Statistical Models for Human Motion Synthesis

Qi Wang

CENTRALE MARSEILLE

École Doctorale : Mathématiques et Informatique de Marseille (ED184)

Laboratoire d'Informatique et Systèmes (UMR 7020)

THÈSE DE DOCTORAT

pour obtenir le grade de

DOCTEUR de l'ÉCOLE CENTRALE de MARSEILLE

Discipline : Informatique

Statistical Models for Human Motion Synthesis

par

WANG Qi

Directeur de thèse : ARTIÈRES Thierry

Soutenue le 09 Juillet 2018

devant le jury composé de :

PAQUET Thierry	Professeur, LITIS, Université de Rouen	Rapporteur
BEVILACQUA Frédéric	Directeur de Recherche de IRCAM, Sorbonne Université	Rapporteur
PÉLACHAUD Catherine	Directrice de Recherche,CNRS,UMPC Sorbonne Université	Examinateur
DENOYER Ludovic	Professeur, LIP6, UMPC Sorbonne Université	Examinateur
FAVRE Benoit	Maître de conférence, LIS,Aix-Marseille Université	Examinateur
BOURDIN Christophe	Professeur, ISM, Aix-Marseille Université	Examinateur
ARTIÈRES Thierry	Professeur, LIS, Ecole Centrale de Marseille	Directeur de thése

Abstract

This thesis focuses on the synthesis of motion capture data with statistical models. Motion synthesis is a task of interest for important application fields such as entertainment, human-computer interaction, robotics, etc. It may be used to drive a virtual character that can be involved in the applications of the virtual reality, animation films or computer games. This thesis focuses on the use of statistical models for motion synthesis with a strong focus on neural networks. From the machine learning point of view designing synthesis models consists in learning generative models. Our starting point lies in two main problems one encounters when dealing with motion capture data synthesis, ensuring realism of postures and motion, and handling the large variability in the synythesized motion. The variability in the data comes first from core individual features, we do not all move the same way but accordingly to our personality, our gender, age, and morphology etc. Moreover there are other short term factors of variation like our emotion, the fact that we are interacting with somebody else, that we are tired etc.

Data driven models have been studied for generating human motion for many years. Models are learned from labelled datasets where motion capture data are recorded while actors are performed various activities like walking, dancing, running, etc. Traditional statistical models such as Hidden Markov Models, Gaussian Processes have been investigated for motion synthesis, demonstrating strengths but also weaknesses. Our work focuses in this line of research and concerns the design of generative models for sequences able to take into account some contextual information, which will represent the factors of variation.

A first part of the thesis present preliminary works that we realized by extending previous approaches relying on Hidden Markov Models and Gaussian Processes to tackle the two main problems related to realism and variability. We first describe an attempt to extend contextual Hidden Markov Models for handling variability in the data by conditioning the parameters of the models to an additional contextual information such as the emotion which which a motion was performed. We then propose a variant of a traditional method

for performing a specific motion synthesis task called Inverse Kinematics, where we exploit Gaussian Processes to enforce realism of each of the postures of a generated motion. These preliminary results show some potential of statistical models for designing human motion synthesis systems. Yet none of these technologies offers the flexibility brought by neural networks and the recent deep learning revolution.

The second part of the thesis describes the works we realized with neural networks and deep architectures. It builds on recurrent neural networks for dealing with sequences and on adversarial learning which was introduced very recently in the deep learning community for designing accurate generative models for complex data. We propose a simple system as a basis synthesis architecture, which combines adversarial learning with sequence autoencoders, and that allows randomly generating realistic motion capture sequences. Starting from this architecture we design few conditional neural models that allow to design synthesis systems that one can control up to some extent by either providing a high level information that the generated sequence should match (e.g. the emotion) or by providing a sequence in the style of which a sequence should be generated.

Keywords: Motion Synthesis; Neural Networks; Adversarial Learning; Generative Models; Motion Capture.

Résumé

Cette thèse port sur la synthèse de séquences de motion capture avec des modèles statistiques. La synthèse ce type de séquences est une tâche pertinente pour des domaines d'application divers tels que le divertissement, l'interaction homme-machine, la robotique, etc. Du point de vue de l'apprentissage machine, la conception de modèles de synthèse consiste à apprendre des modèles génératifs, ici pour des données séquentielles. Notre point de départ réside dans deux problèmes principaux rencontrés lors de la synthèse de données de motion capture, assurer le réalisme des positions et des mouvements, et la gestion de la grande variabilité dans ces données. La variabilité vient d'abord des caractéristiques individuelles, nous ne bougeons pas tous de la même manière mais d'une façon qui dépend de notre personnalité, de notre sexe, de notre âge de notre morphologie, et de facteurs de variation plus court terme tels que notre état émotionnel, que nous soyons fatigués, etc.

Divers modèles statistiques ont été étudiés pour la synthèse de ce type de données. Les modèles sont appris sur des jeux de données étiquetés où les séquences de motion capture sont réalisées par des acteurs exécutant diverses activités comme marcher, danser, courir, etc. Notre travail porte sur ce domaine de recherche et concerne en particulier la conception de modèles générateurs de séquences capables de prendre en compte certaines informations contextuelles pour permettre une génération contrôlée de séquences, et capables de générer des séquences réalistes avec une certaine diversité.

Une première partie présente des travaux préliminaires que nous avons réalisés en étendant des approches de l'état de l'art basées sur des modèles de Markov cachés et des processus gaussiens pour aborder les deux problèmes principaux liés au réalisme et à la variabilité. Nous décrivons d'abord une variante de modèles de Markov cachés contextuels pour gérer la variabilité dans les données en conditionnant les paramètres des modèles à une information contextuelle supplémentaire telle que l'émotion avec laquelle un mouvement a été effectué. Nous proposons ensuite une variante d'une méthode de l'état de l'art utilisée pour réaliser une tâche de synthèse de mouvement spécifique ap-

pelée Inverse Kinematics, où nous exploitons les processus gaussiens pour encourager le réalisme de chacune des postures d'un mouvement généré. Nos résultats montrent un certain potentiel de ces modèles statistiques pour la conception de systèmes de synthèse de mouvement humain. Pourtant, aucune de ces technologies n'offre la flexibilité apportée par les réseaux de neurones et la récente révolution de l'apprentissage profond et de l'apprentissage Adversarial que nous abordons dans la deuxième partie.

La deuxième partie de la thèse décrit les travaux que nous avons réalisés avec des réseaux de neurones et des architectures profondes. Nos travaux s'appuient sur la capacité des réseaux neuronaux récurrents à traiter des séquences complexes et sur l'apprentissage Adversarial qui a été introduit très récemment dans la communauté du Deep Learning pour la conception de modèles génératifs performants pour des données complexes, notamment images. Nous proposons une première architecture simple qui combine l'apprentissage Adversarial et des autoencodeurs de séquences, qui permet de mettre au point des systèmes performants de génération aléatoire de séquences réalistes de motion capture. A partir de cette architecture de base, nous proposons plusieurs variantes d'architectures neuronales conditionnelles qui permettent de concevoir des systèmes de synthèse que l'on peut contrôler dans une certaine mesure en fournissant une information de haut niveau à laquelle la séquence générée doit correspondre, par exemple l'émotion avec laquelle une activité est réalisée. Pour terminer nous décrivons une dernière variante qui permet de réaliser de l'édition de séquences de motion capture, où le système construit permet de générer une séquence dans le style d'une autre séquence, réelle.

Mots Clés: Synthèse de mouvement; Réseaux de Neurones; Apprentissage Contradictoire; Modèles Génératifs; Capture de mouvement.

Acknowledgement

Firstly, I would like to express my sincere appreciation to my supervisor Thierry Artières. He has been supporting me on my research and encouraging me to overcome new difficulties and challenges in my research. He taught me how to conduct scientific research and used his profound knowledge and experience to help me become a qualified researcher. It would have been impossible for me to complete my research works and finish this thesis smoothly without his generous assistance. This experience of collaborating with him will keep benefiting me in my future research career.

Thank Catherine Pelachaud who has a lot of discussion with me about my PhD project. She is a very kind professor and she is always supportive throughout my PhD studying.

Also I thank my friend and collaborators Yu Ding, Jing Huang for their cooperation in our collaborative works. It is an honor to be friend and work with them.

I appreciate to be friend with some excellent people and I would like to thank them for their encouragement and company. They are Hao Dong, Yulong Gong, Yao Lu, Huihuai Qiu, Changshun Wu, Rui Liu, Jian YANG, Ms Qi WANG, Ziyu GUO etc. I also would like to express thanks to all my lovely colleagues in QARMA team. They are very kind and lovely and I am very happy to have them during my PhD studying.

I also would like to thank my best friend Haizi Yu for his help in my research. We had many fruitful discussions which were full with joys and laughter and helped me a lot for my research. He is always optimistic and open-minded which impressed me very much and I am happy to be friend with him.

Finally from my heart, I would like to express my gratitude and love to my family, my dear father, mother, my girlfriend, my two sisters who always supported me whenever and wherever. Especially, I want to express my thanks to my girlfriend and also my best friend Wei HE for her company. Her company makes my life colorful and happy.

Table of Contents

Abstract i

Résumé iii

Acknowledgement v

1 Introduction 1

I Background 7

2 Statistical Models for Sequences 11

2.1 Hidden Markov Models 11

2.1.1 The Three Problems 13

2.1.2 Optimize HMM 14

2.2 Gaussian Process Regression Models 16

2.2.1 Gaussian Process 16

2.2.2 Kernel Functions 16

2.2.3 Gaussian Process Regression 17

2.3 Artificial Neural Networks 21

2.3.1 Fully Connected Neural Networks 21

2.3.2 Recurrent Neural Networks 23

2.3.3 Sequence to Sequence Models 24

2.4 Adversarial Learning 25

2.4.1 Generative Adversarial Networks 26

2.4.2 Adeversarial autoencoders 28

2.5 Conclusion 28

3 Motion Capture 31

3.1 Motion Capture Data 31

3.1.1 Motion Capture . . . 31
3.1.2 Datasets . . . 34
3.2 Motion synthesis tasks and related works . . . 34
3.2.1 Motion synthesis and forecasting . . . 35
3.2.2 Inverse Kinematics . . . 39
3.2.3 Dealing with Styles . . . 40
3.3 Conclusion . . . 40

II Preliminary studies with Hidden Markov Models and Gaussian Processes **41**

4 Contextual HMMs for zero shot learning **45**
4.1 Introduction . . . 46
4.2 Sharing parameters for activity classification with CHMMs . . . 47
4.2.1 Contextual Hidden Markov Models . . . 47
4.2.2 Using one-hot encoding of discrete contextual variables . . . 50
4.3 Zero shot learning via distributed context representation learning . . . 51
4.3.1 Joint Learning of CHMMs parameters and of context representation 52
4.3.2 Derivation of reestimation formulae for θs . . . 53
4.4 Experimental Results . . . 54
4.4.1 DataSet . . . 54
4.4.2 Experimental Setting . . . 55
4.4.3 Activity Classification Results . . . 56
4.4.4 Learning Curve and Convergence of θ . . . 57
4.4.5 Emotion Classification . . . 58
4.5 Conclusion . . . 59

5 Inverse Kinematics using Gaussian Process **61**
5.1 Introduction . . . 62
5.2 Background of Jacobian Method . . . 62
5.3 Inverse Kinematics using Gaussian Process . . . 64
5.3.1 Offline Preprocessing . . . 65
5.3.2 Predicting Gaussian Parameters using Gaussian Processes . . . 66
5.3.3 Online Synthesis . . . 68
5.4 Experiments . . . 69
5.5 Conclusion . . . 70

III Motion synthesis with Neural Networks and Adversarial Learning **75**

6 Generative model **81**
6.1 Introduction . . . 81
6.2 AutoEncoder for Sequences . . . 82
6.3 Adversarial Autoencoder for Sequences . . . 83
6.4 Motion synthesis with a ASAE . . . 85
6.5 Implementation and variants . . . 86
6.6 Conclusion . . . 87

7 Conditional Models **89**
7.1 Conditional synthesis models . . . 89
7.1.1 Conditional RNNs . . . 90
7.1.2 Conditional Adversarial Sequence Auto-Encoders (CASAE) . . . 91
7.1.3 Conditional synthesis from style-free encodings of motion sequences: Adversarial Style Free Sequence Auto-Encoders (ASFSAE) . . . 93
7.2 Motion edition through disentangling factors of variation: Disentangling Adversarial Sequence Auto-Encoder (DASAE) . . . 94
7.3 Conclusion . . . 96

8 Experiments **99**
8.1 Experimental setting . . . 99
8.1.1 Dataset . . . 99
8.1.2 Baselines . . . 100
8.1.3 Implementation details . . . 100
8.2 Objective evaluation . . . 100
8.2.1 Likelihood estimation . . . 100
8.2.2 Diversity and completeness . . . 101
8.2.3 Pose Forecasting . . . 102
8.2.4 Style classification on generated sequences. . . . 104
8.3 Qualitative Evaluation . . . 104
8.3.1 Unconditional models . . . 104
8.3.2 Conditional models. . . . 106
8.4 Latent representation space . . . 107
8.5 Conclusion . . . 108

9 Conclusion **117**

Bibliography 127

Chapter 1

Introduction

This thesis focuses on the synthesis of motion capture data with statistical models. Motion synthesis is a task of interest for important application fields such as entertainment, human-computer interaction, robotics, etc. For instance, motion synthesis may be used to drive a virtual character that can be involved in the applications of the virtual reality, animation films or computer games [45, 54, 60]. Actually motion synthesis covers various settings as we will discuss in this chapter. In particular the synthesis may be performed so that the generated motion matches some predefined constraints. These constraints may be very precise, one may want a motion that match given positions of few specific parts of the body (e.g. hands, feet) at some time stamps. Or the constraints may be softer, such as the motion should represent a specific motion activity (walking, dancing) or it should be the motion of a young people, or of a sad people etc. We specifically work on a complete description of the body as captured by motion capture devices where a human motion is represented by a sequence of postures, each one being represented as a real-valued vector of the angles of the body joints. More difficult settings deal with incomplete data at training time, such as videos, but are out of the scope of this thesis.

Designing automatic tools for generating high precision and controllable motion is a hard task and remains a challenge. Data driven models have been studied for generating human motion for many years. These models are learned from labeled datasets where motion capture data are recorded while actors are performed various activities like walking, dancing, running, etc. Some traditional statistical models such as Hidden Markov Models (HMMs), Gaussian Processes (GPs) have been investigated for motion synthesis [12, 54, 39, 27]. HMM based models yielded interesting results but seem hard to be used for synthesizing sophisticated motions. Gaussian Process based models are not easily scalable to large datasets, which limits their generation ability. Recently, the

revival of neural networks and in particular of recurrent neural networks raised again the interest of researchers for these models to design motion synthesis systems [46, 45, 44].

This thesis focuses on the use of statistical models for motion synthesis with a strong focus on neural networks. Our starting point lies in one particular problem that arises when dealing with motion capture data, their variability. This variability in the data comes first from the inter-individual variability. We do not all move the same way. Consider that we may recognize a friend from behind just looking at the way he walks. It is part of our identity. By the way, this yields a necessity of very accurate motions to be well recognized and accepted by human people. A second related factor of variability lies in our physiology, young people don't move the same way as older ones, fat ones don't move as slim ones etc. Besides such individual factors of variations there are other short term factors like our emotion, our gender, the fact that we are interacting with somebody else, that we are tired etc.

If one wanted to deal with such a number of combinations of these factors of variations, either individual and short term, one should need to collect a huge dataset covering as much combinations as possible and design either one model for each combination or one smart model able to cope with all these combinations. Moreover in practice, either some combinations will be very rare in the training set or some combinations will not occur at all in the training set which raises another problem close to a one shot learning problem.

How to deal with motion data variability is then a key issue for designing motion synthesis systems able to generate realistic motions in a controllable fashion. Few works have been proposed in that direction [92, 48, 94, 39]. Our work follows this line of research and concerns the design of generative models for sequences able to take into account some contextual information, which will represent the factors of variation we just discussed.

Quite naturally we started from the results obtained in previous works that were done in the team on this topic. The first systems I investigated during my thesis were based on Hidden Markov Moels (HMMs). In particular these works relied on parametric HMMs [88], that were extended in [69, 30, 29] to design HMMs whose parameters are conditioned on external factors and that were used for several conditional motion synthesis tasks such as speech conditioned motion synthesis [30]. Our preliminary studies to go further with these models confirmed the potential of these models for some motion capture tasks but at the same time it showed that designing more elaborate models with this technology could become a hard task.

I moved then to explore the use of Gaussian Processes while focusing on a different

motion synthesis task known as Inverse Kinematics (IK). IK is a traditional problem in animation field. It aims at finding a posture or sequence of postures which satisfies some constraints on the position or the sequence of positions for some joints. Yet the sequence of postures obtained using traditional methods is not always realistic and I explored the use of Gaussian Processes to enforce such a realism.

In the last two years I focused my work on artificial neural networks. Actually the impressive revival of neural networks, the deep learning phenomena, came with a revolution for sequence data as well with, for the first time, recurrent neural networks (RNNs) that actually worked well, in particular thanks to the use of specific cells like LSTM units [43] in particular. Even more importantly for my present work the revolution arose in 2014 with the invention of Adveraial Learning and the proposal of Generative adversarial nets (GANs) [36]. Adversarial Learning is a strategy for learning generative models. Many variety of GANs have been proposed since and successfully applied, in particular to images generation [18, 71, 57, 70]. However, most of these models are designed for images and few of them are for sequence data such as motion capture data. The combination of practical RNNs and of Adversarial Learning opened the way to the works I realized in the last two years of my work.

The thesis is organized in three parts. The first part covers background about statistical models for sequences and motion synthesis. It includes two chapters.

Chapter2 presents the statistical models that we will rely on for modeling human motions. We briefly introduce Hidden Markov Models and we present Gaussian Process and Gaussian Process Regression. Then we introduce neural networks including fully connected neural networks and recurrent neural networks. At last, we detail the recent advances in generative models with adversarial learning. Along the chapter we discuss in particular of Sequence to Sequence models and of adversarial autoencoders, two ideas that we will use to design our own models.

Chapter 3 introduces the backgrounds about motion synthesis. We first present what are motion capture data and how these are represented, and present two famous motion capture datasets, the CMU dataset and the Emilya Dataset. Finally we provide a brief review of state of the art in motion synthesis.

The second part reports our preliminary works on using Hidden Markov Models (HMMs) and Gaussian Processes (GPs) for dealing with few aspects of motion data, handling variability with HMMs and generating realistic postures with GPs. This part includes

two chapters.

Chapter 4 investigates the exploitation of contextual HMMs [69, 30] for dealing with motion variability. We first introduce Contextual HMMs and detail how one can use such models for learning from limited datasets by sharing parameters. Then we propose a learning strategy that jointly learns the CHMM parameters and the latent representation of the contextual information (discrete variables in our case) that allow performing some kind of zero shot learning, i.e. generalizing to situations that have not been seen during training. This work has been performed on the task of activity classification, not on a synthesis task, since it was a first and easier step to explore the potential of the method for motion synthesis.

Chapter5 presents our work on inverse kinematics with Gaussian Processes. We first summarize related work about inverse kinematics and illustrate the traditional solution for inverse kinematics, the Jacobian method. Then we detail our method based on Gaussian processes that we used to design a variant of the Jacobian method. It consists in defining a new objective function in the Jacobian method that enforces the method to iteratively generate realistic postures.

The third part of the thesis is dedicated to the use of neural networks. It includes three chapters.

Chapter6 first presents the framework we explored in this third part to build few motion synthesis systems. It is based on the combination of Sequence Autoencoders and Adversarial Autoencoder. We introduce the models and variants and discuss of their relevance with respect to state of the art.

Chapter 7 details few models that we proposed for dealing with the variability of the data in order to build systems synthesizing motion sequences that are both realistic and controllable in some way. First we propose two conditional models which are extensions of the generative model presented in the previous chapter. Then we introduce a strategy for learning editing models that allow to modify a given sequence to change part of its style, i.e some information related to the factors of variation.

Chapter 8 reports experimental results gained with all neural models and compare our results with those of baselines in the field. We provide both objective measures that

compare the models with respect to well defined statistics and predictive tasks. Then we provide few illustrations of generated sequences, more results, including videos of motion sequences, may be found at https://bit.ly/2IanlOX.

The final and ninth chapter summarizes our work and concludes the thesis.

Part I

Background

Introduction

The two following chapters describe background that is useful for reading the remaining of the thesis. Chapter 2 recalls basics about Statistical models for sequence data, namely Hidden Markov Models, Gaussian Processes, and Recurrent Neural Networks. Chapter 3 introduces the motion capture synthesis field, describes the usual tasks, the nature of the data and the datasets we used in our work, and finally provides a short review of previous works on motion synthesis. This state of the art section will be completed in the next chapters for more focused topics.

Chapter 2

Statistical Models for Sequences

This chapter provides a brief review of the main statistical models which are the basis of the methods proposed in this thesis.

We first present Hidden Markov Models which have been the reference model for dealing with sequences (e.g. for designing speech recognition applications) for decades. As such they have been much used in the animation field to deal with motion capture data. We then briefly review Gaussian Processes that are powerful statistical models that have been used for motion capture data as well. Next we introduce neural networks and insist on specific models that we will build on and in particular, that are Recurrent Neural Networks and Sequence to Sequence models. Finally we will introduce the adversarial learning strategy that was invented a few years ago and that has been widely used for designing models able to synthesize accurate and realistic data, for images in particular.

2.1 Hidden Markov Models

Hidden Markov Models (HMMs) are one of the most famous statistical models for dealing with sequences in pattern recognition and machine learning applications. A HMM is a sequence model, or sequence classifier, whose job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels. HMMs are probabilistic sequence model that have been used for developing the first strong models for speech recognition, handwriting recognition, and many natural language processing tasks.

HMMs are particular state space models where the space is discrete. A HMM is the aggregation of a Markov chain (a probabilistic automata over a finite set of states) and

of probability density functions attached to the states. It implements a probability distribution over time-series.

We now introduce the usual notations for HMMs. First an HMM is composed of a Markov chain. We note:

- M is the number of states in the model
- $\mathcal{S} = \{s_1, s_2, ..., s_M\}$ is the set of states of the model
- $\pi = \{\pi_i\}$, is the initial state distribution, the probability of starting in state s_i is: $\pi_i = P(s_i)$.
- $A = \{a_{i,j}\}$ is the transition probability matrix, where $a_{i,j} = P(s_j|s_i)$ gives the transitions probability from state i to state j. Of course $\sum_j a_{i,j} = 1$

Next there are probability density functions attached to states. We note:

- $B = \{b_j(x)\} = p(x|s_j)$ denotes the observation probability density function (pdf) in state s_j. Typically when x is continuous, it is modeled as a Gaussian mixture. In the simpler case where it is a Gaussian distribution with mean μ_j and covariance Σ_j, we get: $b_j(x) = \mathcal{N}(x; \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_j|}} e^{-(x-\mu_j)^\top \Sigma^{-1} (x-\mu_j)/2}$.

Usually one notes $\phi = \Big\{\pi, A, B\Big\}$ the set of of all the parameters of a HMM.

Finally we use the following notations for the data the HMM operates on.

- T is the length of an observation sequence
- $\mathbf{x} = \{x_1, x_2, ..., x_T\}$ is an observation sequence of length T

HMMs are named *Hidden* Markov models since the state is usually hidden so that the information one gets on a sequence, either in training or in testing stage, is partial. One usually introduce the following notations:

- $\mathbf{q} = \{q_1, q_2, ..., q_T\}$ denotes the state sequence

The sequence of states in a HMM is assumed to follow an order one Markov property, meaning that the state at time t depends on the state at time $t-1$ only. In other words, state at time t is a conditionally independent to states before $t-1$, given state at time t. Then the transition probability is actually $a_{i,j} = P(q_t = s_j|q_{t-1} = s_i)$ and a initial state probability stands for $\pi_i = P(q_1 = s_i)$.

According to the assumption of a HMM, an observation sequence $\mathbf{x} = (x_1, x_2, ..., x_T)$ is generated as follows:

1. Draw an initial state $q_1 = s_i$ according to the initial state distribution π
2. Set t = 1;
3. Sample x_t from the pdf in current state q_t, $b_{q_t}(x)$;
4. Draw the new state according to the transition probability distribution in the current state, q_t, $a = \{a_{q_t,j}\}_{j=1,\cdots,M}$;
5. Set $t = t + 1$; if $t < T$ go back to step 3; otherwise stop the procedure.

Figure 2.1 illustrates the process. Each state generates an observation and thus the generated sequence is refer to as **Observation Sequence**. The randomly generated state series is referred to as **State Sequence**.

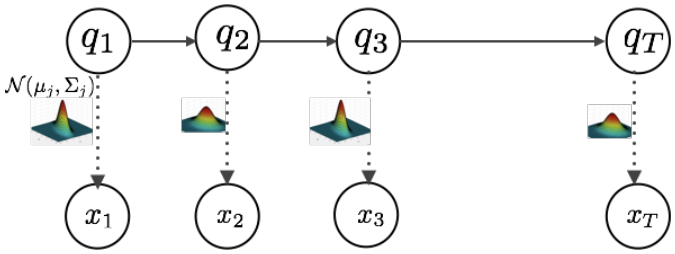


Figure 2.1: Illustration of generation process by a HMM. Each state q_t is responsible for emitting an observation x_t while the model switches from state to state according to a transition probability distribution.

2.1.1 The Three Problems

There are three key problems of interest if one wants to use HMMs [68]:

- **Problem 1**—Given an observation sequence $\mathbf{x} = \{x_1, x_2, ..., x_T\}$ and the model $\phi = \{\pi, A, B\}$, how to compute $p(\mathbf{x}|\phi)$, the likelihood of the observation sequence?
- **Problem 2**—Given an observation sequence $\mathbf{x} = \{x_1, x_2, ..., x_T\}$, how to determine the most likely state sequence $\mathbf{q} = \{q_1, q_2, ..., q_T\}$?
- **Problem 3**— Given a training set of N sequences $\left\{\mathbf{x}^1, ..., \mathbf{x}^N\right\}$ how to find the model's parameters $\phi = \{\pi, A, B\}$ which maximize the likelihood of the training data $p\left(\left\{\mathbf{x}^1, ..., \mathbf{x}^N\right\}|\phi\right)$?

The last problem is the training problem and is the most important. Besides the first and second problem also get involved in the process of solving the problem 3. So we recap now how to optimize a HMM by solving problem 3.

2.1.2 Optimize HMM

Given a training set of N i.i.d. observation sequences $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^N\}$, the HMMs are usually set through Maximum Likelihood (Eq 2.1).

$$\phi^* = \arg\max_{\phi} \log p(\mathbf{x}^1, ..., \mathbf{x}^N; \phi) = \sum_{k=1}^{N} \log p(\mathbf{x}^k; \phi) \tag{2.1}$$

There is no known way to solve the maximum likelihood model analytically. Instead Expectation Maximization (EM) algorithm [68] can be used for solving this problem and iteratively optimise the HMM's parameters.

The EM algorithms consists in iterating two steps:

- The E-Step builds a lower bound of the likelihood of the training set.
- The M-Step updates the model parameters by optimizing the lower bound of the likelihood.

The 'E' and 'M' steps are performed iteratively and after several loops the likelihood probability converges to a maximum. Every iteration new values of the parameters are computed. The reestimation formulae involve the following intermediate variables:

$$\gamma_t^k(j) = p(q_t = s_j | \mathbf{x}^k; \phi) \tag{2.2}$$

$$\xi_t^k(i,j) = p(q_t = s_i, q_{t+1} = s_j | \mathbf{x}^k; \phi) \tag{2.3}$$

where k indexes the number of the training sequence, and where $\gamma_t^k(j)$ denotes the probability of being in state j at time t given the observation sequence $\mathbf{x}^k$, and $\xi_t^k(i,j)$ denotes the probability of transiting from state i at time t to state j at time $t+1$, given the observation sequence $\mathbf{x}^k$.

These quantities may be computed using the following variables which may efficiently be computed via dynamic programming routines (known as forward and backward algorithms):

$$\alpha_t^k(j) = P(x_1, x_2, ...x_t, q_t = s_j; \phi) \tag{2.4}$$

$$\beta_t^k(j) = P(x_{t+1}, x_{t+2}, ...x_T | q_t = s_j; \phi) \tag{2.5}$$

$\alpha_t^k(j)$ is the joint probability of the partial observation sequence (until t) and state $q_t = s_j$ at time t, given the model ϕ. $\alpha_t^k(j)$ is called forward variable and can be obtained by an algorithm called forward procedure [9, 68]. $\beta_t^k(j)$ denotes the probability of the partial sequence (from $t+1$ to T), given the state $q_t = s_i$ and the model ϕ. It is referred to as backward variable and can be computed by backward procedure [9, 68].

Going back to γ and ξ variables, these may be computed efficiently as well since they may be expressed with the above quantities:

$$\gamma_t^k(j) = \frac{\alpha_t^k(j)\beta_t^k(j)}{p(\mathbf{x}^k;\phi)} = \frac{\alpha_t^k(j)\beta_t^k(j)}{\sum_{j=1}^{M} \alpha_t^k(j)\beta_t^k(j)} \tag{2.6}$$

and

$$\xi_t^k(i,j) = \frac{\alpha_t^k(i)a_{i,j}b_j(x_{t+1})\beta_t^k(j)}{p(\mathbf{x}^k;\phi)} = \frac{\alpha_t^k(j)a_{i,j}b_j(x_{t+1})\beta_t^k(j)}{\sum_{j=1}^{M} \alpha_t^k(j)\beta_t^k(j)} \tag{2.7}$$

In the M-step, one uses an auxiliary function (Eq 2.8) derived from the likelihood probability to update the model parameters.

$$\begin{aligned} Q(\phi'|\phi) &= \mathbb{E}_{\mathbf{q}|\mathbf{x},\phi}[\log P(\mathbf{x},\mathbf{q};\phi')] \\ &= \sum_{\mathbf{q}} P(\mathbf{q}|\mathbf{x};\phi)\log P(\mathbf{x},\mathbf{q};\phi') \end{aligned} \tag{2.8}$$

where ϕ' denotes the estimated model parameters, and ϕ is the current model parameters. The estimates of model parameters ϕ' can be obtained by maximising the auxiliary function.

This leads to rather intuitive reestimation formulae such as:

$$\pi(s_i) = \frac{1}{K}\sum_k \gamma_1^k(i) \tag{2.9}$$

$$\mu_(s_i) = \frac{\sum_{k,t} \gamma_t^k(i)x_t^k}{\sum_{k,t} \gamma_t^k(i)} \tag{2.10}$$

2.2 Gaussian Process Regression Models

2.2.1 Gaussian Process

A Gaussian process defines a distribution over functions, $p(f)$, where f is a function mapping some input space $\mathcal{X}$ to $\mathcal{Y}$. If $f(\cdot)$ is a function sampled from a gaussian process, we can denote it as follows:

$$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot)) \tag{2.11}$$

Definition $p(f)$ is a gaussian process if for any finite subcollection $\{x^1, ...x^n\} \subset \mathcal{X}$, the distribution $p(f(x^1), ..., f(x^n))$ has a multivariate gaussian distribution as Eq 2.12.

$$\begin{bmatrix} f(x^1) \\ \vdots \\ f(x^n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x^1) \\ \vdots \\ m(x^n) \end{bmatrix}, \begin{bmatrix} k(x^1,x^1) & \dots & k(x^1,x^m) \\ \vdots & \ddots & \vdots \\ k(x^m,x^1) & \dots & k(x^m,x^m) \end{bmatrix} \right) \tag{2.12}$$

where the covariance matrix is denoted by $\mathbf{K}$.

2.2.2 Kernel Functions

In Eq 2.12, the function $m(\cdot)$ can be any real-valued function. In order to make the resulting matrix K positive semidefinite for any set of points $x^1, ..., x^m \in \mathcal{X}$, the function $k(\cdot,\cdot)$ must be a valid kernel [31].

