

# **An RNN Based Generative Model Of Human Motion**

*Joseph Elliot Yearsley*

Master of Science  
Artificial Intelligence  
School of Informatics  
University of Edinburgh  
2016



# Abstract

In this report, we investigate how an encoder recurrent decoder (ERD) network can be used to generate valid human motions after it has been sufficiently trained to predict the next frame on a valid manifold. We build upon work by Holden et al. [1] by working within the latent variable space of the auto-encoder; therefore we work upon the learnt manifold directly. We show that our model can generate motion which is controlled by low level signals, achieving state of the art results in online motion generation.

# Acknowledgements

I'd like to thank my thesis supervisor, Professor Taku Komura, for providing myself with every resource I required as well as his insights and preparation for further research. I would also like to thank Daniel Holden for his input and helpful discussions.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

*(Joseph Elliot Yearsley)*

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Terminology . . . . .	3
<b>2</b>	<b>Background &amp; Related Work</b>	<b>5</b>
2.1	Neural Networks . . . . .	5
2.1.1	Introduction To Neural Networks . . . . .	5
2.1.2	Convolutional Neural Networks . . . . .	6
2.1.3	Auto-encoders . . . . .	8
2.1.4	Recurrent Neural Networks . . . . .	9
2.2	Motion Generation . . . . .	12
2.2.1	Online Methods . . . . .	13
2.2.2	Offline Methods . . . . .	18
<b>3</b>	<b>System Overview, Evaluative Criteria &amp; Baseline Methods</b>	<b>25</b>
3.1	Overview of the system . . . . .	25
3.1.1	Learning the manifold . . . . .	25
3.1.2	Synthesis of Motion . . . . .	27
3.2	Data . . . . .	27
3.2.1	Original Data Set . . . . .	28
3.2.2	Latent Variables . . . . .	28
3.3	Evaluative Criteria . . . . .	29
3.4	Baseline . . . . .	30
<b>4</b>	<b>Preliminary Experiments</b>	<b>33</b>
4.1	Setup . . . . .	34
4.2	Initialisation of Weights . . . . .	34
4.3	Experiments . . . . .	35
4.4	Evaluation . . . . .	36

<b>5</b>	<b>Prediction of Human Locomotion</b>	<b>41</b>
5.1	Experiments . . . . .	41
5.2	Evaluation . . . . .	42
5.3	Discussion . . . . .	42
<b>6</b>	<b>Synthesis of Human Locomotion</b>	<b>45</b>
6.1	Experiments . . . . .	45
6.1.1	LSTM Width & Depth Experiments . . . . .	46
6.1.2	Type of Control Signal . . . . .	46
6.1.3	Additional Network Alterations . . . . .	47
6.2	Evaluation . . . . .	47
6.2.1	1 LSTM Layer . . . . .	48
6.2.2	2 LSTM Layers . . . . .	49
6.2.3	3 LSTM Layers . . . . .	49
6.2.4	Type of Control Signal . . . . .	49
6.2.5	Additional Network Alterations . . . . .	50
6.2.6	Comparison to Baseline . . . . .	51
<b>7</b>	<b>Discussion, Conclusion &amp; Future Work</b>	<b>55</b>
7.1	Discussion . . . . .	55
7.2	Conclusion . . . . .	56
7.3	Future Work . . . . .	56
7.3.1	Different Models . . . . .	56
7.3.2	Animation Research . . . . .	57
<b>A</b>	<b>Extra Network Images</b>	<b>59</b>
	<b>Bibliography</b>	<b>63</b>

# List of Footnotes

2.1 Jain & Seung used this noise in their network	9
2.2 We make use of this method in our baseline	9
3.1 <a href="https://youtu.be/CC6DaBglgB4?list=PLlpywOtApXsdWetAWY6-3KN5mDZEi7Wr8">https://youtu.be/CC6DaBglgB4?list=PLlpywOtApXsdWetAWY6-3KN5mDZEi7Wr8</a>	30
4.1 <a href="https://github.com/Brimborough/deep-motion-analysis/tree/LSTM">https://github.com/Brimborough/deep-motion-analysis/tree/LSTM</a>	34
4.2 <a href="https://youtu.be/CC6DaBglgB4">https://youtu.be/CC6DaBglgB4</a>	38
4.3 <a href="https://youtu.be/30zcp6gKTn8">https://youtu.be/30zcp6gKTn8</a>	38
4.4 <a href="https://youtu.be/8rmEYn0MqLs">https://youtu.be/8rmEYn0MqLs</a>	38
4.5 <a href="https://youtu.be/mJNtGyWlktg">https://youtu.be/mJNtGyWlktg</a>	38
5.1 <a href="https://youtu.be/8uIq506h0KY">https://youtu.be/8uIq506h0KY</a>	42
5.2 <a href="https://youtu.be/2Y7F_OzcJqo">https://youtu.be/2Y7F_OzcJqo</a>	42
5.3 <a href="https://youtu.be/YKdqSwnG9z0">https://youtu.be/YKdqSwnG9z0</a>	43
6.1 <a href="https://youtu.be/us2kyr4Pdh0">https://youtu.be/us2kyr4Pdh0</a>	48
6.2 <a href="https://youtu.be/FWeOH8agDwQ">https://youtu.be/FWeOH8agDwQ</a>	49
6.3 <a href="https://youtu.be/eLDQ2xXybGE">https://youtu.be/eLDQ2xXybGE</a>	49
6.4 <a href="https://youtu.be/aRguTwGHzUY">https://youtu.be/aRguTwGHzUY</a>	49
6.5 <a href="https://youtu.be/dg0BxaWjVgU">https://youtu.be/dg0BxaWjVgU</a>	50
6.6 <a href="https://youtu.be/yp_Yo92mFYc">https://youtu.be/yp_Yo92mFYc</a>	51
6.7 <a href="https://youtu.be/Ps-885uNaEQ">https://youtu.be/Ps-885uNaEQ</a>	51
6.8 <a href="https://youtu.be/fb_drzzq9YI">https://youtu.be/fb_drzzq9YI</a>	51
6.9 <a href="https://youtu.be/kCehLYqM7Bg">https://youtu.be/kCehLYqM7Bg</a>	51
6.10 <a href="https://youtu.be/yp_Yo92mFYc?t=14s">https://youtu.be/yp_Yo92mFYc?t=14s</a>	51
6.11 <a href="https://youtu.be/hmktSK-6Lg0">https://youtu.be/hmktSK-6Lg0</a>	51
6.12 <a href="https://youtu.be/kCehLYqM7Bg">https://youtu.be/kCehLYqM7Bg</a>	52

# Chapter 1

## Introduction

Virtual reality (VR) has recently been a hit with the gaming industry, with numerous headsets available, such as the Oculus Rift [2] and PlayStation VR [3]. VR immerses the user into a world which is realistic, and opens up opportunities for new virtual environments; for example, users could play video games set on completely different planets [4] or be with friends in a virtual shopping centre [5].

The rise in virtual reality brings novel problems for both the gaming and film industry, such as the ability to generate realistic human motions, since it has been proven VR is a lot more immersive [6]. Therefore we need computer animated individuals to be responsive in an immersive, realistic manner beyond what the current physics based game engines provide [7].

We also need the system to only require low dimensional control signals such that an animator or gamer could easily manipulate the generation process. For example, such a system could be used to model computer generated imagery (CGI) of a walking motion, enabling realistic shoots in less time.

We believe neural networks could be a solution to the problem having dominated recent advancements in many fields, with the likes of Convolutional Networks [8] and Long Short Term Memory networks [9] making up the basis for many novel models; for example, Encoder Recurrent Decoder models [10] which are currently outperforming previous machine learning techniques. However, they have been shown to suffer from averaging and trajectory issues in motion synthesis [11; 12]. We circumvent these issues by also inputting a control signal into our encoder recurrent decoder (ERD) network; showing it can be used to generate valid human motions after it has been sufficiently trained to predict the next frame on a valid manifold. We build upon work by Holden et al. [1] by working within the latent variable space of the auto-encoder; there-

fore we work upon the learnt manifold directly. This results in our model producing state of the art online human locomotion.

**Outline** In this work, we start by introducing the reader to the background of neural networks before providing a review of the current literature on motion generation ([Chapter 2](#)). Next, we introduce the reader to the data, evaluation techniques used & the baseline model ([Chapter 3](#)). We then perform some initial experiments ([Chapter 4](#)) which allow us to set the direction of our work effectively; for example, showing why we chose to use a LSTM [9] over a GRU [10], and other such decisions. We show the difference between prediction ([Chapter 5](#)) and synthesis ([Chapter 6](#)), through experimentation, explaining how we believe the literature should separate the two more clearly ([Chapter 2](#)). We also use [Chapter 5](#) as further preliminary experiments for our main contribution ([Chapter 6](#)), where we explain our state of the art (SOA) results and the experiments conducted to achieve them. Finally we conclude our work by discussing our SOA results and providing suggestions for future work ([Chapter 7](#)).

Whilst we provide quantitative results later on, we believe these are not useful due to our experiments showing it is possible to have a small quantitative error. But upon visualisation we experienced qualitative errors, therefore we provide numerous video links to supplement this work.

Our contribution is the following:

- A novel approach using latent variables, with a LSTM to synthesise SOA human locomotion.

## 1.1 Terminology

Here I will state the longer form of acronyms used in this proposal:

- VR - Virtual Reality
- ERD - Encoder Recurrent Decoder
- RNN - Recurrent Neural Network
- CNN - Convolutional Neural Network
- LSTM - Long Short Term Memory
- DNNs - Deep Neural Networks
- GPU - Graphical Processing Unit
- MSE - Mean Squared Error
- GMM - Gaussian Mixture Model
- SGD - Stochastic Gradient Descent
- BPTT - Back Propagation Through Time
- ReLU - Rectified Linear Unit
- ELU - Exponential Linear Unit
- PReLU - Parameterised ReLU
- SOA - State Of the Art
- FC/FF - Fully Connected



# Chapter 2

## Background & Related Work

In this chapter, we start by describing the background of neural networks ([Section 2.1](#)), giving a brief introduction to back-propagation. Then, we describe Convolutional Neural Networks (CNNs), Auto-encoders and Recurrent Neural Networks (RNNs). Finally, in [Section 2.2](#), we provide a literature review attaining to the latest research in the field, splitting it into offline and online motion synthesis.

### 2.1 Neural Networks

Neural networks have been shown to give state of the art results in multiple tasks, many of the networks have been made up of 3 main parts; feed forward layers (also called fully connected layers), convolutional neural networks and recurrent neural networks. In this section we introduce the reader to a brief history of neural networks ([Section 2.1.1](#)), we then go into more detail on convolutional networks ([Section 2.1.2](#)), auto-encoders ([Section 2.1.3](#)) & recurrent neural networks ([Section 2.1.4](#)).

#### 2.1.1 Introduction To Neural Networks

Neural networks have become mainstream again after an efficient algorithm for back propagation had been suggested by LeCun [[13](#)] and Rumelhart et al. [[14](#)]. These suggestions allowed for deeper neural networks to be trained with differentiable activation functions. Also, these suggestions allowed for an efficient algorithm for back propagation through time to be discovered by Werbos [[15](#)], which in turn allowed for recurrent neural networks to be used effectively. These algorithms led to major breakthroughs, such as convolutional neural networks [[16](#)] and long short term mem-

ory cells [17].

## 2.1.2 Convolutional Neural Networks

Convolutional neural networks (Figure 2.1) had first been proposed by Yann LeCun et al. in a paper on zip code recognition [16], having been inspired by Hubel and Wiesel's discovery of receptive fields within felines [18]. Since then they have been used in many fields [19], providing state of the art results mainly in computer vision problems such as ImageNet classification [20; 21; 22; 23] and 3D Image recognition [24].

Convolutional Neural Networks (CNNs) have the benefit that their weight matrices (feature maps) learn local features, regardless of where these features are placed; for example, a cat placed in the top left corner would still activate a certain filter map when placed within the bottom right corner. However, a Fully Connected (FC) network would not be able to do this since their weight matrices are fixed in place and are not slid around like in CNNs. Also, a FC network does not have multiple weight matrices acting as channels, this also limits their potential use as they can not learn disjoint representations of features; for example, a feature map looking for cats only.

Here we give a brief mathematical description of the CNN. In CNNs the convolution operator is commonly used, which can be seen as merging 2 signals together. If we have 1D discrete signals,  $f, g : Z \rightarrow R$ , then the convolution's output is defined as  $o[n] = f[n] * g[n] = \sum_{u=-\infty}^{\infty} f[u]g[n-u] = \sum_{u=-\infty}^{\infty} f[n-u]g[u]$  [25; 26].

The 2D signal is therefore:  $o[m, n] = f[m, n] * g[m, n] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f[u, v]g[m-u, n-v]$ . If we have an input tensor,  $x_{i,j}^c$ , of shape  $m \times m \times C$ , where  $m$  is the height and width, and  $C$  represents the number of channels (also called filters). Then we define the convolution layer as having a weight tensor  $W_{i,j}^{k,l}$ , with shape  $s \times s \times K \times C$ , where  $s$  represents the height and width,  $C$  the number of channels and  $K$  the number of feature maps. It also has a bias tensor,  $b_{i,j}^k$ , of shape  $(m-r+1) \times (m-r+1) \times K$  [27]. This results in each feature map,  $p^k$ , having entries  $p_{i,j}^k = n \left( \sum_l (W^{k,l} * x^l)_{i,j} + b_{i,j}^k \right)$ , where the function  $n(\bullet)$  is a non-linear activation function such as the logistic function.

In layman terms, we have a weight matrix which we slide over the input by a user defined stride, providing us with an output tensor.

Below we mention some recent advances in research that provide significant improvements in deep learning based results, which we also apply in this research.

1. Rectified Linear Units (ReLUs) [29],  $n(x) = \max(0, x)$ , have been shown to im-

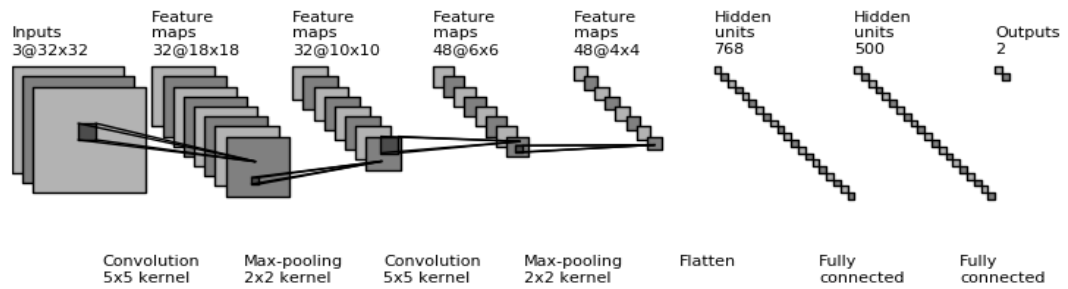


Figure 2.1: Example Convolutional Network, generated by code from [28].

Key information represented as: Channels@Shape and MxN kernel/window shape.

prove the convergence time [30] & not suffer from vanishing/exploding gradients, due to it forcing the network to learn a piecewise linear function.

2. Variants of the ReLU activation have also been shown to improve convergence times; for example, the Parametric-ReLU (PReLU) [31],  $n(x) = \max(0, x) + \alpha(\min(0, x))$ , allowing He et al. to surpass human level performance on ImageNet. The  $\alpha$  is a learnt parameter of the network (Figure 2.2).
3. Exponential Linear Units (ELU) [32],  $n(x) = \max(0, x) + \alpha(\min(0, \exp(x) - 1))$ , have provided the best result on CIFAR-100 dataset, whilst they also provide a more noise-robust deactivation state and show no extra benefit from batch normalization (Figure 2.2).
4. Batch Normalization [33] has been shown to reduce internal covariance shift, it achieves this by normalising each mini-batch at training time and freezing the normalisation parameters at test time, thus ensuring the data is within a set region. This has been shown to reduce the need for regularisation & carefully setting learning rates. It has also allowed the state of the art ImageNet model to achieve the same accuracy in 14 times fewer steps.
5. Dropout [34] was inspired by genetics and genetic programming, it carries forward its input with a fixed probability  $p$  and sets them to 0 otherwise. This forces neurons to learn a useful task in isolation, acting as a regularisation technique, and can also be seen as creating an ensemble of smaller networks with shared parameters.
6. Max-Pooling [8] is generally applied after each convolutional layer reducing the dimensionality and making the network robust to small translations. It is applied

such that it replaces a n-dimension window with the maximum value of that window, therefore down-sampling the data.

