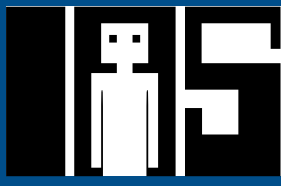
# Learning Action Representations For Primitives-Based Motion Generation

**Lernen Von Aktionsdarstellungen Für Primitivenbasierte Bewegungsgenerierung**
Bachelor thesis by Mark Baierl
Date of submission: September 30, 2020

1. Review: Prof. Dr. Jan Peters
2. Review: Vignesh Prasad
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



## Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Mark Baierl, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 30. September 2020

M. Baierl

# Contents


**1. Introduction** **5**

**2. Foundations And Related Work** **7**
2.1. Autoencoders . . . . . 7
2.2. Variational Autoencoders . . . . . 7
2.3. Recurrent Neural Networks . . . . . 9
2.4. Human Motion Prediction With RNNs . . . . . 11
2.5. Recurrent Latent Models For Sequential Data . . . . . 11
2.6. Probabilistic Movement Primitives . . . . . 12
2.7. AE-ProMP . . . . . 15

**3. Approach** **16**

**4. Experiments** **19**
4.1. The NTU RGB-D Dataset . . . . . 19
4.2. Data Preprocessing . . . . . 19
4.3. Autoencoders . . . . . 21
4.4. Movement Prediction . . . . . 27
4.5. Motion Generation . . . . . 43

**5. Discussion** **50**

**6. Future Work** **52**
6.1. Generative Adversarial Nets . . . . . 52
6.2. Graph Neural Networks . . . . . 52
6.3. Nonlinear Principal Component Analysis . . . . . 53

**A. Appendix** **56**

# Abstract

It is still a bit difficult to teach robots how to perform new tasks, usually involving experts programming the robot manually, or by kinesthetic teaching. This can become tedious as the number of tasks increases. If robots could learn directly from human demonstrations it would be a lot easier to teach robots new tasks. We leverage latent space models, namely a combination of *Autoencoders* (AEs) and *Probabilistic Movement Primitives* (ProMPs), to generate motion from skeletal data of humans' demonstrations. Using this approach it's possible to first learn an action representation and then generate movements corresponding to that action.

# Zusammenfassung

Es ist noch immer schwierig Robotern beizubringen neue Aufgaben zu erledigen. Sie müssen entweder von Experten manuell programmiert werden, oder kinästhetisch trainiert werden. Das kann sehr umständlich sein bei steigender Anzahl an Aufgaben. Wenn Roboter direkt von menschlichen Demonstrationen lernen könnten, wäre es viel einfacher Robotern neue Aufgaben beizubringen. Wir nutzen *Probabilistic Movement Primitives* (ProMPs) in latenten Räumen die durch *Autoencoder* (AE) erzeugt werden. Durch diesen Ansatz ist es möglich zuerst eine Aktionsdarstellung zu lernen und dann Bewegungen entsprechend dieser Aktion zu generieren.

# Acknowledgments

I'd like to take a moment to thank my supervisor Vignesh Prasad, without whom this thesis would not have been possible. He was always patient, open for questions, and supported me in any we he could. He gave really useful feedback, leading me to learn a lot while writing this thesis. At times, he spend huge amounts of time to help me fix problems I would not have been able to figure out on my own. So thank you very much man.

# 1. Introduction

There are lots of ways to teach a robot to perform a task. Many are cumbersome and require a lot of manual labor. Often movements are learned by kinesthetic teaching. However, this is not always possible i.e. pneumatically controlled robots. This is also time-consuming, is robot specific, and doesn't allow for great amounts of generalization. In this thesis, we look at ways to learn action representations and generate movements corresponding to that action.

In figure 1.1 the proposed method of learning and generating motions, while also leveraging latent space models, is visualized. We make heavy use of *Autoencoders* (AEs) [1] in combination with *Probabilistic Movement Primitives* (ProMPs) [2]. Autoencoders encode data into a so-called latent space. We use both together to learn action representations from demonstrations in a latent space smaller than the original skeleton-joint space to increase computational efficiency [3]. We expand on [3] by looking at different models that may fill the role of the encoder and decoder networks of the AEs. The main focus here lays in recurrent models, how well they work for both encoding and decoding trajectories, and how well they work when used with ProMPs. This hopefully enables further work

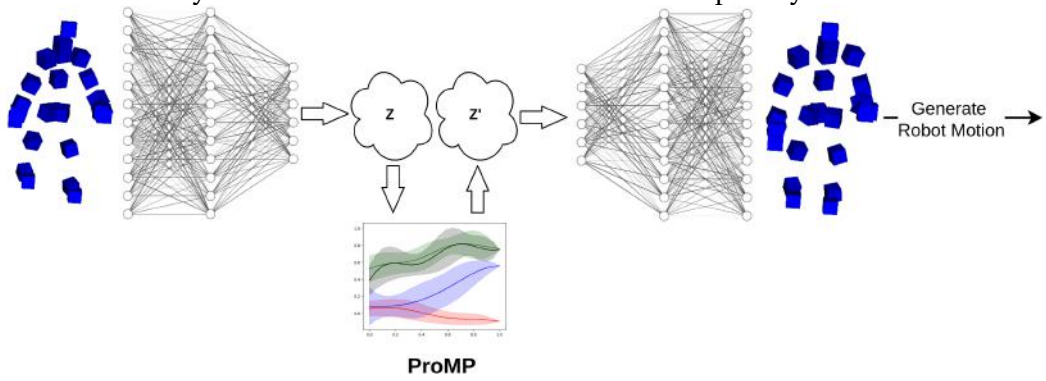



Figure 1.1.: Visualization of the proposed method of generating motion using ProMPs while leveraging latent space models.

in time-critical environments. In the *Foundations And Related Work* chapter, we look into all of these in more detail and also introduce currently used techniques to generate motion. Afterward we conduct multiple experiments on how well different models work as encoders and decoders. We explore how well ProMPs can generate predictions in the various latent spaces of the AEs, and how well motions can be sampled, thus creating new movements, from the ProMPs in the latent spaces. In the discussion chapter, we look at what the experiments showed us and bring them into context with one another.

# 2. Foundations And Related Work

In this chapter, we look at technologies relevant to the work done here. This should give you a firm basis of what we explore in the experiments chapter while also mentioning similar approaches that try to achieve what we are.

## 2.1. Autoencoders

AEs were first introduced by Rumelhart et al. [1] n 1985 and have become more popular again through the rise of neural networks [4]. AEs try to learn the identity function in an unsupervised way using different kinds of neural networks. They usually consist of one encoder and one decoder network. The encoder encodes the inputs it's given into a latent space which the decoder takes and tries to reconstruct the original input from. Comparing how similar the generated values are to the originals allows training these two networks together to achieve a good reconstruction. Figure 2.1 visualizes what this would look like. These AEs are data-specific, meaning that an AE trained on images of trees, would perform poorly when trying to reconstruct images of faces.

## 2.2. Variational Autoencoders

*Variational Autoencoders* (VAEs) [5, 6] assume that the prior over the latent variables is given by a normal distribution $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$, while $p_\theta(\mathbf{x}|\mathbf{z})$ is given by a multivariate normal distribution (in the case of real-valued data). The assumption now is that the true

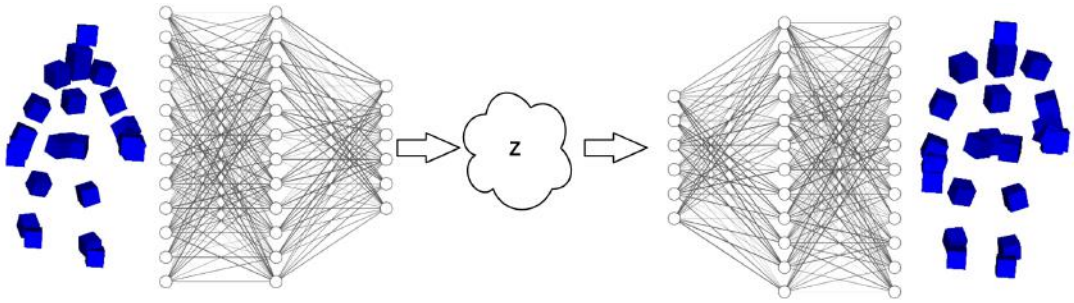



Figure 2.1.: Mockup of how an AE would encode and decode skeletal data. The skeleton on the left represents the input data given to the encoder network. It produces the latent representation z which then gets passed to the decoder network. The decoder then reconstructs the original skeleton with some amount of loss.

posterior is approximate of Gaussian form and thus the approximate variational posterior, a multivariate Gaussian, can be expressed by

$$log(q_\theta(\mathbf{z}|\mathbf{x}^{(i)})) = log(\mathcal{N}(\mathbf{z}; \boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\mathbf{I}). \tag{2.1}$$

Here $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are the outputs of a neural network. We use different kinds of neural networks throughout this thesis to approximate these parameters.

Now we can use the *Kullback-Leibler-Divergenz* (KL) to receive the estimator

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \approx \frac{1}{2}\sum_{j=1}^{J}(1 + log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L}\sum_{l=1}^{L} log(p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)}) \tag{2.2}$$

where $\boldsymbol{z}^{(i,l)} = \boldsymbol{\mu}^i + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}$ and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

What this means for us is that we can train a model to generate the mean and variance of the distribution over the latent space, rather than straight values. This way we can constraint how similar skeletons should look in the latent space by applying a KL Loss. This hopefully generates a more meaningful latent space in which similar values in the skeletal space map onto similar values in the latent space. Having a more meaningful latent space generated this way hopefully enables a better generation of motion in that latent space.

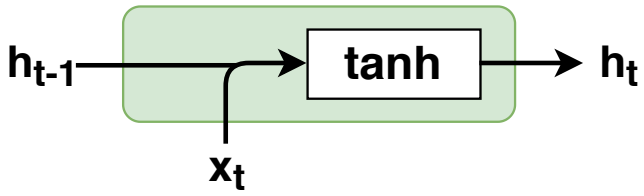



Figure 2.2.: Basic RNN where $h_t$ is the output at time $t$ and the hidden state at time $t+1$.

## 2.3. Recurrent Neural Networks

*Recurrent Neural Networks* (RNN) are a type of neural network containing some sort of recurrence. This means that information can be passed between iterations, giving the network a kind of memory. The basic RNN, we use for one of the AE architectures, looks like this:

$$\mathbf{h}_t = tanh(\mathbf{W}_{ih}\mathbf{x}_t + \mathbf{b}_{ih} + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_{hh}), \tag{2.3}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the weights and biases of the network. Assume this notation for future equations if not stated otherwise.

The recurrence in this simple network is expressed by the hidden state $h$. This hidden state $h_t$ is at time $t$ the output of the network as well as the hidden state at time $t+1$. This network is also visualized in figure 2.2. Many of these networks exist, but only a few are relevant to this thesis. We have a closer look at *Long Short Term Memory* networks now.

### 2.3.1. Long Short Term Memory

First introduced in [7, 8], *Long Short Term Memory* (LSTM) architectures have become a vital method and state of the art in many fields like handwriting generation or speech synthesis [9]. The main idea which separates the LSTM architecture from other RNN architectures is a memory cell that can maintain its state over time. In figure 2.3 you can see this extra memory cell state denoted as $c_t$. We use LSTMs as a baseline later on, as well as an architecture for the AE networks.

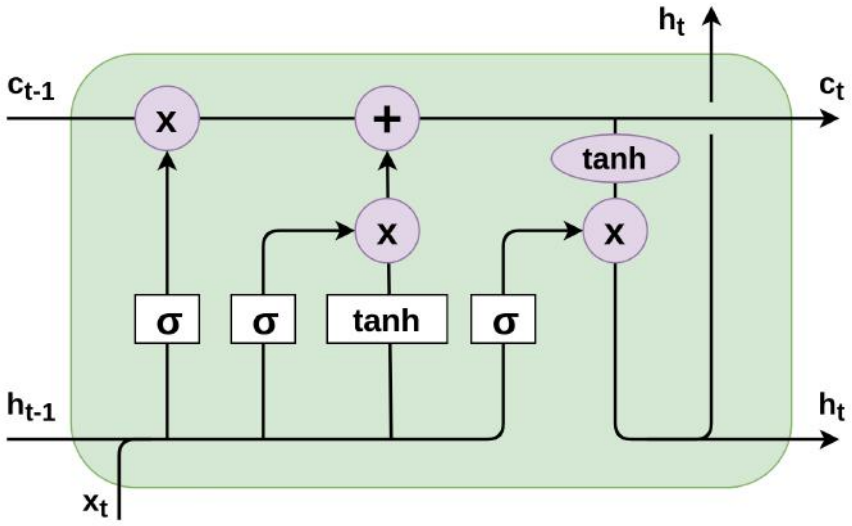



Figure 2.3.: One common variant of a *Long Short Term Memory* network. The input of this network is $x_t$ (and $c_{t-1}$, $h_{t-1}$) and the output is $h_t$. While $h_t$ represents short-term memory familiar from RNNs, $c_t$ represents the long term memory.

For both, the exact architecture,

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_{hi}) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{if}\mathbf{x}_t + \mathbf{b}_{if} + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_{hf}) \\
\mathbf{g}_t &= tanh(\mathbf{W}_{ig}\mathbf{x}_t + \mathbf{b}_{ig} + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_{hg}) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{io}\mathbf{x}_t + \mathbf{b}_{io} + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_{ho}) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t)
\end{aligned} \tag{2.4}$$

is a very typical LSTM also visualized in figure 2.3. Here $h_t$ is the hidden state and output at time $t$, while $c_t$ is the cell state at time $t$. The gates that influence the cell state $c_t$ are the forget ($f_t$), cell ($g_t$), and input gates ($i_t$). As the name would suggest, the forget gate decides which of the previous information to keep and which to forget. The input gate, in combination with the cell gate, adds new information to the cell state. The last step is to combine the output gate ($o_t$) with the cell state to get the new hidden state, as well as the output of the LSTM.

## 2.4. Human Motion Prediction With RNNs

A task related to generating motion is predicting motion. In [10] they use different kinds of RNNs to predict future movement given the past poses of a human body. There are two common complementary performance metrics used to evaluate how well a motion has been predicted leveraged by Martinez that we also make use of later on. *Quantitative* prediction error and *qualitative* motion synthesis. The former is typically measured using a simple mean squared error loss and is useful to measure short-term performance, for example in computer vision where predictions can be continually matched and corrected. The latter can be used for more extended movements, where the goal is to generate feasible motion. This qualitative metric is difficult to meaningfully measure quantitatively. One major problem with using RNN-based methods for predicting future movement is that there is a very noticeable discontinuity in the first few predictions. This can be very harmful when the short-term predictions are crucial. They introduce a *Gated Recurrent Unit* (GRU) architecture that is supposed to fix these shortcomings. A baseline used for comparison is a network called *LSTM-3LR* introduced in [11]. It is basically a three-layer LSTM sandwiched between some linear components. We also use this as a baseline, since the linear components can be replaced by any AE, allowing for better comparison.

## 2.5. Recurrent Latent Models For Sequential Data

In [12] they introduce a model called *Variational RNN* (VRNN). The main idea behind this network is to train a recurrent model alongside the AE to enable better encoding of sequence data. The output hidden state of that RNN is then fed as an additional input to the encoder and decoder. Although useful for many tasks, the issue when looking at it from the point of view of generating motion in the latent space to then decode it, is that the RNN is dependent on the original input provided. Given that there is no input data when generating directly in the latent space, this model isn't suitable for the task at hand.

The general idea of VRNNs, also implemented for example in *Variational Time Series Feature Extractor* (VTFSE) [13], is a useful one though.

One approach we explore in the chapters to come is using recurrent models rather than linear ones for the AEs. The same goes for the VAEs where the mean and variance of the distribution over the latent space get generated by an RNN.

## 2.6. Probabilistic Movement Primitives

*Probabilistic Movement Primitives* (ProMPs) [2, 14] are a probabilistic formulation of Movement Primitives(MPs) [15] that maintain a distribution over trajectories. One advantage of using ProMPs rather than MPs is that they can encode the variance of the movement and because of that, can often directly encode optimal behavior in stochastic systems. Other useful characteristics like temporal modulation and combination of distinct movements are also provided by ProMPs.

### 2.6.1. Encoding Of Trajectory Distribution

ProMPs leverage basis functions to decrease the number of model parameters [14]. A trajectory $\boldsymbol{\tau}$ is represented using a weight vector $\boldsymbol{w}$. The probability to observe $\boldsymbol{\tau}$ given $\boldsymbol{w}$ is given by

$$\boldsymbol{y}_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \boldsymbol{\Phi}_t \boldsymbol{w} + \boldsymbol{\epsilon}_y, \tag{2.5}$$

$$p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_t \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Phi}_t \boldsymbol{w}, \boldsymbol{\Sigma}_y), \tag{2.6}$$

with

$$\boldsymbol{\Phi}_t = \begin{bmatrix} \boldsymbol{\phi}_t \\ \dot{\boldsymbol{\phi}}_t \end{bmatrix}, \boldsymbol{\epsilon}_y \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_y), \tag{2.7}$$

where $n$ is the number of basis functions and $\boldsymbol{\Phi}_t$ is a $n \times 2$ dimensional time-dependent basis matrix for joint positions and velocities, $q_t$ and $\dot{q}_t$ respectively.

To get the trajectory distribution $p(\boldsymbol{\tau}; \boldsymbol{\theta})$ we can marginalize out the weight vector $\boldsymbol{w}$ using a distribution $p(\boldsymbol{w}; \boldsymbol{\theta})$,

$$p(\boldsymbol{\tau}; \boldsymbol{\theta}) = \int p(\boldsymbol{\tau}|\boldsymbol{w}) p(\boldsymbol{w}; \boldsymbol{\theta}) \mathrm{d}\boldsymbol{w}. \tag{2.8}$$

$p(\boldsymbol{\tau};\boldsymbol{\theta})$ is a hierarchical Bayesian Model where the parameters are given by the observation noise variance $\boldsymbol{\Sigma}_y$ and $\boldsymbol{\theta}$ of $p(\boldsymbol{w};\boldsymbol{\theta})$.

### 2.6.2. Temporal Modulation

Temporal modulation enables adjusting the execution speed of any movement generated by the ProMP. This is achieved by introducing a phase variable $z$ which decouples the movement from the time. The movements now range from $z_0 = 0$ to $z_T = 1$ where the velocity is usually constant with $\dot{z}_t = 1/T$. The basis function $\phi_t$ now depends on the phase instead of the time, such that

$$\phi_t = \phi(z_t),\ \dot{\phi}_t = \dot{\phi(z_t)}\dot{z}_t, \tag{2.9}$$

where $\dot{\phi}_t$ is the corresponding derivative.