Given the mean function and a kernel function, one is able to sample functions from the gaussian process defined by the mean and kernel function. Different kernel functions yield different GPs and result in different sampled funtions as shown in Fig 2.2. Next we introduce four common kernel functions and their properties.

a) Radial Basis Function Kernel (RBF)

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\ell^2}) \tag{2.13}$$

RBF kernel is also called Gaussian kernel. It is the most common kernel in GPs. It has only hyperparmeters: output variance σ and lengthscale ℓ. σ determines the average distance of a function away from its mean. ℓ determines the length of the fluctuation in a function.

b) Rational Quadratic Kernel (RQK)

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2(1 + \frac{||\mathbf{x} - \mathbf{x}'||^2}{2\alpha\ell^2})^{-\alpha} \tag{2.14}$$

RQK is similar to adding together many RBF kernels with different lengthscales. A GP with RQK results in functions which vary smoothly across many lengthscales (see Fig 2.2b).

c) Exponentially Sine Squared Kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{2\sin^2(\pi/p * ||\mathbf{x} - \mathbf{x}'||^2)}{\ell^2}) \tag{2.15}$$

This is a periodic kernel and it can be used for modelling periodic functions (see Fig 2.2c). The hyperparameters p controls the period of functions in a GP. ℓ determins the lengthscale in the same way as RBF kernel.

d) Linear Kernel

$$k(\mathbf{x}, \mathbf{x}') = \sigma + \mathbf{x}^\top \mathbf{x}_j \tag{2.16}$$

A GP with a linear kernel yields linear functions (see Fig 2.2d). The linear kernel is dependent on the absolute location of two input points instead of their relative distance as in previous three mentioned kernels.

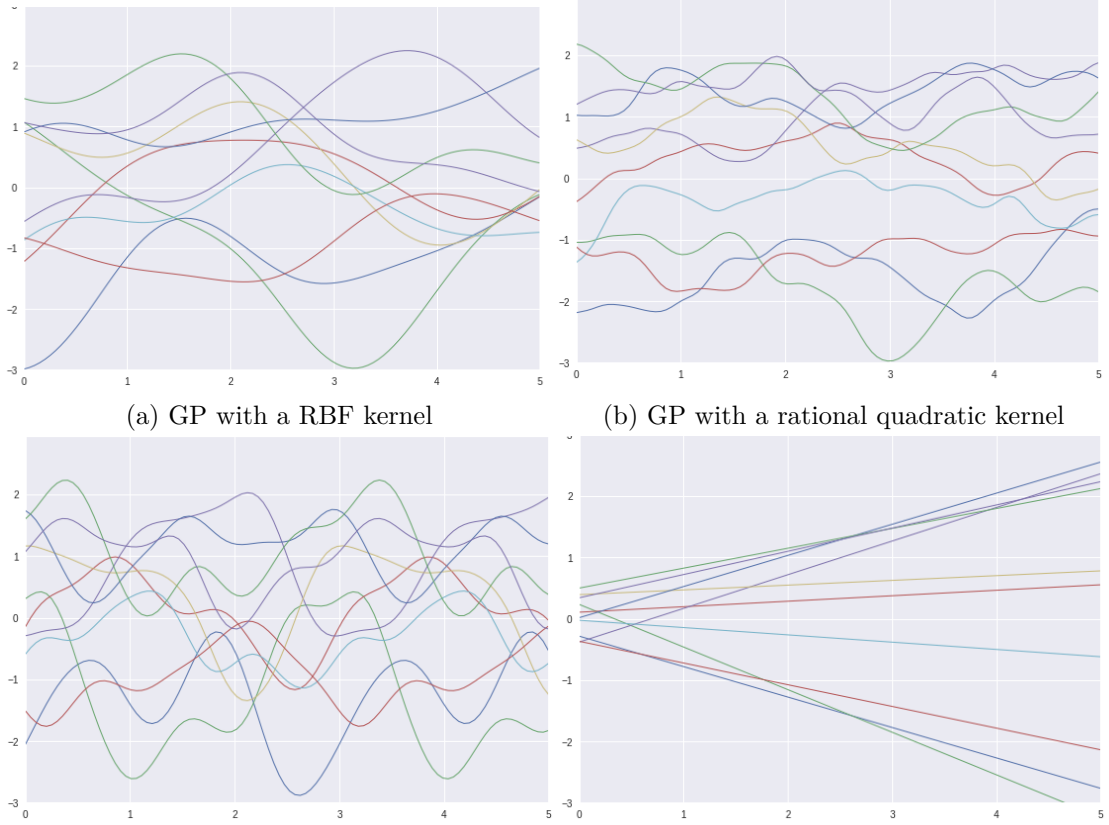
2.2.3 Gaussian Process Regression

Suppose that we have a training set $\{\mathbf{x}^n, y^n\}_{n=1}^N$, with $\mathbf{x}^n \in \mathcal{X}, y^n \in \mathcal{Y}$. We wish to learn the function $f(\cdot) \in \mathcal{F} : \mathcal{X} \times \mathcal{Y}$ and make prediction of $\mathbf{y}^*$ given a new $\mathbf{x}^*$. Unlike the parametric models, we do not directly formalise the function $f(\cdot)$. We assume the prior distribution over the functions $f(\cdot)$ is a zero-mean Gaussian process prior (Eq 2.17).

$$f(\cdot) \sim \mathcal{GP}(0, k(\cdot, \cdot)) \tag{2.17}$$

where $k(\cdot, \cdot)$ is a kernel function. Then the joint distribution over the function values $\mathbf{f} = \{f^n\}_{n=1}^N$, gives the corresponding inputs $\mathbf{X} = \{\mathbf{x}^n\}_{n=1}^N$ is a Gaussian distribution with zero mean and covariance $\mathbf{K}$ as follows:

$$p(\mathbf{f}) = p(f^1, f^2, ..., f^N) = \mathcal{N}(\mathbf{0}, \mathbf{K}) \tag{2.18}$$



(a) GP with a RBF kernel

(b) GP with a rational quadratic kernel

(c) GP with a ExpSineSquare kernel

(d) GP with a linear kernel

Figure 2.2: Functions sampled from four GP with different kernel functions.

where the function value $f^n = f(x^n)$, and covariance matrix $\mathbf{K}$ has elements:

$$\mathbf{K}(m, n) = k(\mathbf{x}^m, \mathbf{x}^n) \tag{2.19}$$

The Gaussian process regression model can be defined as follows:

$$y^n = f^n + \epsilon^n, n = 1, \ldots, N \tag{2.20}$$

where the ϵ measures the uncertainty of the f_n. We assumes it has a gaussian distribution:

$$\epsilon^n \sim p(\epsilon) = \mathcal{N}(0, \sigma^2) \tag{2.21}$$

The conditional probability distribution of y^n given f^n is as follows:

$$p(y^n|f^n) = \mathcal{N}(f^n, \sigma^2) \tag{2.22}$$

Because the noise ϵ is independent for each data example, the joint distribution of $\mathbf{y} = \{y^1, ..., y^N\}^\top$ conditioned on $\mathbf{f}$ has an isotropic gaussian as follows:

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I}) \tag{2.23}$$

From Eq 2.18 and Eq 2.23, we can get the marginal distribution over $\mathbf{y}$ given $\mathbf{X}$ as follows

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) &= \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})d\mathbf{f} \\ &= \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma^2\mathbf{I}) \end{aligned} \tag{2.24}$$

We choose a widely used RBF kernel (see Eq 2.13 to compute the covariance matrix, then the marginal likelihood becomes a function of the hyperparameters θ of the kernel function.

$$p(\mathbf{y}|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{0}, \mathbf{K}_\theta + \sigma^2\mathbf{I}) \tag{2.25}$$

Its logorithm likelihood is :

$$\ln p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2}\det(\mathbf{K}_\theta + \sigma^2\mathbf{I}) - \frac{1}{2}\mathbf{y}^\top(\mathbf{K}_\theta + \sigma^2\mathbf{I})^{-1}\mathbf{y} - \frac{N}{2}\ln(2\pi) \tag{2.26}$$

By maximizing the loglikelihood, one can optimize the parameters θ and σ. Alternatively, in order to make the model immune to overfitting, one can use use Bayesian method to maximising the posterior distribution over the parameters θ.

Making Predictions Given a new input vector $\mathbf{x}^*$, we want to use the above gaussian regression model to predict the target output y^*. From Eq 2.24, we can obtain the joint distribution over $y^*, \mathbf{y}|x^*, \mathbf{X}$ as follows:

$$p(y^*, \mathbf{y}|x^*, \mathbf{X}) = \mathcal{N}(\mathbf{0}, \mathbf{K}_{N+1} + \sigma^2\mathbf{I}) \tag{2.27}$$

where K_{N+1} is a $N+1 \times N+1$ matrix defined as follows

$$\mathbf{K}_{N+1} = \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^\top & k(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix} \tag{2.28}$$

where $\mathbf{k} = (k(\mathbf{x}^1, \mathbf{x}^*), k(\mathbf{x}^2, \mathbf{x}^*), ..., k(\mathbf{x}^N, \mathbf{x}^*))^\top$.

We can evaluate the predictive distribution $p(y^*|\mathbf{x}^*, \mathbf{X})$.

$$p(y^*|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(\mu_*(\mathbf{x}^*), \sigma_*^2(\mathbf{x}^*)) \tag{2.29}$$

where $\mu_*(\mathbf{x}^*)$ and $\sigma_*^2(\mathbf{x}^*)$ are defined as follows:

$$\begin{aligned} \mu_*(\mathbf{x}^*) &= \mathbf{k}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} \\ \sigma_*^2(\mathbf{x}^*) &= k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k} + \sigma^2 \end{aligned} \tag{2.30}$$

By sampling from 2.29, we can predict the value of y^*. The Fig 2.3 shows the sampled functions from a gaussian process regression model fitted on 10 datapoints (red points in the figure).

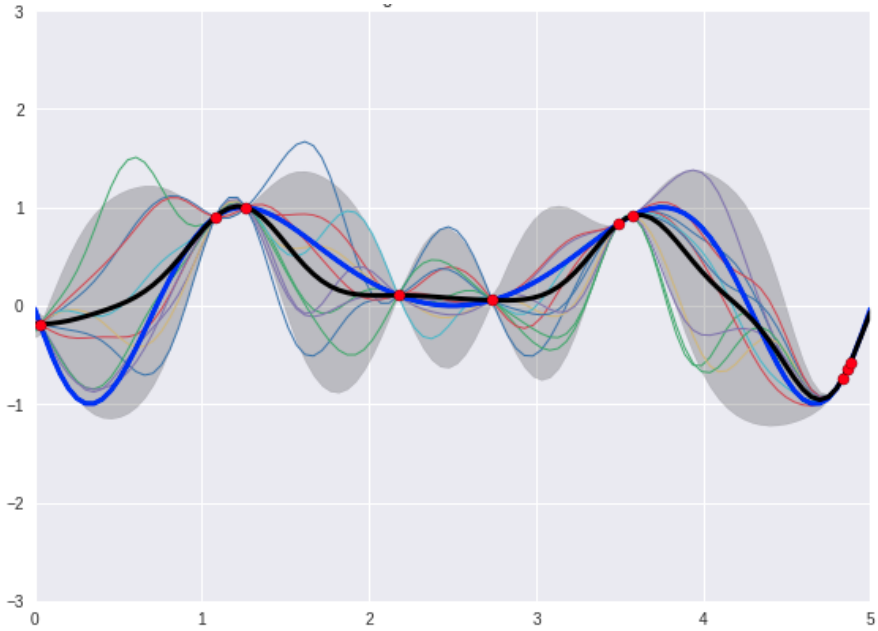


Figure 2.3: Illustration of gaussian process regression. The blue curve is the true function from which the red data points are sampled. The red points are the training examples for fitting the guassian process regression model with a RBF kernel. The black curve is the mean function $\mu(\mathbf{x}^*)$ in Eq 2.30, and the other curves are functions sampled from the gaussian process. The shaded region corresponds to +/- two standard deviations.

2.3 Artificial Neural Networks

Arificial Neural Networks (ANNs) have become a very popular framework in machine learning. ANNs are inspired from the biological neural networks that exist in the human brain [1]. An ANN is composed of nodes called neurons that are linked with weighted connections. In modern ANNs (such as Multi-Layer Perceptrons) the neurons are organised into several layers. These layers are stacked hierarchically to form a multi-layered neural network. The connections between artificial neurons can transmit signal from one to another, and the neuron that receives the signal can process it and then outputs a value for next connected neuron.

Compared with other machine learning models, the main advantage of ANNs is a large modeling capacity and a high flexibility in designing architectures. Using the backpropagation learning algorithm, it can learn relevant representation of data in order to solve the problem of interest such as classification or regression. ANNs have been successfully applied in almost all the tasks in machine learning from image classification, speech recognition to image generation, speech synthesis. In this section we first introduce fully connected neural networks which are the most basic neural network. Then we present recurrent neural networks (RNNs) which have wildly been applied to sequential data, such as speech and video. Finally, we introduce Long Short-Term Memory units (LSTM) which are now very popular to build recurrent models and that allow solving the gradient vanishing problem when dealing with long sequences.

2.3.1 Fully Connected Neural Networks

Fully Connected Neural Networks (FCNNs) are the simplest version of neural networks. In a FCNN, each neuron in one layer is connected to all the neurons in the next layer as shown in Figure 2.4. A FCNN consists of an input layer, one or multiple hidden layers and an output layer. We take a one-layer FCNN (see Fig 2.4) as an example to introduce the feed forward process.

Consider an input vector $\boldsymbol{x} = (x_1, x_2, x_3)^\top \in \mathbb{R}^3$ for which we want to compute the output predicted by the model, $\boldsymbol{y} = (y_1, y_2)^\top \in \mathbb{R}^2$. We feed $\boldsymbol{x}$ into the FCNN. The outputs of the hidden layer are denoted by $\boldsymbol{h} = (h_1, h_2, h_3, h_4)^\top$. Each element in $\boldsymbol{h}$ is the output of one hidden neuron and can be obtained as follows:

$$h_i = \phi(\sum_{j=1}^{3} w_{ij} x_j + b_i) \tag{2.31}$$

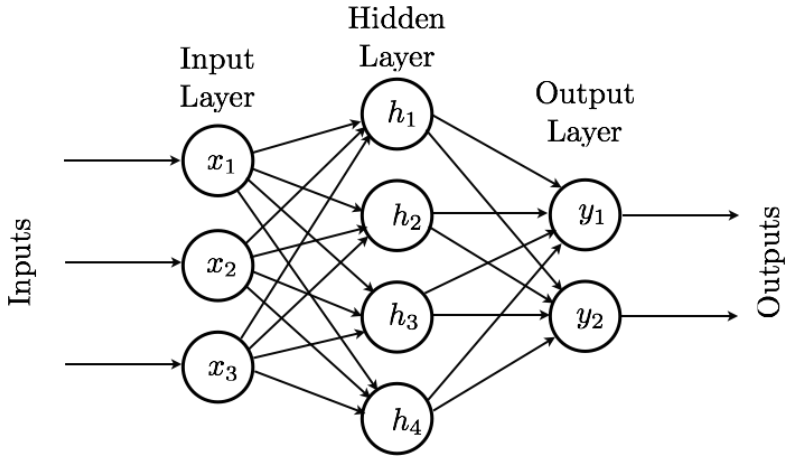


Figure 2.4: Illustration of the structure of fully connected neural networks

where w_{ij} is the weight of the connection from input neuron x_j to the hidden neuron h_i and b_i is the bias of hidden neuron h_i. $\phi(\cdot)$ is an activation function which is usually a non-linear function.

Then it can be written as a matrix form as follows:

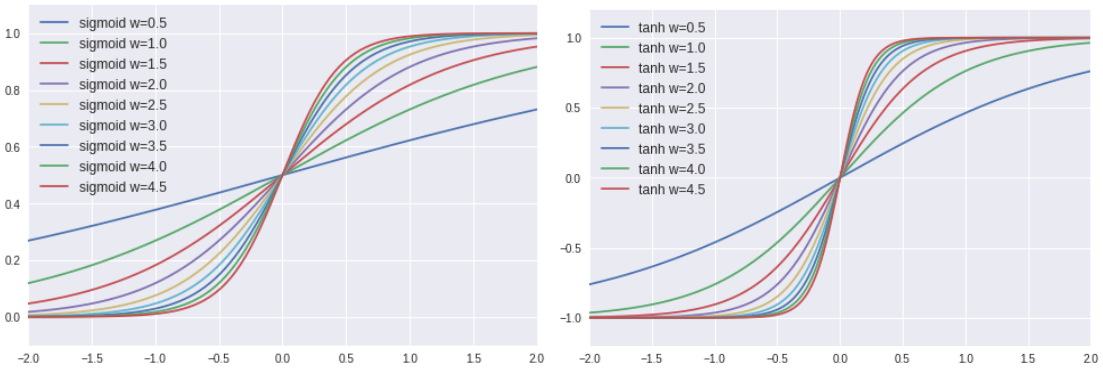
$$\boldsymbol{h} = \phi(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}) \tag{2.32}$$

where $\boldsymbol{W} \in \mathbb{R}^{4\times 3}$ is the weight matrix and $\boldsymbol{b} \in \mathbb{R}^{4\times 1}$ is the bias matrix. Note that the activation function $\phi(\cdot)$ is elementwise. There are different activation functions in practice, and here we show some of them in Figure 2.5. Similarly the outputs $\boldsymbol{y}$ can be computed as follows:

$$\begin{aligned} \boldsymbol{y} &= \psi(\boldsymbol{U} \cdot \boldsymbol{h} + \boldsymbol{c}) \\ &= \psi(\boldsymbol{U} \cdot (\phi(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})) + \boldsymbol{c}) \end{aligned} \tag{2.33}$$

where $\boldsymbol{y}$ denotes the output vector of the output layer, $\boldsymbol{U}$ and $\boldsymbol{c}$ are respectively the weight matrix and bias vector for the output layer

Fig 2.5 shows how the values of the weights w determine the curvature of the activation function. In the learning process, the learning algorithm (usually gradient based learning methods) can adjust the values of parameters $\boldsymbol{w}$ and $\boldsymbol{b}$ and makes each neuron have a different activation function with different curvature and bias. These non-linear activation functions endow the neural network the ability of learning rich features. Because a neural network implements a very non-convex function with respect to its parameters, there is not any analytical solution for optimizing the neural network and gradient based methods such as SGD are used.



(a) Sigmoid Function with respect to different value of w

(b) Tanh Function with respect to different value of w

Figure 2.5: Different activation functions:Sigmoid Function, Tanh Function

2.3.2 Recurrent Neural Networks

Fully connected neural networks are suitable for processing fixed-size data. However, for time series or sequential data, one needs models which are able to learn and to take into account dependencies between successive observations in the sequences. Recurrent neural networks (RNNs) [59] are designed for such data. RNNs have been successfully applied in speech recognition [38], handwriting synthesis [37] and video analysis [64]. Unlike in FCNNs, RNNs include cycles in the connections. Including such recurrent connection allows gathering information as observations in a sequence are processed. The basic structure of an RNN is shown in Fig 2.6.

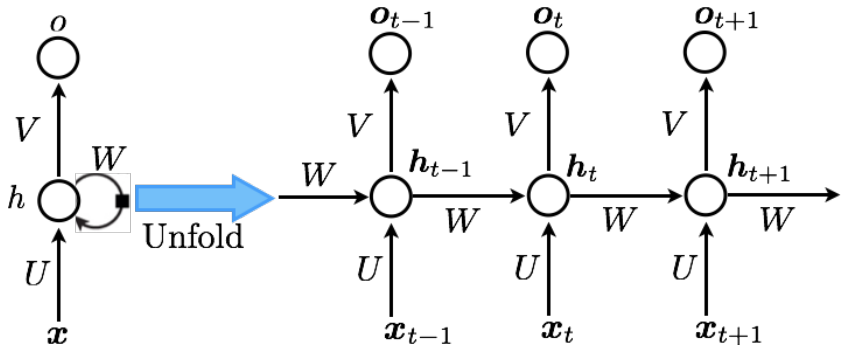


Figure 2.6: Illustration of recurrent connection of RNNs.

Consider a time series denoted by $\mathbf{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)^\top$ with length T. The sequence $\mathbf{x}$ is fed into the RNN frame by frame. At each time step, one neuron will output one hidden state, and after the whole sequence is fed into the RNN, we will obtain a sequence of hidden states denoted by $\mathbf{h} = (\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_T)^\top$. Similarly, we will get a sequence of outputs denoted by $\mathbf{o} = (\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_T)^\top$. Note that in the Figure 2.6,

each node represents an entire vector. Differing from FCNNs, besides the feed-forward connections, each hidden neuron has a recurrent connection from previous time step to current time step. The state $\boldsymbol{h}_t$ can be computed as folows:

$$\begin{aligned} \boldsymbol{h}_t &= \phi(\boldsymbol{U} \cdot \boldsymbol{x}_t + \boldsymbol{W} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}), \text{ if } 1 < t \leq T \\ \boldsymbol{h}_1 &= \phi(\boldsymbol{U} \cdot \boldsymbol{x}_1 + \boldsymbol{b}), \text{ if } t = 1 \end{aligned} \tag{2.34}$$

where $\phi(\cdot)$ is the activation function of hidden layer. The values output neurons are computed as follows:

$$\boldsymbol{o}_t = \psi(\boldsymbol{V} \cdot \boldsymbol{h}_t + \boldsymbol{c}) \tag{2.35}$$

where ψ is the activation function of the output neurons.

RNNs are learned with a variant of back-propagation which consist in unfolding the network a number of times equal to the length of the input sequence, then to perform forward and backward propagation in this unfolded network, while paying attention that all the network copies share the same parameters (hence the gradient is cumulated over all copies).

Long-term Dependencies and LSTM units One of the most challenging problem in RNNs is the difficulty of learning long-term dependencies. Because of the recurrent connections, the gradient of the recurrent connection tend to explode or vanish with long-term interaction, which makes learning long-term dependencies impossible.

For solving the long-dependencies problem, Gated RNNs which include long short-term memory (LSTM) [73] and gated recurrent unit (GRU) [23] were proposed. Both of these two units exhibit a particular architecture that enables skipping the cells (i.e. authorizing the cell to copy previous state) which has appeared to be a key issue to ease gradient flow in deep architectures such as Residual Networks for instance [42].

2.3.3 Sequence to Sequence Models

Sequence to Sequence (S2S) models are a particular architecture that we will intensively use in the design of our systems in the last part of this thesis. We present them in more details now. S2S have been proposed by Ilya Sutskever et al. [74] to build machine translation model. Sequence to Sequence models as end-to-ends model have been widely used in processing sequential data like language, speech. The structure of a S2S model 2.7 is similar to a standard autoencoder where both the encoder and decoder are RNNs.

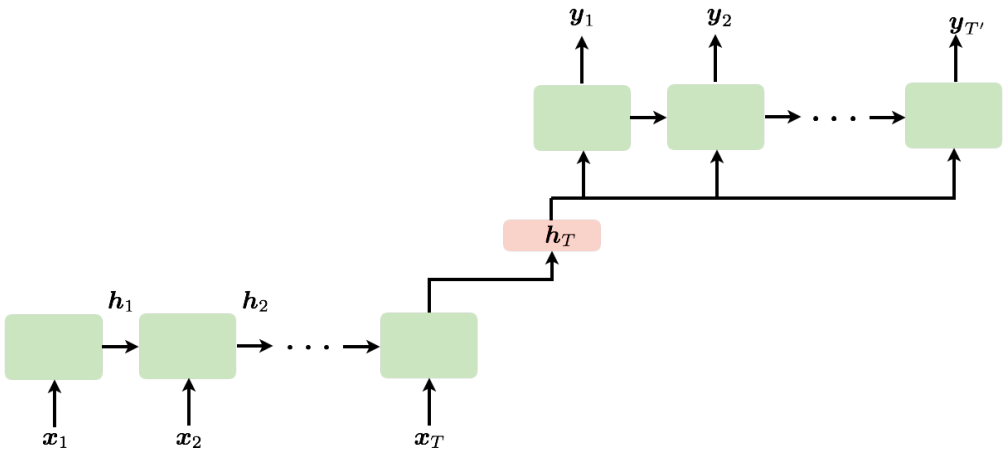


Figure 2.7: Architecture of a sequence to sequence model

Given an input sequence $\mathbf{x} = (x_1, x_2, x_T)^\top$, the encoder of the S2S model maps the whole sequence to a fixed-length latent vector $\boldsymbol{h}_T$. This vector may be seen as a new representation of the full input sequence, we will often call it a latent representation of the input sequence. This latent vector is then expected to contain all the necessary information about the input sequence $\mathbf{x}$ so that the decoder may build the output sequence from it. There are usually two ways to connect the latent vector to the decoder network. One way is to add $\boldsymbol{h}_T$ as input to each time step of the unfolded RNN-based decoder. The other way is to provide $\boldsymbol{h}_T$ as initial state of the RNN decoder. In addition predictions of the output observation at time t, y_t, may be taken as input to the RNN decoder at time $t+1$ to predict y_{t+1}.

2.4 Adversarial Learning

Generative models aim to learn the data distribution $p(x)$ from a training dataset. Once learned they may be used to generate new data by sampling from the data distribution $p(x)$. Yet learning the underlying probability density function of complex and real data such as natural images has remained too difficult for usual methods. The seminal work on Generative Adversarial Networks (GANs) [36], and the parallel work on Variational AutoEncoders (VAEs) [50], have been a significant breakthrough in the field. It has led to many variants and studies and allowed researchers to build more and more accurate generative models for complex data with a main focus on natural images.

2.4.1 Generative Adversarial Networks

Unlike many previous models that rely on learning a generative model through likelihood maximization which may be intractable most of time (e.g. for example, VAEs use a variational lower bound to approximate the true likelihood), Generative Adversarial Networks (GANs) are not learned to directly estimate the likelihood of a training dataset. Instead GANs are learned so that after learning the model may be used to sample data in such a way that the distribution with which it samples data is close (optimally equal to) the underlying distribution of the data in the training set. Hence one may get a machine that can sample data according to the targeted distribution but one cannot actually access to this distribution.

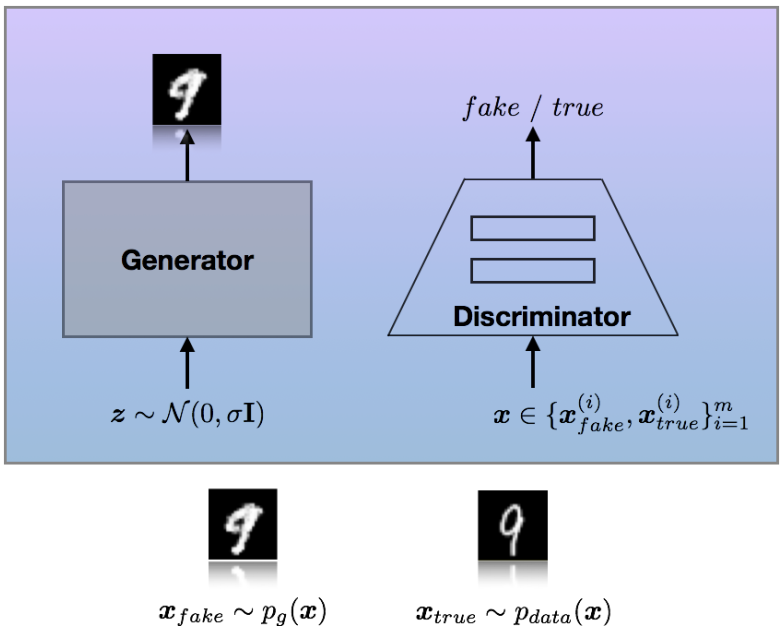


Figure 2.8: structure of Generative Adverarial Nets

Min-Max Game A GAN consist of two models as shown in Fig 2.8. The first model is called *Generator* and denoted by G. It takes as input a latent vector $\boldsymbol{z}$ sampled from a prior distribution $p(\boldsymbol{z}) = \mathcal{N}(0, \boldsymbol{\sigma}\mathbf{I})$ and generates an artificial data. The second model is called *Discriminator* and noted D. It aims at distinguishing if an input is an artificial data generated by the Generator from the *Generator* or a real data sample from the training set. These two models are implemented with neural networks whose parameters are learn through a min-max game with the following objective:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.36}$$

where p_{data} represents the distribution of data in the training set which we want to learn through a GAN. By maximizing the objective function, the *Discriminator* will assign a high value to a real input data and a low value to an input from the *Generator*. In the contrary the *Generator* tries to fool the *Generator* by generating better samples, i.e. ones that the *Discriminator* assign high values. This optimization is like a min-max game where the *Discriminator* and *Generator* compete with each other and improve themselves simultaneously. Hopefully after convergence samples generated by the *Generator* cannot be discriminated by the discriminator while the *Discriminator* is not able to be optimized further because generated data are actually indistinguishable for real ones. The learning algorithm of GANs is shown in Algorithm 1.

Algorithm 1 Min-Max Learning algorithm for GANs [36]

for number of training iterations **do**
  for k steps **do**
    •Sample a minibatch of m noise samples $\{\boldsymbol{z}^1, ..., \boldsymbol{z}^m\}$ from the prior distribution $p_{noise}(\boldsymbol{z})$
    •Sample a minibatch of m real data samples $\{\boldsymbol{x}^1, ..., \boldsymbol{x}^m\}$ from training set
    •Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(\boldsymbol{x}^i) + \log(1 - D(G(\boldsymbol{z}^i))) \right] \tag{2.37}$$

  end for
  •Sample minibatch of m noise samples $\{\boldsymbol{z}^1, ..., \boldsymbol{z}^m\}$ from the prior distribution
  •Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(\boldsymbol{z}^i))\right) \tag{2.38}$$

end for

The *Generator* implicitly defines a probability distribution p_g. One may show that when the min-max game reaches Nash Equilibrium, the *Discriminator* is unable to recognize the fake from real data and the *Generator* implements a distribution on the data that equals p_{data}. Unfortunately this Equilibrium is difficult to reach in practice, mainly because of the non-convex property of the objective. In practice then, the learning of GANs is unstable and many works have been proposed to improve learning, e.g. [6].

In this work we mainly used GANs rather than VAES since GANs are known to generate more realistic data than VAEs, this is the reason why we did not detail these models here.

2.4.2 Adeversarial autoencoders

A particularly interesting variant, called *adversarial autoencoders*, has been proposed in [57]. An autoencoder is a well known neural network architecture that learns to reconstruct a data while passing through an information bottleneck. A simple autoencoder is a NN consisting of first layer, the encoder Enc, that projects the input data in a lower dimension, followed by a second layer, the decoder Dec, that tries to reconstruct the input data from the state of the hidden layer. The state of the hidden layer, named $\mathbf{e}(x)$ is called the encoding of the input x. An autoencoder is learned to minimize the mean reconstruction error $E_{x\sim p_{data}}[\|x-\mathbf{d}(\mathbf{e}(x))\|^2]$ from a training set. In an adversarial autoencoder, an autoencoder is primarily learned as usual to reconstruct true data. The discriminator is learned to distinguish between noise samples z (sampled from a prior distribution p_{noise}) and encodings of true data commputed by the Encoder. Moreover an additional term in the objective requires the encoder Enc to fool the discriminator. This adversarial learning makes the encoder map the input data into a latent space with a given prior distribution. Hence after learning one may use the decoder alone as a generative model by sampling z from the prior distribution $p_{noise}(z)$ then by computing $Dec(z)$. The objective criterion becomes:

$$\begin{aligned}\min_{\mathbf{e},\mathbf{d}} \max_{D} &\mathbf{E}_{x\sim p_{data}}[\|x-\mathbf{d}(\mathbf{e}(x))\|^2] \\ &+ \mathbf{E}_{z\sim p_{noise}}[\log(1-D(z)] \\ &+ \mathbf{E}_{x\sim p_{data}}[\log D(\mathbf{e}(x))]\end{aligned} \tag{2.39}$$

2.5 Conclusion

There exist a variety of statistical models for dealing with sequence data. Hidden Markov models have long been the reference technology, e.g. with many systems built for speech recognition and natural language processing. We naturally started with the idea to use such models for the motion synthesis tasks we wanted to address in this thesis. As a side work we investigated the benefits one could get from using Gaussian Process that have already been used for sequence syntesis. Yet the growing popularity of recurrent neural networks (RNNs) made us change our plans. RNNs have become for some years now the new reference models for recognition and prediction tasks. Besides, Sequence to sequence models and their variants have demonstrated the possibility to compute low dimensional vector representation of structured data and opened the way to many models relying on this capacity. We decided at some point to focus on how to explore the

potential of these models for motion synthesis, in particular when combined adversarial learning strategy.