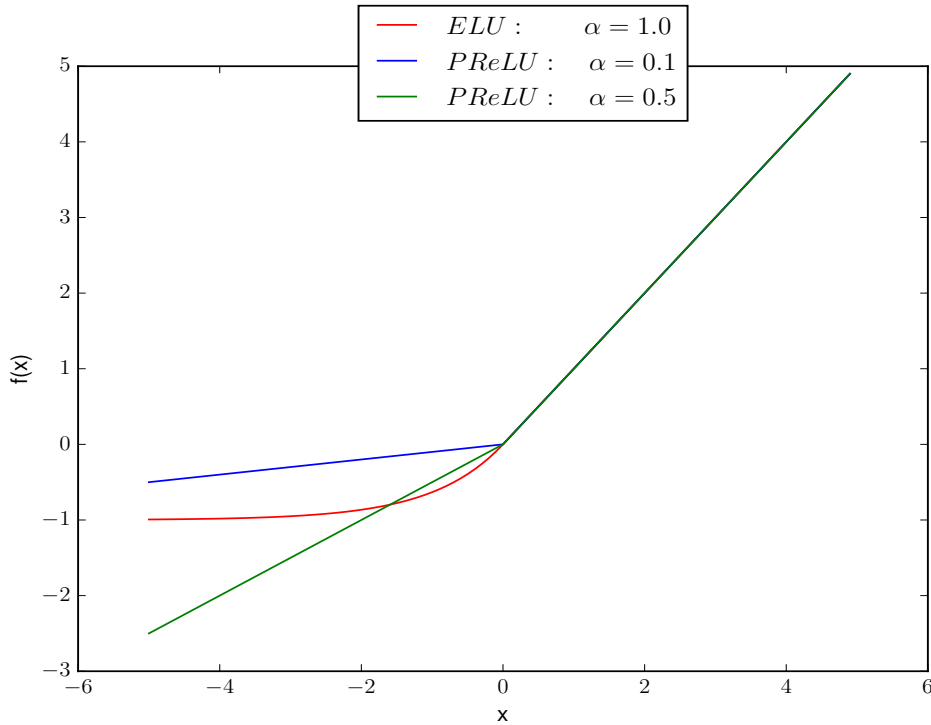


Figure 2.2: Example of PReLU and ELU Activations

### 2.1.3 Auto-encoders

Auto-encoders were first proposed by Zemel [35]. An auto-encoder is made up of an encoder, that converts an input into some latent representation, and a decoder, that reverses the conversion. The encoder transforms the data as follows:

$$f_i(x) = s(Wx + b) \quad (2.1)$$

where  $W$  is a weight matrix of shape  $d' \times d$ ,  $b$  is the bias vector of dimensionality  $d'$  and  $s(\bullet)$  is a non-linear function. And a decoder reverts the transformation like so:

$$g_i(y) = s(W'y + b') \quad (2.2)$$

The idea is to learn a latent variable representation,  $y$ , which can encode the data such that it can be decoded via  $g_i$ . It is also optional for the weights to be tied, so that the  $W' \equiv W^T$ .

Hinton et al. improved on this method by introducing the stacked auto-encoder model as a type of pre-training for a network [36]. Within this model, layers  $1 \dots N$  are added in sequence; training them as auto-encoders by taking the fixed output from the previous layer and producing the expected clean input to the current layer. After each layer has been trained, the weights are frozen and the following layer is then trained. It is necessary to add and remove an output layer at each training step until the actual output layer. Finally, once all layers have been trained the model is fine-tuned, with the weights being un-frozen, hence being updated throughout the network.

Then, Jain et al. proposed a De-noising Convolution Auto-Encoder [37], which used a convolutional auto-encoder with noise added to the input; the expected output was the clean input. This allowed the network to learn how to map from a noisy input to a clean hidden representation, hence learning how to project any noisy inputs back to the manifold. Jain and Seung [37] made use of a similar training technique to Hinton et al.; however they reconstructed the original network input instead of the previous layers output.

These suggestions led to Vincent et al. proposing the stacked de-noising auto-encoder which combined these approaches [38]. They show 3 main types of additive noise:

1. Gaussian Noise<sup>1</sup> 'List of Footnote' entries. footnote/counter. <sup>1</sup>, which samples the noise from a normal distribution, then adds it to the input. This benefits data which can potentially come from noisy environments such as images.
2. Masking Noise, where a fraction of the input elements are forced to 0 much like Dropout. This allows the system to become invariant to data potentially not appearing<sup>2</sup> 'List of Footnote' entries. footnote/counter. <sup>2</sup>.
3. Salt and Pepper Noise, a fraction of the input elements are forced to the 0 or 1 based upon a coin flip.

#### 2.1.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs, [Figure 2.3](#)) have always been of interest to researchers due to neurological observations of feedback connections in the brain.

They were however plagued with vanishing and exploding gradients [39], leading to

---

<sup>1</sup>Jain & Seung used this noise in their network

<sup>2</sup>We make use of this method in our baseline

the invention of long short term memory units (LSTMs) [9] and gated recurrent units (GRUs) [10].

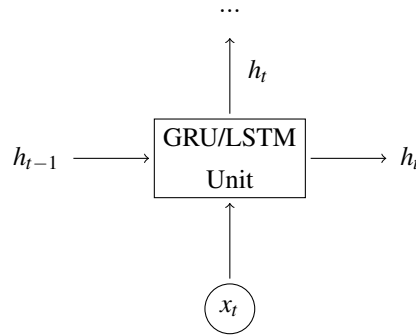


Figure 2.3: Example GRU/LSTM cell in a network

LSTMs make use of a numerous neural networks within a single unit; we now explain the unit referring to [Figure 2.4](#). As can be seen there are numerous gates, represented by activation functions, and element wise functions. Note,  $\sigma$  is the logistic function.

We start at the left of [Figure 2.4](#), where the previous output,  $h_{t-1}$ , is concatenated with the current input,  $x_t$ . Next we come across the forgot gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2.3)$$

This probabilistically learns which data is relevant, and is multiplied with the current memory state, with the update equation provided later.

Then we come across the input gate,  $i_t$ , and a candidate gate,  $c'_t$ , which together form an update for the memory state:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ c'_t &= \tanh(W_f \cdot [h_{t-1}, x_t] + b_f) \end{aligned} \quad (2.4)$$

These are then added to the current state to produce the new state, and the multiplied with the forget cell, providing the new memory state:

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t \quad (2.5)$$

Finally we need to decide what we are going to output, we again have another sigmoid to achieve a probabilistic output tensor, with which we element wise multiply with the current state of the unit. Leaving us with the output of the cell and next time

steps hidden value.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (2.6)$$

$$h_t = o_t \odot \tanh c_t$$

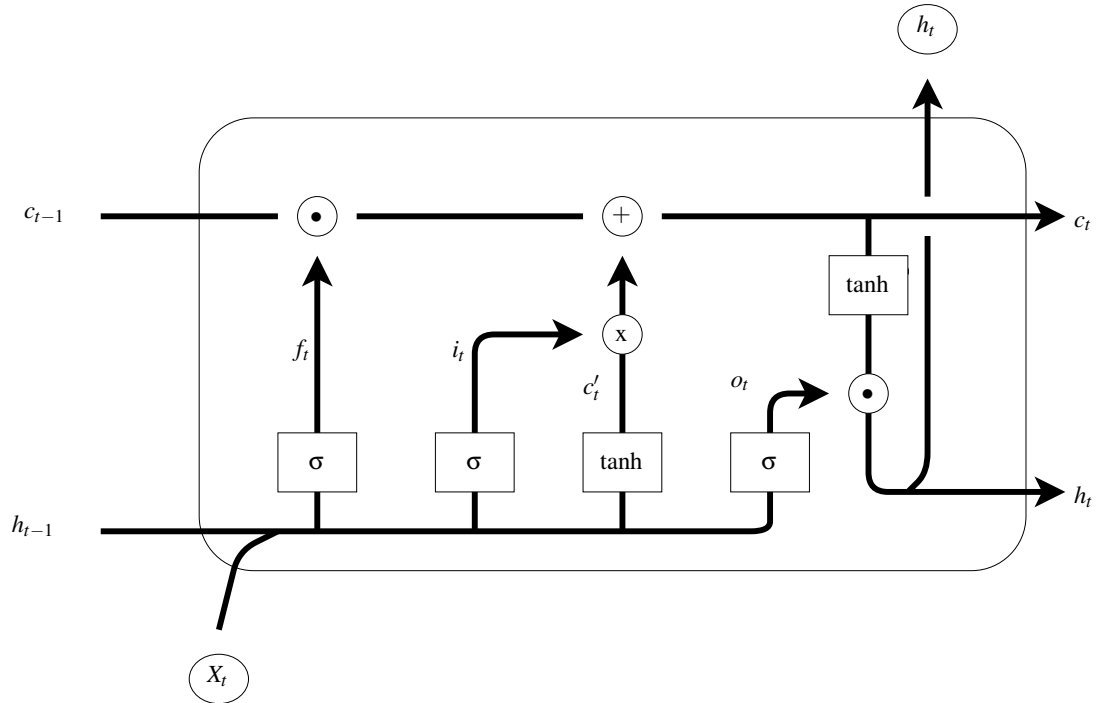


Figure 2.4: A LSTM Cell [40]

Since the first research paper, there have been several additions to LSTMs, peep-holes to the cells state, therefore concatenating  $c_{t-1}$  &  $c_t$  (depending upon where about in diagram the gates occur) into the above equations where  $[h_{t-1}, x_t]$  appear [41]; Another addition is that of skip-connections [42] which simply allow every LSTM layer access to the input as well as the previous layers output, and then skip the outputs of the LSTMs to the output layer.

LSTMs manage to avoid vanishing gradients due to the constant error carousel effect, which is a consequence of the memory cell,  $c_t$ , having a connection to itself in the following time step with weight 1, therefore it remains constant. Exploding gradients are usually avoided by clipping the gradients [9].

LSTMs have since gone on to dominate sequential tasks, such as producing hand writing & predicting text [43]. They have also been extended to 2D applications such as machine translation [44].

Another network which can be seen as a variation of the LSTM is the GRU. GRUs have fewer computations due to them having less gates; we will explain them by refer-

ring to [Figure 2.5](#).

The GRU cell consists of an input  $x_t$ , a hidden cell  $h_t$ , an update gate  $z_t$  and a reset gate  $r_t$ . The update gate is a simpler version of the input gate with the forget gate, regulating how much of the old memory should be kept and the reset gate determining how to mix the new output with the previous memory.

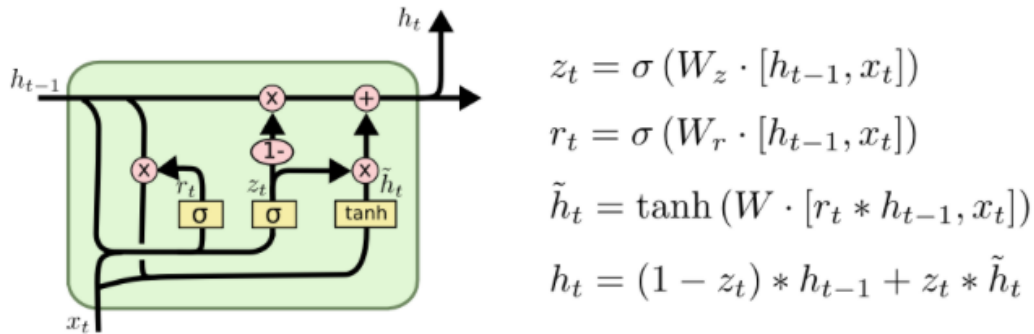


Figure 2.5: A GRU Cell [40]

The GRU avoids vanishing gradients in a similar way to the LSTM, due to the memory being a linear interpolation, therefore allowing the gradient to travel numerous time steps into the past, bypassing potentially saturating non-linearities. The GRU unit has been shown to perform as well as LSTMs whilst performing less computations [45], however they lack potential expressiveness due to fewer gates.

When these units are used to generate data it is common for them to have a seed, which is a set of data used to warm up the state of the units, it can be thought as burn in data, since we do not use the predictions until all of the seed data has been processed.

## 2.2 Motion Generation

Within this section we talk about online motion generation ([Section 2.2.1](#)) and offline motion generation ([Section 2.2.2](#)), noting the differences by reviewing two papers in each section. Firstly, we show the mean pose in [Figure 2.6](#), which is often the crux of motion synthesis. We also highlight the differences between prediction and synthesis'List of Footnote' entries. footnote/counter. <sup>3</sup>

<sup>3</sup> We note throughout the literature these terms are commonly used inter-changeably, however we believe that it would be of greater benefit to have a consistent application of each term.

We think synthesis should represent motion generated for the application of animations, since there is commonly a control signal input. Prediction should be used when the motion generated is based purely upon the input motion; i.e. computer vision problems, since it is the system predicting what will be

We think synthesis should represent motion generated for the application of animations, since there is commonly a control signal input. Prediction should be used when the motion generated is based purely upon the input motion; i.e. computer vision problems, since it is the system predicting what will be produced next.

We recommend these changes so that there will be less ambiguity in the literature..



Figure 2.6: A Mean Pose Which Drifts Around

### 2.2.1 Online Methods

Online methods synthesise the motion in real-time, sometimes with the input of a control signal; this is useful in computer games, where the character is controlled by the player in real-time.

We review 2 such methods, a ERD network by Fragkiadaki et al. [11] and a modelling system which makes use of low dimension control signals by Chai et al. [46]. We review the ERD since it makes use of RNNs to predict motion. We also use the ERD as our baseline method (Chapter 3); whilst we review Chai et al.'s work since it was one of the initial papers to consider using local models to learn from the data & it makes use of low dimensional control signals.

**Motion Modelling System** Chai et al. show it is possible to make use of 6-9, altering based upon each task, retro-reflective markers as control signals to recreate valid motion. They use a photography light above 2 spaced monocular cameras, which make use of epi-polar constraints to gather the 3D position of the markers, representing the low dimensional control signals for the system.

**Data** They build a database of motion using a Vicon capture system with 41 markers, capturing 1 hour of varying motion, such as locomotion and dancing. They

---

produced next.

We recommend these changes so that there will be less ambiguity in the literature.

store the poses as 18 joint angles, noting which angles they use, to create the synthesised motion. They search for the closest stored motion using a neighbour graph (a faster implementation of the nearest neighbour method), using the motion to learn a local linear model which is then used to synthesise novel motion.

The neighbour graph makes use of temporal coherence, having poses as nodes, and an edge between nodes,  $q_i$   $q_j$ , if they satisfy:

$$\|q_i - q_j\|_{L_1} < \max\left\{\frac{f_m \Delta d}{f_c}, \epsilon\right\} \quad (2.7)$$

where:

- The left hand side is the  $L_1$  distance between the nodes.
- $\epsilon$  is a specific search radius
- $f_m$  and  $f_c$  are the cameras rates for the motion capture and control interface.
- $\Delta d$  is the largest  $L_1$  distance between any 2 consecutive poses in the database.

Allowing the process to be sped up by searching the previous poses' neighbours only, instead of the entire graph.

**Local Models** The local models are built by constructing a search term to query the database with, using the euclidean norm between the control signal returned and the input control signal (aligned to the database root position and orientation) with the addition of a smoothing term:

$$\alpha \|c_n - T(z_t, z_0 c_t)\|^2 + (1 - \alpha) \|q_n - 2q_{t-1} + q_{t-2}\|^2 \quad (2.8)$$

where:

- $\alpha$  - is a trade off term between smoothness and control similarity, with a value of 0.8 being used.
- $T(\circ)$  - is a transformation matrix to align the input control signal with the database signals.
- $c_n$  and  $c_t$  are the returned control signal and current control signal respectively.
- $q_n$  represents the returned pose, whilst  $q_{t-1}$   $q_{t-2}$  are the 2 previous poses.

