]*, Feb. 2016. 17

A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial Autoencoders. *arXiv:1511.05644 [cs]*, Nov. 2015. 20, 43

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec. 1943. ISSN 0007-4985, 1522-9602. doi: 10.1007/BF02478259. 10

T. Mukai and S. Kuriyama. Geostatistical Motion Interpolation. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1062–1070, New York, NY, USA, 2005. doi: 10.1145/1186822.1073313. 4

M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation Mocap Database HDM05. Technical report, Universität Bonn, June 2007. 21

V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. ISBN 978-1-60558-907-7. 10

H. Narayanan and S. Mitter. Sample Complexity of Testing the Manifold Hypothesis. In *Advances in Neural Information Processing Systems 23*, pages 1786–1794. 2010. 15

F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy. Berkeley MHAD: A comprehensive Multimodal Human Action Database. In *2013 IEEE Workshop on*

Applications of Computer Vision (WACV), pages 53–60, Jan. 2013. doi: 10.1109/WACV.2013.6474999. 21

S. I. Park, H. J. Shin, and S. Y. Shin. On-line Locomotion Generation Based on Motion Blending. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, pages 105–111, New York, NY, USA, 2002. ISBN 978-1-58113-573-2. doi: 10.1145/545261.545279. 4

V. Pavlovic, J. M. Rehg, and J. MacCormick. Learning Switching Linear Models of Human Motion. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 942–948, Cambridge, MA, USA, 2000. 5

A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv:1511.06434 [cs]*, Nov. 2015. 14

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. 2005. ISBN 978-0-262-18253-9. 6

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv:1401.4082 [cs, stat]*, Jan. 2014. 24

H. Robbins and S. Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951. ISSN 0003-4851. 10

C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, Sept. 1998. ISSN 0272-1716. doi: 10.1109/38.708559. 4

F. Rosenblatt. *The Perceptron: A Perceiving and Recognizing Automaton (Project PARA). Report No. 85-460-1*. 1957. 10

S. T. Roweis and L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, Dec. 2000. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.290.5500.2323. 16

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct. 1986. ISSN 0028-0836. doi: 10.1038/323533a0. 6, 10

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319, July 1998. ISSN 0899-7667. doi: 10.1162/089976698300017467. 16

N. Taubert, M. Löffler, N. Ludolph, A. Christensen, D. Endres, and M. A. Giese. A Virtual Reality Setup for Controllable, Stylized Real-time Interactions Between Humans and

Avatars with Sparse Gaussian Process Dynamical Models. In *Proceedings of the ACM Symposium on Applied Perception*, SAP '13, pages 41–44, New York, NY, USA, 2013. ISBN 978-1-4503-2262-1. doi: 10.1145/2492494.2492515. 5

N. Taubert, D. Endres, and M. Giese. Reactive Virtual Reality Avatar with Controllable Emotional Style based on Hierarchical Gaussian Process Dynamical Models. In *In Proc. ICANN 2014*, page Art.No.25, 2014. 5

G. W. Taylor and G. E. Hinton. Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1025–1032, New York, NY, USA, 2009. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553505. 6

M. Telgarsky. Benefits of depth in neural networks. *arXiv:1602.04485 [cs, stat]*, Feb. 2016. 53

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, Dec. 2000. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.290.5500.2319. 16

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv:1511.01844 [cs, stat]*, Nov. 2015. 27, 43

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390294. 14, 15

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.*, 11:3371–3408, Dec. 2010. ISSN 1532-4435. 2, 10, 14, 17, 24, 57

J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models. In *NIPS*, volume 18, page 3, 2005. 5

J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2): 283–298, Feb. 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1167. 5

K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a Kernel Matrix for Nonlinear Dimensionality Reduction. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, pages 106–, New York, NY, USA, 2004. ISBN 978-1-58113-838-2. doi: 10.1145/1015330.1015345. 16

J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015. 13

F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *arXiv:1511.07122 [cs]*, Nov. 2015. 56

M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, June 2010. doi: 10.1109/CVPR.2010.5539957. 13

Y. Zhou, J. Gao, and K. E. Barner. Locality preserving KSVD for nonlinear manifold learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3372–3376, May 2013. doi: 10.1109/ICASSP.2013.6638283. 16

Appendices

Appendix A

Exact Architecture of the DGHMM

Fig. A.1 shows the exact architecture used in training mode of the DGHMM. As explained in Chapter 3, the encoder has three strided convolutional layers[1] (F=4, S=2, and P=1 in Section 2.3.2) that downsample the motion clip in the temporal domain. The result is then fed to the final layer in the encoder, consisting of two separate dense layers[2] that remove the temporal information and provide two 100 dimensional vectors as the mean and the variance of the latent-code. After computing the final value of Z by the reparametrization trick, Eq. (2.16), the value is then forwarded to the decoder that first projects Z into a higher dimensional space that is reshaped to make the base of the later transposed convolutional layers[3] that upsample the temporal dimensions to the original clip length (240 frames in 60 frames-per-second). Finally, a wide convolutional layer (F=25, S=1, and P=12) reduces the final features count to that of the original clip. The exact objective and the training procedure is explained in Chapter 3

The operation of the model in generative mode is different from the training mode. As seen in Fig. A.2 the encoder part of the model is removed and replaced by samples from a multidimensional Normal distribution. As shown in Chapter 4, the output is joint trajectories of an articulated human skeleton.