The type of basis function $\phi$ to be used is dependent on what kind of motion is to be modeled. There are two main types of motions that use different basis functions. *Stroke and rhythmic-based* motions. For stroke-based movements, we use Gaussian basis functions $b_i^G$, and Von-Mises basis functions $b_i^{VM}$ for rhythmic movements,

$$b_i^G(z) = \exp\left(-\frac{(z_t - c_i)^2}{2h}\right),\ b_i^{VM}(z) = \exp\left(-\frac{cos(2\pi(z_t - c_i))}{h}\right), \tag{2.10}$$

where $c_i$ defines the center for the $i$th basis function and $h$ the width of the basis. Using one of these basis functions we get

$$\phi_i(z_t) = \frac{b_i(z)}{\sum_{j=1}^{n} b_j(z)}, \tag{2.11}$$

the normalized basis function to improve the regression's performance.

### 2.6.3. Encoding Coupling Between Joints

Many tasks require coordinating the movement of multiple joints. Let $\boldsymbol{w_i}$ be a parameter vector for dimension $i$. Then the combined weight vector $\boldsymbol{w}$ is $\boldsymbol{w} = \begin{bmatrix}\boldsymbol{w}_1^T & \cdots & \boldsymbol{w}_n^T\end{bmatrix}^T$. The

basis matrix $\boldsymbol{\Phi_t}$ now extends to a block-diagonal matrix with the basis functions and their derivatives for each dimension. Now $\boldsymbol{y_t}$ contains the angles and velocities of all the joints. Under the assumption $\boldsymbol{w}$ the probability to observe $\boldsymbol{y}$ at a given time $t$ is expressed by

$$p(\boldsymbol{y}_t|\boldsymbol{w}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{y}1,t\\ \vdots \\ \boldsymbol{y}d,t\end{bmatrix}\middle|\middle|\begin{bmatrix}\boldsymbol{\Phi}_t^T & \dots & \mathbf{0}\\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \boldsymbol{\Phi}_t^T\end{bmatrix}\boldsymbol{w}, \boldsymbol{\Sigma}_y\right) = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Psi}_t w, \boldsymbol{\Sigma}_y). \tag{2.12}$$

### 2.6.4. Learning From Demonstration

Assuming a Gaussian distribution for $p(\boldsymbol{w};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_{\boldsymbol{w})}$ we get

$$p(\boldsymbol{y}_t;\boldsymbol{\theta}) \int \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Phi}_t\boldsymbol{w}, \boldsymbol{\Sigma}_y)\mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu_w}, \boldsymbol{\Sigma_w})\mathrm{d}\boldsymbol{w} = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Psi}_t^T\boldsymbol{\mu_w}, \boldsymbol{\Psi}_t^T\boldsymbol{\Sigma_w}\boldsymbol{\Psi}_t + \boldsymbol{\Sigma}_y), \tag{2.13}$$

the distribution of the state $p(\boldsymbol{y}_t|\boldsymbol{\theta})$ at the time $t$. This means the mean and variance can be easily evaluated at time $t$ given multiple demonstrations and $\boldsymbol{\theta} = \{\boldsymbol{\mu_w}, \boldsymbol{\Sigma_w}\}$ using maximum likelihood estimation.

### 2.6.5. Modulation By Conditioning

A for us relevant operation on ProMPs is the ability to condition the ProMP to reach a certain state $\boldsymbol{y}_t^*$ at time $t$. This is achieved by using Bayes theorem after adding a new observation $\boldsymbol{x}_t = [\boldsymbol{y}_t^*, \boldsymbol{\Sigma}_y^*]$ to our probabilistic model. This gives us $p(\boldsymbol{w}|\boldsymbol{x}_t^*) \propto \mathcal{N}(\boldsymbol{y}_t^*|\boldsymbol{\Psi}_t^T\boldsymbol{w}, \boldsymbol{\Sigma}_y^*)p(\boldsymbol{w})$. Here $\boldsymbol{y}_t^*$ is the state vector and represents the desired position and velocity vectors at time $t$ and $\boldsymbol{\Sigma}_y^*$ describes the accuracy of the desired observation.

We get these update rules for Gaussian trajectory distributions:

$$\boldsymbol{\mu_w}^{[new]} = \boldsymbol{\mu_w} + \boldsymbol{\Sigma_w}\boldsymbol{\Psi}_t(\Sigma_y^* + \boldsymbol{\Psi}_t^T\boldsymbol{\Sigma_w}\boldsymbol{\Psi}_t)^{-1}(\boldsymbol{y}_t^* - \boldsymbol{\Psi}_t^T\boldsymbol{\mu_w}), \tag{2.14}$$

$$\boldsymbol{\Sigma_w}^{[new]} = \boldsymbol{\Sigma_w} - \boldsymbol{\Sigma_w}\boldsymbol{\Psi}_t(\boldsymbol{\Sigma}_y^* + \boldsymbol{\Psi}_t^T\boldsymbol{\Sigma_w}\boldsymbol{\Psi}_t)^{-1}\boldsymbol{\Psi}_t^T\boldsymbol{\Sigma_w}. \tag{2.15}$$

We rely heavily on this feature when trying to predict future movements using ProMPs by conditioning it on the first few frames of a trajectory.

## 2.7. AE-ProMP

One major inspiration for this thesis is [3] *Prediction of Human Whole-Body Movements with AE-ProMPs*. What they are trying to is to predict future human whole-body movements $(t+1, ..., t_f)$ at time t, given t partial observations.

The basic idea behind using *AE-ProMP* is to reduce the dimensionality of the input data first, to make the ProMP computationally more efficient. So instead of giving the ProMP demonstrations in the original space, the demonstrations are encoded first. In [3] they show that using *AE-ProMP* is a little less accurate than using straight *ProMPs*, but the computation time is greatly decreased. This makes *AE-ProMPs* more useful for real-time applications like human-robot interaction.

# 3. Approach

The main thing we are interested in is how well different kinds of AEs work together with ProMPs. In figure 3.1 you can see the different AE architectures that we explore in the next chapters. You can roughly split the architectures into five types. The first one is the basic linear AE (figure 3.1a) introduced in chapter 2. It consists of two linear models an encoder and decoder. When talking about a linear model, we specifically mean a network consisting of two linear layers with *leaky relu* as an activation function. We call this model *linear AE*.

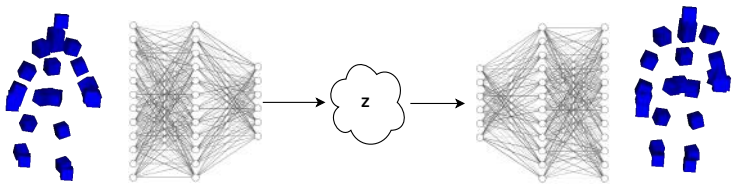
The second group of architectures we look at are AEs where the encoder and decoder both are recurrent models (figure 3.1b). This group consists of two architectures. A basic RNN and an LSTM. The exact architectures of the basic RNN and LSTM used are visualized in figures 2.2 and 2.3 respectively. We refer to these as *RNN AE* and *LSTM AE* in future chapters.

The third architecture is a model with a recurrent VAE as encoder, and a recurrent AE as decoder (figure 3.1c). To be exact both models are LSTMs, but the encoder has additional linear layers after the LSTM layer to output mean and variance. For each of these mean and variance pairs, we sample ten times and average the results to get an encoding. We refer to this network as *LSTM VAE*.

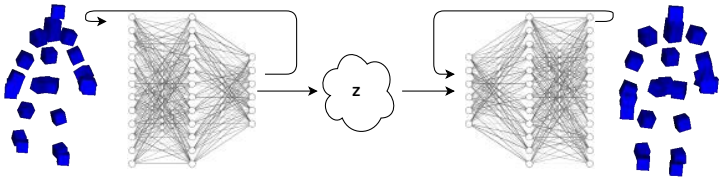
The fourth one has a recurrent encoder, but a linear decoder (figure 3.1d). Again the encoder is an LSTM network, while the decoder has the same architecture as the decoder of the linear AE. We call this one *LSTM AE Linear*

The last architecture uses the same encoder as the LSTM VAE and the same decoder as the linear AE (figure 3.1e). We call this model *LSTM VAE Linear*

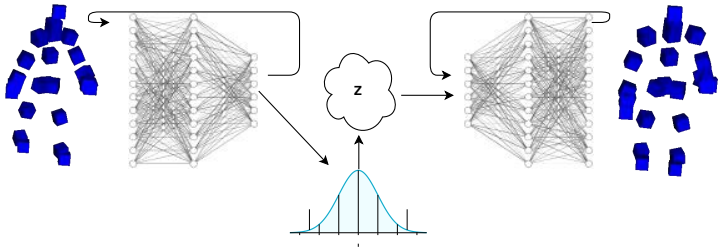
The idea now is to look at how well the different AEs can reconstruct skeletal data and work together with ProMPs to generate motion. The reason behind doing this is that depending on the AE used, the latent space could look very different. The recurrent



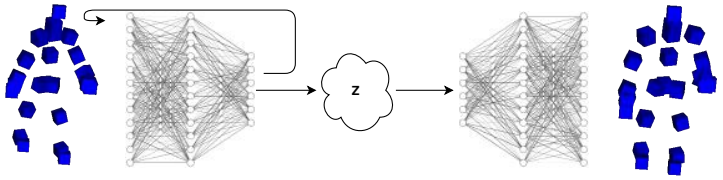

(a) Linear encoder and decoder. (*linear AE*)



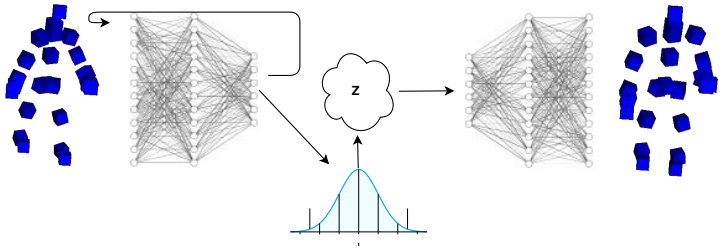

(b) Recurrent encoder and decoder.(*RNN AE, LSTM AE*)




(c) Variational recurrent encoder and recurrent decoder.(*LSTM VAE*)




(d) Recurrent encoder and linear decoder.(*LSTM AE Linear*)




(e) Variational recurrent encoder and linear decoder.(*LSTM VAE Linear*)

Figure 3.1.: Visual representation of AE concepts that we use in the next chapters.

models may somewhat capture coherent sequences better than the linear AE that encodes each frame without knowledge of the rest of the sequence. The VAEs are encouraged to have a more meaningful latent space through the incorporation of the KL Loss.

The way the ProMPs interact with the AEs is visualized in figure 1.1. Each of the AEs is trained separately from the ProMPs and each other. The encoder and decoder networks are of course trained together, so they both understand the same latent space. After training an AE, a ProMP is given 25 latent space demonstrations that were encoded using the AEs encoder. Sampling from the ProMP now creates trajectories in the respective latent space which can then be decoded with the AEs decoder.

# 4. Experiments

In this chapter, we compare and evaluate different variants of models and approaches relevant to leveraging latent space models in movement generation. After a section about how we preprocess the data, we look at different AE architectures and compare their performance on reconstructing skeleton data. After the AE experiments, we take the knowledge about how the AEs perform with different hyperparameters and compare how well they perform in conjunction with some motion prediction approaches. After that, we look at how well the best approaches actually generate movements.

## 4.1. The NTU RGB-D Dataset

As a base dataset for all of the experiments conducted in this thesis, we use the NTU RGB-D dataset [16]. It is a large scale dataset of 3D human actions containing more than 50000 samples. Each of the samples contains four modalities. RGB videos, depth map sequences, infrared videos, and 3D skeletal data. You can see an example of this in figure 4.1. We only use the 3D skeletal data in this work. There are two versions of the dataset available with 60 and 120 actions respectively. Of these two we use the smaller one, since many of the actions of the extended set are less relevant for the specific tasks we set out to do. Each of the 60 action classes has been recorded with multiple cameras in multiple setups and sometimes by multiple people. All actions are repeated twice per setup.

## 4.2. Data Preprocessing

As introduced in the last section we use the skeletal data and to be even more exact only the 25 joint positions of the NTU RGB-D dataset.

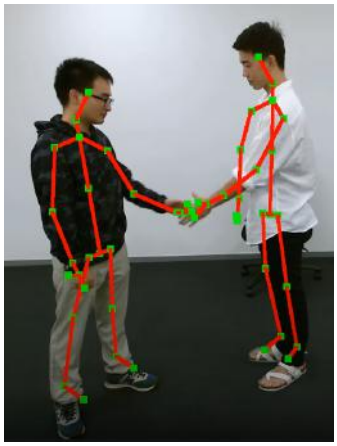


Figure 4.1.: NTU RGB-D dataset example of the *shake hands* action. We only use the green nodes representing joints, like the knees, or end-points, like the head. *Source: https://www.youtube.com/watch?time_continue=1v=dWs9aqgfgQ8feature=emb_logo*

To have a more manageable set of actions, we selected five actions to conduct the experiments with. The actions we selected are *drink water*, *throw*, *hand waving*, *taking a selfie*, and *salute*. Before actually working with this data though, we had to go through every single demonstration manually to discard the ones that could not be used for the tasks at hand. For example, the *throw* action is too broad of a definition to learn. Meaning that we would actually only need demonstrations of a smaller subcategory of *throw*. In this particular case what we really want is *right handed over head throws*. So for this action we discarded all the left-handed, two-handed, and all other movements that are technically throwing motions but aren't similar enough to the rest. Also, some of the demonstrations were corrupted in some way. For some, there may have been some occlusions, while for others the camera would recognize some random object as a human being. These were discarded as well. I ended up with 891 demonstrations.

Having now selected the actions to learn, the next step is to process them in a way that eliminates unnecessary variation in the data. So all skeletons were moved to the same position and rotated in the same direction. Because some demonstrations include a long waiting time before the performer actually starts acting, we cut off a couple of the first frames depending on when the average joint velocity first reaches a certain threshold. This is necessary because ProMPs need their demonstrations to be rather time aligned for them to work properly. We cut off or extended sequences to be 50 frameslong. So if not

mentioned explicitly otherwise, assume a sequence length of 50.

## 4.3. Autoencoders

In this section, we discuss and compare different Autoencoder (AE) architectures. The main goal of this is to find out what kinds of AEs do well on skeletal type data. We look at the models introduced in chapter 3 and compare them in multiple ways. First, we look at the influence of the latent space size on the reconstruction *Mean Squared Error* (MSE).

### 4.3.1. Mean Squared Error By Latent Space Size

First, we compare just the AEs by themselves given different latent space sizes and hidden dimension sizes. The MSEs by hidden and latent dimensions are visualized in figure 4.2. These models have been trained on whole body, action agnostic skeletal data.

Looking at figure 4.2 we can see that the purely linear AE has the lowest MSE out of the bunch, no matter the settings. It is also the only model that encodes movements frame by frame without knowledge of the rest of the movement. All other movements have some sort of hidden state that knows about previous time steps of the sequence in some form. The hope here is that the recurrent type models encode a more meaningful latent space, but just looking at the blank reconstruction MSE the linear AE outperforms all other models.

Only when the random initialization of the LSTM encoder is favorable does the LSTM AE with the linear decoder get similarly low MSEs. And this is one thing to keep in mind here. All the recurrent models are initialized randomly. So even though it looks like the hidden and latent dimensions matter a great deal, they don't. At least not as significantly as it may look on first sight. But for example too high latent space and hidden dimensions do increase the MSE for the LSTM VAE Linear model. Between 50 and 90 hidden dimensions and latent dimensions larger than four seem to be alright though. All of the models seem to struggle a bit with latent space sizes smaller than six and the overall performance is pretty comparable between the RNN AE, LSTM AE, and LSTM VAE. The LSTM AE Linear, from a reconstruction MSE point of view, does seem to outperform its pure LSTM counterpart. For the LSTM VAE and its linear decoder version, the same cannot be said necessarily. The LSTM VAE with the linear decoder is more unstable in how well it reconstructs the movement.

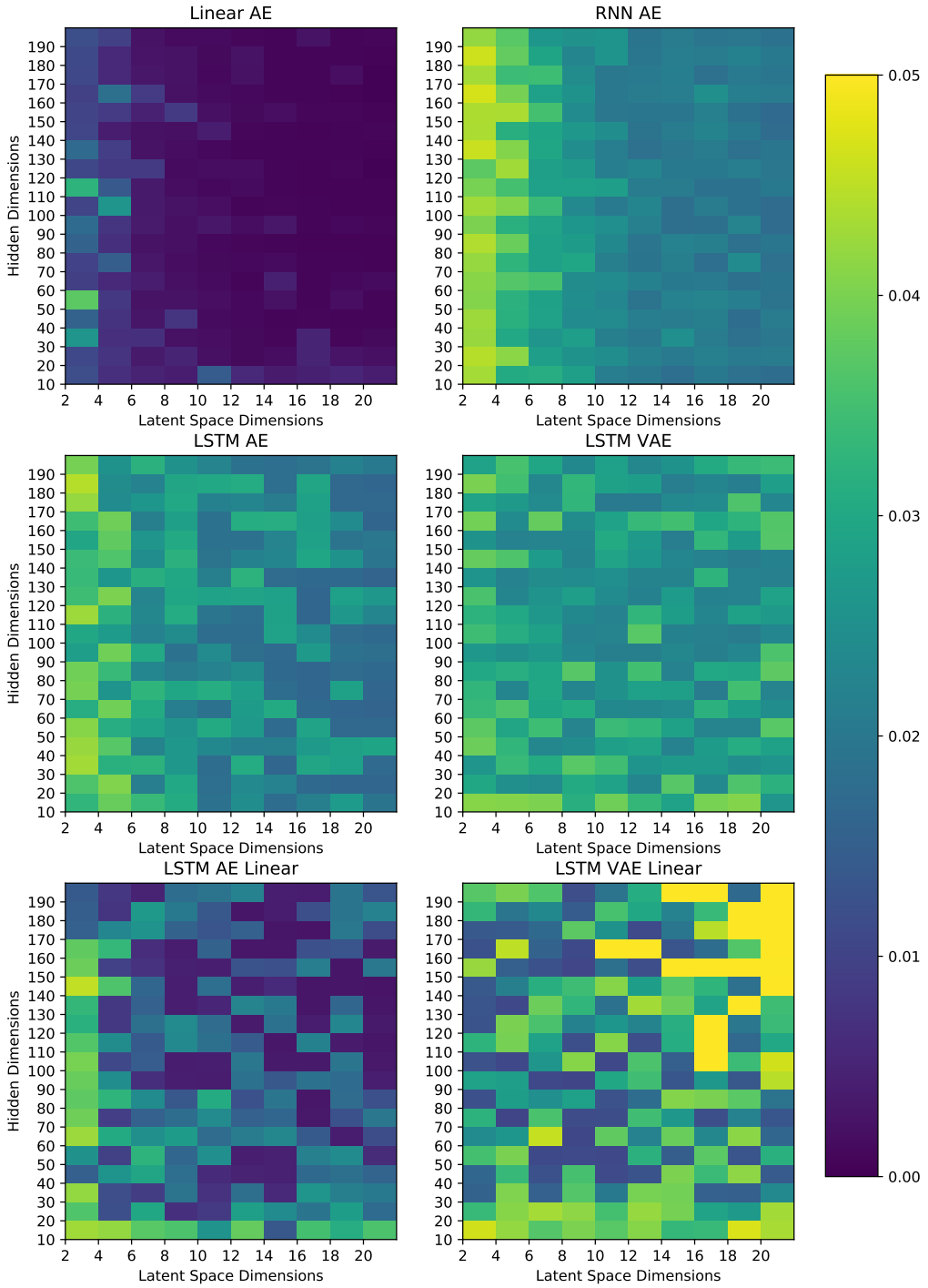



Figure 4.2.: MSE by hidden and latent space dimensions.