Finally constructing the linear models  $q_t$  via PCA upon the  $K$  closest examples:

$$q_t = p_t + U_t w_t \quad (2.9)$$

where:

- $U_t$  is a matrix constructed from the largest eigen vectors in the PCA decomposition matrix.
- $w_t$  is a  $B$ -dimensional low level representation of the current pose  $q_t$ .
- $p_t$  represents the mean pose of the  $K$  closest examples.

A leave one out cross validation process is used to determine the values for  $K$  and  $B$ , which proves expensive in terms of computation.

These local models can be seen as an alternative to feature maps in a CNN.

**Synthesis** Once the local model is retrieved it is used to synthesise motion by taking the root and orientation directly from the low level signals, then minimising the following energy function via gradient descent:

$$\operatorname{argmin}_{w_t} \left( w_t^T U_t^T \Lambda_t^{-1} w_t U_t + \alpha \|f(w_t; U_t, p_t, s, v_t, z_t) - c_t\|^2 + (1 - \alpha) \|U_t w_t + p_t - 2q_{t-1} + q_{t-2}\|^2 \right) \quad (2.10)$$

where:

- The initial term is the energy function of the assumed multivariate Gaussian prior.
- The second term is a scaled euclidean loss between the reconstructed markers and the initial low dimensional markers. With  $f(\circ)$  representing a forward kinematics function which makes the joint angles into 3D marker positions.  $\alpha$  is still kept at 0.8.
- The final term is the smoothing term with the local model factored in.

They initialise the optimisation with the closest example and use the damped least-squares method as a loss function, noting that the algorithm converges rapidly once it has received the local models due to the low dimensional representations.

**Results** In conclusion Chai et al. show their method can use data stored in a database to create smooth motion. They make use of fewer markers therefore allowing the user to be setup for motion capture quicker. However, their system is memory intensive (since it retains all the data to perform PCA locally) and has to perform a lot of data preprocessing, such as constructing a k-d tree. We mention that our system only requires latent variables (which are extracted from the data, [Chapter 3](#)) and matrices multiplications, resulting in less memory being required and fewer computations. Finally they note their system generalises in terms of marker positions from the database, much like our proposed system.

**ERD** Fragkiadaki et al. suggest using a ERD network to perform motion synthesis; the suggested ERD is shown in [Figure 2.7](#).

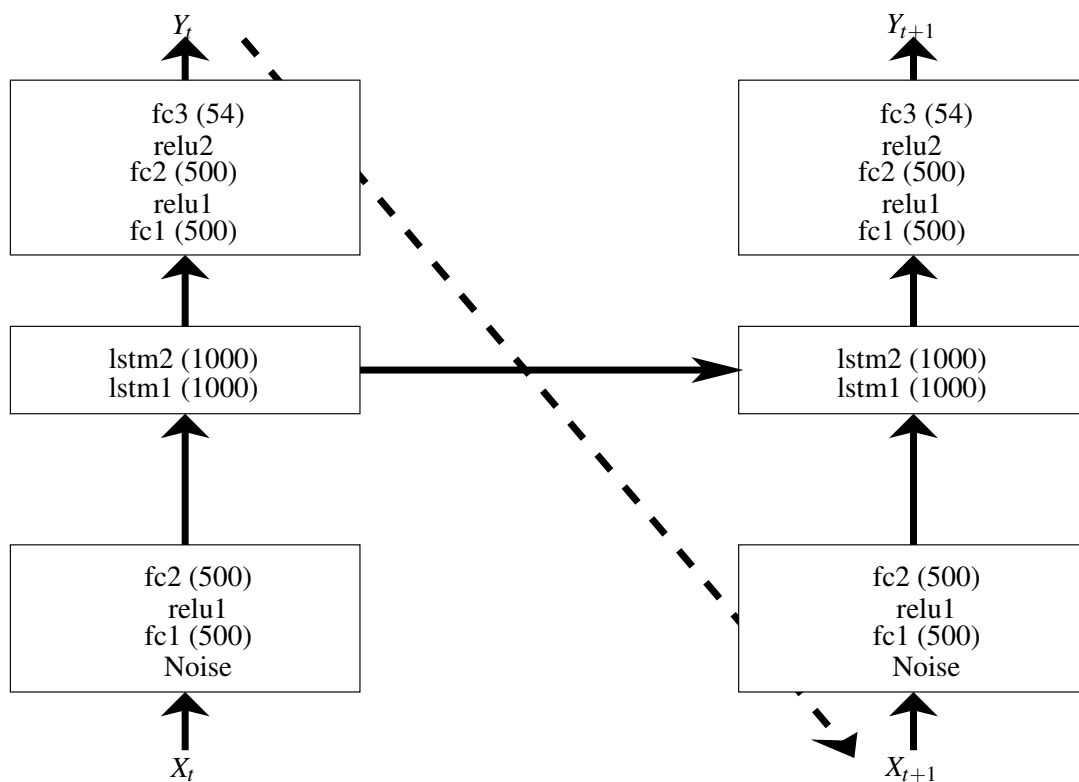


Figure 2.7: Fragkiadaki et al.'s ERD [11]

Here, we discuss some of the coding decisions and outcomes. We do not discuss LSTMs ([Section 2.1.4](#)), ReLUs ([Section 2.1.2](#)), or other research topics explored in this paper (as they are not related to motion generation).

**Data** Fragkiadaki et al. use the H3.6M dataset by Ionescu et al. [47]; which contains 15 activities performed by seven actors, captured using the Vicon system as

in Chai et al., Training on 6 of the actors and testing upon one actor.

The data is in a kinematic tree representation, which is a tree of the joints, where each joint has 3 degrees of freedom. The data also encodes the global position of the body in the  $x$ - $y$  plane, as well as global orientation around the  $z$  plane. All the data is standardised by subtracting the mean and dividing by the standard deviation along each dimension.

**Approaches** They make use of a deterministic approach, predicting the next frame vector, and a probabilistic approach, using the neural network to parametrise a Gaussian mixture model (GMM).

They trained using stochastic gradient descent, using BPTT, clipped the norm of the gradient at 25 and used masked noise, but instead of setting the value to 0, it was perturbed with zero mean Gaussian noise [48]. The masked noise was increased by a noise scheduler so that the network could learn to clean the noise, corresponding to a type of curriculum learning. This was found to be valuable as otherwise errors accumulated quickly within the network.

The probabilistic approach is described in the paper, they note it achieves the smallest quantitative error, however the qualitative results were similar to the deterministic approach using euclidean loss. Therefore we focus on the deterministic approach.

The deterministic approach makes use of time distributed feed forward networks, so that every frame passes through the encoder and decoder, this means that the expected output is a series of frames. These predicted frames are not used in training, they are used in testing at each time step the predicted frame is fed in at position  $t + 1$ . They used euclidean loss between target and predicted joints angles.

**Results** They achieve state of the art results in online motion synthesis, comparing it to the conditional restricted Boltzmann machine (CRBM) by Taylor et al. [49] & a novel approach they call LSTM-3LR, which contains a linear encoder-decoder, with 3 LSTMs of size 1000, similar to the ERD model.

They note that the quantitative results, whilst providing a nice metric for training, do not really have a link to the visualisations, since the LSTM-3LR has the lowest error upon testing, yet falls to the mean pose relatively quickly when visualised. We believe this is down to the mean pose being lots of local minima on the motion manifold; however, we discuss this in [Chapter 7](#). We also observe this in [Chapter 6](#). They find the ERD provides more realistic motion for longer but the generated motion is not as

smooth as the LSTM-3LR, both outperform the CRBM.

In conclusion, Fragkiadaki et al. achieve a pose which is similar to the ground truth and realistic motion for a longer period of time, however they do not make use of a control signal. This is due to their application being geared towards a computer vision problem, where such a signal is not available. We believe a long term prediction is not necessary in computer vision, as other approaches like applying ground truth updates is a common approach; i.e. the Kalman filter. If their system had access to a control signal, it would be a benefit to gamers who need real time control; we explore this method in [Chapter 6](#).

## 2.2.2 Offline Methods

Offline methods required a fixed input to synthesise the entire motion; this is a benefit for animators since they wish to generate the entire sequence at once then make small temporal refinements after the synthesis.

We review 2 such methods, a structural RNN by Jain et al. [12] and a auto-encoder plus feed forward network by Holden et al. [50]. We review both papers as they introduce novel approaches to offline motion generation, with Jain et al. introducing a new recurrent multi-task network, and Holden et al. introducing a novel method to control animations in the latent space.

### 2.2.2.1 S-RNN

Jain et al. introduced a novel model which they call a structural-RNN, a mixture of high level spatio-temporal graphs and RNNs, which are jointly trainable and fully differentiable. They motivate their work for prediction of human motion applied to autonomous vehicles, which is motion synthesis without a control signal.

**St-graphs to S-RNN** In a spatio-temporal graph the nodes represent the problem components, whilst the edges are interactions between components.

Their algorithm, to make a st-graph into a S-RNN, focuses upon the following high level steps (refer to [Figure 2.8](#) for a visualisation):

1. Roll out the st-graph in the temporal dimension and decompose it into a set of contributing factor components. Components determine one decision and are made up of nodes and edges, [Figure 2.8a](#).

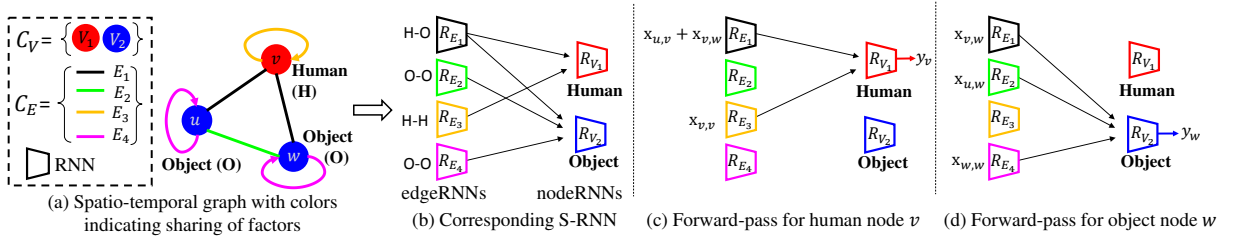


Figure 2.8: Visualisation of algorithm and training [12]

2. Semantically group the components and represent each group by one RNN, [Figure 2.8b](#).
3. They liberally give each independent component a RNN, to ensure a rich mixture.
4. Each nodeRNN combines the outputs of its connected edgeRNNs.

**Training** In training the Node features,  $X_{v,v}$  are concatenated with the corresponding edge outputs ([Figure 2.8c](#)) before being fed into the nodeRNN to predict the node label. The inputs to edge nodes are summed if there are 2 factors in the grouping, for example in [Figure 2.8a](#) the 2 black edges,  $E_1$ , are summed before entering  $R_{E_1}$  in [Figure 2.8b](#), Jain et al. note this is so a fixed network can be scaled to numerous variables.

**Model Setup** For motion generation they use the same dataset (H3.6M), Euclidean loss, the exponential map representation, and gradually add noise like Fragkiadaki et al. ([Section 2.2.1](#)). The resulting S-RNN is made up of 3 nodeRNNs (arm, leg, spine), 4 edgeRNNs (to model the interactions between them) and then an additional 3 edgeRNNs (to act as temporal connections between the nodeRNNs). Skip connections are used, within both RNNs, with edgeRNNs having a FC(256)-FC(256)-LSTM(512) architecture & nodeRNNs having a LSTM(512)-FC(256)-FC(100)-FC( $\circ$ ) architecture. The network is trained using SGD with a scheduled decay in the learning rate.

The architecture represents a complex multi-task ERD network, which we believe to be very interesting as Jonathan Schwarz in our research group has also found multi-task learning to improve his results (unpublished work).

**Evaluation** Within the evaluation stage, Jain et al. follow the experimental setup of Fragkiadaki. When forecasting, they use **50** seed frames to forecast **100** frames into the future. They show that they can forecast human-motion for twice as long as

Fragkiadaki et al. & that the S-RNN is modular; they show this by making the leg nodeRNNs independent of each other, then training a faster and slower walk upon 2 separate models. Swapping the left leg of the faster model with the slower nodeRNN, results in the left leg hopping to keep up.

However, since this behaviour can not be produced when using only one model, this method is an offline approach; it would be of interest if this could be converted to an online approach. As well as their method being offline, they can not control the trajectory of the motion, whereas our proposed system can. We finally note, Jain et al. mention that the ERD and S-RNN perform similarly upon the quantitative error, yet differ drastically when visualised.

### 2.2.2.2 Auto-encoder Convolutional Feed Forward Network

Holden et al. [50] build upon their previous auto-encoder work [1]. They first learn the auto-encoder; then, they insert another convolutional feed-forward network on top which is expected to learn the low dimensional control signals by inserting specific high dimensional signals. Figure 2.9 is Holden et al.'s visualisation of their network.

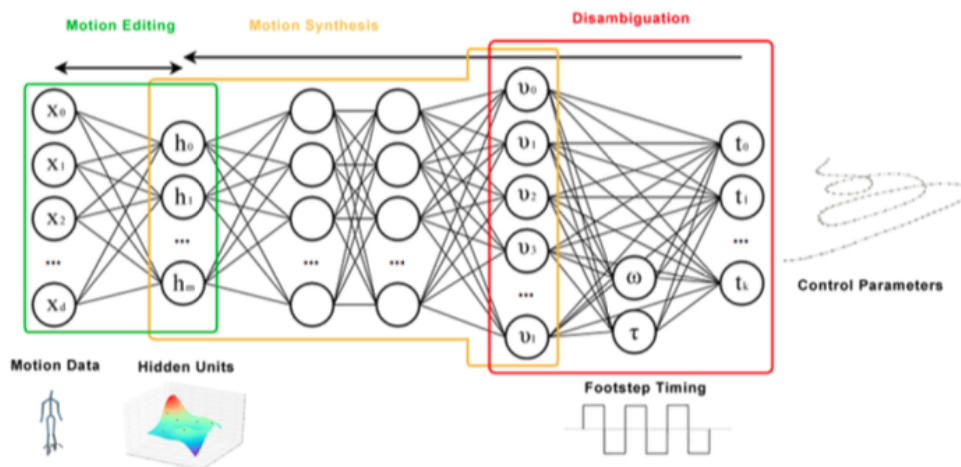


Figure 2.9: Model used by Holden et al. [50]

**The Auto-encoder** The auto-encoder Holden et al. use is slightly different to their previous work; it is a single 1D convolutional layer (kernel size 25, 256 channels) with max-pooling of a  $1 \times 2$  window (Figure A.1). They use a single layer with only one max-pooling operation as otherwise the motion appears too blurred. However, one convolutional layer is not enough to abstract the low level features, therefore they use

a convolutional feed-forward network to map these low level features to the expected motion. They used numerous open source motion databases, training for 15 epochs, using the Adam trainer [51] and Dropout [34] of 0.2, with a Lasso Loss function [52].

This auto-encoder provides the expected output for the convolutional feed-forward (FF) network. The input to the FF network is a concatenation of the control signals and another smaller feed forward convolutional neural network; this network learns the correct regression between a set of parameters, which parametrise the contact states equations, and the high dimensional control signals.

**Control Signals** The control signals represent the global velocity in the  $XZ$  plane, and the rotational velocity around the  $Y$  axis. These are abstracted from the data by averaging the hip and shoulder joints & then performing the cross product with the  $Y$ -axis (vertical axis), leaving a 3 dimensional vector.

