Text-Driven Head Motion Synthesis Using Neural Networks

Bror Tao Sjørslev Bojlén

4th Year Project Report
Cognitive Science
School of Informatics
University of Edinburgh

2017

Abstract

Head motion plays an important nonverbal role in face-to-face communication. In the literature on animated talking agents, there is some work on speech-driven head motion synthesis, but little on text-driven synthesis in spite of the fact that the semantic content of an utterance is an important contributor to head motion. We present an evaluation of different neural network architectures, an analysis of hyperparameters, and finally a text-driven, deep neural network-based system for head motion synthesis. The proposed model performs well in subjective evaluations, and we show that this is in part because of its access to the meaning of words in input sentences.

Acknowledgements

I would like to thank Dr. Hiroshi Shimodaira, my supervisor, for his support throughout this project. His guidance and advice has been invaluable. I would also like to thank all participants in the subjective evaluations for their kind help.

Table of Contents

Chapter 1

Introduction

In human face-to-face communication, speech is only one way in which we share information. There are many nonverbal signals that add nuance, show emotion, and aid understanding. These signals include facial movements (e.g. raising one's eyebrows or smiling), eye movement, hand gestures, and other body movements such as leaning in. One nonverbal signal that has received comparatively little research attention is head motion. It is an important signal, however: animated talking heads with no head motion look robotic, regardless of the quality of facial animations. Talking heads that move randomly appear to float over the background, giving them an eerie appearance [1]. In addition to making agents more lifelike in general, head motion is used to signal agreement or disagreement (through nods or head shaking), questioning (through head tilts), or emphasis (through more sudden movements, often downwards or diagonally) [2]. Head movements are also be used to indicate that the agent is listening and the agent's emotional state [3]. Although a lot of work on animated talking agents has been put into lip motion (i.e. lip-syncing) and facial expressions, head motion is crucial. Munhall *et al.* showed that, in addition to the above, head motion improves speech perception [4], and Hill & Johnston found that head motion alone is enough to discriminate speaker identity and gender [5]. Heylen presents a comprehensive review of the functions of head motion in [2].

Animated talking heads provide an interesting opportunity for human-computer interaction. In recent years we have seen rapid growth in the availability and use of digital assistants, such as Apple's Siri, Amazon's Echo, or Google's Assistant. Users usually interact with these digital assistants through speech and/or text. Animated heads could make these assistants more lifelike and approachable than disembodied voices. This could improve usability, particularly for less technically savvy users. Head motion is important here, as are facial animations, since they can can indicate emotions and active listening in ways that audio alone cannot. Synthesized head motion also has applications in video game and film animation where it could lower the workload of animators by either generating a rough "sketch" to work from or, if the synthesized motion is lifelike enough, by entirely automating the process of animating head motion. Other applications of head motion synthesis include automated customer service agents and agents in kiosks.

Most of the existing work on head motion synthesis is based on auditory speech features. Auditory features present a good starting point, since speech and head motion are highly correlated [6], though it is not a one-to-one mapping (as noted in Yehia *et al.* [7]). However, a ”ground truth” speech signal from which to generate motion is not always available, for example in conversational chatbots or digital assistants. In these cases, one option is to synthesize speech using a text-to-speech (TTS) system and then generate head motion based on this. However, this makes the TTS system a bottleneck in terms of head motion quality. For example, say that we want to synthesize speech and head motion for a question. If the TTS system does not capture the inquiring nature of the sentence (e.g. by raising the pitch at the end of the utterance), then the head motion cannot reflect it, either. Additionally, there may be nuances that a statistical model can pick up from text more easily than from audio. For instance, head motion can have a narrative function. Speakers may use head motion (especially turning left or right) to indicate a shift from direct to indirect discourse, as in when relaying what another person said. Head motion is also used to separate items when listing something [8]. These types of head motion are dependent on the semantic content of a sentence, which might not be captured by auditory features. Graf *et al.* found that the prosodic structure extracted from text is highly correlated with head movement [9]. Prosody refers to the intonations of speech, e.g. pauses, pitch, volume, et cetera. This result bodes well for text-based approaches to head motion synthesis.

This project presents a data-driven and text-driven approach to head motion synthesis using neural networks. Most of the existing work on text-driven models uses rule-based systems and it is common to base these models on prosodic features extracted from the text, rather than on the text itself (for instance in Zhang *et al.* [10]). In contrast, we use deep neural networks and train directly on the text. Our final model performs well in subjective evaluations. Chapter 2 gives background information for this project and chapter 3 outlines our approach. In chapter 4 we present the experiments we performed to optimize the many hyperparameters that go into building a system like this. In chapter 5 the best models are compared and evaluated through subjective evaluations, and we discuss the results. Finally, chapter 6 summarizes the results and outlines some promising areas to pursue in future work. The main contributions of this project are:

- A neural network-based model for text-based head motion synthesis. Its generated head motion is rated as ”good - fairly natural” in subjective evaluations.
- An analysis of, and experiments about, some of the design decisions behind such a model, e.g. model architecture and ways of representing the input.
- Experimental evidence that having access to the semantic meaning of a sentence improves synthesized head motion, highlighting the potential of text-based (rather than audio-based) approaches to head motion synthesis.

Chapter 2

Background

2.1 Previous work

In the literature on talking agents, there is a lot of work on synthesising and synchronising lip motion. Some of this is text-driven, for example in Zoric *et al.* [11]. However, the text is first synthesized to speech, and the animation is based on this audio.

There is some work on synthesizing head motion from auditory features. Many projects have used Hidden Markov Models (HMMs) [3, 12]. More recently, progress has been made on using neural models [13]. To the best of our knowledge, the work that is most similar to this project is [14], in which Haag & Shimodaira train bidirectional Long Short-Term Memory (LSTM) neural networks using stacked bottleneck features with auditory data as the input.

Busso *et al.* present a model for head motion synthesis based on Hidden Markov Models [12]. In subjective evaluations, their model scored as well as real recorded motion in naturalness. Head poses were grouped into discrete clusters, and HMMs taking audio features as input were used to model sequences of these poses. The most likely sequence of head poses was then interpolated and smoothed to form a continuous sequence. Though their model performed well in subjective evaluations, it was based on a dataset of a single actor speaking lines with different emotions. It is not clear how well this generalizes to different personalities or contexts. In addition, both speech and head motion are sequential types of data where each timestep is dependent on the sequence leading up to it – in other words, the Markov property of HMMs may limit the quality and expressiveness of synthesized motion.

Ding *et al.* also generate head motion based on speech features, but they do so using neural networks [13]. They use a dataset of audio and video recordings of a single news anchor presenting the news, along with recordings of ten other newsanchors to pre-train their model. To pre-train the model, the researchers stack Restricted Boltzmann Machines (RBMs) to form a Deep Belief Network. RBMs are trained one at a time, and finally a target layer is added at the end to form a deep neural network that can be fine-tuned via standard backpropagation. The researchers found that using data from many newsanchors for pre-training improved their model's performance, while using it in the main training phrase degraded it. They hypothesize that this is because of differences in head motion patterns among different speakers. This supports the finding in Hill & Johnston [5] that head motion is speaker-dependent. However, without subjective evaluations it is difficult to evaluate how well their model performed compared to real head motion.

Haag & Shimodaira also trained a deep neural network on speech features, but they used a bidirectional recurrent network [14]. First, a deep feed-forward neural network with a bottleneck layer was trained on data from multiple speakers. A bottleneck layer is a layer in a neural network that has a smaller number of neurons than the others. The activations in this layer (”bottleneck features”) are a low-dimensional representation of the input, the idea being that it captures the most salient parts. This bottleneck layer is then used to generate bottleneck features which are used in conjunction with speech features in a separate recurrent neural network. They found that bidirectional recurrent networks outperform feed-forward networks in objective scores (as in Ding *et al.* [13]), but not in subjective evaluations. In their subjective evaluations there is no comparison to real head motion, making it difficult to evaluate the current state of neural network-based approaches to head motion synthesis.

2.2 Recurrent neural networks

Standard feed-forward neural networks map n-dimensional input vectors to m-dimensional outputs by passing the input through layers of ”neurons”. Each layer l is parameterised by a weight matrix $W^{(l)}$, a bias vector $\mathbf{b}^{(l)}$ and a nonlinear activation function $g^{(l)}$. A layer takes the previous layer's output as its input. The input vector $\mathbf{h}^{(l-1)}$ to a layer l is multiplied by the weight matrix and added to the bias, then passed through the activation function in order to give the layer's output $\mathbf{h}^{(l)}$:

$$\mathbf{h}^{(l)} = g^{(l)}(W^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$$

The sigmoid function is commonly used as an activation function. The final layer's output $\mathbf{h}^{(L)}$ (a vector of size m) is compared to the correct response $\mathbf{y}$ (also a vector of size m) through some loss function, commonly mean squared error for regression problems:

$$E_{MSE} = \frac{1}{n}\sum_{i=0}^{n}\mathbf{h}_i^{(L)} - \mathbf{y}_i$$

Neural networks are trained using backpropagation, in which this loss function is differentiated with respect to the weights in the network. These weights are then updated accordingly to minimize the error.

Recurrent neural networks (RNNs) generalize this model to sequences. They do so by taking a sequence of vectors $\mathbf{I}_1, \mathbf{I}_2, ..., \mathbf{I}_T$ as input in succession for T timesteps and by adding connections from each layer $\mathbf{h}_t^{(l)}$ to itself in the next timestep $\mathbf{h}_{t+1}^{(l)}$. The output of a layer at a given timestep is, in a sense, an internal state that allows the network to store temporal dependencies. See Fig. 2.1 for a visualization of an RNN with two hidden layers over three timesteps t_1, t_2, t_3. The RNN's output is the sequence of vectors $\mathbf{O}_1, \mathbf{O}_2, ..., \mathbf{O}_T$.

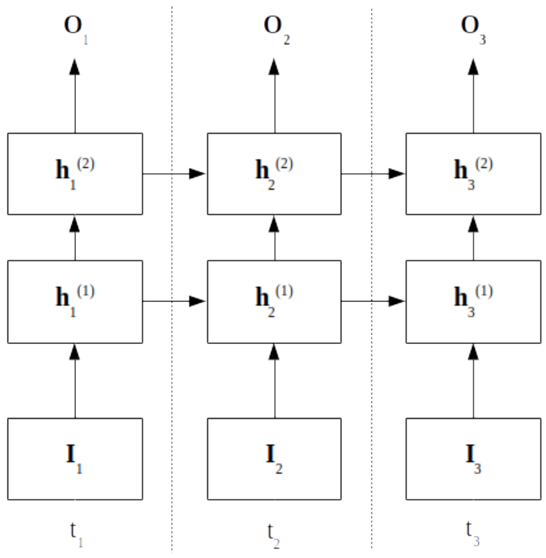


Figure 2.1: A recurrent neural network

At each timestep, the network only has access to the data before that timestep. However, future data is often useful. In order to use future data, we can use bidirectional RNNs [15]. In bidirectional RRNs, each recurrent layer is extended such that one part of it views the data in the ”correct” order, and another part views it in reverse order. Forward and backward states do not pass their inputs to one another. This allows the network to use data from both the past and the future when predicting for a timestep.

Long-term temporal dependencies pose a problem. The recurrent extension of backpropagation, backpropagation through time, unrolls the network over the timesteps [16]. This means that the outputs in early timesteps are passed through the activation functions many times, and when differentiating, their gradients can become very small – the ”vanishing gradient” problem [17]. Similarly, if a gradient is > 1, it can ”explode” when unrolled over many timesteps. In addition, there is a recency bias – timesteps closer to the current one exert a larger influence than timesteps further away.