Chapter 3

Motion Capture

This chapter introduces necessary concepts about motion capture and motion synthesis that we will use in the manuscript. Motion Capture Data (Mocap) are wildly used today in animation, games, etc [62, 54, 32, 46]. Creating new animations usually requires capturing motion capture data, which is a time-consuming and labor-expensive process which explains why automatically learning how to synthesize motion sequences is an issue. We first introduce the motion capture data that are used in the field and the way they are represented. Then we present the various tasks that are covered by motion synthesis such as the synthesis of motion sequences, the control of motion and motion editing etc, and we briefly review statistical models that have been proposed for these tasks.

3.1 Motion Capture Data

3.1.1 Motion Capture

Motion Capture (Mocap) is the process of recording the movement of objects or people. In filmmaking or video game development, it refers to recording the actions of human actors. Then the captured Mocap dataset can be used to animate digital character models in 2D or 3D computer animation.

Euler Angles There are different ways to parameterize the orientation of a rigid body. Euler Angles [3] is one common way used in motion capture data. It comprises three degree-of-freedom rotation along three axes x, y, z (see Figure 3.1).

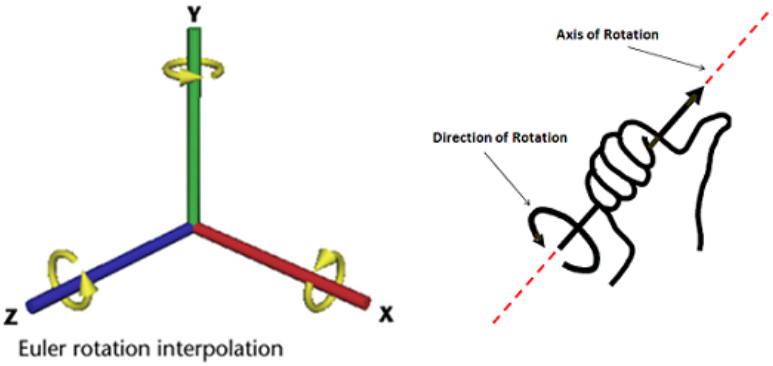


Figure 3.1: rotations along x, y, z axes

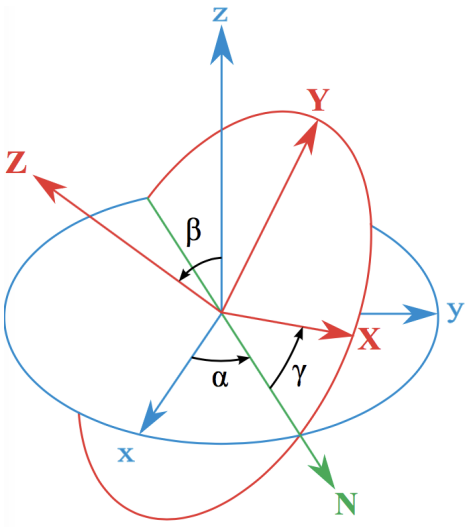


Figure 3.2: Euler Angles (α, β, γ) in order of "Z-X-Z". The blue axes are the initial frame, and the red axes represents the frame fixed in the rigid body [3].

We denote the 3 degree of freedoms (DOFs) of Euler angles as α, β, γ. According to the order of axes to rotate, there are 12 different sets of Euler angles definitions. Different authors may use different settings. Euler angles can be defined by intrinsic rotations. The rotated frame XYZ may be imagined to be initially aligned with coordinate axis xyz, before undergoing the three elemental rotations represented by Euler angles. Alternatively one may use extrinsic rotations that are elemental rotations that occur about the axes of the fixed coordinate system xyz. The XYZ system rotates, while the reference coordinate system does not.

Here we introduce one set of Euler angles in order of "Z-X-Z" (see Figure 3.2). In Figure 3.2, the blue axes x,y,z denotes the original frame and the red axes of X, Y, Z denotes the rotated frame fixed in the rigid body. The blue frame represents the original state of the rigid body, and the red frame is the final state of the rigid body.

The orientation of the rigid body from blue frame to the red frame is denoted by Euler Angles (α, β, γ) in order of "Z-X-Z". This process can be achieved as follows:

1 At the begining, the blue and red frames overlap completely.

2. Rotate the rigid body α along Z axis of the rotated frame (red in Figure 3.2)

3. Rotate the rigid body β along X axis of the rotated frame.

4. Rotate the rigid body γ along Z axis of the rotated frame.

There are different formats to represent motion capture data. BVH format and "ASF/AMC" formats are two common formats for motion capture data.

BVH file comprises two parts: a header part and a data part. The header section describes the hierarchy and initial pose of the skeleton. For example, it defines the name of each joints, the relationship of parent-child nodes and their relative position in the initial pose. In the data section, the actual motion data are contained. Each line is one frame of the motion sequence. Each frame includes the global position of the root joint which also represents the position of the body. Then it contains the global orientation of the root joint. The remaining data are the euler angles of the orientation of each joint.

ASF/AMC format uses two individual files: a ASF file and a AMC file. The ASF file defines the skeleton of the character. It is similar to the header part of BVH file. The AMC file defines the motion data the same with the data section in a BVH file. There is no essential difference between ASF/AMC format and BVH format. They both use Euler angles to represent poses. The only difference for different formats are the parsing process and display software.

In my work, I used two famous motion capture datasets: the Emilya Dataset [32] and the CMU mocap dataset [2]. The Emilya dataset was gathered by LTCI group of Telecom Paristech and is dedicated to studying emotional body expression in daily actions [32]. All motions of Emilya dataset are precisely labeled with activity and emotion by volunteers. These activity and emotion labels are very important for learning supervised models. The CMU mocap dataset includes a large number of activities captured under many different scenarios, such as locomotion , and interaction with environments etc. Because the dataset is not precisely labeled and only imprecise descriptions are provided for each action, it is more suitable for unsupervised learning. In next sections, I introduce these two motion datasets respectively.

3.1.2 Datasets

Emilya Dataset

The Emilya Dataset [32] is a motion capture dataset which is captured for studying emotional body expression in daily actions. It consists of motion capture data captured from 12 actors. Each actor is asked to perform 8 different activities under eight different emotions. The 8 activities are 'Being Seated', 'Sitting Down', 'Lift', 'Throw', 'Knocking on the Door', 'Move Books', 'Simple Walk', 'Walk with something in the Hands'. The eight Emotions are 'Anger', 'Anxiety', 'Joy', 'Neutral', 'Panic Fear', 'Pride', 'Sadness' and 'Shame'. Each of the 12 actors performed 12 times each activity-emotion combination. There are then a total of 9216 sequences in the dataset. Its framerate is 120 frames per second. Motion sequences are represented as sequences of frames (of length between $200 - 500$ depending on the activity) where each frame is a 72-dimensional vector whose components correspond to 69 joint angles of 23 joints in skeleton and 3 global translation coordinates of root joint. All motion sequences are saved in '.bvh' format. All the joint rotations are represented as Euler angles in order of Y-X-Z.

CMU Mocap Dataset

The CMU mocap dataset is collected and maintained by Graphics Laboratory of Carnegie Mellon University [2]. It contains data from a large variety of actors and activities, such as locomotion actions, sport activities, and interaction with environment etc. There are a total of 144 activities and each activities has multiple trials. A part of the data is listed in Figure 3.4. The CMU motion data are saved in 'ASF/AMC' format files where its joint orientations are represented as Euler angles in order of X-Y-Z. In my work, the CMU dataset is used for training gaussian process model (see chapter 5)

3.2 Motion synthesis tasks and related works

How to learn patterns from motion capture data and how to learn models that automatically generate realistic motion capture data has then become a very relevant research topic, which is related to learning generative models in the machine learning field. Motion synthesis actually covers a few different tasks, including pure motion synthesis, motion forecasting and inverse kinematics.

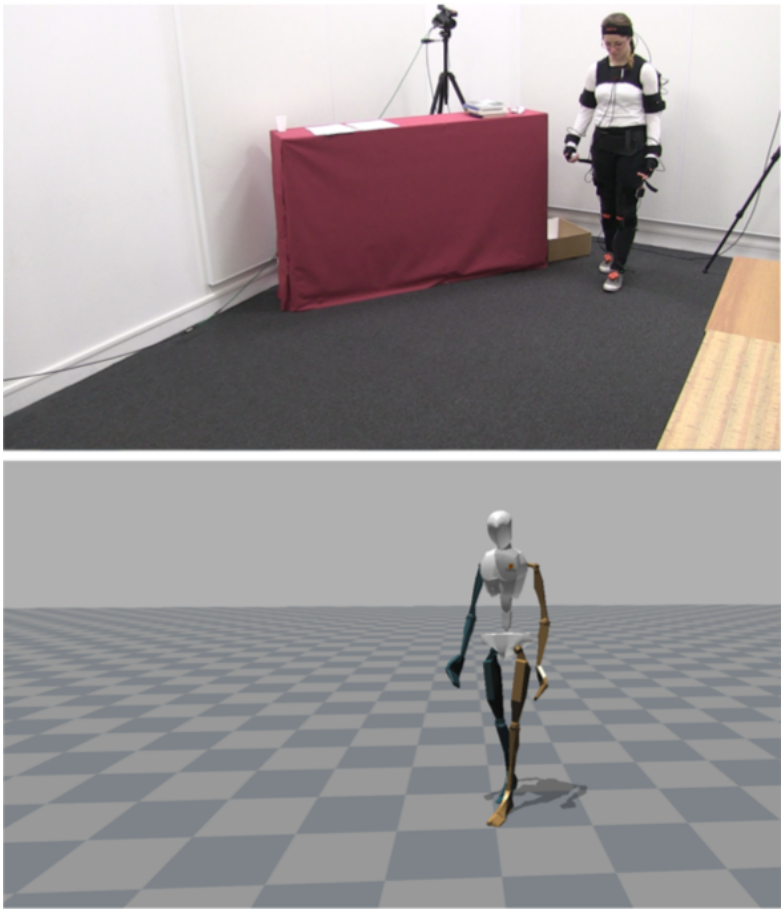


Figure 3.3: Illustration of Motion Capture[32]

3.2.1 Motion synthesis and forecasting

The most popular task is that of generating randomly motion sequences (e.g. [15]), a closely related task is motion forecasting where one wants a system to complete a motion sequence from its beginning, e.g. giving the first frames [33]. These are traditional tasks on sequence data related to the design of generative models for such data.

Hidden Markov Models. Hidden markov models (HMMs), as a reference technology for dealing with sequences, have been deeply investigated for synthesizing human motion [12, 30, 89, 28], especially with the invention of methods for generating smooth trajectories from a learned HMM [77]. Some researchers focus on synthesizing simple head motion rather than the motion of the body. Busso *et al.* proposed a HMM based model for synthesizing the head motion according to the speech features [15]. Ding *et al.* uses a fully parameterized HMM to synthesis the motions of human eyebrows [30]. The motions such as eyebrow motion, head motion etc, usually have low degree of freedoms

Subject #1 (climb, swing, hang on playground equipment) file index		
Image	Trial #	Motion Description
	1	playground - forward jumps, turn around
	2	playground - climb
	3	playground - climb, hang, swing
	4	playground - climb
	5	playground - climb, go under
	6	playground - climb, sit, dangle legs, descend
	7	playground - climb, sit, dangle legs, jump down
	8	playground - climb, sit, dangle legs, rock back, lower self to ground
	9	playground - climb, hang, hold self up with arms straight, swing, drop, sit, dangle legs, go under
	10	playground - climb, swing, lean back, drop
	11	playground - climb, hang, lean over, jump down
	12	playground - climb, pull up, dangle, sit, lower self to ground
	13	playground - climb, go under, jump down
	14	playground - climb, jump down, dangle, legs push off against

Subject #2 (various expressions and human behaviors) file index		
Image	Trial #	Motion Description
	1	walk
	2	walk
	3	run/jog
	4	jump, balance
	5	punch/strike
	6	bend over, scoop up, rise, lift arm
	7	swordplay
	8	swordplay
	9	swordplay
	10	wash self

Figure 3.4: A part of subjects of CMU dataset: Subject #1 and Subject #2

(DOFs). For the whole-body motion which has more DOFs, HMMs are not necessarily a good model because they do not encode high-order temporal dependencies easily. Hence, few works based on HMMs were done for whole-body motion synthesis [12].

For generating sequences, autoregressive recurrent neural networks have been successfully applied in tasks of handwriting synthesis, language translation etc [37, 33]. These models learn the distribution of current frame conditioned on the previous frames. However, they are not able to learn global information. Seq2Seq model [74] is a RNN-wise autoencoder which is applied in semi-supervised tasks and language translation. Although it can learn global information, it cannot generate new sequences. Sequential variational autoencoder is proposed by [11] to generate texts. It learns a probabilistic graphical model by variational inference. However, it still needs to manipulate the computation of complex probability. For example, in order to make the KL-divergence term tractable, one must choose a simple prior distribution which limits its flexibility.

Neural Networks. In the recent years, (deep) neural networks got an overwhelming success in machine learning. Neural Networks are popular for their high capacity of representation and scalability to large dataset.

Some researchers have studied using convolutional neural networks to model human motion [46, 45]. Holden et al. use convolutional neural networks (CNNs) to learn a manifold of human motion [46]. In [45], a neural network is proposed for motion controlling. Although these works achieves good results on motion synthesis, CNNs are not good choices for modelling sequential data or time series because it is not able to applied on variable-length data. In contrast, because of using recurrent connections between two successive timesteps, Recurrent Neural Networks (RNNs) are suited for dealing with variable length sequences [37, 73, 23]. Unlike in Hidden Markov Models (HMMs) where states are discrete and finite, RNNs use hidden units to represent a continuous state space with real values, hence it is endowed with an increasing modeling power. RNNs have been successfully used as sequence synthesis models for handwriting [37] and speech [80] synthesis. Few works have applied RNN to motion capture data, in particular for forecasting future motion based on an past sequence. One main work of this kind, with which we will compare our aproach to is proposed in [33], where an RNN-based network called Encoder-Recurrent-Decoder (ERD) combines an encoder decoder structure with recurrent connections on the intermediate state space. Motion synthesis mainly corresponds to generative models in machine learning field. Some breakthroughs about generative models have been made, such as Variational Autoencoder (VAEs) [50] and Generative Adversarial Nets (GANs) [36] etc. VAEs is derived from variational inference and get an evidence lower bound (ELBO) of the log likelihood of real data. By optimising the ELBO, it is theoretically able to get a good estimation of real likelihood of real data. However,the generated samples from VAEs are usually blurry due to its training objective based on pixel-wise Gaussian likelihood [55]. GANs bypasses the complicated manipulation of probability, and it leverages two models which are competing with each other during the training process to improve the quality of generated samples. This model has been demonstrated very efficient and good at generating realistic images [65, 67, 96]. We will continue discuss about adversarial learning in chapter 6.

Specific Architectures. We detail recent two specific architectures to which we will compare our neural networks based systems.

The **Encoder-Recurrent-Decoder** (ERD) was proposed in [33] for motion forecasting, i.e. frame prediction. An ERD model consists of an encoder, a RNN, and a decoder as shown in Figure 3.5. Both the encoder and the decoder are built with standard fully

connected layers. At each time step t, the raw frame x_t of the input sequence $\mathbf{x} = (x_1, ..., x_T)^\top$ passes through the encoder to get a representation of x_t. Then the output of the encoder passes through the RNN. At last, the decoder outputs the prediction of the next raw frame $\hat{x}_{t+1}$. The encoder and decoder enhance the representation ability of standard RNN and hence can achieve better performance on frame prediction than standard RNN.

The **LSTM3LR** is a standard recurrent neural networks composed of three LSTM layers. This model was used as a (strong) baseline for a frame prediction task in [33]. At each time step t, the LSTM3LR takes as input x_t and outputs the prediction of the frame at next time step $\hat{x}_{t+1}$, based on this input frame and on its previous state that summarizes the history of the input sequence (see Fig 3.6).

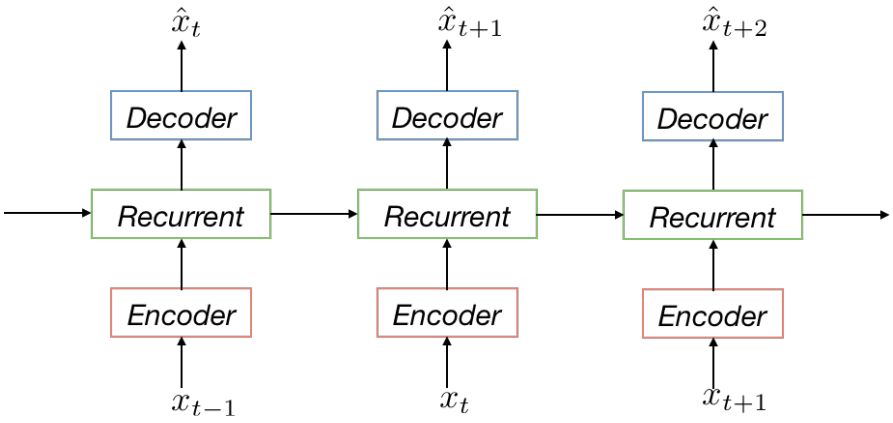


Figure 3.5: Structure of ERD

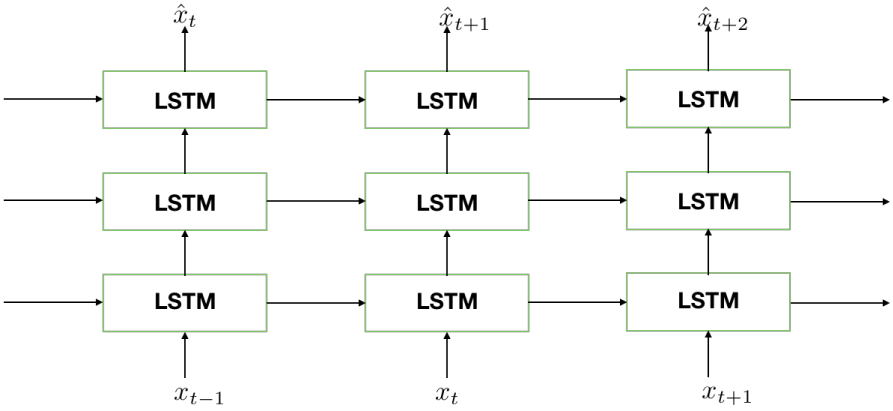


Figure 3.6: Structure of LSTM3LR

3.2.2 Inverse Kinematics

Inverse Kinematics (IK) is the problem of generating postures for human motion simulations from a set of constraints. It is widely used in robotics and computer animation. IK has been studied since 1980s [90, 81, 63, 20, 7, 95]. The original definition of IK is that for a robotic arm as shown in the left of Figure 3.7, given the the position of the end-effector (the red joint in Figure 3.7), one wants to compute the joint angles of the remaining joints. In animation field, IK can be used to synthesize a human posture from some given positional constraints on one or multiple joints as shown in the right of Figure 3.7. Furthermore, if a trajectory of the positions of one or multiple joints are given, one can synthesize a sequence of postures.

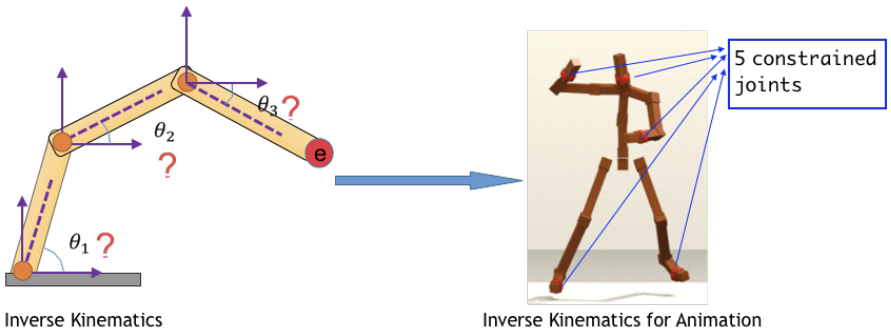


Figure 3.7: Left: Standard inverse kinematics problem. Right: Inverse kinematics for animation. Note that the red balls represents the constrained joints or end-effector.

However, given constraints, the animation solution is not unique due to the kinematic redundancy. For example, in a simple hand reaching task, an animation solution is designed for a virtual character to reach a target object. There are multiple ways for the character to reach the target object. The conventional geometric method for IK problem such as Jacobian method [13] looks for the naive solution which may satisfy all given constraints but ignore the interdependence of joints. The generated postures are often unnatural or even false in the final solution. For solving this problem, some methods have been proposed using different strategies, e.g. the body optimization by center mass position [10], dynamical system optimization [56] and the minimization of kinematic energy algorithm [47]. Besides, many different methods have been proposed to improve the IK performance for quality and speed, such as the analytic methods [87, 78, 19], the procedural numerical methods [8, 83, 14, 5], the example based approaches [93, 51, 72].

In recent years, machine learning models have been employed to simulate human motions in applications such as motion synthesis, style representation etc [79, 53, 12, 88]. All these works use statistical methods to describe the postures of human body. Compared

to the geometric methods, statistical methods usually generate more natural postures by learning the real human motion.

3.2.3 Dealing with Styles

Human motions can be altered by many factors, such as emotions, age, gender of the actor who performs the motion. These factors are expressed as some kind of style in the human motion. Dealing with these styles embedded in human motion is very challenging. Only few works have dealt with this. Wilson *et al.* proposed a parametric HMM for recognizing gestures under variation. Then [69] extended parametric HMM to contextual hidden markov model. These parametric models showed promising results for dealing with motion styles. In addition, Brand *et al.* proposed a HMM based motion synthesis model which is able to deal with styles of human motion [12]. Hsu *et al.* proposed a time warping method to align two motions and a linear model to learn the stylistic difference between two motions [48]. This model is able to transfer the style between two similar motions. However, if the context of two motions is different so that the alignment fails, this model cannot transfer style between these two motions. Xia *et al.* construct a series of local mixtures of autoregressive models to capture the complex relationships between styles of motion [92]. For each input frame, it will construct a new mixtured model using the closest examples in the data set. This method can transfer styles between two heterogeneous motions. Yumer proposed a spectral method for style transferring [94]. This method use Discrete Fourier Transform to compute the difference of two motions in the frequency domain and treat this difference as style spectral information so that one can applied this style spectral information to another motion sequence.

3.3 Conclusion

The design of accurate motion capture sequences is a difficult task that has been handled with HMMs, GPs and NNs. Many models have been proposed but still we are still missing models that may be learned on available datasets of limited size and that allow generating a variety of motion sequences that match some predefined high level feature.

Part II

Preliminary studies with Hidden Markov Models and Gaussian Processes

Introduction

The two following chapters describe preliminary works that were performed while exploring the potential of statistical models for motion synthesis tasks. We report in the next chapter a study that we realized to extend contextual Hidden Markov Models to the zero shot learning situation where some combination of contextual variables have not been seen at training stage. Then in the chapter after we describe a study in the use of Gaussian Processes for Inverse Kinematics, a specific motion synthesis task.

Chapter 4

Contextual HMMs for zero shot learning

There has been a number of works that aim at learning statistical models on corpus of motion capture data, either for activity classification or for synthesis. When working with motion capture data one key difficulty comes from the variability since the way one moves depends on many factors which are related to the individual, like her morphology, age, habits, country and also on contextual features such as emotion, role etc. We are concerned with the design of models able to handle this variability when the context of training data is known, e.g. one knows if a motion capture was performed by a man or a woman, under which emotion it was performed etc.

We consider here a classification task, activity recognition, which is simpler to evaluate than synthesis. And we focus on handling categorical contextual variables only since the case of continuous contextual variables is straightforward to handle to reach our goals. Then the training data are labeled with a class label and with labels corresponding to, eventually multiple, contextual variables.

To design models able to deal with context diversity there are usually two main possibilities. The first solution is to collect a large dataset including all the variability one wants to take into account, e.g. including samples corresponding to all the possible combination of individual and contextual features, and to exploit models with a high modeling capacity. An alternative is to design models within which the computation is factorized in some way that allows learning them from a limited training dataset where not all possible combinations of individual and contextual are observed.

The work in this chapter is a first attempt in this direction, designing models that take

into account contextual information and that are able to disentangle the influence on these factors allowing kind of zero shot learning. More specifically we investigate how to learn HMM models for activity recognition with data from the Emilya dataset when the training set does not include all possible combination of (activity, emotion) pairs.

4.1 Introduction

We are interested in designing a generic system for motion capture data (for classification and/or synthesis tasks) able to deal with data gathered in various contexts related to the gender, the age, the morphology, the emotion, the role...

Unfortunately the standard way of learning statistical models requires collecting a large corpus of data corresponding to all the cases one want to cover. This turns out to be a difficulty when one wants to learn such a generic system which should work for a large combinatorial number of different context variables, i.e. factors of variation of the data.

Actually it is quite likely that not all combinations of these contextual variables will occur in the training set or that some combinations will be rare. To overcome these problems one has to design methods that are first able to take into account contextual information and second are able to disentangle the influence on these many contextual factors. This latter property would make possible generalizing to any data whatever the combination of the contextual variables would occur or not in the training set.

Some data-driven models have been studied for dealing with motion style [76, 4, 75]. Although researchers have achieved some great results, learning style representation is still a challenging task, we will focus in this chapter on variants of Hidden Markov Models that allow taking into account external variables to modify the density a HMM implements. The original work by [89] on parametric HMMs was further explored in [69] where a contextual HMM (CHMM) is proposed for conditioning the probability distribution of a HMM. A CHMM exploits an additional information provided for each training sequence (in its simplest form it may be a discrete label) and that informs about the context of the sequence. It might represent the age or the gender of an actor performing a motion or anything that is assumed to slightly modify the density of sequences. Such an information is used to parameterize the HMM's parameters and in particular the mean and covariance matrix of all Gaussian probability density functions in all states. Ding et al [30] proposed a fully parameterized HMM variant where state transition probabilities are parameterized as well. These works have shown the ability of CHMMs to perform high quality avatar animation from speech [28] and for laughter

animation [29].

The work in this chapter is a preliminary step in this direction that we explore with Contextual Hidden Markov Models. To simplify the study we focus on activity classification task only that we want to perform with the Emilya dataset whose motion have been performed under specific emotion. While a naive approach would consist in learning one HMM model for every activity performed under each emotion, we explore ways to share parameters between these models and propose to learn distributed representation of emotion enabling to deal with (activity, emotion) that have not been encountered in the training set.

In the following sections, we first present CHMMs in section 4.2.1 and we explain in section 4.2.2 how these models could be used for sharing parameters between activity models in order to cope with rare (activity, emotion) settings. Then we propose in section 4.3 to learn a distributed representation of the context information jointly to the CHMM parameters to cope with the setting where not all combinations of (class, context) have been observed in the training set. We provide experimental results in section 4.4.

4.2 Sharing parameters for activity classification with CHMMs

We first describe how one may use CHMMs to efficiently learn models of activity while samples of these activities have been performed in a few contexts, e.g. various emotions. We first remind what CHMMs are then we describe how to design a system that would share parameters between activity models using a classical one-hot encoding of contextual variables as it has been done in previous works. We finally discuss the limitation of this strategy for our goals.

4.2.1 Contextual Hidden Markov Models

We introduce the contextual hidden markov models on which our proposed approach is based. Contextual Hidden Markov Models (CHMMs) [69] are a variant of HMMs that have originally been proposed as an extension of parametric HMMs [89] for dealing with variability in the data. The main idea is to make some or all of HMMs parameters (Gaussian means, Covariance matrices and transition probabilities) dependent on what we hereafter call external variables. These variables might represent additional contextual variables like the gender of a gesturer in a gesture recognition task, the estimated

signal to noise ratio in a speech recognition task... The contextual variables then represent some available additional information that bears some information on the data which is not easy to integrate in a HMM based recognition system. Such a modeling framework has been used both to design accurate recognition systems and to design synthesis systems that are conditioned on some input, e.g. to animate an avatar based on its speech signal [28].

We now explain how the contextual variable is taken into account in a CHMM. Consider an observation sequence x and the associated contextual information θ, that we consider from now on a vector of variables.

Let consider a CHMM and let focus on the simpler case where only Gaussian means are parameterized in order to make a much clearer explanation of CHMMs (note that both Gaussian means and covariance matrices may be parameterized by the contextual input θ [69]). Assume that the CHMM model contains M states $\mathcal{S} = (s_1, s_2, ...s_M)$ and that the emission distribution $p(x_t|s_j)$ is a Gaussian distribution $\mathcal{N}(\mu_j, \Sigma_j)$ with a parameterized mean μ_j and a non parameterized covariance matrix Σ_j.

When dealing with the input sequence x with associated contextual variables θ the parameterized mean of the Gaussian distribution in state s_j, μ_j is defined according to:

$$\mu_j = \bar{\mu}_j + W_j\theta \tag{4.1}$$

where $\bar{\mu}_j \in \mathbb{R}^d$ is a context independent mean (i.e. an offset vector) and $W_j \in \mathbb{R}^{d\times c}$ are parameters that determine how θ modifies the offset mean $\bar{\mu}_j$ (d is the dimension of the observation space and c is the dimension of θ, $c = |\theta|$, i.e. number of contextual variables). If θ variables were useless and did not include information to model the data, then W_j could be set to 0 and one would recover a standard HMM whose Gaussian distribution in state j is $\bar{\mu}_j$ and is independent on θ. In the above definition of μ_j, θ is given and $\bar{\mu}_j$ and W_j are the parameters to be estimated during the learning process. Actually when θ is known, means of Gaussan distribution may be computed according to Eq. 4.1 and a CHMM model is instanciated as a standard HMM.

The conditional dependencies between variables in a CHMM are illustrated in Fig.4.1. Moreover in [69], not only mean μ but also the covariance matrix Σ are parameterized by contextual input θ. By doing this, the shape of the gaussian distribution may be controlled by the contextual information, as shown in Figure 4.2 .

Training a Contextual HMM is similar to the training of a HMM and relies on the Baum-Welch Expectation Maximization(EM) algorithm [68, 69]. Assume we have a set

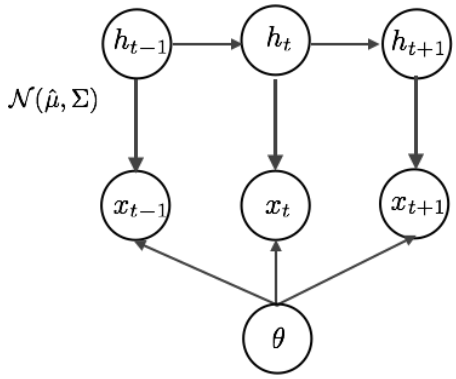


Figure 4.1: Conditional dependencies in a CHMM

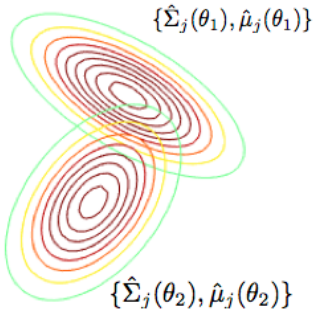


Figure 4.2: How a covariance matrix may be changed according to contextual variables. The two Gaussian distribution are two instances of a Gaussian distribution in a CHMM state computed with two different θ (from [69])

of N sequences $\{\boldsymbol{x}^k\}_{k=1}^N$, where $\boldsymbol{x}^k \in \mathcal{X}$ is the k^{th} sequence of observation vectors $\boldsymbol{x}^k = (x_1^k, ..., x_T^k)$. Assume that a (or a vector of) contextual variable θ^k is available for each sequence in the training set $\boldsymbol{x}^k$. We then note the complete dataset as $\mathcal{D} = \{(\boldsymbol{x}^k, \theta^k)\}_N$ where each sample consists of an observation sequence and the value of the contextual variable.

The details of EM algorithm for training HMM and notations of HMMs can be found in section 2.1. In the case of CHMMs, in the Maximisation-step, one updates the parameters of the CHMM by optimizing the auxiliary function (see Eq 2.8) of the objective function which yields the following update for the transformation matrix W_j [69]:

$$W_j = \left[\sum_{k,t} \gamma_t^k(j)(x_t^k - \bar{\mu}_j)(\theta^k)^\top\right]\left[\sum_{k,t} \gamma_t^k(j)\theta^k(\theta^k)^\top\right]^{-1} \tag{4.2}$$

where we use standard notations for HMMs as in chapter 2.1: $\gamma_t^k(j)$ denotes the prob-

ability of being in state S_j at time t given the observed sequence k (see Eq 2.2), θ^k stands for the contextual variables associated to the k^{th} sequence, and x_t^k denotes the observation vector (the frame) at time t in the k^{th} sequence.