**Footstep Timing** They found that the heel and toe contacts of both feet could be modelled through 4 square waves, taking advantage of the mathematical properties of the sine wave.

$$F(\omega, \tau) = \begin{bmatrix} \text{sign}(\sin(c\omega + a^h) - b^h - \tau^{lh}) \\ \text{sign}(\sin(c\omega + a^t) - b^t - \tau^{lt}) \\ \text{sign}(\sin(c\omega + a^h) - b^h - \tau^{rh}) \\ \text{sign}(\sin(c\omega + a^t) - b^t - \tau^{rt}) \end{bmatrix}^T \quad (2.11)$$

Everything but  $\omega$  and  $\tau$  are constants (found through experimentation), where:

$$\begin{aligned} a^h &= -0.1 \\ a^t &= b^t = b^h = 0 \\ c &= 1 \end{aligned} \quad (2.12)$$

They found that the parameters  $\omega$  &  $\tau$  could be extracted like so:

$$\omega_i = \sum_{x=0}^i \Delta\omega_x \quad (2.13)$$

$$\Delta\omega_x = \frac{\pi}{L_x} \quad (2.14)$$

where  $L_x$  (wavelength of the steps) is calculated by subtracting the timings of adjacent off-to-on frames and then averaging them by 4 (for the 4 contacts). By using  $\Delta\omega$ , it

allows the footstep frequency to alter throughout the motion.

$$\tau_i = \cos \frac{\pi d_i}{u_i + d_i} \quad (2.15)$$

where  $d_i$  &  $u_i$  represent the number of times the foot is down or up (calculated by using a threshold height from the ground) within the gait cycle which contains frame  $i$ .

These parameters are then made into a tensor  $\Gamma = \{\tau^{lh}, \tau^{lt}, \tau^{rh}, \tau^{rt}, \omega\}$ . They are then used as expected outputs given the control signals to train a small 2 layer convolutional neural network with a ReLU activation, followed by a Linear activation (Figure A.2) in the same manner as before. Holden et al. do not provide any specifics upon the filters here, suggesting it is up to the users which values are chosen.

**The FF Network** Here we briefly describe the convolutional feed-forward network (Figure A.3); The model is made up of 2 convolutional layers using ReLU activations, 64 and 128 channels respectively, followed by a max-pooling layer with a kernel of shape 1x2. Then there is a final convolution of 256 channels with a ReLU activation.

The control signals are concatenated with the foot timings to provide the high dimensional control signals to this network. The FF network is then trained, as explained previously, using the auto-encoders latent variables as the target outputs.

**Motion Editing** When editing motion in the hidden space, they add constraints which are then optimised separately; these are added due to the outputted motion not following the trajectory directly or other constraints being broken. They describe 3 constraints; positional, bone length and trajectory. These are then minimised within the latent space using gradient descent to minimise the summed cost below a set threshold. The decoding part of the auto-encoder is commonly used to retrieve the variables described below.

The positional constraint is:

$$Pos(H) = \sum_j \|v_r^H + \omega^H \times p_j^H + v_j^H - v'_j\|_2^2 \quad (2.16)$$

where:

- $v_r^H$  is the root velocity
- $v_j^H$  is the joint  $j$ 's velocity
- $v'_j$  is the target velocity

- $p_j^H$  is the joint  $j$ 's position

The bone length constraint is:

$$Bone(H) = \sum_i \sum_b \left| \|p_{b_{j_1}}^{H_i} - p_{b_{j_2}}^{H_i}\| - l_b \right|_2^2 \quad (2.17)$$

where:

- $p_{b_{j_1}}^{H_i}, p_{b_{j_2}}^{H_i}$  are the 2 end joints of  $b$  at frame  $i$ .
- $l_b$  is the length of bone  $b$ .

The trajectory constraint is:

$$Traj(H) = \|\omega^H - \omega'\|_2^2 + \|v_r^H - v_r'\|_2^2 \quad (2.18)$$

By performing this optimisation within the hidden space they found that the position of the motion & the rigidity of each bone is kept.

**Results** Holden et al. show their system is capable of transforming the style of an animation, generating crowd animations, following a animated curve using the correct speed and style, whilst providing smooth human motion. Their system can even perform online allowing the animator to alter specific time frames, if the convolution window is known; however their system can not generate continuous motion, whereas our proposed system can, which is useful in areas such as gaming.



# Chapter 3

## System Overview, Evaluative Criteria & Baseline Methods

In this chapter we start by discussing the overview of the synthesis system, which is made up of two separate networks. We discuss how the manifold is learnt (Section 3.1.1), describing the network, & briefly explain our generative system (Section 3.1.2). Then, we describe the latent data and its origins in more detail (Section 3.2) before explaining our evaluative methods (Section 3.3). We finally detail our baseline methods (Section 3.4), which are heavily based upon work by Fragkiadaki et al. (Section 2.2.1), however they are altered for use upon the CMU dataset.

### 3.1 Overview of the system

In this section, we describe our overall system; for our system to synthesise motion we need two different networks, one to get the global latent variables from the data (Section 3.1.1, Represented by the green in Figure 3.1), and then another to synthesise the motion using these latent variables (Section 3.1.2, Represented by the red in Figure 3.1). In Figure 3.1 we show a diagram which represents the entire system.

#### 3.1.1 Learning the manifold

In this section we provide the foundations of our work, since we are building upon work by Holden et al. [1]. Previously methods such as PCA [46], GPs [53] and RBFs [54] had been used. However, these methods suffered from requiring a lot of pre-processing and lots of computation; whether it be scaling (memory) issues or com-

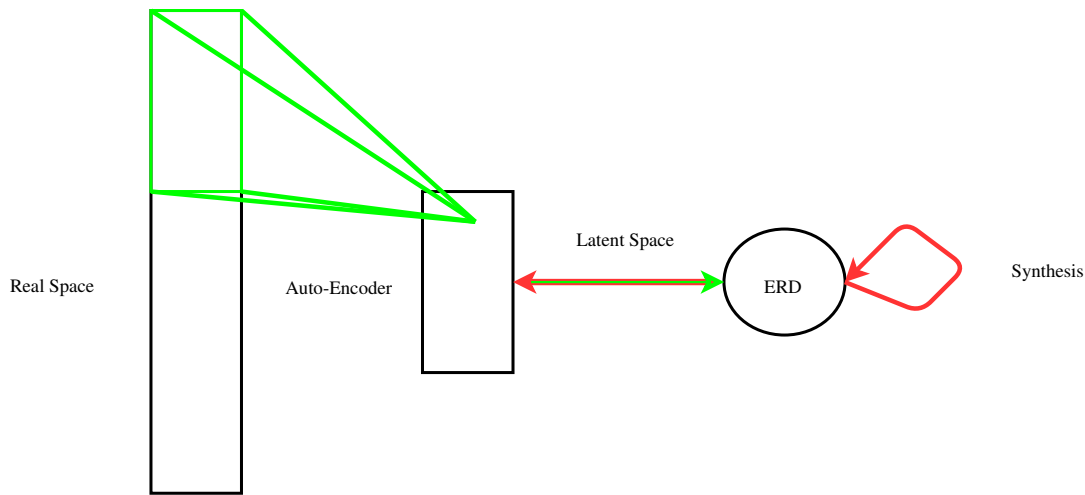


Figure 3.1: Overview of entire system - different colours represent separate systems

putational issues.

Holden et al. used a stacked de-noising auto-encoder to learn a motion manifold, upon the CMU database of human motion [55]. They originally used a SGD scheduler, Masked Noise with probability of 0.1 and trained the network like Vincent et al. [38]. Using the Lasso loss function [52], which is simply MSE plus a scaled sparsity constraint. The window size of the kernels was 15, such that small actions were captured by the initial layers with latter layers capturing larger actions.

Holden et al. found that their network was very capable of fixing corrupt motion, and able to learn the underlying motion manifold. Their network had the advantage that once it was trained the number of computations were relatively small, simple matrix multiplications, therefore running in  $O(1)$ , and memory usage was also  $O(1)$ , since the matrices were of constant size throughout.

This work has since then been improved upon, however not published, finding better results with the network in Figure 3.2. This network is very similar to the original, however there are some changes:

1. The activations have been altered to be ReLU activation functions.
2. The data was pre-processed to be 240 time frames with an overlap of 120.
3. The window size used in the convolution was altered to 25 from 15, to match the increase in overall frames.
4. Training was performed using the Adam Trainer with default settings [51].
5. The masked noise probability was increased from 0.1 to 0.3.

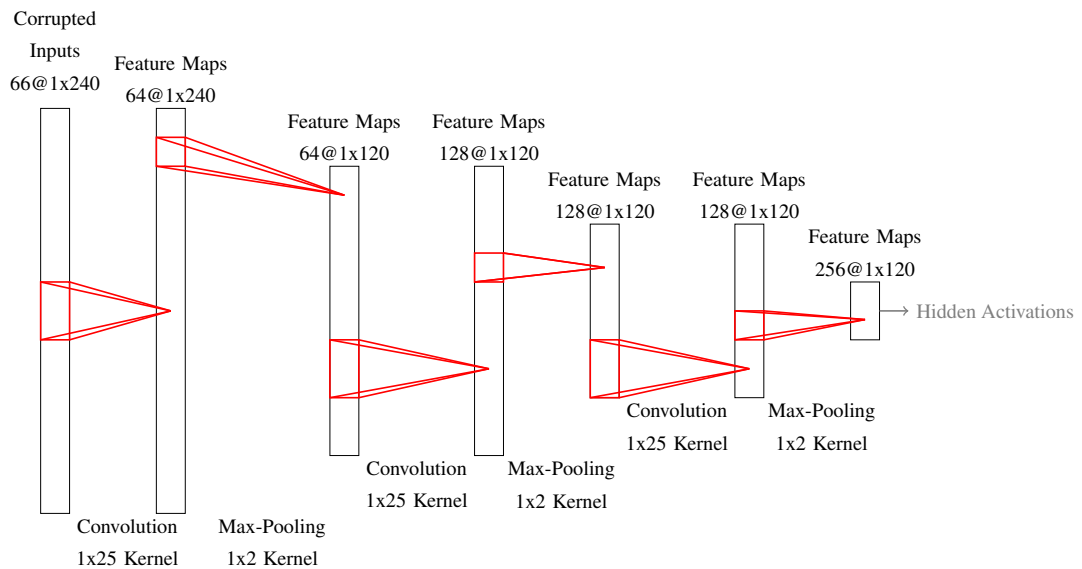


Figure 3.2: Improved Network Used

The hidden activations in [Figure 3.2](#) are not output as part of Holden’s network, these are the latent variables we extract to run our system upon, however we discuss these more in [Section 3.2](#). In [Section 3.2.1](#), we discuss in detail the data Holden used; we do not describe it here as the network and the data are weakly coupled.

### 3.1.2 Synthesis of Motion

In this section, we briefly introduce how our Encoder-Recurrent-Decoder (ERD) network will be setup; we mention further details and decisions in [Chapters 4, 5 & 6](#). Our synthesis model is based on a ERD network ([Figure 3.3](#)). It takes the auto-encoder’s latent variables ([Section 3.2.2](#)) as an input sequence and generates novel latent variables. These latent variables are then decoded, using the auto-encoder, to retrieve our real spaced visualisation. In [Section 3.2.2](#), we discuss in detail the data we use; we do not describe it here as the network and the data are weakly coupled.

## 3.2 Data

We will be using the hidden latent variables ([Section 3.2.2](#)) to train a LSTM so that we can synthesise human motion. We make use of a computer equipped with a Nvidia Quadro K620 GPU, which offers significant speed up of matrix multiplications over traditional CPUs.

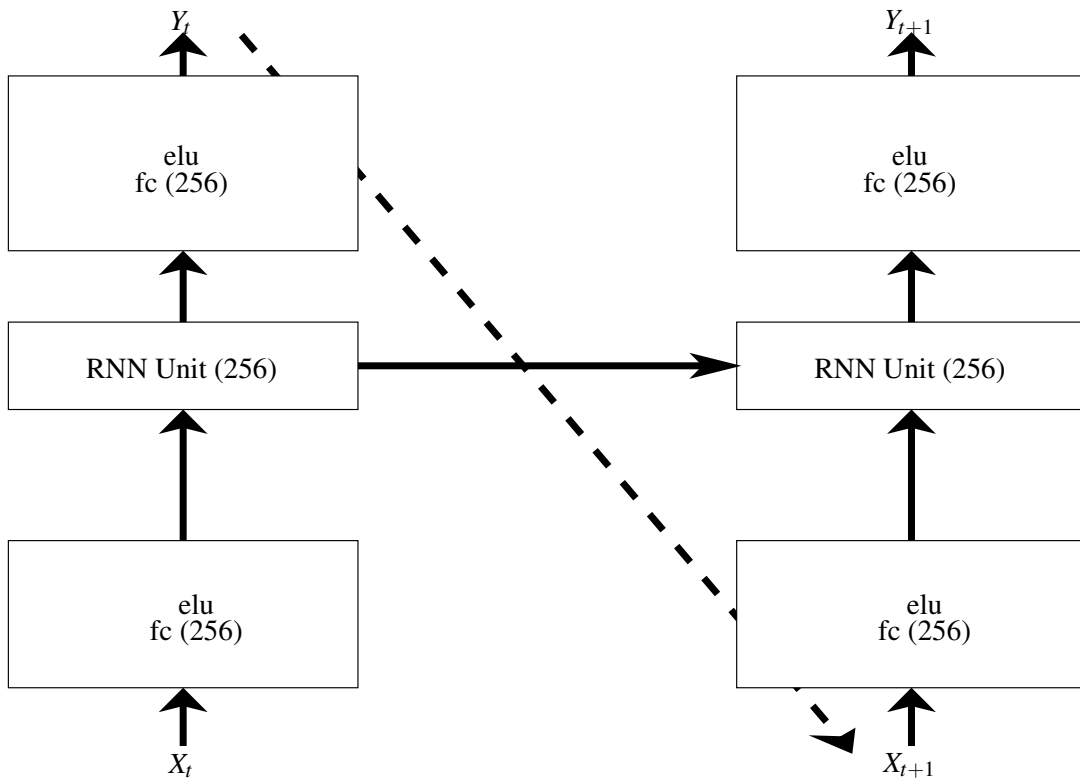


Figure 3.3: Example ERD Network

### 3.2.1 Original Data Set

The auto-encoder described in [Section 3.1.1](#) is trained upon the full CMU dataset [55], which is pre-processed to 240 frames, with 120 overlapping. Each frame is represented by a vector of length 66, which is from 21 joints having 3 degrees of freedom, with the addition of the global position in the  $x$ - $z$  plane and global orientation in the  $y$  plane. From now on we call these global variables our low dimensional control signal. The 1<sup>st</sup> and 2<sup>nd</sup> dimensions are flipped so that the 1D convolution pools the temporal dimension and convolves the frame's vector, using 66 channels, 1 for each scalar.

Before being input to the system the data is standardised by subtracting the mean and dividing by the standard deviation. The data is de-normalised when visualising, so that we visualise the original (real) space representation. We save the outputs of the system, as represented in [Figure 3.2](#).

### 3.2.2 Latent Variables

We take the latent variables and only keep the locomotion samples, leaving us with a tensor of the shape  $321 \times 256 \times 30$ , 321 samples, with 256 channels and 30 time steps.

Keeping only the locomotion data, as online synthesis is a hard research topic, therefore proof is needed the system can synthesise locomotion before it learns to synthesise other motions, such as jumping.

We swap the 1<sup>st</sup> and 2<sup>nd</sup> dimensions so that we have the correct shape for the LSTM, 321x30x256. We pre-process these samples by normalising them as before (Section 3.2.1); then, we permute the data and split it into 310 training examples and 11 testing examples. We do not make a validation set, as in our experiments we found it not to provide any difference. The restricted size of the samples factored into our decision for the number of testing examples to use.

We also split this data up to form input and expected output sequences. Taking 1...29 (along the temporal axis) as the input sequences and 2...30 as the output sequences. We explain more about why this is the case in Chapter 4. In Chapter 6 we explain how we alter this tensor by including the low dimensional control signals.

Before placing the data into the auto-encoder to visualise, we de-normalise our data & swap back the dimensions to retrieve the latent representation.