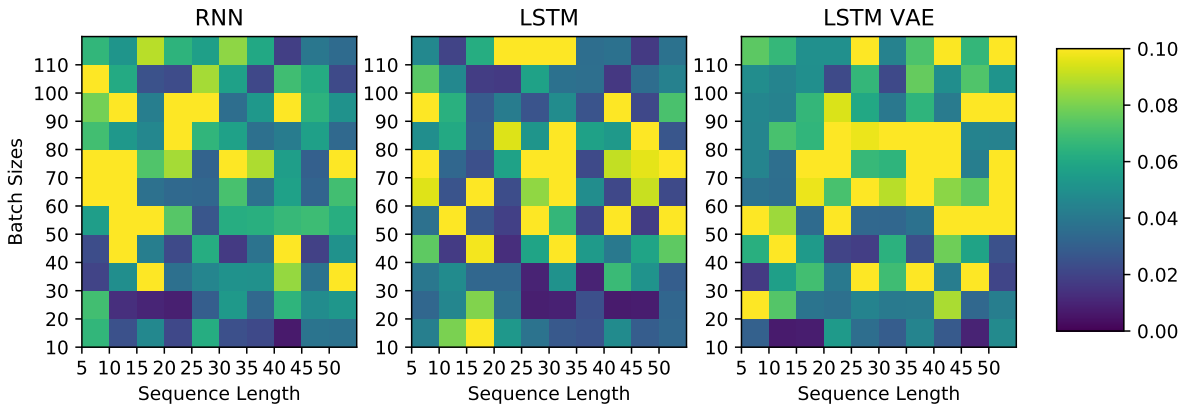



Figure 4.3.: MSE with different batch sizes and sequence lengths using full-body data.

Going forward we stick to a hidden dimension of 60 and a latent dimension of 16 since all models seem to work fine with those values.

### 4.3.2. Mean Squared Error By Sequence Length

In figure 4.3 you can see a comparison between using different batch sizes and sequence lengths. The sequence length is the number of frames in a sequence. As you can see in the three plots, the MSE gets really high for batch sizes larger than 40 and is best for really small batch sizes. The difference between how well the RNN and LSTM perform on short and long sequences is to be somewhat expected. Because of the random initialization, the LSTM takes longer to update the internal cell state and produce actual reconstructions, while the RNN with its simple recurrence performs well, very quickly. For us, this means that we use sequence lengths of 50 and batch sizes of 10.

### 4.3.3. Influence Of KL Loss

One experiment only relevant to the variational models is a comparison of how much the latent space should be constrained. As introduced in the foundations chapter, these kinds of models use a KL Loss to restrict how similar movements should be represented in the latent space. Here it is quite important to find the balance between getting a more useful latent space and getting a reconstruction closer to the ground truth. In figures 4.4 and 4.5 the influence of the KL Loss on the reconstruction MSE is visualized. Both models seem

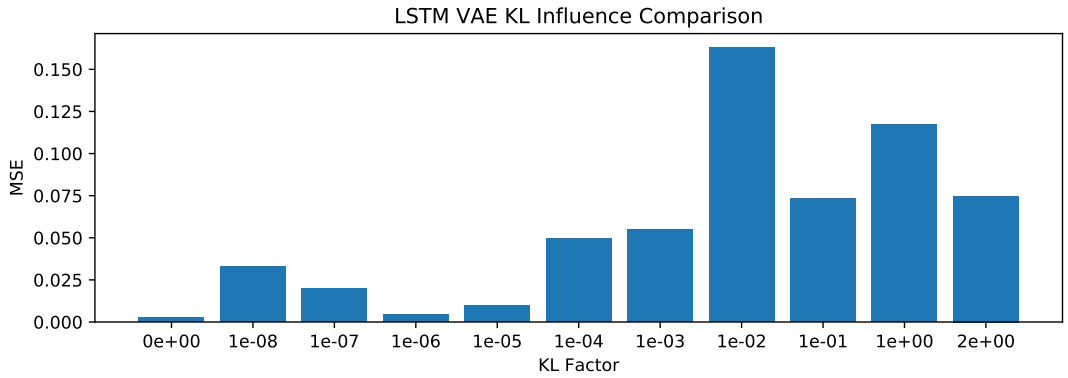



Figure 4.4.: Here you can see the negative impact of increasing the influence of the KL loss on the MSE of the reconstructions. The *KL Factor* is simply multiplied with the KL loss when training the *VAE*. This makes it possible to control how much the latent space should be constrained.

to behave pretty similarly here. For both the MSE gets higher as the influence of the KL Loss on the training of the models rises. There is some amount of randomness involved again in how well the models perform with the different KL Losses. Again, this happens because of the random initialization and random sampling from the test set. Given that this experiment shows that increasing the influence of the KL Loss increases the MSE, but we still want some influence to test the effectiveness of VAEs, we use a KL Factor of $1e^{-6}$ for further experiments (the KL Factor simply gets multiplied with the KL Loss when training the model).

### 4.3.4. 2D Latent Space Visualization

To better understand how movements might be mapped into a latent space, you can find visualizations of movements encoded into a 2D latent space in figures 4.6, A.2, and A.3 for the *salute*, *throw*, and *hand waving* actions respectively. For each AE 30 trajectories were plotted starting out with a low alpha value and getting more opaque as time goes on.

Even though not everything seen in these figures is guaranteed to generalize well for other latent dimension sizes, it does give us some useful insights into how the different AEs may organize their latent spaces. The three actions picked for this experiment all have different qualities that are worth looking at. The most simplistic of the bunch is the *salute* action. As you can see in figure 4.6 all of the AEs represent actions of this category in a very similar way, with the vast majority of demonstrations ending up in the same point.

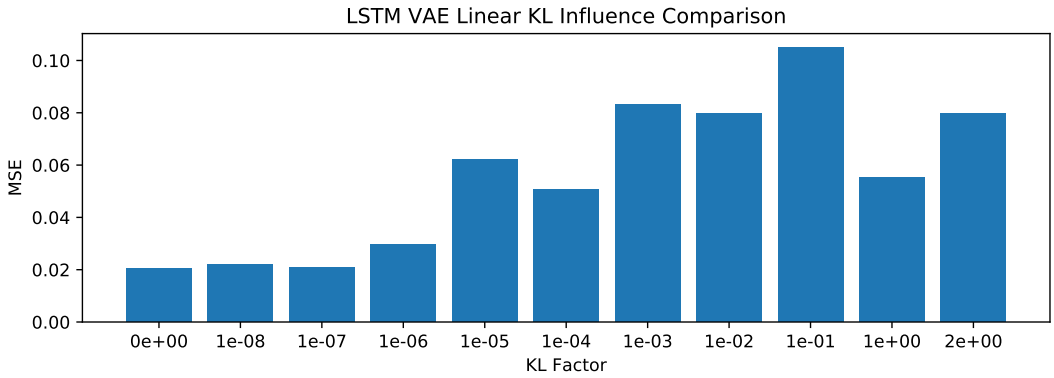



Figure 4.5.: Here you can see the same comparison as in 4.4 but for the *LSTM VAE* with the linear decoder.

The linear AE seems to have the largest amount of variance in how close the different demonstrations are located to one another, but still, all of them end up at the same point. For the LSTM AE and VAE it is worth mentioning how similar their latent spaces end up looking compared to the ones created by their linear decoder counterparts. This is particularly noticeable in the LSTM VAE.

Similar representation becomes even more significant when looking at the other movements. When comparing how the *salute* and *throw* actions (figures 4.6 and A.2) look in their latent space representations, one thing you will notice is that for some of the AEs the trajectories look very similar. This is especially the case for the LSTM VAE and its linear decoder version. They both have this kind of "hook" looking end towards the top left corner in which many of the trajectories end. This is useful since this is what we want to a certain extend. Both of the motions end really similarly, by lowering their right arm to be next to their bodies. This behavior is problematic when movements that are different still get mapped to the same position in the latent space. Both LSTM AEs also capture this, but to a lesser extend. The linear and RNN AE do also show signs of this behavior, but much less so than either of the LSTM models. Throwing is obviously a very different and more chaotic movement than saluting, so we would really only want and expect the very first and the very last parts of the movement to be similar. In all of the models it is visible that the middle portion is different, so no surprises here.

The last action, *hand waving*, is actually a rather special one because it is the only action of the three that has a fair amount of sitting demonstrations. So what we would expect to see here is two independent regions where the different kinds of trajectories are mapped to, because the demonstrations are either sitting or standing ones, never both. To be even more exact we would expect one region to be as similar to the *salute* action as the *throw*

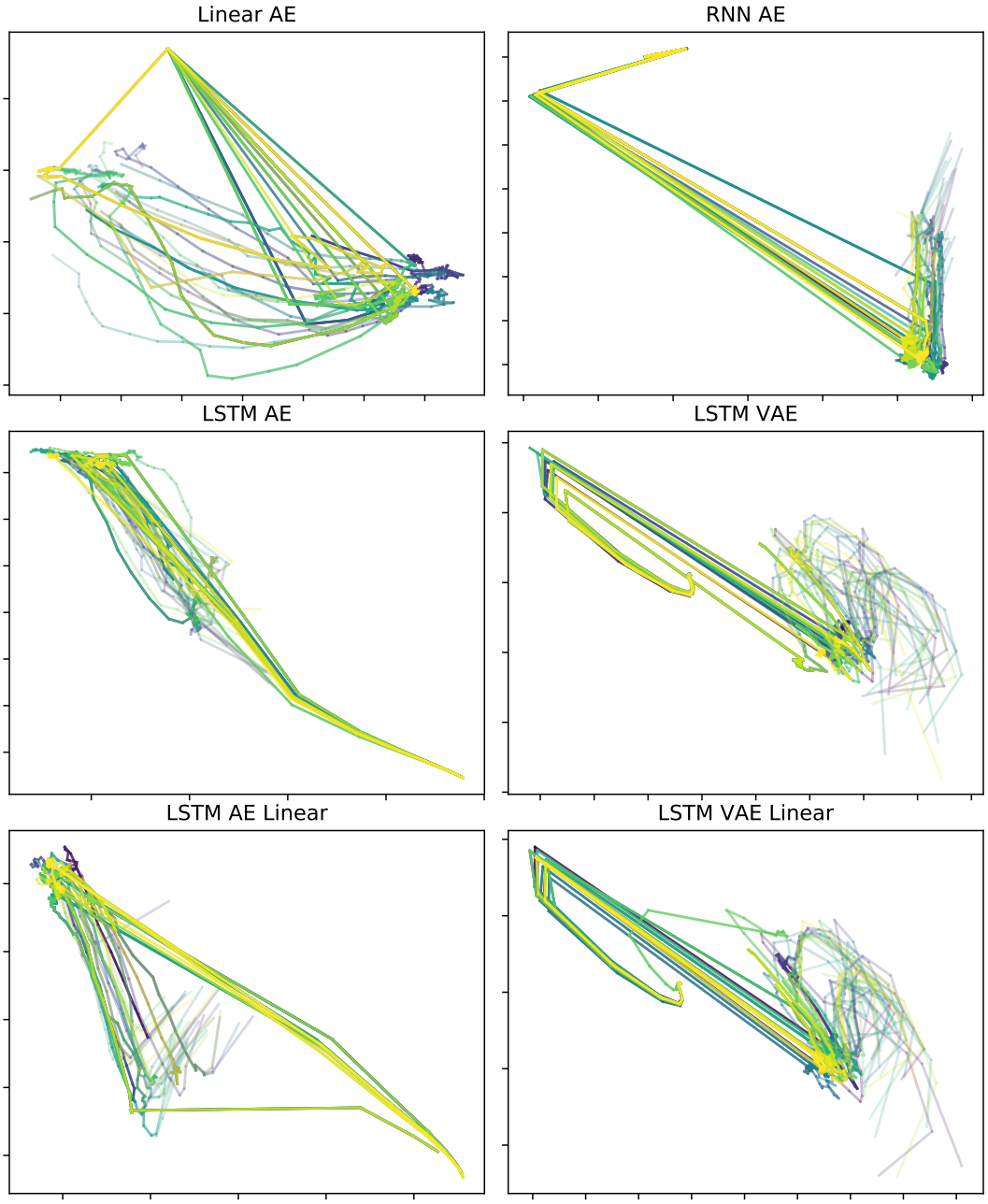



Figure 4.6.: 2D latent space generated by different AEs for the *salute* action.

action is. This is the case because waving your arm also requires raising and lowering it at the beginning and end of the action. This spacial distinction is not necessarily required for the decoder to be able to correctly decode the movement, but it should be easier to do so if they don't overlap much. Looking at figure A.3 that is exactly what we can observe for many of the AEs. Again for both VAEs the similarity is quite obvious with the sitting variation of the movement (probably) positioned a bit further down. This is not something only the VAEs do though. As far as we can tell all of the AEs do this besides maybe the LSTM AE.

### 4.3.5. Conclusion

When looking just at the AEs without any further context the experiments could suggest that using a linear AE is better than using one of the other models. Looking at the visualizations of the latent spaces though and with the main goal of this work in mind being to generate movements in the latent space, it seems plausible that having a more meaningful latent space could outweigh perfect reconstruction performance. Especially given that the reconstructions of the linear AE visibly stutter when looking at the output manually.

As for the hyperparameters we analyzed, we will for most experiments stick with a latent dimension of 16, a hidden dimension of 60, a sequence length of 50, a batch size of 10, and for the variational models a KL influence factor of $1e^{-6}$.

## 4.4. Movement Prediction

Although not exactly the target of this thesis, the ability to predict movement gives us useful insights into how well movements can be generated. This is especially relevant since it is very difficult to gauge whether a generated movement is *good* or *bad*. For many of the experiments in the context of movement prediction, we use two baselines that achieve good results in predicting future movement. The first is a network introduced in [11] called *LSTM-3LR* which is basically a three-layer *LSTM* that is surrounded by encoder and decoder layers. We'll be using a slight variation of this, namely replacing the Autoencoder layers with the networks we tested in the previous experiments. The other major baseline is a simple *Gated Recurrent Unit* (GRU). This is inspired by the approach in [10]. Although we also changed the setup in the same way as the other baseline to enable a more direct

comparison between the different approaches. This enables us to use the same AE for all of them.

To predict future movement using a ProMP we condition it on the first n frames and then sample from it. The baselines work a bit different, namely by iteratively passing the output of the previous step back into the model as the next input. These baseline RNNs were trained by inputting a frame and then comparing the output to the next frame of the sequence.

### 4.4.1. Basis For ProMP

Since we heavily use ProMPs in all of the experiments, the first step is to identify what settings the ProMP should run on. In this case, this means how many basis functions, or degrees, should be used in combination with what scale, or variance. In figure 4.7 you can see a comparison for some of the AEs and how well they predict future movements in combination with a ProMP with the respective settings. These values are averaged over multiple ProMPs, one for each of the five actions.

Each ProMP was given 25 demonstrations in the respective latent space (or joint space for the ProMP without AE) for the action that ProMP should learn. Not all actions are predicted equally well and it does seem to matter quite a bit what combination of AE, basis functions, and variance we use. From figure 4.7 we can see a general trend that high degrees require lower amounts of variances. From further testing, we were able to gather that lower degrees don't work well for predicting movements, especially for the ProMP trained directly in the joint space. Some settings turned out to be numerically unstable and at random would create basically useless predictions. This is for example also the case for the joint space ProMP using high scale values. Because of this, we found that using 25 basis functions and a scale of 0.001 works well for prediction in all spaces and all actions. We will have a look at this again for the generation task where this numerical instability does not seem to be a problem.

Something worth mentioning is that figure 4.7 shows that the recurrent AEs have a higher tolerance against this numerical instability and are less choosy in the kinds of settings used. Especially the models trained in the LSTM AE and VAE spaces have really low MSE in comparison to the other models.