In order to get around these limitations, we use Long-Short Term Memory (LSTM) cells instead of plain recurrent neurons in our networks. In brief, these cells add analogue gates – one for input, one for output, and a forget gate. These gates determine whether the internal cell state is written to, returned, or forgotten, respectively. Because these gates are analogue, they are differentiable and we can still use backpropagation [18]. Now, the internal cell state can carry information over many timesteps without losing it to vanishing gradients.

2.2.1 Encoder-decoder models

One limitation of recurrent neural networks is that the input and output sequences must be of the same length. At each timestep, the model takes an input vector and returns an output vector. One architecture that gets around this, and that has achieved very promising results in the domain of machine translation, is an encoder-decoder network [19]. This model consists of two recurrent neural networks – an encoder network and a decoder network that are trained jointly. First the input sequence is fed through the encoder network. The encoder's final output $\mathbf{O}_T^{enc}$ is sometimes referred to as the "thought vector", the intuition being that it encodes the meaning of the entire input sequence. This thought vector is then passed as the initial input to the decoder network [20]. During prediction, the decoder network's output $\mathbf{O}_t^{dec}$ at each timestep t is passed as the input to the next timestep. See Fig. 2.2 for a visualization.

This means that the input and output sequences can have different lengths. In machine translation, encoder-decoder models map sequences of words in one language to words in another language. The decoder stops predicting words when a special "<STOP>" token is returned. In this domain, we are not operating on discrete output tokens but rather on continuous sequences of head motion. Therefore, the decoder keeps returning outputs until we stop it. There are few (if any) applications of head motion synthesis that do not have audio of the output sentence available, whether through a text-to-speech system or a recording. Therefore, we stop the decoder when it has returned enough timesteps to create head motion for the associated output audio.

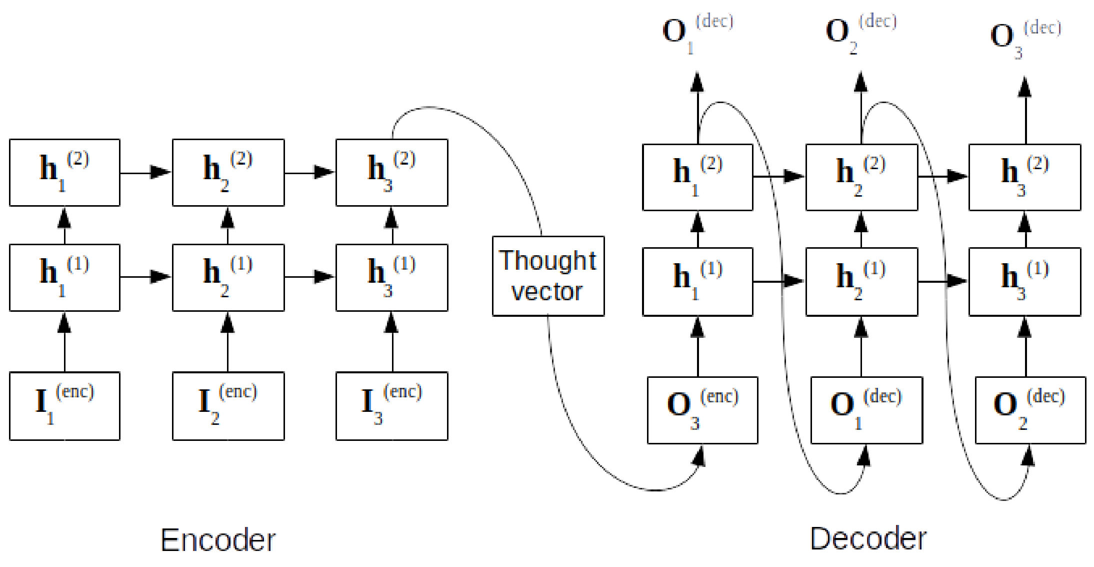


Figure 2.2: An encoder-decoder model

2.3 Word embeddings

Word embeddings are a way of representing words as real-valued vectors. In this project, they are used to represent the input sentence to our model. Word embedding models are based on word co-occurences in large corpora.

Each word in the vocabulary is represented as a k-dimensional vector. Words that are similar in meaning tend to have vectors that point in roughly the same direction, so that the closest vectors (in terms of cosine similarity) to e.g. the word ”frog” are the vectors of ”toad”, ”lizard”, and so on. In addition, word vectors can also represent relationships between words. For example, the vector from ”queen” to ”king” is similar to the vector from ”princess” to ”prince”. In this way, the semantic meaning of a word is represented in continuous vector space.

There are several different pre-trained word embeddings available. One that is commonly used is GloVe (”Global Vectors”) [21]. The GloVe model is trained to learn word vectors in which the dot product of two words’ vectors $(\mathbf{v}_1, \mathbf{v}_2)$ is equal to the log probability of the words’ (w_1, w_2) co-occurrence in the training corpus:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \log p(w_1|w_2)$$

We chose to use GloVe in this project because there are pre-trained models available based on several different datasets – Wikipedia and news articles, aggregated websites, or posts from the Twitter social network. We believed that the Twitter corpus would reflect conversational speech more closely and thus give better results – see section 4.2 for details of our investigation into this.

All words that are not in the embedding’s vocabulary, and thus do not have a vector representation, are all represented by the same ”`<UNKNOWN>`” vector. To avoid unknown words, we could have trained our own word embeddings on the dataset we used. However, we deliberately chose not to do so due to our dataset’s small size. Word embeddings are usually trained on corpora consisting of billions of tokens and with vocabularies of millions of words. In contrast, the dataset we used has just 60 000 tokens and a vocabulary of less than 4000 words.

2.4 Euler angles

Euler angles represent rigid body rotations by three partial rotations around the x, y, and z axes. Euler angles can be either intrinsic or extrinsic – in the dataset used in this project, they are intrinsic, meaning that each partial rotation rotates the entire coordinate system.

In the dataset used, the order of rotations is x, z, y. As an example, say that a given rotation is represented by the vector $[\psi, \theta, \phi]$ where $\psi, \theta, \phi \in \mathbf{R}$. First, the entire coordinate system (and the rigid body) are rotated around the x-axis by ψ degrees. Then, around the z- and y-axes by θ and ϕ degrees, respectively. Fig. 2.3 shows how the final orientation is reached by performing these rotations one at a time.

In the dataset used, head motion is represented by sequences of Euler angles. The set of Euler angles at each timestep is absolute, i.e. they represent an orientation based on a rotation from the same starting orientation.

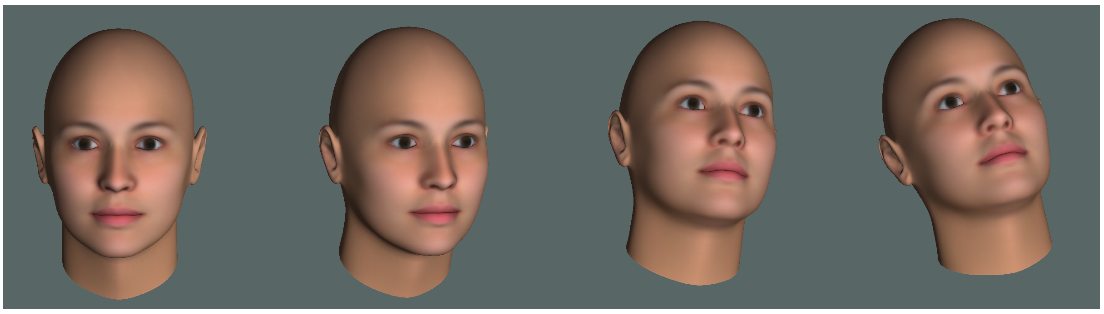


Figure 2.3: How Euler angles represent an orientation through a series of rotations

Chapter 3

Approach

3.1 Overview

Our proposed system can be seen in Fig. 3.1. The boxes inside the dotted rectangle represent data that we did not create ourselves, i.e. the dataset of recorded head motion (see section 3.2) and the pre-trained word embeddings. In brief, we first preprocess the data as detailed in section 3.4. This data is then used to train our models, the most promising of which is a deep recurrent neural network.

We performed several experiments to find the optimal way of representing the data, of training the model, and the best model architecture (chapter 4). We then compared two models in subjective evaluations (chapter 5).

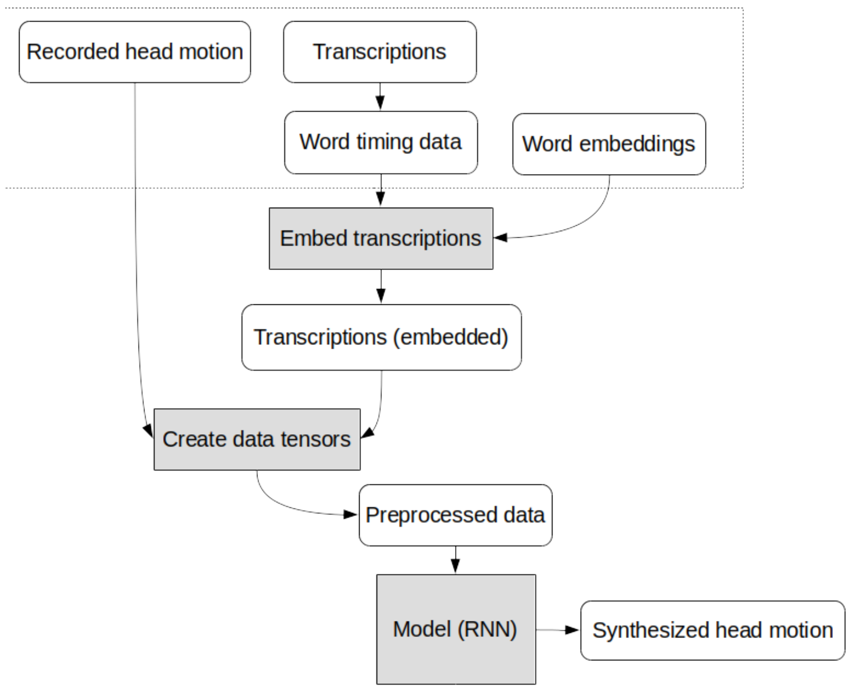


Figure 3.1: The proposed system

3.2 Dataset

In this project, we use the *University of Edinburgh Speaker Personality and MoCap Dataset* [22]. This dataset contains head motion, audio and transcriptions for 13 actors speaking in a casual, conversational setting. Several sessions were recorded for each actor. In each session, the actor was asked to act extroverted, introverted, or neutral. Because of the conversational nature of the recordings, this dataset is qualitatively different from those used in previous studies on head motion synthesis [12, 13]. These recordings show a lot of expressive motion, and they are more informal in nature than e.g. the recordings of newsanchors in Ding *et al.* [13]. Dall *et al.* found that recordings of conversational speech were consistently rated as more natural than recordings of read-aloud text in subjective evaluations [23]. If their findings generalize to head motion as well as audio, this dataset may allow for more natural synthesized head motion. In all, there is about 10 hours of recorded motion.

In addition to head motion, the dataset also contains motion of the upper body – 26 markers were placed on the actors' head, chest, shoulders, elbows, and hands. For this project, we will only use the motion from the 4 markers placed on the actors' heads. This motion is represented in six dimensions: rotations represented as Euler angles and translations along three axes.

Busso *et al.* found that the patterns of head motion associated with emotional speech are different from those associated with neutral speech [3]. For this reason, we only train our models on sessions where the actor acted extroverted under the assumption that this is more expressive than data from neutral or introverted sessions.