4.2.2 Using one-hot encoding of discrete contextual variables

We describe a first strategy for learning CHMMs when samples of activities have been performed in a few contexts, e.g. various emotions. It is inspired from works in [69, 29].

Assume that there are M activities $\mathcal{A} = \{a_1, ..., a_M\}$ performed under N different contexts $\mathcal{C} = \{c_1, ..., c_N\}$ (e.g. emotions) in the training set. Any training sequence then comes with a pair of labels (a, c), namely its corresponding activity a and emotion c.

There are two main ways for doing activity classification using HMMs in such a setting. The first one is to learn one model per activity a_j from all training sequences corresponding to this activity, i.e. training sequence x with label (a_j, c). This model, being learned for all the contexts, should be designed to be robust to variability brought by the various contexts, which is not easy to achieve. The other one is to learn $M \times N$ models, i.e. a model for every possibility of activity and context (a_i, c_j). The second method takes into account the contextual information during training stage, so it is expected that one can get better performance of activity classification through this method. However, for leveraging the contextual information, one must train one HMM for each (activity, context) pair, which results in $M \times N$ HMMs to be trained. This might make this strategy unscalable to the number of contexts. Besides, a training set for a specific pair (a_i, c_j) might be too small to correctly learn all of these models and it may even happen that there are no samples for some label pair (a_i, c_j) in the training set.

In this situation CHMMs offer an alternative way to design an activity classification system. One can learn one CHMM per activity a_i, noted ϕ^{a_i}, from all training sequence x with activity label a_i and contextual label c_j. The context label c_j may be encoded as a contextual vector denoted by $\theta(c_j)$. Doing so one can smartly handle variability brought by the contexts. Some parameters are shared (e.g. transition probabilities, covariance matrices and mean offsets $\bar{\mu}$) by subsets of sequences having different contextual labels which might make the model able to learn on a training set with fewer training samples for each (a_i, c_j) pair.

One key issue is to define the encoding of the contexts denoted by $\theta(c_j)$ (for $c_j \in \mathcal{C}$) since these will strongly affect the behavior of the method. A simple and natural choice is to use what is known as one-hot codes: In that case the vector $\theta(c_j)$ corresponding

to the j^{th} possible context c_j is a M dimensional vector whose components are all set to zero except the j^{th} one which is set to one.

$$\begin{aligned} &\theta(c_j) \in \mathbb{R}^N \\ &\theta(c_j)(k) = 1 \text{ iff } k = j \end{aligned} \tag{4.3}$$

At test time, given a test sequence $\mathbf{x}$ either its associated contextual label c is known, and predicting the activity is done with:

$$a_* = \arg\max_a p(x|\phi^a_{\theta(c)}) \tag{4.4}$$

Or, if the context of a test sequence $\mathbf{x}$ is unknown, predicting the activity is done through:

$$a_* = \arg\max_a \left[\arg\max_c p(x|\phi^a_{\theta(c)})\right] \tag{4.5}$$

Actually this strategy allows actually sharing part of the parameters of the CHMM models. Indeed the mean of a Gaussian distribution in state s_i for a sequence corresponding to the contextual variable equal to c is defined as:

$$\mu_i = \bar{\mu}_i + W_i\,[:, c] \tag{4.6}$$

where $W_i\,[:, c]$ stands for the c^{th} column of matrix W_i.

Although this strategy has been successfully in previous studies, it cannot be used in the context where some combinations of contextual variables have not been seen in the training stage. We build on this strategy to propose a novel algorithm in the next section.

4.3 Zero shot learning via distributed context representation learning

The strategy that we just described has been explored for dealing with sequences performed under various contexts and styles [69, 29] and actually allowed a finer modeling by taking into account the contextual variable effect while keeping the learnability feasible

by sharing part of the parameters between different contexts (in the case we described $\bar{\mu}$ parameters).

One could go a little further and use a distributed representation of the contextual information instead of the one hot encoding described previously. Actually one hot encoding does allow to share part of the parameters for dealing with sequences performed in different contexts but a better sharing strategy could be achieved if one was able to project discrete contextual variables in a continuous representation space where each dimension would represent a particular feature of the observation sequences. For instance one may imagine that the emotion will alter the motion sequence corresponding to a given activity according to few factors of variation like the speed of the motion, the straightness of the body etc. Rather than modeling these effects separately for each emotion one could first project emotion in a representation space whose dimensions are related to the speed, the straightness etc, then use the modeling strategy described above to learn models of activity under emotion. One key advantage of this approach is to actually share the full set of parameters, with the extra avantage that the learned models could be used for (activity, emotion) settings which have not been seen in the training stage.

Unfortunately the representation of discrete contextual variables (e.g. emotions) in such a distributed representation space is unknown and hard to guess. What we propose here is an algorithm for jointly learning all the parameters of the system, the CHMM parameters as well as the distributed representations of emotions.

We firstly describe the procedure that we propose and the corresponding reestimation formulae for contextual variables, and explain how we derive theses results.

4.3.1 Joint Learning of CHMMs parameters and of context representation

We focus now on the case where instead of being given the θ's for all contexts at training time (e.g. one ht encoding), these θ's are treated as unknown parameters that should be learned with W_j's and with all other parameters of the CHMMs. Since θ's and W's interplay there is no closed form solution to embed in a standard EM algorithm. Instead we propose a coordinate ascent like algorithm that alternates between optimizing the CHMM parameters (W's and HMM parameters) while θ's are kept fixed, then optimizing θ's while CHMM parameters remain fixed (See Algorithm 2). In that case both steps may be efficiently performed with standard EM. Reestimation formulas for θ's may be

obtained analogously to equation 4.2, given that all other parameters are kept fixed. Naturally, updating θ_l is based on all training sequences x whose context label c is c_l. The sums above range over all sequences k whose contextual/emotion label is l, i.e. $c(x^k) = l$.

$$\theta_l = \Big[\sum_{k,t,j} \gamma_t^k(j) W_j^\top \Sigma_j^{-1} W_j\Big]^{-1} \Big[\sum_{k,t,j} \gamma_t^k(j) W_j^\top \Sigma_j^{-1}(x_t^k - \bar{\mu}_j)\Big] \tag{4.7}$$

Algorithm 2 Joint training of CHMMs and of contextual vectors

```
procedure TRAINING
    Train one HMM per activity, yielding M HMMs.
    Initialize CHMMs from HMMs
    Randomly initialize θ vectors
    repeat
        Fix θ's and optimize W's using EM (Cf. Eq. 4.2)
        Fix W's and optimize θs using EM (Cf. Eq. 4.7)
    until Convergence
end procedure
```

4.3.2 Derivation of reestimation formulae for θs

Particular EM algorithms are derived by considering the auxiliary function $Q(\phi'|\phi)$, where ϕ denotes the current value of the parameters of the model, and ϕ' represents the updated value of the model parameters. Q is the expectation of the log probability of the observable and hidden data together given the observables and parameters ϕ, and it has been denoted in Eq 2.8. where $\mathbf{x}$ is the observable data and $\mathbf{q}$ is a hidden sequence. This corresponds to the "Expectation" step in EM algorithm. It is proved that if ϕ' is chosen to increase the value of the auxiliary function Q ,then the likelihood of the observed data $P(\mathbf{x}|\phi)$ increases as well. This updating of values of ϕ' is the "Maximization" step. The auxiliary function Q can be further written as follows by substituting transition probablities a_{ij}:

$$Q(\phi'|\phi) = \mathbb{E}_{\mathbf{q}|\mathbf{x};\phi}\Big[\log \prod_t a_{q_{t-1}q_t} P(\mathbf{x}_t|q_t;\phi')\Big] \tag{4.8}$$

Using the Markov property, Eq 4.8 can be written as follows:

$$\begin{aligned} Q(\phi'|\phi) &= \mathbb{E}_{\mathbf{q}|\mathbf{x};\phi}\Big[\sum_t \log a_{q_{t-1}q_t} + \sum_t \log P(\mathbf{x}_t|q_t;\phi')\Big] \\ &= \sum_t \mathbb{E}_{\mathbf{q}|\mathbf{x};\phi}\Big[\log a_{q_{t-1}q_t} + \log P(\mathbf{x}_t|q_t;\phi')\Big] \\ &= \sum_{t,j} P(q_t = j|\mathbf{x};\phi)\Big[\sum_i P(q_{t-1} = i|\mathbf{x};\phi)\log a_{ij} + \log P(\mathbf{x}_t|q_t = j;\phi')\Big] \end{aligned} \tag{4.9}$$

As it is denoted above, we write $P(q_t = j|\mathbf{x};\phi)$ as $\gamma_t(j)$. $\gamma_t(j)$ can be computed by "forward/backward" algorithm as Eq 2.6. Then in the "Maximization" step, we compute ϕ' to increase the Q. By taking the derivative of Eq 4.8 and substituting γ_{tj}, we arrive at:

$$\frac{\partial Q}{\partial \phi'} = \sum_t \sum_j \gamma_t(j) \frac{\frac{\partial}{\partial \phi'} P(\mathbf{x}_t|q_t = j;\phi')}{P(\mathbf{x}_t|q_t = j;\phi')} \tag{4.10}$$

By setting Eq 4.10 to zero, we can solve for ϕ'. The emission probability $P(\mathbf{x}_t|q_t = j;\phi')$ is assumed as a multivariate gaussian distribution with mean μ_j and covariance Σ. μ_j is defined in Eq 4.1 where it includes two parameters W_j and θ. We derive the analytical solution of W_j and θ respectively according to Eq 4.10.

$$\begin{aligned} \frac{\partial Q}{\partial W_j} &= \sum_t \sum_j \gamma_t(j) \frac{\frac{\partial}{\partial W_j} P(\mathbf{x}_t|q_t = j;\phi')}{P(\mathbf{x}_t|q_t = j;\phi')} \\ &= \sum_t \sum_j \gamma_t(j)(\mathbf{x}_t - \hat{\mu}_j)^T \Sigma_j^{-1} \frac{\partial \mu_j}{\partial W_j} \end{aligned} \tag{4.11}$$

By setting Eq 4.11 to zero, we can get the solution of W_j (see Eq 4.2). Similarly for θ, we can get the derivative as follows:

$$\begin{aligned} \frac{\partial Q}{\partial \theta} &= \sum_t \sum_j \gamma_t(j) \frac{\frac{\partial}{\partial \theta} P(\mathbf{x}_t|q_t = j;\phi')}{P(\mathbf{x}_t|q_t = j;\phi')} \\ &= \sum_t \sum_j \gamma_t(j)(\mathbf{x}_t - \hat{\mu}_j)^T \Sigma_j^{-1} \frac{\partial \mu_j}{\partial \theta} \end{aligned} \tag{4.12}$$

By setting Eq 4.12 to zero, we get the solution of θ (see Eq 4.7).

4.4 Experimental Results

4.4.1 DataSet

We used Emiliya dataset (see section 3.1.2) to evaluate our proposed alogrithm. In the preprocessing stage, we ignore the translation coordinates and only keep the 69 joint angles. In order to speed up the learning process, we use PCA to reduce the dimensions of each frame to 20 dimensions. Our model are trained on only two activities: "Simple Walk" and "Walk with something in the Hands" which are much similar and confusable (e.g. sequences of 'SW' in 'Anxiety' may be very similar to "WH', see Figure 4.3). So CHMM could hopefully improve the accuracy of activity classification of these two activities.

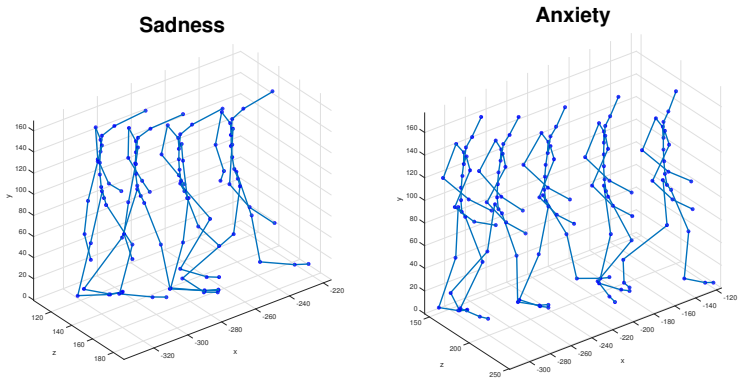


Figure 4.3: 'Simple Walk' in Sadness and Anxiety looks like a motion "Walk with something in the Hands"

4.4.2 Experimental Setting

We report averaged experimental results gained with cross validation on few splits of the training, validation and test data. Stratified cross validation is used so that data from all actors occur in the three datasets. Also in all these experiments we select a few (activity, emotion) pairs that we make only appear in the test set (we call these *missing pairs*) to investigate if the models that we compare may generalize well to cases that have not been observed in the training stage. Test set performances are then reported as *Test performance* based on the results on the test set except on *missing pairs* and as *Missing pairs performance* computed on *missing pairs* only. We compare our approach with a HMM baseline system where we use one HMM per (activity, emotion) combination, yield 64 HMMs. In any case we use one CHMM per activity. Whatever the models (HMMs, CHMMs) each model is a ergodic model (i.e. all transitions are allowed) with 12 states and we use single Gaussian emission probability densities for emission probabilities. All models are trained through Maximum Likelihood Estimation. We trained standard HMM first to initialize most of CHMM parameters, initial state probabilities $\{\pi_i\}$, transition probabilities $\{a_{ij}\}$, means of Gaussian emission probability densities $\{\bar{\mu}_j\}$, and co-variance matrices of Gaussian emission probability densities $\{\Sigma_j\}$. Once HMMs are trained, CHMMs are built by copying the HMM parameters and by initializing W's parameters to small random values, then the CHMM parameters are refined using formulas as in Eq. 4.2 and/or Eq. 4.7.

4.4.3 Activity Classification Results

We first report activity classification results (Figure 4.4). We investigated two settings. In any case recognition of a test sample is performed without any information on it (i.e. the corresponding emotion is unknown). We report results in two settings, by learning with all training data available (models are named *CHMM+* and *HMM+*) and learning with small training sets by randomly choosing one fourth of available training material (models are named *CHMM-* and *HMM-*).

The most advantageous experimental setting here is when one has at his disposal a complete training set, i.e. training samples are available for any label pair (activity, emotion). This setting corresponds to the curves *HMM Train*, *HMM Valid* and *HMM Test* (resp. *CHMM Train/Validation/Test*) which show the performance of the HMM baseline and of our CHMM approach on the three datasets as a function of the dimension of θ (HMM performances are independent θ and plotted as constant). One may see that the while CHMM is outperformed by the HMM baseline on all datasets when enough training data is available (+ curves), CHMM outperforms HMMs when the training data set is smaller (− curves). This comes from the fact that there is a parameter sharing schema between activity-emotion models in CHMMs. Also it is worth noticing that the dimension of θ does not seem to impact much the performance here, meaning one may significantly compress the size of the activity models with respect to the baseline HMM system without decreasing accuracy.

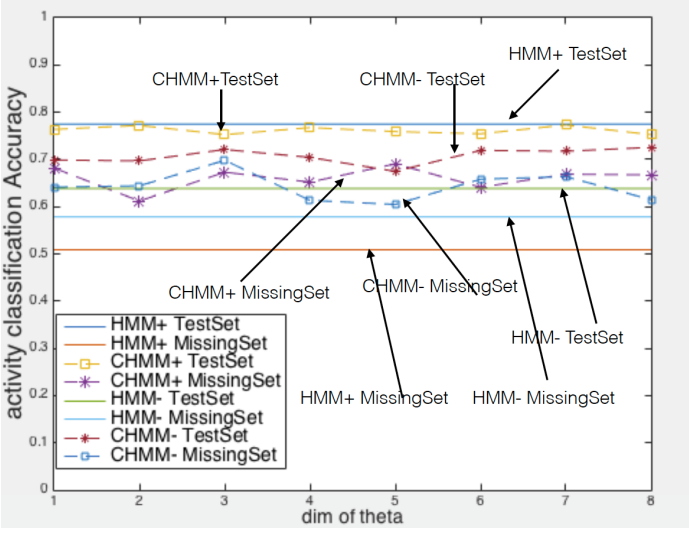


Figure 4.4: Accuracy of Activity Classification wrt. $|\theta|$s.

The figure also shows four additional curves with are called *HMM+/− Missing Pairs*

and *CHMM+/− Missing Pairs*. These curves correspond to the performance on *Missing Pairs* data only, a more difficult setting. Both our approach and the HMM baseline naturally perform lower on these data but CHMM performance drop is significantly smaller than the one of HMMs, especially when less training data is available.

4.4.4 Learning Curve and Convergence of θ

We provide some addition experimental results that provide some light what is happening. To start with, we focus on how the emotion information is used during training. The curves in figure 4.5 shows how the likelihood of the data increases during training of two activity models with 2-dimensional θ's. One clearly sees the first part of the training, up to iteration 25, that correspond to initialization through standard HMM learning, and the second part of the curves from iteration 15 where CHMM parameters and θ's are learned together. We can see there is a great increasing on the likelihood when parameters of CHMM W and θ start being learned, which means that CHMMs significantly is leveraging contextual information and the contextual information does improve the model. Figure 4.6 shows the trajectories described by the θ vectors of the eight emotions along the training process. One sees for instance that the representations of *anxiety* and *shame* move away from one another while *anger* and *joy* become closer. As these results show emotion is clearly taken into account by the markovian models and actually help improve modeling accuracy.

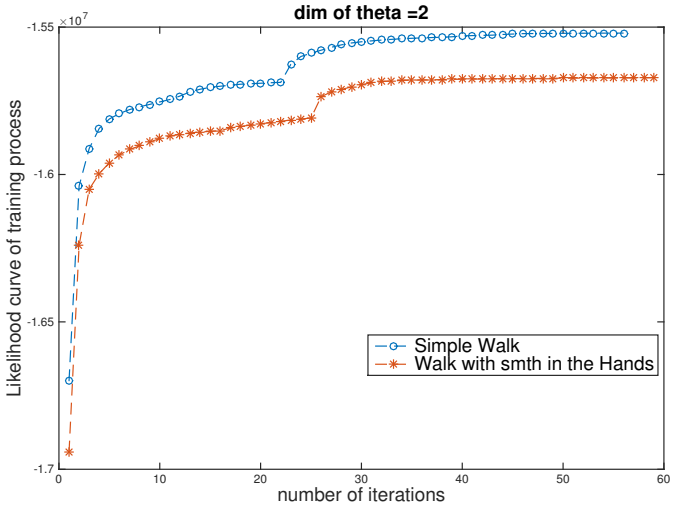


Figure 4.5: Evolution of data Likelihood during training of CHMMs (dimension of $\theta = 2$). Two steps learning: Standard HMM learning up to iteration 25 then initialization of CHMM refinement from iteration 25.

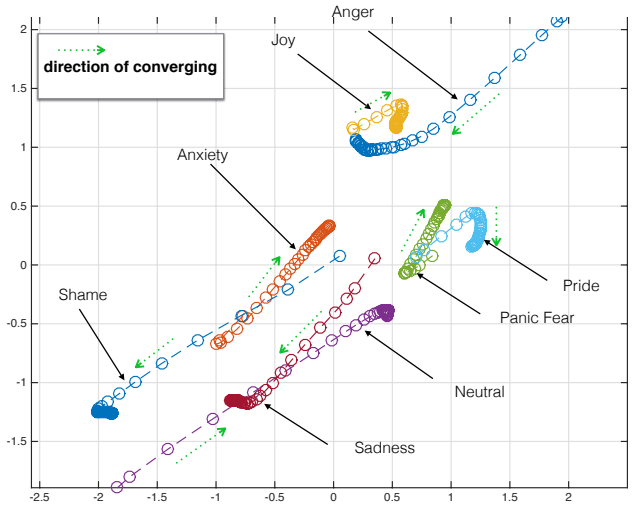


Figure 4.6: Trajectories of the 2D representations (θ) of the 8 emotions along the training process (random initialization).

4.4.5 Emotion Classification

Besides doing activity classification, we also can do emotion classification once that our models are learned. For HMMs, given a test sequence $\mathbf{x}$ we can compute the likelihood of $\mathbf{x}$ on each HMM. We select the HMM on which we obtain the highest likelihood of $\mathbf{x}$, and the emotion corresponding to this HMM is the predicted emotion of $\mathbf{x}$. In Fig 4.7, one sees that the accuracy is reasonably good for the training, validation and test sets when the dimension of θ increases and also that as before CHMMs outperform HMMs in small training set experiments while HMMs are more accurate with large training sets. Looking at missing pairs performance (HMMs cannot perform on these data) it seems quite puzzling that the performance drops to almost random when computed on missing pairs only. This seems a little contradictory with previous promising results and we don't have clear explanations for this phenomenon.

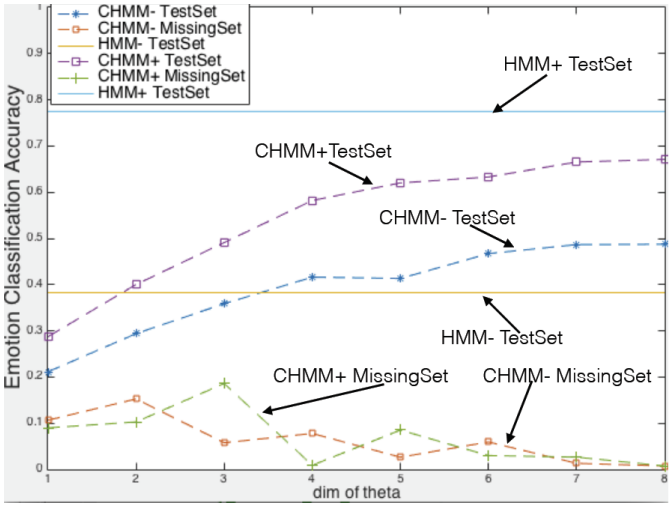


Figure 4.7: Accuracy of Emotion Classification wrt. $|\theta|$.

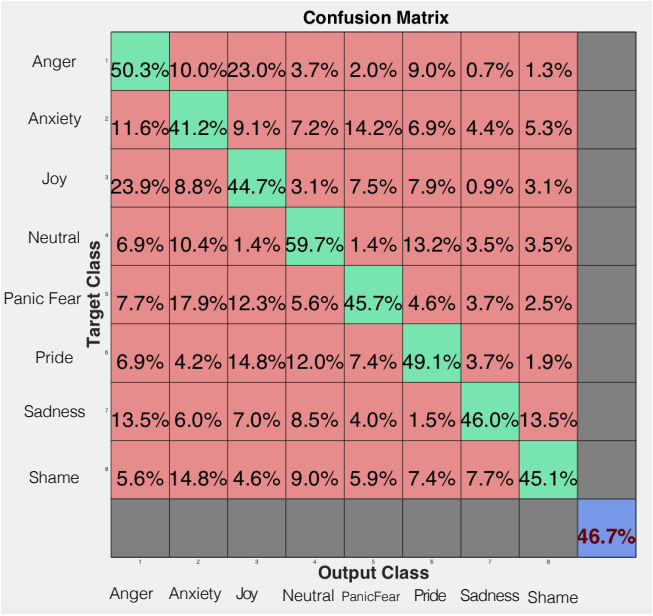


Figure 4.8: Confusion Matrix of CHMM- on Test Set. ($|\theta| = 6$)

4.5 Conclusion

This chapter reports preliminary results that we obtained when trying to exploit contextual HMMs to take into account contextual information in the particular case when some combinations of factors of variation, here the activity and the emotion under which it was performed, were not see at training time. We proposed an algorithm for jointly learning the contextual information representation and the CHMM's parameters. Pre-

liminary experiments show that our proposal enables, up to some extent, recognizing an activity performed under an emotion for which there was no training samples. Yet the results were not as impressive as we expected and the extension of such a kind of models for dealing with more sophisticated synthesis tasks appeared to us a difficult task. We rather decided to explore other statistical frameworks that we describe in the remaining of the thesis.

Chapter 5

Inverse Kinematics using Gaussian Process

We discussed in Chapter 4 how to leverage contextual information to improve the learning of CHMMs and proposed an algorithm for learning continuous representation of contextual information. Although the experimental results are promising, handling variability with CHMMs appeared still challenging and troublesome.

We explore here another family of statistical models for dealing with complex human motion, Gaussian processes, that have also been used for synthesizing human motion for years [54, 82, 39]. These works have shown that Gaussian Processes have a potential to synthesizing realistic human motions. In this chapter we investigate the potential of Gaussian Processes by focusing on a specific task in animation field for which these models appear particularly relevant: Inverse Kinematics (IK) [13].

Inverse kinematics allows one to synthesize human motions from a given trajectory of some joints. Inverse kinematics can be solved through geometric method Jacobian Method [13]. Jacobian method is a geometric method for IK problem. However, its main problem is that the posture generated from the objective function of Jacobian method often looks unrealistic or even wrong although it can satisfy the positional constraints. A straightforward way for solving this problem is to add a penalised term into its objective function in order to penalise wrong postures. We focus on this direction and propose a specific method for solving the ambiguity problem of conventional Jacobian method.

5.1 Introduction

Inverse kinematics have been studied for many decades, see section 3.2.2 for an introduction. The Jacobian method is a geometric method which approximately and iteratively generates a posture which may satisfy given constraints on few joints. This method is very efficient for finding a posture in the solution space. Unfortunately in the solution space, most postures are unrealistic or even wrong so that the Jacobian method often generates an unrealistic or wrong posture. Compared with pure geometric methods, statistical models for IK usually generate more natural and realistic posture since they can learn the pattern from real human posture or motions. For example, Grochow *et al.* proposed a Gaussian Process Latent Variable Model (GPLVM) which is able to generate realistic postures from given positional trajectory of constrained joints [39]. However, this model is not as fast as traditional Jacobian method.

Here, in order to obtain both a good performance on speed of generation and a good quality of generated postures, we proposed an IK method which combines the Jacobian method and Gaussian Process. To do so we introduce an new objective function for the conventional Jacobian method. In this objective function, a prior probability density function over the generated posture is added as a penalization term into the standard objective function of the Jacobian method. This penalty term can penalize the posture which is not realistic or bad-quality. This prior distribution is assumed to be a Gaussian distribution and can be estimated by a Gaussian Process regression model which has been created off-line on a dataset of real postures. We demonstrate that our solver can generate high-quality postures with high real-time speed. In particular our method is much faster than the pure statistical models such as GPLVM [39]. Compared with the traditional geometric method, our method can synthesize more natural postures.

In the following sections, we first introduce the conventional Jacobian method and its objective function in section 5.2. Then in section 5.3, we details the pipeline of our proposed method for inverse kinematics. In section 5.4, the experiment and results are presented.

5.2 Background of Jacobian Method

The Jacobian method [13] is a geometric optimization method for solving the IK problem. As shown in Fig 5.1, assume we have a robotic arm composed with three joints and the joint rotations of the three joints can be represented by their joint angels $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$.

Given a target position e (blue point in Fig 5.1), we want to compute the joint angles of these three joints so that the end-effector/joint (red point in the Fig 5.1) reach the target point. Before we explain the Jacobian method for IK, we introduce the some notations.

- $\boldsymbol{e}$ denotes the coordinates of the target position.
- $\boldsymbol{e}(i)$ is the position of the end-effector/joint at the i^{th} iteration and $\boldsymbol{e}(1)$ represents the initial position of the end-effector/joint.
- $\boldsymbol{\theta}(i)$ represents the joint angles at the i^{th} iteration.
- $\Delta\boldsymbol{e}(i)$ is the vector from $\boldsymbol{e}(i)$ to e, $\Delta\boldsymbol{e}(i) = \boldsymbol{e} - \boldsymbol{e}(i)$
- $\Delta\boldsymbol{\theta}(i)$ the joint angles the joints shall rotate at each iteration i.

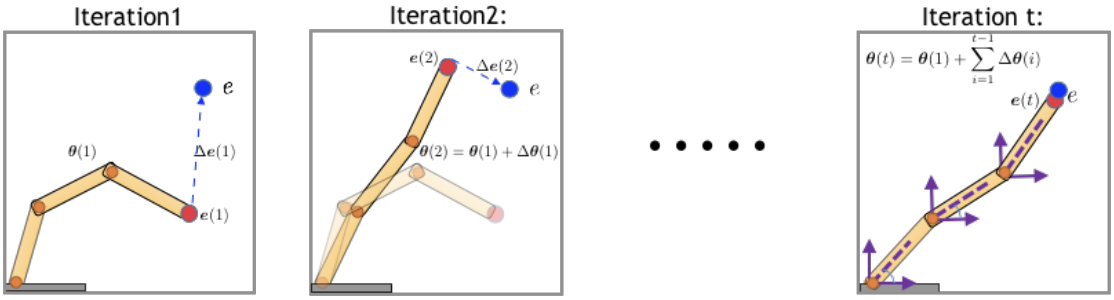


Figure 5.1: Jacobian Method for Inverse Kinematics

The Jacobian method is an iterative method. Instead of computing directly the final joint angles of the the joints, the Jacobian method computes a small step of the joint angles $\Delta\boldsymbol{\theta}(i)$ at every iteration i. After several iterations, we can obtain the final joint angles by summing all these small steps $\Delta\boldsymbol{\theta}(i)$ as follows.

$$\begin{aligned}\boldsymbol{\theta}(t) &= \boldsymbol{\theta}(t-1) + \Delta\boldsymbol{\theta}(t-1) \\ &= \boldsymbol{\theta}(1) + \sum_{i=1}^{t-1} \Delta\boldsymbol{\theta}(i)\end{aligned} \tag{5.1}$$

At each iteration i, the $\Delta\boldsymbol{\theta}(i)$ is computed as follows:

$$\Delta\boldsymbol{\theta}(i) = \arg\min_{\Delta\boldsymbol{\theta}(i)} (||\mathbf{J}\Delta\boldsymbol{\theta}(i) - \Delta\boldsymbol{e}(i)||) \tag{5.2}$$

where $\mathbf{J}$ is the Jacobian matrix. Jacobian matrix is a matrix in which each element is the gradient of the position of the end-effector/joint e with respect to the joint angles $\boldsymbol{\theta}$

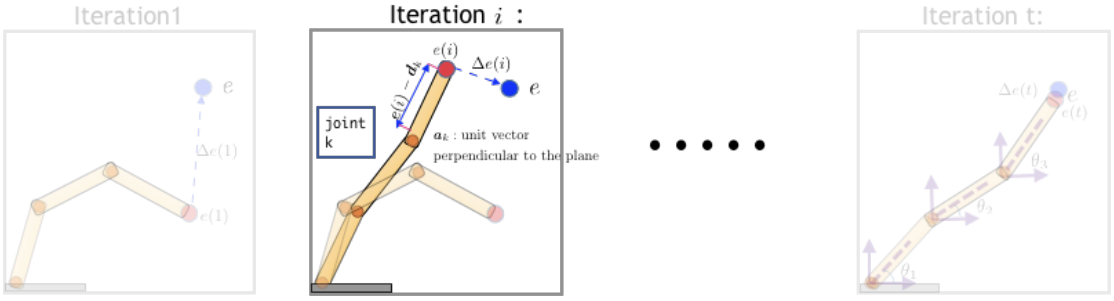


Figure 5.2: Illustration of computing the Jacobian matrix

(see Eq 5.3).

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \boldsymbol{e}_x}{\partial \boldsymbol{\theta}_1} & \frac{\partial \boldsymbol{e}_x}{\partial \boldsymbol{\theta}_2} & \cdots & \frac{\partial \boldsymbol{e}_x}{\partial \boldsymbol{\theta}_n} \\ \frac{\partial \boldsymbol{e}_y}{\partial \boldsymbol{\theta}_1} & \frac{\partial \boldsymbol{e}_y}{\partial \boldsymbol{\theta}_2} & \cdots & \frac{\partial \boldsymbol{e}_y}{\partial \boldsymbol{\theta}_n} \\ \frac{\partial \boldsymbol{e}_z}{\partial \boldsymbol{\theta}_1} & \frac{\partial \boldsymbol{e}_z}{\partial \boldsymbol{\theta}_2} & \cdots & \frac{\partial \boldsymbol{e}_z}{\partial \boldsymbol{\theta}_n} \end{bmatrix} \tag{5.3}$$

where $\boldsymbol{e}_x$,$\boldsymbol{e}_y$,$\boldsymbol{e}_z$ respectively represent the coordinates on x,y,z axis. The Jacobian matrix can be computed by the following vector operation:

$$\mathbf{J} = \boldsymbol{a}_k \times (\boldsymbol{e}(i) - \boldsymbol{d}_k) \tag{5.4}$$

where $\boldsymbol{a}_k$ is the unit vector which is perpendicular to the rotation plane of the joint k. $\boldsymbol{d}_k$ is the positional coordinates of the joint k. By optimizing Eq 5.2, we can get the values of $\Delta\boldsymbol{\theta}(i)$ as follows:

$$\Delta\boldsymbol{\theta}(i) = \mathbf{J}^\top(\mathbf{J}\mathbf{J}^\top + \lambda\mathbf{I})^{-1}\Delta\boldsymbol{e}(i) \tag{5.5}$$

5.3 Inverse Kinematics using Gaussian Process

Given a target position denoted by $\boldsymbol{e}$ and a humanoid character with an initial posture denoted by $\boldsymbol{\theta}^*$, we want to get a realistic and natural posture $\boldsymbol{\theta}$ so that the constrained joint is as close as possible to the target position. As we can see in the Fig 5.3, the left posture is the initial posture of the humanoid character, and the joint marked by the red point is the constrained joint which we expect to reach the target position (the blue point). The left posture is a posture generated by our method, and the constrained joint reached or got very close to the target position while the posture still looks natural and

realistic. Note that the number of the constrained joints can be more than one, but we here use only one constrained joint to illustrate the Inverse Kinematics.