### 3.3 Evaluative Criteria

We evaluate our work in two ways: quantitatively and qualitatively. For both methods we experimented with different values for the number of seed frames, with the total number of frames always being equal to 240. We experimented with 160, 80, 40 and 16 seed frames. In a majority of our results we mention only using 16 seed frames and give a brief overview on other results; this is due to the results using less seed frames, therefore being the most impressive.

In our quantitative results we use the RMSE between our prediction and the real space data. This is different from Jain et al. and Fragiadaki et al. (Chapter 2) since they use the euclidean difference, between their prediction and ground truth, however this is just a scaled MSE. In addition to this, both describe that their quantitative results do not have a strong connection to the generation of human like motion. In Chapter 4, we do not provide quantitative results, due to the experiments only being preliminary to justify certain decisions for later architectures (Chapters 5 & 6).

We evaluate our qualitative results by looking at the same predictions as above (in the real (visual) space), mentioning our observations and providing videos so the reader can make their own conclusions (Playlist'List of Footnote' entries. footnote/counter.

<sup>1</sup>). In the videos we provide 3 different motions; the original capture (root), the motion after it has been only through the auto-encoder (reconstruction) and the output from our system (prediction). We do this so that it can be observed how well our system is doing against its input (reconstruction data) and the original data, with the latter being useful in [Chapter 6](#).

In [Chapter 6](#), we observe the following samples [1, 2, 5, 8, 10], picking these as they are the most independent, with other samples looking close to these motions. Whilst in [Chapters 4 & 5](#) we explain the sample chosen for the evaluation, due to these experiments being preliminary, helping us decide design choices for [Chapters 6](#).

We do not provide other people's observations like Jain et al., due to the fact that observations are quite clearly human like, or the mean pose. We also do not provide images, as when we visualised these the pose matched the root; however, in the global space shown in the video the motion was not following the trajectory, hence the images add no value.

## 3.4 Baseline

For our baseline we use an ERD model very similar to Fragkiadaki et al. ([Section 2.2.1](#)); however, we do make some changes:

1. Final Layer Size - changed to 66 since we have a vector of 66, not 54. Visualisation in [Figure A.4](#).
2. Optimizer - instead of SGD with decay we make use of the Nadam trainer [[56](#)], which is the Adam trainer [[51](#)] with Nesterov momentum. In our experiments this has shown better results over the Adam trainer.
3. Epochs ran for - We altered this from 10000 to 3800, due to the variations being much smaller in the CMU locomotion dataset.
4. Noise scheduler - Since we altered the number of epochs we also altered when the noise was applied to the inputs, adding increasing noise every 500 epochs after the initial noise is added after 300 epochs.
5. Noise - The following noise values were added [0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.7], since these are the values listed upon Jain's code base [[48](#)].

---

<sup>1</sup><https://youtu.be/CC6DaBglgB4?list=PLlpyw0tApXsdWetAWY6-3KN5mDZEi7Wr8>

6. In [Chapter 6](#), we also input the control signal to the baseline. To accommodate this change we run the system for 9000 epochs; we also change after how many epochs we add noise, adding a new noise value after every 1500 epochs.

A majority of these changes came from using a different database, with the coding decisions explained in [Chapter 4](#).

We implemented the model using Keras [\[57\]](#) and it ran for 17 seconds an epoch on a Nvidia GTX Titan. Keras is a framework for neural networks built upon Theano [\[58\]](#) which can calculate the derivatives automatically, we discuss this further in [Chapter 4](#).



# Chapter 4

## Preliminary Experiments

In this chapter we mention our initial experiments and the design decisions arising from them, we then stick by these decisions throughout our further experiments ([Chapters 5 & 6](#)).

In [Chapter 2](#), we mention multiple types of RNNs, regularisations and activation functions. We tried various experiment combinations, with different sized layers and depths of the model, to determine how best to proceed. First, we explain the basic model we will be experimenting with, giving reasons as to why we chose the framework and optimiser ([Section 4.1](#)). We then explain why we decided to use Glorot initialisation [[59](#)] for our layers, showing the mathematical basis ([Section 4.2](#)). We describe the initial experiments to decide upon which RNN cell to use ([Section 4.3](#)), taking into consideration the number of units, the activation functions used and regularisation. We finally evaluate the results gathered and some design choices ([Section 4.4](#)).

In overview, we experiment with the following to influence our design decisions:

1. RNN Cells: LSTM vs. GRU
2. Activation Function : ReLU vs. ELU
3. Regularisation: BN vs. Dropout vs. No Regularisation
4. Number of hidden units in RNN: 128, 256 or 512
5. Number of epochs: 50 or 200

## 4.1 Setup

We describe our initial design decisions in this section, including framework, activations and initial model used. We used the Keras [57] framework with Theano [58] as the back-end, this provided us with multiple benefits:

1. We could focus on the models rather than the implementations of the cells.
2. We could compile the models to CUDA code, allowing the code to be run on the GPU.
3. We could implement complex models, such as skip-connections, through the graph API provided.

We decided upon the NAdam optimiser which is Nesterov momentum added to the Adam trainer [51], Dozat showed it is often empirically better than the Adam trainer [56]. We train the models for 200 epochs, explaining why we did so in Section 4.4.

For our base model we start off with a ERD network consisting of FC(256)-LSTM(256)-FC(256), we use this as our base as Fragkiadaki et al. [11] cited how the encoder-decoder helped their network.

We keep the LSTM activations the same as Hochreiter et al. [9], whilst for the feed forward networks (FC) we decided upon the ELU activation, with  $\alpha$  set to 1.0, giving further details as to why in Section 4.4.

All of our code is available here'List of Footnote' entries. footnote/counter. <sup>1</sup>

## 4.2 Initialisation of Weights

To initialise the layers we made use of Glorot et al.'s suggestion [59] of returning weights from a normal distribution with variation:

$$\text{Var}(W_i) = \frac{2}{n_{in} + n_{out}} \quad (4.1)$$

This is based upon the assumption that weights and inputs are normalised, therefore have a mean of zero. This results in the joint variance being equal to  $\text{Var}(X_i W_i) = \text{Var}(X_i) \text{Var}(W_i)$ , therefore:

$$\begin{aligned} \text{Var}(Y) &= \sum_0^n \text{Var}(W_i) \text{Var}(X_i) \\ &= n \text{Var}(W_i) \text{Var}(X_i) \end{aligned} \quad (4.2)$$

---

<sup>1</sup><https://github.com/Brimborough/deep-motion-analysis/tree/LSTM>

Since we wish the output variance,  $Var(Y)$ , to equal the input variance,  $Var(X_i)$ , then  $Var(W_i)$  must equal:

$$Var(W_i) = \frac{1}{n} \quad (4.3)$$

We wish for this to occur in both the forward and backward pass, so Glorot et al. sum both passes resulting in [Equation 4.1](#). We used this initialisation since it is mathematically grounded and recommended by Yoshua Bengio [\[60\]](#).

## 4.3 Experiments

In our initial experiments we test both RNN units with only one recurrent layer, this indicates the better unit to carry on the work with. Our model is made up of a FC(256)-Recurrent(Z)-FC(256), where  $Z$  is a parameter we tested with values  $[128,256,512]$  ([Figure 4.1](#)).

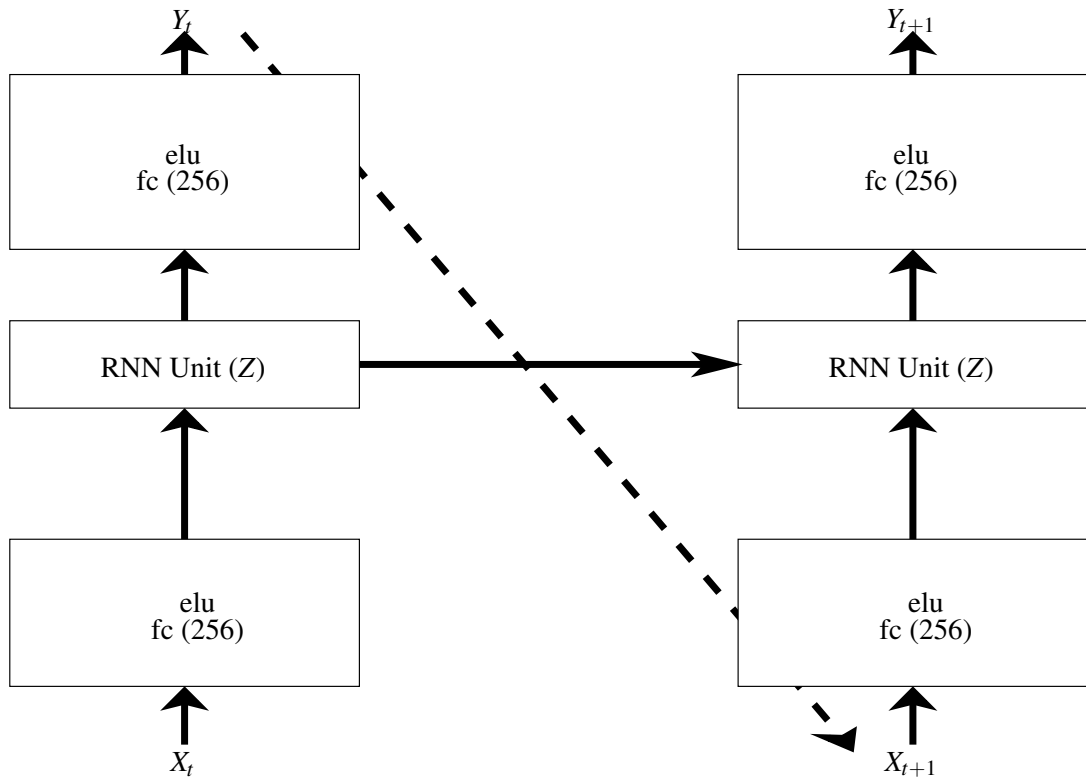


Figure 4.1: Network Used

We also make use of two different regularisation schemes; Dropout and Batch Normalisation (BN), both placed before the output layer. We based this upon the findings of Pham et al. [\[61\]](#), who found that placing the Dropout layer just before the decoding layers performed best. We test these along with the other models as the

hyper-parameters need tuning together. To tune the hyper-parameters we make use of Spearmint [62], a Bayesian optimiser of parameters. In BN experiments we have the batch size as the hyper-parameter being altered whilst in Dropout it is the probability of a unit being reduced to zero.

Spearmint takes a minimum and maximum value for the parameter being tuned, we set them as:

1. Dropout -  $0 < d < 0.3$
2. Batch -  $0 < b < 30$

We choose these bounds as a result of prior experience. We ran spearmint for 20 jobs, each job is represented by running our system for 100 epochs, returning the error so that spearmint can calculate the new parameters to use in the next job.

## 4.4 Evaluation

We first explain the choice of the ELU activation function, it was noted by Clevert et al. [32] that the ELU can bring the mean of the activations closer to 0, due to the possible negative values resulting from the function. This is similar to Batch Normalization, without the computational complexity. Clevert et al. also showed how the ELU activation made it easier for the network to learn the required function, compared to the ReLU activation. We show this experimentally in a FF(256)-LSTM(256)-FF(256) network, visualising the mean activations after the 1<sup>st</sup> FC layer in Figure 4.2. Also, we analysed our data after normalisation to find that the data lay in the range of  $-0.14$  to 65. This was an effect of the latent variables from Holden et al. [1] starting at 0, resulting from the ReLU activation, therefore the ELU could capture this range, were as the ReLU activation could not.

We also noticed that training 200 epochs did not vastly improve the error cost over training for 50 epochs (Figure 4.3), however upon visualising we noticed 200 epochs allowed the system to learn the temporal structure better. We believe that this is due to the mean pose, which we discuss further in Section 7.1.

We briefly explain the overall results of Batch Normalisation (BN), Dropout and the number of units we observed. We do not go into detail as the results are easily disambiguated from the visualisations; however, we do provide some videos which show 184 frames being predicted with 48 frames used as the seed, for each architecture.

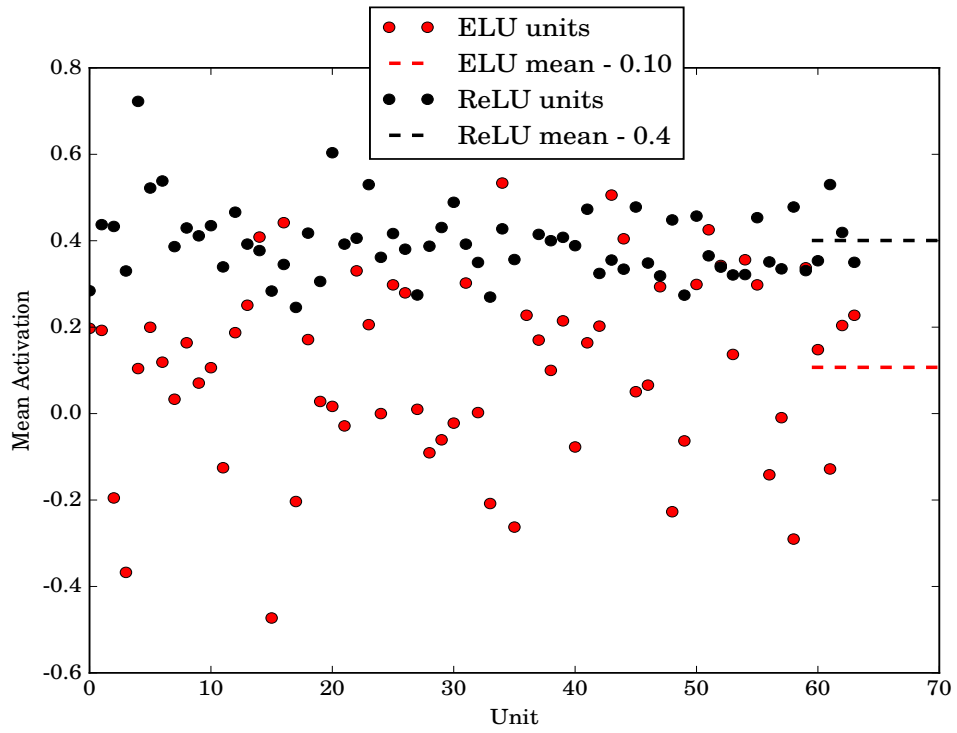


Figure 4.2: Mean Activation Of Each Units (Shown in groups of 4) &amp; Overall

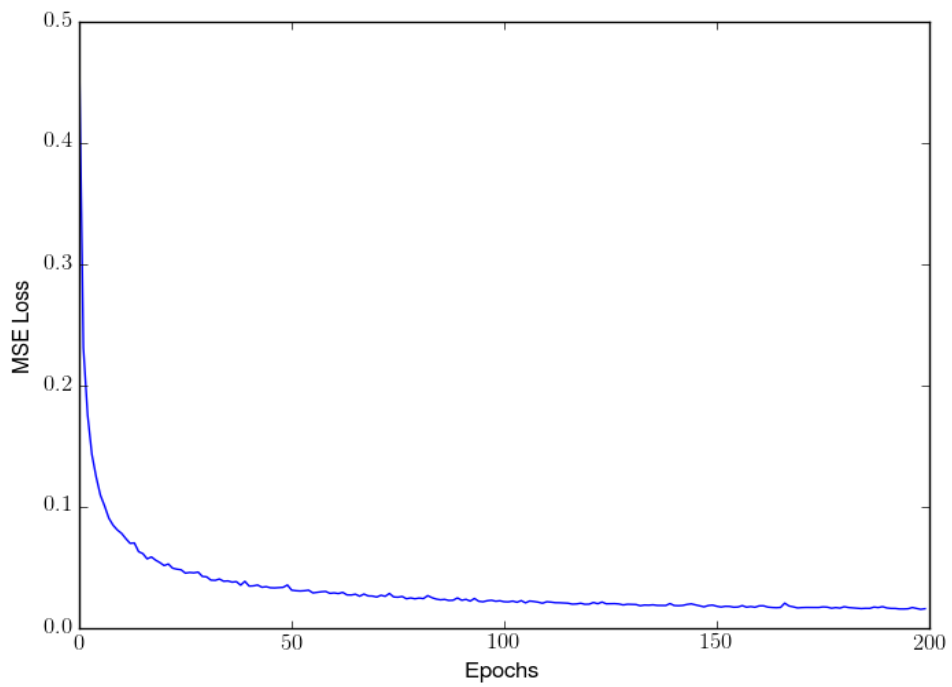


Figure 4.3: Loss shown over 200 epochs (The network used ELU activations)

We first of all mention that through experimentation having  $Z = 256$ , in our base model, performed better than either 128 or 512 (Video comparison'List of Footnote' entries. footnote/counter. <sup>2</sup>). We believe this is due to the latent variables already being encoded, therefore having less units results in a critical point of encoding too far, whilst having twice as many units results in the strength of encoding being lost in the network, which also leads to worse results. The number of 256 seems quite important as Holden et al. [50] and Jain et al. [12] both mention this number of units as working well for them.