Hill & Johnston [5] found that head motion is speaker-dependent – in fact, it is enough to individuate speakers. However, we decided against training our models on individual speakers and rather opted to train on all extroverted sessions. There are two reasons for this. The first is that in our experiments, we found performance improvements from using more data (see section 4.3). This is not an uncommon result in machine learning [24]. The second reason is that though different people have different patterns in head motion, at this stage we are trying to model more general trends (e.g. downward nods for emphasis). This is, once again, not uncommon. Take the MNIST dataset of handwritten digits [25]. It contains digits written by many different people, even though they have different styles of handwriting. Models trained on this dataset learn the universal trends, and individual handwriting patterns can be seen as a form of noise. Though universal patterns in head motion may be less well-defined than in handwritten digits, a similar principle applies in this case.

3.3 Features

The input to our model is text. There are different ways to represent this in a way that functions as input to a neural network.

One option is to represent words as one-hot encoded vectors where all elements but one are equal to 0, and one element – corresponding to the index of a word – is equal to 1. This is the approach used in most neural machine translation systems. However, this would require the model to learn the semantic meaning of words based only on our dataset – in a sense, it would be learning its own word embeddings. Since we are particularly interested in how the semantic content of a sentence influences head motion, and pre-trained word embeddings are available, it is sensible to use these instead. This is the representation used in this project.

Other options are to represent the input text in terms of characters or phonemes. We did not use these approaches for two reasons: firstly, because they do not have the benefits of pre-trained word embeddings, i.e. capturing the semantic meaning of text. Secondly, it makes synchronizing the input and output sequences more difficult – as noted above, recurrent neural networks require that the input and output sequences have the same length.

3.4 Preprocessing

The first step of preprocessing was splitting the dataset into sequences. Whereas audio-based systems often use sliding windows as their input and output sequences [3, 13], we used the dataset's transcriptions as a guide to segment the data into distinct sequences. Each line of the transcriptions corresponds roughly to one utterance – see Table 3.1 for examples.

that's true
yeah that's true that's a good point good point
I suppose it does come down to luck a bit on the day and might must do I dunno
although I've heard that there was more uh
more danger to uh bungee jumping but then I mean
yeah jumping from a metal box in the sky
it's uh

Table 3.1: Typical sentence transcriptions

In encoder-decoder models, the input and output sequences can have different lengths – the inputs are simply sequences of word embeddings. However, this is not the case for standard recurrent neural networks. The text and head motion sequences must have the same length, yet there are many more frames of head motion than there are words in a sentence. We get around this problem by repeating the embedding for a given word for as many timesteps as it is spoken. This timing data is gotten through automatic segmentation of the audio waveform. When there is silence between words, this is represented by vectors of zeros. The rightmost column of Table 3.2 shows what an example input sequence might look like for the sentence ”hello how are you”.

In the transcribed conversational speech, there were many hesitation words such as ”um”, ”uh”, et cetera – about 3% of the tokens in total. Additionally, the transcribers had used several different spellings of these. Most of these hesitation words were

Word	Input sequence
Hello	embedding(*hello*)
	...
	embedding(*hello*)
(pause)	[0 0 0 ... 0]
	...
	[0 0 0 ... 0]
How	embedding(*how*)
	...
	embedding(*how*)
Are	embedding(*are*)
	...
	embedding(*are*)
You	embedding(*you*)
	...
	embedding(*you*)

Table 3.2: An example input sequence

not in the vocabulary of the pre-trained word embeddings used, so we encoded all hesitation words using the same embedding for the word ”um”. In addition, many contractions like ”they’re” or ”hasn’t” do not have pre-trained embeddings. When these words appear, we replace them with their component words. For example, if the word ”didn’t” is spoken for 100 timesteps, the input sequence will have 50 timesteps of the embedding for ”did” followed 50 timesteps of the embedding for ”not”.

The head motion in the dataset was recorded at 100Hz. Since smooth animations can be achieved with just 25Hz (i.e. 25 frames per second), we downsampled the data by a factor of 4 to reduce training time and make file sizes more manageable. Before doing so, we first made sure that the head motion did not contain frequencies above the Nyquist frequency in order to avoid aliasing. If the sampling frequency is f_s , then any frequencies f in the range $\frac{f_s}{2} < f < f_s$ will be folded back into the Nyquist band from 0 to $\frac{f_s}{2}$ and cause aliasing. We applied the Fast Fourier Transform to each feature in the dataset’s head motion to generate the power spectrum in Fig. 3.2. Because there are few to no frequencies above 12.5Hz, we did not have to use a low pass filter before downsampling.

The head motion data consists of 6 features – 3 Euler angles and 3 translations. We added delta features to these, i.e. their first derivatives, resulting in a total of 12 output features. Haag & Shimodaira [14] found delta features to improve performance when training models on the same dataset as in this project. Since we are operating on sequences of discrete timesteps we take the finite difference quotient d_t at each timestep t, an approximation to the derivative:

$$d_t = \frac{f_{t+1} - f_{t-1}}{2}$$

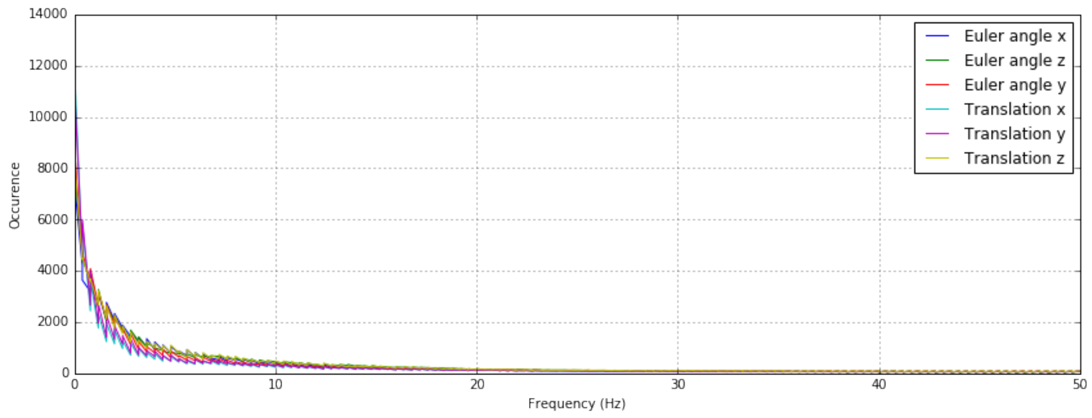


Figure 3.2: Power spectrum for head motion in the dataset

where f_t is the value of a feature at timestep t. Before adding delta features, we first normalize each feature to have a mean of 0 and a variance of 1 such that they all weigh equally in the error.

After this, the 2354 sequences in the extroverted data were randomly shuffled and split into training, validation and test sets. 80% of the sequences were used for training, 10% for validation, and 10% as a held-out test set that was only used to compare models at the end of our experiments.

3.5 Implementation

Initially, models were implemented using the Theano framework [26], but early on we moved to using Keras [27]. Keras is a high-level neural network Python framework that allows for quick experimentation and iteration. Keras runs on one of two backends: Theano or TensorFlow [28]. These are both frameworks that build computational graphs operating on tensors. These graphs are compiled to fast C code, and functions are easily differentiated by moving backwards through the graph and using the chain rule. Theano was used as the Keras backend as it is a more mature and optimized framework.

For the encoder-decoder models, we used TensorFlow. Keras does not implement an API for this architecture, and TensorFlow provides building blocks for neural networks that are not present in Theano.

All neural networks were trained by gradient descent, specifically using the Adam optimizer [29]. This method estimates the first and second moments of the network's gradients and uses these to compute individual learning rates for each parameter. To reduce overfitting, networks used early stopping, i.e. they trained until the validation error had seen no improvement for 10 epochs. The lowest validation error was usually reached after about 30-40 epochs depending on the particular architecture used. To initialize the networks' weights, we use Xavier uniform initialization [30], in which the initial weights are drawn from a uniform distribution in the range

$$\left[-\sqrt{\frac{6}{n_{in}+n_{out}}} \quad , \quad \sqrt{\frac{6}{n_{in}+n_{out}}} \right]$$

where n_{in} is the number of inputs to the layer and n_{out} is the number of output units. This helps keep the weight variance from layer to layer the same, and thus helps avoid vanishing and exploding gradients.

Experiments were run on either an Nvidia Tesla K80 GPU or an Nvidia Titan X GPU. The head motion was animated using an internal tool at the Centre for Speech Technology Research at the University of Edinburgh.

The survey for subjective evaluations was implemented as a web app using the Django web framework [31].

Chapter 4

Experiments

4.1 Loss function

An early observation in this project was that the mean squared error of Euler angles does not necessarily correspond to what we are trying to minimize, namely the angle between two orientations. This can be seen in Fig. 4.1, which shows a comparison of the root mean square (RMS) difference between randomly generated sets of Euler angles and the angle between them. As a specific example, see the Euler angles in Table 4.1.

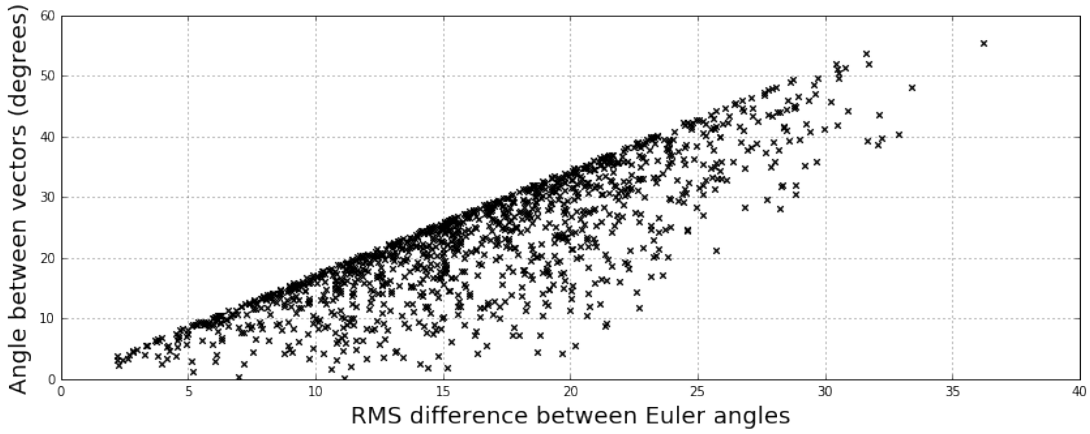


Figure 4.1: RMS difference between Euler angles vs. angle difference

Euler angle	MSE	Angle difference
$\begin{bmatrix}12 & 15 & 25\end{bmatrix}$ $\begin{bmatrix}-15 & 10 & 5\end{bmatrix}$	384.6	20.07°
$\begin{bmatrix}12 & 15 & 25\end{bmatrix}$ $\begin{bmatrix}7 & -15 & 10\end{bmatrix}$	383.3	31.80°

Table 4.1: An example of two sets of Euler angles that, while having a very similar MSE, do not represent similar differences in orientation.

As a potential solution, we investigated whether a different representation of head motion would be more appropriate, namely quaternions. However, comparing the RMS difference between quaternions to angle difference showed that these suffer from the same problem as Euler angles (see Fig. 4.2).