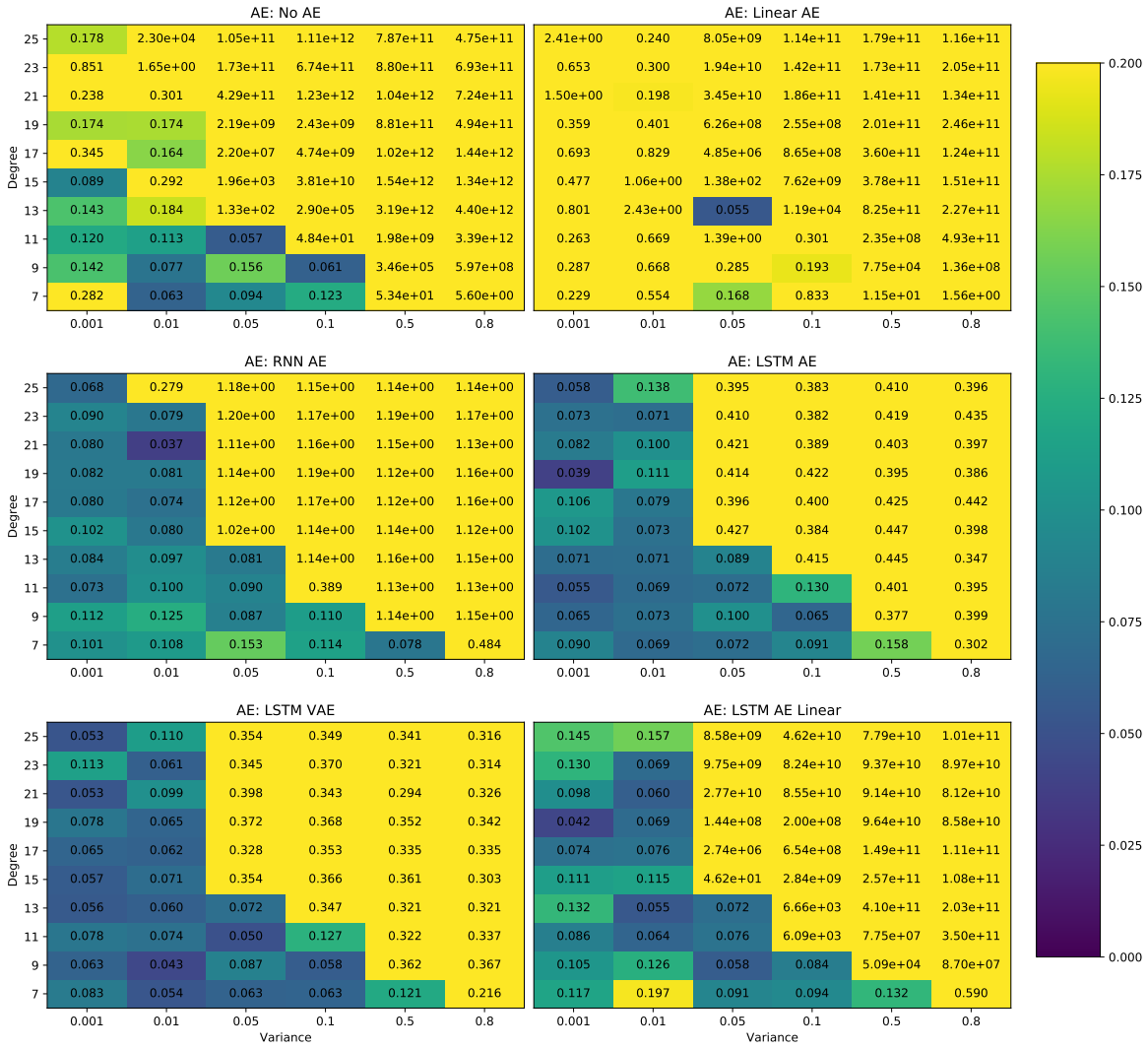



Figure 4.7.: Comparison of using different amounts of basis functions and different scale values for the ProMP. The ProMPs were conditioned with multiple sets of partial trajectories. The MSEs were averaged over all of the five actions and different fractions of the number of frames given. As you can see high degrees mixed with high variances often result in numerical instabilities.

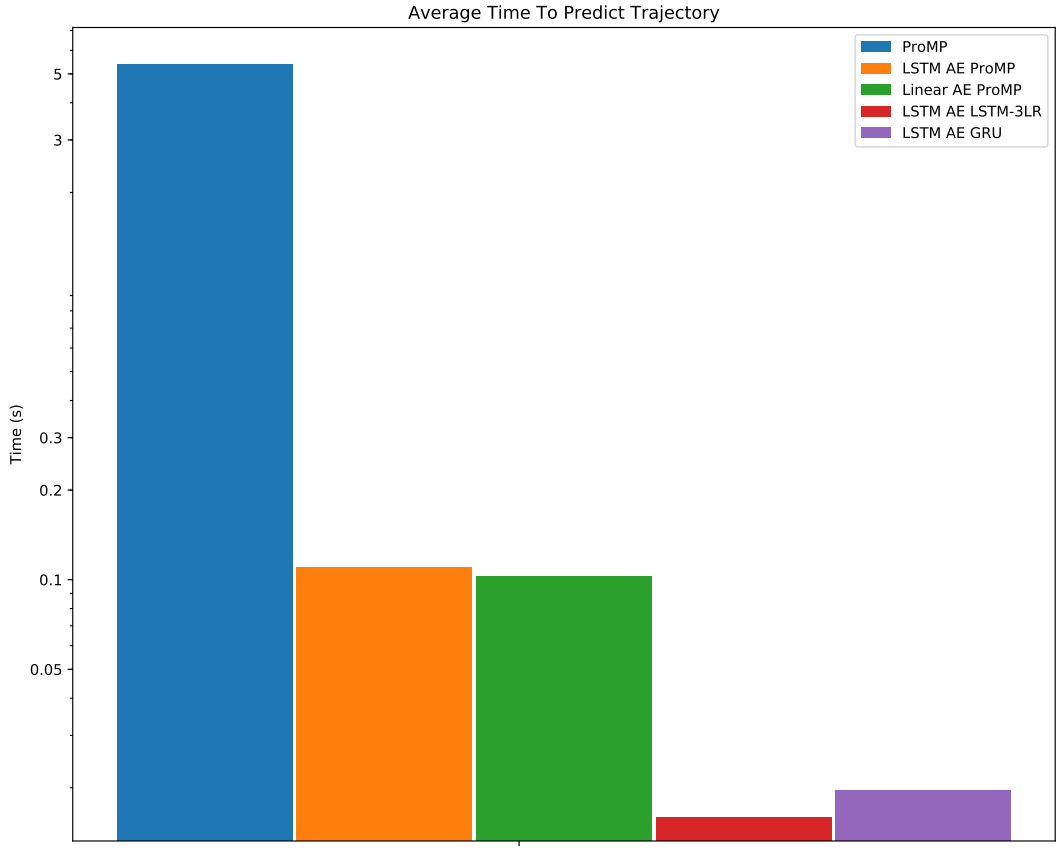



Figure 4.8.: Here you can see the average time it takes for a selection of models to predict a 50 frame long sequence. Using the *LSTM AE* rather than one of the other models does not make a large difference when comparing the different predictive models against one another, to make it necessary to compare the other AEs and VAEs as well.

### 4.4.2. Prediction Time

The main idea behind using *AE-ProMP* is that it is a lot faster to generate trajectories if the space in which they are generated is smaller [3]. This is crucial when trying to create motion in real-time, for example for interaction with human beings. In this section, we explore this a bit more and compare the time it takes to predict a trajectory using the aforementioned baselines and also how much faster using *AE-ProMPs* rather than *ProMPs* really is.

So the first thing to test is whether or not it is actually viable to use AE ProMP instead of ProMP. It turns out that it is a lot quicker to do so than generating them by using a ProMP in the joint space. The overhead from using a recurrent AE instead of a linear one seems neglectable

In figure 4.8 you can see the average time it takes the different models to predict a 50 frame long trajectory. The exact time is not too import, since it is very hardware dependent. You can see that the baselines actually outperform all ProMP based models by quite a bit. This is the case because the conditioning takes a long time. A lower number of basis functions also reduces the time it takes to sample from a ProMP. As you will see when we do this comparison again for the generation task, computing time is decreased a lot if neither of these is an issue. The time scale is non-linear so the joint space ProMP is a lot slower than it may seem at first sight. To predict a 50 frame long sequence with the joint space ProMP takes on average five seconds, while the AE ProMPs only take a bit more than 0.1 seconds. Note that these times cannot be directly compared to the times of other experiments, since the computation time is dependent on other experiments running at the same time. All of the predictions in this experiment were made in a similar enough environment, and have been conducted enough times to be a good comparison between computation times.

Since the ProMP takes much longer to generate the trajectory than the AEs take to decode them both the linear and LSTM based AEs perform similarly good in terms of prediction time. The linear AE does seem to be a little bit faster though.

### 4.4.3. Comparison Of Autoencoders In AE-ProMP

Having now established that from a computation time point of view it is worth using ProMPs in conjunction with AEs we now compare how well the different AE models

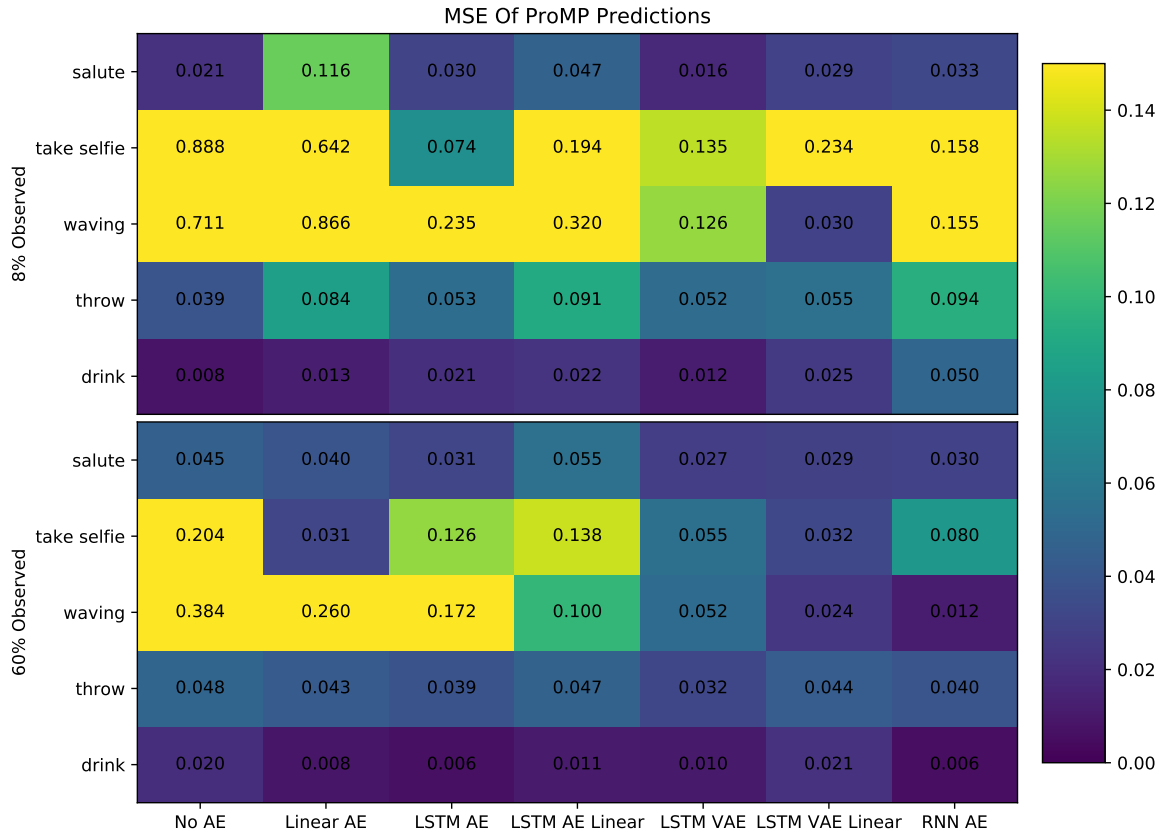



Figure 4.9.: Comparison of the MSE for predicting future movement for different actions, each learned by a ProMP given 25 demonstrations. These ProMPs were trained and tested in the latent space generated by the corresponding AEs of the x-axis.

perform together with ProMPs. In this experiment we compare how well the models can predict future movements. The measurement used for this is again the MSE.

A ProMP was trained in the latent space of each of the AEs with 25 demonstrations of the action it is supposed to predict. The ProMPs were then conditioned on a part of the trajectory it is supposed to predict. In this case, we looked at 8% and 60% observed. You can see the results of this experiment in figure 4.9.

For all experiments conducted in this section, we used the Gaussian basis function for all of the movements regardless of what they look like. The only movement that is somewhat periodic is the *hand waving* motion anyways and even that action is partially stroke based. The reason for using only the Gaussian basis function is that through trial and error we found out that the Von Mises function often creates numerical errors when using a low scale for the ProMP. Since we found out in one of the previous sections that using 25 basis functions with a scale of 0.001 works best to avoid numerical instabilities and part of the waving motion is stroke based anyways we'll use the same settings for all actions.

Having said that, it does seem like the *hand waving* action seems to be one of the two most difficult motions for the ProMPs to predict. The other being the *take selfie action*. The *selfie taking* action is rather difficult to accurately predict because there is no knowing how long a person may wait before taking the picture. Meaning there is no way the models can accurately assess when they should be taking the arm down again. Also, you can take selfies from many different angles, making it hard to know how high to lift the arms. As you can see in figure 4.9 when the ProMP gets to observe 60% of the movement the arm position of which the person takes the image is already revealed, making the MSE go down for all of the model types.

The action that seems to be the easiest to predict is the *drink* action, followed by the *salute* and the *throw* actions. Only the linear AE seems to have some problems with the *salute* action when observing 8% of the observation. The models with the lowest MSE in the majority of cases are the LSTM VAE based models. Depending on the action either the linear decoder version or the full LSTM version does a little bit better, but overall the results are pretty comparable. Interestingly the RNN seems to do best on the *hand waving* action when having observed 60% of the trajectory. And for other actions too this simple RNN model produces good results.

This experiment suggests that to decrease the prediction MSE of a ProMP trained in an AE-generated latent space, one should use a recurrent model especially one with slight variational influence. Because the RNN is outperformed when not much of the trajectory

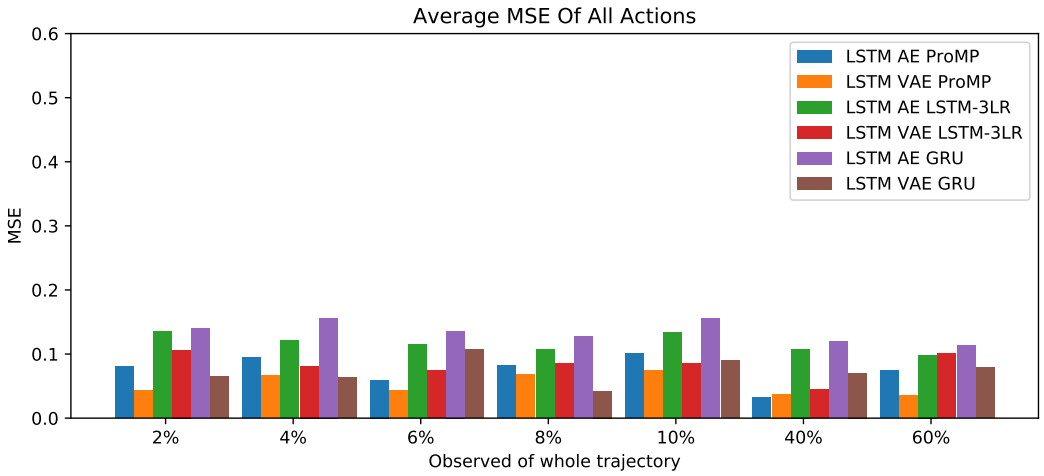



Figure 4.10.: Comparison between different methods of predicting future trajectories. All of these predictions were performed in a 16-dimensional latent spaces. The aforementioned *LSTM AE* and *LSTM VAE* is used for this.

is observed, we will from now on focus on using LSTM AE/VAE and the respective linear decoder versions.

### 4.4.4. Comparison With Baseline

For this experiment, we'll again use the average MSE between the original trajectory and the one predicted using ProMPs and the baseline RNNs. Because the difference in complexity of all the movements is so large, we compare the average MSE for all of them separately. As AE we now use LSTM AE/VAE and their respective linear decoder versions. After encoding the trajectory with the same AE/VAE the models were trained with, we use *ProMP*, *LSTM-3LR*, and *GRU* to predict the trajectory.

The results of this experiment are visualized in figures 4.10 and 4.11 for the pure LSTM and linear decoder versions respectively. In the appendix, you can find these same results but separate for each action in the figures A.4, A.5,A.6, and A.7.

Looking at figures 4.10 and 4.11, the VAE versions outperform their AE counterparts in terms of prediction MSE. In some cases of the linear decoder, the difference is quite significant even. Also, the baselines seem to perform better with a linear decoder in most cases. This seems to have the opposite effect for the ProMP though. For the versions with an LSTM decoder both ProMPs outperform the baseline approaches. That is every AE

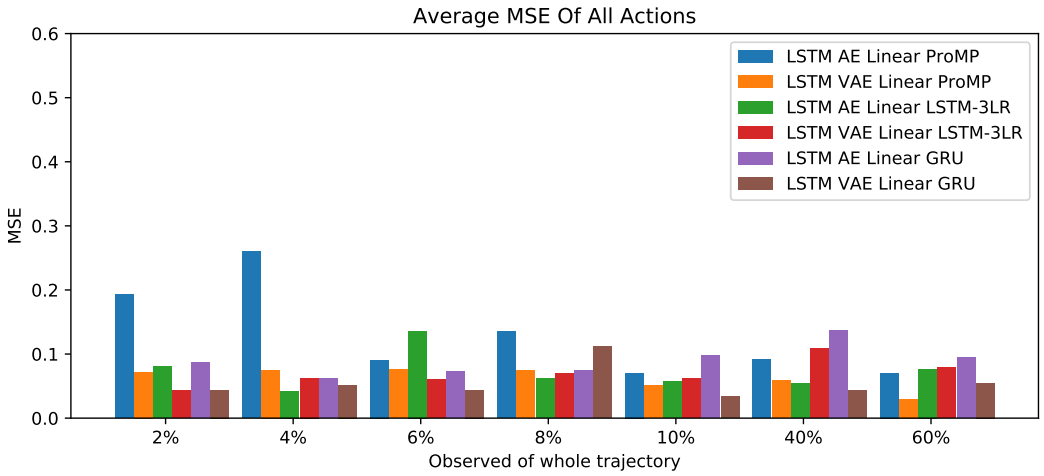



Figure 4.11.: Same as figure 4.10 but using the version with the linear decoder.

ProMP gets a lower prediction MSE than the other AE approaches and the VAE ProMP gets lower MSEs than the VAE baselines. There are some exceptions to this, but this seems to be mostly what we can observe.

For the linear decoder, versions the exact opposite is the case. The baselines mostly outperform the ProMP approaches. Only when having observed 60% of the trajectory here does the ProMP outperform the baselines.

There is however an advantage that models with a simple linear decoder have in terms of reconstruction MSE that is not captured by these figures. The first few frames of a decoded trajectory are going to be terrible for the LSTM decoder models, because of the random initialization. Since the whole sequence is decoded and only the generated frames are compared with the original, this disadvantage is not reflected in the results. We talk about this effect more in later sections.