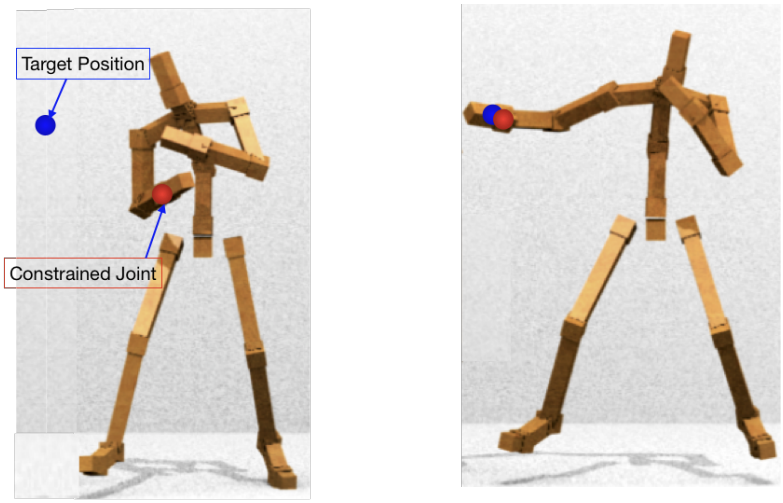


Figure 5.3: Illustration of Inverse Kinematics Problem. Left: the initial posture of the humanoid character. Right: the final posture of the humanoid character. Blue point: the target position. Red point: the constrained joint

As we mentioned above most geometric methods such as the Jacobian method suffer from a problem that the generated postures may be unrealistic or unnatural. We propose a method called Multivariate Gaussian based Inverse Kinematics (MGIK) for addressing this problem. Our method is based on the traditional Jacobian method, while the difference is that we add a penalization term in the original objective function of the standard Jacobian method in Eq 5.2 in order to enforce the generated posture to be realistic. We use a Gaussian Process regression model to learn a mapping from the target position to the parameters ($\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$) of a Gaussian distribution. Therefore, in the synthesis process, given the target position, one can use the Gaussian Process regression model to get a prior distribution. Then one can use the predicted prior to constrain the generated postures by our method. The whole pipeline of the learning and the synthesis process is illustrated in Fig 5.4. Next we will detail the whole learning and synthesis process.

5.3.1 Offline Preprocessing

Data Preprocessing First of all, we use postures extracted from motion capture sequences to construct a dataset $\mathbf{D} = \{\boldsymbol{\theta}^i\}_{i=1}^N$. Each posture $\boldsymbol{\theta}^i = [t_1, t_2, t_3, r_1, r_2, ..., r_m]$. is composed of the 3 degree of freedoms (DOFs) related to global position of joint angles

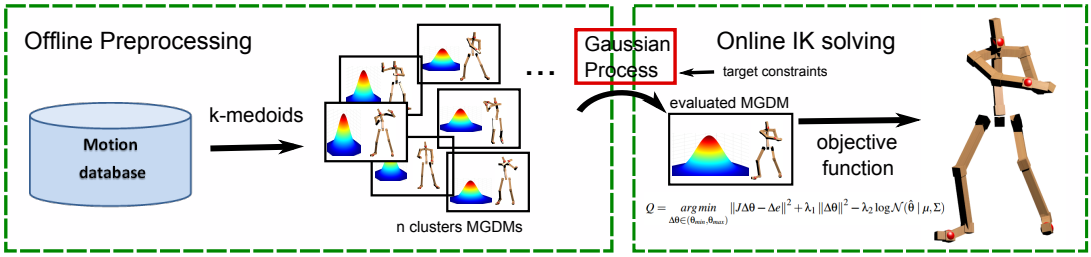


Figure 5.4: Pipeline of the learning and the inference process for the proposed method.

and 3 DOFs related to the global orientation of the root joint and the remaining DOFs related to the rotation of the remaining joints. We remove the global information related to the position and orientation of the root joint, and only keep the rotation angles of the other joints $\boldsymbol{\theta}^i = [r_4, ..., r_m]$. Note that the postures $\boldsymbol{\theta}^i$ has the same representation format with the posture of the humanoid character $\boldsymbol{\theta}$.

Clustering In our method we need to build a prior distribution over postures. However, in the posture dataset $\mathbf{D}$, postures from different activities may vary greatly. Hence it will be very inaccurate if we use a unique Gaussian distribution to model all these postures. In order to deal with the variations in the posture dataset, we cluster the large posture dataset into M clusters and we assume postures in one same cluster $j \in (1, 2, \cdots, M)$ satisfy a Gaussian distribution $p_j(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\mu}^j, \boldsymbol{\Sigma}^j)$. The parameters $\boldsymbol{\mu}^j$ and $\boldsymbol{\Sigma}^j$ can be estimated from the postures of cluster j. The cluster number M is a hyper-parameter that has to be set manually.

5.3.2 Predicting Gaussian Parameters using Gaussian Processes

Here we introduce how to use Gaussian Process to obtain a good Gaussian prior in order to constrain the generated posture. We have M candidates of Gaussian distributions $\{p_j(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\mu}^j, \boldsymbol{\Sigma}^j)\}_{j=1}^M$. One straight forward way is to select one of the M Gaussian distributions while we generate each posture. For example, one can select a Gaussian distribution whose centroid is the closest to the initial posture. However, different Gaussian distribution $p_j(\boldsymbol{\theta})$ corresponds to different clusters and then yields different final postures. Especially, when one is generating a motion sequence from a sequence of target positions, we always expect the generated motion sequence smooth and coherent. Allowing too much freedom in the choice of the distribution for each generated posture will result in the incoherency of the generated motion sequence. In order to avoid this problem, we trained a Gaussian Process Regression Model (see section 2.2.3) to learn a

mapping from the target position $\boldsymbol{e}$ to a Gaussian distribution whose parameters are $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

We firstly built a dataset for training Gaussian process regression model. For each cluster $j \in (1, 2, ..., M)$, we denote its centroid posture by $\theta^{c(j)}$. For each centroid posture $\theta^{c(j)}$, we compute the position of the constrained joint and denote it by $\boldsymbol{e}^j$. We then built a dataset $\{\boldsymbol{e}^j, \boldsymbol{\mu}^j, \boldsymbol{\Sigma}^j\}_{j=1}^M$. We build a regression model from the target position $\boldsymbol{e}^{(j)}$ as follows:

$$\begin{aligned} \boldsymbol{\mu}^j &= f(\boldsymbol{e}^j) + \boldsymbol{\epsilon}_1^j, j \in (1, ..., M) \\ \boldsymbol{\Sigma}^j &= g(\boldsymbol{e}^j) + \boldsymbol{\epsilon}_2^j, j \in (1, ..., M) \end{aligned} \tag{5.6}$$

We assume both $f(\cdot)$ and $g(\cdot)$ satisfy a Gaussian Process as follows:

$$\begin{aligned} f(\cdot) &\sim \mathcal{GP}(0, k(\cdot, \cdot)) \\ g(\cdot) &\sim \mathcal{GP}(0, k(\cdot, \cdot)) \end{aligned} \tag{5.7}$$

The prediction error $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ have a normal distribution as follows:

$$\begin{aligned} \epsilon_1^j &\sim p(\epsilon_1) = \mathcal{N}(0, \sigma_1^2) \\ \epsilon_2^j &\sim p(\epsilon_2) = \mathcal{N}(0, \sigma_2^2) \end{aligned} \tag{5.8}$$

Then according to the Gaussian Process Regression Model as introduced in section 2.2.3, we can predict the values of each element of $\boldsymbol{\mu}^*$ at row a and $\boldsymbol{\Sigma}^*$ at row a and column b given a new target position $\boldsymbol{e}^*$ as follows:

$$\begin{aligned} \boldsymbol{\mu}_a^* &= \mathbf{k}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathcal{M}_a \\ \boldsymbol{\Sigma}_{ab}^* &= \mathbf{k}^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathcal{S}_{ab} \end{aligned} \tag{5.9}$$

where $\mathcal{M}_a = (\boldsymbol{\mu}_a^1, ..., \boldsymbol{\mu}_a^M)^\top$ and $\mathcal{S}_{ab} = (\boldsymbol{\Sigma}_{ab}^1, ..., \boldsymbol{\Sigma}_{ab}^M)^\top$. $\mathbf{K}$ and $\mathbf{k}$ are defined respectively in Eq 5.10 and Eq 5.11:

$$\mathbf{K} = \begin{bmatrix} k(\boldsymbol{e}^1, \boldsymbol{e}^1) & k(\boldsymbol{e}^1, \boldsymbol{e}^2) & \cdots & k(\boldsymbol{e}^1, \boldsymbol{e}^M) \\ k(\boldsymbol{e}^2, \boldsymbol{e}^1) & k(\boldsymbol{e}^2, \boldsymbol{e}^2) & \cdots & k(\boldsymbol{e}^2, \boldsymbol{e}^M) \\ \vdots & \vdots & \cdots & \vdots \\ k(\boldsymbol{e}^M, \boldsymbol{e}^1) & k(\boldsymbol{e}^M, \boldsymbol{e}^2) & \cdots & k(\boldsymbol{e}^M, \boldsymbol{e}^M) \end{bmatrix} \tag{5.10}$$

$$\mathbf{k} = \begin{bmatrix} k(\boldsymbol{e}^1, \boldsymbol{e}^*) \\ k(\boldsymbol{e}^2, \boldsymbol{e}^*) \\ \vdots \\ k(\boldsymbol{e}^M, \boldsymbol{e}^*) \end{bmatrix} \tag{5.11}$$

We choose a RBF kernel (see Eq 2.13) as the kernel function of our Gaussian process.

At last we can obtain the predicted Gaussian prior as follows:

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*) \tag{5.12}$$

5.3.3 Online Synthesis

We have described the standard Jacobian method in section 5.2. Now based on Jacobian method and the predicted gaussian prior distribution in Eq 5.12, we devised our objective function for each iteration of Jacobian method.

For each iteration i, the objective function is defined as:

$$\mathcal{L} = \|\mathbf{J}\Delta\boldsymbol{\theta}(i) - \Delta\boldsymbol{e}(i)\|^2 + \lambda_1 \|\Delta\boldsymbol{\theta}(i)\|^2 - \lambda_2 \log \mathcal{N}(\boldsymbol{\theta}(i+1) \mid \boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*) \tag{5.13}$$

where $\boldsymbol{\theta}(i+1) = \boldsymbol{\theta}(i) + \Delta\boldsymbol{\theta}(i)$ and $\boldsymbol{\mu}^*$ and $\boldsymbol{\Sigma}^*$ are defined in Eq 5.9. The other notations have been defined in section 5.2. The second term in the Eq 5.13 is a regulariser for $\Delta\boldsymbol{\theta}(i)$ and it makes the values of $\Delta\boldsymbol{\theta}(i)$ not too large for each iteration.

By minimizing the objective function $\mathcal{L}$, one can get the solution of $\Delta\boldsymbol{\theta}(i)$ for each iteration i.

$$\Delta\boldsymbol{\theta}(i) = \arg\min_{\Delta\boldsymbol{\theta}(i)} \|\mathbf{J}\Delta\boldsymbol{\theta}(i) - \Delta\boldsymbol{e}(i)\|^2 + \lambda_1 \|\Delta\boldsymbol{\theta}(i)\|^2 + \lambda_2 [\frac{1}{2}(\boldsymbol{\theta}(i+1) - \boldsymbol{\mu}^*)^\top (\boldsymbol{\Sigma}^*)^{-1} (\boldsymbol{\theta}(i+1) - \boldsymbol{\mu}^*)] \tag{5.14}$$

We set $\frac{\partial \mathcal{L}}{\partial \Delta\boldsymbol{\theta}} = 0$ to minimize the objective function. The function can be solved as:

$$\Delta\boldsymbol{\theta}(i) = (2\mathbf{J}^\top\mathbf{J} + 2\lambda_1 + \lambda_2(\boldsymbol{\Sigma}^*)^{-1}(2\mathbf{J}^\top\Delta\boldsymbol{e}(i) + \lambda_2(\boldsymbol{\Sigma}^*)^{-1}\boldsymbol{\mu}^* - \lambda_2(\boldsymbol{\Sigma}^*)^{-1}\boldsymbol{\theta}(i)) \tag{5.15}$$

The final posture at the last iteration can be computed by Eq 5.1.

Dynamically Tune the Hyperparameter λ_2 Because our method is based on Jacobian method which is an iterative optimization method. The first two terms in the objective function (Eq 5.13) will decrease dramatically as the iteration goes on. Hence the hyperparamter λ_2 cannot be fixed for all the iterations, otherwise the last term will dominate the whole loss function as the iteration goes on and the first term may never be close enough to zero. In order to avoid this problem, we dynamically tune the values of the hyperparamter λ_2 and make it proportional to the $\Delta \boldsymbol{e}(i)$.

5.4 Experiments

We compared our proposed method with two baselines: the Jacobian Damped Least Squares (JDLS) [14] which is a variant of Jacobian method and the Gaussian Process (GP) regression model which takes the target position as input and predicts the posture. For our method, we set the number of the clusters to 10 and 100 and we refer to these two setting as MGIK10 and MGIK100. For the dataset, we use CMU dataset 3.1.2 and select 10 sequences of sports including tennis, golf,shooting, boxing. We treat the hand joints, feet joints, and neck joint as the constrained joints (red points shown in Fig 5.5). We use the trajectories of the 5 constrained joints of different motion sequences as input to reconstruct the whole motion sequence using our methods MGIK and the baseline mothods. We report the target error which is the distance between the constrained joints and the target positions at the final iteration in Table 5.2. Meanwhile we evaluate the joint error using the mean square error between the reconstructed postures and the groundtruth. The results show that our method MGIK10 and MGIK100 achieve smallest error on both target error and joint error.

The reconstructed motions are shown in Fig 5.5. In Fig 5.5, we show a reconstructed motion "golf" from our method and the baselines. As we can see, JDLS generated some unnatural postures. GP generates natural postures from large data set; however, it distance between its constrained joints and position is large. In contrast, with our method, the constrained joints can reach the target position well and the reconstructed postures are similar to the true postures.

Then we evaluate the generation speed of different methods. The test is performed by an Intel Xeon CPU 2.93GHz with one process unit. A performance comparison for solving one posture using various methods is shown in Tab. 5.1, and the speed performance is counted by millisecond (msc) per frame. In general, our solution performs worse in computation speed due to the run-time MGDM construction using GP. However, it is

methods	average msc	reaching targets	posture
MGIK10	1.1	yes	natural
MGIK100	3.1	yes	natural
JDLS	0.55	yes	unnatural
GP	2.31	no	natural

Table 5.1: Performance comparison of our solution using 10 clusters, 100 clusters and with other methods: Damped Least Squares IK (JDLS) [14], Gaussian Process IK (GP).

stable and able to generate natural postures with multiple constraints. It is also much faster than other machine learning techniques such as [40, 53, 91]. Considering that both the Jacobian solutions and matrix operations are parallelizable [41], the speed of our MGIK algorithm can be further improved by using multiple threads programming. More examples are shown in Tab. 5.2 by generating motion sequences given 5 target-constraint trajectories for head, left wrist, right wrist, left foot, right foot. We evaluate the speed performance in millisecond (msc) for each whole sequence. The mean squared error (MSE) (averaged by all 66 DOFs) in joint space (joint MSE in radian) and the MSE of the distance (in meter) between end-effectors and targets (averaged by 5 end-effectors) are also computed. Different from the distance metric, a small difference in joint space error metric may show a large variation for a specific joint. Our MGIK solution offers high quality results both in joint space and targeting space compared to other methods. For the speed performance, the most expensive step is the run-time estimation of the gaussian distribution. By reducing the size of GP kernel, the performance can be improved significantly.

Applications The MGIK solution can be used for various applications with high quality virtual character simulation, such as posturing, motion sequence editing (Fig 5.6), motion reconstruction, etc. More examples can be found in https://bit.ly/2K54ZuV.

5.5 Conclusion

We designed a new objective function for the conventional Jacobian method to solve the inverse kinematics problem. We leverage the Gaussian process regression model to fit on M Gaussian distributions to predict a prior distribution according to the given target position. Then the predicted Gaussian distribution is merged into the objective function of the traditional Jacobian method. The experimental results demonstrate that our method is fast and accurate and is able to generate natural and realistic postures, compared to the baselines. As an extension of the conventional Jacobian solution, this

ex.	methods	msc	targets mse	joints mse
tennis	MGIK10	3426.7	0.00329	0.151
3050	MGIK100	9737.5	0.00385	0.109
frames	JDLS	1582.42	0.00863	0.288
	GP100	6813.11	0.250	0.106
golf	MGIK10	3330.11	0.00197	0.150
2900	MGIK100	9201.82	0.00202	0.0810
frames	JDLS	1399.58	0.00864	0.239
	GP100	6473.05	0.151	0.0653
shooting	MGIK10	4321.12	0.00178	0.127
3850	MGIK100	12001.2	0.00237	0.0902
frames	JDLS	1870.23	0.00879	0.212
	GP100	8593.11	0.213	0.0838
boxing	MGIK10	3529.44	0.00275	0.228758
3069	MGIK100	9914.28	0.00360	0.111023
frames	JDLS	1687.35	0.00826	0.37863
	GP100	6845.56	0.396	0.118878

Table 5.2: Comparison using 5 target constraints between our solution (MGIK) using 10 clusters, 100 clusters and other methods: Damped Least Squares IK (JDLS) [14], Gaussian Process IK (GP).

method could be integrated into traditional animation pipelines. Although by solving IK, we were able to generate realistic postures and to edit motion sequence, using such models to design a fully generative framework still appeared a hard task. And the deep learning wave, especially the success of LSTM recurrent models for sequences, combined with the invention of adversarial learning for learning powerful generative models for complex data, incited us to move to explore neural networks, which we describe in the next part.

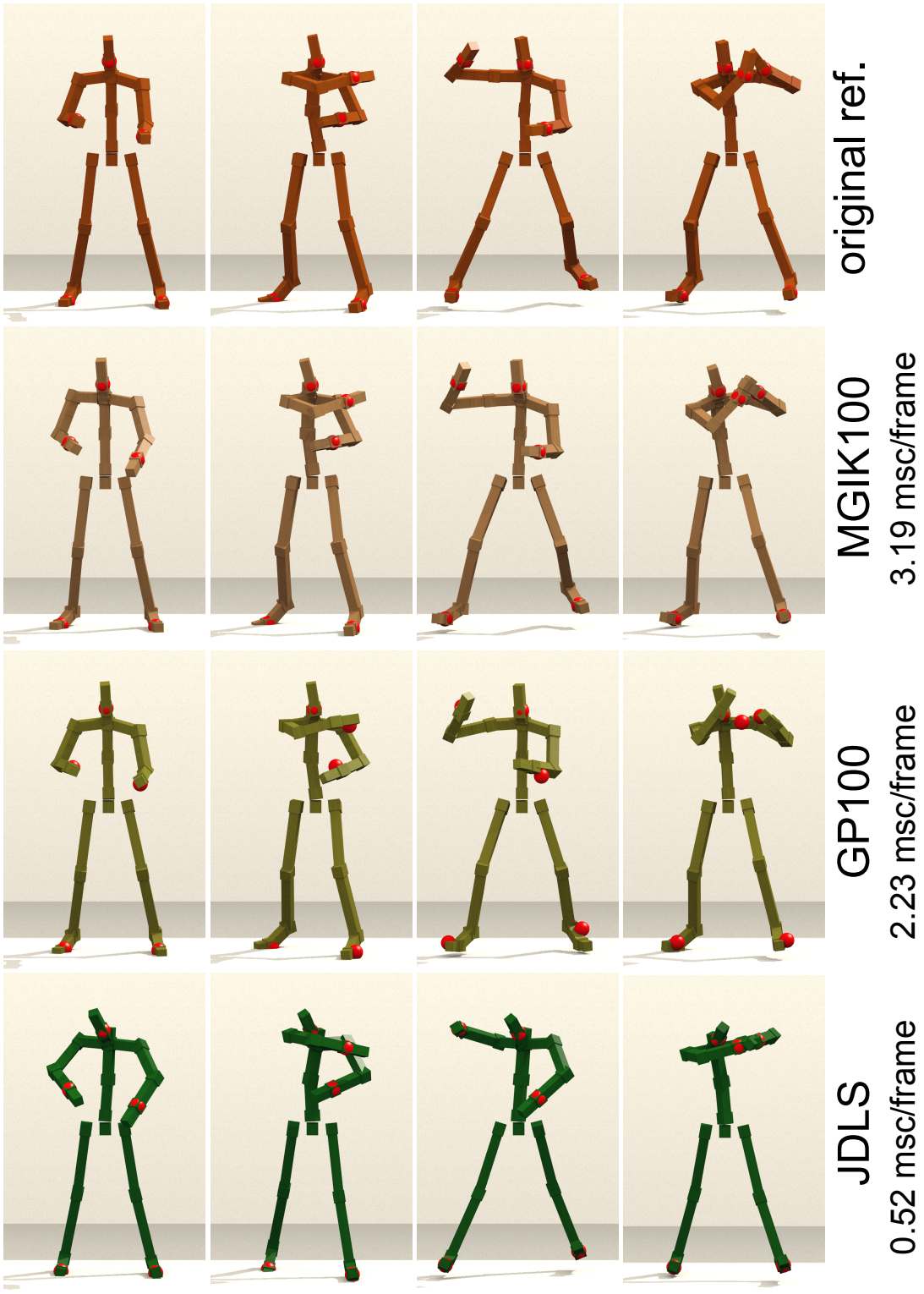


Figure 5.5: The figure illustrates the generated postures using 4 different solutions according to the given targets (red ball) in the tennis example, from top to bottom: original mocap data, our MGIK (100 cluster samplers), Gaussian process (GP) (100 samplers), JDLS (without constraints) [14].

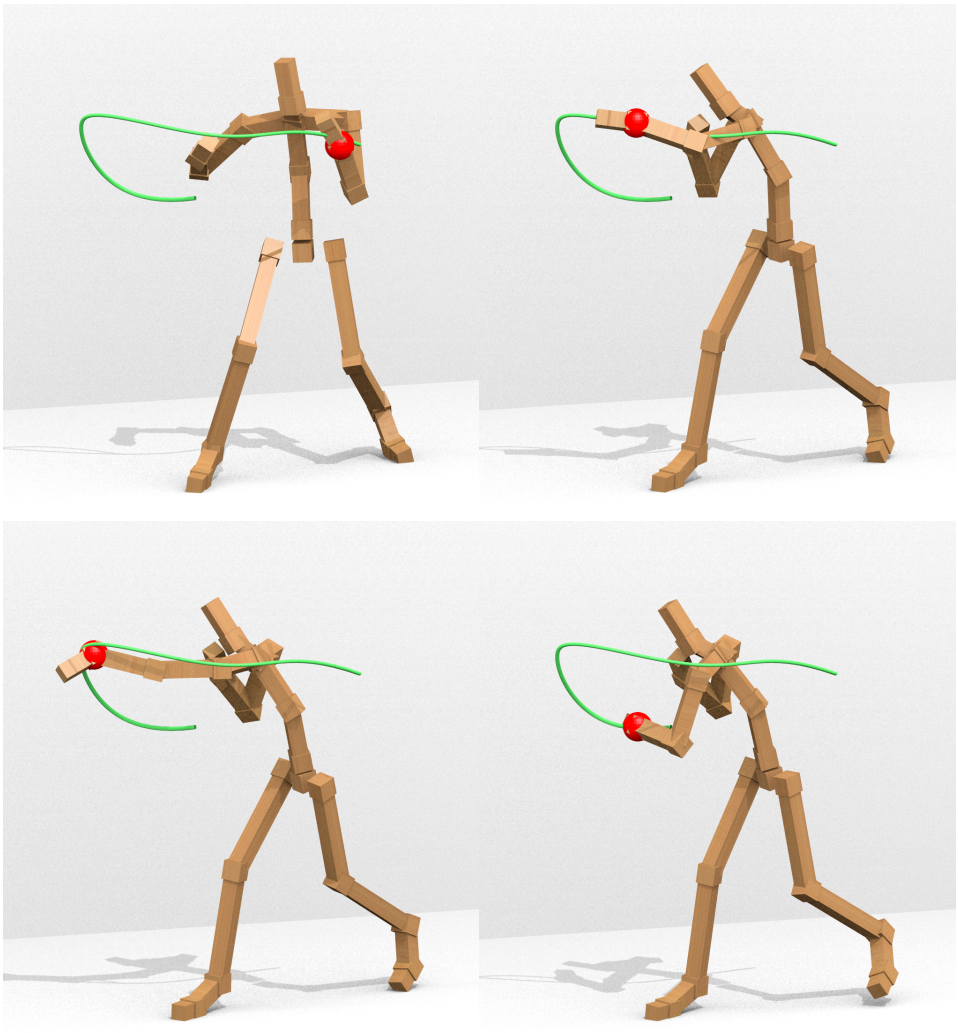


Figure 5.6: The trajectory is used for generating or modifying sequences.

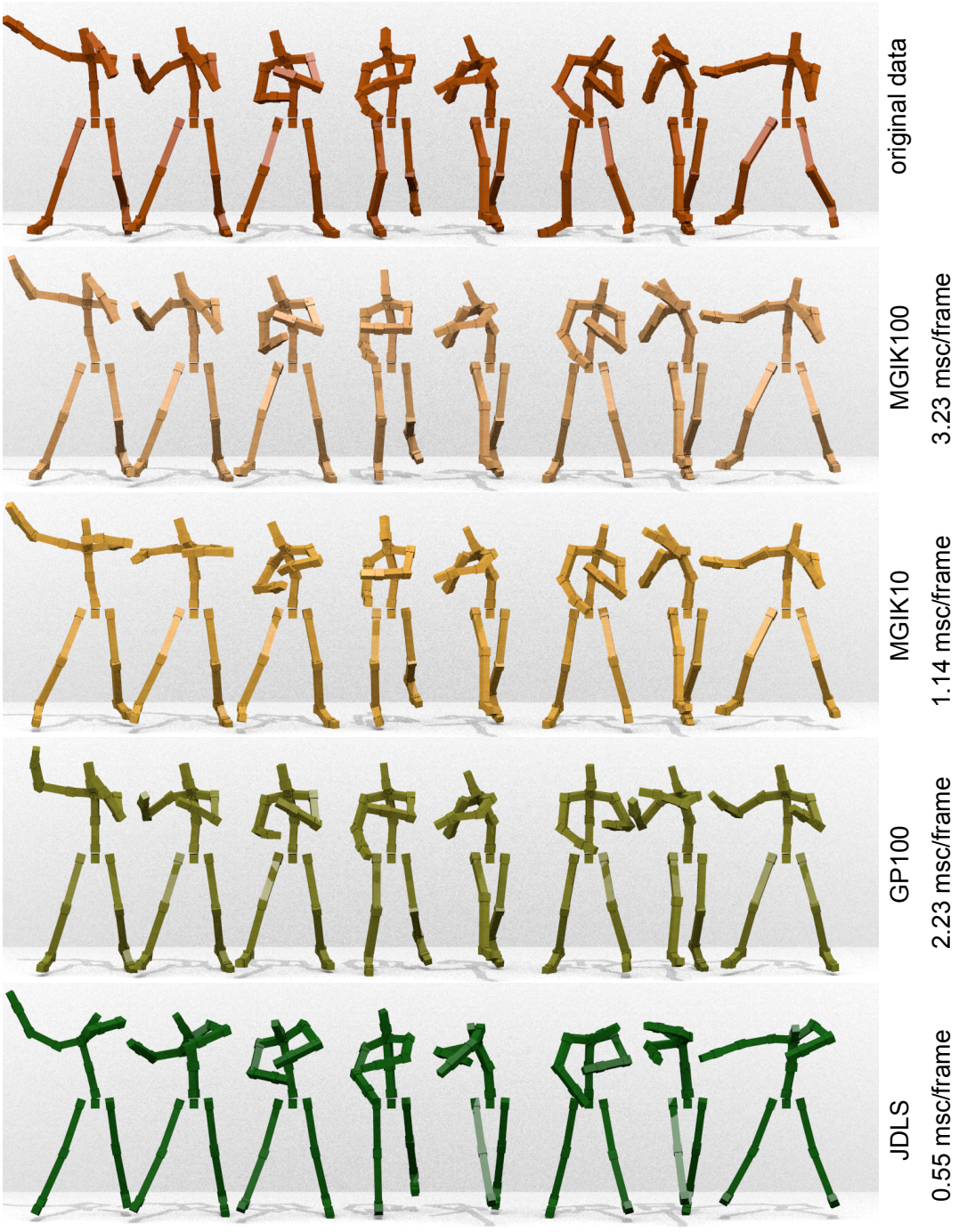


Figure 5.7: This figure illustrates the generated postures by different solutions according to the given targets in a boxing example. From top to bottom: original data, MGIK with 100 clusters, MGIK with 10 clusters, Gaussian process (GP) (100 samplers) [40], JDLS (without manually setup constraints) [14].

Part III

Motion synthesis with Neural Networks and Adversarial Learning

Introduction

The three next chapters present our work on adversarial learning for designing motion synthesis systems. These works have been realized in the second part of the thesis. Actually our preliminary studies had shown we were able to gain interesting results with non neural architectures but HMMs as well as GP dit not appear to us as easy to design models for solving complex motion synthesis tasks that we wanted to handle. Besides, recurrent neural networks were becoming more and more powerful and together with the discovery of adversarial learning for generative models in 2014 this appeared as a unique combination for building more elaborated synthesis systems.

We investigate how adversarial learning may be used for various animation tasks related to human motion synthesis. We propose a learning framework that we decline for building various models corresponding to various needs: A random synthesis generator that randomly produces realistic motion capture trajectories; conditional variants that allow controlling the synthesis by providing high level features that the animation should match; A style transfer model that allows transforming an existing animation in the style of another one. Our work is built on the adversarial learning strategy that has been proposed in the machine learning field very recently (2014) for learning accurate generative models on complex data, and that has been shown to provide impressive results, mainly on image data.

How to learn patterns from motion capture data and learn models that automatically generate realistic motion capture data has then become a very relevant research topic, which is related to learning generative models in the machine learning field.