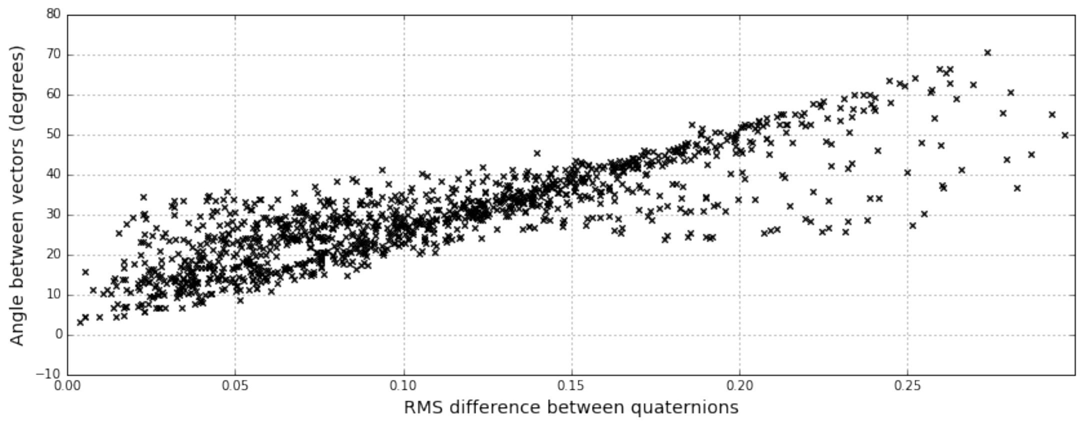


Figure 4.2: RMS difference between quaternions vs. angle difference

We implemented a custom loss function to more accurately represent the error we are trying to minimize. This function works by rotating the same initial unit vector by the two sets of Euler angles that are being compared. It then returns the cosine difference between these newly rotated vectors.

To rotate this initial unit vector, we first create the rotation matrices for our Euler angles with order x, z, y. Let c_x, c_z, c_y be shorthand for $\cos x, \cos z, \cos y$ and s_x, s_z, s_y be shorthand for $\sin x, \sin z, \sin y$. The rotation matrices around each axis R_x, R_z, R_y are given by

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{bmatrix}$$

$$R_z = \begin{bmatrix} c_z & -s_z & 0 \\ s_z & c_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} c_y & 0 & s_y \\ 0 & 1 & 0 \\ -s_y & 0 & c_y \end{bmatrix}$$

Multiplying these together, we get

$$R_{xzy} = \begin{bmatrix} c_z c_y & -s_z & c_z s_y \\ c_x s_z c_y + s_x s_y & c_x c_z & c_x s_z s_y - s_x c_y \\ s_x s_z c_y - c_x s_y & -s_x c_z & s_x s_z s_y + c_x c_y \end{bmatrix}$$

Multiplying a vector by this matrix gives the rotated vector.

This custom loss function is very inefficient compared to e.g. MSE. While MSE can be efficiently calculated on large tensors, this custom loss function has to loop through each timestep and construct rotation matrices at each of them (a computationally expensive operation, especially due to using sines and cosines). This also means that we cannot compute the gradients for many sequences at once, so we lose the benefits of training on minibatches of sequences. With minibatches, gradient updates are calculated based on the error of many input/output pairs at once. This reduces the noisiness in gradients and allows the optimizer to take larger step sizes. Because of these two reasons, this custom loss function is drastically slower to train than MSE. In addition, this loss function has the same global minimum as MSE: when two sets of Euler angles equal each other, the MSE is 0, as is the angle difference between the orientations. We did not see any clear improvements in the synthesized head motion when using this loss function, so we did not pursue it further. The following experiments are all trained using MSE loss.

4.2 Word embeddings

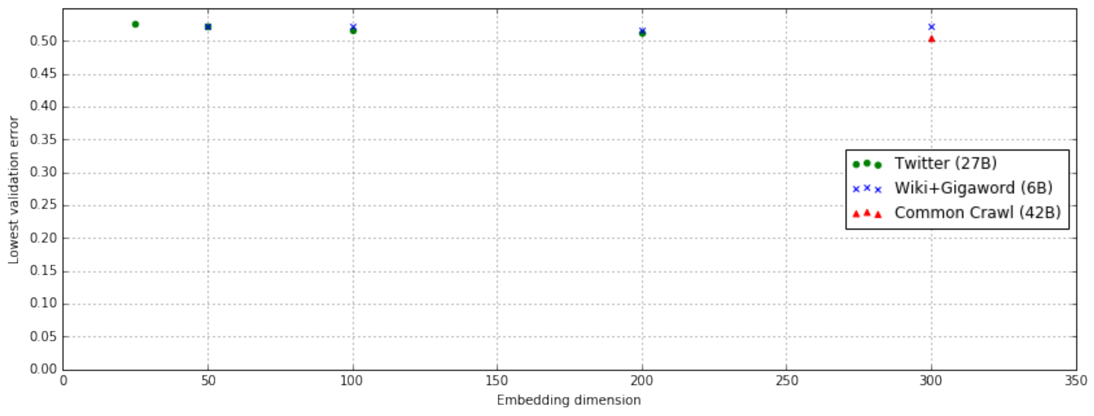
To decide on appropriate word embeddings, we trained several recurrent neural networks on GloVe embeddings from different datasets and with different dimensionalities. We hypothesized that the embeddings trained on Twitter data would perform better, since social network posts are more similar to conversational text than e.g. Wikipedia and thus the embeddings might better represent the meanings of words in a casual conversational context. We also compared embeddings trained on Common Crawl (a large collection of websites), and on Wikipedia and the Gigaword corpus of news articles.

The model trained was a baseline RNN with the architecture specified in Table 4.2. Between the hidden layers, dropout layers were added that drop connections with $p = 0.5$. This has been shown to reduce overfitting since the network cannot rely on very specific (but fragile) combinations of weights [32].

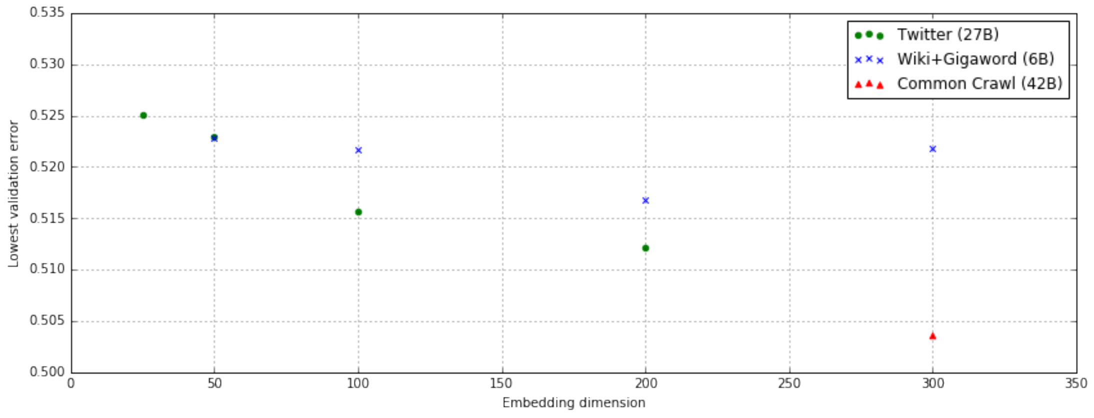
Layer type	No. of cells
Input	300
Feedforward	100
Feedforward	100
LSTM	100
LSTM	100
Output	12

Table 4.2: RNN architecture in embedding experiment

Our results are summarised in Fig. 4.3. We found that embeddings of higher dimensionalities tend to mean lower validation set error. The MSE improvement from the highest validation error to the lowest is 4.08%.



(a) Overview



(b) Zoomed-in view

Figure 4.3: Comparison of word embeddings

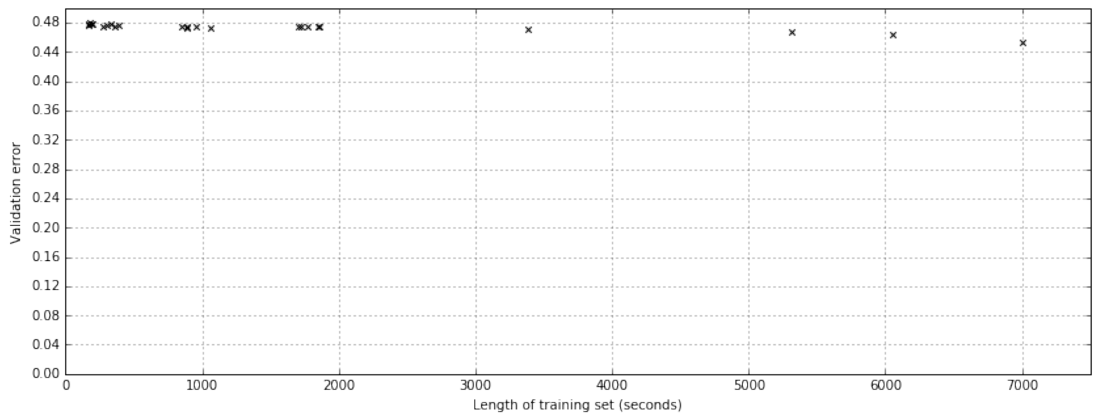
Embeddings trained on Wikipedia and Gigaword gave higher errors. This could be because of the fact that these embeddings were trained on 6 billion tokens (rather than 27 or 42 billion for Twitter and Common Crawl, respectively), or because the domains of Wikipedia and Gigaword (encyclopedia and news articles) is further from conversational text than social network posts or websites. For example, the seven most common words in our dataset are ”I”, ”you”, ”yeah”, ”the”, ”and”, ”like”, and ”um” – most of which are less likely to be found in more formal contexts. Together, these seven words make up more than 20% of the tokens in the dataset. Going forward, we used the 300-dimensional embeddings trained on the Common Crawl corpus.

4.3 Training set size

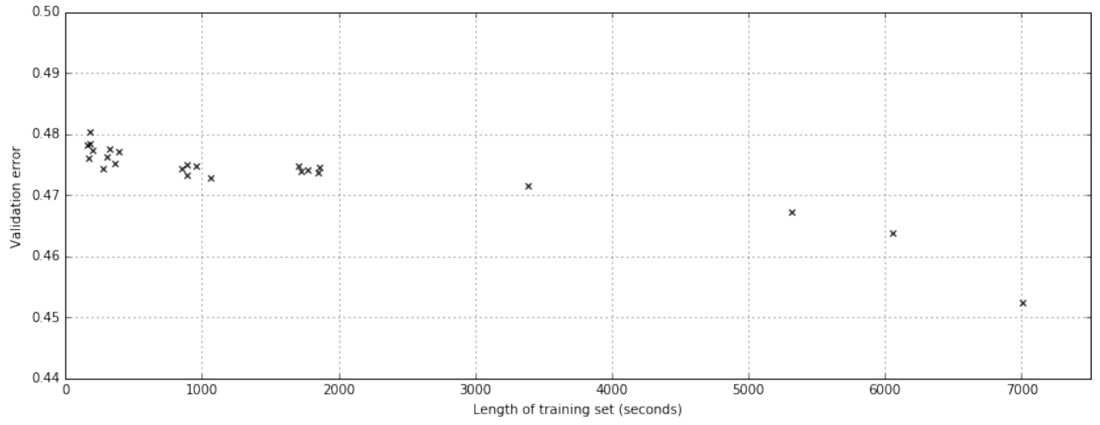
In the same vein as the above experiment, we trained several RNNs using different amounts of training data. Here, the hypothesis was that using more data would improve performance, replicating the results in Banko & Brill [24]. We sampled training sets of different sizes randomly from the entire dataset of extroverted speakers. For small amounts of training data (fewer than 500 sentences), we repeated the experiment

5 times with different samples. All models were validated on the same held-out validation set. The RNN architecure used was the same as in Table 4.2, but with double the number of cells in feedforward layers.

See Fig. 4.4 for the results. We found that larger training sets lowered the validation error by up to 5.82%. This was a contributing factor in the decision to train models on all extroverted speakers rather than speaker-dependent models. Our results hint that the amount of training data is one limiting factor for the model's performance, and that a larger dataset would be beneficial.