There is quite a difference again in how well different actions are predicted. And again the *hand waving* and *take selfie* actions seem to be the biggest problems as you can see in figures A.4, A.5,A.6, and A.7. You can also see there that the baselines do even worse in many cases for these actions, confirming the difficulties of learning these actions.

### 4.4.5. Comparison Of KL Loss Influences On VAE ProMPs

The point of this experiment is to look at the influence of the KL Loss on VAE-ProMP. The experiment on the influence of the KL Loss on the reconstruction MSE of VAEs showed that

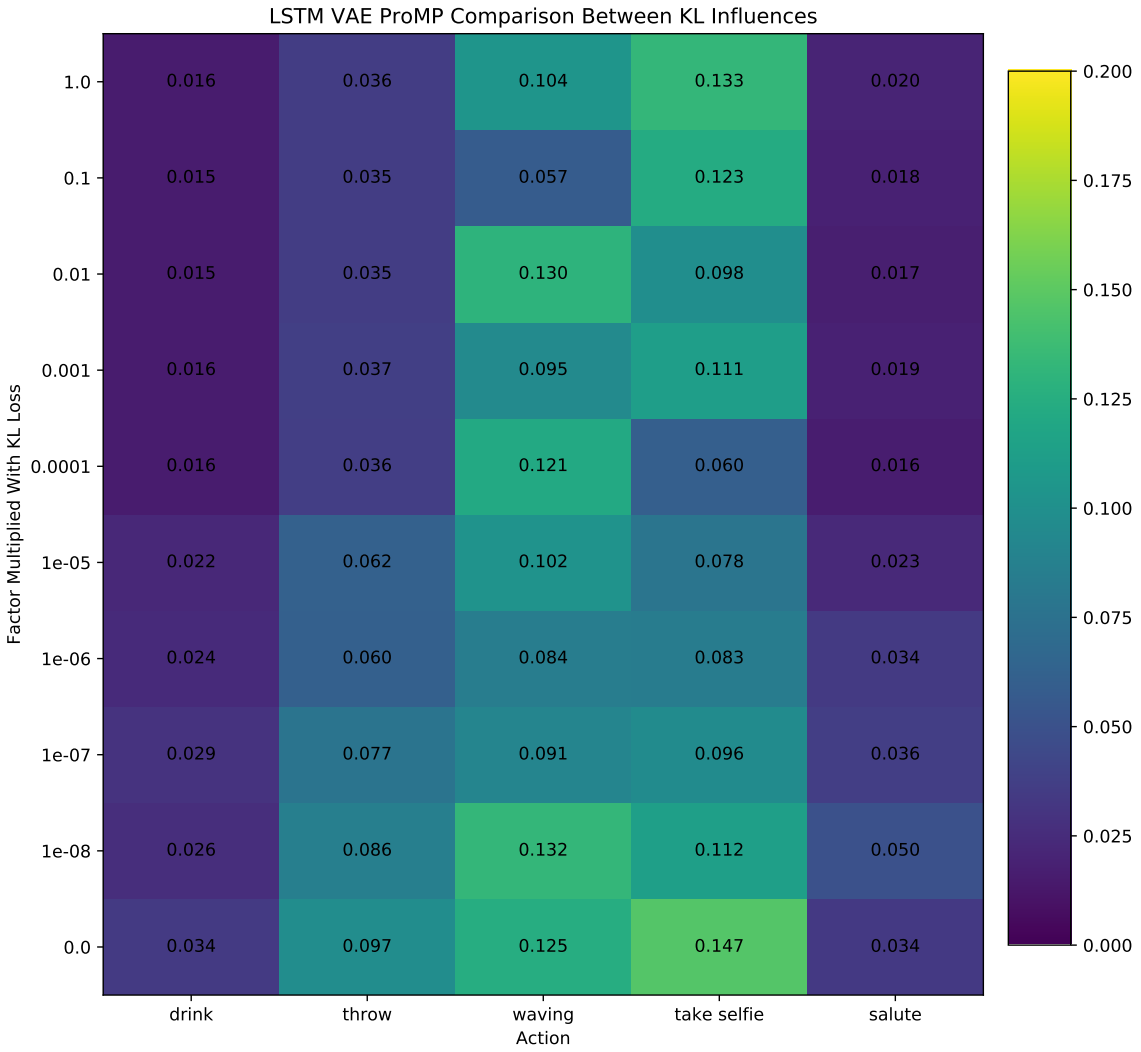



Figure 4.12.: Comparison between the five movements on how well they can be predicted by a ProMP trained on the latent spaces of *LSTM VAEs* given different amounts of KL Loss influences.

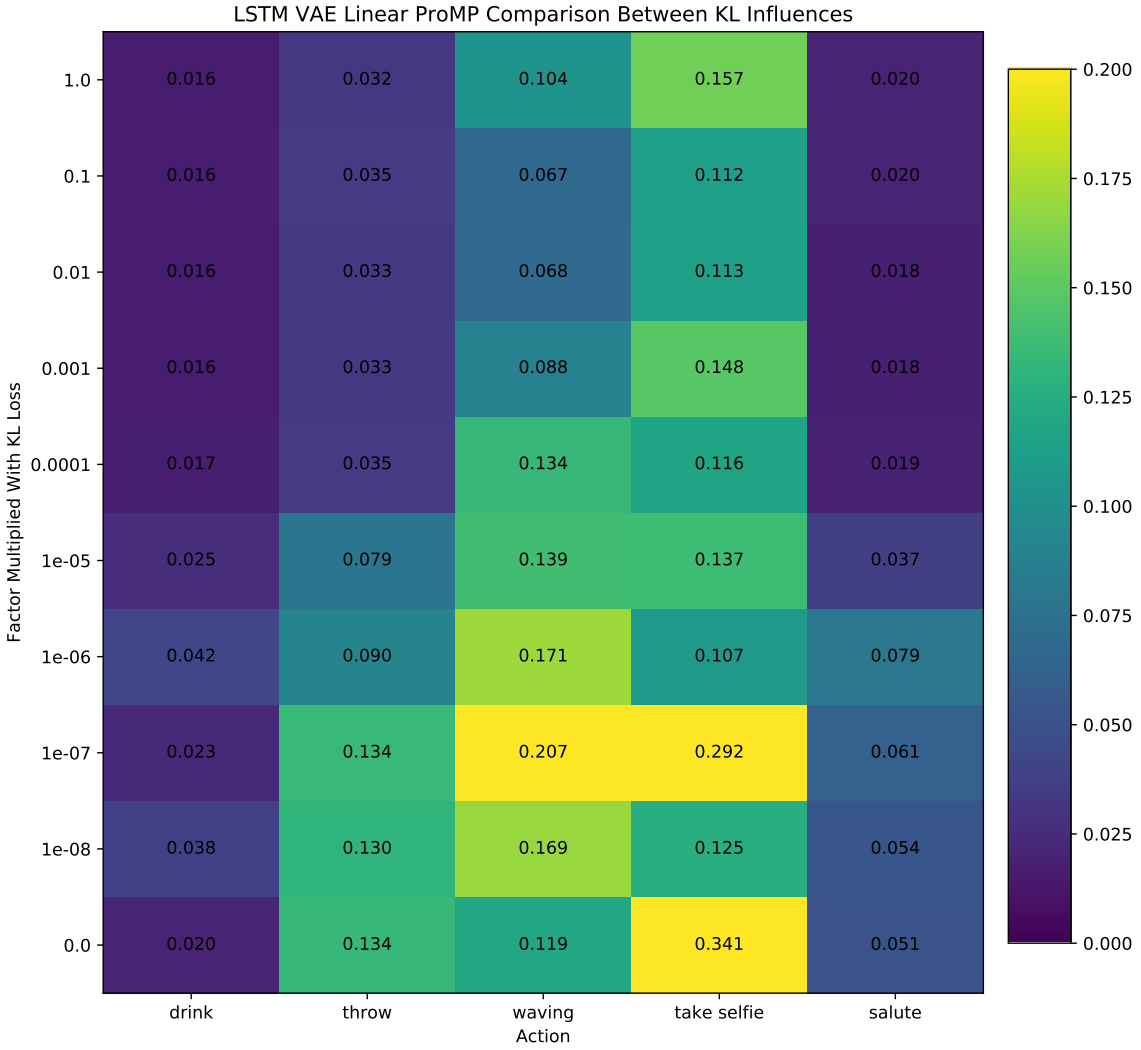



Figure 4.13.: Same experiment as in figure 4.12, but using the version with the Linear decoder.

having a low KL influence decreases MSE. So for this, we look again at each of the actions separately since they seem to differ quite a bit. You can see the results of this experiment visualized in figures 4.12 and 4.13 for the LSTM VAE and LSTM VAE Linear respectively.

In these figures, you can see how both models seem to profit from increasing the influence the KL Loss has on the training. That is the exact opposite of what we previously observed when analyzing the AEs without predicting anything. The main reason that comes to mind for why this might happen is that sometimes having a static person is good enough to get a relatively high MSE. This means that if the latent space is highly constrained, one way the model can lower its training loss is by just making sure the latent space fits the constraint, while the reconstructions become still standing skeletons. We take a closer look at this phenomenon in further sections.

The MSEs of the linear decoder version VAE depicted in figure 4.13 lead us to the same conclusions as to the LSTM decoder version. The MSEs for KL Factors smaller than $1e^{-5}$ are higher than the ones of the LSTM decoder model, though not by a huge amount.

Another thing worth mentioning is why the error seems to be higher for a VAE without KL Loss than for an AE with a very similar architecture. The main reason for this is the random sampling that is performed after the linear layers of the VAE output a mean and variance.

### 4.4.6. Quality Of Predictions By KL Factor

Here we further inspect how the KL Factor influences the predictions made by a ProMP in a latent space created by a VAE. This time we look at what the predictions look like in the joint space after being decoded. You can see the visualizations of these predicted trajectories in the figures 4.14 and 4.15 each for one of our VAE models. To give you a rough idea of the effects different KL Factors can have on the predictions made by a ProMP, we visualized predictions for the *throw* action as an example. For each of the two VAE variants, three different models were trained with different amounts of KL influence reaching from 0 to 1. Then ProMPs got their 25 demonstrations in the respective latent spaces. After conditioning them on the first 8% of a trajectory they were to predict, we sampled from them. In figure A.1 you can see some samples from the original dataset.

The results of this experiment seem to confirm our speculations in the previous section on why higher KL Factors lead to higher MSEs in pure AE reconstruction and lower MSE in ProMP prediction. As you can see in figure 4.14 the movement generated in the 0 KL Factor latent space is just a poor prediction in this case. This is a pretty extreme example

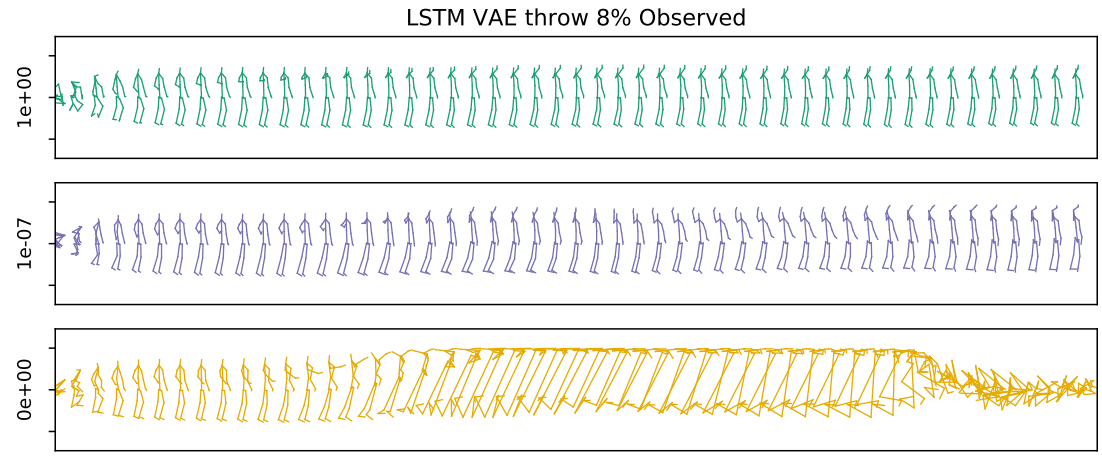



Figure 4.14.: Example of how different KL Factors (labeled on the left) affect the predictions made by a ProMP. The ProMP that generated these trajectories was trained by an LSTM VAE using said KL Factor.

and some predictions, especially for other actions, actually look a bit better. The reason why this prediction is so poor is because, as mentioned in the previous section, a VAE without KL Loss basically becomes a less accurate AE because of the random sampling taking place during encoding. This makes the latent space less meaningful without further constraints. The other extreme can also be seen in the same figure. If the whole KL Loss is added to the total Loss in training, you can end up with very stiff predictions. This is the case because the VAE learns to prioritize the latent space over the reconstructed joint space. The effects of this are even more dramatic in the full KL Loss sample in figure 4.15.

From these examples only the middle KL Factor is really usable. You can clearly make out the throwing motion in both figures. There is one rather obvious difference between the one from the LSTM decoder and the one with the Linear decoder. The skeletons of the LSTM decoder are much more coherent and remain the same size throughout the whole motion. The disadvantage, as mentioned in the previous section, is that the first few frames generated by an LSTM decoder are never meaningful or even usable because of the random initialization.

### 4.4.7. Quality Of Predictions

Now we compare some more of the trajectories generated by various combinations of AEs and ProMPs. In figure 4.16 you can see trajectories generated by the models using the

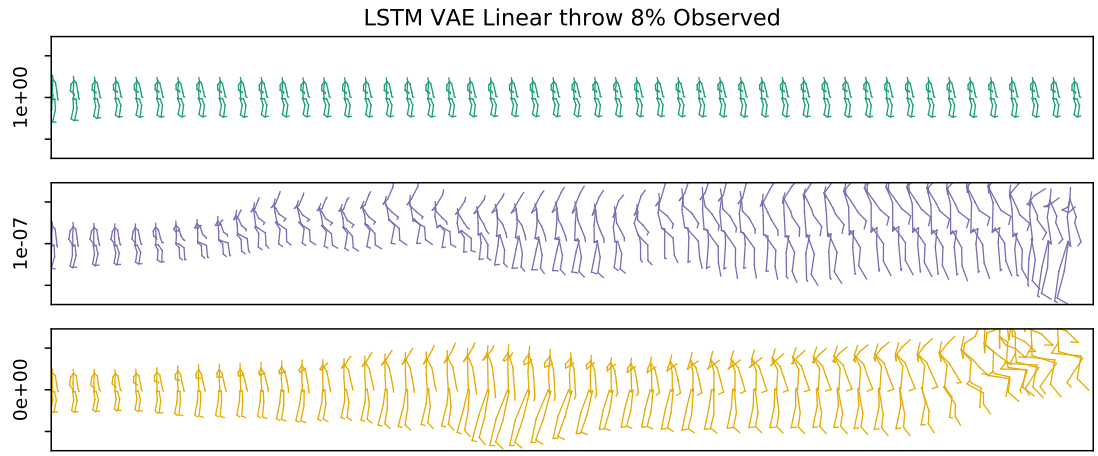



Figure 4.15.: Shows the same predictions made in figure 4.14 but this time the predicted trajectory was decoded with a linear decoder.

more basic or no AEs while in figure 4.17 you can see trajectories generated by the models using LSTM based AEs. For this experiment, we compare how well the models seem to reconstruct the *throw* action. We chose this action, because it is a relatively complex one and it is distinct enough that you can tell from a 2D plot when the action is performed. Also, we supply 8% of the trajectory that is to be predicted.

Looking at figure 4.16 you can see that the joint space ProMP and the linear AE ProMP have a couple of things in common. Both have good reconstructions from the first frame on. Both have the arch of the arm movement at around the same time, and both kind of stop working close to the end. This actually happens to the RNN version too, also at around the same time. Our best guess as to why this happens is because most demonstrations were finished at around that time so the ProMP couldn't really learn how to behave from thereon. The RNN also suffers from the problem of random initialization at the beginning. After the first couple of frames, it performs the action rather well, although the whole body position seems less harmonious than in the other two models.

Moving on to figure 4.17 you can see the trajectories generated by the LSTM based models. As expected both LSTM decoder models suffer from random initialization. Both non-variational models loose coherence at the end with one of them collapsing while the other one expands. We suspect that this is for the same reason as for the basic models. The main point of this figure is to show the trade-offs between using LSTM decoders and Linear decoders for AEs and VAEs. The main difference we noticed between using AEs and VAEs is that VAEs are a lot more static. This does go too far in many cases, like the

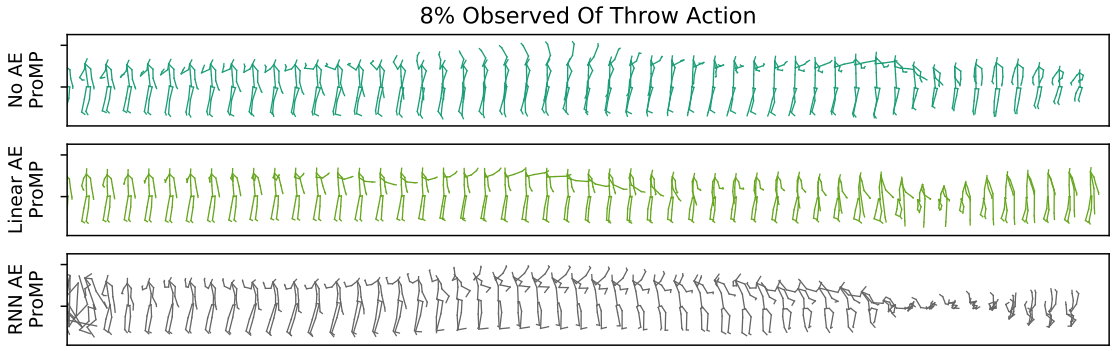



Figure 4.16.: Here you can see some samples of trajectories generated by some basic AE – ProMP combinations. The trajectory of the joint space ProMP took around 5 seconds to generate, while the other two took around 0.1 seconds each.

example you can see in figure 4.17. If you didn't know that the trajectory is supposed to be a throwing movement, one might interpret it as a person taking a selfie. Here the linear decoder helps a little bit. You can see a bit more of the throwing movement, although not anatomically correct, because the arm gets a lot longer.

The non-variational AE with the Linear decoder does not do a better job of predicting this motion than the LSTM decoder version. In fact, we'd argue that it's the worst of the four samples.