We chose the 8 layer network for three reasons: first, with the recent advances in parallel computations on Graphics Processing Units (GPUs) and also in training deep neural networks, e.g. Glorot and Bengio [2010]; He et al. [2015a]; Kingma and Ba [2014], these models are accessible and continue to supersede any other shallow neural models with exceeding performance gain [He et al., 2015b; Telgarsky, 2016]. Additionally, in practice, we observed that deeper architectures performed better in the generative task. On the other hand, the deepest model that we could still fit into our GPU memory was 8 layers deep, therefore we kept with this design.

[1] Section 2.3.1
[2] Eq. (2.2)
[3] Section 2.3.2

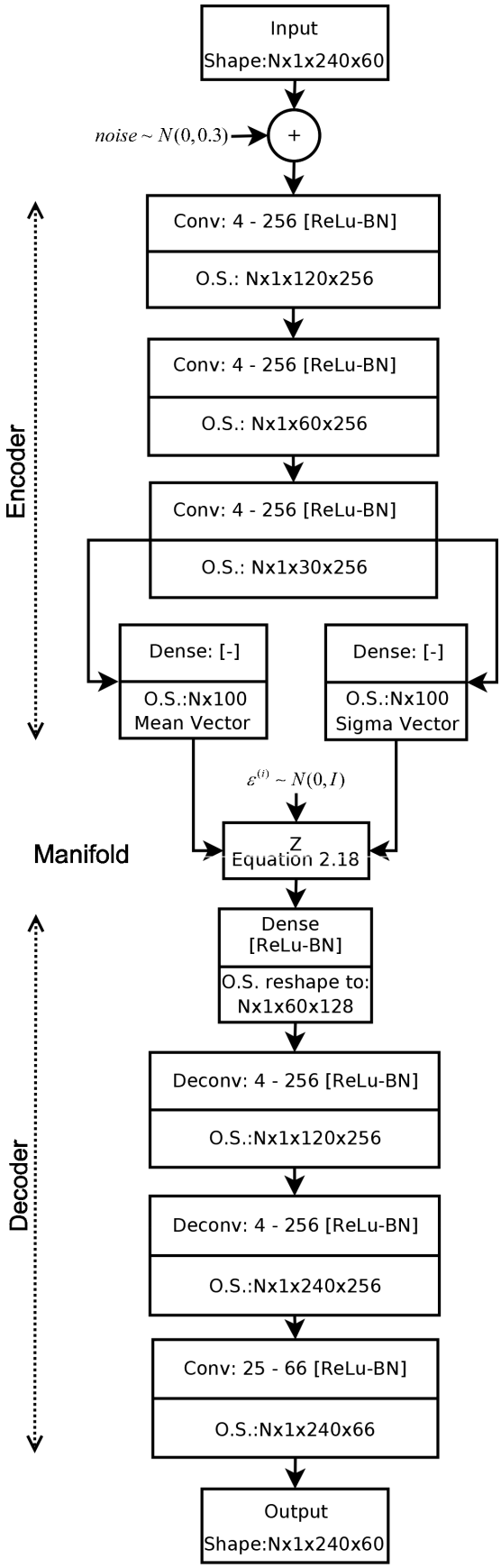


Figure A.1: DGHMM in training mode. The shape of the tensor in each stage of the network is represented by 4 numbers $N \times H \times W \times C$, where N is the batch size, H is equal to 1 since the convolution will be performed only along the time axis, W is the temporal width, and C is the number of channels or features of that tensor. Abbreviations: O.S.: output shape; Conv: convolutional layer; Deconv: deconvolutional layer; BN: batch normalization.