We noticed that the GRUs were not able to perform as well as the LSTMs, which we believe is down to the lack of gates (Video comparison'List of Footnote' entries. footnote/counter. <sup>3</sup>).

Spearmint always returned values within the range of  $10 \pm 3$ , for the batch size, therefore we decided to take 10 as our batch size for future experiments, as the loss of BN reduced to a reasonable level. Upon visualising the BN results, we noticed they were inconsistent depending upon the run. We experienced visualisations which were the same standard to the Dropout experiments, potentially better, however we also saw that sometimes, if the final batch statistics were not correct, the motion came out stretched (Video comparison'List of Footnote' entries. footnote/counter. <sup>4</sup>). To counteract this we could have tested different batch sizes and ensured our batches have each variation of motion in them, however this is tedious and not generalisable, especially when compared to the Dropout results. We did see that BN was better at predicting the trajectory when compared to using Dropout or no regularisation. We discuss BN more in Chapter 7, as it would be interesting to apply it with a larger dataset.

For the Dropout experiments, we noticed that generally the value returned was quite low,  $\sim 0.1$ , which gave us motion with the joint constraints being kept visually. We compared this to a network with no regularisation and realised that although the quantitative error was greater, the visualisation was better (as can be seen here'List of Footnote' entries. footnote/counter. <sup>5</sup>).

Although not using Dropout performs the best with 1 layer, we keep it in consideration for when we experiment with more layers (Chapters 5 & 6), this is due to it not having any significant errors such as BN and being easier to tune in relation to the performance

---

<sup>2</sup><https://youtu.be/CC6DaBglgB4>

<sup>3</sup><https://youtu.be/30zcp6gKTn8>

<sup>4</sup><https://youtu.be/8rmEYn0MqLs>

<sup>5</sup><https://youtu.be/mJNtGyWlktg>

improvement.

We conclude our evaluation by stating that we have seen using 256 hidden units in a LSTM cell, ELU activations for the FC layers and no regularisation performed best. We make use of these decisions in [Chapters 5 & 6](#).



# Chapter 5

## Prediction of Human Locomotion

Prediction of human motion only makes use of the previous frames with no external signal, it is therefore inherently interesting in the field of computer animation. If you can predict the next frames without a control signal, then with a signal the prediction should be better, therefore leading to realistic synthesis.

In this chapter we outline our experiments (Section 5.1), which include altering the depth and number of hidden units in certain layers. We follow this up with the evaluation of these experiments (Section 5.2), mentioning why we think certain results performed better than others. The provided visualisations show sample 3, with 184 frames being predicted & 48 frames used as the seed, in each architecture.

### 5.1 Experiments

We train the following networks in the same manner as Chapter 4. We experiment with two different depths of LSTM layers, and a differing number of units in each layer.

First, we provide a single experiment to accompany the experiments in Chapter 4, we investigate altering the number of hidden units within the encoder (E) to 128: E(128)-LSTM(256)-LSTM(512)-D(256). We can not alter the decoders hidden units since our output representation needs to be of size 256 & we do not use Dropout in this experiment.

Also, we investigate two different networks, one with 2 LSTMs both having 256 units, and one with a E(256)-LSTM(256)-LSTM(512)-D(256) (Figure A.5). We investigate these models with & without the addition of Dropout, with probability 0.1.

## 5.2 Evaluation

When using 1 LSTM layer with a smaller encoder we find that the representation goes into a mean pose, compared to using 256 hidden units in the encoder<sup>1</sup>; we believe this is because the representation is encoded too much. The quantitative results (Figure 5.1) show that there is weak correlation between the error and visualisation, since the worse visualisation (E(128) network) has a smaller RMSE. By using 256 hidden units we see that the problem is that it does not follow the control signal very well. To challenge this we stack more layers as deeper models can encode a more complex structure.

We see that adding another layer helps with trajectory issues<sup>2</sup>, however it also seems to over-fit a walking motion. To counteract that we dropped units with 0.1 probability (Chapter 4). This allowed the unit to not over-fit however it still wished to turn around; we can see that with 2 LSTM layers, consisting of 256 hidden units, the motion resists turning around longer. We also show, yet again, that the quantitative results do not allow for a proper evaluation (Figure 5.1), since the error difference is relatively small when compared to the visual differences.

Table 5.1: **Root Mean Squared Error (RMSE)** between the test set and the ground truth (real spaced data).

Model	RMSE
LSTM(256)-LSTM(512)-Dropout(0.1)	1.5
LSTM(256)-LSTM(256)-Dropout(0.1)	1.56
LSTM(256)-LSTM(256)	1.73
E(256)-LSTM(256)	1.81
E(128)-LSTM(256)	1.58
Baseline	1.68

## 5.3 Discussion

In this section we discuss how the LSTM(256)-LSTM(256)-Dropout(0.1) ERD network compares to the baseline when generating motion; we notice the quantitative

<sup>1</sup><https://youtu.be/8uIq506h0KY>

<sup>2</sup>[https://youtu.be/2Y7F\\_OzcJqo](https://youtu.be/2Y7F_OzcJqo)

result for the baseline is worse when compared to our network (Table 5.1).

However, when we visualise the two networks with the samples described in Section 3, using 34 frames as seed motion and predicting 192 frames, we notice that the difference is not that large. <sup>3</sup>

The baseline performs well in predicting the motions trajectory, however the motion looks robotic and skips frames, whereas the ERD manages to copy the ground truth motion a majority of the time, with human like motion. Although it should be noted that it also suffers from going to the mean pose after a longer period of time. We suggest using a control signal to fix the mean posing in Chapter 6.

---

<sup>3</sup><https://youtu.be/YKdqSwnG9z0>



# Chapter 6

## Synthesis of Human Locomotion

In this chapter we experiment with the addition of control signals. We decided to add control signals as Holden et al [50] found them to help in offline synthesis, it is also a natural step since a gamer will always want the ability to alter the character’s motion in real-time.

We explain the experiments we performed in order to decide how many future control signals to use as input; then, we explain how we experimented with different network architectures, to see if the addition of a control signal altered any previous effects we have observed. We also try to mitigate some issues we observe by adding skip connections & masked noise to our best networks (Section 6.1).

We finish by explaining our results, and some of our decisions when the data was inconclusive (Section 6.2), before comparing our best architectures to the baseline and showing we achieve state of the art results for online motion generation (Section 6.2.6).

### 6.1 Experiments

In this section, we start by experimenting with numerous network depths & widths, before researching different types of control signal. Finally testing if previously noted methods in Section 2.2.1, such as skip connections and gradually adding masked noise, improve results.

We start our experiments by using just the initial control signal from the following latent variable, since each variable is composed of numerous frames in the real space; hence there are a lot of available signals we could have chosen. We make use of the ratio between latent variables and real frames, 1 to 8, as a heuristic from which to get a control signal. We train like explained in Chapter 4.

### 6.1.1 LSTM Width & Depth Experiments

Initially, we experiment with altering the width of the single LSTM cell to be 256 or 512 units.

Next, we try every combination of 2 LSTM layers with either 256 or 512 units. We add 0.1 Dropout to the final LSTM layer of 256 units, and 0.2 Dropout to the final LSTM layer of 512 units because there are more units; we found these values to perform best through experimentation.

Finally, we experiment with 3 layers; with the addition of a third layer, the complexity that the network can model is increased, therefore we also increase the value of Dropout to 0.2. We found this value gave the best results through experimentation; this value is also proposed by Pham et al. [61]. For the 3 LSTM layers, we made the 1<sup>st</sup> LSTM layer contain 256 units, and then altered the remaining 2 between every combination of 256 & 512 units.

### 6.1.2 Type of Control Signal

We experimented with altering the control signal given to a network consisting of 3 LSTM layers with 256 units. We can alter the control signal since a single frame in the latent space represents numerous frames in the real space, thus giving us a lot of different signals with which we can experiment. To make it easier to decide which signals to use, since frames overlap in the latent variables, we use the ratio of latent variables to the number of input frames; thus we represent 1 latent variable with 8 real frames. We experiment with the signals below, choosing these four as they capture lots of variations and allow some data to be inferred.

1. First - We take the control signal from the first frame in the subsequent 8 real frames.
2. All - We use all of the subsequent frames' control signals.
3. First Halfway Last - We use the first, halfway and final control signal of the following 8 real frames.
4. Alternate (1<sup>st</sup>, 3<sup>rd</sup>, etc...) - We use alternating control signals from the subsequent 8 frames.

We take these control signal vectors from time  $t + 1$  and concatenate them with the latent space vectors at time  $t$ . At test time, the control signals are from the ground truth

frame, since the signals are supposed to be externally input; i.e. from the user. We also experimented with placing them into the recurrent layers directly, however this did not help the system.

### 6.1.3 Additional Network Alterations

In [Section 2.2](#), we note Jain et al. [12] made use of skip connections and Fragkiadaki et al. [11] made use of masked noise - both state that it helped their models. We therefore experiment with these additions to our model using 3 LSTM layers with 256 units and the first control signal of the subsequent frame. We experiment with every combination of network.

Since the skip connections drastically increase the size of the vector going into the final decoding layer, we experiment with both a single layer decoder and a two layer decoder, keeping the decoder layers with 256 units.

Also, we removed the Dropout layer, since the noise should circumvent the need for it, and we do not believe it to be beneficial when using skip connections.

Finally, we trained the network for 1200 epochs (increasing the noise variation every 300 epochs) or, for 300 epochs depending if it had noise or skip connections (without noise).

## 6.2 Evaluation

Here we evaluate the results gathered from the above experiments; we do not show all visualisations, instead we show the most important ones and mention those that did not perform to our satisfaction. We note how differing the number of seed frames affects the outputs, however we do not show the visualisations. For the visualisations we do show, they all start with seed motion cut from the visualisation, to allow for easier evaluation of the synthesis.

We start by evaluating the different depths & widths we experimented with ([Section 6.2.1 to 6.2.3](#)), giving abstract observations before we give in-depth evaluations upon the different type of control signal we could use ([Section 6.2.4](#)) & additional network alterations ([Section 6.2.5](#)). Finally we evaluate our two best performing networks with respect to the baseline's performance, showing our state of the art results ([Section 6.2.6](#)).

Table 6.1: **Root Mean Squared Error (RMSE)** between the test set<sup>1</sup> and the ground truth (real spaced data). We use 16 frames as seed frames, synthesising 224 frames.

Model	RMSE
Baseline	1.68
Baseline with Control Signal	0.98
LSTM(256)	1.44
LSTM(512)	1.44
LSTM(256)-LSTM(512)-Dropout(0.1)	1.38
LSTM(256)-LSTM(256)-Dropout(0.1)	1.40
LSTM(512)-LSTM(256)-Dropout(0.1)	1.41
LSTM(512)-LSTM(512)-Dropout(0.2)	1.36
LSTM(256)-LSTM(256)-LSTM(256)-Dropout(0.2)	1.39
LSTM(256)-LSTM(256)-LSTM(512)-Dropout(0.2)	1.45
LSTM(256)-LSTM(512)-LSTM(512)-Dropout(0.2)	1.39
LSTM(256)-LSTM(256)-LSTM(256)-Dropout(0.2) with All Signal	1.44
LSTM(256)-LSTM(256)-LSTM(256)-Dropout(0.2) with Alt. Signal	1.57
LSTM(256)-LSTM(256)-LSTM(256)-Dropout(0.2) with First-Half-Last Signal	1.37
LSTM(256)-LSTM(256)-LSTM(256)-Dropout(0.2) with Skip Connections	1.26

### 6.2.1 1 LSTM Layer

In this section, we show the outcomes of changing the width of the LSTM layer. List of Footnote' entries. footnote/counter. <sup>2</sup>. We discover that by having 512 units the network can escape the mean pose (local minima) easier, however it still suffers from averaging and the drifting. This suggests that the network needs more layers to model the complexity, rather than a larger manifold, which is the effect of having 512 units.

We can see from [Table 6.1](#) that both single LSTM layer models perform equally in terms of quantitative analysis.

We observed that by using the seed frames mentioned in [Chapter 3](#), more seed frames provided better generations; however, we sometimes saw that these also took the network into local minima. We believe this is due to the hidden representation entering a state (local minima) it has not encountered before, therefore it can not escape

<sup>1</sup>Non-baseline models may have accumulated errors from the auto-encoder (which we did not train specifically on the locomotion data), since we are comparing in the visual space after the data has been decoded.

<sup>2</sup><https://youtu.be/us2kyr4Pdh0>

that minima. However, this is not easily proven.

### 6.2.2 2 LSTM Layers

In this section, we show the outcomes of differing the number of units within the layers'List of Footnote' entries. footnote/counter. <sup>3</sup>. We observe that the LSTM(256)-LSTM(256) network performs consistently to a good standard, whilst the LSTM(512)-LSTM(512) performs excellent on some samples and average on others. The other variations consistently performed average.

Also, we observe that the quantitative results show minimal difference, reinforcing our idea they are show no strong correlation with the visualised results.

For other seed values we observed the same as in [Section 6.2.1](#).

### 6.2.3 3 LSTM Layers

In this section, we evaluate the use of 3 LSTM layers in our network, we visualise every combination but LSTM(256)-LSTM(512)-LSTM(256), as it performs extremely similarly to LSTM(256)-LSTM(256)-LSTM(512)'List of Footnote' entries. footnote/counter. <sup>4</sup>.

We observe that 3 layers of LSTM(256) perform consistently better than any other network, whilst every network has a sample it performs best upon. However, we judge consistency to be a better metric, as a gamer can correct small errors in the control signal by trying small alterations. For other seed values we observed the same as in [Section 6.2.1](#).

We mention again that the quantitative error ([Table 6.1](#)) gives no obvious insight into how the synthesised motion will look.

### 6.2.4 Type of Control Signal

Here, we only visualise the network containing 3 LSTM layers which contain 256 units'List of Footnote' entries. footnote/counter. <sup>5</sup>. We give in-depth comments upon this visualisation whilst giving an overall summary on the 1 and 2 LSTM layers we also experimented with, which had 256 hidden units.

---

<sup>3</sup><https://youtu.be/FWeOH8agDwQ>

<sup>4</sup><https://youtu.be/eLDQ2xXybGE>

<sup>5</sup><https://youtu.be/aRguTwGHzUY>

Overall, we noticed using 1 LSTM layer with multiple control signals led to the network not being able to learn the proper synthesis function. In contrast, we saw when using 2 LSTM layers, with all the signal or only one signal, the network produced some impressive results.

For 3 LSTM layers, we noticed that giving alternate signals allowed the motion to mimic the root at times; for example, in samples 2, 8 & 10, it generalises to realise the correct trajectory. It does however go to the mean pose in certain samples, i.e. Sample 1, and is slower in terms of velocity when compared to the root motion. We also noticed that giving it the First-Half-Last (FHL) signal results in some mean poses and inconsistent synthesis; giving it one signal or all of the signal allows it to mimic the reconstruction, however it does not generalise to the root. We suspect that when the network is given alternate signals it has to fill in the blank signals, allowing it to generalise. As can be seen in the video<sup>5</sup>, the inconsistency of the FHL signal seems weakly correlated with the quantitative error (Table 6.1).

With all other signals suffering from inconsistency issues, and animators preferably only wanting to input one signal, we place only the first subsequent signal into the network in other experiments.