Even though the models with the Linear decoder don't do a whole lot better than the other models, in fact, they do worse in most cases, they do have two advantages over the others. Both are a consequence of the random initialization of the recurrent style models. The first, we mentioned before, is that the first few frames are reconstructed well. The second is that because there is no randomness in the decoding process, sequences could be decoded multiple times and each time the result would be the same. For all of the RNN based models, this is not the case and can actually make the difference between an awful reconstruction and a good one.

### 4.4.8. Conclusion

In this section, we conducted basic experiments to find out what hyperparameters to use for other experiments. We found out that to keep numerical instability to a minimum the ProMPs used for predictions had to have 25 basis functions with a scale of 0.001. That is quite a few basis functions and the time required to predict a motion definitely

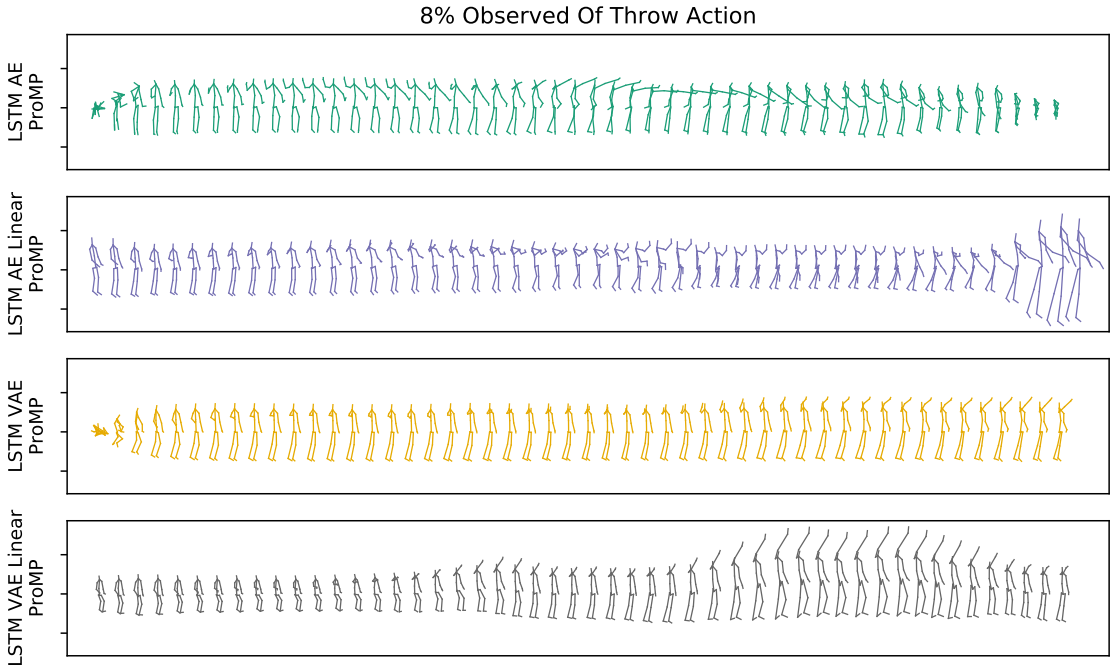



Figure 4.17.: Some predicted trajectories generated by the four LSTM based models. Notice how the models not using a linear decoder are more stable in the body size, but the first few frames are messed up.

would be lower with fewer basis functions. But even still using AE ProMPs, instead of joint space ProMPs, decreases the prediction time from approximately 5 seconds to around 0.1 seconds for a 50 frame sequence.

Afterward we compared how well the different AEs we explored in the previous chapter work together with motion prediction using ProMPs.

We discussed the viability of using VAEs with ProMPs for different KL Factors and concluded that small amounts of KL Loss work best, although high KL Losses produce lower prediction MSEs.

Looking at the quality of the predictions, it seems like using LSTMs as a decoder for predictions is the way to go. The only two advantages linear decoders have over the recurrent decoders are that the first few frames get decoded just as good as the rest and that multiple decodings result in the same sequence every time. The first is a non-issue in predicting movements because you can simply decode the input frames as well, an thus actually get reconstructions for all of the predicted frames. The second most likely won't matter and could be a disadvantage depending on the circumstance.

## 4.5. Motion Generation

Since the quality of a generated motion is quite difficult to quantify, we compare the ability of the different approaches based on how much they look like the intended movement. Because RNNs require input to output something, the first frame was randomly initialized for the LSTM-3LR baseline. ProMPs do not have this restriction, so we could simply sample from it directly. We use some of the AEs and VAEs introduced in the experiments section and we use the same denotation to describe them.

As we explained in the previous chapter, we used 25 basis functions and a scale of 0.001 for predicting future movements due to numerical instabilities within the ProMP. This is only an issue when we need to condition the ProMP before we sample from it. For movement generation, this is thus a non-issue. So for all of the ProMPs trained in a latent space, we now use only 7 basis functions with a scale of 0.05. The joint space ProMP still requires the original settings to produce good results, since its dimensionality is a lot higher. The number of basis functions was before the largest holdback for the ProMP. As you can see in figure 4.18 the times have drastically improved. All models became faster because there was nothing to encode, and in the case of the ProMPs also no necessary conditioning. Now there is a noticeable difference in how fast the different AEs decode

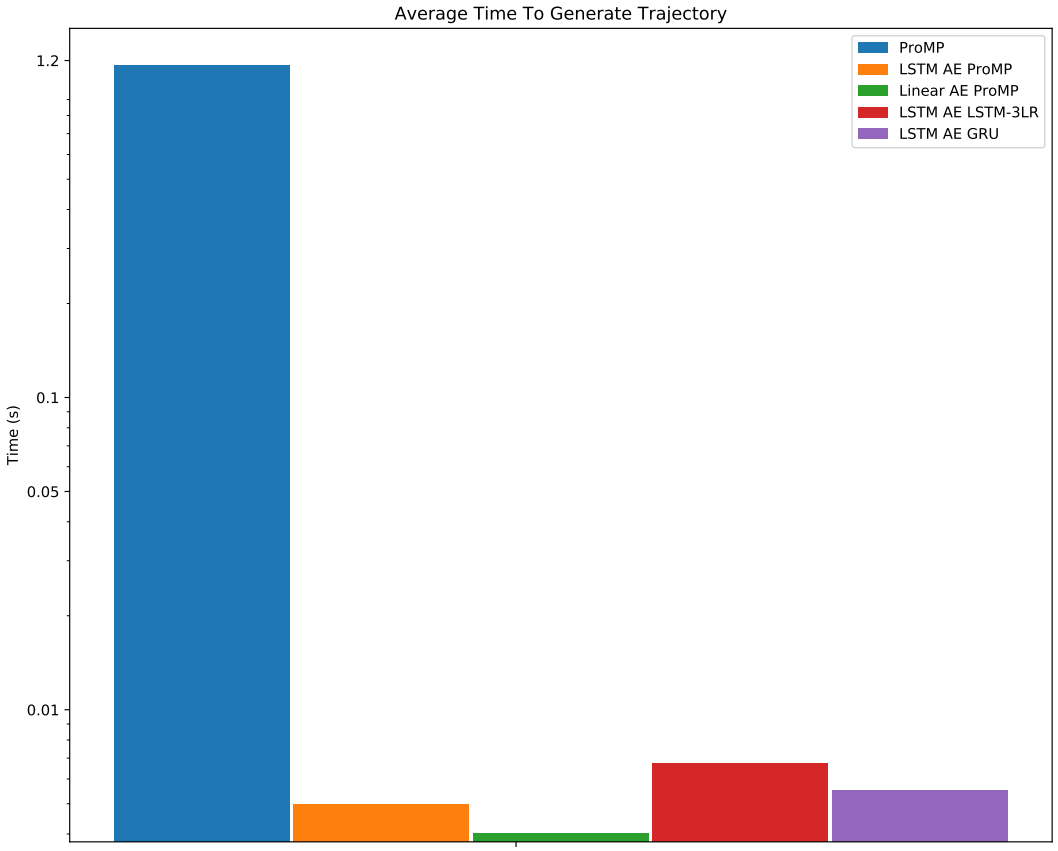



Figure 4.18.: Average time it takes each of the models to generate a 50 frame sequence. The time axis is again non-linear. For generations, the AE ProMP approaches outperform the baseline due to the smaller amount of basis functions (7).

the trajectory. Both AE ProMPs outperform the baselines now, and still vastly outperform the joint space ProMP.

To compare how well the different setups perform on generating motion we first look at how well they perform on generating *throw*, *take selfie*, and *salute* motions. As in the previous chapter, the generation was performed in the various latent spaces of the AEs (besides the ProMP trained without an AE of course).

You will find some sample movements generated from the models in the figures 4.19, 4.20 and 4.21. As in the previous qualitative experiments, the random initialization problem occurs here as well. The difference is that it can now be harmful. When predicting movements this problem is irrelevant in our opinion, because the input sequence can just be decoded with the generated sequence, making sure that by the time the predicted frames get decoded the hidden state is stable. The situation is a bit different in generation from scratch because there are no input sequences that could be decoded together with the actual generation. I'm sure there are ways around this issue, but as it stands right now this is a little bit of a disadvantage against the linear decoder versions.

As you can see in figures 4.19, 4.20, and 4.21 the AE and the LSTM AE with the linear decoder produce as expected good results right away. They also do seem to perform quite good in general, albeit much more conservative in how far they move. So the only action the linear decoder models struggled with a lot was the *throw* action. One instance where the linear decoder seems to make a large difference is when using the baseline LSTM-3LR to generate the movement. Most trajectories generated from the LSTM-3LR with an LSTM decoder could barely be considered of human shape, which is why it wasn't included in the figures.

The models using an LSTM as an encoder have in common that the size of the human body is more constant than without using one. The only exception to this is at the end of some of the trajectories when the bodies simply collapse.

One result of using VAEs is that the movements are more static, meaning that two frames next to each other differ less than other models. In some cases, you will see that there isn't any recognizable movement at all, like in the *salute* motion generated by the LSTM AE Linear ProMP. This seems to happen more often when the motion doesn't actually have that much movement in it. In terms of MSE it might be cheaper for the model to learn what the best mean standing pose is and just output that for the majority of inputs. Using VAEs isn't all bad though. When it doesn't get stuck in the same pose, the movements generated are very stable, but we'd argue that for most actions the VAEs in the way we

used them are just to restrictive to enable good generalization for motion representation. Even if the KL influence is small.

Another result of this staticity is that very dynamic movements turn into very unmotivated versions of the original. A good example of this is the *throw* motion generated by the ProMP with LSTM AE and the linear decoder version. The movement is generated well and it is possible to see what action was generated, but the skeleton seems a bit hesitant to let the hand get too far away. The same is true for the other two motions as well, but since they are rather static, to begin with it isn't that noticeable.

An interesting thing worth mentioning is that some of the movements have demonstrations where the person performing the task is sitting. So some of the samples retrieved from the ProMPs also look like the movement is performed while the person is sitting. This can be seen in some of the examples from the *take selfie* action. This clear distinction isn't made all the time though. Sometimes, like the ProMP not using an AE, the movement is performed in a weird semi-sitting position.

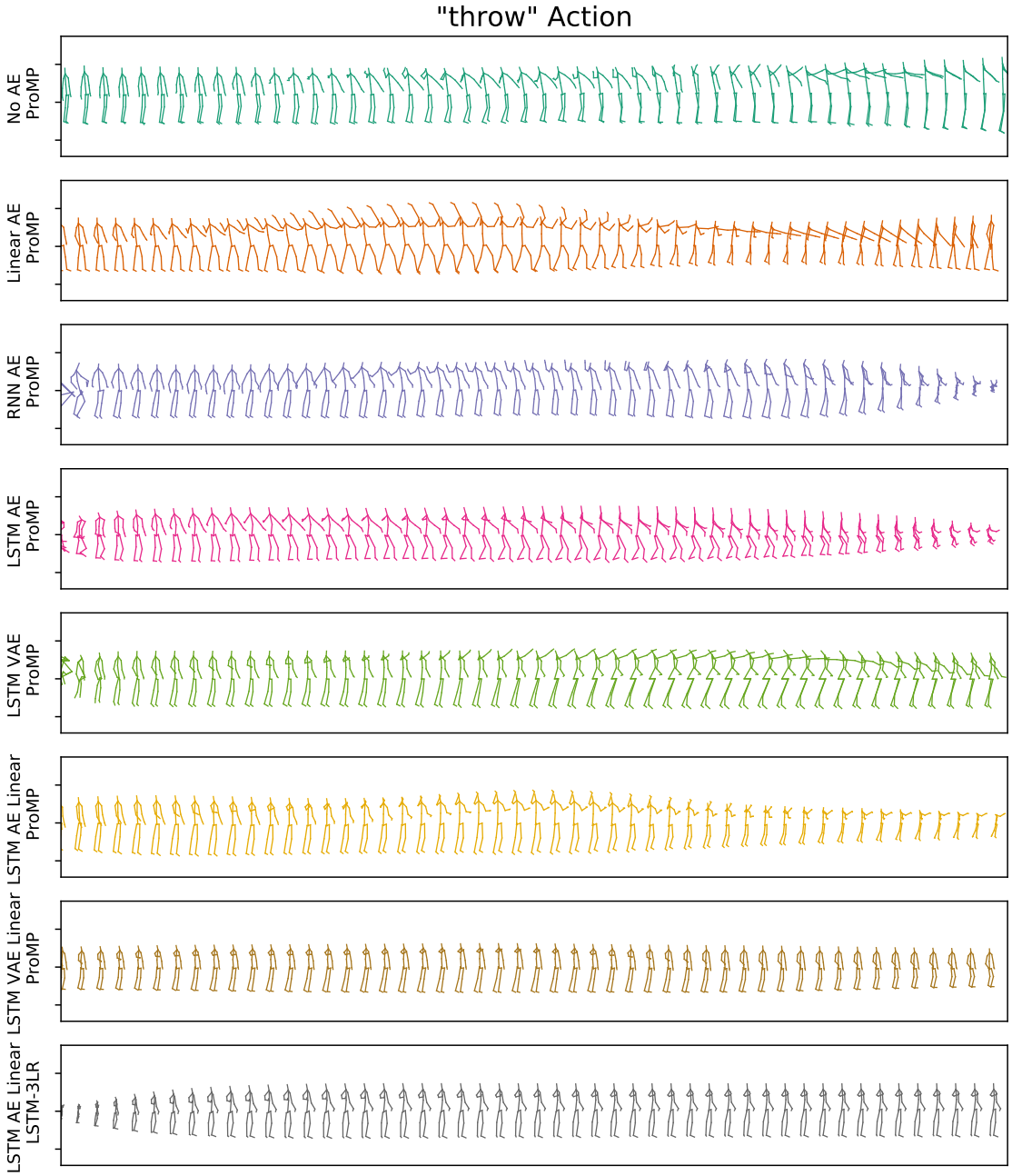



Figure 4.19.: Some samples of the *throw* action generated by various models.

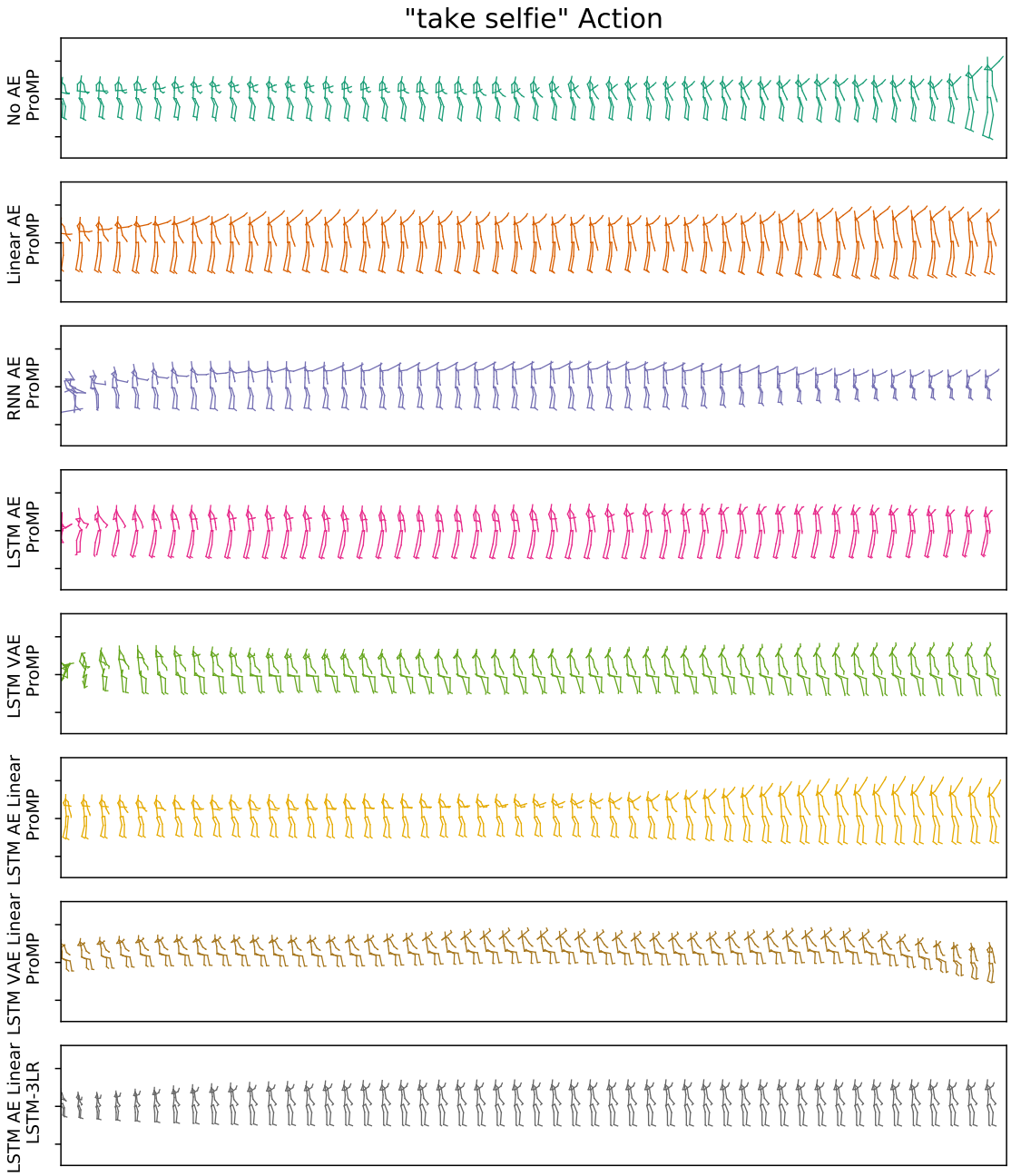



Figure 4.20.: Some samples of the *take selfie* action generated by various models.