(a) Overview



(b) Zoomed-in view

Figure 4.4: Effects of training set size on validation error

4.4 Embedding interpolation

The blue line in Fig. 4.5 shows the current model's prediction for Euler angle x for a typical sequence. The vertical lines represent word boundaries. While the ground truth (the green dotted line) is quite smooth, the predictions by the current model tend to have sharp peaks at word boundaries – for example at timesteps 18 and 24 in this figure. These sudden movements lead to unnatural head motion. We hypothesized that this was because of the input representation. As described in section 3.4, the embedding for each word is repeated for as many timesteps as that word is spoken. This means that the input sequence contains relatively long periods in which there is no change in the input, and when a change does occur (at a new word), then there is a sudden change in the output prediction.

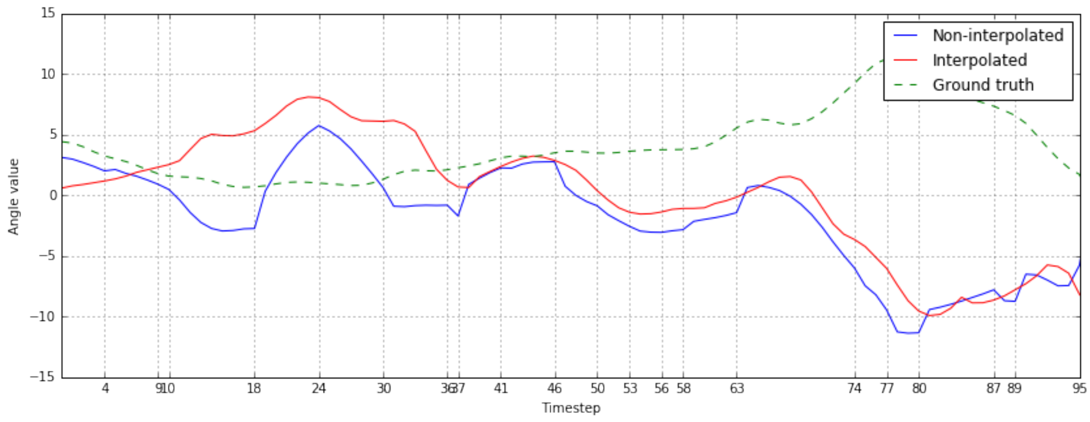


Figure 4.5: Euler angle *x* over a typical sequence

In order to alleviate this problem, we tried linearly interpolating each dimension of the input. First, we find the points at the center of each word, shown as the red diamonds in Fig. 4.6. This figure shows one of the 300 dimensions of the input sequence for the same sentence as Fig. 4.5. These points are linearly interpolated to give the modified input – the blue line.

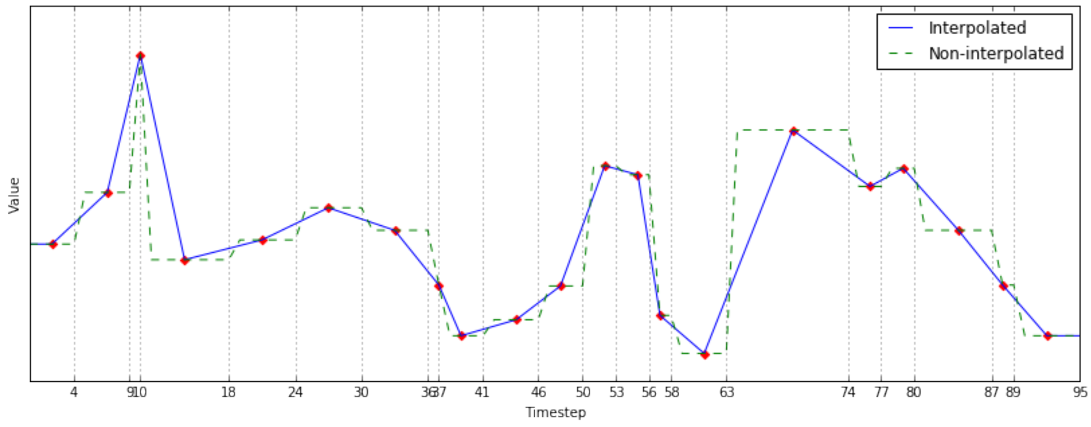


Figure 4.6: Dimension 1 of an interpolated and non-interpolated input sequence

This interpolated input leads to a qualitative difference in the prediction head motion, as can be seen in Fig. 4.5. The predicted motion from interpolated input (the red line) is smoother and does not exhibit the sharp changes that we get when using non-interpolated data. This also leads to a quantitative change in error on the validation set. Table 4.3 shows that the model performs worse when using interpolated inputs (the validation error increases by 6.06%). However, the head motion looks a lot more lifelike and this outweighs the change in validation error.

Input data type	Validation set error
Non-interpolated	0.528
Interpolated	0.560

Table 4.3: Result of interpolating input data

Interpolation of word embeddings raises one question. It is not clear to what extent the input sequence captures the meaning of the input words when we modify the word embeddings in this way. We will evaluate this in subjective evaluations (chapter 5) by comparing our RNN model with a ”dummy” model that, instead of using the word embeddings for the input sentence, uses embeddings from random words. If these models perform equally well, it would imply that the RNN model learns lifelike head motion from the timing of the words rather than from their meaning.

4.5 Relative data

One thing to note about Fig. 4.5 is that the predicted motion has been scaled by a factor of 10. Without this scaling, the output looks more like Fig. 4.7. The RNN returns predictions where each feature has a very low variance compared to that of the training data. We believe that this may have to do with how head motion is represented in our dataset. Each frame of head motion is represented as a rotation from the same absolute starting point. However, the most important aspect of head motion is how it moves over time as opposed to how it moves in relation to this initial orientation. For instance, a head shake communicates the same meaning whether the speaker is facing slightly left or slightly right. While training, the model encounters meaningful head gestures (e.g. nods or shakes) that occur at different orientations. One potential explanation for the low-variance output is that the model cannot learn simple rules for head motion – in a sense, training data at different orientations ”cancel out” and the best way for the RNN to minimize error is by returning relatively stationary head motion.

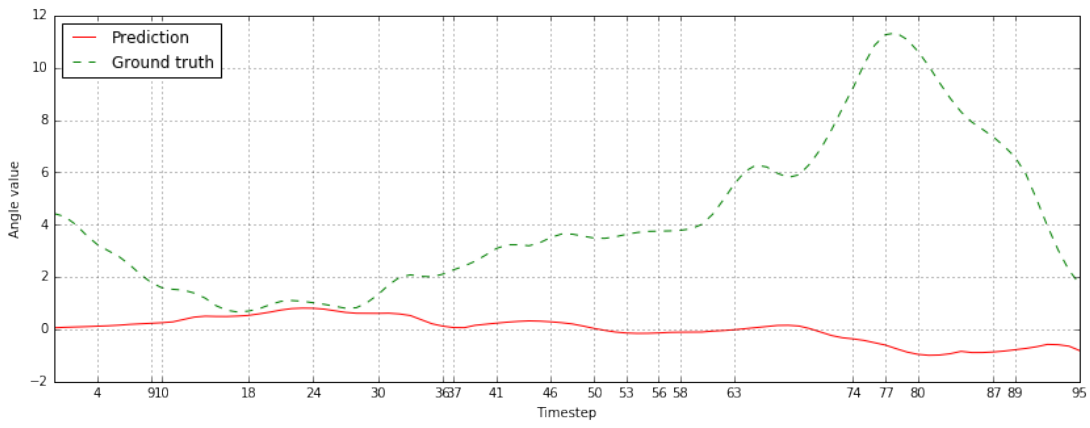


Figure 4.7: Unscaled prediction for Euler angle *x*

In spite of its low variance, the resulting head motion looks quite lifelike when scaled up. However, we also tried to train models on relative, rather than absolute, head motion. In this relative representation, each timestep is represented as the difference from the last timestep. If each frame of head motion for timestep t is given as $\mathbf{u}_t = \begin{bmatrix}\psi_t & \theta_t & \phi_t & v_{x,t} & v_{y,t} & v_{z,t}\end{bmatrix}$ (where ψ_t, θ_t, ϕ_t are Euler angles and $v_{x,t}, v_{y,t}, v_{z,t}$ are translations), then the corresponding relative frame $\mathbf{r}_t$ is simply $\mathbf{r}_t = \mathbf{u}_t - \mathbf{u}_{t-1}$.

We trained two RNNs using the architecture in Table 4.4. When reconstructing the relative model's output (i.e. making it absolute again) and evaluating both models on the validation set, there is almost no difference in the two models' MSEs. However, there is a more important qualitive difference. The output of the relative model, when reconstructed to absolute data, tends to drift and this leads to very large and unnatural sweeps in head motion, particularly in long sequences. Fig. 4.8 shows the predictions for Euler angle x for a typical sentence. The relative model's prediction gradually drifts off, leading to unnatural head motion.

Because of this drifting problem, we did not use relative data – we used the absolute data as it is given in the dataset. However, we do scale model's predictions such that their variance matches the variance of the training set – see section 4.8.

Layer type	No. of cells
Input	300
Feedforward	200
Feedforward	100
LSTM	100
LSTM	100
Output	12

Table 4.4: RNN architecture in absolute/relative data experiment

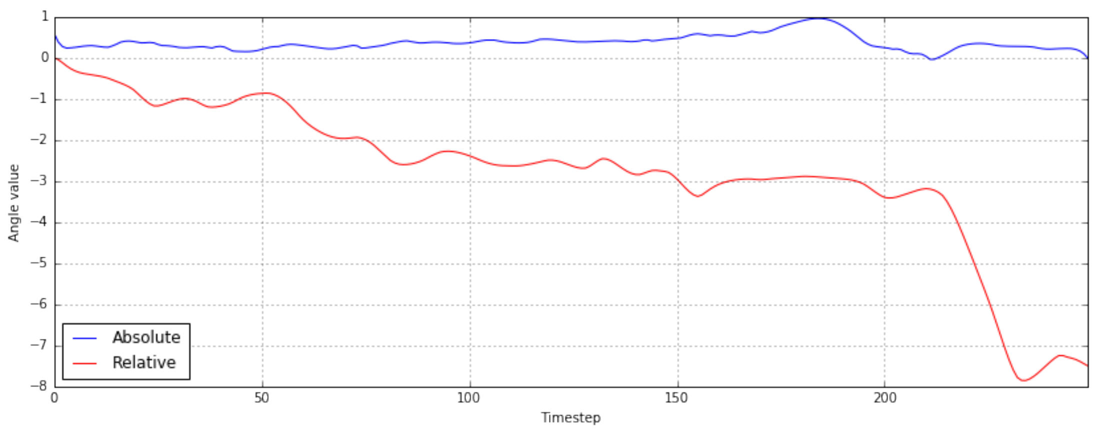


Figure 4.8: Feature drift when using relative data

4.6 Encoder-decoder model

We also tried using an entirely different architecture – an encoder-decoder model as described in section 2.2.1. Most applications of head motion synthesis will have the timing information of words available, i.e. how long each word is spoken for. This information can come from either a text-to-speech system or from automatic word segmentation (as in the dataset we used in this project). However, this encoder-decoder model is an attempt to build a purely text-based system for head motion synthesis.

We trained a model for 100 epochs. Both the encoder and decoder networks had one hidden layer with 300 LSTM cells. We implemented the model in TensorFlow.

One initial observation is that this model's training error was orders of magnitude smaller than that of our RNN models. This is because we use teacher forcing to train the encoder-decoder model. With teacher forcing, the decoder network is fed the true values at each timestep, whereas during prediction it is fed its own previous output. Fig. 4.9 shows the decoder network with and without teacher forcing. When using teacher forcing, the decoder receives the true head motion $\mathbf{y}_t$ at each timestep t as opposed to the model's previous prediction $\mathbf{O}_{t-1}^{(dec)}$.