Despite learning an accurate and realistic generator of motion capture trajectories is still a difficult and open problem, getting such a pure random generator of human motion trajectories is useless in practice. A more relevant question is how to design a model which generates animation while being controllable by the designer. The control might stand for some additional inputs to the generator, with a high level interpretation so

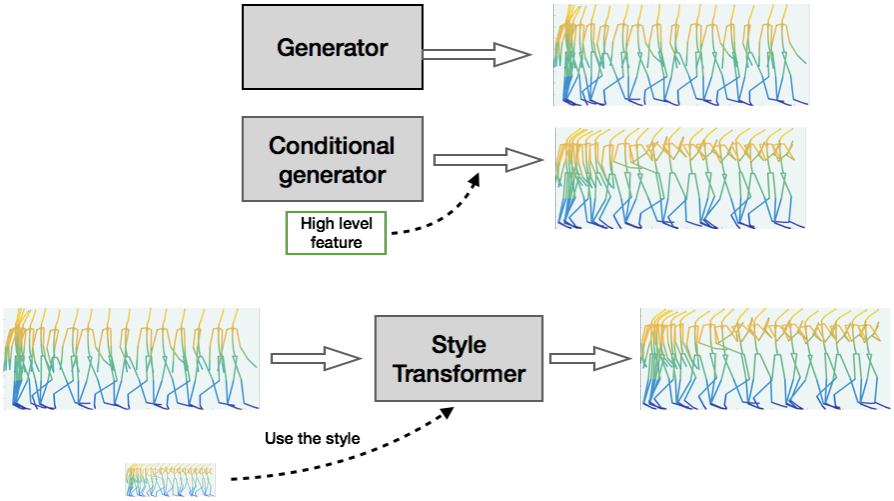


Figure 5.8: Illustration of models and tasks investigated in the paper. A pure generative model outputs data randomly according to a particular probability distribution but is out of designer's control (top). A conditional model may randomly produce animations that match particular settings that are manually set by the designer, e.g. emotion (middle). A style transfer model allows to change the style of an existing animation into the style of another animation (bottom).

that these may be set by the designer (e.g. enabling a designer to ask for an animation of a man walking with pride). One very particular way to control the synthesis is to provide example of what we want, where the generator should be able to generate a sequence *in the style of* another one. Models for transfering styles have been studied by a number of researchers in various fields (e.g. images etc), but style transferring is still a difficult and open problem which requires ad-hoc solutions and that has not been much studied for animation data.

We focus on adversarial learning architectures for animation data and explore their potential for dealing with the three animation tasks we discussed previously including motion synthesis, conditional motion synthesis and style transferring. Adversarial learning been popularized in very recent years in the machine learning community with the proposition of Generative Adversarial Networks [36]. It was originally proposed for learning accurate generative models for complex data such as images of natural scenes, and has been extended since for dealing with high resolution images, for learning conditional generative and style transferring models etc [61, 58, 17]. Despite the success of Generative Adversarial Nets (GANs) for images and videos only few researchers started to play with these models for animation and motion capture data [84].

We argue here that the three tasks discussed above (see Figure 5.8), learning a purely random generator, a conditional generator, and a style transferring model, may be tack-

led efficiently with flexible statistical models like neural networks learned within the framework of adversarial learning.

The three next chapters describe the works we did in this line of research. The first chapter, chapter 6 introduces the basic architecture that we relied on which is the combination of auto-encoder for sequences and of adversarial learning. The second chapter, chapter 7, details a number of systems that we built from this basis architecture to design few synthesis systems corresponding to different needs. The third chapter provides experimental results.

The main contributions of this part are as follows:

- We propose a generic modeling framework for various tasks related to the synthesis of motion capture trajectories.
- In particular we propose a generative model for synthesizing realistic motion capture sequences, which builds on sequence autoencoders and on adversarial learning.
- We build on our generative model to propose conditional variants enabling control on the synthesized animation by learning to generate motion sequences which fit some given monitoring parameters.
- We propose a model for transferring style that builds on disentangling ideas in order to learn to transfer the style of a motion sequence to another one.

Part of the work presented here has been published in [84] and [86].

Chapter 6

Generative model

We first present a basis model that we will build upon for designing synthesis models for various needs. The basis model is a pure generative model that allows to generate motion sequences that are similar to the ones in the training set. This model aims at generating realistic motion capture trajectories but does not offer any control to the designer on what motion is synthesized. To achieve realistic synthesized motion capture trajectories we rely on sequence autoencoders and apply the adversarial learning strategy to these models.

6.1 Introduction

We describe in the following a generative architecture for sequences, called Adversarial Sequence Auto-Encoder (ASAE), which actually is a sequence to sequence auto-encoder learned with an adversarial learning strategy. A sequence autoencoder is a recurrent neural architecture which is composed of an encoder and of a decoder, both are recurrent neural networks. The encoder iteratively builds a hidden representation of an input sequence from which the decoder reconstructs the input sequence. Sequence autoencoders leverage the well known ability of a recurrent model to capture in its hidden layer's activations (a low dimensional vector of real values) a relevant representation of a full sequence.

The sequence auto-encoder architecture makes the model learn to decode the full sequence from its encoding while the adversarial learning makes the model learn to encode motion sequences into a low dimensional representation space with a chosen prior, e.g. a Gaussian distribution. After learning, starting from a noise vector sampled from the

prior distribution, one can process it with the decoder yielding a a new generated motion sequence.

In the following, we consider a training dataset of N unlabeled sequences $\mathbf{X} = \{\mathbf{x}^i, i = 1...N\}$ where every sequence $\mathbf{x}^i$ is a sequence of frames $\mathbf{x}^i = (x_1^i, ..., x_T^i)$ of any length T, and frames are p-dimensional real feature vectors, $\forall t, x_t^i \in \mathbb{R}^p$.

6.2 AutoEncoder for Sequences

To start with we consider models able to map sequences to sequences, i.e. Sequence AutoEncoders (SAE), which have already been investigated e.g. in the Natural Language Processing field as a particular instance of Sequence to Sequence models (S2S). S2S have been proposed for machine translation tasks [74] and are well adapted when there are complex and eventually long term dependencies or forward references in the sequence data. A sequence Auto Encoder (SAE) is a S2S where the desired output sequence is the input sequence itself (see Fig 6.1 and Fig 6.2). A SAE aims at encoding the full sequence as a representation vector in the hidden space then to reconstruct the full sequence from this representation. It is then a model that computes low dimensional representation of full sequences.

The processing of an input sequence $\mathbf{x}$ is described in the algorithm 3 where f_e and f_d are implemented with (eventually stacked layers of) recurrent or Long Short Term Memory (LSTM) cells, and g is implemented with (eventually stacked) fully connected layers. h_0^e stands for an initial state of recurrent units in the encoder (we use $h_0^e = 0$) and h_t^e stands for the representation of the input sequence up to time t as computed by the encoder. The encoder's hidden state is iteratively updated as the encoder process successive elements of the input sequence. The last hidden state h_T^e is then input to the decoder, which is again a recurrent neural network, that iteratively computes a hidden state h_t^d form which frames $\hat{x}_t$ are reconstructed.

The idea of SAE is that the last hidden state computed by the encoder h_T^e should be an encoding of the full input sequence $\mathbf{x}$ to enable fully reconstructing it from h_T^e. Hence h_T^e is called the encoding of the input sequence and noted $\mathbf{e}(\mathbf{x})$. Ideally this encoding, shortened as e below, includes all the information about the input sequence (e.g. walking activity with a particular speed) and the time varying hidden state h_t^d includes a more short term information about where one is in this motion (e.g. which leg to put forward) so that h_{t+1}^d may be computed from h_t^d and e. Finally x_t may be reconstructed from h_t^d.

Algorithm 3 Processing a sequence $\mathbf{x}$ with a sequence autoencoder model

```
/* Encoding step */
```
$h_0^e = 0$
for $t = 1, ..., T$ **do**
    $h_t^e = f_e(x_t, h_{t-1}^e)$
end for
```
/* Decoding step */
```
$e = h_T^e$
```
/* e is the encoding of x */
```
$h_0^d = 0$
for $t = 1, ..., T$ **do**
    $h_t^d = f_d(h_{t-1}^d, e)$
    $\hat{x}_t = g(h_t^d)$
end for
`Output :` $\hat{\mathbf{x}} = (\hat{x}_1, ..., \hat{x}_T)$

The encoder model in this SAE is the part of the model that computes the encoding, e, of a sequence $\mathbf{x}$, and the decoder model is the part of the model that computes the predicted sequence $\hat{\mathbf{x}} = (\hat{x}_1, ..., \hat{x}_T)$ from the encoding e of a sequence. Learning of a sequence autoencoder is done by minimizing the average reconstruction cost on the training set $\mathbf{E}_{x \sim p_{data}} [\delta(\mathbf{x}, \mathbf{d}(\mathbf{e}(\mathbf{x})))]$ where δ is a distance between a sequence $\mathbf{x}$ and its reconstruction by the autoencoder $\hat{\mathbf{x}} = \mathbf{d}(\mathbf{e}(\mathbf{x}))$ (e.g. the sum of euclidean distance between frames and their corresponding reconstruction).

6.3 Adversarial Autoencoder for Sequences

A RNNs' key feature that we exploit here lies in their capacity to transform a variable length sequence into a fixed sized vector [21]. As an input sequence is processed in a SAE a representation of the sequence, h_t, is iteratively built in the hidden layer of the encoder, yielding a final latent representation of the whole input sequence, h, from which the decoder RNN reconstructs the input sequence. We leverage this feature to build Adversarial Sequence AutoEncoders (ASAE).

We define an *Adversarial Sequence AutoEncoder* (ASAE) as the association of a SAE and of a discriminator D enforcing the encodings h of training sequences to follow a prior distribution (e.g. a Gaussian distribution p_n). The model is illustrated in Figure 6.1 and Figure 6.2.

The learning of a ASAE resumes to optimizing the following criterion with respect to the autoencoder parameters (summarized here as $\mathbf{e}$ and $\mathbf{d}$) and to the parameters of the

discriminator D, where all these models are implemented as neural networks:

$$\begin{aligned} \min_{\mathbf{e},\mathbf{d}} \max_{\mathrm{D}} \quad & \mathbb{E}_{\mathbf{x}\sim p_{data}} \left[\delta(\mathbf{x}, \mathbf{d}(\mathbf{e}(\mathbf{x})))\right] \\ & + \mathbb{E}_{\mathbf{x}\sim p_{data}} \left[log\left(\mathrm{D}(\mathbf{e}(x))\right)\right] \\ & + \mathbb{E}_{z\sim p_n} \left[log(1-\mathrm{D}(z))\right] \end{aligned} \tag{6.1}$$

where the discriminator D takes as input a sample $z \in \mathbb{R}^p$ in the encoding latent space and outputs a decision whether z comes from the prior distribution p_{noise} or if it is the encoding h_T of a true sequence data from X. The discriminator D is learned to distinguish between random vectors z following a prior distribution (a Gaussian distribution in our case) and the encodings h_T of full sequences from the training set X while the encoder is learned to project training sequences in the encoding space so as to fool the discriminator.

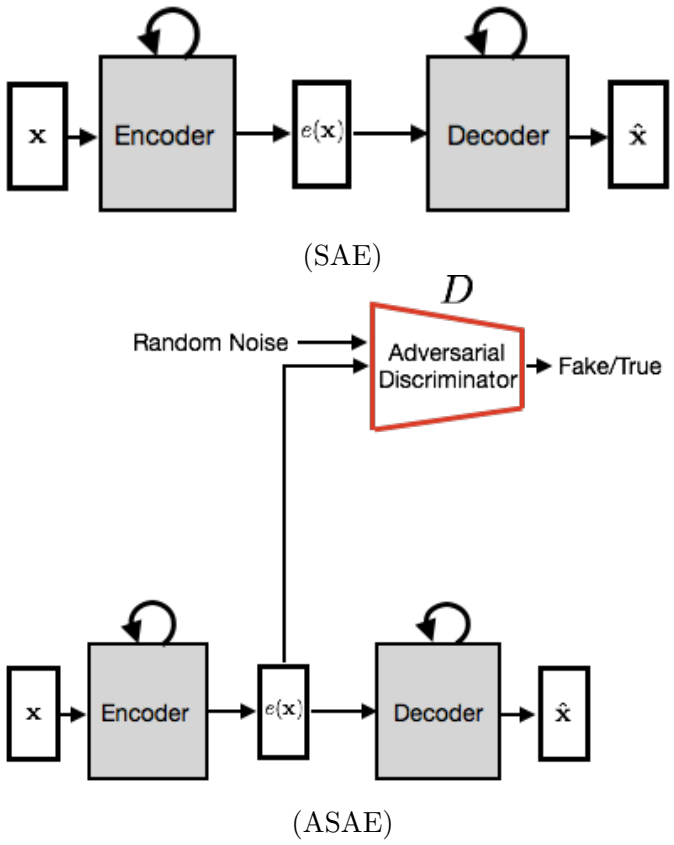
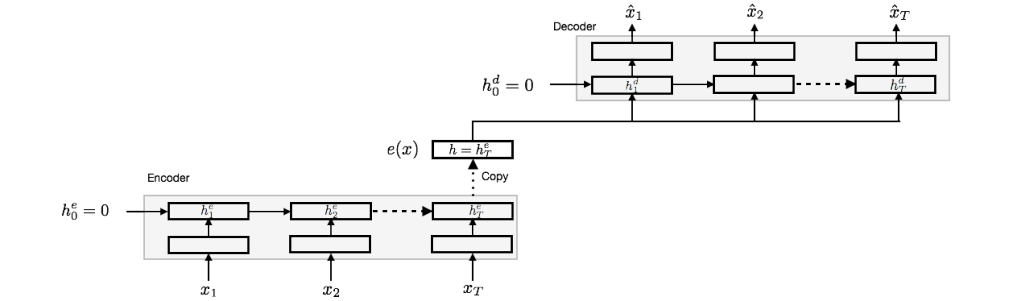
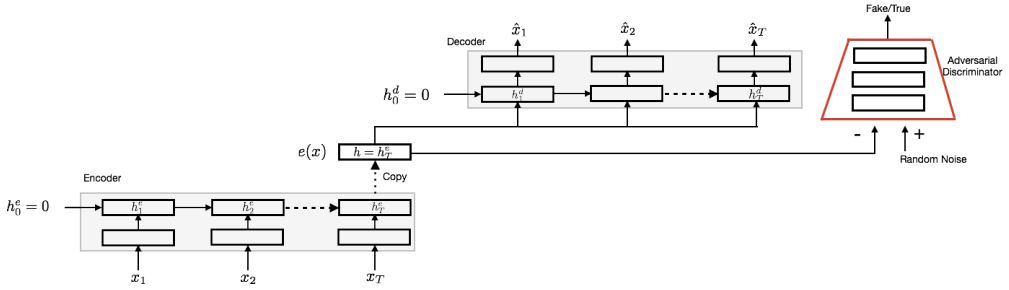


Figure 6.1: Sequence Autoencoder (SAE) (top) and Adversarial Sequence AutoEncoder (ASAE) (bottom). The encoder and the decoder are implemented with RNNs. In a ASAE an additional discriminator is used to constrain the distribution of encodings of training data to fit a given noise distribution. After training the decoder may be used as a random generator where randomization is provided by the noise distribution.



(SAE)



(ASAE)

Figure 6.2: Sequence Autoencoder (top) and a Adversarial Sequence AutoEncoder (bottom) where models are unfolded in time. See Algorithm 3 for explanations on the processing of a sequence in these models, and Algorithm 4 for explanations on how to generate a sequence with a learned ASAE.

6.4 Motion synthesis with a ASAE

Once an ASAE has been learned, one may remove the encoder part and use the decoder part as a synthesis model by sampling a latent vector $z \sim p_{noise}$ then by decoding this latent vector using the decoder as explained above, yielding $\mathbf{d}(z)$, see Algorithm 4.

Algorithm 4 Randomly generating a sequence $\hat{\mathbf{x}}$ with the Decoder of a trained ASAE

`Input: Length of the generated sequence =` T

`Sample` z `from` p_{noise}
$h_0^d = 0$
for $t = 1, ..., T$ **do**
  $h_t^d = f_d(h_{t-1}^d, z)$
  $\hat{x}_t = g(h_t^d)$
end for
`Output :` $\hat{\mathbf{x}} = (\hat{x}_1, ..., \hat{x}_T)$

To go a little further in motion synthesis one may use the above models for generating a variety of motion sequences by inputing a time varying trajectory for h rather than a fixed vector drawn from noise distribution p_{noise}, as in Algorithm 5.

Many approaches may be used for generating meaningful trajectories $(h_1, h_2, ..., h_T)$. For instance, assume that two motion encodings h^1 and h^2 correspond to encodings of a slow and of a fast walk motion activity. Then one may compute a linear interpolation interpolating from $h_1 = h^1$ to $h_T = h^2$ enabling synthesizing a walk motion activity of increasing speed, while there were eventually no such motion in the training set.

Algorithm 5 Generating a sequence $\hat{\mathbf{x}}$ from an encoding series $\mathbf{e}$ with the decoder of a trained ASAE

INPUT $\mathbf{e} = (e_1, e_2, ..., e_T)$

$h_0^d = 0$
for $t = 1, ..., T$ **do**
  $h_t^d = f_d(h_{t-1}^d, e_t)$
  $\hat{x}_t = g(h_t^d)$
end for
`Output :` $\hat{\mathbf{x}} = (\hat{x}_1, ..., \hat{x}_T)$

6.5 Implementation and variants

We implemented variants of the above modeling by learning autoencoders to not only optimize a reconstruction loss of a sequence as in traditional SAE but also to optimize a reconstruction of the first order derivatives of frames (alike well known Δ features use in speech recognition tools). This enforces the model to output a sequence whose dynamics is closer to the one in the input sequence, yielding more realistic animation. We add the suffix "-Δ" (for Δ loss) to models' names to indicate this additional loss, e.g. ASAE-Δ is an Adverarial SAE model which is learned with the additional Δ loss. Also, rather than training our model on full sequences we rather use a windowing process to work on shorter sequences (1 or 2 seconds lengthed). This allow overcoming the difficulty of dealing with very long sequences while at the same time it focuses the learning of the models on shorter term dynamics that we want to capture. Note that this strategy does not prevent from generating long sequences. Actually the length of a generated sequence is an input to the generation process (Cf. Algorithm 4). Moreover learning such short sequences allows capturing the dynamics of most motion and our trained generators do generate long sequences as realistic as short ones.

6.6 Conclusion

The models that we presented in this chapter are pure generative models able to produce realistic motion animation randomly. Although this is an important brick and a difficult problem, such a pure random generator is often useless in practice. On the contrary in the animation field, one may hope not only to generate realistic motion sequences but also to be able to control some key features on the animation.

From this point of view the work in this chapter is more a framework within which we will design more elaborate models in the next chapter, by introducing constraints, with adversarial learning, on what is modeled and taken into account by both the encoder and the decoder.

Chapter 7

Conditional Models

In the animation area, one needs to capture multiple variant styles of motions for controlling each animation character, and a key feature is the high level control a designer may have on synthesized animation. We present here few extensions of the generative model in previous chapter towards this direction.

We first present two conditional variants of the generative model that we design to exploit an additional input information (e.g. the emotion with which an activity is performed) that is used to accordingly change the generation process. These models provide the designer more control on generated sequences.

Moreover, we investigate the design of style transferring models by building on a recent topic known as disentangling factors of variations [58, 17] that has mainly been studied for images data, alike most of the adversarial learning literature. This style transferring architecture allows in particular to synthesize a new sequence which is a transformation of an input sequence *in the style* of another input sequence.

7.1 Conditional synthesis models

In the animation field, one hopes not only to generate realistic motion sequences but also to be able to control some key features on the animation. Here, we consider a supervised case where we assume that in addition to the usual supervision information (e.g. activity label), some contextual information is available, such as the emotion with which a motion is performed, the gender of the actor, her age, her weight. All these factors do influence the motion. Taking these into account would allow capturing finer the motion dynamics, enabling generating nicer and more realistic motion.

Conditional generative models provide another way to control the generating process. These usually use discrete labels or other contextual information as auxiliary inputs to condition the synthesis process of a generative models. Following the success of GANs in images, some conditional models have been proposed [61, 18]. Mirza et al [61] proposed a simple conditional GAN which takes the discrete label as input to both generator and discriminator. Chen et al [18] add mutual information constraint on the objective function of GAN and can learn conditioned generative models for images. These models are all applied in image and based on convolutional neural networks which are not designed for sequential data like motion capture data.

The idea is to build contextual generative models that allow to synthesize motion that fit some specific high level settings, e.g. enabling a designer to ask for the synthesis of a walking motion from a happy old man, where the activity *walking*, the emotion *happy*, and the actor's physiology *old man* would be given by the designer. We do not consider the use of multiple side information here and only consider a single one, but the extension to multiple side information is straightforward provided an appropriately labeled dataset is available at training time.

In this section we consider that some additional information is available for sequences in $\mathbf{X}$. The training set is then a collection of pairs $(\mathbf{x}^i, \mathbf{s}^i)$ with $\mathbf{x}^i \in \mathbf{X}$, and $\mathbf{s}^i \in \mathbf{S}$ where $\mathbf{s}^i$ denotes the additional information relative to training sequence $\mathbf{x}^i$. It may stand for any additional information, such as the emotion with which motion in $\mathbf{x}$ is performed, the gender of the actor, her age, her weight... We propose two conditional extensions of ASAE for exploiting side information.

A Conditional Adversarial Sequence Autoencoder (CASAE), is an extension of pure generative model ASAE, it is built as the concatenation of a conditional encoder and of a conditional decoder that both take as additional input a style information. Alternatively a ASFSAE first computes a *style-free* sequence representation of a sequence from which a sequence is reconstructed using a conditional decoder. We first discuss how one can use conditional information to alter the prediction of a RNN model to build contextual encoders and decoders. Then we present our two models.

7.1.1 Conditional RNNs

Designing a RNN which takes into account a side information such as a contextual information is actually straightforward. Assuming the contextual information remains fixed all along the sequence a contextual RNN is designed by just using the contextual

information as an extra input every time step. Actually this may simply be implemented by concatenating the contextual observation to each observation in the input sequence and to process the sequence of such augmented observations as usual in standard RNNs. Note that when the contextual information is discrete (as it is in our experiments), one may use a one-hot encoding of it or an embedding. Figure 7.1 illustrates a contextual RNN that exploits a contextual information s, and where the final part of the model is not shown. It might be a decoder as in the models we will present in the remaining of this chapter, or it might be a classification layer of one is interested in sequence classification. The figure shows the folded and unfolded views of a contextual RNN.

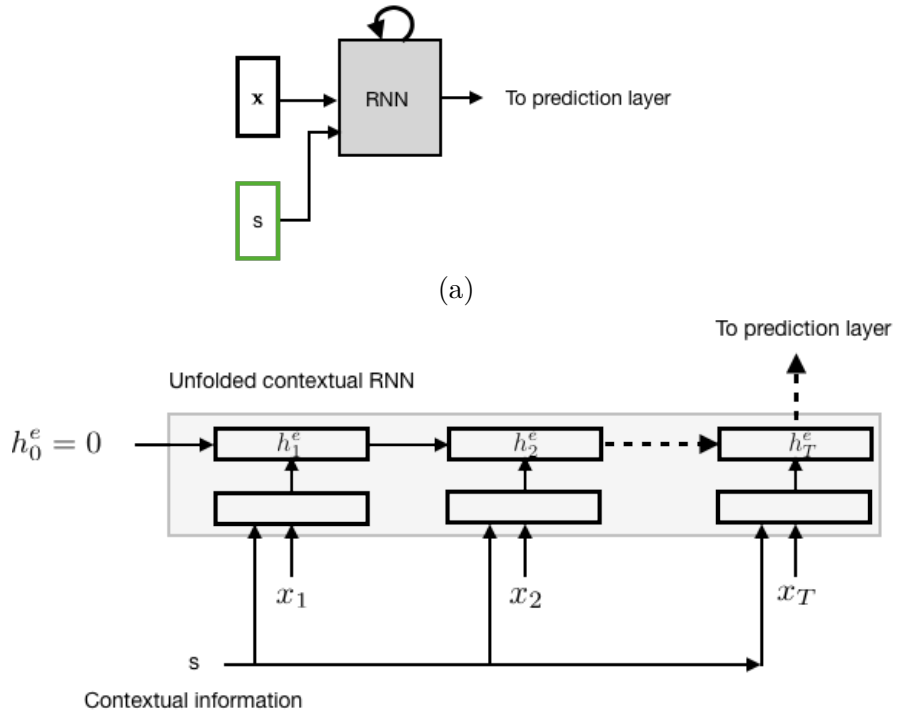


Figure 7.1: Contextual versions of a RNN model (a) and its unfolded representation (b). See text for explanations.

7.1.2 Conditional Adversarial Sequence Auto-Encoders (CASAE)

A number of works have focused on designing markovian models able to deal with contextual information for motion synthesis [69, 27, 85] and more generally on images in the context of adversarial learning of neural networks [61, 24]. A simple yet effective idea to deal with such a setting consists in putting this contextual information as an additional input [69, 27], which is the strategy we explored in our particular context. We first investigated a conditional variant of the pure generative model where the side in-

formation (activity, emotion, or whatever additional information) is fed as an additional input at every step in the encoder and in the decoder. Such a model is learned from an accordingly labeled dataset. It is illustrated in Figure 7.2 (top). When considering a discrete side information (as we do in our experiments) with K possible values, we use a one-hot-code encoding of this contextual variable, i.e. a binary vector of size K which is zero everywhere but at the position of the variable value (note that embedding layers could also be used here but this do not bring any improvements). Processing an input sequence with a CASAE is detailed in Algorithm 6.

Algorithm 6 Processing a sequence $\mathbf{x}$ with a conditional sequence autoencoder model

`Inputs: Sequence x and its side information` s

`/* Encoding step */`
$h_0^e = 0$
for $t = 1, ..., T$ **do**
  $h_t^e = f_e(x_t, h_{t-1}^e, s)$
end for

`/* Decoding step */`
$e = h_T^e$
$h_0^d = 0$
for $t = 1, ..., T$ **do**
  $h_t^d = f_d(h_{t-1}^d, e, s)$
  $\hat{x}_t = g(h_t^d)$
end for

`Output :` $\hat{\mathbf{x}} = (\hat{x}_1, ..., \hat{x}_T)$

This conditional model is learned by optimizing the following loss:

$$\begin{aligned}\min_{\mathbf{e},\mathbf{d}} \max_{D} \quad & \mathbf{E}_{x\sim p_d}\left[\delta(\mathbf{x}, \mathbf{d}(\mathbf{e}(\mathbf{x}, s), s))\right] \\ & + \mathbf{E}_{x\sim p_x}\left[log\left(1 - D(\mathbf{e}(x, s))\right)\right] \\ & + \mathbf{E}_{z\sim p_n}\left[logD(z)\right]\end{aligned}$$

where s denotes the side information for input $\mathbf{x}$, it is used as additional input to $\mathbf{e}$ and $\mathbf{d}$, δ is a distance on sequences. Once such a model is trained it may be used for generating sequences with a given side information using the Algorithm 7.

Algorithm 7 Generating a sequence $\mathbf{x}$ with side information $\mathbf{s}$

`Input:` `Side information` $\mathbf{s}$

`Sample` z with p_{noise}
$h_0^d = 0$
for $t = 1, ..., T$ **do**
  $h_t^d = f_d(h_{t-1}^d, z, \mathbf{s})$
  $\hat{x}_t = g(h_t^d)$
end for
`Output :` $\hat{\mathbf{x}} = (\hat{x}_1, ..., \hat{x}_T)$

7.1.3 Conditional synthesis from style-free encodings of motion sequences: Adversarial Style Free Sequence Auto-Encoders (ASF-SAE)

We study now an alternative view of the problem and by first computing a *style-free* encoding of the input sequence, from which a stylized version of it may be synthesized using a conditional decoder as the one used in previous model. In other words, if the side information that we consider is the emotion, we aim at learning an encoding of a motion capture sequence which includes the core information about the motion but not the emotion under which it was performed. From this encoding, using a conditional decoder RNNs, a new sequence may be generated corresponding to the input motion but performed under another emotion.

The architecture of the model is shown in Figure 7.2 (bottom). It includes two main components. It is an Adversarial Sequence Autoencoder (ASAE) as described above where the encoder is similar to what we used in section 6.3 while the decoder is taken from the model described in section 7.1. The latter takes as additional input, every time step, a one hot encoding of the input sequence's style/emotion label. The second main component of the model is a style (e.g. emotion) discriminator, D_s, which is learned to recover the style information of the input sequence from its encoding representation. Following the idea in [34], the generator is learned so as to fool this discriminator by back-propagating the reverse of the style discriminator gradient in the encoder.

The rationale behind this architecture is to ensure that after learning, the emotion information is removed in the input sequence's encoding as much as possible so that one may generate a new version of the input sequence with a new, manually chosen, style (emotion). The model is learned by optimizing the following loss:

$$\min_{\mathbf{e},\mathbf{d}} \max_{D_a,D_s} \quad \mathbf{E}_{x,s\sim p_d}\left[\delta\left(\mathbf{x},\mathbf{d}\left(\mathbf{e}(\mathbf{x}),s\right)\right)\right] + \mathbf{E}_{x,s\sim p_d}\left[log\left(1-D_a(\mathbf{e}(x))\right)\right] + \mathbf{E}_{z\sim p_n}\left[logD_a(z)\right] - \mathbf{E}_{x,s\sim p_d}\left[H_s\left(D_s(\mathbf{e}(x))\right)\right]$$

where δ is a distance between sequences, p_d stands for the empirical distribution of data (pairs of a sequence and of its style label, (x, s)), and H_s stands for the cross entropy criterion for the style discriminator D_s (it is a classical classification loss).

7.2 Motion edition through disentangling factors of variation: Disentangling Adversarial Sequence Auto-Encoder (DASAE)

Previous section has focused on learning pure and conditional generative models. We focus now on motion editing, which aims at modifying an input motion capture sequence. Motion edition has been studied in the last decade by researchers in animation and graphics fields with dedicated models [92, 39, 94]. Besides, building on the adversarial learning idea, a number of models have recently been proposed for image edition [18, 67] and more generally for the disentanglement of content and style in images [58, 16, 17] and videos [25].

We specifically focus here on style transferring where one wants to generate a new sequence from an existing one by transforming it in the style of another sequence. This calls for a mechanism for computing the style of a sequence. Following the same line of research in previous sections works we build on ASAE for designing a system able to transfer style from a sequence to another one. We consider the same setting as in previous section. The training set consists of a number of sequences that are performed under a particular style (e.g. emotion), where the number of styles is finite and the style information is available at training time as a categorical label information (but dealing with continuous variables would be straightforward).

Some motion editing techniques have been proposed in the animation field, such as inverse kinematics [39], style transferring [54, 92, 48, 94]. Inverse kinematics aims at finding the postures which satisfy given constraints, such as the trajectory of few joint

position. Style transferring aims at extracting a style from one motion or at transferring this style to another motion.

Style transferring has a growing interest in the machine learning field especially for images with the spread of adversarial learning strategy [35, 52]. For instance the paper [35] leverages convolutional neural networks to learn the image features and uses a gram matrix to separate style and content features of input image.

Following previous work in section 7.1.3 we propose to design an editing model by considering continuous style information that are inferred from an input sequence and by learning to synthesize with such an input. The architecture of the model is illustrated in Figure 7.3. It is again composed of a sequence auto-encoder and of multiple discriminators.

Importantly, when given an input sequence $\mathbf{x}$, the encoder produces two encodings (i.e. low dimensional vectors), one for the core information of the input sequence, $\mathbf{e}_c(\mathbf{x})$, called content, (e.g. the activity which is performed) while the other one encodes the style information, $\mathbf{e}_s(\mathbf{x})$ (e.g. the emotion). Both encodings are input to the decoder. As in ASAE a discriminator D_a enforces the content encoding to obey a given prior distribution. Moreover two style discriminators are added to the model to enforce the content encoding not to include information about the side information while the style encoding should. The first discriminator D_s takes as input the content encoding and it is used in the same way as above. Its parameters are learned to recover the style information from the content encoding but the reverse of its gradient is back-propagated in the encoder to make content encoding free from this style information. The second style discriminator D_c takes as input the style encoding. It is learned to recover the style label and its gradient is back-propagated in the encoder as is, so that the encoder should learn to actually include the style information in the style encoding. All together the learning is cast as the following optimization problem:

$$\begin{aligned}\min_{\mathbf{e},\mathbf{d},D_c}\max_{D_a,D_s} \quad & \mathbf{E}_{x,s\sim p_d}\left[\delta\left(\mathbf{x},\mathbf{d}\left(\mathbf{e}(\mathbf{x})\right)\right)\right] \\ & +\mathbf{E}_{x,s\sim p_d}\left[log\left(1-D_a(\mathbf{e}_c(x))\right)\right] \\ & +\mathbf{E}_{z\sim p_n}\left[logD_a(z)\right] \\ & -\mathbf{E}_{x,s\sim p_d}\left[H_s\left(D_s(\mathbf{e}_s(x))\right)\right] \\ & +\mathbf{E}_{x,s\sim p_d}\left[H_s\left(D_c(\mathbf{e}_c(x))\right)\right]\end{aligned}$$

where $\mathbf{e}(\mathbf{x}) = [\mathbf{e}_c(\mathbf{x}), \mathbf{e}_s(\mathbf{x})]$ stands for the encoding of an input sequence into a pair of

a content encoding (the vector $\mathbf{e}_c(\mathbf{x})$) and of a style encoding (the vector $\mathbf{e}_s(\mathbf{x})$). The models D_a and D_s are two adversarial discriminators. D_a tries to distinguish the output of encoder from the prior noise. D_s aims to predict the style label from the content encoding $\mathbf{e}_c(\mathbf{x})$. The generator is learned to flue both adversarial discriminators. D_a is learned by minimising the cross entropy metric and D_s is learned by minimising the categorical cross-entropy. Finally D_c is learned to recognize the style label from the style encoding $\mathbf{e}_s(x)$ and the encoder is learned with the same goal (i.e. the gradient is not reversed when back propagating in the encoder).