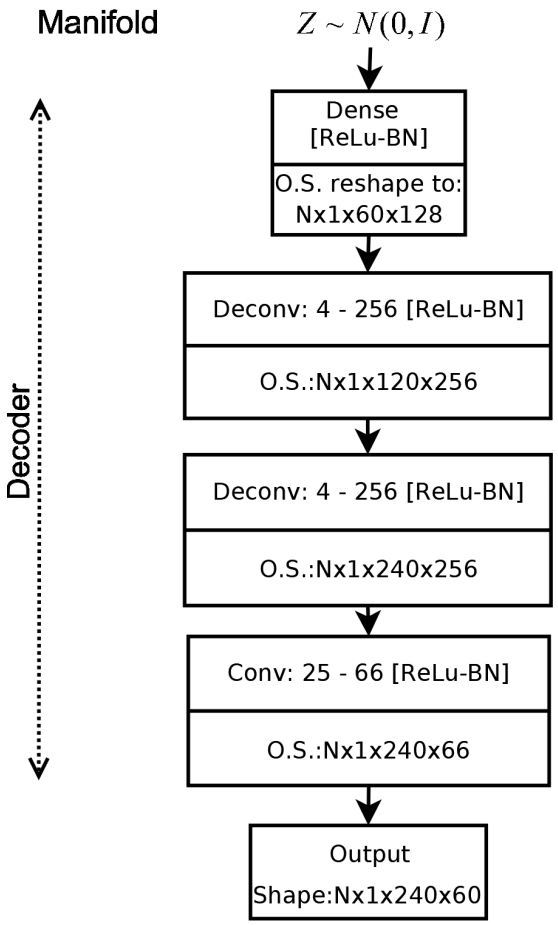


Figure A.2: DGHMM in generative mode. In this mode, the encoder part is replaced by samples from a 100 dimensional Normal distribution. Abbreviations: O.S.: output shape; Conv: convolutional layer; Deconv: deconvolutional layer; BN: batch normalization.

Appendix B

Model Hyperparameters

In this part, we mention the points regarding our choice of the model hyperparameters for the DGHMM. For choosing a set of model hyperparameters, we did around-the-clock experiments, mostly in vain, however essential to determine a *working* selection. Our attempt was severely bounded by the equipment at our disposal[1] and the limited time of the thesis. Therefore, at this stage of the research we cannot make further claims regarding the performance of the currently chosen hyperparameter set, other than it works; nevertheless, here we share some insights that we gained during the experiments.
The influential hyperparameters include kernel width in different convolutional layers, depth of the network, type of the input noise, and its amount. Contrary to Holden et al. [2015], we don't use wide kernels throughout our network, rather narrow ones with width four[2]. Wide kernels provide means for including temporally global information and produce smoother joint trajectories, yet they introduce more parameters per layer for the same architecture, which increases the chance of overfitting; additionally, the receptive field size for the final layers of a deep architecture is already very wide with respect to the input. Moreover, a smarter way exists to grow the receptive field size without using wide kernels, e.g. dilated convolutions [Yu and Koltun, 2015]. Further discussion in this direction would be out of the scope of the present research, nevertheless, we note that having wide kernels is essential for the work of the generative model, since it provides wide range temporal information so that the whole clip won't be meaningless. The existing dense layers in the architecture of the DGHMM, already provide means for wide range connections, still, we tend to completely remove them in future, in favor of a fully convolutional architecture that would be able to generate motion clips with arbitrary length.
On the depth of the network, contrary to the shallow network of Holden et al. [2015, 2016], our model has 8 layers that are considered deep in the context of DNNs[3]. We

[1] A single Nvidia 980Ti GPU proved not to be enough for validating different sets of hyperparameters while training a DNN.

[2] With the exception of the final layer that has a wide kernel with the width of 25.

[3] The classic Alexnet [Krizhevsky et al., 2012] that achieved state-of-the-art performance in 2012 ImageNet Classification task is also 8 layers deep.

found out that in the case of generating task guided motion, Section 5.1, the manifold of the deeper architecture performed worst than the manifold of a shallow network. That might be the reason behind single layer manifold of Holden et al. [2016]. On the other hand, we observed that a single layer manifold, on top of a deep regressor, overfitted the task-specific data while being trained end-to-end. Additionally, the KL loss of the training objective, Eq. (2.15), didn't converge for the shallow networks, which avoided the manifold to be a generative one. In the end, we chose the deep version of the network for the generative model, hence the title of this thesis.

Regarding the type and amount of the input noise, following Vincent et al. [2010], we used a DAE as our base model and extended it with the latent-code regularization of a VAE. In practice, we observed no significant difference between the manifold of a DGHMM with denoising criterion and without it, yet we chose to include it since the literature considers it as a useful regularization method that also avoids learning identity function by the Autoencoder (AE) network, Section 2.4.1. In contrast to Holden et al. [2015, 2016], we use a Gaussian noise as the corruption procedure rather the binomial noise. Gaussian noise is preferred for continuous signals as pointed out by Vincent et al. [2010].

In conclusion, further systematic hyperparameter search can potentially suggest more options to improve the quality of the samples of the generative model, but a qualitative criterion to measure this quality is still a vital missing component[4].

[4]Refer to Section 5.1 for further discussion on this issue.