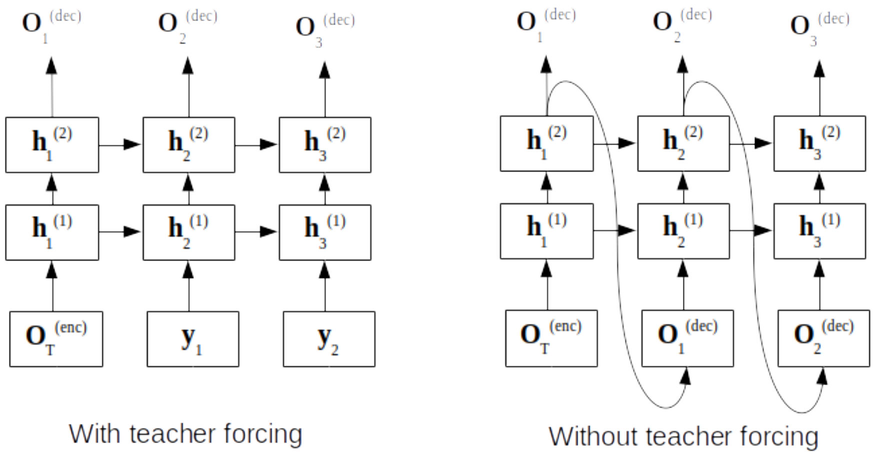


Figure 4.9: The decoder network with and without teacher forcing.

After a few epochs, the encoder-decoder model starts to vary greatly in its validation set error in subsequent epochs – see Fig. 4.11a (note the log scale of the y-axis). It may go from a low validation error in one epoch to a very high one (orders of magnitude larger) in the next, and back again. This is due to the qualities of the model's predictions. Fig. 4.10 shows one typical example. The model's output (the blue curve) exhibits a slow, gradual change with a large variance. During training, the scale of the predictions changes – Fig. 4.11b shows how the validation error for each epoch tends to rise and fall with the variance of the predictions. Because of this large variance, each feature of the output is scaled down to have the same variance as the training set, resulting in sequences that look like the red line in Fig. 4.10.

We also trained the same model using the relative data representation detailed in section 4.5, but the problem of drift was exacerbated by the already large range of move-

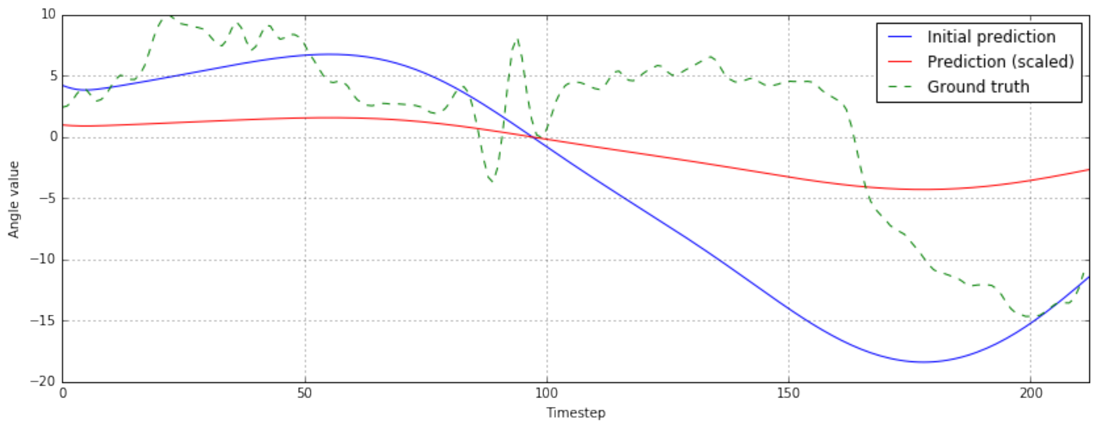


Figure 4.10: A typical encoder-decoder model prediction for Euler angle *x*

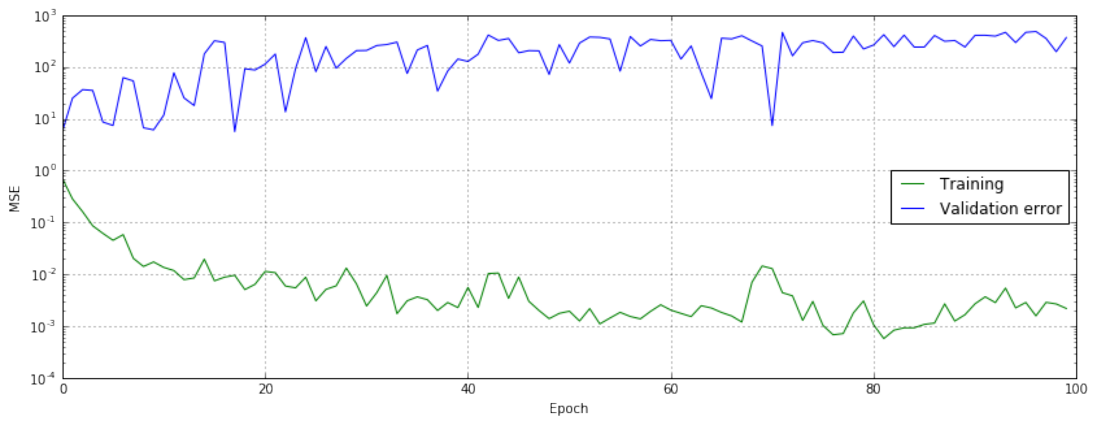
ment in the encoder-decoder model's predictions. This contributed to our decision to use absolute data for our final models. In general, the predicted head motion from this model is very unnatural. The head moves quite slowly and moves to unnatural orientations. We believe that this is because the decoder network does not have direct access to the input sentence's timing information. Head motion has many short movements that often match the cadence of a spoken sentence. Without this timing information, the decoder network does not learn to predict the type of rapid movements that we want – it is underfitting to the training set. A good next step for future work may be to use attention (see section 6.1).

4.7 Objective results

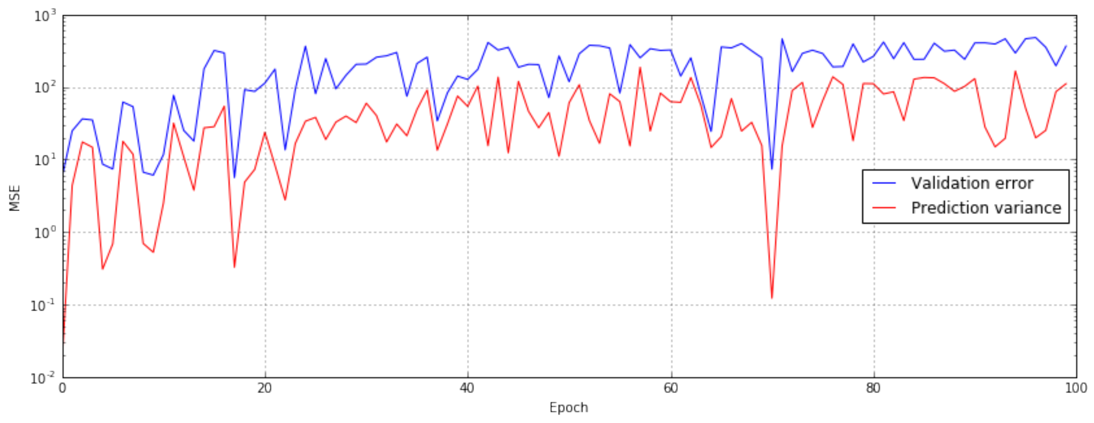
As a final step before subjective evaluations, we trained different model architectures to find the optimum number and sizes of layers. All models were trained on absolute (as opposed to relative) data, using 300-dimensional pre-trained word embeddings that were linearly interpolated along each of the 300 input dimensions. Table 4.5 shows the architectures that we trained. As an example, the architecture "200F-100F-100R" is a feedforward layer of 200 cells ("200F"), followed by a feedforward layer of 100 cells ("100F"), followed by a bidirectional recurrent layer of 100 LSTM cells ("100R"). Between each layer, we added dropout layers that drop connections with probability $p = 0.5$. Each architecture has a final feedforward layer of 12 cells to give output of the correct dimensionality. Models were trained until they saw no improvement in validation loss for 10 epochs.

The results are summarized in Table 4.5 and plotted in Fig. 4.12. The test errors are lower than the validation errors in previous experiments. This is because although the dataset was shuffled before splitting it into training, validation, and test sets, the test set presumably ended up having data that is relatively easy to learn. We can still use the relative performance of the models to compare them.

As Fig. 4.12 shows, there is not a clear pattern in which more layers lowers the test



(a) Training and validation error



(b) Validation error compared to average prediction variance

Figure 4.11: Encoder-decoder model training

error. The architecture with the highest test error performs 9.37% worse than the best model, which, while not insignificant, is also not as large a difference as expected. One potential explanation for the lack of a clear pattern is that different random weight initializations can affect the network's performance – it may get caught in different local minima. To test this, we trained the best network ("300F-200R-200R") 20 times with different random initializations. The results, shown by the red crosses in in Fig. 4.12c, demonstrate that the performance of a single architecture can vary a great deal depending on how its weights are initialized. In other words, it is likely that any difference in MSE that is shown in these results is an effect of the network's initialization rather than its architecture.

Another potential reason for the lack of a clear pattern is that the mean squared error is very sensitive to both time lags and scale differences. Another metric is the average correlation coefficient or ACC. First, we take the Pearson correlation coefficient r for each feature in a prediction and in the ground truth head motion. The Pearson correlation between two vectors $\mathbf{x}$ and $\mathbf{y}$ with respective means $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ (where each vector has length N) is given by

No. of layers	Architecture	Millions of parameters	Test set error
1	100R	0.32	0.4162
	200R	0.81	0.4164
	300R	1.45	0.4178
2	200F-100R	0.30	0.4077
	300F-200R	0.90	0.4018
	200R-100R	1.20	0.4224
	200R-200R	1.77	0.4152
3	300F-200F-100R	0.39	0.3965
	200F-200F-200R	0.75	0.4016
	300F-200R-100R	1.30	0.3984
	300F-200R-200R	**1.86**	**0.3862**
4	300F-200F-100F-100R	0.33	0.4086
	200F-100F-100R-100R	0.48	0.4156
	200F-200F-200R-100R	1.15	0.4123
	300F-300F-200R-200R	1.95	0.4165
5	300F-200F-100F-100F-100R	0.34	0.4198
	300F-200F-100F-100R-100R	0.57	0.4116
	300F-200F-200F-200R-100R	1.24	0.4180
	300F-200F-200R-100R-100R	1.44	0.4181
	300F-300F-300R-200R-100R	3.31	0.4198

Table 4.5: RNN architectures evaluated

$$r(\mathbf{x},\mathbf{y}) = \frac{s_{\mathbf{xy}}}{s_{\mathbf{x}}s_{\mathbf{y}}} = \frac{\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{y}_i - \bar{\mathbf{y}})}{\sqrt{[\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})^2][\sum_{i=1}^{N}(\mathbf{y}_i - \bar{\mathbf{y}})^2]}}$$

where $s_{\mathbf{xy}}$ is the covariance of the two vectors and $s_{\mathbf{x}}$ and $s_{\mathbf{y}}$ are the standard deviations of $\mathbf{x}$ and $\mathbf{y}$, respectively. To get the ACC, we take the mean of the Pearson correlations over all F features and all S sequences in the test set. With $\mathbf{f}_{i,j}^{(pred)}$ as the jth predicted feature in the ith sequence (a vector), and $\mathbf{f}_{i,j}^{(true)}$ as the corresponding ground truth feature:

$$ACC = \frac{1}{S \times F}\sum_{s=1}^{S}\sum_{f=1}^{F} r(\mathbf{f}_{s,f}^{(pred)}, \mathbf{f}_{s,f}^{(true)})$$