### 6.2.5 Additional Network Alterations

In this section, we give an overall outcome of the alternate network architectures, seeing that adding the skip connections to the standard 3 deep ERD helped it track the control signal better. List of Footnote entries. footnote/counter. <sup>6</sup>.

We do not visualise the other results as they do not improve the network, we saw that noise does not help which we speculate is due to the input representation already being in a latent space representation. Also, having 2 decoding layers added no benefit to the network. We evaluate the skip connections further in Section 6.2.6.

In terms of the quantitative result (Table 6.1), we see that it is an improvement, suggesting the quantitative result is weakly correlated to the qualitative result.

Finally, we note that we tried inserting the control signal directly into the LSTM layers directly by concatenating it with the output of the encoder, however this resulted in scattered motion, which did not look like human motion.

---

<sup>6</sup><https://youtu.be/dg0BxaWjVgU>

## 6.2.6 Comparison to Baseline

Here, we give brief comparisons of the baseline method against two of our own networks, E(256)-LSTM(256)-LSTM(256)-LSTM(256)-Dropout(0.2)-D(256), with one having the addition of skip connections'List of Footnote' entries. [footnote/counter.](#) <sup>7</sup>. We also compare our best performing network to the baseline, side by side, both without the control signal'List of Footnote' entries. [footnote/counter.](#) <sup>8</sup> and with the control signal'List of Footnote' entries. [footnote/counter.](#) <sup>9</sup> (BWC). We finish by comparing our two best performing networks side by side'List of Footnote' entries. [footnote/counter.](#) <sup>10</sup>.

We first compare the quantitative results ([Table 6.1](#)) seeing that both of our methods improve drastically over the baseline without a control signal, however, when the baseline has access to the control signal it achieves the best results. We have seen in previous experiments ([Chapter 5](#)) that the quantitative error gives a simple view that the visualisation could potentially be better; we say potentially as we have seen the visualisation to perform worse than other models, yet have a lower RMSE.

Upon visualising<sup>7</sup> we see that the baseline emulates the poses quite well with the trajectory being off. Also, we see that at times the baseline seems robotic, with structured movements and joints latching onto other close motions with matching joints, before quickly switching back. This matches to Fragkiadaki et al.'s comments on their ERD interpolating between motions. We see this quite clearly in the 1<sup>st</sup> Sample'List of Footnote' entries. [footnote/counter.](#) <sup>11</sup>, where it experiences a joint jump in the first few seconds, before interpolating to a side stepping motion, then to a walking on the spot motion, before finishing with human-like motion. We fixed the trajectory issues by giving the baseline access to the control signal also'List of Footnote' entries. [footnote/counter.](#) <sup>12</sup>, seeing that the trajectory issues have been mitigated, but the motion is still not smooth. We believe the motion could be smoothed with a more in-depth investigation into this method, which we describe further in [Chapter 7](#).

When we visualise our network without skip connections<sup>10</sup> (which from now on we denote as ON1), it manages to synthesise realistic motion much like the reconstruction data. At times it is a little slow reacting to the signal, i.e. Sample 1, however this could

---

<sup>7</sup>[https://youtu.be/yp\\_Yo92mFYc](https://youtu.be/yp_Yo92mFYc)

<sup>8</sup><https://youtu.be/Ps-885uNaEQ>

<sup>9</sup>[https://youtu.be/fb\\_drzzq9YI](https://youtu.be/fb_drzzq9YI)

<sup>10</sup><https://youtu.be/kCehLYqM7Bg>

<sup>11</sup>[https://youtu.be/yp\\_Yo92mFYc?t=14s](https://youtu.be/yp_Yo92mFYc?t=14s)

<sup>12</sup><https://youtu.be/hmktSK-6Lg0>

be due to the rather small training set. The motion produced by ON1 never mean poses, which we believe is due to the control signal keeping it out of local minima by directing the signals, in the network, towards other motions. Also, we see that ON1 has smoother motion compared to the BWC<sup>9</sup>, we believe this is due to ourselves splitting up the task of learning the manifold and synthesis. Whereas our baseline model tries performing both tasks at once, which makes it hard to find the motion manifold effectively, since the noisy inputs could overlap with other motions; this is due to the similarity between motions. Holden et al. [1] found that the similarities in the CMU dataset are sometimes strange, such as an old-man walking style being close to a side-stepping motion.

However we noticed that the signal was still slow and we believe this was due to the control signal being degraded throughout the network. The addition of skip connections to the network, which we now denote as ON2, allowed the network to learn how to decode all the outputs into something very similar to the ground truth 'List of Footnote' entries. footnote/counter. <sup>13</sup> (the reconstruction data); for example, Sample 1. ON2 managed to follow the trajectory and style effectively, however we noticed the velocity of the movements were seemingly smaller as strides seemed shorter. We also notice that in Sample 5 with the ON2 network the motion drifts slightly whilst turning abruptly. This can be fixed by constraints upon the motion generation as shown by Holden et al. [50], however we leave these discussions till Chapter 7.

We conclude this section by stating that our work has managed to effectively generate novel motion, which correctly follows the style and trajectory of the input character based upon the low dimensional signal. We show a novel methodology by allowing the LSTM to learn within the latent space of the motion, and by using an external control signal. We provide a visualisation of our final network in Figure 6.1.

---

<sup>13</sup><https://youtu.be/kCehLYqM7Bg>

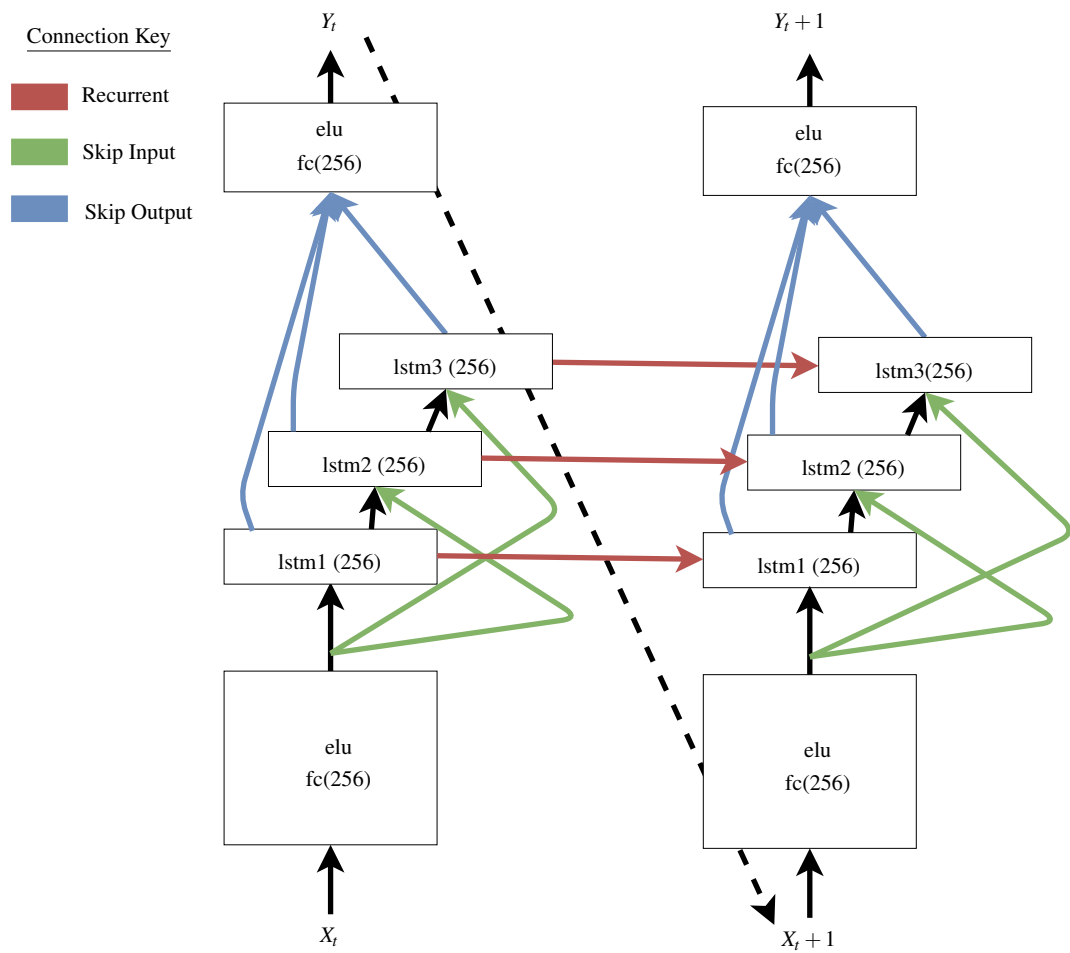


Figure 6.1: Final ERD Network with 3 LSTM Layers and Skip Connections (ON2)



# Chapter 7

## Discussion, Conclusion & Future Work

### 7.1 Discussion

In the Introduction ([Chapter 1](#)), we hypothesised that neural networks could be used to synthesise realistic human-like locomotion. Also, we mentioned the issues which have to be overcome in any possible solution. Our research has shown that neural networks are successful in being able to synthesise human-like locomotion, and it is possible to circumvent the issues mentioned; by using a control signal and working within the latent variables.

We show that by using a E(256)-3xLSTM(256)-D(256) network with skip connections on the latent variable data we can generate realistic human-like locomotion; avoiding the mean pose by providing a control signal from future steps. We believe the mean pose is actually local minima on the manifold, and that by providing the network with a control signal, it pushes the network away from these minima, therefore circumventing issues experienced in previous research ([Chapter 2](#)).

We have experienced the control signal being slow, which we believe is due to the auto-encoder convolving over many frames, both future and past. Therefore, the control signal we input into the network might be weakly correlated with the actual data, due to many large convolutions when producing the latent representation. If this is the case, it might take a few control signals being input to the network to retrieve a good control signal. We saw when experimenting with different signals ([Chapter 6](#)), the alternative signals allowed the network to generalise extremely well when compared to the root. We believe that if the auto-encoder is altered to allow for clearly defined control signals to be extracted, then our methodology would perform a lot better (discussed further in [Section 7.3](#)). However these slow signals are not really a problem, as

the user would keep reinforcing it by sending the signal into the system again.

The final thing we discuss is the small amount of drift experienced by the network; this is not a large issue as it is common to fix minor issues like this after visualisation has taken place. In [Chapter 2](#) we explain how Holden et al. [50] fixed this issue, which we believe also applies to our system.

## 7.2 Conclusion

To conclude, we have shown that a novel methodology can be used to generate state of the art online human locomotion. Importantly, we have seen that this methodology needs a control signal to direct the motion and prevent the averaging issues.

## 7.3 Future Work

By showing this methodology to work, we have opened up a lot of potential further research. We describe these in two sections, the first describes different models we would like to research, the second describes further research with our current model.

### 7.3.1 Different Models

Whilst our method can produce online animation, the auto-encoder part also effects future frames and previous frames. It would be of interest to research a possibility which did not affect other frames, with the LSTM learning more of the temporal dependencies. One such example could be ensuring 30 frames, when convolved and pooled, end up producing one hidden latent variable, therefore the animator would input 1 control signal per second (presuming 30 frames per second). We noticed that the baseline (with a control signal) followed the trajectory and that the generated motion copied the ground truth's style. We believe with more investigation it would be possible for this to be an online solution. We do however note that it takes a lot longer to learn the synthesis function, and that having multiple tasks to learn might benefit this system; we could use reconstruction of the input and synthesis of the following frame as the tasks. We believe this might help due to observations by Jonathan Schwarz in our research group (work not published).

Another possibility would be using a sequence to sequence model, introduced by Sutskever et al. [63], since this model learns to map from a sequence to a sequence. We

could define a sequence as being 30 frames and input a control signal in some manner, this would have the benefit of learning the manifold and producing, online, 1 second of motion per prediction.

Also, we would like to see Jain et al.'s work [12] being extended to include a control signal, since their network can also reproduce motion longer than the ERD of Fragkiadaki et al. [11].

The final possibility would be using a Variational RNN (V-RNN)[64; 65]. A V-RNN learns a probabilistic latent representation,  $p(z|x)$ , and then generates from that latent space,  $p(x|z)$ , by sampling from the distribution  $z$ ; therefore it learns the true probabilistic distribution of the data. If the generation is noisy then it could be placed through a generative adversarial network (GAN) [66] like Larson et al. [67].

### 7.3.2 Animation Research

With the rise of VR there is going to be a lot more data corresponding to realistic situations, it would be of interest to see the effects of using this data to learn the mapping from in-place local motion (i.e. walking on the spot) to the global space of the game. This could also be placed with hardware to extract further signals, such as foot contacts, to correctly synthesise the motion performed.

In addition to this, it would be of interest to see our model trained upon the full CMU dataset, since Holden et al. [50] mention their network needs to be specifically trained upon certain motion sets. We hypothesise our model would be capable of learning multiple different variations due to it working in a latent space which can clearly recreate all of these variations. Also, it would be interesting to see batch normalisation used with this technique since the batches should be larger & more varied, hence they should capture the correct statistics.

Finally, it would be intriguing to use higher level control signals, like Holden et al. [50], which would allow more flexibility when controlling the learnt motion. Therefore, allowing the user (i.e. animator or gamer) more control over the generation.



# Appendix A

## Extra Network Images

In this appendix we provide extra diagrams for the reader, so that if any explanation confuses them, they can refer to the visual representation.