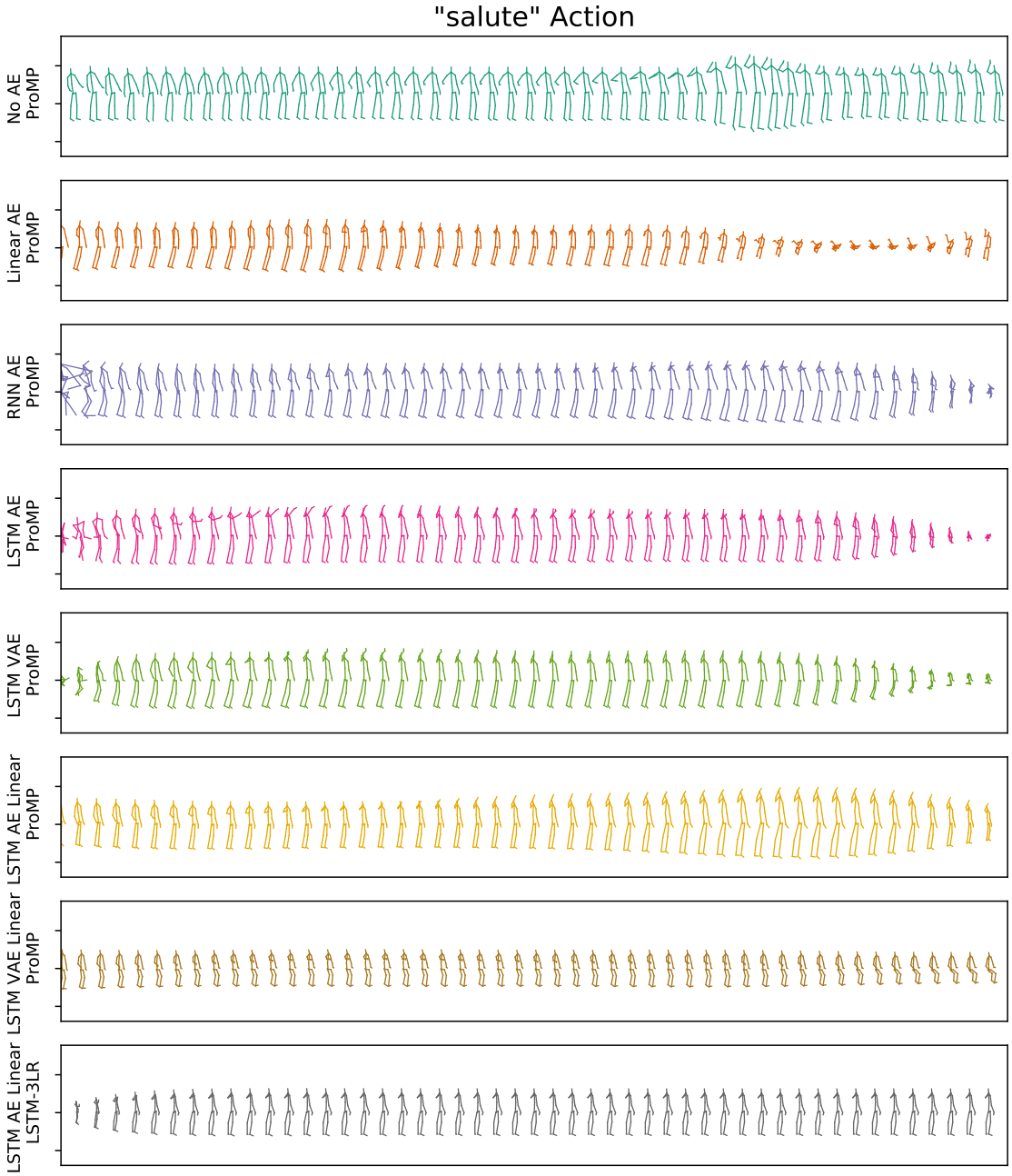



Figure 4.21.: Some samples of the *salute* action generated by various models.

# 5. Discussion

In this chapter, we visit all the experiments conducted and try to make sense of the results in a coherent manner. In the first experiments section, we looked at how different AEs perform on their own in terms of reconstruction MSE. The results of these experiments suggest that in terms of pure reconstruction MSE the linear AE is best suited for reconstructing skeletal data. The only model that got close to the MSE of the linear AE is the LSTM AE Linear model. The quality of these reconstructions in terms of MSE is far more inconsistent than the full linear AE and strongly depends on the random initialization of the LSTM.

Another experiment revealed how the KL Loss influences the reconstruction MSE of the two VAEs. For both of them, higher KL Loss influence means higher MSE, suggesting that the KL influence is to be kept low.

We also compared how some actions would look like in a 2D latent space and saw that the variational models map movements into tighter space. This is useful when demonstrations of the same action are mapped into similar spaces, and rather harmful when this happens for different actions. One possible way to take advantage of this behavior could be to train action specific VAEs, though we have not tested this since we wanted the AEs to be action agnostic.

In the second section, we took a look at movement prediction using ProMPs with the various AEs. The rationale behind this being that analyzing how well a model can predict future trajectories can give us useful insights into how well the models work and with what settings, while also allowing quantification of how well the models work.

Through experiments, we found out, that using a high number of basis functions together with a low scale for the ProMPs is necessary for prediction because otherwise, the conditioning of the ProMP became too numerically unstable. Here it is worth noting that because of this stability issue, we could not use the Von Mises basis function for the prediction task, because it was even more unstable for low scale values. We could have used it again in the generation generation section since the scale is higher there. The only

action that could be categorized as periodic is the *hand waving* action, and that action also includes a lot of stroke based motions. That might be the reason why there is not a noticeable difference in generated movements between the two basis functions. So for simplicity's sake, we used the Gaussian basis function for all actions.

With the settings found in the previous experiments, we could now test how well the different AEs work together with ProMPs to predict movement. After finding out that variational LSTM AEs do pretty well, we tested what the influence of the KL Loss would change about the prediction MSE in VAE-ProMPs. The results are the exact opposite when compared against the earlier results where we looked at the pure reconstruction ability of AEs by themselves. Exploring this a bit further by analyzing what the generated predictions look like, it turns out that high KL Factors lead to more restricted motion, but more stable poses. This reduces the prediction MSE but also reduces the value of the prediction significantly in terms of how well the essence of the action is restored.

From looking at visual representations of the predicted trajectories from different models, a for the generation of trajectories important advantage of the linear decoder models becomes apparent. They produce good reconstructions right away rather than taking a while at the beginning to balance out the random initialization like the LSTM decoders need to.

When we looked at the results of the experiments conducted for the target of motion generation, we of course saw these same results. The main difference is that this advantage is really important for generating motion, while it is rather neglectable for predicting motion because the first few decoded frames are the ones given as input and thus become irrelevant. This is not the case for generating motion since there is no input given to the models, and the whole movement may be important.

For the generation tasks, we were able to use much lower numbers of basis function and higher variances because we do not need to condition these ProMPs. This nullifies the issue of the numerical instabilities we experienced in the prediction experiments.

A direct result of this is that the time it takes to generate a motion is drastically reduced. The conditioning also takes a lot of time, which means that the ProMPs generate motions faster than they can predict them. This is why the baselines are faster for predicting motion while the ProMP based models outperform the baselines for generating motion.

# 6. Future Work

Here are a couple of topics that could further build on and improve the approaches used in this work.

## 6.1. Generative Adversarial Nets

The basic idea behind *Generative Adversarial Nets* (GANs) [17] is to have two models, a generative model $G$ and a discriminative model $D$. The GAN framework now pitches these two networks against one another. $G$ tries to generate data while $D$ learns to discriminate between data from the dataset and data generated by $G$. The idea now would be to use this framework to improve the generative capacity of the ProMP and the decoder. In this scenario, $G$ would be the combination between a ProMP and a decoder, while $D$ could be implemented in any number of ways.

## 6.2. Graph Neural Networks

The incorporation of *Graph Neural Networks* (GNN) [18] into the encoders and decoders could help to reconstruct more realistic skeletons from the latent space. This could prevent phenomenons like the stretching of an arm to perform a throwing motion. Here the skeletons could be viewed as a directed graph where the root is the middle of the upper body [19]. This helps to further encode the dependencies between different bones and joints. This can then express for example that the position of the ankle is also dependent on where the knee and hip joints are.

## 6.3. Nonlinear Principal Component Analysis

Another approach we could have used to transform the trajectories into the latent space is the Nonlinear Principal Component Analysis [20, 21, 22]. Put simply, this adds a nonlinear layer to the encoder and decoder networks of a linear AE. The nonlinearity of this approach, and the nonlinear latent space created by it, could enable better reconstructions. This would have the advantages of using a linear AE with ProMP, like good decoding of the first few frames, while also possibly more accurately representing the skeleton in the latent space.

# Bibliography


[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[2] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic Movement Primitives," in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2616–2624, Curran Associates, Inc.

[3] O. Dermy, M. Chaveroche, F. Colas, F. Charpillet, and S. Ivaldi, "Prediction of Human Whole-Body Movements with AE-ProMPs," in *IEEE-RAS 18th International Conference on Humanoid Robots (HUMANOIDS 2018)*, pp. 572–579.

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

[5] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes,"

[6] C. Doersch, "Tutorial on Variational Autoencoders,"

[7] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*.

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

[9] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," vol. 28, no. 10, pp. 2222–2232.

[10] J. Martinez, M. J. Black, and J. Romero, "On Human Motion Prediction Using Recurrent Neural Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4674–4683, IEEE.

[11] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, “Recurrent Network Models for Human Dynamics,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4346–4354, IEEE.

[12] J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio, “A Recurrent Latent Variable Model for Sequential Data,”

[13] M. Chaveroche, A. Malaisé, F. Colas, F. Charpillet, and S. Ivaldi, “A Variational Time Series Feature Extractor for Action Prediction,”

[14] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Using probabilistic movement primitives in robotics,” vol. 42, no. 3, pp. 529–551.

[15] A. J. Ijspeert, J. Nakanishi, and S. Schaal, “Learning Attractor Landscapes for Learning Motor Primitives,” in *Advances in Neural Information Processing Systems 15* (S. Becker, S. Thrun, and K. Obermayer, eds.), pp. 1547–1554, MIT Press.

[16] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, “Ntu rgb+d: A large scale dataset for 3d human activity analysis,” in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[17] 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 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc.

[18] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The Graph Neural Network Model,” vol. 20, no. 1, pp. 61–80.

[19] L. Shi, Y. Zhang, J. Cheng, and H. Lu, “Skeleton-Based Action Recognition With Directed Graph Neural Networks,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7904–7913, IEEE.

[20] M. Scholz and R. Vigario, “Nonlinear PCA: A new hierarchical approach,” p. 6.

[21] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” vol. 37, no. 2, pp. 233–243.

[22] M. Scholz, M. Fraunholz, and J. Selbig, “Nonlinear Principal Component Analysis: Neural Network Models and Applications,” in *Principal Manifolds for Data Visualization and Dimension Reduction* (A. N. Gorban, B. Kégl, D. C. Wunsch, and A. Y. Zinovyev, eds.), Lecture Notes in Computational Science and Enginee, pp. 44–67, Springer.

# A. Appendix

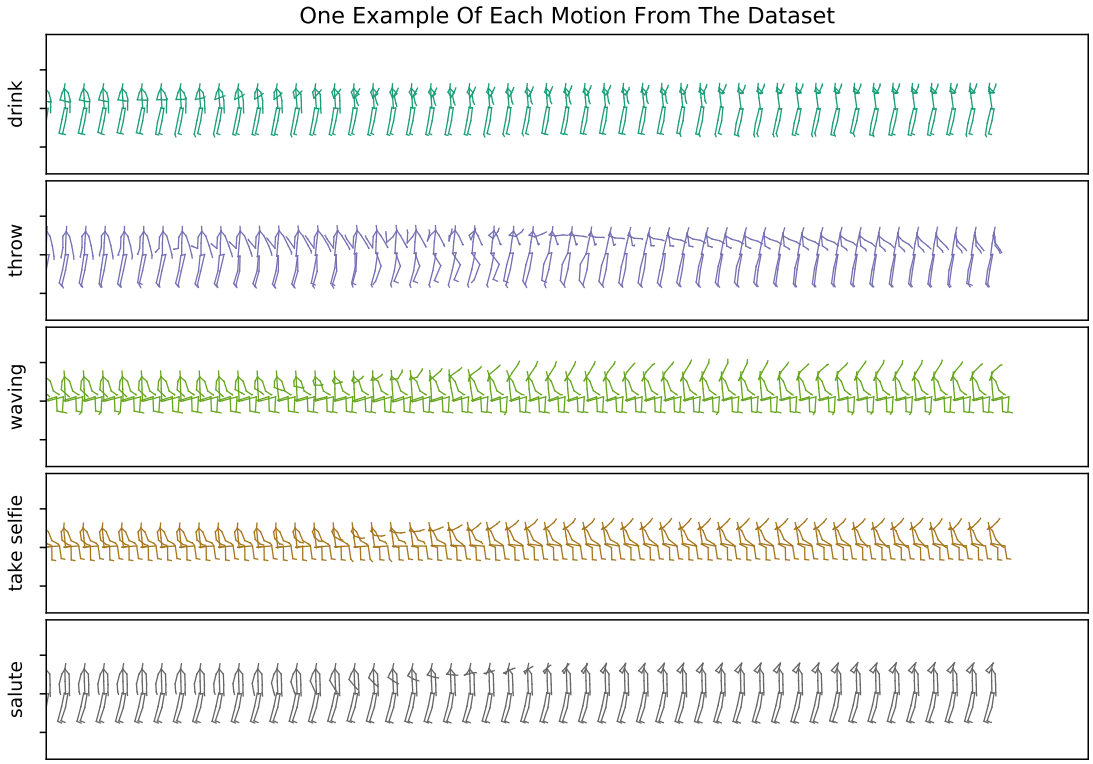



Figure A.1.: Here you can see an example of what each action might look like.

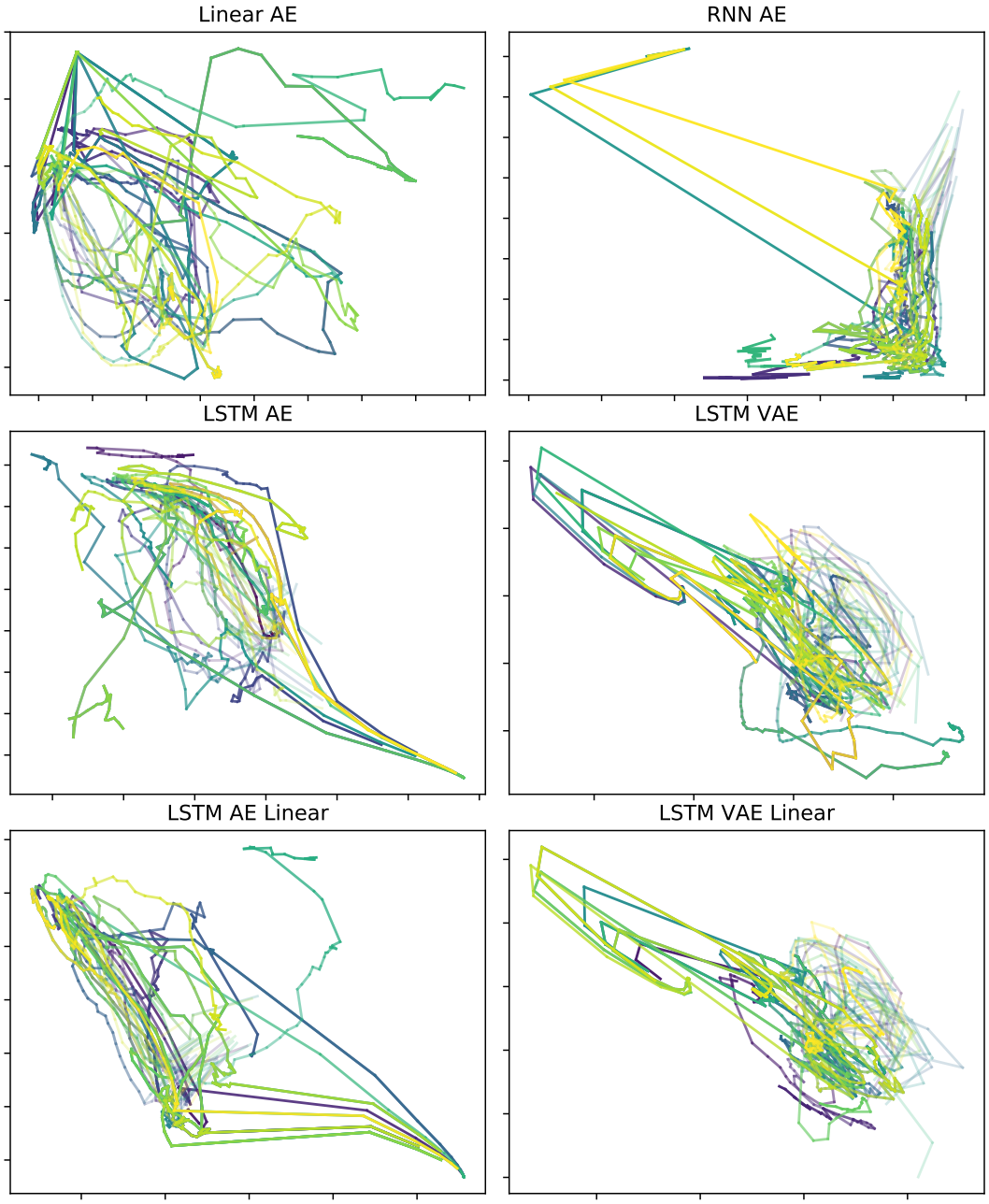



Figure A.2.: 2D latent space generated by different AEs for the *throw* action.