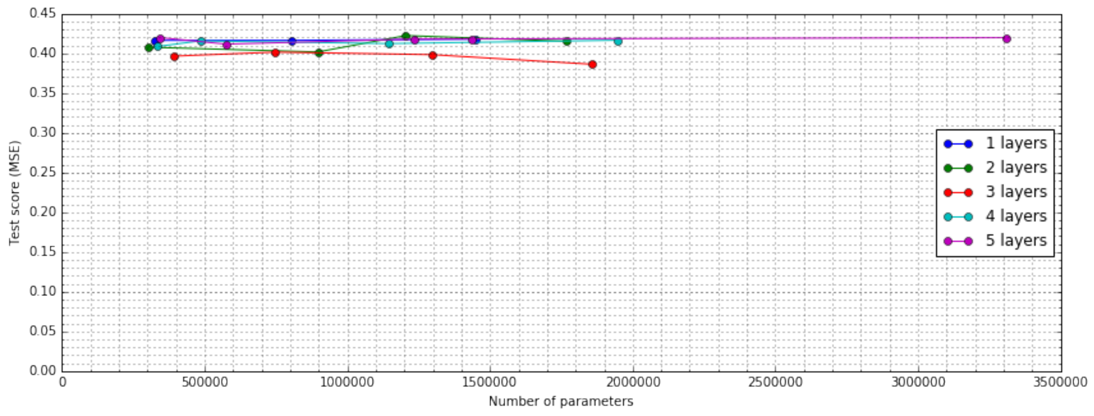
ACC lies in the range $[-1,1]$ where -1 is a perfect inverse correlation, 1 is a perfect correlation, and 0 is no correlation.

Fig. 4.13 shows the ACC for each model's predictions with the ground truth. Again, the red crosses show the ACCs achieved by training the same architecture many times with different initializations. The ACC metric shows a similar picture to the MSE: it appears that a large part of the differences in model performance can be accounted for by the random weight initialization. This indicates that the model architecture itself (in terms of number of layers and parameters) does not make a large difference in performance, and that even small models may be able to learn to model the dataset quite well.

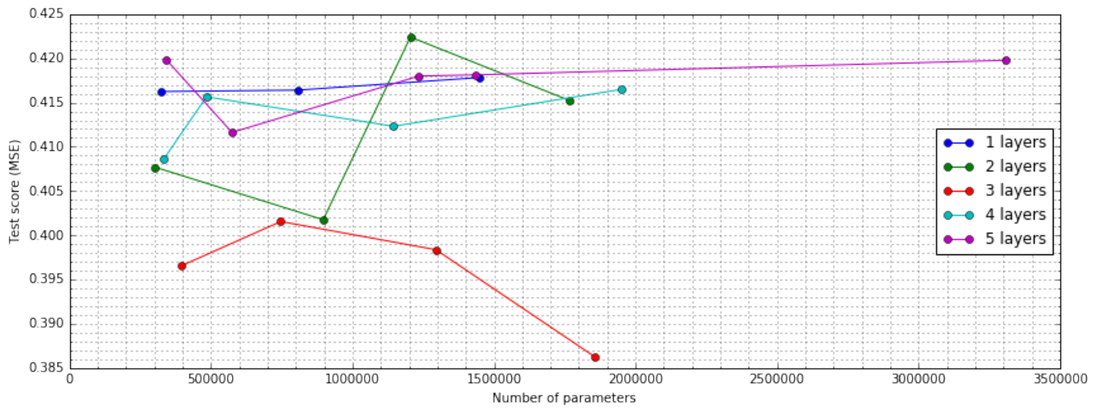
The fact that random initializations affect the models' performance to such a large extent means that we are less confident in the results from our earlier experiments. Even though they showed sensible patterns (such as more training data improving the model), the difference in performance when varying these factors was never more than 7% and here we found random initializations to affect peformance to an even greater extent. Though our model did well in subjective evaluations, repeating these earlier experiments several times and averaging out the results may show that, for example, different word embeddings are in fact better or that data from a single speaker is enough to train good models. Section 6.1 presents more options for further work.

4.8 Post-processing

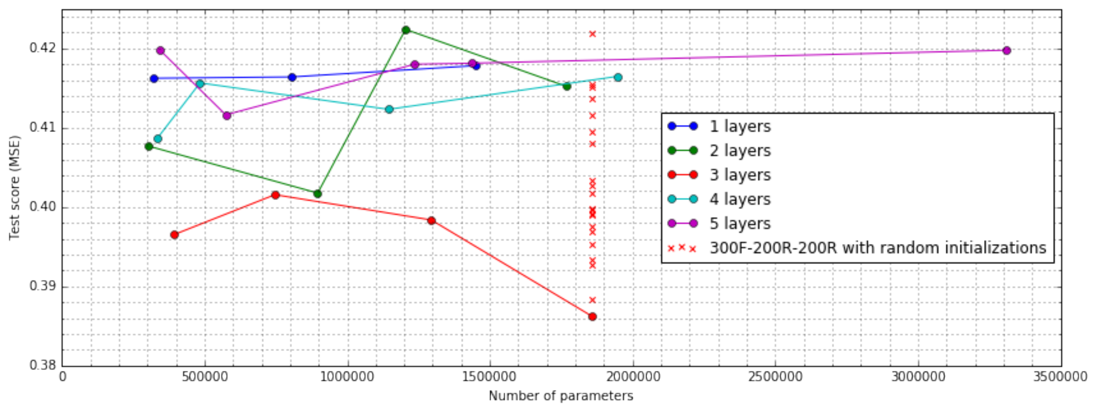
We found that the head motion generated by these models was almost always quite stationary, i.e. each feature exhibited a low variance. This looked very unnatural when animated. In order to get around this, we scale all predictions such that they have the same variance as the training set. This gives far more lifelike head motion.



(a) Architecture comparison



(b) Zoomed-in view



(c) The red crosses show a single network's performance when re-trained with different random initializations.

Figure 4.12: RNN architectures' errors on test set

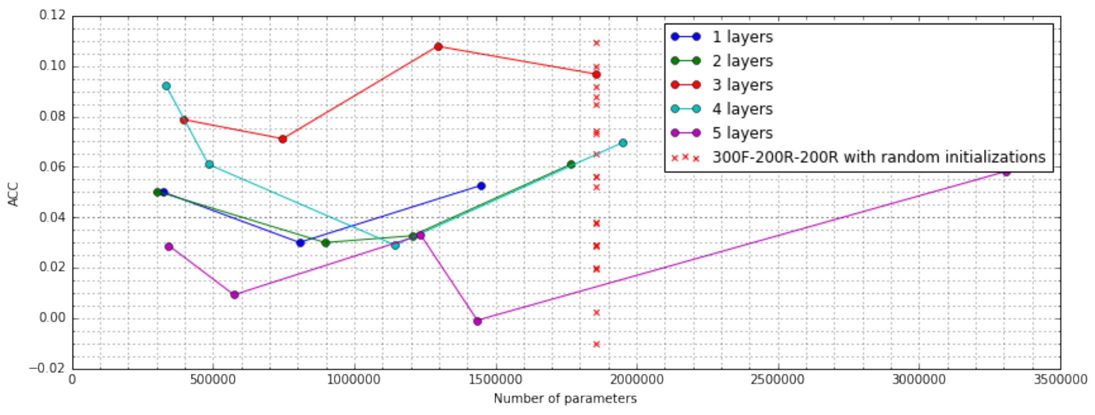


Figure 4.13: RNN architectures' ACCs

Chapter 5

Subjective Evaluations

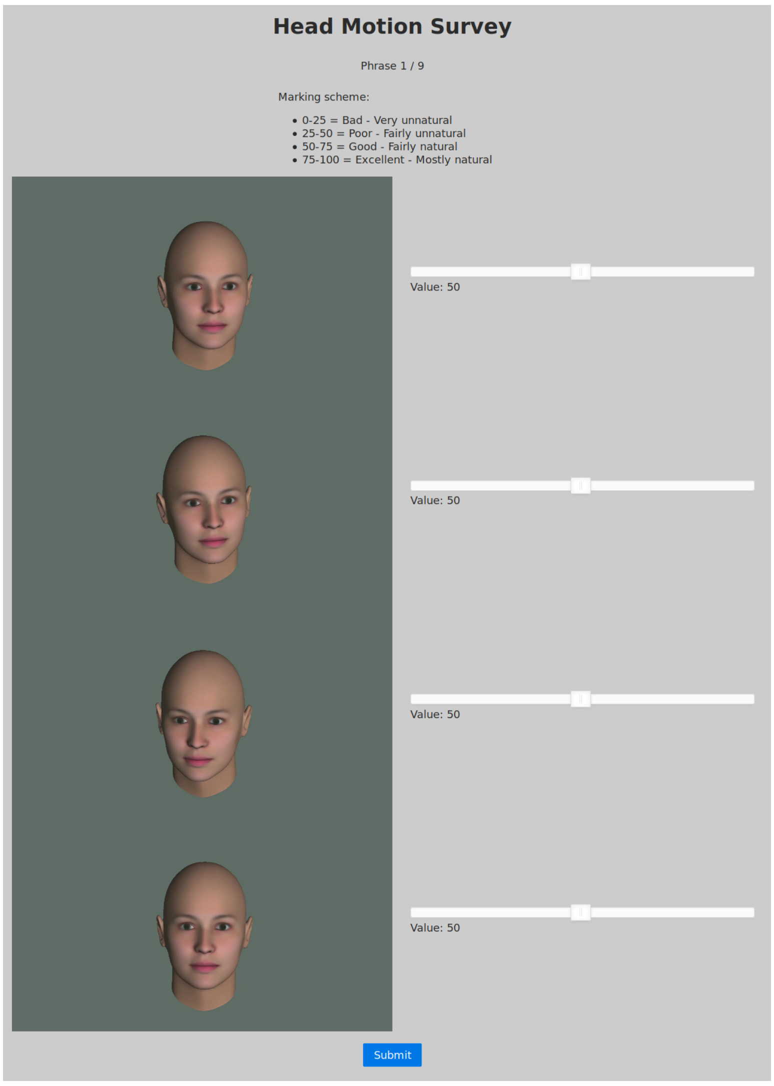
5.1 Experimental setup

In order to compare the best RNN model and the encoder-decoder model, we set up a subjective evaluation. We used the RNN model with the lowest MSE ("300F-200R-200R") as the best RNN model. 20 participants, all fluent English speakers, watched video clips of head motion and evaluated them on a 100-point scale. In order to focus only on head motion, the animated heads did not include eye movement or lip-sync. Due to issues with our animation tool, participants were shown heads animated with only rotations – not translations. This mirrors the approach in Busso *et al.* [12]. Participants were shown videos for 9 phrases from our test set. Each phrase was between 5 and 9 seconds long. For each phrase, 4 videos were shown in randomized order. One was the ground truth head motion, one was the RNN's prediction, one was the encoder-decoder model's prediction, and one came from the RNN model but with random words given as input as opposed to the actual phrase. Participants were not told the source of the videos. Additionally, the 9 phrases were shown in random order. Fig. 5.1 shows a screenshot of what the participants saw. Participants were told to score the videos according to the descriptions in Table 5.1.