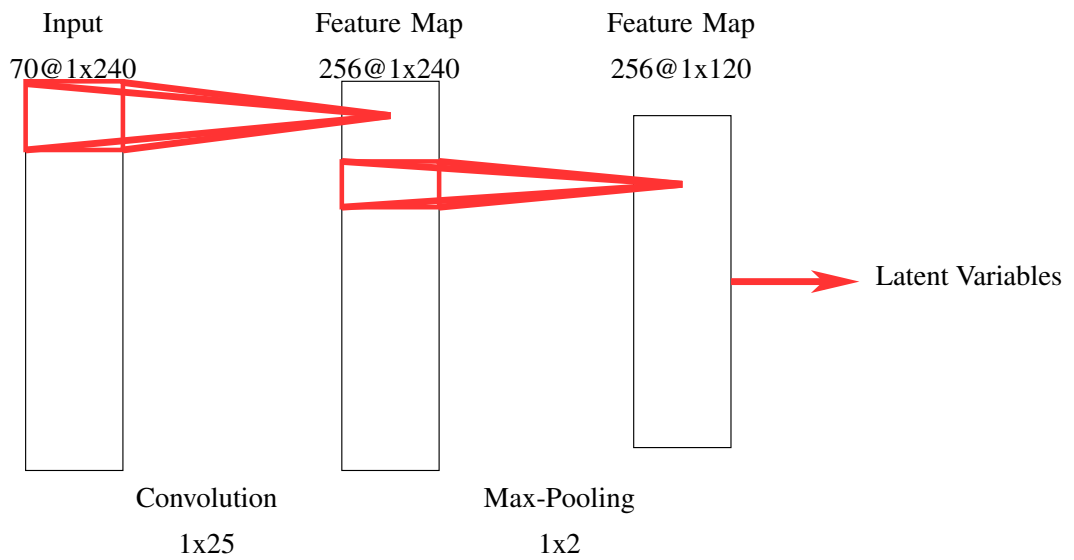


Figure A.1: Auto-Encoder Used

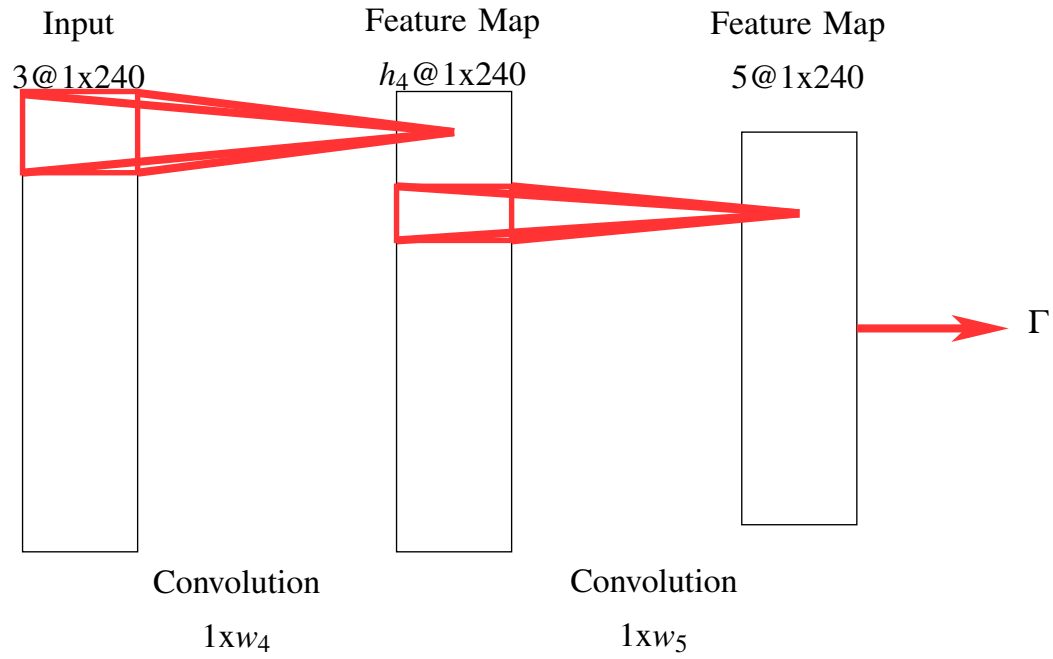


Figure A.2: Foot Timings Convolution

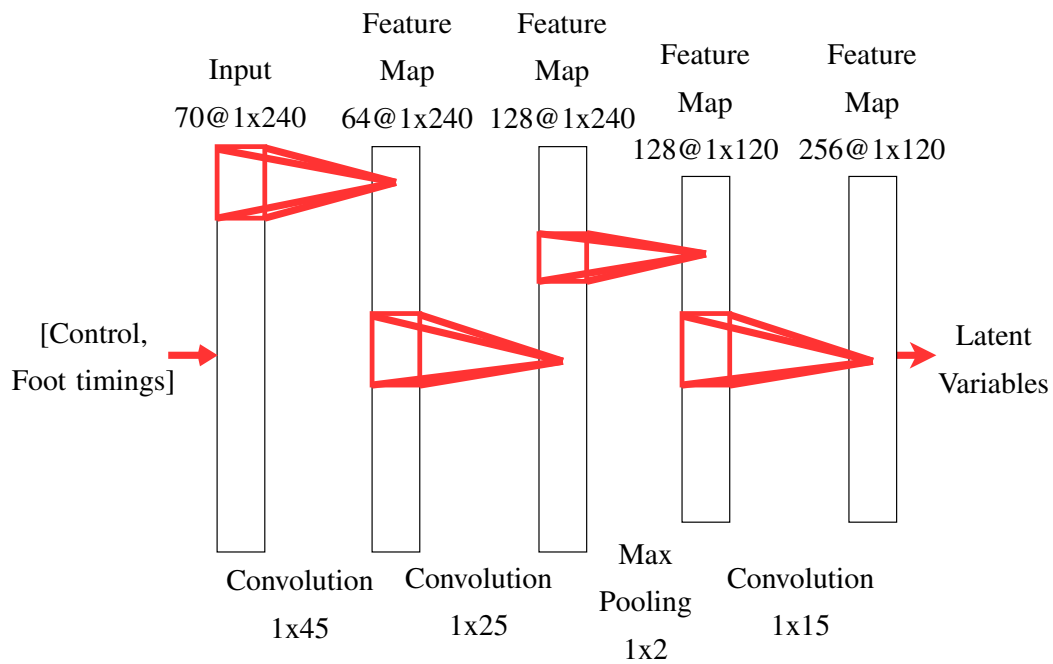


Figure A.3: Convolutional Feed Forward Network

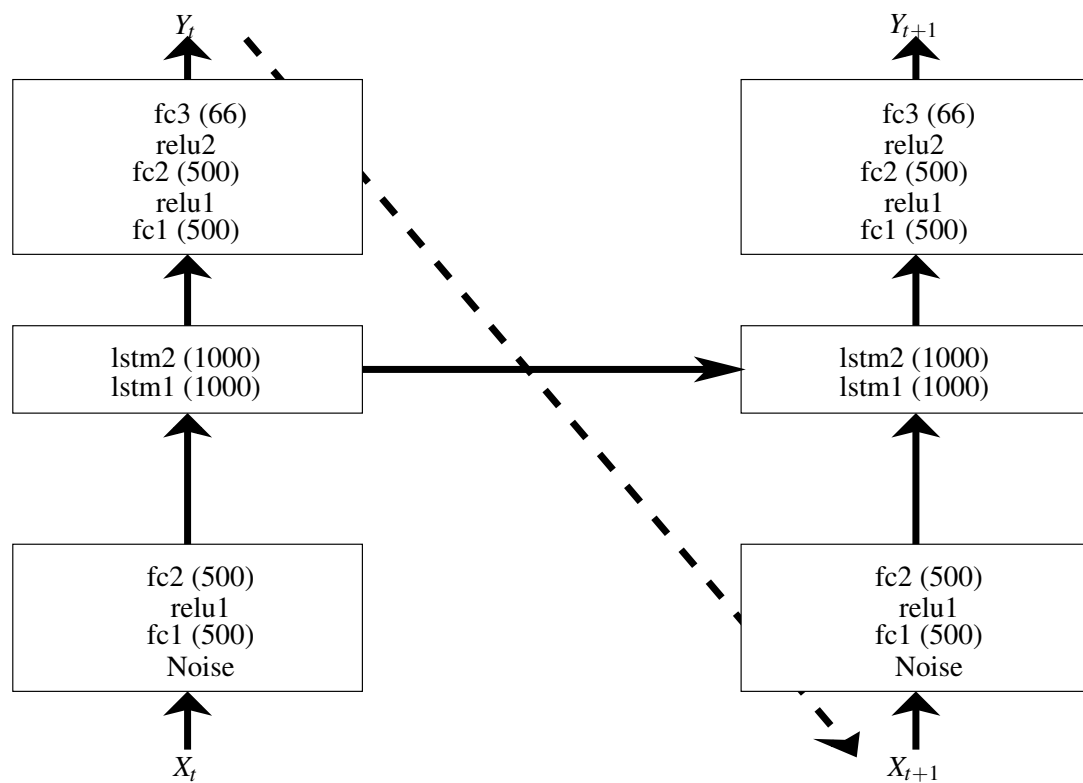


Figure A.4: Altered ERD [11]

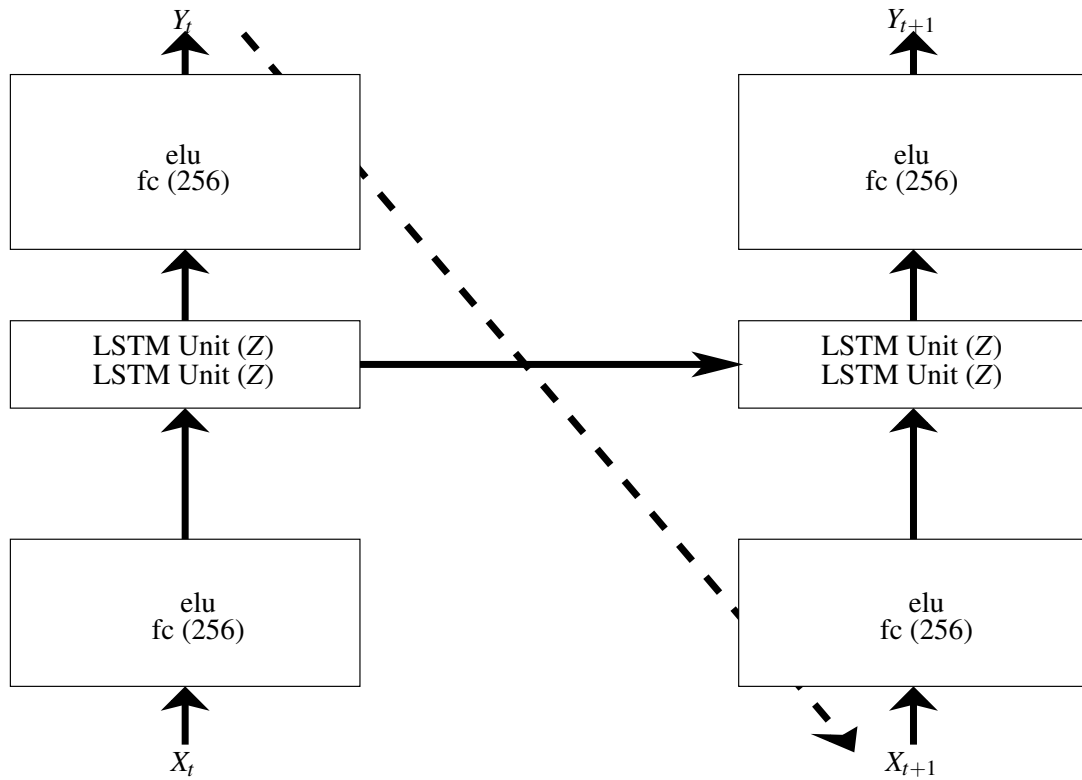


Figure A.5: Network experimented with,  $Z = [\{256,512\},\{256,256\}]$

# Bibliography

- [1] D. Holden, J. Saito, T. Komura, and T. Joyce, “Learning motion manifolds with convolutional autoencoders,” in *SIGGRAPH Asia 2015 Technical Briefs*. ACM, 2015, p. 18.
- [2] “Oculus rift,” 2015. [Online]. Available: <http://www.oculusvr.com/rift>
- [3] “Play station virtual reality,” 2015. [Online]. Available: <https://www.playstation.com/en-gb/explore/playstation-vr/>
- [4] “The void,” 2016. [Online]. Available: <https://thevoid.com>
- [5] “Virtual reality shopping mall,” 2016. [Online]. Available: <http://www.the-virtualmall.com/zamma/>
- [6] M. Lorenz, M. Busch, L. Rentzos, M. Tscheligi, P. Klimant, and P. Frhlich, “I’m there! the influence of virtual reality and mixed reality environments combined with two different navigation methods on presence,” in *Virtual Reality (VR), 2015 IEEE*, March 2015, pp. 223–224.
- [7] I. Millington, *Game physics engine development*. Morgan Kaufmann Publishers Amsterdam, 2007.
- [8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [9] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [10] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: <http://arxiv.org/abs/1409.1259>

- [11] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, “Recurrent network models for human dynamics,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4346–4354.
- [12] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, “Structural-rnn: Deep learning on spatio-temporal graphs,” *arXiv preprint arXiv:1511.05298*, 2015.
- [13] Y. LeCun, “A Theoretical Framework for Back-Propagation,” 1988.
- [14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, oct 1986. [Online]. Available: <http://www.nature.com/doi/10.1038/323533a0>
- [15] P. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58337>
- [16] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 396–404. [Online]. Available: <http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf>
- [17] S. Hochreiter and J. Jürgen Schmidhuber, “LONG SHORT-TERM MEMORY,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: <http://www7.informatik.tu-muenchen.de/~hochreithttp://www.idsia.ch/~juergen>
- [18] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex.” *The Journal of physiology*, vol. 160, no. 1, pp. 106–54, jan 1962. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pubmed/14449617http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC1359523>
- [19] Y. LeCun and Y. Bengio, “Convolutional networks for images, speech, and time series,” in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1998, ch. Convolutional Networks

- for Images, Speech, and Time Series, pp. 255–258. [Online]. Available: <http://dl.acm.org/citation.cfm?id=303568.303704>
- [20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [22] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” *CoRR*, dec 2013. [Online]. Available: <http://arxiv.org/abs/1312.6229>
- [23] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [24] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning Spatiotemporal Features with 3D Convolutional Networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, dec 2015, pp. 4489–4497. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7410867>
- [25] D. Haussler, “Convolution kernels on discrete structures,” Citeseer, Tech. Rep., 1999.
- [26] “LeNet.” [Online]. Available: <http://deeplearning.net/tutorial/lenet.html>
- [27] “CS231n - CNN.” [Online]. Available: <http://cs231n.github.io/convolutional-networks/>
- [28] “draw convolutional network.” [Online]. Available: [https://github.com/gwding/draw\\_convnet](https://github.com/gwding/draw_convnet)
- [29] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” *Journal of Machine Learning Research*, 2011.

- [30] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, may 2015. [Online]. Available: <http://www.nature.com/doifinder/10.1038/nature14539>
- [31] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: <http://arxiv.org/abs/1502.01852>
- [32] D. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus),” *CoRR*, vol. abs/1511.07289, 2015. [Online]. Available: <http://arxiv.org/abs/1511.07289>
- [33] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [34] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [35] R. S. Zemel, “A minimum description length framework for unsupervised learning,” Ph.D. dissertation, University of Toronto, 1993.
- [36] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [37] V. Jain and S. Seung, “Natural image denoising with convolutional networks,” in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2009, pp. 769–776. [Online]. Available: <http://papers.nips.cc/paper/3506-natural-image-denoising-with-convolutional-networks.pdf>
- [38] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion,” *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.
- [39] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen netzen,” *Diploma, Technische Universität München*, p. 91, 1991.

- [40] C. Olah, “Understanding lstm networks,” 2016. [Online]. Available: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png>
- [41] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, “Learning precise timing with lstm recurrent networks,” *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.
- [42] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [43] A. Graves, “Generating sequences with recurrent neural networks,” *arXiv preprint arXiv:1308.0850*, 2013.
- [44] N. Kalchbrenner, I. Danihelka, and A. Graves, “Grid long short-term memory,” *arXiv preprint arXiv:1507.01526*, 2015.
- [45] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.
- [46] J. Chai and J. K. Hodgins, “Performance animation from low-dimensional control signals,” in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 686–696.
- [47] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, “Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, jul 2014.
- [48] “Structural RNN Code.” [Online]. Available: <https://github.com/asheshjain399/RNNexp/tree/srnn>
- [49] G. W. Taylor, G. E. Hinton, and S. T. Roweis, “Modeling human motion using binary latent variables,” in *Advances in neural information processing systems*, 2006, pp. 1345–1352.
- [50] D. Holden, J. Saito, and T. Komura, “A deep learning framework for character motion synthesis and editing,” *ACM Trans. Graph.*, vol. 35, no. 4, pp. 138:1–138:11, Jul. 2016. [Online]. Available: <http://doi.acm.org/10.1145/2897824.2925975>

- [51] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [52] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.
- [53] T. Mukai and S. Kuriyama, “Geostatistical motion interpolation,” in *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3. ACM, 2005, pp. 1062–1070.
- [54] C. F. Rose III, P.-P. J. Sloan, and M. F. Cohen, “Artist-directed inverse-kinematics using radial basis function interpolation,” in *Computer Graphics Forum*, vol. 20, no. 3. Wiley Online Library, 2001, pp. 239–250.
- [55] CMU, “Cmu motion capture database.” [Online]. Available: <http://mocap.cs.cmu.edu>
- [56] T. Dozat, “Incorporating Nesterov Momentum into Adam,” Stanford, Tech. Rep., 2015.
- [57] F. Chollet, “Keras,” 2015. [Online]. Available: <https://github.com/fchollet/keras>
- [58] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Blecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries,

- D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: <http://arxiv.org/abs/1605.02688>
- [59] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks.” in *Aistats*, vol. 9, 2010, pp. 249–256.
- [60] Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.
- [61] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, “Dropout improves recurrent neural networks for handwriting recognition,” in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014, pp. 285–290.
- [62] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in *Advances in neural information processing systems*, 2012, pp. 2951–2959.
- [63] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
- [64] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, “A recurrent latent variable model for sequential data,” in *Advances in neural information processing systems*, 2015, pp. 2980–2988.
- [65] O. Fabius and J. R. van Amersfoort, “Variational recurrent auto-encoders,” *arXiv preprint arXiv:1412.6581*, 2014.
- [66] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.
- [67] A. B. L. Larsen, S. K. Sønderby, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” *arXiv preprint arXiv:1512.09300*, 2015.