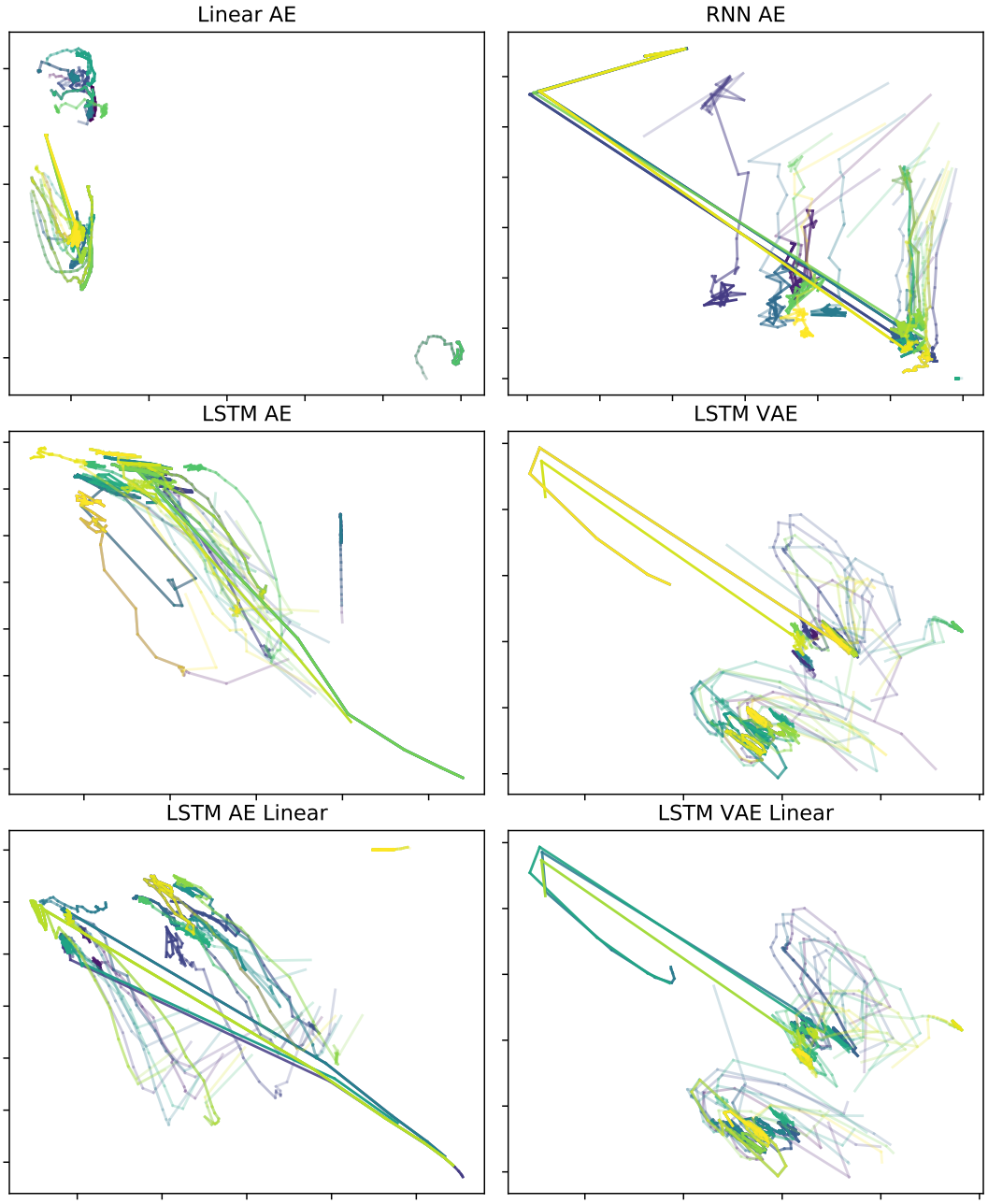



Figure A.3.: 2D latent space generated by different AEs for the *waving* action.

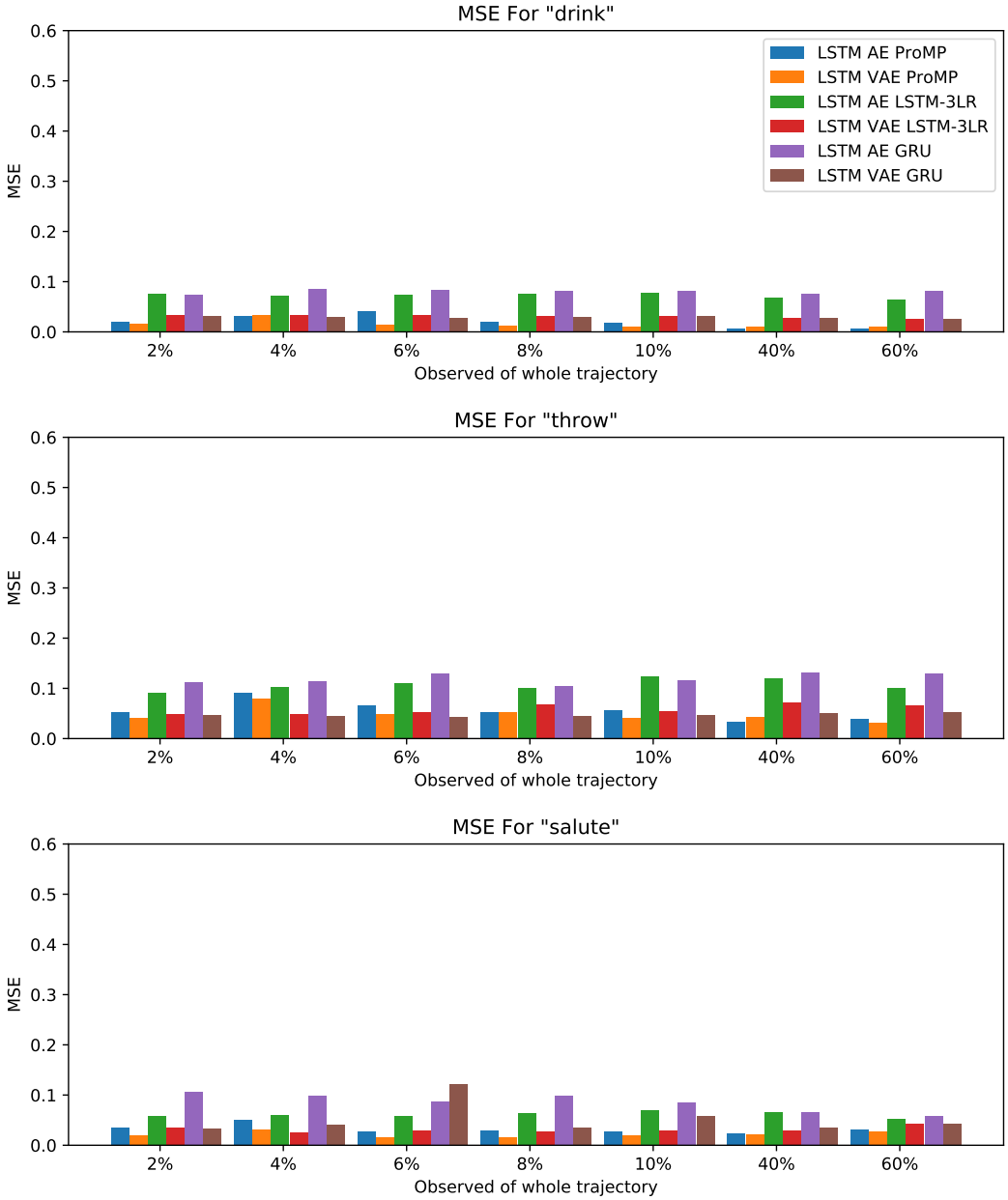



Figure A.4.: Comparison of prediction MSE between different models.

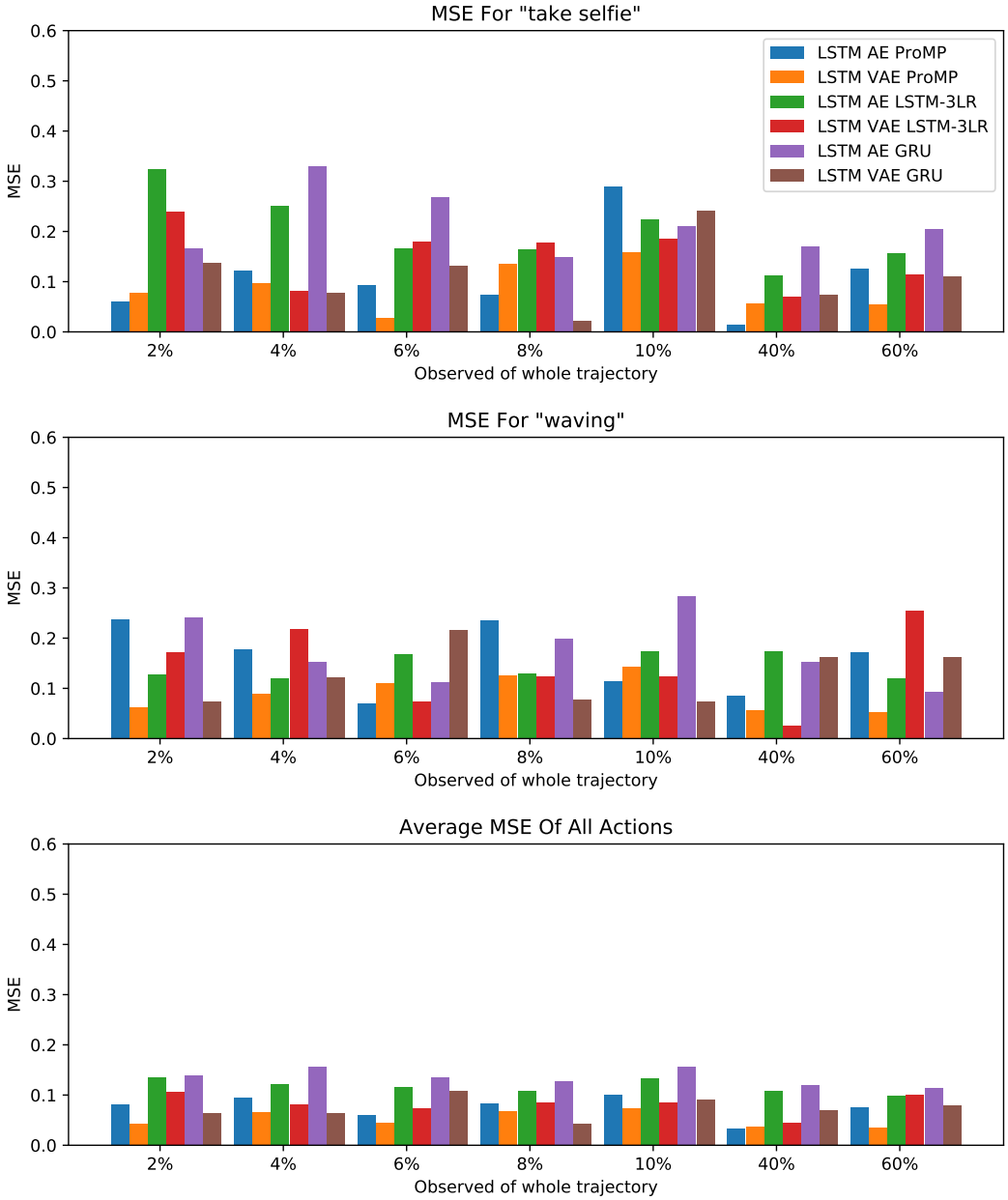



Figure A.5.: Continuation of A.4.

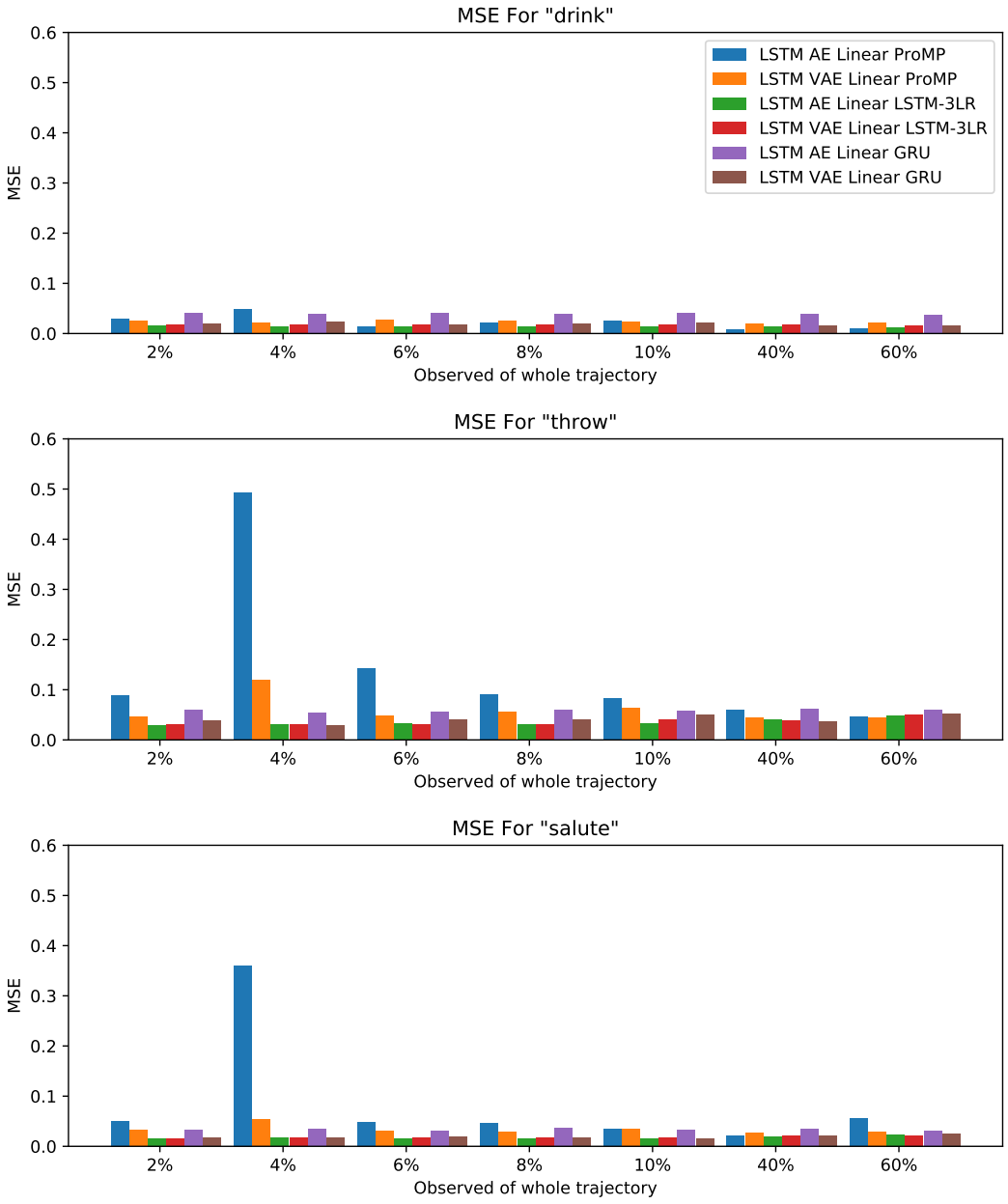



Figure A.6.: Comparison of prediction MSE between different models using a linear decoder.

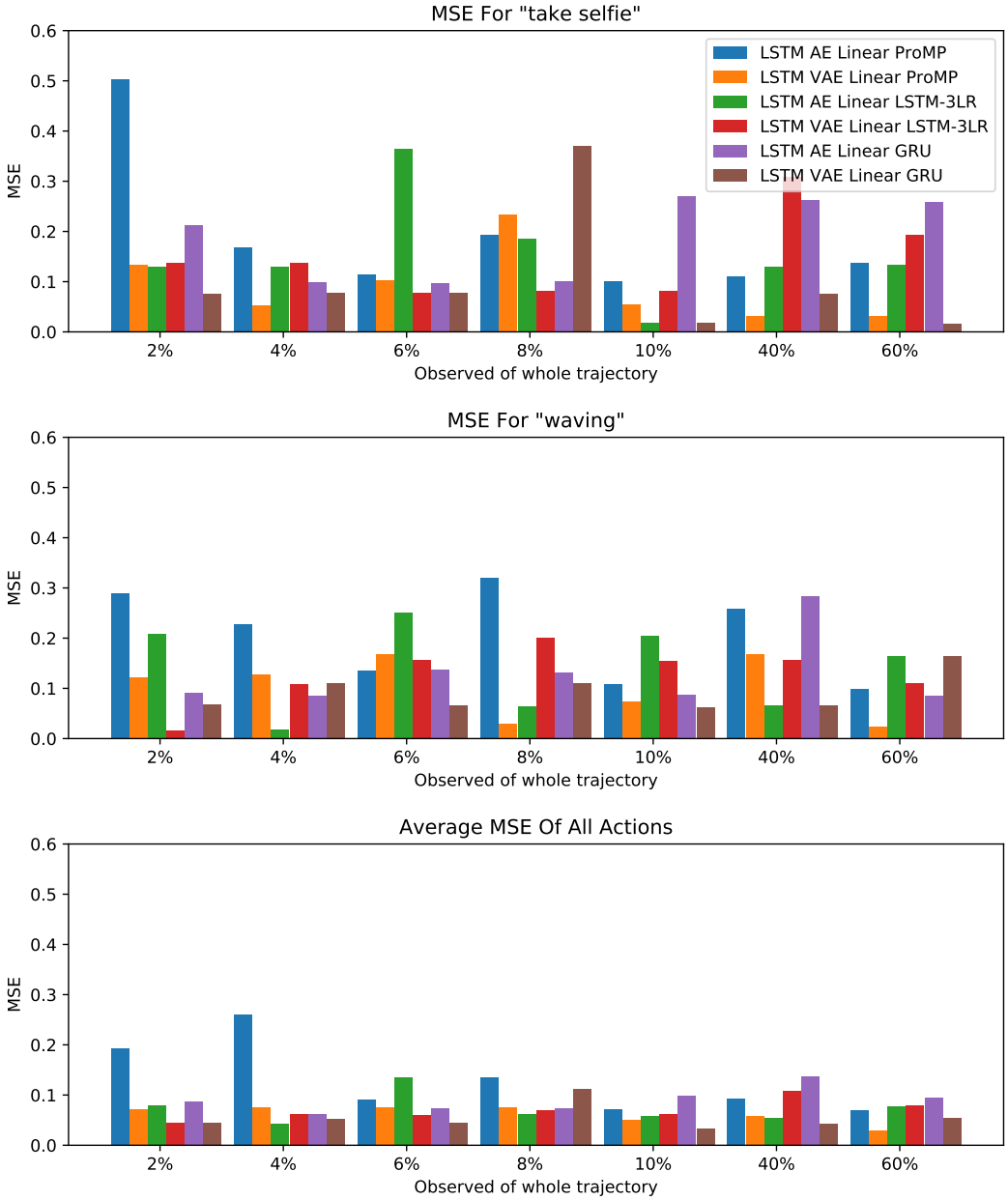



Figure A.7.: Continuation of A.6.