The RNN model with random words as input was included in the subjective evaluation to find out whether our model gained anything from the semantic meaning of the input words, or whether it relied exclusively on the timing information. To construct the random inputs, each word of the true sentence was replaced by a random word in the word embeddings' vocabulary. Importantly, the new random words kept the same timing information as the original sentence, i.e. each random replacement was added for as many timesteps as the original word.

5.2 Results

2 participants' responses were removed due to giving very low scores to the ground truth head motion, leaving us with scores from 18 participants. The results are summa-


Head Motion Survey

Phrase 1 / 9

Marking scheme:

- 0-25 = Bad - Very unnatural
- 25-50 = Poor - Fairly unnatural
- 50-75 = Good - Fairly natural
- 75-100 = Excellent - Mostly natural

Value: 50

Value: 50

Value: 50

Value: 50

Figure 5.1: Screenshot of the head motion survey

rized in Table 5.2 and plotted in Fig. 5.2. As expected, the ground truth got the highest mean score, and the encoder-decoder model got the lowest (due to its unnaturally slow, sweeping motion).

A *t*-test confirms that the difference in mean scores between our best model, the RNN, and the actual head motion is statistically significant with $p < 0.00001$. In other words, our model does not achieve lifelike head motion that is indistinguishable from the ground truth. However, the mean score of the RNN model is 60.38 which is in the ”good - fairly natural” range as described by Table 5.1 which participants used to rate videos. Note that the mean score for true head motion (69.15) is also in this range (presumably because the lack of eye movement or lip-sync in the animations caused reviewers to give lower scores across the board). We believe that our model generates head motion that is lifelike enough to be used in production systems.

We are also interested in the difference between the RNN model and the RNN with random input. Another *t*-test shows that the higher mean of the RNN with true input is

Score range	Description
0 - 25	Bad - very unnatural
25 - 50	Poor - fairly unnatural
50 - 75	Good - fairly natural
75 - 100	Excellent - mostly natural

Table 5.1: Score descriptions

	Ground truth	RNN	Random RNN	Encoder-decoder
Mean	69.15	60.38	54.72	24.06
Variance	33.47	33.08	34.47	35.91

Table 5.2: Subjective results

indeed statistically significant with $p < 0.01$. This means that we are confident that the model is learning to use the meaning of words to synthesize head motion, rather than only using the timing information (i.e. how long each word is spoken for).

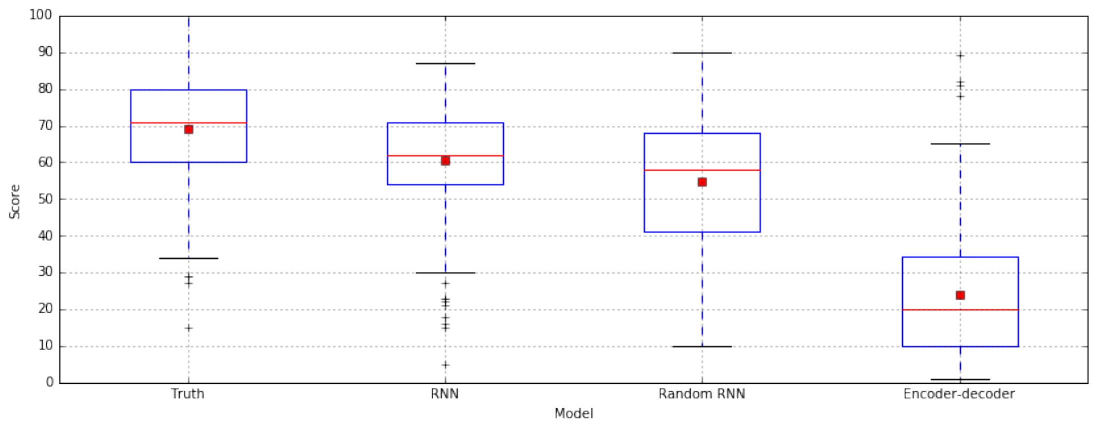


Figure 5.2: Subjective results. The red line represents the median, and the red square represents the mean. Crosses represent outliers, i.e. points that are more than 1.5 standard deviations from the first and third quartiles.

Chapter 6

Conclusion

We propose a model that achieves good results in subjective evaluations. This shows that representing the input text in a way that incorporates timing information, as we did, is a good starting point for text-based, neural network-based head motion synthesis. Our experiments show that currently, our recurrent neural networks perform better than encoder-decoder networks, though there are modifications to encoder-decoder models that could conceivably improve them (see section 6.1).

We also found that the performance of RNN models is highly dependent on the network's random weight initialization. We hypothesize that a smaller network than the one we used (with three hidden layers) could perform comparably with careful weight initialization or with pre-training.

Finally, our subjective evaluations showed that the RNN performed better when it had access to the words in the input sentence as opposed to only their timing information (i.e. the RNN with true input did better than the one with random input). This is promising for text-based head motion synthesis systems. It suggests that the semantic content of a sentence improves the predicted head motion, and this semantic content is readily available in text-based systems (in contrast to audio-based systems). It is worth noting that Busso *et al.* [12] present an HMM-based, speech-based system whose motion reviewers rated as being more lifelike than ground truth. Based on our finding that access to the specific words in a sentence improves model performance, we believe that combining a text-based approach with an audio-based one that has access to the prosodic structure of speech is likely to result in very good performance.

6.1 Further work

This project raises several promising areas of further research. First, it has been shown that there is a correlation between speech and head motion [6], and speech-based head motion synthesis systems have shown promising results [12]. A model that combines auditory and text-based input may be able to use both the prosodic structure of a sentence and its semantic meaning to synthesize lifelike head motion. One way to use prosodic features without needing audio could be to automatically extract these features from text as in Graf *et al.* [9]. Other options include synthesizing speech using an off-the-shelf text-to-speech system and using auditory features from this.

Though the encoder-decoder model used in this project performed badly in both objective and subjective evaluations, there may be ways to improve it. As described in section 4.6, its unnatural head motion may be caused by a lack of timing information. Perhaps adding this information could improve performance, either by passing word durations as an explicit input to the model, or by using attention to give the decoder access to the entire input sequence while decoding. In broad strokes, attention works by passing the encoder network's states from all input timesteps to the decoder, rather than only its final state. The decoder then weighs how much to use each of these states for its output.

In spite of the finding that head motion is speaker-dependent [5], we trained our models on data from different speakers in order to gain the benefits of more data. More work is needed to investigate whether training on individual speakers improves performance. It is difficult to evaluate this objectively, since the test sets will be different, but subjective evaluations could shed light on whether synthetic head motion based on different speakers can be individuated.

We found the random initialization of an RNN to play a large role in its performance, improving its test MSE by as much as 10%. In the future, we would like to repeat some of the experiments in this project in a way that takes this into account. For example, the experiments could be repeated many times and the results averaged out. Another option is to use pre-training to lower the chance that a network will get stuck in bad local minima. Ding *et al.* [13] found significant improvements in deep neural network-based head motion synthesis by pre-training a deep belief network and then adding an output layer to form a deep neural network. A similar procedure may improve RNN models in which the first layers are feedforward layers. We believe that pre-training could potentially give a large performance boost.

Bibliography

[1] E. Cosatto, J. Ostermann, H. P. Graf, and J. Schroeter, "Lifelike talking faces for interactive services," *Proceedings of the IEEE*, vol. 91, no. 9, pp. 1406–1429, 2003.

[2] D. Heylen, "Challenges ahead: head movements and other social acts during conversations," in *Proceedings of the Joint Symposium on Virtual Social Agents: AISB'05 Social Intelligence and Interaction in Animals, Robots and Agents*, The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, 2005.

[3] C. Busso, Z. Deng, M. Grimm, U. Neumann, and S. Narayanan, "Rigid head motion in expressive facial animation: Analysis and synthesis," *IEEE Transaction on Audio, Speech and Language Processing*, vol. 15, no. 3, pp. 1075–1086, 2006.

[4] K. G. Munhall, J. A. Jones, D. E. Callan, T. Kuratate, and E. Vatikiotis-Bateson, "Visual prosody and speech intelligibility head movement improves auditory speech perception," *Psychological science*, vol. 15, no. 2, pp. 133–137, 2004.

[5] H. Hill and A. Johnston, "Categorizing sex and identity from the biological motion of faces," *Current biology*, vol. 11, no. 11, pp. 880–885, 2001.

[6] T. Kuratate, K. G. Munhall, P. Rubin, E. Vatikiotis-Bateson, and H. Yehia, "Audio-visual synthesis of talking faces from speech production correlates.," in *EuroSpeech*, 1999.

[7] H. C. Yehia, T. Kuratate, and E. Vatikiotis-Bateson, "Linking facial animation, head motion and speech acoustics," *Journal of Phonetics*, vol. 30, no. 3, pp. 555–568, 2002.

[8] E. Z. McClave, "Linguistic functions of head movements in the context of speech," *Journal of pragmatics*, vol. 32, no. 7, pp. 855–878, 2000.

[9] H. P. Graf, E. Cosatto, V. Strom, and F. J. Huang, "Visual prosody: facial movements accompanying speech," in *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pp. 396–401, IEEE, 2002.

[10] S. Zhang, Z. Wu, H. M. Meng, and L. Cai, "Head movement synthesis based on semantic and prosodic features for a Chinese expressive avatar," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 4, pp. IV–837, IEEE, 2007.

[11] G. Zoric, K. Smid, and I. S. Pandzic, “Automated gesturing for virtual characters: speech-driven and text-driven approaches,” in *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pp. 295–300, IEEE, 2005.

[12] C. Busso, Z. Deng, U. Neumann, and S. Narayanan, “Learning expressive human-like head motion sequences from speech,” in *Data-driven 3D facial animation* (Z. Deng and U. Neumann, eds.), ch. 6, pp. 113–131, Springer, 2008.

[13] C. Ding, L. Xie, and P. Zhu, “Head motion synthesis from speech using deep neural networks,” *Multimedia Tools and Applications*, vol. 74, no. 22, pp. 9871–9888, 2015.

[14] K. Haag and H. Shimodaira, “Bidirectional LSTM Networks Employing Stacked Bottleneck Features for Expressive Speech-Driven Head Motion Synthesis,” in *International Conference on Intelligent Virtual Agents*, pp. 198–207, Springer, 2016.

[15] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[16] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[17] 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.

[18] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[19] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[20] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[21] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation,” in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[22] K. Haag and H. Shimodaira, “The University of Edinburgh Speaker Personality and MoCap Dataset,” in *Proceedings of the Facial Analysis and Animation*, FAA ’15, (New York, NY, USA), pp. 8:1–8:2, ACM, 2015.

[23] R. Dall, J. Yamagishi, and S. King, “Rating naturalness in speech synthesis: The effect of style and expectation,” *Proceedings Speech Prosody, Dublin, Ireland*, 2014.

[24] M. Banko and E. Brill, “Scaling to very very large corpora for natural language disambiguation,” in *Proceedings of the 39th annual meeting on association for computational linguistics*, pp. 26–33, Association for Computational Linguistics, 2001.

[25] Y. LeCun, C. Cortes, and C. J. Burges, “The MNIST database of handwritten digits,” 1998.

[26] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[27] F. Chollet, “Keras.” `https://github.com/fchollet/keras`, 2015.

[28] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: A system for large-scale machine learning,” in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[29] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[30] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks.,” in *Aistats*, vol. 9, pp. 249–256, 2010.

[31] Django Software Foundation, “Django.” `https://djangoproject.com`, 2017.

[32] 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.

[33] Y. Channua, “Text-driven head motion synthesis using neural networks.” Unpublished MSc thesis, University of Edinburgh, 2016.

[34] V. Bălănescu, “Expressive head motion synthesis using convolutional autoencoders.” Unpublished MSc thesis, University of Edinburgh, 2016.