Once such a model has been learned one may transfer the style of a sequence x_2 to another sequence x_1 as follows (Figure 7.4). Both sequences are processed by the encoder, yielding a pair of latent codes for each of the two sequences x_i, (c_i, s_i). Then the decoder is used to process the pair of latent codes (c_1, s_2) composed of the content latent code of sequence x_1 and the style latent code of sequence x_2.

Note that, while in previous section the side information used at inference time was discrete, which we assume limits the variability of the synthesized sequences with respect to this additional input, we are able to deal here with continuous style information that are inferred from input sequences. We believe this brings a much better behavior with respect to the variability of the generated sequences.

7.3 Conclusion

Within the framework that we introduced in previous chapter we were able to design various models that fit different animation tasks, giving control on the animation synthesis.

Indeed, beyond the pure generative models of previous chapter the conditional models we presented here allow synthesizing motion under a specific context where the context may be chosen by hand at synthesis time.

Finally we were able to design a motion sequence editing model that allows transforming an input motion sequence to match the style of a second sequence.

As has been shown many extension may be easily proposed to the pure generative one to match particular settings, we explored few others that we do not discuss here, we preferred focusing on architectures that actually yield improvements in the experiments.

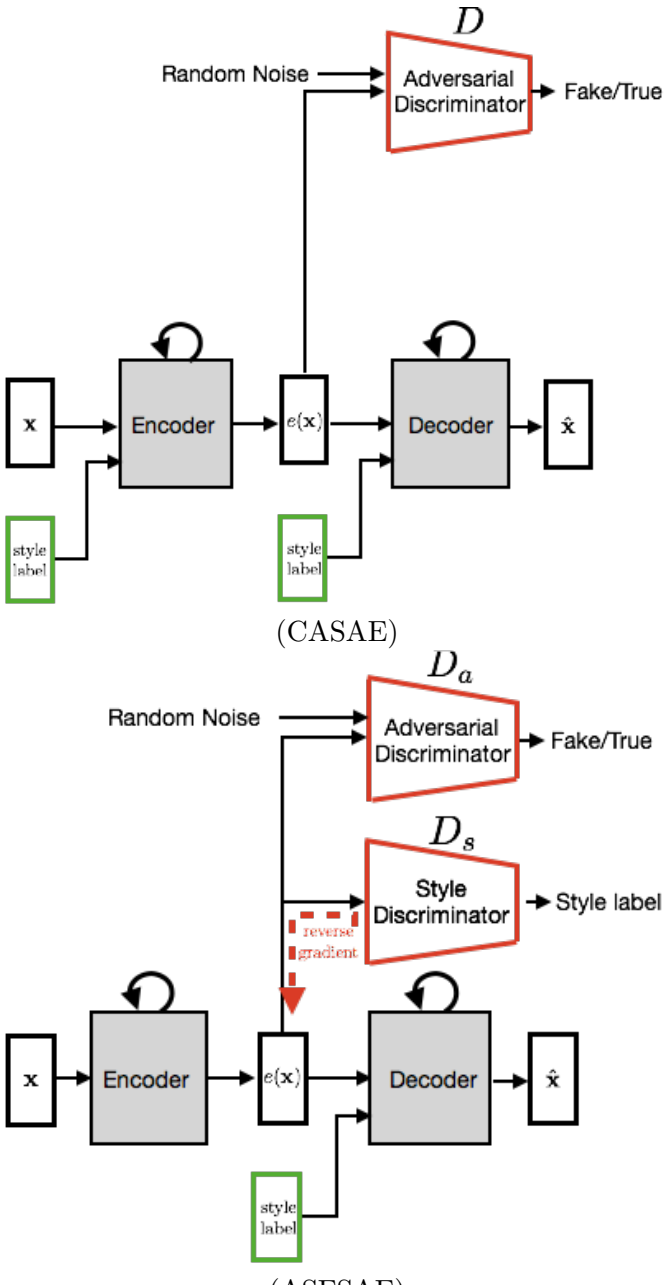


Figure 7.2: Conditional generative models. CASAE are an extension of pure generative model ASAE, they are the concatenation of a conditional encoder and of a conditional decoder that take as additional input a style information (left). ASFSAE are learned with adversarial learning to compute a *style-free* sequence representation from which a sequence is reconstructed with a conditional decoder (right).

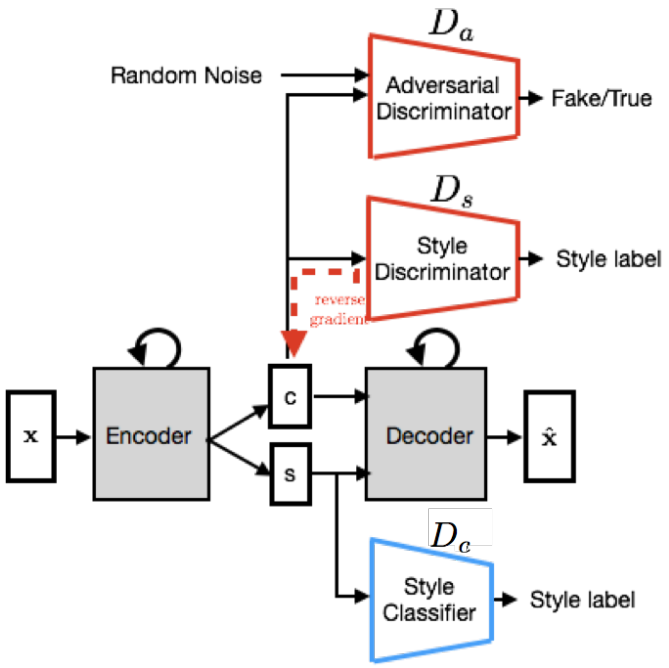


Figure 7.3: Structure of a Style Transferring Model (DASAE)

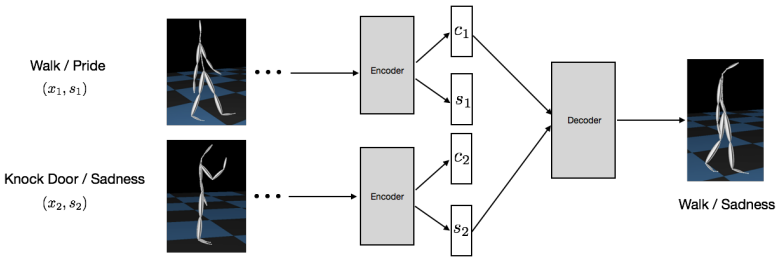


Figure 7.4: Illustration of style transferring using a DASAE style transferring model. An first input animation sequence is transformed into two latent vectors, a content code x_1 and a style code s_1. A second animation sequence is transformed analogously into a content code and a style code, x_2 and s_2. A new animation sequence is output based on the content latent code x_1 and the style latent code s_2.

Chapter 8

Experiments

We provide in this chapter a number of experimental results comparing the various models we proposed in chapter 6 and chapter 7 with state of the art models. We first detail the experimental setting, then we detail our results. Evaluating animation synthesis is not straightforward. We first provide objective results including both a number of statistics and the results of simple to evaluate tasks like forecasting or emotion classification on generated data. Then we include a number of subjective results in the shape of synthesized animations. More results may be found at https://bit.ly/2rwoGok.

8.1 Experimental setting

8.1.1 Dataset

We performed experiments with the Emilya Dataset [32]. We selected 60% of data as training set, 20% of data as validation set and 20 % as test set. We split the dataset across $(activity, emotion, actor)$ pairs to make sure the they are distributed evenly in training, validation and test set. For global translation information, we removed the horizontal position coordinates and normalized the vertical position with respect to 12 actors. We kept the all three global orientation and other joint angles and normalized them using Max-Min method to the range $(0, 1)$. For training our models, we cut the sequences in windows of size 200 (1.7 seconds approximately). At the end, we get a training set with 78 982 sequences of two hundreds 70-dimensional frames (one sequence is about 1.7s length), the validation set includes 21 614 sequences and the test set 38 637.

8.1.2 Baselines

We compare our models including Δ variants to few non adversarially learned baselines including sequence autoencoder(SAE) models [74], variational autoencoders for sequences (SVAE) [11] and ERD [33]. Additionally for forecasting experiments only we also provide comparative results to LSTM3LR [33]. Since LSTM3LR is an autoregressive model, it is not actually suitable for generating human motions. Hence we just use it as a baseline for the frame prediction task. Note that to generate sequences with non adversarially learned models (e.g. SAE), we first estimate, after training, a Gaussian distribution of the latent codes computed from training sequences. Then we use this distribution and the decoder part to define a generator as we did for adversarially learned models.

8.1.3 Implementation details

All models have been implemented with Keras [22]. We used sckit-learn [66] to gridsearch the hyperparameters of these the models, such as the learning rate, the optimization strategy, the network structure etc.

The detailed structures of ASAE, CASAE, ASFSAE and DASAE models are listed in Table 8.5 and the final hyperparameters are in Table 8.7. The structure of SAE is the same as the one of ASAE, see Table 8.5. Optimal learning rate and optimization method are listed in table 8.7.

The detailed structure and hyperparameters for VAEs are listed in Table 8.10. Baseline models LSTM-3LR and ERD are implemented as in [49], their hyperparameters and detailed structures are provided in Table 8.8

8.2 Objective evaluation

8.2.1 Likelihood estimation

We start with experiments that compare the models as generative models through the computation of the likelihood of test data. The higher this quantity the better the generative model. Of course one cannot exactly compute such a likelihood and we rely on the method proposed by [36, 26], using Gaussian Parzen Estimator, to estimate it as follows. To compute the likelihood of test data by a generative model the method consists in first using the generative model to generate a large number of data, then using

these data to fit a Gaussian Parzen estimator in order to get an estimated probability density function (pdf) on data that correspond to the generative model. Finally this pdf is used to compute the likelihood of a separate test dataset.

To exploit this idea on sequences we consider fixed length generated sequences that we reshape as vectors. We randomly select 10 000 synthesized sequences with a generative model, and use these data to learn a Parzen estimator. Then we compute the log-likelihood of a random subset of 10 000 test sequences under this estimator. Note that to get a generative model from the standard learning of a sequence autoencoder (i.e not adversarially trained) we first estimate the distribution of encodings, assuming a Gaussian distribution and generate sequences by first sampling a noise vector using this estimated noise distribution then feeding this to the decoder part of the autoencoder.

The results of these computation are reported in Table 8.1. Preliminary results on ASAE and CASAE have shown a systematic improvement with Δ variants so that we implemented only this variant of DASAE models. It may be seen strong differences between all the models. The ERD model produces the significantly lowest likelihood while SAE, SVAE and ASAE are close from each others. Conditional models CASAE, ASFSAE are significantly better than other models and are more or less equivalent. Moreover the style code dimension has a strong impact in DASAE models and 3 dimensional codes seem to give best results here. Yet the likelihood computation is not the ultimate metric we are interested in and we will report results for a style dimension equal to 3, 5 and 8 in the remaining of the paper.

8.2.2 Diversity and completeness

Next we report in Table 8.2 statistics that provide insights on the diversity of the sequences generated by a particular model, their quality, and their completeness (i.e. do generated data cover the whole variety of true sequences?). Statistics are computed as follows for one particular generative model. We generated a set of 60 000 sequences. For each generated sequence we computed its minimum distance to a true sequence from the validation and test set ($\approx$ 60 000 sequences). We report this average minimum distance as G2T criterion (Generated to True). We compute similarly T2G (True to Generated) and compute similar criterion on sequences of Δ features computed from generated and true sequences.

One more or less observes the same hierarchy betwwen models than in Table 8.1. Adversarial models almost systematically outperform non adversarially learned ones on all

Models	Likelihood
SAE	1730 ± 10.5
ERD	1369 ± 123
SVAE	1719 ± 19
ASAE	1781±11.2
ASAE-Δ	1797 ±4.6
CASAE	1802 ± 7.61
CASAE-Δ	1804 ± 7.2
ASFSAE-Δ	1805 ± 6.4
DASAE-d2-Δ	1809±11
DASAE-d3-Δ	1815±11
DASAE-d5-Δ	1808± 10
DASAE-d6-Δ	1781± 8.9
DASAE-d8-Δ	1762± 10.4
DASAE-d10-Δ	1744± 11

Table 8.1: Likelihood evaluation of test data with Gaussian Parzen estimator (following [36], see text) for various models. DASAE-*dx* stand for Disentangling models with a style encoding size of dimension equal to x. Standard deviation of estimations are provided after ±.

criterion, meaning that generated sequences are somehow more realistic (lower G2T) and that all modes of the true distribution are better covered (low T2G). Amongst adversarially learned models SVAE, ASAE, CASAE and ASFSAE perform more or less similarly, with ASFSAE outperforming other models on GE2T and T2G criterion, while CASAE, ASAE and ASFSAE are on par on "-Δ" criterion. Finally DASAE with 3 or 5 dimensional style encoding outperform significantly all other models whatever the criterion. Moreover the "-Δ" models show impressive gains on modeling the dynamics of the motion capture data, which should likely yield to more realistic generated animations.

8.2.3 Pose Forecasting

We evaluated few of our models through the task of pose forecasting, i.e. frame prediction at a given horizon ahead (e.g. 1s) based on a prefix sequence of frames. We compared our approach with state of the art baselines, the Encoder Recurrent Decoder (ERD) [33] and a regular LSTM neural network with 3 layers (LSTM_3LR), that have been proposed for such a task. ERD and LSTM_3LR are reported in [33] as state of the art in such a frame prediction. We randomly selected 10 sequences with 200 frames for each pair of activity and emotion (i.e. 64 pairs). We used a prefix of 60 frames (0.5 second) as a sequence prefix and we evaluated the abilty of models to predict the frames 120 next frames (i.e. up to horizon of 1 second). While ERD and LSTM are designed for

Models	*G2T*	*T2G*	*G2T*-Δ	*T2G*-Δ
SAE	0.6925	0.607	0.0597	0.0424
ERD	0.4152	0.6947	0.0201	0.0429
SVAE	0.5724	0.59	0.1688	0.0567
ASAE	0.5261	0.5279	0.0642	0.0441
ASAE-Δ	0.5196	0.5205	0.0175	0.0322
CASAE	0.5438	0.5227	0.0668	0.0454
CASAE-Δ	0.515	0.5058	0.0226	0.0305
ASFSAE-Δ	0.4834	0.499	0.0176	0.0322
DASAE-d3-Δ	0.4528	0.5145	0.0144	0.034
DASAE-d5-Δ	0.4715	0.5225	0.0149	0.0337
DASAE-d8-Δ	0.5	0.5619	0.0148	0.0336

Table 8.2: Statistics on distance between generated and true sequences (smaller is better). Report results are distances that have been normalized by the number of frames in a sequence. *G2T* (*T2G*) stands for the minimum distance of a generated (resp. true) sequence to the set of True (resp; Generated) sequences. ″-Δ″ criterion are computed on sequences of Δ coefficients.

such a task, ASAE is not.

To perform such a forecasting with ASAE, we used the prefix sequences with its corresponding speeds as inputs in the encoder to get a latent vector. Then we fed the latent vector with its speeds to the decoder to generate a 180 frames lengthed sequence. The first 60 frames are a reconstruction of the input sequence and we treat the last 120 frames as the forecasted sequence by the ASAE. The results are reported in Table 8.3. One sees that LSTM_3LR are more accurate for short term forecasting, which is natural since this task is directly related to their learning criterion while it is not the case for our models. Yet for longer term forecasting, ASAE happens to perform better, which we interpret as the ability of ASAE to accurately capture in the encoding of a sequence its fundamental dynamics.

	75ms	225ms	300ms	500ms	750ms	1000ms
LSTM_3LR	**0.269**	**0.401**	**0.454**	0.568	0.678	0.766
ERD	0.455	0.534	0.568	0.647	0.719	0.764
ASAE	0.429	0.474	0.50	**0.567**	**0.612**	**0.654**

Table 8.3: Forecasting performance (Mean Squared Error per frame) of LSTM_3LR, ERD and ASAE for frame forecasting at an horizon of 75ms to 1 second.

Models	Accuracy
True Sequences	82%
CASAE	41.78%
ASFSAE	45.33%
DASAE-d3	48.02%
DASAE-d5	55.52%
DASAE-d8	67%

Table 8.4: Emotion classification accuracy on test sequences and on sequences generated by conditional models, with style encoding sizes of 2 to 5 for DASAE.

8.2.4 Style classification on generated sequences.

Finally Table 8.4 reports accuracy of an emotion classifier operating on sequences (with a similar architecture as the encoder in our models) that has been learned on training data and that is evaluated on transformed sequences by the various models that we presented and that may be used to change the style of an input sequence, CASAE, ASFSAE and DASAE. The performance of the emotion classifier on true test sequences is about 82% and serves a a reference performance that our models could ideally reach. Although the achieved accuracy on data generate by our models is significantly lower than this ideal performance, it may be seen that sequences generated with our models are far from random (which is about 12.5%) with the emotion being recognized at 41% at least. ASFSAE seem to work better than CASAE which is not surprising and DASAE may allow generating data that are recognized at up to 67% accuracy (depending on the style encoding size) demonstrating the ability of our framework to indeed transfer style between sequences.

8.3 Qualitative Evaluation

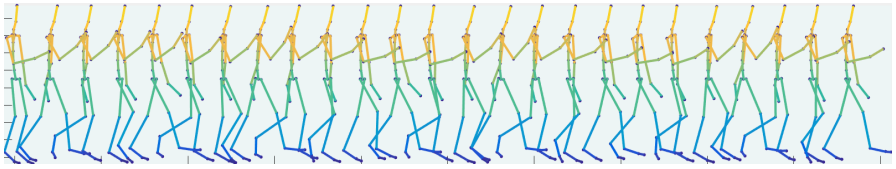
We illustrate now in a few figures some examples of animations produced by our models. More examples and videos of animations may be found at `https://bit.ly/2rwoGok`.

8.3.1 Unconditional models

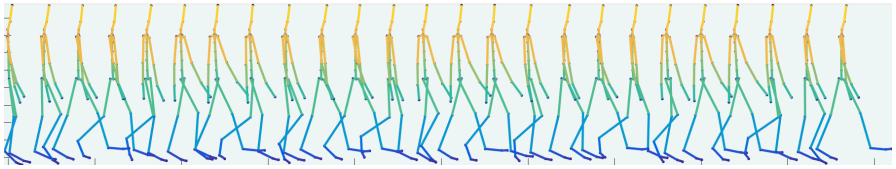
First we explore the behaviour of unconditional models and provide examples of synthesized motion samples.

We start with Figure 8.1 that shows a number of animations produced by the simplest model ASAE. These animations have been created randomly by the model using Algo-

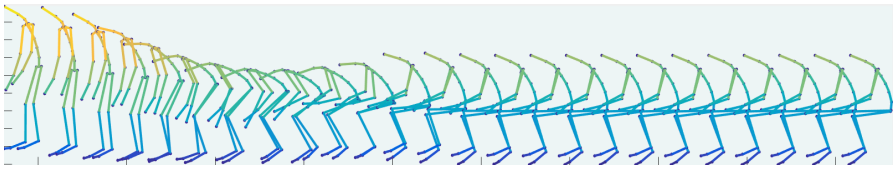
rithm 4. Depending on the case the synthesized animation may correspond to various activities that were observed in the training set (e.g. walking, sitting down etc). Note that the labeling of the synthesized sequences is performed manually.



(a) "Walk with smth in the Hands"



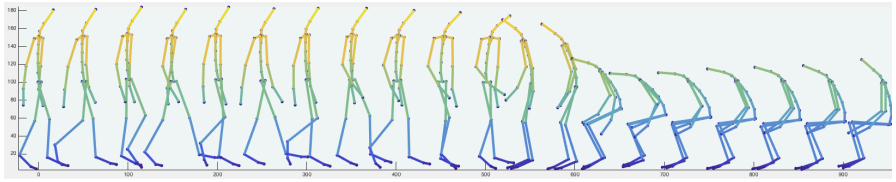
(b) "Simple Walk"



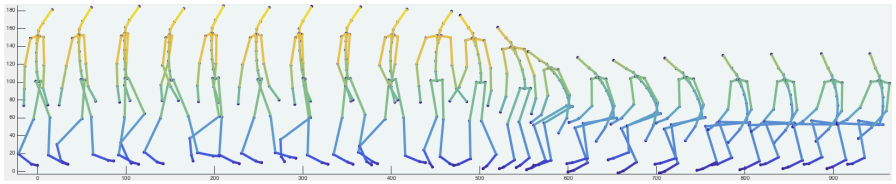
(c) "Sitting Down"

Figure 8.1: Motion examples which are randomly generated from ASAE

Note that one may synthesize a variety of motions with the interpolation method detailed in Algorithm 5. Figure 8.2 shows two generated motion using such an interpolation method in the latent space, using SAE and ASAE models. This example is typical of what we get on such examples where one sees a slightly smoother trajectory generated by ASAE with respect to SAE.



(a) SAE



(b) ASAE

Figure 8.2: Interpolation between two activities, "Simple Walk" and "Sitting Down" with the generative model gained with SAE and with ASAE.

Actually one may mine a little deeper what is learned by such models by exploring what is learned in the latent codes of motion data. Figure 8.3 shows animations produced when letting two components of the latent code vary from -2 to $+2$, the remaining components being fixed. More precisely we first sampled (using the prior normal distribution) a 50-dimensional latent code of a ASAE generator. Starting from it, we built 25 latent codes by making two components of the latent code vary (we arbitrarily chose the 8-th and 25-th components) from -2.0 to 2.0 with step of 1.0. We used the 25 latent code to generate new sequences, which we show to see how each of the two components impacts on the generated motions. We can see here that the 8-th dimension more or less controls the positions of the hands while the 25-th dimension controls the bending of the spine.

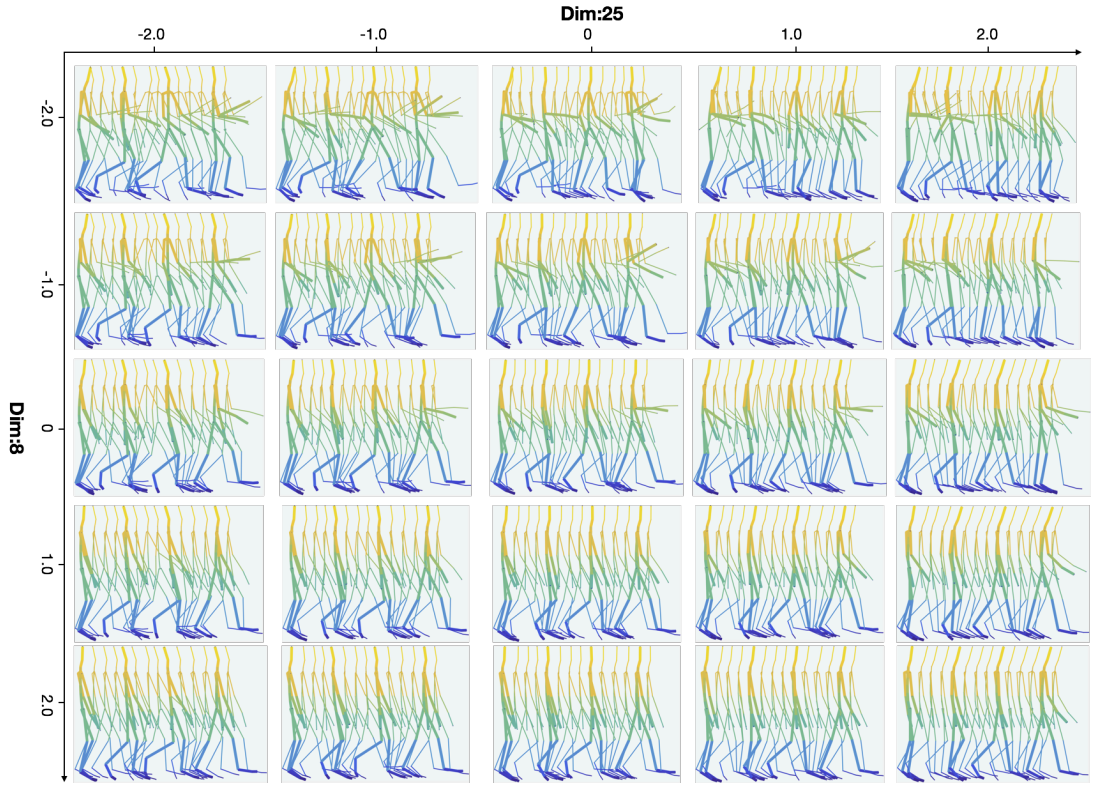


Figure 8.3: Exploring the latent space of an ASAE. The figure shows the 25 animations synthesized with a randomly sampled 50-dimensional latent code whose components 8 and 25 vary between -2 to $+2$ while other components remain fixed. One can see that the 8-th component controls the position of the hands, while the 25-th component controls the bending of the spine.

8.3.2 Conditional models.

We consider now the conditional variants that we described in the section 7.1 and algorithm 7. We first provide examples of animations gained with CASAE in Figure 8.4

where the emotion is used as side information (i.e. the model is then an emotion conditioned model). Next we show similar animations gained with ASFSAE models in Figure 8.5. In both cases the additional information is a one-hot-code encoding of the label, i.e. a binary vector of size 8 which is null everywhere but at the position of the label.

The 5 animations in the two figures are built so as to be comparable. To get this we built these animations using the following process. We first choose a test sequence $\mathbf{x}^{(i)}$ and feed it and its true emotion label into the encoder of one model to get a latent vector. Then we feed this latent vector as well as one of the 8 emotion encodings into the model's decoder to generate a new sequence. We reproduce this for the two models. The two figures then show five generated sequences which are variants of a test sequence computed with CASAE and ASFSAE models with 5 different emotion label inputs.

These qualitative results show that conditional ASAE (CASAE and ASFSAE) can learn the variations corresponding to the contextual label and generate plausible motions matching the specified activity or emotion.

Again, one may mine the learned latent space of a conditional modem by exploring how modifying one component of it modifies the generation process. Figure 8.6 illustrates this idea and reports motions that have been generated by a CASAE model with various emotion labels (one per row) when using a constant (randomly generated) latent code when a single component (11-th) only varies from -2.0 to 2.0 (The latent code is then the same for all motions in a column of the figure). Figure 8.7 shows similar results for a ASFSAE model.

Style transfer. Finally we show an illustrative example of style transferring in Figure 8.8 where a DASAE is used to transform three test sequences so that their style match the one of a target sequence. One sees that the style of the target sequence looks somehow energic and probably the emotion was joy or pride and that the transformed sequences seem to match better this style than the original ones.

8.4 Latent representation space

We investigated the latent space learned by the DASAE model. Figure 8.9 shows the style latent vectors of 10 000 training samples with a model exploiting a two dimensional style latent code. One can see that the style latent vectors corresponding to various emotion appear well separated into different areas of the latent space, which means our model indeed captured well the style information .

Autoencoder		Adversarial Discriminator	
Layer Name	type,activation,output size	Layer Name	type,activation,output size
Input	(200, 69)	Input	(1,dim of latent codes)
Layer1	LSTM, 'tanh',(200, 100)	Layer1	Dense,'relu',(1,100)
Layer2	LSTM, 'tanh',(200,100)	Layer2	Dense,'relu',(1,40)
Layer3	LSTM, 'tanh',(1,50)	Output	Dense, 'sigmoid',(1,1)
Layer4	Dense, 'linear',(1,dim of latent codes)	×	×
Layer5	Repeat, (200,50)	×	×
Layer6	LSTM,'tanh', (200,100)	×	×
Layer7	LSTM, 'tanh',(200,100)	×	×
Output	RNN, 'sigmoid',(200,69)	×	×

Table 8.5: Detailed Structure of Sequence Autoencoder and Discriminator(only for ASAE-based models)

Style Discriminator		Style Classifier	
Layer Name	type,activation,output size	Layer Name	type,activation,output size
Input	(1, 50)	Input	(1, dim of s)
Layer1	Dense,'relu',(1,100)	Layer1	Dense,'relu',(1,100)
Layer2	Dense,'relu',(1,40)	Layer2	Dense,'relu',(1,40)
Output	Dense, 'softmax',(1,8)	Output	Dense, 'softmax',(1,8)

Table 8.6: Detailed Structure of Style Discriminator and Style Classifier

8.5 Conclusion

The results in chis chapter show that the adversarial learning do lead to more accurate generative models than more standard models.

Our models, the pure generative one and conditional ones actually allow synthesizing motion under a specific context where the context may be chosen by hand at synthesis time.

Finally we were able to design a motion sequence editing model that allows transforming an input motion sequence to match the style of a second sequence.

Both conditional models and editing model have been showed to produce realistic motion sequence that actually encode the desired information, here the emotion.

This work both demonstrates the potential of neural architectures trained within the adversarial framework for the synthesis of realistic motion capture sequences, and may be seen as a first step towards highly flexible synthesis systems allowing a designer to synthesize sequences with a high level of monitoring.

hyperparameter	SAE	ASAE	CASAE	ASFSAE	DASAE-d*
batch size	200	200	200	200	200
optimiser for autoencoder	'Rmsprop' lr=0.001	'Rmsprop' lr=0.001	'Rmsprop' lr=0.001	'Rmsprop' lr=0.001	'Rmsprop' lr=0.001
optimiser for adversarial discriminator	× ×	'Adam' lr=0.001	'Adam' lr=0.001	'Adam' lr=0.001	'Adam' lr=0.001
optimiser for style discriminator	× ×	'Adam' lr=0.001	× ×	'Adam' lr=0.001	'Adam' lr=0.001
optimiser for style classifier	× ×	'Adam' lr=0.001	× ×	× ×	'Adam' lr=0.001
loss weights	×	[1.0,1e-3]	[1.0,1e-3]	[1.0,1e-3,1e-3]	[1.0,1e-3,5e-4,1e-2]

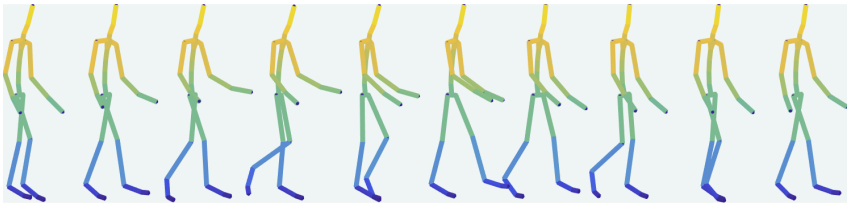
Table 8.7: Other hyperparameters

hyperparameter	LSTM_3LR / ERD
batch size	200
truncate_gradient	100
clip norm	25
momentum	0.99
inital learning rate	0.001
decay_schedule	[1.5e3, 4.5e3]
decay_rate_schedule	[0.1, 0.1]
noise_schedule	[250, 0.5e3, 1e3, 1.3e3, 2e3, 2.5e3, 3.3e3]
noise_rate_schedule	[5e-4,1e-3, 5e-3, 1e-2,1.5e-2, 2.5e-2, 3.5e-2]

Table 8.8: Hyperparameters of LSTM-3LR and ERD

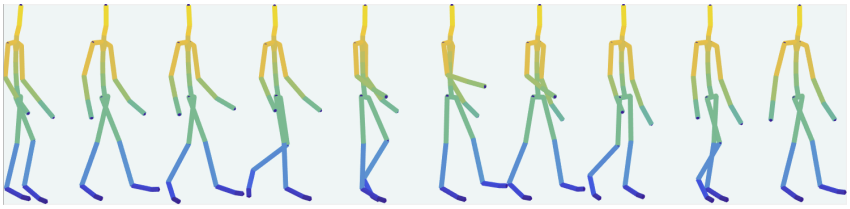
Model	LSTM_3LR	ERD
Input	LSTM,'tanh',(200,1000)	FullyConnect,'Relu',(200,500)
Layer1	LSTM ,'tanh',(200,1000)	FullyConnect,'Linear',(200,500)
Layer2	LSTM ,'tanh',(200,1000)	LSTM ,'tanh',(200, 1000)
Layer3	Dense, 'sigmoid', (200,69)	LSTM ,'tanh',(200,1000)
Layer4	×	FullyConnect,'Relu',(200,500)
Layer5	×	FullyConnect,'Relu',(200,100)
Layer6	×	FullyConnect,'Sigmoid',(200,69)

Table 8.9: Detailed structure of LSTM-3LR and ERD

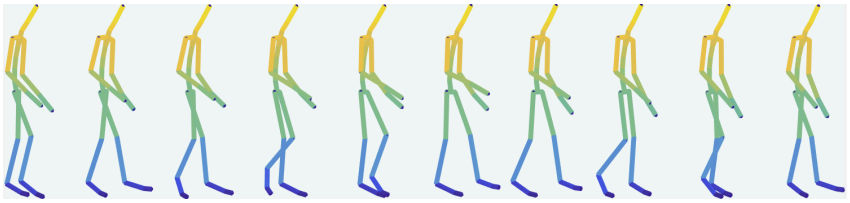


(a) "Anger"

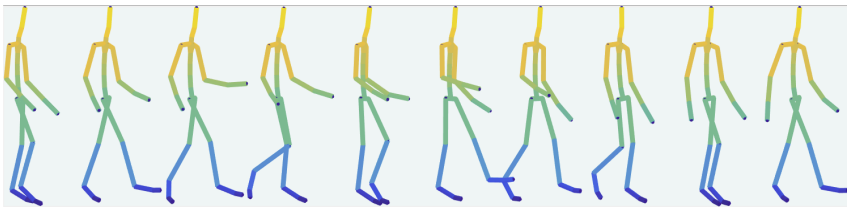
(b) "Panic Fear"



(c) "Pride"

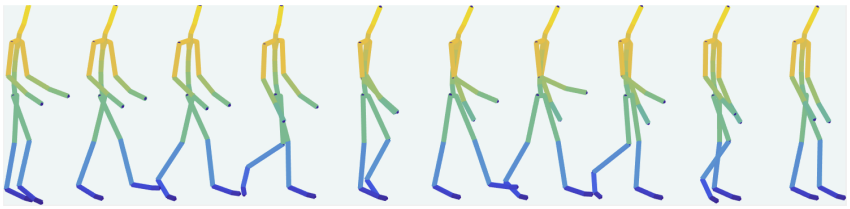


(d) "Shame"

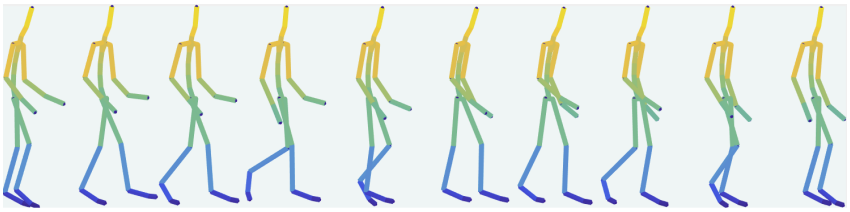


(e) "Joy"

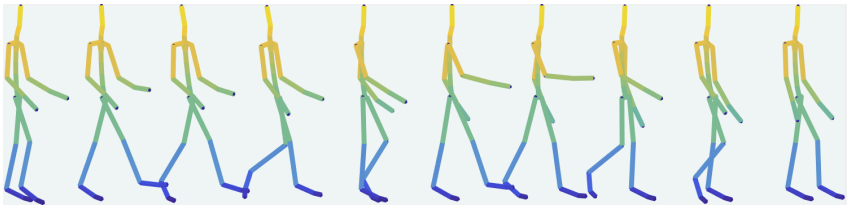
Figure 8.4: Examples of generated motion with CASAE with various emotion condition (see text for explanations). The 5 motions are comparable, i.e. obtained in similar situation, than the 5 motions in Figure 8.5.



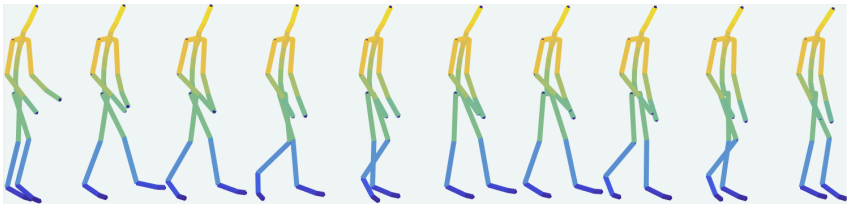
(a) "Anger"



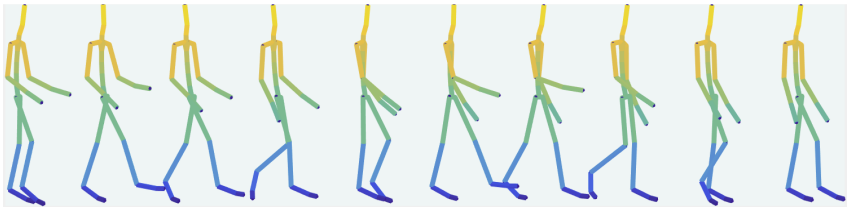
(b) "Panic Fear"



(c) "Pride"



(d) "Shame"



(e) "Joy"

Figure 8.5: Examples of generated motion with ASFSAE with various emotion condition (see text for explanations). The 5 motions are comparable, i.e. obtained in similar situation, than the 5 motions in Figure 8.4.

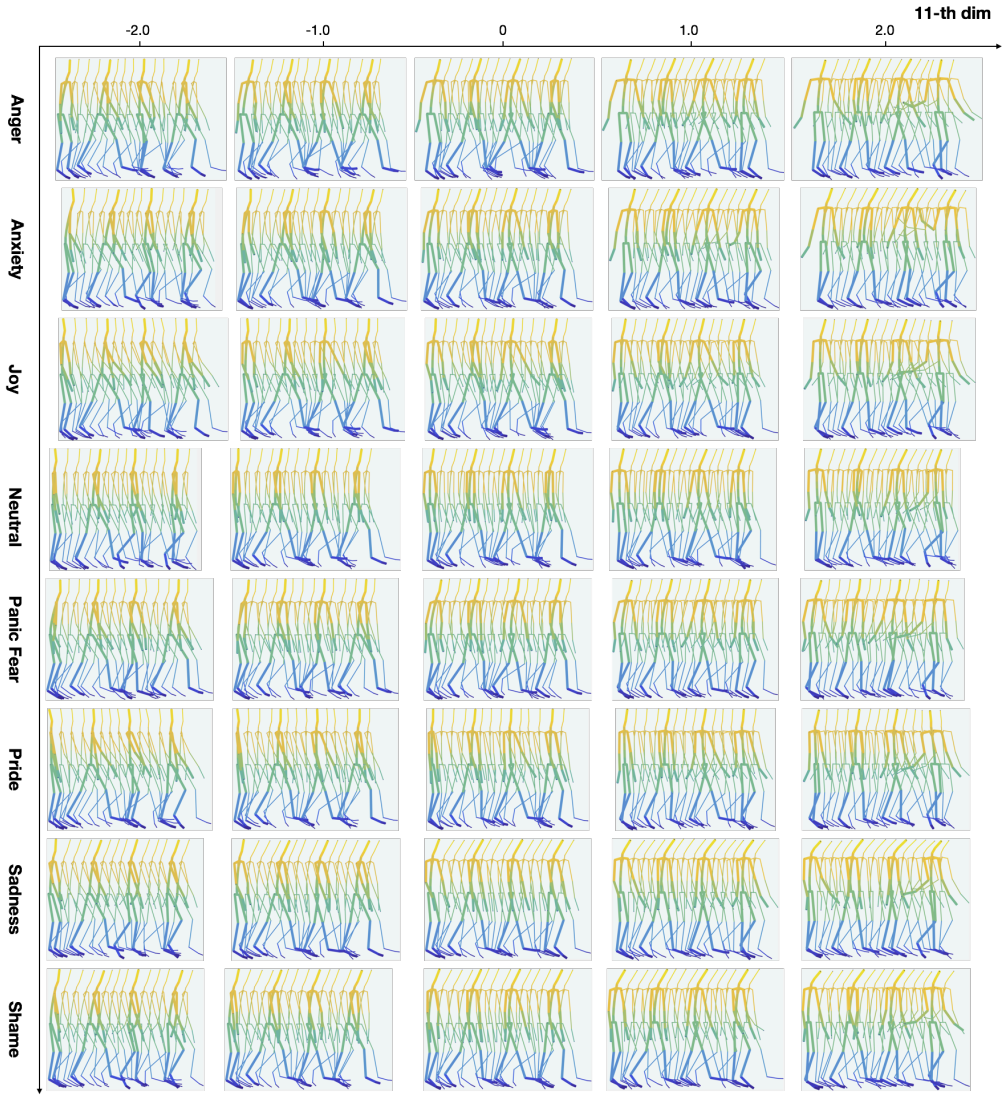


Figure 8.6: Exploring the latent space of CASAE. Motions are generated by synthesizing animations with a CASAE model with various emotion labels. All sequences are generated from a randomly generated latent code whose 11-th dimension only varies, from -2.0 to 2.0. The latent code used is constant in a column.

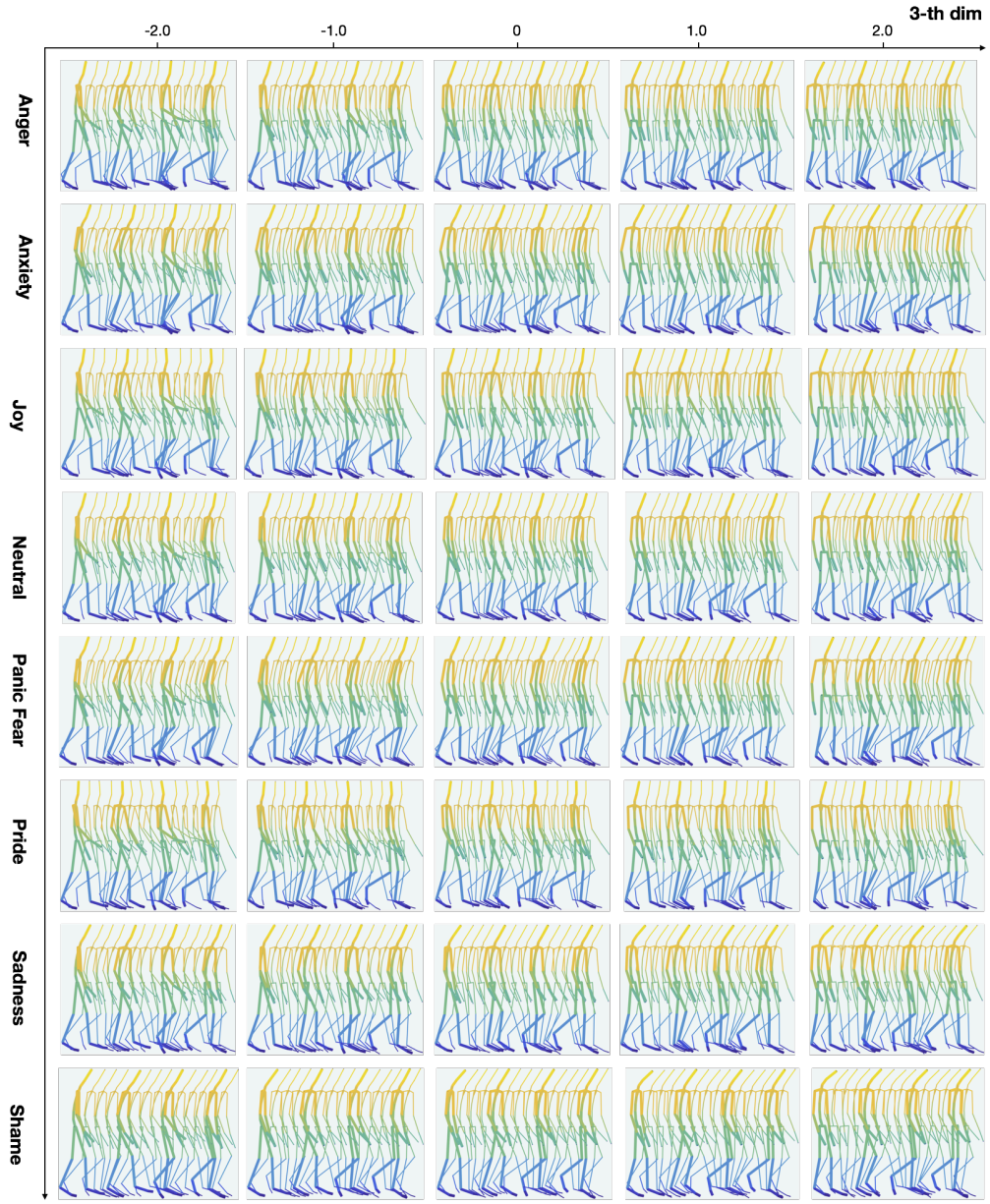


Figure 8.7: Exploring the latent space of ASFSAE. Motions are generated by synthesizing animations with a CASAE model with various emotion labels. All sequences are generated from a randomly generated latent code whose 3-rd dimension only varies, from -2.0 to 2.0. The latent code used is constant in a column.

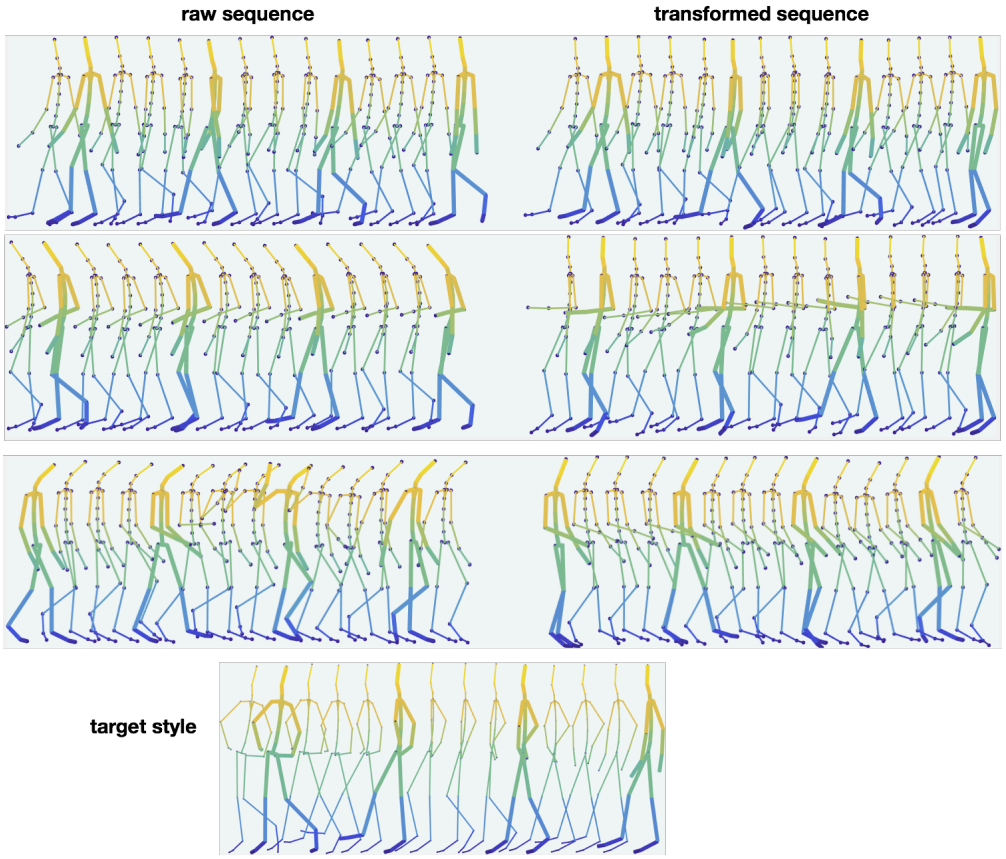


Figure 8.8: Style transferring with DASAE. The three sequences on the left are transformed in the three sequence on the right by applying a DASAE that transform input sequences so that their style match the style of the target sequence.

Layer Name	type,activation,output size	hyperparameters	Values
Layer1	LSTM, 'tanh',(200,100)	batch size	200
Layer2	LSTM, 'tanh', (200,100)	optimiser	'Rmsprop'
Layer3	LSTM, 'tanh', (1, 50)	learning rate	0.001
Layer4	Dense, 'linear', (1, 50)	×	×
Layer5	Dense, 'linear', (1, 50)	×	×
Layer6	Sampling, ,(1,50)	×	×
Layer7	Repeat, (200,50)	×	×
Layer8	LSTM, 'tanh',(200,100)	×	×
Layer9	LSTM, 'tanh', (200,100)	×	×
Layer10	RNN, 'sigmoid', (200,69)	×	×

Table 8.10: Detailed structure and hyperparameters of SVAE

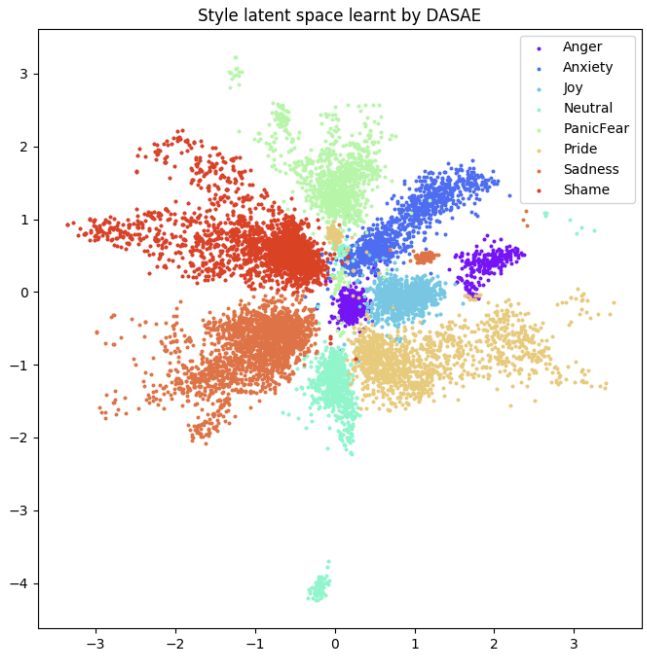


Figure 8.9: Illustration of the style latent vectors of 10 000 training samples using a DASAE model. Each color stands for one kind of emotion.

Chapter 9

Conclusion

Synthesizing realistic motion capture data with a high variability and with a mechanism enabling some control on the synthesis process is a difficult technical challenge. We explored a number of tracks in this thesis to this end.

In a first step we performed preliminary works on statistical models that have traditionnaly been used for such tasks, namely HMMs and Gaussian Process. We explored the use of these models for the modeling and the classification of motion sequences (HMMs) and for inverse kinematics (GPs), two subtasks of the more general motion synthesis topic.

We first focused on Contextual HMMs for exploiting contextual information. We designed a new EM learning algorithm for learning both the CHMM's parameters and the contextual representation. We evaluated our learning algorithm on activity recognition task, which show that the learned contextual representation can indeed improve the accuracy on the activity recognition task. Next we presented our work on Gaussian Processes for Inverse Kinematics. While he traditional Jacobian method is a pure optimization method which does not guarantee realistic posture, we proposed a new objective function. We leverage Gaussian Process to predict a prior distribution on the future postures. While we were able to propose improvements in both cases, where our work shows interesting results, we also encountered limitations which made us change the focus to designing neural networks based systems, taking inspiration from recent works in the machine learning and the deep leaning literature.

In a second step, following a current trend in machine learning, we focused our work on neural networks. We designed a number of models sharing a common basis architecture which combines sequence autoencoders and adversarial learning. Starting with this

architecture we first designed a pure generative model that allows randomly generating motion sequences that correspond to the underlying density probability in the training set.

Going further, we explored variants of the basis architecture that enable designing systems that take into account a contextual information, the emotion with which an activity was performed in our case. These models differ by the way adversarial learning is used to control the generation process. They may provide a designer with some control on the generated sequences.

Finally, we proposed a model for the style transferring task, where one wants to alter an input sequence with some feature (e.g. the emotion) that is present in a second input sequence. We built on the conditional models and added a new decomposition of the hidden representation of a sequence which allows putting suitable adversarial constraints with respect to style. The model embed style and content information into two latent vectors: a context latent vector and a style latent vector. The results show that our proposed model successfully separate the style and content and hence is able to transfer the style of one motion sequence to another.

Although there could be some follow up of the works in the preliminary part the most promising perspectives probably concern the neural networks models. As the last chapter showed it is possible to structure the latent space using various adversarial constraints. It is then expected that one could disentangle multiple factors of variations, instead of one (the emotion) in this work, using extensions of our ideas. Of course this would require datasets that are labeled accordingly which we do not have yet.

This yield to another extension of this work that would concern the labeling information needed to learn models. As few studies have shown recently such disentanglement models, separating content from style, may be learned with few, or no, supervision on the style labels. This open a new line of research that would allow learning controllable models for motion synthesis with today datasets where the many factors of variation of a motion sequence are not actually known for every training sequence.

Finally it may happen that our models generate motion sequences where some postures are unrealistic. We tried to include additional constraints on generated frames (or more generally on pairs or n-grams of generated frames) using a discriminator classifying True/Fake as in traditional GANs to ensure these all frames are realistic, this did not bring any benefit yet so that we did not include these results. Yet it seems that it should be a good idea that the realism of each of the generated frames would be enforced by some term in the objective loss used for training.

Bibliography

[1] Aritificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network.

[2] Cmu graphics lab motion capture database. http://mocap.cs.cmu.edu/.

[3] Euler angles. https://en.wikipedia.org/wiki/Euler_angles#cite_note-Euler-1.

[4] Omid Alemi, William Li, and Philippe Pasquier. Affect-expressive movement generation with factored conditional restricted boltzmann machines. In *Affective Computing and Intelligent Interaction, International Conference on*, pages 442–448. IEEE, 2015.

[5] Andreas Aristidou and Joan Lasenby. Fabrik: A fast, iterative solver for the inverse kinematics problem. *Graph. Models*, 73(5):243–260, September 2011.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[7] P. Baerlocher and R. Boulic. Task-priority formulations for the kinematic control of highly redundant articulated structures. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 1, pages 323–329 vol.1, Oct 1998.

[8] J. Baillieul. Kinematic programming alternatives for redundant manipulators. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 722–728, Mar 1985.

[9] Leonard E Baum and John Alonzo Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3):360–363, 1967.

[10] Ronan Boulic, Ramon Mas, and Daniel Thalmann. A robust approach for the control of the center of mass with inverse kinetics. *Computers & Graphics*, 20(5):693–701, 1996.

[11] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. *Iclr*, pages 1–13, 2016.

[12] Matthew Brand and Aaron Hertzmann. Style machines. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192. ACM Press/Addison-Wesley Publishing Co., 2000.

[13] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. 2004.

[14] Samuel R. Buss and Jin-Su Kim. Selectively damped least squares for inverse kinematics. volume 10, pages 37–49, 2004.

[15] Carlos Busso, Zhigang Deng, Ulrich Neumann, and Shrikanth Narayanan. Natural head motion synthesis driven by acoustic prosodic features. *Computer Animation and Virtual Worlds*, 16(3-4):283–290, 2005.

[16] M. Chen and L. Denoyer. Multi-view generative adversarial networks. *CoRR*, abs/1611.02019, 2016.

[17] Mickael Chen, Ludovic Denoyer, and Thierry Artières. Multi-view data generation without view supervision. In *International Conference on Learning Representations*, 2018.

[18] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *arXiv:1606.03657 [cs.LG]*, pages 1–14, 2016.

[19] Diane Chi, Monica Costa, Liwei Zhao, and Norman Badler. The emote model for effort and shape. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 173–182. ACM Press/Addison-Wesley Publishing Co., 2000.

[20] Stefano Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *Robotics and Automation, IEEE Transactions on*, 13(3):398–410, 1997.

[21] Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[22] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[24] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *CoRR*, 2015.

[25] Emily Denton and Vighnesh Birodkar. Unsupervised learning of disentangled representations from video. *CoRR*, abs/1705.10915, 2017.

[26] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. *Arxiv*, pages 1–10, 2015.

[27] Y. Ding, K. Prepin, J. Huang, C. Pelachaud, and T. Artières. Laughter animation synthesis. In *AAMAS*, 2014.

[28] Yu Ding, Catherine Pelachaud, and Thierry Artieres. Modeling multimodal behaviors from speech prosody. In *International Workshop on Intelligent Virtual Agents*, pages 217–228. Springer, 2013.

[29] Yu Ding, Ken Prepin, Jing Huang, Catherine Pelachaud, and Thierry Artières. Laughter animation synthesis. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 773–780. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[30] Yu Ding, Mathieu Radenen, Thierry Artieres, and Catherine Pelachaud. Speech-driven eyebrow motion synthesis with contextual markovian models. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3756–3760. IEEE, 2013.

[31] Chuong B Do. Gaussian processes. *Stanford University, Stanford, CA, accessed Dec*, 5:2017, 2007.

[32] Nesrine Fourati and Catherine Pelachaud. Emilya: Emotional body expression in daily actions database. In *LREC*, pages 3486–3493, 2014.

[33] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent Network Models for Human Dynamics. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4346–4354, 2015.

[34] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*. JMLR Workshop and Conference Proceedings, 2015.

[35] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[36] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

[37] Alex Graves. Generating Sequences with Recurrent Neural Networks. *Technical Reports*, pages 1–43, 2013.

[38] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[39] K Grochow, S L Martin, A Hertzmann, and Z Popovic. Style-based inverse kinematics. *Acm Transactions on Graphics*, 23(3):522–531, 2004.

[40] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based inverse kinematics. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 522–531, New York, NY, USA, 2004. ACM.

[41] Pawan Harish, Mentar Mahmudi, Benoît Le Callennec, and Ronan Boulic. Parallel inverse kinematics for multithreaded architectures. *ACM Trans. Graph.*, 35(2):19:1–19:13, February 2016.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[44] Daniel Holden, Ikhsanul Habibie, Ikuo Kusajima, and Taku Komura. Fast neural style transfer for motion data. *IEEE Computer Graphics and Applications*, 37(4):42–49, 2017.

[45] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):42, 2017.

[46] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics*, 35(4):1–11, 2016.

[47] John M Hollerbach and Ki C Suh. Redundancy resolution of manipulators through torque optimization. *Robotics and Automation, IEEE Journal of*, 3(4):308–316, 1987.

[48] Eugene Hsu, Kari Pulli, and Jovan Popović. Style translation for human motion. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1082–1089. ACM, 2005.

[49] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[50] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[51] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, August 2004.

[52] Guillaume Lample, Neil Zeghidour, Nicolas Usunier, Antoine Bordes, Ludovic Denoyer, and Marc'Aurelio Ranzato. Fader networks: Manipulating images by sliding attributes. *arXiv preprint arXiv:1706.00409*, 2017.

[53] Neil D Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in neural information processing systems*, 16(3):329–336, 2004.

[54] S. Levine, J.M. Wang, A. Harauxand Z. Popović, and V. Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics*, 31(4):1–10, 2012.

[55] Yijun Li, Sifei Liu, Jimei Yang, and Ming-Hsuan Yang. Generative face completion.

[56] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.*, 24(3):1071–1081, July 2005.

[57] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial Autoencoders. *arXiv*, pages 1–10, 2015.

[58] M. Mathieu, J. Jake Zhao, P. Sprechmann, A. Ramesh, and Y. LeCun. Disentangling factors of variation in deep representations using adversarial training. *CoRR*, abs/1611.03383, 2016.

[59] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[60] Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

[61] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *CoRR*, pages 1–7, 2014.

[62] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation mocap database hdm05. Technical Report CG-2007-2, Universität Bonn, June 2007.

[63] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of dynamic systems, measurement, and control*, 108(3):163–171, 1986.

[64] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4694–4702. IEEE, 2015.

[65] Anh Nguyen, Jason Yosinski, Yoshua Bengio, Alexey Dosovitskiy, and Jeff Clune. Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space. *Iccv*, (3), 2017.

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible conditional gans for image editing. *CoRR*, abs/1611.06355, 2016.

[68] Lawrence R Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[69] Mathieu Radenen and Thierry Artieres. Contextual hidden markov models. In *ICASSP*, 2012.

[70] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv*, pages 1–15, 2015.

[71] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative Adversarial Text to Image Synthesis. *Icml*, pages 1060–1069, 2016.

[72] Charles F. Rose, Iii Peter-pike, J. Sloan, and Michael F. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum*, 20:239–250, 2001.

[73] Jürgen Schmidhuber and Sepp Hochreiter. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[74] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. *Nips*, pages 3104–3112, 2014.

[75] Nick Taubert, Andrea Christensen, Dominik Endres, and Martin A Giese. Online simulation of emotional interactive behaviors with hierarchical gaussian process dynamical models. In *Proc. the ACM Symposium on Applied Perception*, pages 25–32. ACM, 2012.

[76] Graham W Taylor and Geoffrey E Hinton. Factored conditional restricted boltzmann machines for modeling motion style. In *Proc.the 26th ICML*, pages 1025–1032. ACM, 2009.

[77] Keiichi Tokuda, Takayoshi Yoshimura, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *IEEE International Conference on Acoustics, Speech, and Signal Processing. ICASSP 2000, 5-9 June, 2000, Hilton Hotel and Convention Center, Istanbul, Turkey*, pages 1315–1318, 2000.

[78] Deepak Tolani, Ambarish Goswami, and Norman I Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models*, 62(5):353–388, 2000.

[79] Raquel Urtasun, Pascal Glardon, Ronan Boulic, Daniel Thalmann, and Pascal Fua. Style-based motion synthesis. In *Computer Graphics Forum*, volume 23, pages 799–812. Wiley Online Library, 2004.

[80] A van den Oord, S Dieleman, H Zen, K Simonyan, O Vinyals, A Graves, N Kalchbrenner, A Senior, and K Kavukcuoglu. wavenet: A generative model for raw audio. *arXiv*, pages 846–849, 2015.

[81] C W Wampler, II. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Trans. Syst. Man Cybern.*, 16(1):93–101, January 1986.

[82] Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2008.

[83] Li-Chun Tommy Wang and Chih Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *Robotics and Automation, IEEE Transactions on*, 7(4):489–499, 1991.

[84] Qi Wang and Thierry Artieres. Motion capture synthesis with adversarial learning. In *International Conference on Intelligent Virtual Agents*, pages 467–470. Springer, 2017.

[85] Qi Wang, Thierry Artières, and Yu Ding. Learning activity patterns performed with emotion. In *Proceedings of the 3rd International Symposium on Movement and Computing, MOCO*, 2016.

[86] Qi WANG, CHEN Mickael, Artières Thierry, and Denoyer Ludovic. Transferring style in motion capture sequences with adversarial learning. In *Artificial Neural Networks, Computational Intelligence and Machine Learning(ESANN), 2019 European Symposium on*, 2018.

[87] L. Weber, S. W. Smoliar, and N. I. Badler. An architecture for the simulation of human movement. In *Proceedings of the 1978 Annual Conference - Volume 2*, ACM '78, pages 737–745, New York, NY, USA, 1978. ACM.

[88] Andrew D Wilson and Aaron F Bobick. Parametric hidden markov models for gesture recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(9):884–900, 1999.

[89] Andrew D Wilson and Aaron F Bobick. Hidden markov models for modeling and recognizing gesture under variation. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):123–160, 2001.

[90] W Wolovich and H Elliott. A computational technique for inverse kinematics. In *The 23rd IEEE Conference on Decision and Control*, number 23, pages 1359–1363, 1984.

[91] Xiaomao Wu, Maxime Tournier, and Lionel Reveret. Natural character posing from a large motion database. *Computer Graphics and Applications, IEEE*, 31(3):69–77, 2011.

[92] S. Xia, C. Wang, J. Chai, and J. Hodgins. Realtime style transfer for unlabeled heterogeneous human motion. *ACM Transactions on Graphics (TOG)*, 34(4):119, 2015.

[93] Katsu Yamane, James J. Kuffner, and Jessica K. Hodgins. Synthesizing animations of human manipulation tasks. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 532–539, New York, NY, USA, 2004. ACM.

[94] M Ersin Yumer and Niloy J Mitra. Spectral style transfer for human motion between independent actions. *ACM Transactions on Graphics (TOG)*, 35(4):137, 2016.

[95] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph.*, 13:313–336, October 1994.

[96] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks.