A Data-Driven Approach For Automatic Visual Speech In Swedish Speech Synthesis Applications

JOEL HAGROT
A Data-Driven Approach For Automatic Visual Speech In Swedish Speech Synthesis Applications

JOEL HAGROT
Abstract

This project investigates the use of artificial neural networks for visual speech synthesis. The objective was to produce a framework for animated chat bots in Swedish. A survey of the literature on the topic revealed that the state-of-the-art approach was using ANNs with either audio or phoneme sequences as input.

Three subjective surveys were conducted, both in the context of the final product, and in a more neutral context with less post-processing. They compared the ground truth, captured using the deep-sensing camera of the Iphone X, against both the ANN model and a baseline model. The statistical analysis used mixed effects models to find any statistically significant differences. Also, the temporal dynamics and the error were analyzed.

The results show that a relatively simple ANN was capable of learning a mapping from phoneme sequences to blend shape weight sequences with satisfactory results, except for the fact that certain consonant requirements were unfulfilled. The issues with certain consonants were also observed in the ground truth, to some extent. Post-processing with consonant-specific overlays made the ANN’s animations indistinguishable from the ground truth and the subjects perceived them as more realistic than the baseline model’s animations.

The ANN model proved useful in learning the temporal dynamics and coarticulation effects for vowels, but may have needed more data to properly satisfy the requirements of certain consonants. For the purposes of the intended product, these requirements can be satisfied using consonant-specific overlays.
Sammanfattning

Detta projekt utreder hur artificiella neuronnät kan användas för visuell talsyntes. Ändamålet var att ta fram ett ramverk för animerade chatbotar på svenska. En översikt över litteraturen kom fram till att state-of-the-art-metoden var att använda artificiella neuronnät med antingen ljud eller fonemsekvenser som indata.

Tre enkäter genomfördes, både i den slutgiltiga produktens kontext, samt i en mer neutral kontext med mindrebearbetning. De jämförde sanningsdatat, inspelat med Iphone X:s djupsensorkamera, med både neuronnätsmodellen och en grundläggande så kallad baselinemodell. Den statistiska analysen använde mixed effects-modeller för att hitta statistiskt signifikanta skillnader i resultaten. Även den temporala dynamiken analyserades.

Resultaten visar att ett relativt enkelt neuronnät kunde lära sig att generera blendshapesekvenser utifrån fonemsekvenser med tillfredsställande resultat, förutom att krav såsom läppslutning för vissa konsonanter inte alltid uppfylldes. Problemen med konsonanter kunde också i viss mån ses i sanningsdatat. Detta kunde löses med hjälp av konsonantspecifik bearbetning, vilket gjorde att neuronnätets animationer var oskiljbara från sanningsdatat och att de samtidigt upplevdes vara bättre än baselinemodellens animationer.

Sammanfattningsvis så lärde sig neuronnätet vokaler väl, men hade antagligen behövt mer data för att på ett tillfredsställande sätt uppfylla kraven för vissa konsonanter. För den slutgiltiga produktens skull kan dessa krav ändå uppnås med hjälp av konsonantspecifik bearbetning.
3 Related work
3.1 Data-driven rule-based models
3.2 Machine learning models
  3.2.1 Hidden Markov models
  3.2.2 Decision tree models
  3.2.3 Artificial neural network models

4 Method
4.1 Overview
4.2 Data set
  4.2.1 Data generation
  4.2.2 Data transformation
    Output space reduction
    Zero-centering and normalization
    Input space reduction
    Data augmentation
4.3 Training
4.4 Network architecture and settings
  4.4.1 Input layer
  4.4.2 Loss function constants
  4.4.3 Dropout
  4.4.4 Window size
  4.4.5 Transfer function
  4.4.6 Network initialization
  4.4.7 Early stopping
  4.4.8 Final configuration and validation accuracy
4.5 Platform
4.6 Presentation
  4.6.1 Complexity of the 3D model
  4.6.2 Rigging
  4.6.3 Blend shape implementation
  4.6.4 Baseline model
  4.6.5 Limitations and manual fixes
    Sequence modifications
    Consonant requirements and tongue animation
    Upper lip adjustments
    The post-processing of the Unity approach
4.7 Evaluation
  4.7.1 Subjective evaluation
Sample selection .......................... 65
The surveys ................................. 65
Statistical analysis ......................... 72
4.7.2 Objective evaluation .................. 75

5 Results ................................. 76
5.1 Subjective evaluation ................. 76
5.1.1 Sample demographics ............... 76
5.1.2 Results from the first survey ...... 79
5.1.3 Results from the second survey ... 81
5.1.4 Results from the third survey ..... 85
5.2 Objective evaluation .................. 87

6 Discussion ............................. 90
6.1 Limitations and obstacles ............ 90
6.2 The results ............................. 91
6.3 Ethical and societal aspects ......... 93
6.4 Conclusions ........................... 93
6.5 Future work ........................... 94

Bibliography .............................. 95

A The rendering pipeline ................. 104

B Acoustics .............................. 106

C Convolutional and recurrent neural networks ........................................ 107
C.1 Convolutional neural networks ........ 107
C.2 Recurrent neural networks .......... 109

D Optimization algorithms ............. 112

E Batch normalization .................... 115

F Complementary results ............... 116
F.1 The first survey ....................... 116
F.2 The second survey .................... 121
F.3 The third survey ...................... 126
Chapter 1

Introduction

This chapter will introduce the reader to the objective of the thesis. It begins with a brief (for detail, see Background) overview of the problem and the state of the art. This builds up to the specification of the problem statement. Then follows a description of the product that was developed and the company the project was conducted at. The product is described in this chapter in order to separate it from the academic content.

1.1 Intended reader

The knowledge of the reader is assumed to be that of postgraduate students in computer science and related subjects. The thesis provides an introduction to the theory of linguistics and machine learning that is necessary to fully understand the topic. Other readers may still find the results and the discussion interesting, however, and a full technical understanding should not be necessary.

1.2 Overview

This master’s thesis project aims to explore data-driven methods for synthesizing automatic visual speech, specifically from text input, on a 3D model. It focuses on the visual part, with the text-to-speech audio synthesis being provided by a third party in the final product, though no audio synthesis was needed to provide the academic results of this paper.
Visual speech is the animation of the mouth region of a computer generated character model while it is speaking, to mimic what real life speech looks like.

It has been shown in many studies that providing visual speech along with the audio improves the audience’s comprehension (Alexanderson et al. 2014; Cohen et al. 1993; Sumby et al. 1954; Siciliano et al. 2003). For example, Cohen et al. (1993) claimed that comprehension of speech increased from 55% when it was just audio to 72% when the participants could view the video of the speaker simultaneously. Visual speech has also been shown to stimulate the learning of new languages (Bosseler et al. 2003). This is related to the fact that providing the visual component can make users more engaged (Walker et al. 1994; Ostermann et al. 2000).

Computer-generated visual speech has been a research area since the 1970s, beginning with the works of Parke (1972). It can be considered as a subfield of computer facial animation and speech synthesis. Improving the efficiency of creating visual speech is a research question that has started to become more relevant since the late 20th century, when digital media such as movies and computer games started to feature more advanced computer graphics. This has increased the need for high-fidelity speech animation, and being able to efficiently produce it becomes a relevant topic since traditional means of creating visual speech either demand a lot of resources or do not appear realistic (Orvalho et al. 2012).

Parent (2012a) divides animation into three categories: artistic animation where the engine interpolates between handcrafted poses and transformations, performance capture animation (he referred to it as "data-driven" but the terminology was substituted here to prevent confusion with the data-driven computational models which are the subject of this thesis) where the data is captured through for example motion capture, and procedural animation which is controlled by a computational model with various rules.

The performance capture and artistic animation approaches for creating visual speech necessitates motion capture of live actors, with the added manual refinement by animators, or simply animation by hand from scratch. Both methods are labor intensive, but they can give very realistic results. They are therefore often used in movies, where performance capture is now the common way to generate visual speech (Orvalho et al. 2012).
The procedural paradigm can be subdivided into data-driven (or statistical) models and rule-based models. Both have to make some assumptions about speech, such as partitioning speech into a set of fixed units, such as visemes (Fisher 1968), to make the creation of the computational model a feasible process. Visemes can be thought of as the visual components of phonemes, and phonemes are essentially the sounds of speech (they are described in more detail in section 2.1.1, p. 9).

Rule-based algorithms use a number of heuristics and assumptions to generate the animation, and this is usually therefore a quick method suitable for games and other situations with a lot of content. The results appear less realistic, however, than actual motion capture or handmade animation by skillful artists. This is because too many assumptions of how speech works have to be made in the model, and visual speech is very complex. A common approach is to partition visual speech into various configurations of the mouth, such as visemes, and then interpolating between these and defining their interdependencies using more or less complex models.

Another way to produce visual speech animation in an automatic fashion is by using real-life data. These data-driven methods require less assumptions and leave it up to the data, and thus reality, to define the model to a greater degree. This is especially true when it comes to machine learning approaches, and in particular the more complex models such as artificial neural networks (ANNs), which are the focus of this thesis project. Machine learning models learn functions that mimic pre-recorded ground truth data on limited sets of samples, typically, when it comes to visual speech, recorded using motion capture. ANNs have grown more popular in a wide array of applications thanks to the increasingly available computational resources. Since the 1990s and the beginning of the new millennium, they have proven to surpass other approaches in computer vision, text-to-speech synthesis and more (Schmidhuber 2014; Fan et al. 2014). In the 2010s, several data-driven methods for visual speech have been explored, including a few involving ANNs such as Taylor et al. (2016), Taylor et al. (2017), and Karras et al. (2017), all of whom yielded more successful results compared to earlier approaches. This is usually measured by the difference to the ground truth data, and a subjective user study.

Depending on the application, different types of inputs and outputs are used (though non-procedural approaches do not really use
inputs in this sense). Some use text or phoneme data as input, others use audio. Some output a 2D video of processed images, while others output the animation data controlling a 3D model. The representation of the output does not necessarily change the implications of the results as drastically as it may intuitively sound, however. This is because the results of visual speech have more to do with the dynamics - the temporal curves of the transitions between the speech units or whatever structure is assumed. Thus, the representation is often less important when doing comparisons in subjective studies.

It is also possible to make a distinction between approaches simulating the facial musculature and approaches simply trying to visually mimic the effects of these physiological mechanisms (Massaro et al. 2012). Examples of the former can be found in Platt et al. (1981) and Waters et al. (1990), but it is more difficult and computationally demanding (Cohen et al. 1993). For this reason, virtually all related work, as well as this paper, focuses on the non-physiological approach.

1.3 Problem statement

This thesis aims to investigate how well artificial neural networks can automatically produce visual speech in Swedish, given text input. The objective is to produce natural and realistic looking animation, synchronized to the speech audio. This will be measured against the motion-captured ground truth data and a baseline model using surveys where the subjects are asked to compare the animations from the various models when played side by side, and objective measurements of the error against the ground truth.

The final algorithm produced by this project will, given text input, drive the geometry of the face of a 3D character model in a real-time chat bot application. The objective is for the ANN to be able to mimic the ground truth, which is achieved if the surveys show that they are indistinguishable and if the measured error is small.

1.4 The employer

The project is carried out at the AI department of Softronic, which is an IT-consulting firm located in Stockholm, Sweden.¹

¹https://www.softronic.se/om-oss/ (visited on 2018-03-07)
Their customers range from government authorities and municipalities to banks and others. Examples include Naturvårdsverket, Haninge kommun, 1177 Vårdguiden and Ticket.²

1.5 The product

Softronic have identified an interest for animated 3D chat bot avatars in Swedish. The solutions available at the time of writing were virtually non-existent. This section describes the resulting product of this thesis project in detail.

The final application allows a customer, e.g. a service provider or business, to provide its clients with a chat bot avatar rendered in WebGL which can be integrated into an HTML website, for example.

Since the ANN is run client-side and without GPU access, which TensorFlow’s JavaScript API does not support (the frameworks are described at the bottom of this section as well as in section 4.5, p. 56), inference takes too long to be done in real-time while chatting. Transitioning it to Node.js would likely solve this problem, however.

For the time being, the customer is able to customize the avatar’s appearance and voice in a JavaScript application, as well as insert new sentences to be stored at the push of a button, for real-time access. The front-end of the application can be seen in figure 1.1. The back-end of the application then calls the text-to-speech (TTS or audio synthesis) provider, Acapela³, which generates the audio file and viseme sequence. Since it is a pretty rough viseme partitioning, it does not make any distinction between phonemes that look approximately the same when pronounced. It was found that the phonemes of the text are directly mapped to the visemes (many-to-one), and that the mapping is context-independent. This means that it is possible to infer the actual phonemes using a dictionary of word-to-phoneme sequence mappings, word by word. Each viseme of the TTS sequence is mapped to an arbitrary phoneme belonging to the viseme, and each phoneme is then replaced with the corresponding phoneme (if it belongs to the same viseme) of the looked-up sequence. If a word is not found in the dictionary, an entry is written in an error log on the server. Acapela was chosen because it was unique at providing many different voices

²https://www.softronic.se/case/ (visited on 2018-03-07)
³https://www.acapela-group.com/ (visited on 2018-03-11)
for Swedish, and they were also customizable, meaning that the product is able to allow voice customization. The application finally generates the animation by sending the time-stamped phoneme data into the ANN, followed by some post-processing on the prediction, and then retargets it to the actual 3D model. The application then inserts this result into a database for quick access along with the character configuration. It also stores the audio file on the server. After this, the animation is played back. Thus, we get an automatic lip synchronization to the third-party TTS audio, played in parallel. This procedure is illustrated in figure 1.2.

![Figure 1.1: The user interface of the JavaScript application allowing customization of the chat bot avatar.](image)

A scenario of a typical use case of serving the avatar in a chat is that the chat bot, provided by the customer, receives a message from a client chatting with it. It then fetches an answer from the customer’s chat bot API (interfacing to e.g. a database of pre-defined replies or a machine learning algorithm) in text format. If the reply is stored on the server, the animation and audio are simply fetched and played back. Otherwise, the reply is sent to the back-end and an animation is generated as described above.

The avatar is customizable in terms of appearance, voice, gender, rate of speech and clothes, and this can be done in the application that the customer uses for avatar setup. This allows the customer to attain a unique profile for a company for example, and also customize its avatar to its clients’ needs and identity. For example, a service for
those new to a language might want a slower speaking avatar for increased comprehension (as mentioned above, visual speech does benefit both comprehension and language learning). It is also possible to retarget the rig to any arbitrary mesh. This is useful to create speaking anthropomorphic animals, for instance.

The front-end is made in JavaScript and HTML/CSS. It uses the 3D engine BabylonJS\(^4\) to drive WebGL, which means that real-time 3D graphics can be delivered to the client using no more extra data than the models and textures need (p. 57), and it is then processed on the client’s hardware. This means that virtually all (98%) modern browser and hardware, including smartphones, will be able to use it, without the use of plug-ins.\(^5\) The resulting product can be integrated into any HTML website.\(^6\) The front-end also makes use of the machine learning framework, TensorFlow\(^7\), which is a popular open-source solution. The back-end is made in PHP and is used to communicate with Acapela’s API and process the replies, as well as manage the database. The ANN itself is, however, run client-side (for now). In most cases, these computations will be made on the customer’s hardware during avatar setup in advance, while the chat itself only requires hardware capable of rendering a basic 3D model.

\[\text{Figure 1.2: Flowchart of the JavaScript avatar customization application. Caching mechanics etc. have been omitted. “BWS” is an acronym for “Blendshape weight sequence”}\]

\(^4\)https://www.babylonjs.com/ (visited on 2018-03-07)

\(^5\)https://webglstats.com/ (visited on 2018-03-07)

\(^6\)https://www.khronos.org/webgl/wiki/Getting_Started (visited on 2018-03-07)

\(^7\)https://www.tensorflow.org/ (visited on 2018-03-07)
Chapter 2

Background

This chapter will give an introduction to the technical details as well as some historical background of the area. First, the history of the research area of visual speech is provided. Then follows an introduction to the basic concepts of computer generated graphics. Finally, the reader is introduced to the area of machine learning and, in particular, neural networks.

2.1 Visual speech

This section begins with an overview of the theory and terminology of speech communication and phonetics, and then introduces the reader to the history of basic procedural models for computer generated visual speech.

2.1.1 Speech theory

Acoustics and physiology

Sound is the oscillation of molecules in the air. For speech, this oscillation is generated at first in the vocal cords, part of the so-called larynx, which receive the air pushed from the respiratory system. The vocal tract, above the larynx, then serves as a variable acoustic filter. It contains, for instance, the tongue, which is very important in manipulating the size and shape of the oral cavity (Case et al. 2012). The configuration of the articulators, i.e. the lips, tongue and so on, of the mouth controls the resonant frequencies, formants, which define the
various vowel sounds. The various articulators play different roles in
the production of speech, and visual speech aims to capture their com-
plex interaction. For more details about acoustics, see Appendix B.

Phonetics

In order to describe the various sounds in speech, some definitions
have to be made. The smallest components, usually even smaller than
syllables (a unit whose definition has not really been agreed on), are
called phones and phonemes (Case et al. 2012; Ladd 2011). Being a more
physiological and acoustic aspect of speech, phones describe the ac-
tual sounds produced. As a useful abstraction, the phones can be
grouped into sets whose elements do not change the meaning of the
word. Such a set of phones is defined as a phoneme. For instance,
the p’s in spat and pat are actually two different phones since the p
in pat is pronounced with aspiration (a puff of air), but are defined as
the same phoneme (Case et al. 2012). Thus, a phoneme is the smallest
unit of speech, or sound, that can turn one word into another.¹ This
means that one, or several, letter(s) can represent different phonemes
depending on the word, language and dialect. They are commonly
presented as a symbol within slashes, and will be presented in this
way throughout this paper. The International Phonetic Association
provides a standard for the notation (Ladd 2011). For instance, /p/ of
English pat and the /b/ of English bat are two different phonemes
(Case et al. 2012). There are about 27 phonemes in Swedish (though it
varies with the dialect) (Riad 1997).

Articulator movement and phoneme definitions

To produce the various sounds, the articulators will have to assume
various positions. The phonemes are grouped thereafter into differ-
ent types. The most important terms when discussing visual speech
are the ones that have a visible external effect. This could be through
both the motion of external articulators (and the internal ones visi-
ble through the opening of the mouth), but also more indirect effects
such as timing differences due to aspiration. Different phonemes will
involve different articulators and have different effects on the facial
geometry. Some even have certain requirements, such as complete lip

¹https://www.merriam-webster.com/dictionary/phoneme (visited on
2018-03-11)
closure, that must be met. Because of this, they can often be grouped into visemes, each able to represent one or more phonemes visually.

Consonants are pronounced by closing some part of the vocal tract, completely or partially (Case et al. 2012). The ones stopping the air flow completely, including the air flow though the nasal cavity, are called stop consonants, and the plosive ones release this puff of built-up air upon pronunciation. Examples of stop consonants are /p, t, k, b, d, g/. Nasal consonants allow air through the nasal cavities, by blocking the air through the mouth. Examples are /m, n/ (Case et al. 2012; Riad 1997). Fricatives allow air through a small gap between articulators, for instance between the tongue and the "ceiling" of the mouth when pronouncing the s in safe. Examples are /s, f/. Combining stop consonants with fricatives in turn produces affricates, such as the ch in chair. When the articulators are moved close, but not close enough to produce a fricative, approximants are produced. They include the y in yes and the r in red, whose corresponding phonemes are /j, r/. Tapping the ceiling of the mouth with the tip of the tongue swiftly, as in the dd in ladder, is simply called a tap or flap, and this is also common for the r in Swedish words such as korp. On the other hand, "rolling" the r by relaxing the tongue to allow the air stream to make it oscillate, is called a trill, which in Swedish can be more or less common depending on the dialect, the emphasis and the speaker. The consonants can be classified into primarily the following categories, according to Case et al. (2012) (though further subdivision is possible (IPA 2015)), and the examples are taken from both Case et al. (2012) and IPA (2015):

- **Bilabial**: E.g. /p, b, m/, where the lips close completely.
- **Labiodental**: E.g. /f, v/, where the lower lip touches the upper teeth.
- **Dental**: E.g. /θ, ð/, where the tip of the tongue touches the upper front teeth to produce the th in theta or the harder th in the.
- **Alveolar**: E.g. /t, d, n, s, z, r/, where the tip of the tongue touches the area right behind the upper front teeth (the front of the alveolar ridge).
- **Retroflex**: A certain type of r where the tip of the tongue goes a bit further back (the back of the alveolar ridge), found in Swedish in the assimilations of /r/ when followed by e.g. /d/ (Riad 1997).
• **Palatoalveolar**: /ś, ẓ/, where the tip of the tongue instead touches the postalveolar palate, in order to pronounce sh sounds.

• **Palatal**: /j/, where the back of the tongue (the **dorsum**) instead touches the hard palate, e.g. in order to pronounce the y in yes.

• **Velar**: /k, ɡ, ɳ/, where the tongue dorsum instead touches the soft palate, e.g. in order to pronounce the y in yes.

Opening the vocal tract to obstruct the air stream less, while having the vocal cords vibrate, generates **vowels** (Case et al. 2012). There are a few important terms for describing the positioning of the visible articulators in order to produce different vowels, since the timbre (the frequency distribution of the formants) of the vowels is regulated through the tongue position and the lip configuration. **Lip rounding** occurs when the lips are rounded to form the shape of an "o", and is present for example in the /u/ in *rule*. **Lip spreading** denotes the spreading of the lips, and occurs in the /i/ (**ea**) in *eat*. The **neutral** lip configuration occurs in the so-called **schwa** (ø) at the beginning of *about*. Based on the articulator configuration, they can be classified into the following categories:

• **Front cardinals**: /i, e, ɛ, a, æ/, which occur in e.g. English *feed*, *yes* and Swedish *bil*, *fall* and *nät* (Case et al. 2012; Riad 1997).

• **Back cardinals**: /u, o, ɑ, ø/, which occur in e.g. Swedish *borr* and *rot* (Case et al. 2012; Riad 1997).

• The **schwa**, ø, found at the beginning of *about*, is a more neutral vowel (Case et al. 2012).

One last important class are the diphthongs, which are like vowels, but typically describe a transition between two vowel phonemes. The configuration of the articulators, and thus the formants, transition from the setting of one vowel to another. Examples in English are plenty, such as *eye* and *mouse* (Case et al. 2012; Riad 1997).

Coarticulation

When it comes to modeling speech, models that are purely based on dividing speech into independent units such as phonemes will miss the important aspect of **coarticulation**. Coarticulation describes how
the articulator movements corresponding to the phonemes are in fact highly interdependent, and the phonemes are coarticulated. This is done in order to maximize the economy of motion. *Anticipatory* (also called *right-to-left*, or *forward*) coarticulation describes dependencies between a phoneme and future phonemes. On the other hand, *carry-over* (also called *left-to-right*, *backward*, or *perseveratory*) coarticulation describes dependencies between a phoneme and previous phonemes. The dependency can be several phonemes in length (Mattheyses et al. 2015). One time estimate is due to Schwartz et al. (2014), where it was found that visual coarticulation just in terms of anticipation can be in the 100 to 200 ms range, and the corresponding auditory time estimate is much shorter at approximately 40 ms.

There is much speculation as to how the neurological process is structured. Studies suggest, however, that coarticulation is learned over the years from childhood. For instance, Thompson et al. (1979) showed that older subjects demonstrated earlier anticipatory coarticulation.

In Swedish, it can be noted that as a result of coarticulation, /r/ is often assimilated into the following consonant such as /d/, /n/, /s/ or /t/ into retroflex consonants (Riad 1997). The long /r/, /r:/, is usually not assimilated, and it can only be long in Swedish when the preceding vowel is short.

The speaking rate also influences coarticulation. Fougeron et al. (1998) showed that in French speech, increased rate leads to pitch displacements and a reduction in pitch range. Lindblom (1963b) found through a spectrographic study of Swedish subjects that as rate increases, vowels are reduced to a schwa. Taylor et al. (2014) studied how phonemes are excluded upon increased speech rate. It was found that English speakers tend to omit /h/, /t/ and /j/ 40%, 20% and 18% of the time, respectively, and replace for instance /z/ with /s/ and /t/ with /d/ 16% and 9% of the time, respectively. The effects were more obvious visually than audibly.

Lindblom (1983) suggested two aspects of motor economy in speech: consonant-vowel coarticulation and vowel reduction. Consonant-vowel coarticulation is simply the coarticulation between a consonant and the neighboring vowel. Vowel reduction is the change of formants and duration of a vowel depending on the amount of following phonemes. He also suggested a model of two constraints, where the synergy constraint governed the articulator relationships,
while the rate constraint was simply the physiological limit of articulatory rate. Lindblom (1963a) also found a greater degree of VC (vowel-consonant) coarticulation than CV (consonant-vowel) coarticulation.

An early example of experiments to evaluate this phenomenon is due to Öhman (1966). In his paper, spectrograms show how the formant transitions are altered depending on the phoneme context, which in this case was defined by the first phoneme. It was observed, from a study of the spectrograms of VCV utterances (vowel-consonant-vowel, e.g. ada), that the speaker partially anticipates the last vowel upon production of the consonant, and also that the consonant is partially anticipated upon production of the first vowel, even when the vowel is prolonged.

MacNeilage et al. (1969) observed both anticipatory and carry-over coarticulation, by studying the articulator movements of a speaker during the production of CVC (consonant-vowel-consonant) utterances, with different phrase contexts. For instance, the last consonant was swapped between different recordings when studying the carry-over coarticulation between the first consonant and the vowel. It was shown that both types of coarticulation are present. As for carry-over coarticulation, the consonant did not affect the following vowel as much as the vowel affected the last consonant, and the differences between vowels were attributed to whether they were high or low. The articulators tended to "undershoot" when producing the consonant, likely because of physiological inertia. Carry-over coarticulation was observed in most cases, while anticipatory coarticulation was always observed. The physiological model suggested to explain the coarticulation is one where the consonant articulation is superimposed upon the transition phase of the diphthongs between the vowels. This is similar to the computational model suggested by Öhman (1967), described in the next section.

2.1.2 Rule-based models

Computer generated visual speech has been a research area since the 1970s, and the works of Parke. In 1972, he defined the first 3D model for simulating facial gestures (Mattheyses et al. 2015; Parke 1972). Since the human face is a complex non-rigid surface and computationally expensive to simulate, he approximated the human
face using 250 polygons and 400 vertices, which he deemed sufficient. For animation, he used cosine interpolation between blend shapes, to mimic the nonlinearity of the facial motion. Parke’s 3D model was later used by Massaro et al. (1990) and Beskow (1995).

Öhman (1967) suggested a computational model involving coarticulation, one year after his experiments on it in Öhman (1966). It models the articulator configuration over the duration of a VCV utterance as a function of the interpolation between the vowels, with the added displacement of the consonant’s target configuration at a degree (a factor) determined by a coarticulation function as well as a time-varying factor. This is known as a look-ahead model, which means that it uses anticipatory coarticulation as early as possible (Perkell 1990).

Perkell (1990) investigated several models, including the look-ahead model by Öhman (1967), but also a so-called time-locked model due to Bell-Berti et al. (1980), and a hybrid model. A time-locked (or coproduction (Cohen et al. 1993)) model is one where the anticipation begins at a fixed time, and not as early as possible as in a look-ahead model. The hybrid model had two phases - a gradual initial phase starting as early as possible, as well as a more rapid second phase. The conclusion was that any strong model of any of the three types was rejected, and a mixture was proposed instead.

Löfqvist (1990) defined so-called dominance functions as a way of modeling speech with coarticulation. A dominance function can be described as a pair of negative exponential functions to define a gradual increase followed by a decrease, signifying the dominance of a certain articulator configuration for a phoneme. For example, a consonant might have a low dominance for lip rounding, but higher dominance for the tongue, which means that the lip rounding and thus the external mouth shape for the consonant will be more influenced by neighboring vowels.

Cohen et al. (1993) then implemented the dominance model that was proposed by Löfqvist (1990) (described above), and it will be referred to as Massaro’s model from now on. For each point in time, the average of all segments’ values is chosen as the output. It was able to adapt to the rate of speech, in that the rate did not simply change the speed of the animation but changed it in a more natural fashion - increased rate led to more overlap of the dominance functions, resulting in what could be perceived as increased coarticulation. Decreased rate analogously led to less overlap, leading to more articulated speech.
However, Massaro’s model does not ensure requirements such as a closure in bilabials (Mattheyses et al. 2015; Beskow 2004), and this is also shown in the video comparison in Edwards et al. (2016), which featured Massaro’s updated model (Massaro et al. 2012) which had undergone data-driven training for setting the parameters (Beskow 2004).

An approach for simulating different levels of articulation is outlined in Edwards et al. (2016). The level of articulation, from hypo articulation (mumbling) to hyper articulation (over-enunciation), was parameterized into two parameters in a linear combination - JA (jaw movement level) and LI (lip movement level) - along with the weighted blend shape, spanning a two-dimensional plane of different levels of articulation. Without the JA and LI action, the speech animation would not be static, however, but mumbling. The input text was converted to phonemes using an HMM-based forced alignment tool, and they were in turn mapped to visemes. The coarticulation model was rule-based, and the visemes were processed for amplitudes and timings (by the authors referred to as articulation and coarticulation, respectively), by analyzing the lexical stresses and omitted syllables in the audio input. Some requirements were formulated, such as that the production of bilabials must involve lip closure. Other examples are that phonemes such as /l/ and /t/ are considered tongue-only and thus should not influence the mouth shape which was instead determined by the neighboring phonemes, and that lip-heavy phonemes are articulated simultaneously and with the lip shape of labiodental and bilabial neighbors. They simulated coarticulation up to two phonemes away, depending on the phoneme-specific time offsets taken from empirical time measurements from other studies. The algorithm is capable of extracting the JALI data from audio, through spectral analysis. Depending on the phoneme type (fricative, vowel, etc.), different frequency regions are analyzed and the JA and LI parameters are set depending on the intensity of the relevant frequencies for the phoneme type. The model was compared to the ground truth, the dominance model in Massaro et al. (2012) (described above) and the dynamic viseme model due to Taylor et al. (2012) (described on p. 33) in an objective test. The dominance model was the worst because of anticipation failure, excessive coarticulation and the lack of required lip closure for certain phonemes. The dynamic viseme model seemed to blur phonemes. The JALI method
over-enunciated certain phonemes, but still fared better than the other two models.

### 2.1.3 Multimodality and perception

Speech synthesis is a broad area, and this thesis is focused on the visual component. However, speech is multimodal, and is composed of both the auditory stream as well as the visual data from the movement of the articulators, such as the jaw, lips and the velum (Mattheyes et al. 2015). This multimodality has been shown to be important in human communication.

Visual stimuli has been shown in many studies to improve comprehension. Alexanderson et al. (2014) explored how well subjects could comprehend sentences by checking how many out of three keywords that the subjects managed to pick up, using different modes of speech for audio and video (quiet, normal and loud/"Lombard" speech). It was shown that in all combinations of modes, the addition of the visual component increased comprehension compared to the auditory components alone, typically by approximately one word. Also, the high-intensity visual modes increased comprehension further, since the articulatory movements were wider. This can be seen in the experiments involving incongruent modes, such as the one mixing the normal mode’s audio with the loud mode’s video compared to the normal mode’s audio and video. It is, however, not necessarily a good measure of how realistic or natural the speech looks. In an early work, Massaro et al. (1990) found that subjects were influenced by both visual and auditory information. Cohen et al. (1993) later claimed that comprehension of speech (sped up by a factor three) increased from 55% when it was just audio to 72% when the participants could view the video of the speaker simultaneously. The comprehension was only 4% when only the video was shown, meaning that the effect of presenting a video of the speaker along with the audio is superadditive. Siciliano et al. (2003) experimented with the increased intelligibility due to the visual component in English, Dutch and Swedish. It was found that intelligibility was increased by 20%. Synthetic faces were significantly worse at increasing intelligibility for all languages but Swedish. This was due to the synthetic model which was found to be lacking in pronouncing English consonants.

Neurological studies have also indicated the importance of the
multimodality of speech. One example, due to Benoit et al. (2010), showed that witnessing incongruent audio and video, i.e. a video showing a person pronouncing one phoneme along with the audio of another phoneme, causes a certain neurological response. This is because when the two components are perceived as incongruent, additional processing is required. The subsequent confusion from incongruent speech can lead to an illusion called the McGurk effect. McGurk et al. (1976) identified this, where subjects reported perceiving /da/ when actually seeing /ga/ and hearing /ba/, and the experiments conducted by Andersen (2010) showed, for instance, that subjects perceived the auditory /aba/ as /ada/ when the visual component was /ada/.

With the knowledge that the multimodality in speech leads to the stimuli influencing one another, and with the subsequent illusions in mind, it is clear that bad visual speech can have negative effects on comprehension.

Visual speech has also been shown to stimulate the learning of new languages (Bosseler et al. 2003; Massaro et al. 2012), and Massaro has over the course of many years developed programs featuring a character called Baldi to help deaf and autistic children learn words and grammar, and also other children to learn a second language. The results were positive (Massaro et al. 2012).

A psychological phenomenon to be taken into account is the uncanny valley. Mori, a robotics professor, discovered it in 1970 and it has not gained attention until decades later (Mori et al. 2012). He plotted the perceiver’s affinity for the humanoid model as a function of the model’s human likeness. The phenomenon is characterized by the fact that the function value increases fairly linearly up to 80%, after which the value dips into the uncanny valley, before ascending steeply towards 100%. The idea is that to maximize the perceiver’s affinity for the model, the developer should aim for either a low to moderate human likeness or make sure that the likeness is very high. It can thus paradoxically be considered safer to pursue less human likeness.

## 2.2 Computer generated graphics

This section introduces the reader to the concepts and state of the art of computer graphics and animation.
2.2.1 The structure of a model

In computer graphics, a 3D model is typically constructed by polygons or faces (and for most applications, they are rendered as triangles), that stick to each other at their edges, creating a mesh. They are defined by the positions of their vertices and in which direction their normals point (Bao et al. 2011a). In some cases, such as in the popular format OBJ created by Wavefront, the vertices contain additional information about the texture coordinates, how the light shading is smoothed across the mesh and more (Wavefront 2012). A triangulated 3D model with simple flat shading (where each polygon is considered flat and no smoothing is taking place) can be seen to the left in figure 2.1.

![Figure 2.1: A triangulated topology and a demonstration of the impact of textures. Rendered in the JavaScript application. From the left: Flat shading with white textures; Smoothed shading with diffuse textures; Smoothed shading with diffuse textures and normal maps.](image)

2.2.2 Textures and materials

There are usually various types of textures (also commonly referred to as maps) applied to 3D models. They are simply 2D images, and are mapped to the 3D mesh using the so-called UV mapping data of the model. The set of textures applied to an object and their settings are usually referred to as a material. The so-called diffuse or color texture defines the base color. The bump or normal texture alters the shading and the normals in order to create an illusion of depth in the sur-
face. The impact of diffuse and normal maps is demonstrated in figure 2.1. Other maps, such as specular, reflection and glossiness maps, can alter how shiny, reflective or glossy an object appears in different areas. In most graphics engines, it is also possible to define a transparency/opacity map. The eyebrows in figure 2.1 use it to allow for easy customization of their color and shape by simply changing the texture of their mesh instead of the whole body’s texture. The types of textures mentioned in this section are as of 2018 often found in real-time graphics engines like BabylonJS\(^2\) (used for the JavaScript application in this project), but more advanced rendering engines such as VRay\(^3\) often allow more physically accurate models and other taxonomies of textures.

### 2.2.3 Computational complexity and rendering

The complexity of the texture is decoupled from the 3D geometry complexity in that the texture can contain any amount of details to be mapped to any amount of polygons. This is why normal and bump maps are helpful in creating an illusion of detail while keeping the actual geometric complexity of the model down. The process of converting a 3D scene to the pixels of a 2D screen is commonly referred to as rendering (Salomon 2011). The complexity of the models is important to ensure that rendering the 3D scene is computationally feasible.

To assess a model’s computational complexity, the number of polygons (usually, at most, quadrilaterals but later typically converted to triangles) and vertices as well as the resolution of the textures should be observed. The polygon and vertex numbers are of great importance, since computations are done vertex-by-vertex and polygon-by-polygon.

Of course, when delivering models and textures over the Internet for client-side rendering as in the case of WebGL, the developer also has to consider the file sizes and the resulting downloading time for the client.

For a more in-depth description of the rendering process and light computations, see Appendix A.

\(^2\)https://doc.babylonjs.com/babylon101/materials (visited on 2018-03-09)

\(^3\)https://docs.chaosgroup.com/display/VRAY3MAX/Materials#Materials-V-RayMaterials (visited on 2018-03-09)
2.2.4 Animation and rigging

Various approaches can be used to animate a 3D model. One is to build a skeleton or rig, consisting of bones. Each vertex has a weight for each bone that it is assigned to, which defines how much the vertex follows the bone’s transformations. This process is often referred to as rigging or skinning. It allows the animator to animate a large set of vertices and polygons just by moving the bones and the skeleton, or by setting up an inverse kinematic rig to easily control longer chains of bones (Parent 2012b). It is commonly used for the body, but not so often for the facial geometry since the face can be tedious to animate with a large set of bones. An example of a bone rig can be seen in figure 4.6 (p. 59).

Other approaches are more common for the face. A common approach is the parameterized model, where the configuration of the geometry is a linear combination of different parameters. The parameters are preferably few, independent of one another, yet still cover as many configurations as possible together. A popular parameterized model, with conformational parameters, is due to Parke (1972). Other so-called expressive parameters include upper lip position and jaw rotation. Another approach is the use of blend shapes or morph targets, which define different configurations of the entire geometry. A facial expression can be expressed as a linear combination of the blend shapes, making a wide space of expressions possible. Finally, there are muscle-based models due to e.g. Platt et al. (1981), which are a type of parameterized model where the parameters are based on actual human physiology. However, muscle-based models can be difficult to implement and are also computationally expensive, when they are aimed towards physical accuracy (Parent 2012c).

It is also possible to directly animate the vertices, but that would be cumbersome for a complex mesh.

To generate the animation, the engine interpolates between the expressions or configurations that have been set by the animator or algorithm. Alternatively, a motion-capture performance or a procedural algorithm can be used (Parent 2012a).
2.3 Machine learning

This section will introduce the reader to the machine learning theory necessary to understand the rest of the paper and, in particular, artificial neural networks.

2.3.1 Fundamental and general concepts

Machine learning encompasses algorithms that use data to learn a pattern or function. It is a class of algorithms that has been important in driving artificial intelligence and the automation of society. Due to the increase of available computational power as well as storage capabilities, the use of machine learning algorithms to process vast amounts of data has seen a great rise of interest since the turn of the millennium. It typically involves optimization in the sense that the objective is to learn a function that minimizes some cost function, which in itself is an important design decision in that it needs to accurately model how correct the algorithm is.

There are two fundamental classes, unsupervised and supervised learning. Unsupervised learning infers some structure in the input data, for example a clustering of data points in some $n$-dimensional space. Supervised learning, which is the class that this paper is concerned with, learns some function from a set of training data. The training data contains input and output pairs, whose variables are often referred to as features, and the goal is to learn the underlying function that maps from each input element to the corresponding output element. If the task is a classification problem, with a discrete and typically finite output space, the function is a decision boundary that splits the data points into different classes. This is used to, for example, classify gender from pictures. If the task is a regression problem, the function maps input elements to real values. Visual speech is thus a regression problem, in that the objective is to learn a function that maps from a sequence of phonemes or a representation of audio to the corresponding displacements or parameters of the facial geometry.

It is not sufficient, however, that the inferred function maps correctly for most or all of the training data. Especially if the model is complex and the amount of training data is not enough, there is likely to be a tendency for it to overfit to the training data. Consider fitting
a curve to a set of points in a two-dimensional space. If the model is complex, the curve can be made to map to arbitrary data points, and map to all of the points in the training data correctly. However, such a function might not accurately represent the underlying distribution of the training data. This means that it will fail to generalize to new, previously unseen data, also commonly referred to as test data, during inference, i.e. run-time application, since it is too adapted to the training data. In these cases, regularization techniques are used. Common examples are penalizing too large or "dramatic" weight values by adding a regularization term to the cost function which is being minimized, to avoid the model being too ad hoc adapted to the training data. Also, to monitor how well the model generalizes to unseen data, a subset of the training set, called the validation set, is often used. This subset is not used by the algorithm for learning during training, but when overfitting is occurring, the cost function or error will typically increase on this subset even though it is decreased or stable on the training set. A dilemma encountered regarding the problem of overfitting is the bias versus variance trade-off. The bias is how much the model is subject to prior assumptions. The variance is how much the model varies with different training data. A complex model might thus have a low bias, but on the other hand a high variance and tendency to overfit. On the other hand, a simpler model will have a lower variance but a higher bias, instead leading to underfitting. To compensate the low variance of high-bias predictors, it is possible to combine them using ensemble techniques, where several predictors are trained in parallel using different initializations, parameters or different machine learning algorithms altogether, and then having their predictions averaged upon inference time.

Finally, when comparing models, previously unseen data must be used as the test data. This can not be done on the training or validation sets, since the model’s parameters have been determined using them, giving the model an unfair advantage (if compared against other models not trained on the same data set). The accuracy on the data will then not be a good indicator of the generalization ability. The goal of machine learning algorithms is usually not to minimize the cost function on the training data, but on unseen data. This data can be from the same distribution as the training data (i.e. gathered from the same source and using the same methods). It can also be from an entirely new distribution, which naturally is an even more difficult case and
puts the model’s ability to generalize to the test.

It is then possible to measure e.g. the loss or the accuracy (the percentage of correct predictions) on the test set when the model has been trained. The test set needs to be large enough for this to be a good generalization measurement, hence a common split ratio is to partition it into 50% training data, 25% validation data and 25% test data.

If the data set is small, it is also possible to perform k-fold cross-validation. If the number of elements in the data set is \( n \), this is done by splitting the data set into \( k < n \) (e.g. 5 or 10) roughly equal subsets (\( k = n \) is known as leave-one-out cross-validation), and then train on the union of \( k - 1 \) subsets and test on the subset not included in the training subsets. This is then done once for each of the \( k \) possible selections of train and test subsets, and the resulting \( k \) different test accuracies or losses is then averaged to yield the cross-validation test loss, an estimate of the test loss. Cross-validation can also be used when splitting the data into training and validation data sets.

### 2.3.2 Artificial neural networks

Artificial neural networks (ANNs) are inspired by the biological neurons in our brains. Just like the neurons in our brains are connected to each other to form a network, ANNs can be shaped in many ways for different purposes to encode time dependencies, simulate memory, model nonlinear dependencies (meaning that the output is not simply a linear combination or multiple of the input features) and more.

The original neuron, or perceptron, introduced by McCulloch et al. (1943), was an attempt at modeling the biological neuron (Bishop 2006). The neuron takes in a sum of simultaneous inputs, which together form the activity in the neuron, and, if the activity is above some threshold, the neuron is excited.

#### General structure

Basic feed-forward neural networks are divided into layers, with the nodes in the \( i \)th layer passing their outputs into the nodes of layer \( i + 1 \). Typically, all neurons in layer \( i \) are connected to all neurons in layer \( i + 1 \), but sparse networks are also used (Bishop 2006). The smallest have just one input layer and one output layer (one-layer neural networks). The output is just a linear combination of the input in this case, i.e. there is only one layer of weights, which are on the connections.
The input layer has as many neurons as there are features in the input data. Each hidden layer has an arbitrary number of neurons, which can be chosen according to the desired complexity. The hidden layers introduce nonlinearity by having nonlinear transfer functions (or activation functions), which map the input levels of the neurons to their corresponding output levels. An ANN with multiple hidden layers is called a deep neural network (DNN).

The nonlinear transfer functions are important, because the function defined by a network with linear transfer functions in the hidden layer neurons can also be represented without the hidden layers, since a composition of linear transformations is in itself simply a linear transformation (Bishop 2006). Examples of common nonlinear transfer functions are \( \tanh(x) \), \( \text{ReLU}(x) = \max(0, x) \) (Rectified Linear Unit), and \( \sigma(x) = \frac{1}{1+\exp(-x)} \) (sigmoid). Sigmoid transfer functions are not recommended, since they squash the output to \([0, 1] \) with a saturated activation at these interval borders and thus kill the gradients. In fact, tanh transfer functions have the same problem. Both of them also suffer from the expensive computation of \( \exp(x) \). The ReLU transfer function, on the other hand, is cheap to compute and does not saturate, which makes network training several times faster than when using saturating transfer functions (Krizhevsky et al. 2012). It obviously has zero-gradients for negative activations, however, which can result in some weights never being updated (see the gradient descent formulation below). Instead, using leaky ReLUs can solve this: \( \text{LeakyReLU}(x) = \max(\alpha x, x) \), where \( \alpha \) is a small constant, e.g. 0.01. It will still introduce nonlinearity and is even faster than regular ReLUs, and also does not have any dead gradients (Maas et al. 2013).

Finally, the signals come to the output layer, which is typically just a linear combination of the last hidden layer, and has as many neurons as there are features or classes in the output data, depending on the task.

It can be detrimental to introduce too many layers (and it is not only because of overfitting), but residual nets can help fight the issues by having a bypass around each layer, making the network learn \( F(x) \) in \( H(x) = F(x) + x \) where \( F(x) \) is a layer and \( H(x) \) the target function of the layer, instead of \( F(x) \). This means that if the number of layers is superfluous, it is easy to learn the identity function for some of the
layers by setting the weights to zero in $F(x)$, and the approach won ILSVRC 2015 using 152 layers (He et al. 2015a). This is a lot compared to e.g. the Imagenet 2014 winner Simonyan et al. (2014) with 16-19 layers. A trick to reduce the computational complexity of added layers is to use $1 \times 1$ convolutions to simply reduce the depth, and use residual bypass as well (He et al. 2015a).

The neurons calculate a weighted sum (Bishop 2006). Each input, or dendrite, has a weight $w_i$ and an input value $x_i$. Typically, one of the inputs, $x_0$ is a bias, which is a constant input value, while the other $N$ inputs $x_1, x_2, \ldots, x_N$ are the outputs, or axons, of the neurons in the previous layer. The activation for the jth neuron, or the score, $s_j$, is thus:

$$s_j = \sum_{i=1}^{N} w_i x_i + x_0$$

This can also, for an entire layer, be expressed in matrix notation as:

$$s = W x + b$$

where $W$ is the matrix of weights for the layer, and is of size $K \times d$ where $K$ is the number of outputs of the layer and $d$ the number of input neurons, $x$ is the vector of input neurons ($d \times 1$) and $b$ is the vector of biases ($K \times 1$) for the output neurons. The result, $s$, is then the vector of output neuron scores ($K \times 1$).

After this (though only in hidden layers), $s_j$ is sent into a transfer function, $h$, yielding the final output $z_j$ of the neuron:

$$z_j = h(s_j)$$

The neuron is thus very similar to the perceptron due to Rosenblatt (1962), and thus, the feed-forward neural network is often referred to as a multi-layer perceptron (MLP). However, the difference is that the perceptron uses a step function, whereas the transfer functions in a neural network are continuous and thus differentiable with respect to the parameters (e.g. weights), an important feature when training the network (Bishop 2006).

Finally, in the output layer, a function could be applied to the scores of the layer which can be both positive and negative. In classification tasks, the $\text{softmax}$ operation gives a probabilistic interpretation of the values of $s$, normalizing them, yielding the vector of probabilities $p$ for the various classes: $\text{softmax}(s) = \frac{\exp(s_j)}{\sum_{s_i \in s} \exp(s_i)}, \forall s_j \in s$, after which the class with the highest value in $p$ is chosen as the prediction.
Convolutional and recurrent neural networks

*Convolutional neural networks* (CNNs), a category of DNNs, use *convolution* and *pooling* layers to downsample the signal, allowing different layers to process different representations of the data. They are widely used today in pattern processing tasks, especially when processing images, because of their ability to use more abstract and less complex representations of them, thus making the computations more feasible.

*Recurrent neural networks* (RNNs) are neural networks that use a time delay to incorporate some of the information of the previous time steps in the current calculation.

For more information on these classes of neural networks, see Appendix C.

Network initialization

Weight initialization can be more or less important, depending on the optimizer and transfer functions used. Some optimizers are better at avoiding local minima despite bad initializations. When using saturating transfer functions, the initialization needs to avoid the saturated ranges of the transfer functions.

Glorot et al. (2010) derived an initialization scheme, commonly referred to as *Xavier initialization*, based on the variance of the layers’ weights. It attempts to preserve the variance in layers during forward and backward propagation and initializes the weights of layer \( i \), \( W^{(i)} \), so that \( \text{Var}(W^{(i)}) = \frac{2}{n_i + n_{i+1}} \), where \( n_i \) is the number of weights in layer \( i \) and \( W^{(i)} \) is the weight matrix of layer \( i \). This implies initializing it like so: \( W^{(i)} \sim N(0, \frac{2}{n_i + n_{i+1}}) \). Since Xavier initialization was not derived with ReLU in consideration, He et al. (2015b) derived an initialization method for ReLU transfer functions. In this approach, \( \text{Var}(W^{(i)}) = \frac{2}{n_i} \), and thus initialized according to: \( W^{(i)} \sim N(0, \frac{2}{n_i}) \).

Training

When the network is to be trained, typically the squared error is minimized, or some other loss function. When it comes to classification tasks, cross-entropy loss can be used instead, to compare the predicted distribution (the model predicts the probabilities of the different class labels) to the true distribution. It is computed by simply normalizing
the probability for the labels, taking the predicted probability value for
the correct label, $p_y$, and then calculating $-\log(p_y)$.

A regularization term is often added, to control the complexity of
the model. This regularization term can simply be, for example, the
magnitude of the weights, i.e. $\lambda||W||^2 = \lambda \sum_{w_{ij} \in W} w_{ij}^2$, where $W$ is
the matrix of weights and $\lambda$ is the regularization factor, controlling how
hard to regulate the weights. This is known as $L_2$ regularization. Additionally, it is possible to use dropout, which is when random neurons’
activations are set to zero while training. A different random selection
of such "dead" neurons is used for each training sample. This can be
viewed effectively as training different models in an ensemble, espe-
cially when doing a so-called Monte Carlo approximation where several
forward passes are performed and the average prediction chosen. It
forces the network to have a redundant representation, in the sense
that complex co-adaptation between different neurons is made much
more difficult, since neurons can no longer depend on each other’s
presence (Krizhevsky et al. 2012). The CNN due to Krizhevsky et al.
(2012) needed dropout in order to stop overfitting, however it con-
verged slower with it. They then, upon test time, simply multiplied
the outputs of all neurons with 0.5, which is approximately equivalent
to taking the mean of the predictions of different network representa-
tions due to dropout. Another way to decrease the overfitting is to use
data augmentation. It is a simple way to generate more training data,
by applying various transformations (but keeping the ground truth
for these transformed copies). Applicable transformations for pho-
tographs include translation, rotation, cropping, distortion and more
(Li et al. 2016a). For instance, Krizhevsky et al. (2012) used random
alternations of the RGB channels, translations and flipping, and so did
Simonyan et al. (2014), but with the addition of cropping as well. If the
amount of data still is not enough, it is common to use transfer learning,
where the model is trained on a larger similar data set first. Then, de-
pending on the size of one’s specific data set, only a few of the higher
(more abstract) layers are trained, e.g. just the final fully connected
layers, with a small learning rate, while the weights of the earlier lay-
ers are frozen. If one’s data set is larger, more layers may be trained
(Li et al. 2016a).

For regression tasks, such as the one this paper is concerned
with, we can let the total error, or loss (or cost), $l$, for $n$ predic-
tions $z_1, z_2, \ldots, z_n$ (one for each input element) be defined as
\[ l = \frac{1}{n} \sum_{i=1}^{n} (y_i - z_i)^2, \] where \( y_1, y_2, \ldots, y_n \) are the ground truth samples from the training data. When it comes to RNNs, the loss function can be a sum of the loss of all time steps.

To adapt the network with respect to the loss and thus train it for one iteration, *backpropagation* is a common method. First, the network’s output is calculated (the input is propagated forward through the network), after which the loss \( l \) can be calculated for this iteration. Now, the loss is backpropagated through the network, to compute the gradients of the loss with respect to the weights, i.e. how much the loss depends on each weight. The gradients of the loss with respect to each weight can be calculated using the chain rule since it is a composition of functions:

\[ \frac{\partial l}{\partial w_{ij}} = \frac{\partial l}{\partial s_i} \frac{\partial s_i}{\partial w_{ij}} \]

where \( w_{ij} \) is the weight from neuron \( j \) to neuron \( i \), \( s_i \) is the activation in neuron \( i \), and neuron \( i \) is in the output layer. The loss is backpropagated through neuron \( i \) back to \( w_{ij} \). The loss is then backpropagated further to calculate the gradient with respect to the other weights.

When the backpropagation is finished, the parameters (weights and biases) will be adjusted. This can be done using various methods, but a simple way is to do it proportionally to the gradient, in *gradient descent*:

\[
W^{(t+1)} = W^{(t)} - \eta \frac{\partial l}{\partial W} \\
b^{(t+1)} = b^{(t)} - \eta \frac{\partial l}{\partial b}
\]

where \( \eta \) is the learning rate and \( W^{(t)} \) and \( b^{(t)} \) are the matrices of weights and biases, respectively, for a certain layer and a certain time \( t \). In a network with more than one layer, the gradient is propagated to earlier layers, and there is one set of \( W \) and \( b \) matrices for each layer.

There are other optimization techniques as well. In gradient descent, the loss is calculated over an entire data set, the parameters are updated, and then the procedure is iterated again. Thus, the update is calculated after an entire *epoch* (iteration over the entire data set). *Stochastic gradient descent* (SGD) converges faster, and involves calculating the loss and updating the parameters for each randomly selected data element, resulting in many updates per epoch, but this can be noisy. Instead, it is common to partition the data set into random equal-sized subsets called *mini-batches*, and then gradient descent
is performed on each mini-batch. The size of the mini-batches has to be set, typically manually, as part of the hyper-parameters, which also include the regularization factor and the learning rate. For each mini-batch $D$, a simple loss function with mean squared error (MSE) and $L_2$ regularization would be:

$$l = \frac{1}{|D|} \sum_{(y,z) \in D} (y - z)^2 + \lambda \sum_{w_{ij} \in W} w_{ij}^2$$

where each $(y, z)$ pair is the prediction and ground truth for a sample of the mini-batch.

When training, monitoring the loss on the training set and the accuracy on the validation set over the epochs will give some indication as to how well the hyper-parameters are set. If the learning rate is set appropriately, the loss curve should decay exponentially, i.e. approximately look like $f(x) = \frac{1}{e^x}$ (but can have high-frequency oscillations). This is simply because too high learning rates could make the gradient descent algorithm step too far and miss the local minimum, step around it back and forth, or even diverge from it, and too small learning rates will just make the learning very slow. An exponentially decaying loss curve means that the parameters are descending down the valley of a local minimum, and this descent will be steeper at first because the randomly initialized parameters are likely far from optimal. If the learning rate is small enough, the loss will converge after a while. A bad initialization is obvious if the loss remains stable for a long while in the beginning, and then suddenly starts to decay. At the same time, the accuracy on the validation and training sets must be watched so that they decrease in a similar fashion - having the training accuracy or loss decrease and the validation accuracy increase is a sign of overfitting.

When performing SGD, the ratio of the step to the magnitude of the weights, $r = \frac{-\eta \partial l}{||W||}$, can indicate if the learning rate is too small or large ($r$ should be roughly around the order of $10^{-3}$). This is also reflected in the fact that it is ideal to make larger steps when the gradient is small, and smaller steps when the gradient is large, or else the algorithm will risk either converging slowly or stepping over the local minimum (Sutton 1986). Ideally, the learning rate should be adapted, to preserve $r$.

For a description of the more advanced optimization algorithms, including the ones with adaptive learning rates, see Appendix D.
When training, learning is optimized by having unit Gaussian activations. This means that the activations are distributed like a unit Gaussian, with a normalized variance, i.e. $\sigma^2 = 1$, and mean (zero-centered), i.e. $\mu = 0$. The fact that data can sometimes shift in mean and variance when taken from different sources is referred to as internal covariate shift (Ioffe et al. 2015). The shift is not only due to the data itself, but also to the shifts out of the outputs of previous layers due to the training procedure. Thus, depending on the source of the data, the target function to be modeled could be different, and also, the inputs to the transfer functions could be drastically different. Because of this, some transfer functions, especially saturating ones, can give rather meaningless outputs and dead gradients when the inputs are not fairly zero-centered. By normalizing the data at each layer, this effect is prevented. ReLUs will then on average be closer to introducing their nonlinearity around 0 without the weights in $W$ having to approach extreme values or the gradients of saturating functions becoming saturated. This technique, called batch normalization (Ioffe et al. 2015), requires that for each mini-batch, the mean $\mu$ and standard deviation $\sigma$ for the layer and the batch is calculated, in order to normalize the activations. For more details, see Appendix E.

Setting the hyper-parameters such as the learning rate, regularization factors and so on, can be done through a grid search (or a random search, since it is even more efficient (Bergstra et al. 2012)) and with, if the data set is small, cross-validation. If the data set is large enough, simply splitting it into fixed training and validation sets is feasible. The procedure is to simply run a few epochs of training using parameters from a uniform distribution within a wide interval (identified using just a few sparse quick tests), i.e. in a "grid" (or using a random search on the interval), and then validate using the validation set. The interval is then successively shrunk, and the number of epochs for each test is increased, as the ideal hyper-parameters are further approximated.
Chapter 3

Related work

This chapter will introduce the reader to the state of the art of data-driven procedural approaches to visual speech in chronological order. For the earlier history of rule-based models, see section 2.1.2 (p. 13). First, a summary of the main works can be found in table 3.1.

Table 3.1: A summary of the main papers on (mainly) data-driven procedural approaches, in reverse chronological order. Referred studies found in the table are marked in bold.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Input predictor</th>
<th>Output predictor</th>
<th>Main findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karras et al. (2017)</td>
<td>Windowed audio sequence</td>
<td>CNN (12 layers)</td>
<td>Preferred 87% and 23% of the time when compared against Massaro et al. (2012) and the ground truth, respectively, with only 3-5 minutes of training data.</td>
</tr>
<tr>
<td>Taylor et al. (2016)</td>
<td>Windowed audio sequence</td>
<td>DNN (3 hidden layers, 2,000 units each)</td>
<td>Outperformed the state-of-the-art HMM approach at the time, due to Wang et al. (2015).</td>
</tr>
<tr>
<td>Fan et al. (2016)</td>
<td>Phoneme and audio sequences</td>
<td>BLSTM-RNN (3 hidden layers, 128 units each)</td>
<td>Outperformed, in order, the dynamic visemes and dominance models due to Taylor et al. (2012) and Massaro et al. (2012), respectively, in error against samples from KB-2k. Not data-driven, however.</td>
</tr>
<tr>
<td>Edwards et al. (2016)</td>
<td>Phoneme sequence</td>
<td>N/A</td>
<td>Outperformed, in order, the dynamic visemes and dominance models due to Taylor et al. (2012) and Massaro et al. (2012), respectively, in error against samples from KB-2k. Not data-driven, however.</td>
</tr>
<tr>
<td>Kim et al. (2015)</td>
<td>Windowed phoneme sequence</td>
<td>Decision tree (random forests)</td>
<td>Outperformed, in order, the dynamic visemes and dominance models due to Taylor et al. (2012) and Massaro et al. (2012), respectively, in error against samples from KB-2k. Not data-driven, however.</td>
</tr>
<tr>
<td>Zhang et al. (2013)</td>
<td>Audio sequence</td>
<td>DNN-HMM (7 layers, 2,000 units each)</td>
<td>Outperformed, in order, the dynamic visemes and dominance models due to Taylor et al. (2012) and Massaro et al. (2012), respectively, in error against samples from KB-2k. Not data-driven, however.</td>
</tr>
<tr>
<td>Taylor et al. (2012)</td>
<td>Phoneme sequence</td>
<td>Dynamic visemes</td>
<td>Favored to static visemes. Used 8 hours of data (KB-2k set).</td>
</tr>
</tbody>
</table>

Indistinguished from ground truth data.

Outperformed, in objective study, and won by 47 and 50 out of 50 votes in the preference study in the same order. Ground truth won over it by 40 out of 50 votes.

Outperformed, in subjective study. Used 8 hours of data (KB-2k data set).

Outperformed, in subjective study. Used 8 hours of data (KB-2k data set).

Indistinguished from ground data.

Outperformed, in order, the dynamic visemes and dominance models due to Taylor et al. (2012) and Massaro et al. (2012), respectively, in error against samples from KB-2k. Not data-driven, however.
3.1 Data-driven rule-based models

Deng et al. (2006) investigated triphone visemes, in order to try to capture enough context to provide coarticulation. The triphone visemes were learned from motion capture data and audio from speech.

De Martino et al. (2006) presented another data-driven method using context-dependent visemes. By collecting the first stationary points (i.e. the viseme targets) of the facial motion of each phoneme in different contexts, various context-dependent visemes could then be identified by k-NN clustering, as well as the temporal dynamics. Diphones and triphones were mapped to different visemes. This is thus a similar approach to Deng et al. (2006), mentioned above. Cubic splines were used for interpolation. They claimed that the model captured anticipatory and perseveratory coarticulation well. The model has fewer parameters than Cohen et al. (1993) and similar rule-based models (see section 2.1.2), and makes relatively few assumptions. The influence of the rate of speech on coarticulation and the dynamics was not taken into account, however, and was mentioned as a possible future step for their work.

A physically based data-driven model was presented by Sifakis et al. (2006), using a muscle model based on Sifakis et al. (2005). Coarticulation was handled by capturing muscle activation data (derived from motion capture data) not just for the duration of each phoneme, but also in some of its neighborhood, which extracted what is essentially the dominance data. Thus, they captured the temporal dynamics as well as the displacements. A problem was that when the rate of speech increased, the model did not capture the inertia-related coarticulation effects.

3.2 Machine learning models

3.2.1 Hidden Markov models

When it comes to hidden Markov models (HMMs), the HMMI method due to Choi et al. (2001) was the best HMM that Shengli et al. (2005) found, and for this reason also the one used in the comparisons in Taylor et al. (2016). It is a modification of the HMM approach, and stands for Hidden Markov Model Inversion. What sets it apart from other
HMM approaches is the fact that it does not use the Viterbi sequence, which is sensitive to noise in the input (Shengli et al. 2005).

Ben Youssef et al. (2009) trained HMMs by using cepstral coefficients from audio. Different contexts were tested. The output was then compared to EMA-captured motion data.

Another HMM approach is outlined in Wang et al. (2012). HMM models were compared against rule-based dominance models. The model was developed for the purpose of pronunciation training. The dominance model proved to be better, but more training data for the HMMs would be needed, as well as longer sequences as input to them.

Taylor et al. (2012) noted that a fixed mapping from phonemes to the visual speech units is an inaccurate assumption, due to coarticulation and the fact that a phoneme, by definition, is a group of sounds that have the same meaning in a language. An active appearance model (AAM) was used for gathering motion capture data of 34 vertices for the sentences of the KB-2k data set, which was created for the project. They defined a gesture as a certain articulator configuration, and a dynamic viseme as a group of gestures that have the same visual function. An HMM was used to cluster the gestures into these dynamic visemes. The clustering revealed that phonemes are widely distributed among the dynamic visemes, meaning that one and the same phoneme indeed takes on a wide array of visual characteristics depending on the context, and over 90% of the dynamic visemes spanned over two phonemes. The approach led to 150 dynamic visemes, each containing several phoneme sequences. For a sequence of phonemes, a sequence of dynamic visemes that contains all of the phonemes is searched for, using a cost function that minimizes the discontinuity between the dynamic visemes (which have to be interpolated between). This means that depending on the context, a phoneme sequence can map to many different dynamic viseme sequences. A user preference study showed that dynamic visemes were favored to static visemes.

### 3.2.2 Decision tree models

A decision tree approach was explored in Kim et al. (2015). Random forests were used to predict the displacements of the articulators using a sliding window of 11 frames in width over a phonetic sequence, after which the average of the predictions for a certain frame was chosen. An error term to deal with input data errors was also introduced,
to handle misalignment errors between video and audio as well as missing values (e.g. from the occlusion of reference points in motion capture). The output was an AAM of the KB-2k actor’s mouth region, i.e. a parametric model of the face. Several benchmark experiments were conducted. The one for visual speech compared the error of the predictions of the approach to those of the dynamic visemes approach from Taylor et al. (2012) and the HMM approach from Zen et al. (2007) for the KB-2k data set (due to Taylor et al. (2012)). The approach significantly outperformed the HMM-based approach, which in turn significantly outperformed the dynamic visemes approach. A user preference study was also conducted, where users saw the mouth region of the character driven by two of the approaches side by side and asked to vote for the most natural-looking animation. Winning 47 and 50 out of 50 votes when compared against the HMM and dynamic visemes approaches respectively, this experiment confirmed the results of the benchmark experiment. The ground truth was still preferred over it with 40 votes to 10, however.

3.2.3 Artificial neural network models

Since the beginning of the new millennium, ANN approaches have become more common, especially in recent years.

An early attempt at implementing an artificial neural network, or a time-delay neural network (TDNN) to be precise, for visual speech is provided by Massaro et al. (1999). The network used the MFCC representation of audio as input. It was compared to a rule-based model (which used phoneme labels as input) in a lip reading test. The TDNN was found to be worse in this regard. However, this does not really tell us how it fared against the rule-based model in terms of realism, only how well it articulated.

A fairer comparison between ANNs and state-of-the-art rule-based models is provided in Beskow (2004), who compared two ANNs against the time-locked model by Cohen et al. (1993) (based on (Löfqvist 1990)) and the look-ahead model due to Öhman (1967) (both are described in section 2.1.2, p. 13). Two experiments were conducted. One was to compare the outputs to ground truth data and observe the root mean squared error (RMSE) and the correlation coefficient. Cohen et al. (1993) turned out to be significantly better that the rest, having both the lowest RMSE and the highest correlation.
coefficient. Another statistically significant result was that one of the ANN models showed a higher correlation coefficient than Öhman (1967). The other experiment was a lip reading test, which did not find any statistically significant differences between the four approaches.

A comparison between RNNs and bidirectional TDNNs (incorporating both backward and forward context) is found in Savran et al. (2006). The best model was the bidirectional TDNN with nine frames centered on the frame to be predicted.

Zhang et al. (2013) defined a model that mapped from audio (MFCC) to 9,000 different triphone states using a DNN. Then, an HMM mapped the states to the lips images. In a, notably small, user preference study, the model was indistinguishable from the ground truth.

Fan et al. (2016) used a bidirectional LSTM recurrent neural network (BLSTM-RNN) with BPTT to control AAM model parameters using audio (through MFCC). It was a deep BLSTM-RNN in the sense that multiple BLSTM layers were stacked. Each layer took the data both forwards and backwards and passed the prediction onward to the next layer, after the last of which a prediction was made for the frame. The output was essentially a stitched sequence of 2D images. It was then compared to Wang et al. (2015) (which was identified as a state-of-the-art triphone HMM approach) in a user preference study, which showed that the BLSTM-RNN was preferred most of the time. The HMM did not require nearly as much training, however.

Taylor et al. (2016) outlines a DNN approach using a sliding window and audio input (using MFCC). The best input and output window sizes were found to be 340 ms and 100 ms, respectively. The input window size is thus roughly equal to the one used in Kim et al. (2015) (11 frames at 30 frames per second). The KB-2k data set (8 hours, 2500 sentences) from Taylor et al. (2012) was used for training, and a grid search was performed to set the hyper-parameters, choosing the parameters yielding the smallest error on a small validation set after 20 epochs. The model had three hidden layers (which was found to be best out of 1-7 hidden layers) of 2,000 RLUs each (best out of 100-4,000), with 0.5% dropout (best out of 0-0.9%) and a learning rate of 0.0001 (best out of 0.00001-0.01), with linear units in the fully connected output layer. Batch normalization was also used. Backpropagation was used during training, with NAG and an MSE loss function. Benchmark comparisons were made against the HMMI approach from
Choi et al. (2001), since it was shown to outperform other HMM approaches (Shengli et al. 2005), to drive an AAM of the KB-2k actor’s mouth region. The HMMI approach was much worse because of its discontinuous trajectories, especially showing problems at phoneme boundaries due to the many state changes. A user preference study was also conducted, where the users rated how realistic they deemed the animations to be. Ground truth, rendered in the same way as the other approaches, was also included to control for rendering artifacts. HMMI, this approach, and ground truth were perceived as real 19%, 68% and 78% of the time, respectively. It was also found that velar consonants yielded a greater error than e.g. fricatives. The authors explain this by the fact that velar consonants rely more on the use of the tongue, and the mouth shape is thus affected more by coarticulation. Short pauses were found to be the most difficult aspect, especially when they were longer than the input window size, which eliminates the available context. They did not, however, find any correlation between phoneme length and error.

One year later, a new DNN approach was explored in Taylor et al. (2017). This time, it used phoneme sequences instead, with the animation of an actual 3D model as output. Sliding windows is justified by the fact that it directly learns coarticulation from the data, as opposed to e.g. Edwards et al. (2016) and Taylor et al. (2012), and it is claimed that it is better at capturing this highly local context than RNN approaches which capture a much longer context (thus often requiring more training data). DNNs also require minimal assumptions and no ad hoc interpolation, which is good since the temporal dynamics, i.e. the required interpolation curve, can vary a lot between visemes. Training was performed on the KB-2k data set once again (8 hours, 2500 sentences), and each face configuration was parameterized as a weighted sum of modes (effectively blend shapes). 16 modes were able to cover 99% of the variation. The sliding window approach means, as in previous approaches, that several predictions are made for each frame (according to the input window size), and that the predictions overlap each other (according to the output window size), and the mean is chosen as the final prediction for each frame. The model was a fully connected feed-forward neural network with three fully connected hidden layers with 3,000 units (using a hyperbolic tangent transfer function) each and with 50% dropout, followed by a fully connected output layer. Mini-batches of size 100 with SGD were used for
training, minimizing the squared loss against the ground truth AAM. In this approach, the best input and output window sizes were found to be 11 and 5 frames, respectively, which is like in Taylor et al. (2016). When testing other input-output window size pairs, it was found that, when increasing them from 1, the error declined greatly down to when having 7 and 3 frames, respectively, and converged at around 11 and 5 frames, respectively. They also experimented with, instead of the raw feature representation, using a more linguistically motivated one, with features such as "The phoneme is a nasal consonant". It was found that this led to a slight improvement. For evaluation, the squared loss of the model’s output (represented by an AAM of the KB-2k actor’s mouth region) on the test data from the KB-2k data set was compared to that of the HMM approach from Zen et al. (2007) (identified as the state-of-the-art HMM model), the dynamic viseme model from Taylor et al. (2012), the LSTM model due to Fan et al. (2015) and Fan et al. (2016) (which was found to have outperformed basic HMMs), as well as the decision tree model due to Kim et al. (2015). The model outperformed the rest, but was tightly followed by, in order, the decision tree model and the HMM model, and, much further behind, the LSTM model and the dynamic visemes model. Another user preference study for the same test set as well as data from new speakers (meaning the distribution comes from another actor than the one trained on), where users saw two animations side by side and chose which looked the most natural, was also conducted. The model significantly outperformed the other models by between 38 and 50 wins out of 50 sentences from the KB-2k data set, but without a statistically significant difference against the ground truth. It also significantly outperformed the other models by between 15 and 24 wins out of the 24 sentences from novel speakers.

Another DNN approach, which like Taylor et al. (2017) is part of SIGGRAPH 2017, was explored in Karras et al. (2017). It used audio as input and the output was the vertex positions for a fixed mesh. It was thus not weighted blend shapes like in most of the approaches mentioned above. The paper mentions the issue of procedural and rule-based methods not accomplishing convincing results for non-phoneme sounds. This weakness is also mentioned in Taylor et al. (2016), described above. The audio was segmented into 500 ms non-overlapping segments, and the network output the configuration for the middle of the segment. No memory was kept of previous
outputs - the predictions are made segment by segment, as the network was completely feed-forward. First, the audio was sent into five convolutional layers, which were aimed at learning the features of the audio. The first layer took the formant information. The window was 520 ms, i.e. 260 ms in each direction, and was divided into 64 overlapping audio frames and 32 autocorrelation coefficients in each (instead of MFCC which yielded worse results). It was found that the window could be decreased to 100 ms in each direction, but not further than that without sacrificing the performance. This was also claimed by Schwartz et al. (2014), mentioned above. With each layer, the formant features (the number of autocorrelation coefficients) were shrunk while the abstract features were increased, in order to "push" the formant information to the abstract internal representation. The features were thus roughly transitioned from formants to intonation, emphasis and phonemes. The next five convolutional layers, called the articulation network, then processed the abstract representation by similarly shrinking the number of frames (64 originally in the first five layers) until it reached 1 in the final layer. Thus, the convolution subsampled the time dimension in this part of the network. Finally, two fully connected layers expanded the representation to the 3D positions of 5022 vertices. In addition to this, the mood was also input to the network, with the ambition to separate mood and speech and thus eliminate the problem of ambiguous data (resulting from different moods). The emotional state vector (typically 16- or 24-dimensional) was learned from the training data, and concatenated into the representation between each layer in the articulation network. Having it concatenated into each layer might not have been necessary, but it was thought that it made the emotion affect every level of abstraction, including different levels of coarticulation (each layer subsampled the time further). After culling many emotion vector values, only 33 remained. They can then be used upon inference to affect the emotion of the visual speech. To further address the problem of ambiguous input and output data pairs (different outputs for the same input), which tend to lead to the network simply outputting the average of them, a three-way loss function was implemented. It penalized having too many mood swings. It also had an MSE over the vertex position error, as well as a term that penalized having any vertex motion not found in the data. The motion term helped against temporal instability. The model was then trained for 500
epochs, minimizing this error function, using the Adam optimizer with default parameters, and random mini-batches consisting of 50 temporal pairs. Some data augmentation was used - pitch-shifting, reverb, time-stretching, random equalization and random nonlinear distortion did not help, but bidirectional time-shifting by about 16 ms did. For regularization, multiplicative noise to the input of every convolutional layer (which is recommended by Srivastava et al. (2014)) was also used. The remaining jitter, mostly noticeable for certain phonemes, was solved by ensembling by performing two predictions for every frame with 4 ms in-between them, with the average chosen as the final prediction, as well as some Gaussian smoothing for the eyebrows. The amount of training data was notably small, at only 3-5 minutes. For evaluation, 20 subjects were asked to judge which of two approaches looked the most natural in side-by-side comparisons of novel sentences. The approach was compared to the dominance model from Massaro et al. (2012) and the motion capture data for 13 audio clips which were 3-8 seconds long. The approach was preferred 87% and 23% of the time when compared to the dominance model and motion capture data, respectively. Notably, just like the neural network due to Taylor et al. (2017) (which was by objective results better than Taylor et al. (2012), which was in turn better than Massaro et al. (2012) according to Edwards et al. (2016)) it is thus also superior to the dominance model, despite the drastic difference in training data size - Taylor et al. (2017) used eight hours of training data (and had phonemes as input rather than audio, the latter of which is likely more demanding in terms of data) compared to 3-5 minutes.
Chapter 4

Method

In this chapter, the method for investigating the problem statement (section 1.3, p. 4) is explained and justified. It explains the data generation and the implementation details as well as the statistical methods used to compare the final algorithm against the ground truth and a baseline model, which are also explained here. The choices regarding the hyper-parameters and various techniques are motivated in terms of the accuracy on the validation set.

The other components of the implementation, such as graphical user interfaces and external TTS engine integration, are not relevant to the academic component of the project and not elaborated upon here. For an overview of the product, the reader is referred to section 1.5 (p. 5).

4.1 Overview

The artificial neural network implemented was inspired by the latest state of the art in ANN applications to visual speech by Taylor et al. (2017) and Karras et al. (2017) from SIGGRAPH 2017, explained above on pages 36-37 and 37-39, respectively.

The input to the network was a phoneme sequence, just like in e.g. Taylor et al. (2017). When training as well as during inference for the experiments of this project, the input data was simply text and audio that were converted to sequences of phonemes. The training procedure is illustrated in figure 4.1.

The network used a sliding window over the input sequence to capture the local temporal dynamics, and also a sliding output win-
dow for the duration of which a subsequence of weights was predicted for each position of the window. This is an approach also encountered in Kim et al. (2015), Taylor et al. (2017), and, though with audio input, in Karras et al. (2017). The sliding window approach in Taylor et al. (2017) greatly outperformed both the decision tree approach due to Kim et al. (2015) (which also used sliding windows) and the dynamic visemes approach due to Taylor et al. (2012), in both objective and subjective experiments. Thus, using sliding windows was identified to be the state-of-the-art approach. The intuitive explanation for this is that, as mentioned in Taylor et al. (2017), the context of coarticulation is highly local in time.

Padding consisting of $K_x - 1$ frames, where $K_x$ is the width of the window, was inserted on both sides of the sequence so that all frames of the actual sequence got an equal amount of windows positioned over them. The padding frames added in the beginning of the phoneme (input) and ground truth sequences were equal in phoneme and weight values to the first frames of the respective sequence, and the ending padding was handled correspondingly.

The output of the network was the vector of weights for the blend shapes defining a facial configuration, which was simply a linear combination of the blend shapes. Similar output representations were also used in e.g. Taylor et al. (2017) and Edwards et al. (2016). Let $K_x$ be
an odd integer denoting the length of the window. Similarly to the approach in Taylor et al. (2017), the network outputs the curves of the blend shape weights for the length of each window. Thus, for each frame $i$ and parameter (blend shape weight) $p$, there were $K_x$ predicted values, one from each of the windows centered at $\{i - (K_x - 1)/2, i - (K_x - 1)/2 + 1, \ldots, i, \ldots, i + (K_x - 1)/2\}$. The resulting output was then the average of these $K_x$ values. The procedure is illustrated in figure 4.2.

Figure 4.2: Illustration of the windowing scheme when $K_x = 5$, showing two windows $i$ (it would be third window in this case) and $i + 1$, their predictions and their resulting averaged prediction for one output feature (blend shape $j$), but only for the four frames where they overlap each other. The first two padded frames have been omitted, as well as the rest of the $K_x$ windows affecting each frame.
4.2 Data set

4.2.1 Data generation

The data set was generated using the deep-sensing camera of the Iphone X and its ARKit framework\(^1\), which was a relatively simple way to perform motion capture. The output was, for each frame, the corresponding weights for the weighted sum of pre-defined blend shapes defining the facial expression, at 60 frames per second. This was later re-sampled to 30 frames per second because of the added computation times and judging the additional 30 frames per second to be unnecessary.

The blend shapes used initially are presented in table 4.1.\(^2\)

When recording, a simple application extracted the video along with the blend shape weights and the vertex positions as defined by ARKit. The audio from the videos was automatically labelled with phonemes and their time stamps using HMM-based forced alignment, a method also used in many other studies (Alexanderson et al. 2014; Edwards et al. 2016; Taylor et al. 2017). In this case, the Montreal Forced Aligner\(^3\) was used to convert from text (first transcribing it to a phoneme sequence word by word using a dictionary) and audio to a phoneme sequence using a set of 45 phonemes, including three "phonemes" denoting silence, pause and null. This also included long and short versions of each vowel and the Swedish assimilations of /r/ (the retroflex consonants) as the individual phonemes /t̚, d̚, s̚/ and so on.

The data was generated by the author, speaking sentences taken from the NST database of phonetically balanced Swedish sentences (CLARINO NB – Språkbanken 2016). The MFA model was trained on this database, and then this thesis project used sentences from the training set that the MFA model was trained on to make sure that the sentences were labelled as correctly as possible.

611 sentences were recorded. The first 311 sentences are found in the file r4670001.spl, which contains the sentences of the first speaker.

\(^{1}\)https://developer.apple.com/documentation/arkit (visited on 2018-08-17)


of the NST data set (NST contains hundreds of utterances spoken by various speakers). The rest of the sentences were sentences 13 through 312 in r4670299.spl. However, since the recording of sentence 26 from the latter set was corrupted, 610 sentences remained. The sentences had 4,564 words for a total of 46 minutes and 21 seconds.

It was later found that the quality of the rendering of some of the blend shape files in the JavaScript application was unacceptable. This could either be because of limitations in the Iphone and the recording or because the manually defined blend shapes in the JavaScript application (only a subset of the ARKit blend shapes) and the implementation of them (including the rigging) might not always work well with the ARKit blend shapes (see section 4.6 for details about the rigging and blend shape implementation). There are limitations in the BabylonJS engine in terms of the possible number of blend shapes (see section 4.6.5, p. 60), so it was necessary to implement blend shapes manually as different configurations of the rig’s bones. Most of the problematic recordings were found to be unacceptable even in the Unity setting of the third survey, which did not have these restrictions and used a pre-defined character model. Thus, sentences ended up being removed from the data set to make sure only those yielding a satisfactory output in the JavaScript application were kept.

In the end, 469 sentences remained, with 2,823 words for a total of 31 minutes and 8 seconds. Sentence lengths varied between 1 and 23 words. The data set was also split in three parts. First, the test set used for the subjective evaluation was a set of ten moderately long sentences (see table 4.4, p. 73). Having reserved these ten sentences for the test set, the validation set consisted of the 50 first sentences and the training set of the last 409 sentences of the remaining set.

The data was sampled at the desired framerate of 30 frames per second out of the original data recorded at 60 frames per second, by linearly approximating the ground truth weights for each frame. This made computations more manageable and is a sufficient frame rate also used in related papers, such as Karras et al. (2017) and Taylor et al. (2017).
Table 4.1: The ARKit blend shapes used and their effects on the 3D model. Blend shapes later removed in the output space reduction (p. 46) are marked with asterisks.

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>jawForward</td>
<td>Forward movement of the lower jaw</td>
</tr>
<tr>
<td>jawLeft*</td>
<td>Leftward movement of the lower jaw</td>
</tr>
<tr>
<td>jawRight*</td>
<td>Rightward movement of the lower jaw</td>
</tr>
<tr>
<td>jawOpen</td>
<td>Opening of the lower jaw</td>
</tr>
<tr>
<td>mouthClose</td>
<td>Closure of the lips independent of jaw position</td>
</tr>
<tr>
<td>mouthFunnel</td>
<td>Contraction of both lips into an open shape</td>
</tr>
<tr>
<td>mouthPucker</td>
<td>Contraction and compression of both closed lips</td>
</tr>
<tr>
<td>mouthLeft*</td>
<td>Leftward movement of both lips together</td>
</tr>
<tr>
<td>mouthRight*</td>
<td>Rightward movement of both lips together</td>
</tr>
<tr>
<td>mouthSmileLeft</td>
<td>Upward movement of the left corner of the mouth</td>
</tr>
<tr>
<td>mouthSmileRight</td>
<td>Upward movement of the right corner of the mouth</td>
</tr>
<tr>
<td>mouthFrownLeft*</td>
<td>Downward movement of the left corner of the mouth</td>
</tr>
<tr>
<td>mouthFrownRight*</td>
<td>Downward movement of the right corner of the mouth</td>
</tr>
<tr>
<td>mouthDimpleLeft</td>
<td>Backward movement of the left corner of the mouth</td>
</tr>
<tr>
<td>mouthDimpleRight</td>
<td>Backward movement of the right corner of the mouth</td>
</tr>
<tr>
<td>mouthStretchLeft</td>
<td>Leftward movement of the left corner of the mouth</td>
</tr>
<tr>
<td>mouthStretchRight</td>
<td>Rightward movement of the right corner of the mouth</td>
</tr>
<tr>
<td>mouthRollLower</td>
<td>Movement of the lower lip toward the inside of the mouth</td>
</tr>
<tr>
<td>mouthRollUpper</td>
<td>Movement of the upper lip toward the inside of the mouth</td>
</tr>
<tr>
<td>mouthShrugLower</td>
<td>Outward movement of the lower lip</td>
</tr>
<tr>
<td>mouthShrugUpper</td>
<td>Outward movement of the upper lip</td>
</tr>
<tr>
<td>mouthPressLeft</td>
<td>Upward compression of the lower lip on the left side</td>
</tr>
<tr>
<td>mouthPressRight</td>
<td>Upward compression of the lower lip on the right side</td>
</tr>
<tr>
<td>mouthLowerDownLeft</td>
<td>Downward movement of the lower lip on the left side</td>
</tr>
<tr>
<td>mouthLowerDownRight</td>
<td>Downward movement of the lower lip on the right side</td>
</tr>
<tr>
<td>mouthUpperUpLeft</td>
<td>Upward movement of the upper lip on the left side</td>
</tr>
<tr>
<td>mouthUpperUpRight</td>
<td>Upward movement of the upper lip on the right side</td>
</tr>
<tr>
<td>cheekPuff</td>
<td>Outward movement of both cheeks</td>
</tr>
<tr>
<td>cheekSquintLeft</td>
<td>Upward movement of the cheek around and below the left eye</td>
</tr>
<tr>
<td>cheekSquintRight</td>
<td>Upward movement of the cheek around and below the right eye</td>
</tr>
<tr>
<td>noseSneerLeft</td>
<td>Raising of the left side of the nose around the nostril</td>
</tr>
<tr>
<td>noseSneerRight</td>
<td>Raising of the right side of the nose around the nostril</td>
</tr>
</tbody>
</table>
4.2.2 Data transformation

Output space reduction

The first data transformation which was found to improve the training process was utilizing the simplifying assumption that facial expressions are symmetrical. Asymmetrical expressions can be considered noise in the data and due to imperfections in the actor. They were eliminated by merging the output features that had left and right counterparts into a side feature by simply keeping only the left part, which was an arbitrary choice. For example, the blend shapes jawLeft and jawRight were merged into jawSide which simply was assigned the values of jawLeft.

After this, removing the output features that were deemed too difficult to predict was also beneficial. They do not assist in predicting other parameters, and can be considered noise. This was done by training the network and plotting the SMAPE (symmetric mean absolute percentage error, which measures the relative error) for each individual output feature. This was done both for the training set and the validation set. The features that were consistently found to be unpredictable were jawSide, mouthSide and mouthFrownSide, and they were removed, and marked with asterisks in table 4.1. This was not a surprise, because jawSide and mouthSide were originally two features each (left and right pairs) denoting sidewards movement in a global direction, and not a movement inwards or outwards from the center which would be possible to mirror horizontally. Thus, the merge of certain left and right features (described in the beginning of this section) made them superfluous, contradicting the assumption that all gestures are symmetrical. However, mouthFrownSide denotes vertical movement, but perhaps a movement not significant to speech. This resulted in reducing the output feature space from ARKit’s original 32 mouth-related blend shapes to 18 blend shapes:

jawForward, jawOpen, mouthClose, mouthFunnel, mouthPucker, mouthSmileSide, mouthDimpleSide, mouthStretchSide, mouthRollLower, mouthRollUpper, mouthShrugLower, mouthShrugUpper, mouthPressSide, mouthLowerDownSide, mouthUpperUpSide, cheekPuff, cheekSquintSide, noseSneerSide.

To get an understanding of the magnitudes of the errors and
losses, the resulting distribution of the ground truth weight values after the output space reduction is illustrated in figure 4.3, and had a mean of 0.091 and a standard deviation of 0.088.

![Graph showing probability density and weight values distribution](image)

**Figure 4.3:** Density, rug and box plots illustrating the distribution of the weight values in the ground truth after output space reduction.

**Zero-centering and normalization**

Zero-centering and normalizing the data was also attempted. Bear in mind this is not zero-centering as in centering the mean around zero (p. 30), but rather a zero-centering of the start and end poses. Zero-centering was done by subtracting a linear combination of the configuration in the first frame and the configuration in the last frame
from each frame of the case, so that the first and last frames were zero vectors and the middle frame had half of the first and last frames subtracted from it. If we let $x_f$ be the weight vector for frame $f \in [0, n - 1]$, the value was given like so: $x_f = x_f - \left( \frac{f}{n-1} \cdot x_{n-1} + (1 - \frac{f}{n-1}) \cdot x_0 \right)$. This simplifies the network’s task because it will always start and end in the same neutral configuration, and it will learn the offsets from this neutral configuration. It consistently improved the training and was used.

Normalizing the output features to a common range of e.g. $[0, 1]$ should prevent features of smaller magnitudes from not affecting the gradient of the error less. However, it did not improve the performance and was therefore not used. An explanation for this is that the output features remaining after the output feature pruning described above (p. 46) were of roughly the same order of magnitude.

**Input space reduction**

As for the input space, reducing the number of phonemes led to very slight but consistent improvements when experimenting on smaller subsets of the data set, but in the end was not beneficial on the full data set, no matter the network complexity. Several runs with different parameters were conducted to be able to attribute the results to performance rather than chance. An explanation for this is that while a smaller input space and thus a less complex model still likely requires less data to achieve convergence, the model using the full input space was able to achieve a smaller error when using the entire data set since it is now sufficiently large. The configurations tested are described here.

A naive first approach tried reducing the original 45 phonemes or tokens to 25, by merging long vowels with their short versions. /d/ and /t/ were merged with each other, and their corresponding retroflex consonants /q/ and /t/ were merged as well. /s/ and /z/ (as found in the Swedish words "tjog" at the beginning and in "fors" at the end, respectively) were also merged, but /i/, as in the beginning of the Swedish word "sjal", was left as a stand-alone phoneme. Also, some vowels were merged such as /æ/ with /e/ and /e/, as well as /y/ with /i/, because of their similarities. Finally, the silent "phonemes" (silence, short pause and null) were also merged since they should have the same effect on speech. This naive approach did
not work very well, however.

The best approach, which was beneficial on the smaller data subset, was found after iteratively merging different combinations (essentially trial and error). It merged long vowels with their short versions, and also /g/ with /k/ and /f/ with /v/. In the end, despite over ten sessions with different network parameters, it was never found to be as good on the full data set as when using the full input space.

### Data augmentation

Data augmentation (section 2.3.2, p. 27) is an important method of improving generalization and preventing overfitting. In Karras et al. (2017), noise was input to each layer to prevent overfitting (as recommended by Srivastava et al. (2014)). Another relevant type of data augmentation, also found in Karras et al. (2017), is random time-shifting, where the data was linearly interpolated (which gave better results than cubic interpolation) to provide a new time-shifted ground truth pose. The amount of time-shifting was up to half a frame. Various other augmentation approaches exist (Li et al. 2016a; Krizhevsky et al. 2012; Simonyan et al. 2014), though these were handling picture data. For the purpose of this thesis project, the noise and random time-shifting approaches used in Karras et al. (2017) were deemed relevant.

The project attempted inserting and removing frames at random. When inserting, the inserted frame got the mean value of the neighboring frames. The phoneme assigned to the inserted frame was that of the neighboring frames, which both had the same phoneme since only frames surrounded by frames having the same phonemes were considered. When removing, the neighboring frames $y_{i-1}$ and $y_{i+1}$ were each mutated with the removed frame $y_i$ like so: $y_a = 0.5 \cdot y_a + 0.5 \cdot y_i, \forall a \in \{i - 1, i + 1\}$, where $i \in [2, n_x - 1]$ and $y = \{y_1, y_2, \ldots, y_{n_x}\}$ is the ground truth sequence. Various probabilities for removing and inserting frames were attempted, but this data augmentation technique was not found to improve the results and was thus not used.

### 4.3 Training

Training was performed by feeding the network with the phoneme data and then evaluating how well the blend shape weights of the network’s output match the weights extracted in the motion capture (see
To do this, a loss function was defined. In most earlier ANN approaches, there was at least one MSE term. These include Karras et al. (2017) and Taylor et al. (2016), and, although just a regular squared error, Taylor et al. (2017). Karras et al. (2017) additionally used a so-called motion term to discourage movement of articulators not moving in the training data. This was to improve temporal stability. It was defined as the MSE of the difference of movement (i.e. the finite difference of positions in adjacent frames) in the output and the data. (A regularization term was also introduced for the emotion vector used, but that would have been irrelevant for this project.)

Let $y = \{y_1, y_2, \ldots, y_n\}$ be the ground truth sequence for some arbitrary case, $x = \{x_1, x_2, \ldots, x_n\}$ be its corresponding phoneme sequence, $f(\cdot)$ be the function computed by the network, and $\theta$ be the set of model parameters. Each of the $n_x$ elements of $y$ and the prediction $\hat{y} = f(x|\theta)$ consists of $n_b$ output parameters or features. The loss function $l(y, \hat{y})$ used in this paper was the sum of the MSE of the blend shape weights in the prediction versus the ground truth, an $L_2$ regularization term to control the network weight magnitudes, and, inspired by Karras et al. (2017), a motion term $\phi(\cdot)$. $\phi(\cdot)$ was defined as the mean absolute difference (the vertical bars indicate absolute value) of the movements of the parameters in the prediction versus the ground truth in each frame:

$$\phi(y, \hat{y}) = \frac{1}{n_x - 1} \sum_{i=1}^{n_x-1} \sum_{j=1}^{n_b} |(y_{i+1,j} - y_{i,j}) - (\hat{y}_{i+1,j} - \hat{y}_{i,j})|$$

The introduction of $\phi$ was found to consistently improve the results. The loss function was then:

$$l(y, \hat{y}) = \frac{1}{n_x} \sum_{i=1}^{n_x} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n_l} ||W_j||^2 + \varphi \phi(y, \hat{y})$$

where $W = \{W_1, W_2, \ldots, W_{n_l}\}$ is the set of weight matrices of $n_l$ layers and $\lambda$ and $\varphi$ are the factors for the regularization and motion terms, respectively.

To calculate the updates of the network, the Adam optimizer (p. 113) due to Kingma et al. (2015) was used, since it was clearly among the best methods. It was used in many earlier studies, such as Karras et al. (2017) and Suwajanakorn et al. (2017). Taylor et al. (2017) used SGD, while Taylor et al. (2016) used NAG. Both of these are in fact
outperformed by the Adam optimizer (Kingma et al. 2015). Additionally, Karras et al. (2017) used a ramping schedule for the learning rate, ramping it up tenfold at first and then decreasing it gradually, with a ramping down using a smooth curve the last 30 epochs out of 500 epochs. In this project, however, the learning rate, \( \eta \), was decreased according to the series \( \{10^a, 5 \cdot 10^{a-1}, 10^{a-1}, 5 \cdot 10^{a-2}, 10^{a-2}, \ldots \} \), where \( \eta = 10^a \) initially, whenever the training loss increased. In this case, \( a \) was set to \(-3\) initially, i.e. \( \eta = 10^{-3} \), and then \( \eta \) was decreased as described. All other optimizer parameters were TensorFlow’s default values.

Training in mini-batches of size 100, like Taylor et al. (2017) and Karras et al. (2017) did, was attempted. Additionally, it is usually of great benefit to use batch normalization (p. 30). Using batch normalization was proven in earlier studies to make training faster even for networks using ReLU transfer functions (Ioffe et al. 2015). Examples of other approaches using it are Taylor et al. (2016) and He et al. (2015a) with ReLUs, and, in a sense, Karras et al. (2017). However, the use of mini-batches and batch normalization was not found to improve the training process and was thus avoided. Instead, updates were made case by case.

### 4.4 Network architecture and settings

#### 4.4.1 Input layer

When receiving the data, it has to be represented in a suitable way. In this case, it was a sequence of phonemes, each phoneme represented by a discrete scalar value. Since there is no stronger relation between two phonemes that happen to be assigned neighboring identity values than those who are not, the network is not supposed to learn a mapping from the identity scalar to some geometric displacement, but needs to be able to completely distinguish different phonemes. In other words, there is no assumption that the phoneme indices are of ordinal nature. To avoid treating them as such, each frame’s phoneme scalar was converted to a one-hot vector. Thus, the input layer had \( K_x \times n_p \) neurons where \( n_p \) is the amount of phonemes, each frame of the current window being fed into its own \( n_p \) neurons.
4.4.2 Loss function constants

In the loss function (p. 50), there were two constants that needed to be set. The best values, found via a grid search, were \( \lambda = 1.8 \cdot 10^{-8} \) and \( \varphi = 1.0 \cdot 10^{-4} \).

4.4.3 Dropout

Dropout (section 2.3.2, p. 27) is a common form of regularization. It was used in e.g. Taylor et al. (2017) and Krizhevsky et al. (2012) with 50% probability, and in Taylor et al. (2016) with 0.5% probability. It does not simply make the model less complex, but discourages complex co-adaptations between different neurons. Various values were attempted, from 0% to 80%, and it was found to be beneficial to the training procedure. After a grid search and some random searching, it was used in the first and second hidden layers with 60% and 10% probability, respectively.

4.4.4 Window size

The window size is an important design decision, as larger windows make the input and output representations more complex, incurring a greater need for training data. At the same time, the input and output windows need to be long enough to be able to capture the context and output the corresponding curvature. Increasing the input window size beyond 11 frames (367 ms) did not decrease the error in Taylor et al. (2017), but it would be expected for the error to increase if the context was too large, an effect not observed or examined in that study. In Karras et al. (2017), decreasing the context from their chosen 520 ms led to noticeably worse performance when going below 200 ms. Schwartz et al. (2014) found that just the anticipatory coarticulation context is between 100 to 200 ms. Chandrasekaran et al. (2009) measured the time-to-voice (the length of the anticipatory movement before the voice is audible) of the phonemes /m p b f/, and found that the mean time-to-voice of the phonemes were 137-240 ms after a pause and 127-188 ms after a vowel. Since phoneme lengths are in the approximate range of 100-200 ms (Taylor et al. (2016) measured 330-390 ms for /s/, and Lindblom (1983) claimed the average syllable length was 160-200 ms), Taylor et al. (2017) and Karras et al. (2017) really only used up to approximately 4 and 5 phonemes as context, respectively. This thesis
project started using $K_x = 17$ frames at 30 frames per second, i.e. 567 ms of context, because it was thought to be a good starting point and possibly optimal considering the references above. The optimal $K_x$ was found to be 15 frames, however, i.e. 500 ms, which is still in line with the aforementioned references.

### 4.4.5 Transfer function

The transfer function (section 2.3.2, p. 24) is another important design decision. Taylor et al. (2017) used a hyperbolic tangent transfer function ($\tanh(x)$), while Karras et al. (2017) and Krizhevsky et al. (2012) used ReLUs (except in the fully connected layers in the former). ReLUs are clearly a better option than hyperbolic transfer functions, but leaky ReLUs are even faster than regular ReLUs, and also do not have any dead gradients (Maas et al. 2013). Thus, this project tried using both ReLUs and leaky ReLUs in all hidden layers.

Leaky ReLUs were found to outperform ReLUs at alpha values (the slope of the activation function when $x < 0$) between 0.1 and 0.2, and the best configuration used an alpha value at 0.1.

### 4.4.6 Network initialization

Weight and bias initialization (section 2.3.2, p. 26) needs to be taken into account even when using optimizers with momentum. In this case, it was found to be beneficial to use He initialization (see p. 26) for layers with ReLU transfer functions. Layers without ReLU transfer functions were initialized according to the heuristic $W^{(i)} \sim N(0, 0.05)$. Biases were initialized to zero.

### 4.4.7 Early stopping

After regularization has been applied to mitigate overfitting, a complementary way to avoid it in the saved model is to keep track of the validation error in relation to the training error. If the error increases on the validation set while decreasing on the training set, the network is overfitting to the training set. This project kept track of the best validation error and saved the state when a new validation minimum occurred. When the validation accuracy had not decreased in ten epochs, the training procedure was terminated.
4.4.8 Final configuration and validation accuracy

The ANN does not have to be very deep when the input is simply a sequence of phonemes. Karras et al. (2017), who, however, used audio as input, used a fairly deep network with 12 layers in total, including two fully connected layers at the end. They changed the dimensions of the representation with the depth, and they claimed this allowed the learning of different abstract representations, from learning formants to emphasis and phonemes out of audio. The network in Taylor et al. (2017), on the other hand, was only five layers deep with an input layer, three hidden layers (3,000 units per layer) and one fully connected output layer. Since the input representation for this project was a sequence of phonemes just like in Taylor et al. (2017), and also uses the sliding window technique, this project used a similar structure as a starting point.

After experimenting by carefully starting with one hidden layer with varying sizes around the size of the input layer and adding both smaller and larger layers only when they were found to improve the validation accuracy, and also complementing with random searches regarding the dropout rates and sizes, the best architecture was found and is summarized in table 4.2. \( n_f = 45 \) is the number of phonemes in the input space, while \( n_b = 18 \) denotes the number of blend shapes in the output space. It is relatively small, comparing to related works. The most similar among them, Taylor et al. (2017), which also mapped a phoneme sequence, by sliding windows, to a blend shape weight sequence, used a lot more data, allowing larger architectures to train longer without overfitting, and also had a much smaller output space with only eight blend shapes. The parameters in table 4.2 led to, at best, a validation error of \( 1.865 \cdot 10^{-3} \) after 13 epochs. The validation error was the MSE on the validation set, after inverting the zero-centering of the predictions (p. 47). Other configurations rarely got below \( 1.94 \cdot 10^{-3} \), and naive configurations with two hidden layers with the same loss function constants typically ended up at around \( 2.05 \cdot 10^{-3} \). Average validation error before training was \( 5.37 \cdot 10^{-3} \) for the chosen configuration when running it four times. The validation and training error as well as the training loss for the best execution of the chosen configuration is plotted against the number of epochs in figure 4.4. These values can be compared to the distribution plots of the ground truth weight values in figure 4.3.
Figure 4.4: Plot of the MSE and loss function values for the chosen configuration during training.

Table 4.2: The parameter settings of the layers, from the input layer at the top to the output layer at the bottom.

<table>
<thead>
<tr>
<th>Size</th>
<th>Explicit size</th>
<th>Dropout rate</th>
<th>Transfer function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_x \cdot n_f$</td>
<td>675</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$0.5 \cdot K_x \cdot n_f$</td>
<td>338</td>
<td>0.6</td>
<td>Leaky ReLU ($\alpha = 0.1$)</td>
</tr>
<tr>
<td>$1.0 \cdot K_x \cdot n_b$</td>
<td>270</td>
<td>0.1</td>
<td>Leaky ReLU ($\alpha = 0.1$)</td>
</tr>
<tr>
<td>$K_x \cdot n_b$</td>
<td>270</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
4.5 Platform

The front-end for the final product and the primary evaluation was made in JavaScript and HTML/CSS in order to prepare the product for its intended use (section 1.3, p. 4). This made it convenient to use the machine learning framework TensorFlow, which is a popular open-source solution, since it has a JavaScript API.\footnote{https://www.tensorflow.org/ (visited on 2018-03-07)} The JavaScript API of TensorFlow was not needed for this evaluation, but was of use for the final product. Instead, just the Python version was used for the academic results of this project. Furthermore, the GPU version of TensorFlow was used to speed up computations. The JavaScript application then used BabylonJS to drive WebGL for the real-time graphics. This application was the foundation of the final product, but because of the limitations, elaborated upon in section 4.6, which incurred the need for heavy post-processing, a third subjective evaluation was conducted which allowed using the blend shape weight sequences directly in Unity 2018.2.8f1\footnote{https://unity3d.com/ (visited on 2018-03-07)} instead.

The project used two physical platforms for training:

- A desktop running Windows 7 x64 with an EVGA GeForce GTX 1080 FTW with 8 GB GDDR5X memory, Intel Core i7 2600K at 3.4 Ghz and 24 GB of DDR3 RAM at 1600 MHz.

- A server running Ubuntu 16.04.4 LTS with the Linux 4.15.0-1017-gcp kernel and an Nvidia Tesla K80 with 12.6 GB of GDDR5 memory, Intel Xeon at 2.2 GHz and 6 GB of RAM.

These platforms were roughly equivalent for this purpose, and both needed around 1.3 minutes to train the model outlined above for one epoch of 409 training samples. The networks usually reached the minimum validation error at 10-20 epochs and, using the early stopping heuristic described on page 53, models would typically run for around 25 epochs and take around 30 minutes to train on either platform.
4.6 Presentation

The ANN model output a sequence of weights which was a linear combination of a subset (p. 46) of the blend shapes defined in the ARKit API (p. 43). The final step was to present the sequence as an animation of a 3D model resembling a human, and is outlined in this section. This section begins with describing the implementation of the JavaScript application, and later the Unity approach used for the third survey of the subjective evaluation.

4.6.1 Complexity of the 3D model

The human was modeled, rigged and textured by the author, and its topology prior to triangulation is presented in figure 4.5. It consisted of 18,372 vertices, plus 9,932 vertices for the lashes and 1,510 for the left eye which was then instantiated and mirrored to minimize the memory load. The file containing the 3D model data was at 4.4 MB. The textures were adaptable in resolution, but having them at a fairly low size but still in high quality for their purpose required 15.5 MB, the normal map responsible for the bulk (13.5 MB) since it was still in the lossless PNG format. (The normal map was further compressed later for the final product.)

4.6.2 Rigging

The model had 10 facial bones (see p. 20) capable of moving its lips, tongue and jaw, as well as other bones used for rotating the head and manipulating limbs. The eyes were set to rotate to follow the camera. As for the face, seven bones were used for the lips, two for the tongue (one back and one front) and one for the jaw’s rotation. The rig is presented in figure 4.6.

4.6.3 Blend shape implementation

In the JavaScript application, the ARKit blend shapes were defined in the space of the facial bones, rather than as actual blend shapes or morph targets. Actual blend shapes are displacements of the individual vertices. In this case, the blend shapes were defined as various configurations of the facial bones, controlling the vertices. This means
that while each individual blend shape is achievable in this implementation, the linear combination of blend shapes might not be exactly as expected in an implementation with pure blend shapes. This was a necessary workaround, however, since BabylonJS limits the number of blend shapes to 8, and this project needed 18 in the end.

When using the blend shape weight sequences to create the animations, the sequences were slightly modified first. These modifications are described in the next section. Then, the bones were set to the linear combinations of the blend shapes defined by these modified sequences.

4.6.4 Baseline model

The baseline model was implemented for the first two surveys to get an additional measurement to relate to. It overlayed consonant

\textsuperscript{https://doc.babylonjs.com/how_to/how_to_use_morphtargets#limitations}\textsuperscript{8}
visemes on top of the transitions between the vowels, inspired by MacNeilage et al. (1969) and Öhman (1967) (described in sections 2.1.1-2.1.2).

It constructed segments from the phoneme sequence, each segment consisting of a vowel or silence and all subsequent phonemes until, but not including, the next vowel or silence. The first frame of the segment was considered the apex, and the apex was assigned a viseme corresponding to the segment’s first phoneme, i.e. its vowel. A viseme was thus constructed for each vowel. The visemes were defined by trial and error such that the MSE was approximately minimized to under 0.01 on average, as seen on page 88. Then, an ease-in/ease-out cubic Bezier curve, like the one in figure 4.7, was used to interpolate between the segment apices. To avoid a slow fade-in to the first audible phoneme after the silence in the beginning, another silent phoneme was inserted five frames before the first audible phoneme to make the fade-in shorter.

After this, the sequence was overlayed with the consonant visemes and upper lip adjustment just like for the ANN and ground truth (see p. 61).

Starting by constructing the vowel sequence like this, and then overlaying it with the consonant visemes, led to a much more natural mouth shape animation that mimicked coarticulation better than simply interpolating between visemes of all phonemes.
4.6.5 Limitations and manual fixes

Some issues needed to be manually adjusted, in both the ground truth data as well as the predictions. It could be because of limitations in the Iphone X, or because of the bone rig and the blend shapes not working perfectly well with it, even after reducing the data set (p. 44).

First of all, the blend shape weight sequence was modified for both the ground truth and the ANN model (this does not apply for the baseline model). The jawOpen blend shape was shifted, the sequence was "zero-centered" to achieve a neutral pose in the start and ending, and a maximum threshold was enforced on certain blend shapes. These changes, with the parameter settings that are defined below, were applied to both the ground truth and the predicted sequences to the same extent for the subjective evaluation. The ground truth sequences’ values for the blend shapes that define global sidewards movement, i.e. jawLeft, jawRight, mouthLeft and mouthRight, were set to zero, following the same assumption of symmetric speech as described in section 4.2.2 (p. 46) that was made during training. This is because this kind of global asymmetric movement was not part of the ANN model, and would make it a fairer comparison since subjects in the subjective evaluation thus would not be able to tell which model is the ground truth by any characteristics such as sideways jaw movement.

Furthermore, and the procedures addressing the following issues were used by the baseline model as well, it was also necessary to further strengthen certain consonants to ensure lip closure and other consonant-specific requirements. Also, the tongue was not part of the data gathered by the Iphone X and thus needed to be animated based on the phoneme sequence. Another issue was that the upper lip seemed to cover the upper teeth too much. It was fixed separately by overlaying upper lip movements on top of the sequence, in both the ground truth and the ANN and baseline models.

The implementations of these solutions are described below.

Sequence modifications

The range of values of the jawOpen blend shape was transformed to a lower range, to ensure lip closure. The values of the sequence were shifted like so:

\[ w_t = w_t - \frac{w_{\max} - w_t}{w_{\max} - w_{\min}} \cdot a \]
where $w_t$, $w_{\text{max}}$, and $w_{\text{min}}$ are the \textit{jawOpen} weight sequence’s value at time $t$ and maximum and minimum values, respectively, while $a = 0.07$ is a constant denoting the maximum subtraction value. It shifted the range so that the maximum values remain while the smallest are decreased by $a$ at most.

Then, the sequence was “zero-centered” to achieve a neutral pose in the start and ending. This was done in the same way as when modifying the sequences before they entered the network in the learning procedure (see p. 48).

Finally, a maximum threshold was enforced on certain blend shapes which sometimes had values that would create extreme expressions in the bone rig. The blend shapes that were concerned were \textit{jawOpen} and \textit{mouthPucker}, and this happened mainly in sentence 1 in the test set (table 4.4). These extreme values only occurred in the ground truth sequences, but the thresholds were enforced on both the ground truth sequences and the predictions. This was done by limiting the values to a new maximum ceiling $x_{\text{max}}$, and then pushing down all values between a threshold $x_{\text{thr}}$ and $x_{\text{max}}$ to a new linear curve to make it smoother:

$$w_t' = \begin{cases} x_{\text{thr}} + \frac{x_{\text{max}} - x_{\text{thr}}}{w_{\text{max}} - x_{\text{thr}}} \cdot (w_t - x_{\text{thr}}), & \text{if } x_{\text{thr}} < w_t \\ w_t, & \text{otherwise} \end{cases}$$

where $w_t$ is the sequence’s weight value for the blend shape at time $t$, and $x_{\text{max}} = a \cdot w^*_{\text{max}}$, where $a \in [0, 1]$ is a real-valued constant and $w^*_{\text{max}}$ is the blend shape’s globally maximum ground truth value across all cases. For the two blend shapes \textit{jawOpen} and \textit{mouthPucker}, $a$ was selected to be 0.2 and 0.45. $x_{\text{thr}} = b \cdot x_{\text{max}}$, where $b \in [0, 1]$ is a real-valued constant like $a$, and chosen to be 0.8 and 0.9 for the two blend shapes. These values were chosen while inspecting their effects on the ground truths of the ten test sentences, and led to more natural looking ground truths for the few sentences that had extreme values, while not noticeably influencing those that did not.

\textbf{Consonant requirements and tongue animation}

To implement the consonant requirements and the animation of the tongue, the phoneme sequence had to be provided as well as the blend shape weight sequence when animating the model in the JavaScript application. For all consonants, a viseme was defined, which also de-
defined the tongue’s position, so that it looked like the character was pronouncing the consonant when the bones were configured according to only the consonant viseme. Then, for each consonant in the phoneme sequence, the consonant’s viseme was blended into the animation according to cubic Bézier curves, one for fading in and one for fading out, meeting at the full strength of 1.0. They all had the end points \((0, 0)\) and \((a, 1)\) and the control points \((0.5a, 0)\) and \((0.5a, 1)\) for easing in and out (hence it commonly being referred to as an \textit{ease-in/ease-out} curve), where \(a\) was the length of the curve. This can be formulated like so:

\[
\begin{align*}
    x &= \frac{a}{2} \cdot 3 \cdot (1 - t)^2 \cdot t + \frac{a}{2} \cdot 3 \cdot (1 - t) \cdot t^2 + a \cdot t^3 \\
    y &= 3 \cdot (1 - t) \cdot t^2 + t^3
\end{align*}
\]

where \(0 \leq t \leq 1\).

For all fade-in curves, \(a = 1.5l\) where \(l\) was the length of the phoneme in frames, and each fade-in curve started before its phoneme so that its maximum was situated in the beginning of the phoneme. Each viseme had a manually configured fade-out parameter controlling the length \(a\) of its fade-out Bézier curve. Typically, and by default, they were also set to \(1.5l\), but modified for some consonants such as /t/. These parameters were set using trial and error until it looked realistic. A fade-in curve is shown in figure 4.7. The fade-out curves were identical but inverted.

**Upper lip adjustments**

As for the upper lip adjustments, upwards movement of the upper lip was added to the sequence by partitioning it into segments consisting of contiguous sounds. For all sentences, this would be the entire sequence except for the starting and ending silence. The segment was assigned an upwards movement of the upper lip with a fade-in curve reaching its maximum strength after five frames, and then a long fade out until the end of the segment. For each new vowel, the strength was renewed, with the new vowel’s curve taking over when its value got stronger than the old curve. The Bézier curves used were not quite symmetric like the one in figure 4.7, but had control points at \((0.6a, 0)\) and \((0.75a, 1)\) instead.
The post-processing of the Unity approach

The third survey’s post-processing involved, before a zero-centering like for the first surveys and the training process (see p. 48), a similar transformation as the one for the jawOpen blend shape described in section 4.6.5 (p. 60), but for the mouthShrugLower and mouthShrugUpper blend shapes and with \( a \) set to 0.4 and 0.6, respectively, defined like this:

\[
w_t = w_t^\ast + \frac{w_t - w_{min}}{w_{max} - w_{min}} \cdot a
\]

This increased the larger values for these blend shapes, achieving a stronger pronunciation for most importantly bilabial and labiodental phonemes, e.g. /b/ and /f/. However, since the post-processing had to be the same for all models just like in the other surveys, tuning it to make it look satisfactory on the ground truth and across all phonemes (no consonant- or phoneme-specific alterations were made for the Unity approach) still did not always enforce lip closure during, most noticeably, bilabials in the ANN model’s animations.
4.7 Evaluation

The subjective evaluation consisted of three surveys which investigated whether people could distinguish the ANN model’s performance from the ground truth and a baseline model.

Also, the MSE against the ground truth as well as the trajectories of the models were used in an objective evaluation.

4.7.1 Subjective evaluation

As for the subjective evaluation, three surveys were conducted, and dynamic websites were developed for this purpose. An option was to evaluate user comprehension, as in Beskow (2004), but the results of that study favored the more unnatural looking rule-based model since it articulated more. The purpose of the algorithm in this paper was to produce natural-looking speech, however, so simple user preference blind tests were conducted.

The first two of them used videos captured from the implementation of the JavaScript application, which required much more post-processing because of the limitations of the graphics engine (see section 4.6, p. 57). The first survey used a rating scale, while the last two were stricter preference tests. The first two surveys also included a baseline model. The baseline model was a simple rule-based model, and is further described above (p. 58). As for the ground truth data, the model was simply animated by the motion capture data. For the first two surveys, the ground truth and the ANN animations (as well as the baseline) were improved by certain modifications in a post-processing step (see section 4.6.5, p. 60) since these two surveys used videos taken from the animations created in the JavaScript application and thus had the common issue of retargeting to the bones rig.

The third survey was conducted to investigate the differences when using minimal post-processing, by sending the blend shape sequences directly into Unity and a pre-defined character model after only minor post-processing, where all ARKit blend shapes could be defined (p. 63). The ANN model had, even after this minor post-processing, problems with lip closure during, most noticeably, bilabials.
Sample selection

The surveys went out to Facebook friends, mainly. This could cause concern for a response bias, mainly in the form of wanting to please the author. For this reason, the analysis relies more on the differences between the models rather than the absolute rating categories. Also, this issue motivated the introductory text formulation to not say anything about which models were being compared. If it had revealed that the models being compared were the ground truth, the ANN and, in the case of the first two surveys, a baseline model, there would have been a risk of the subjects trying to find out which was the ANN model and favoring it, would there have been a case of pleasing the researcher.

The subjects were invited to the surveys one by one, and the subjects of the last two surveys also participated in the first survey, apart from one subject. The invitations were made both through a Facebook post (for the first survey only) as well as via Facebook messages and in real life.

The surveys

First, a maximum of one response for each subject was ensured by integrating Facebook authentication into the websites. Upon successfully logging in, the Facebook API replied with the user’s name and id. The user id then went through a keyed hash algorithm (SHA-256) to ensure that it was kept anonymous and confidential. The hashed user id was then signed, and sent along with the digital signature from the back-end to the client as a cookie. Subjects were allowed another way of access as well by the author generating an individual password for them, in case Facebook authentication was considered intrusive, but that was never necessary. Upon survey completion, the back-end checked whether the signature in the cookie was valid, as both a proof of the user having logged in and as a way to ensure the user identity stored in it was untampered. The cookie was then removed from the client, logging the user out. The results were stored in a MySQL database, indexed by the hashed user id, and with the timestamp of survey completion along with them. If a user would have completed the survey a second time, the new results would overwrite the old ones. The subjects were informed of the data stored about them in a privacy policy document.

After having read the introductory text on the first page seen in fig-
Figure 4.8 (the corresponding text for the second and third surveys was only a slightly modified version to update the estimated time and the rating instructions) and logging in via Facebook, the user was redirected to the second page which featured three introductory questions. The questions are presented along with the answer options (presented as radio buttons) in Table 4.3. A screenshot can be seen in figure 4.9. These questions were asked because they could influence a subject’s proficiency in rating Swedish visual speech. Age might predict a subject’s previous exposure to animated speech through media such as video games, and knowing Swedish pronunciation could make the rating task easier for the subject.

Table 4.3: The introductory questions used in the subjective evaluation.

<table>
<thead>
<tr>
<th>Question</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you have any experience in mouth/speech animation or visual speech technology?</td>
<td>None, Small, Moderate, Expert</td>
</tr>
<tr>
<td>What is your age?</td>
<td>{0-9, 10-19, ..., 80-89, 90+}</td>
</tr>
<tr>
<td>What is your proficiency in Swedish?</td>
<td>None, Small, Moderate, Good, Native speaker</td>
</tr>
</tbody>
</table>

Subjects were then, for each of the ten sentences listed in Table 4.4, presented with three synchronized pre-rendered animation clips from the three different models side by side at a time, looping for as long as the subject desired. In the second and third surveys, there were two clips instead, where in the second survey they showed two out of three models (three pair-wise comparisons for each sentence), and in the third survey just the ground truth and the ANN model (one comparison for each sentence). The subjects had a pause/play button as well as a volume control. Each video had a name assigned to it, displayed above each video: A, B and so on, from left to right. The layout of the rating pages can be seen in the screenshot in figure 4.11 for the first survey, and in figure 4.10 for the third survey. The layout of the second survey was like the third, but used the videos of the first.
Survey: A data-driven approach to Swedish visual speech

Thank you for helping me out by providing data for my Master thesis! There are 13 comparisons (10 sentences since there are repeated) to be made, and it shouldn’t take more than 15 minutes.

Facebook authentication is used below to ensure a minimum of one response per subject. If you want another way to authenticate yourself, please contact me. If you take the survey more than once, the new answers will overwrite the old one. Cookies are used to keep track of your login status, but your answers will be stored anonymously by hashing (keyed) your Facebook user id. By continuing, you agree to the use of cookies and the privacy policy.

The thesis is about using machine learning to animate a character’s mouth given the sentence’s phonemes (sounds). In this case, iPhone X data is being used to train a neural network, that outputs parameters controlling a rig implemented in Javascript.

To evaluate the performance of the three different models compared in this survey, you are asked to rate how realistic the animations look. This is subjective, of course, but keep in mind how the mouth coarticulates, i.e. moves as little as possible in real speech. Simulating/modeling this economy of motion is the main challenge in speech animation.

You will see three synchronized animations at a time, side by side, each from one of the three models, and hear the corresponding audio. Above each video you’ll also see the score you’ve assigned to it to make things easier. The order of the models and sentences is random, so a model might be on the left (and be called "A") in one question (sentence) and on the right (and be called "C") in the next. Then, rate how realistic they look on a scale from 1 to 9. If you think two or more animations deserve an equal rating, you will be asked to specify the codes of these, if you have any preference.

Please note that it is only the animations of the mouth that are to be rated, not the graphics of the 3D model itself.

The required data usage (mainly from the videos) is only about 9 MB. The videos are 320x320 (shrink according to screen size) and in 60 fps.

A large screen is recommended, but if you’re on a smartphone, try landscape mode. If you have any synchronization or playback speed issues even after looping a video several times, try disabling hardware acceleration in your browser. Here’s a couple of articles: Chrome, Firefox. If that doesn’t work either, please try again from a device with higher processing performance (some older smartphone devices might not be able to quite cope with the multi-canvas video rendering).

If you discover any bugs or have any questions, or just feedback, please let me know on Facebook or at ihagot [at] kth [dot] se. Thank you!

Figure 4.8: The intro page of the forms.
Survey: A data-driven approach to Swedish visual speech

First, please let me ask a few questions about you.

Do you have any experience in mouth/speech animation or visual speech technology?

- None
- Small
- Moderate
- Expert

What is your age?

- 0-9
- 10-19
- 20-29
- 30-39
- 40-49
- 50-59
- 60-69
- 70-79
- 80-89
- 90+

What is your proficiency in Swedish?

- None
- Small
- Moderate
- Good
- Native speaker

Figure 4.9: The second page of the forms.

Survey: A data-driven approach to Swedish visual speech

2/15. Just der fer han det do oj jak i fi friend.

Please indicate which animation you prefer:

A

B

A large pause is recommended, but if you’re on a smartphone, try changing order. If you have any questions or playback speed issues, write down your video session notes, or try changing the audio or visual feedback. Here’s a sample of video. Online, Facebook

If you encounter any bugs or have any questions, please let me know on Facebook or at [example@example.com].

Figure 4.10: The rating page of the form in the third survey.
Survey: A data-driven approach to Swedish visual speech

Please rate the realism of the animations (not the 3D model itself) on a scale from 1 to 9, where 9 is "Very realistic", 5 is "Neutral" and 1 is "Very unrealistic".

<table>
<thead>
<tr>
<th>Very unrealistic</th>
<th>Unrealistic</th>
<th>Neutral</th>
<th>Realistic</th>
<th>Very realistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Since 3 are tied (same score), please still list your order of preference among them here, from left to right. If you have see, you can order them in any order. A "<" means that you think the model(s) to the left of it is worse than the model(s) to the right of it. You can also swap the order by clicking the "<->" button. The caption below acts as confirmation of your intended choice.

So, out of the ones you ranked as equivalent, you think C and B are equivalent and the best, while A is the worst.

A large screen is recommended, but if you’ve on a smartphone, try landscape mode. If you have any synchronization or playback speed issues even after looping a video several times, try disabling hardware acceleration in your browser. Here’s a couple of articles: Chrome, Firefox.

If you discover any bugs or have any questions, please let me know on Facebook or at phager[@]kth[.]edu.

Figure 4.11: The rating page of the form in the first survey.
One video was recorded for each sentence and model. In the first two surveys, they were recorded in the JavaScript application at 1920 × 974 pixels using the AVC/AAC codecs in an MP4 container with 1,227 kb/s video at 60 frames per second (the animation featured 30 frames per second, but was interpolated by BabylonJS into 60 frames per second) and 44.1 kHz stereo audio at 132 kb/s. In the third survey, the codecs and audio settings were the same but with the resolution at 826 × 446 and the video bit and frame rates at 163 kb/s and 30 frames per second.

To ensure synchronization, instead of having separate videos and enforcing synchronization between them, it was found more effective to render one composite video and rendering different areas of it on different HTML canvases.

The composite videos, one for each sentence and each consisting of all models side by side, were for the first two surveys rendered at 960 × 320 pixels using AVC/AAC codecs in an MP4 container with 768 kb/s video at 60 frames per second and 48 kHz stereo audio at 128 kb/s. For the third survey, the same codecs, bit rates and audio settings were used, but with the resolution set to 640 × 320 pixels, and with the frame rate at 30 frames per second.

The composite video, hidden from display, was then displayed on three (or two, in the second and third surveys) 320 × 320 HTML5 canvases side by side. Displaying each model in a random position each time was made possible by cropping the original video in each canvas, deciding at random which 320 × 320 portion (starting at horizontal position 0, 320 or, in the first survey, 640) of the original video to display in each of the canvases. The audio track was from the baseline model’s rendering in the first two surveys and from the ground truth rendering in the third survey, but this is of no importance since the audio tracks were strictly synchronized and identical.

The question formulation in the first survey, displayed between the videos and the radio button scales, was:

*Please rate the realism of the animations (not the 3D model itself) on a scale from 0 to 10, where 10 is "Very realistic", 5 is "Neutral" and 0 is "Very unrealistic".*

In the second and third surveys, it was instead simply:
Please indicate which animation you prefer:

All video clips were rendered in the same engine and context for the respective surveys. The order of the sentences was randomized, as well as which model that appeared on which canvas. This method of evaluation has been frequently used in similar works (Taylor et al. 2012; Taylor et al. 2017; Karras et al. 2017), particularly the simpler preference tests of the second and third surveys where the subjects were only asked to indicate which they preferred by clicking on the left or right buttons, but with the addition of allowing ties by clicking on the middle button.

However, in the first survey, the subjects were asked to rate the realism of each video on a scale from 1 to 9, rather than simply indicating which was preferred. 1, 3, 5, 7 and 9 corresponded to Very unrealistic, Unrealistic, Neutral, Realistic and Very realistic, respectively. The scale was used because it allowed for additional measurement of the distance between the models apart from just how many subjects preferred each model. The scale also allowed to rate models equally to avoid forcing an answer which could risk being random rather than actual preference. The three scales, one for each video, were placed as rows of radio buttons in a table with the corresponding rating at the top.

If a subject rated any models equally in the first survey, a simple GUI, seen in figure 4.11, dynamically appeared to ask the subject to indicate their order of preference, if they had any. This was implemented to avoid forcing the subjects to enter ratings above or below the intended rating in an effort to indicate an ordering. This GUI had an instructional text above it as well as a dynamic text summarizing the subject’s selection, as a feedback for clarification. The text above it was:

Since 2 are tied (same score), please still list your order of preference among them here, from left to right, if you have any. If you don’t have any preference between two models, ties are allowed by selecting “=” instead of “<” between them. A “<” means that you think the model(s) to the left of it is worse than the model(s) to the right of it. You can also swap the order by clicking the “<->” button. The caption below acts as confirmation of your intended choice.
The text below it was updated in real time depending on what the subject entered in the GUI. It could be for example: *So, out of the ones you ranked as equivalent, you still have no preference among them.*

Another example is: *So, out of the ones you ranked as equivalent, you think C is the best, while A and B are equivalent and the worst.*

In all three surveys, after randomizing the sentence order, the first three sentences (comparisons in the second survey) were copied to the back, and then these first three judgments were ignored. This was to allow the subjects to get familiar with the task and avoid training effects during the study, a principle also used in e.g. Taylor et al. (2016) (five omitted, leaving on average 26 judgments on each out of 20 sentences), and Alexanderson et al. (2014). For the third of these first three sentences, the ground truth was displayed in the ANN model’s randomly assigned canvas as well (correspondingly, in the last two surveys, the two videos of this third comparison were the same), as a test of the subject’s credibility and ability to judge correctly. If they got very different scores (or, in the second and third surveys, if it was not a tie), it would be a hint that the subject’s choices were not to be trusted. Furthermore, the time taken for each question was measured and part of the collected data, since it might be another indicator of the subject’s thoroughness as conspicuously short time measurements could indicate sloppiness.

The number of questions, excluding the questions about personal details, was thus 13 for the first and third surveys, and 33 for the second survey (three pair-wise comparisons for each sentence).

The ten sentences from the test set are presented in table 4.4.

Compilations of the final videos used for the first two surveys\(^7\) and the third survey\(^8\) have been uploaded for viewing.

### Statistical analysis

Criticism of using ordinal scales and assuming them to be of the interval nature has been raised, and such assumptions are a common problem (Jamieson 2004). On the other hand, it has been noted that parametric analysis methods are robust to uses of ordinal scales (Norman 2010), and even regression methods which also assume interval

\(^7\)https://drive.google.com/open?id=1boI0xwhSjd7e957HpDE5OeVvrTlfsdTS (visited on 2018-11-19)

\(^8\)https://drive.google.com/open?id=14VStE84DTPjX5y5SwTqq8uno bM-Q2nfr (visited on 2018-11-19)
Table 4.4: The sentences used in the subjective evaluation.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Det är en av de märkligaste motsättningarna hos honom.</td>
</tr>
<tr>
<td>1</td>
<td>Informationsteknik är man i stort behov av, menar han.</td>
</tr>
<tr>
<td>2</td>
<td>Staten har satt löner långt under gällande villkor.</td>
</tr>
<tr>
<td>3</td>
<td>En utmärkt men livsfarlig farkost i det iskalla vattnet.</td>
</tr>
<tr>
<td>4</td>
<td>Skulle det då ha varit annorlunda?</td>
</tr>
<tr>
<td>5</td>
<td>Ty om någon enda bestående sanning återstår, är barn liv.</td>
</tr>
<tr>
<td>6</td>
<td>Just därför kan det ju dra ut på tiden i flera år.</td>
</tr>
<tr>
<td>7</td>
<td>I början hade det mest varit hemmafruar.</td>
</tr>
<tr>
<td>8</td>
<td>Medan de åt och drack, blev de snart torra och varma igen.</td>
</tr>
<tr>
<td>9</td>
<td>Kantigheten återkommer i målningarnas annars helt motsatta uttryck.</td>
</tr>
</tbody>
</table>

data. In this study, however, the dependent variable, which will be referred to as $y$ in this section, was treated as an ordinal variable.

Ordinal logistic regression was used for the analysis, by implementing a proportional-odds cumulative logit mixed effects model in R 3.5.1 (R Core Team 2018) using the libraries ordinal (Christensen 2018a), car (Fox et al. 2011), psych (Revelle 2018), RVAideMemoire (Hervé 2018) and emmeans (Lenth 2018a). The analysis, following Mangiafico (2016), Lenth (2018c) and Lenth (2018b), investigated if any statistically significant differences between the animation models could be found. In the second and third surveys, a win was encoded as scoring two points, a loss as zero points, and a tie as one point for each model. The regression model assumes an ordinal dependent variable, a categorical independent variable (a dummy encoding representing the animation model) of at least two levels, and repeated measures across subjects and sentences. Because the independence assumption of the observations would fail if the ratings (per user and per sentence) were considered independent of each other, the mixed effects model accounted for this non-independence by using intercepts and slopes for each subject and sentence, which is necessary to avoid Type I errors (Barr et al. 2013). The logit (log-odds) score for model $i$, subject $j$ and sentence $k$ was modeled like this:
logit \left( p(y_{i,j,k} \leq g) \right) = \ln \left( \frac{p(y_{i,j,k} \leq g)}{p(y_{i,j,k} > g)} \right) = \\
\beta_0 + U_{0,j} + S_{0,k} - (\beta_1 + U_{1,j} + S_{1,k})X_i + \epsilon_{i,j,k}

where \( \beta, U \) and \( S \) denote "universal", user and sentence intercept and slope variables, a subscripted 0 denotes intercept, a subscripted 1 denotes slope, \( \epsilon \) is the error term, and \( g \in \{0, 1, 2, \ldots, 8\} \) (there is one \( \beta_0 \) intercept for each of the 9 levels of \( y \) except the last one). For example, \( U_{0,j} \) is the intercept for user \( j \) while \( U_{1,j} \) is the slope. \( X_i \) was turned into an unordered factor in R and was thus essentially dummy encoded. Since this is a cumulative logit model, it predicts the cumulative log odds of getting less than a certain score given a certain model. For each score, it uses different \( \beta_0 \) intercepts depending on the level of the dependent variable. The model will, after it has been fitted to the data, allow calculations of the probabilities of getting different scores given different animation models. It was formulated in lmer-like notation like this:

\[ y \sim x + (1+x|u) + (1+x|s) \]

The dependent score variable \( y \) and the independent model variable \( x \) in the data \( d \) were converted to ordered and non-ordered factors. Then, the model was fitted to the data. After the model was fitted, the Anova function provided a type II analysis of deviance, which is preferred according to Langsrud (2003). The function used a Wald chi-square test to generate a p-value denoting the probability of getting an estimate for the coefficient of \( x \) at least as extreme as the one calculated, given the data and that the null hypothesis is true. The null hypothesis in this case would be that the coefficient for \( x \) (the animation model), i.e. \( \beta_1 \), is zero, and thus that the independent variable \( x \) is not a predictor of the dependent variable \( y \) - in other words, that the animation models are equivalent. Then, once the analysis of deviance had found a significant impact of the independent variable, a pairwise Tukey post-hoc test was used to analyze the differences between the individual models and get the p-values of the comparisons as well as the data for plotting the mean scores and their inferred probability distributions with 95% confidence intervals.
When plotting the inferred means and probability distributions for each individual sentence, the same commands were run but with a model stripped of the sentence random effect and with just a random intercept for each subject (because, for each sentence, there was only one observation for each model and subject), run on the results of one sentence at a time.

It was also necessary to verify that there was no violation of the regression model’s assumption of proportional odds. The proportional odds assumption is implicit in this regression model and means that for all levels of the dependent variable, the ratio of the odds ratio $OR$ of two levels of the independent variable are the same. In other words, with the dependent variable $y$ and the first two levels of the independent variable $x$:

$$\frac{OR(y \leq g_0 | x = 0)}{OR(y \leq g_0 | x = 1)} = \frac{OR(y \leq g_1 | x = 0)}{OR(y \leq g_1 | x = 1)},$$

$\forall g_0, g_1 \in \{0, 1, 2, \ldots, 8\}$

This was verified in R using the nominal_test and scale_test functions and making sure they did not return a significant p-value for the independent variable (Mangiafico 2016; Christensen 2018b).

### 4.7.2 Objective evaluation

In the objective evaluation, the MSE of the predicted bone displacements against the ground truth data was calculated in a comparison against the baseline model, in the global space of the JavaScript application. The MSE was calculated across all of the lip and jaw bones, since they were the ones that differed between the models. Also, the blend shape weight MSE of the prediction against the ground truth, before any post-processing, was calculated. MSE has also been used in other related studies (Taylor et al. 2016; Taylor et al. 2017) to measure the deviance from the ground truth or other approaches.

Furthermore, the y-coordinate of the jaw bone in the JavaScript application when pronouncing sentence 3 was plotted against time to gain some insight into the differences in temporal dynamics.
Chapter 5

Results

This chapter presents the results of the evaluation methods described in Chapter 4. It only features the representations of the results necessary for the analysis. For more representations such as number-of-wins tables (like those found in related works such as Karras et al. (2017) and Taylor et al. (2017)), as well as the "raw" means data and more, see Appendix F.

5.1 Subjective evaluation

5.1.1 Sample demographics

As described in section 4.7.1 (p. 65), the subjects were found on Facebook. The subjects of the second and third surveys are subsets of the subjects of the first survey except for one subject (who identified itself as having "None" in animation experience, being aged 50-59, and having a "Moderate" Swedish proficiency) who was not in the first survey. The demographics of the subjects of each survey is summarized by the distributions of the answers to the first questions for each survey, presented in figures 5.1-5.3. The total number of subjects for the first, second and third surveys were 15, 9 and 10, respectively.
Figure 5.1: Distribution of age among the subjects for each survey. Empty age groups are omitted.

Figure 5.2: Distribution of the experience in mouth/speech animation or visual speech technology for each survey.
Figure 5.3: Distribution of the proficiency in Swedish for each survey.
5.1.2 Results from the first survey

The results from the first survey are summarized in this section. For more representations as well as the "raw" means data, see Appendix section F.1.

The preference question that appeared upon ties was never answered. Only 2 out of 15 subjects rated the ANN and ground truth animations equal in the third question, even though they were exactly the same (see p. 72).

The statistical analysis showed that some of the models were, in total, statistically significantly different from each other at the 0.05 level, and the proportional odds assumption tests passed for the independent variable. The post-hoc analysis showed that the differences between the ground truth and baseline models ($p < 0.0001$) as well as between the ANN and baseline models ($p = 0.0037$) were statistically significant at the 0.05 level, while the difference between the ground truth and ANN models was not ($p = 0.1801$).

Figure 5.4 shows the inferred means derived from the regression, for each sentence as well as across the entire set, with their 95% confidence intervals. Statistically significant results are highlighted in it, as well. The post-hoc analysis found significant differences between the ground truth and baseline models in sentences 2, 5 and 7, with p-values at 0.0110, 0.0031 and 0.0314, respectively. In sentence 8, it found significant differences between both the ground truth and baseline models ($p = 0.0026$) as well as between the ground truth and ANN models ($p = 0.0209$).
Figure 5.4: The inferred mean ratings for the three models for all sentences and in total, with 95% confidence intervals. Superscripted symbols with a dash between them indicate a statistically significant difference between their corresponding models for that comparison.
5.1.3 Results from the second survey

The results from the second survey are summarized in this section. For more representations as well as the "raw" means data, see Appendix section F.2.

Instead of ratings, the results were encoded as points, where wins and losses yielded 2 and 0 points respectively, while ties yielded 1 point to each model.

Only 4 out of 9 subjects rated the animations equal in the third question, even though they were exactly the same (see p. 72).

The statistical analysis showed that all models except the ground truth and ANN models were statistically significantly different from each other at the 0.05 level, and the proportional odds assumption tests passed for the independent variable. In total, the post-hoc analysis showed statistically significant differences between the baseline and ANN models \( p = 0.0234 \) and between the baseline and ground truth models \( p = 0.0276 \), but not between the ANN and ground truth models \( p = 0.8835 \).

Figures 5.5-5.7 show the inferred means derived from the regression, with their 95% confidence intervals, for each pair-wise comparison. Statistically significant results are highlighted in them, as well.

Apart from across all sentences, some comparisons within sentences were found to be statistically significant at the 0.05 level. When comparing the ground truth and ANN models, the differences in sentences 5 and 9 were significant with p-values at 0.0271 and 0.0259, respectively. The baseline and ground truth comparison yielded significant differences in sentences 2, 4 and 5 with p-values at 0.0056, 0.0271 and 0.0056, respectively. When comparing the baseline and ANN models, the differences in sentences 0, 2, 4, 6 and 7 were significant with p-values at 0.0109, 0.0056, 0.01528, 0.04076 and 0.04076, respectively.
Figure 5.5: The inferred mean points for the GT and ANN models for all sentences and in total, with 95% confidence intervals. Superscripted symbols with a dash between them indicate a statistically significant difference between their corresponding models for that comparison.
Figure 5.6: The inferred mean points for the GT and baseline models for all sentences and in total, with 95% confidence intervals. Superscripted symbols with a dash between them indicate a statistically significant difference between their corresponding models for that comparison.
Figure 5.7: The inferred mean points for the ANN and baseline models for all sentences and in total, with 95% confidence intervals. Superscripted symbols with a dash between them indicate a statistically significant difference between their corresponding models for that comparison.
5.1.4 Results from the third survey

The results from the third survey are summarized in this section. For more representations as well as the “raw” means data, see Appendix section F.3.

Instead of ratings, the results were encoded as points, where wins and losses yielded 2 and 0 points respectively, while ties yielded 1 point to each model.

Only one subject rated the animations equal in the third question, even though they were exactly the same (see p. 72).

The statistical analysis showed that the models were statistically significantly different from each other at the 0.05 level, and the proportional odds assumption tests passed for the independent variable. The ANOVA showed a p-value at 0.0112.

Figure 5.8 shows the inferred means derived from the regression, with their 95% confidence intervals. As well as for the total mean comparison, the mean comparisons for sentences 0, 1, 2, 6, 7 and 8 showed significant differences at the 0.05 level as well, with ANOVA p-values at 0.005492, 0.005492, 0.01528, 0.0004304, 0.005492, 0.0001245, respectively.
Figure 5.8: The inferred mean points for both models for all sentences and in total, with 95% confidence intervals. Superscripted symbols with a dash between them indicate a statistically significant difference between their corresponding models for that comparison.
5.2 Objective evaluation

Predictions for sentences in the test set took on average 73 milliseconds per sentence on the desktop platform (p. 56).

First, the MSE for the ANN model against the ground truth in blend shape space is presented in figure 5.9. The MSE against the ground truth for both models, calculated across all bones in global space in the JavaScript application, for each sentence and in total, is presented in figure 5.10. To get an understanding of the differences in temporal dynamics, the trajectories of the jaw bone’s y-coordinate when pronouncing sentence 3 are presented in figure 5.11.

![Figure 5.9: The blend shape space MSE of the ANN model against the ground truth, for each sentence and in total.](image-url)
Figure 5.10: The bone-wise global space MSE gotten against the ground truth in the JavaScript application for each model and sentence, as well as across all sentences.
Figure 5.11: The y-coordinate values (global space) of the jaw bone for each model, for sentence 3, relative to the neutral configuration.
Chapter 6

Discussion

This chapter identifies and discusses the findings in the results, in order to answer the problem statement (p. 4) and arrive at the conclusions provided in the final section.

6.1 Limitations and obstacles

The limitations to the results of this project are partially due to the sub-optimal accuracy gotten from the Iphone X camera. Even in the Unity setting of the third survey, where no post-processing was used, some offsets needed to be implemented, and many of the data samples needed to be removed since they were beyond fixing using any universal set of operations. This led to a smaller than expected data set, as well. This could have been mitigated by checking each sample after recording it, re-recording if necessary, but that would have been much more time consuming for the limited time scope of this thesis project.

As mentioned in section 4.6.5 (p. 60), BabylonJS did not support more than eight blend shapes, meaning that the blend shape data had to be retargeted to a bone rig. This was an issue for the JavaScript application, creating the need for post-processing that essentially equalized the models during consonants. The third survey provided a more thorough comparison for consonants without this post-processing.

A problem of the surveys was that subjects might miss the objective by selecting the model that matched the audio the best, or have a bias towards the one that articulated more. The matching of the perceived intensity of audio and video played together is important, as
highlighted by Alexanderson et al. (2014), where the results for the incongruent audio-video pairings show that pairing normally spoken speech audio with whispered speech video was perceived as more "eerie" than pairing the same audio with video of loud speech. An attempt was made to mitigate this effect, by making the models articulate to an equal extent. No corresponding effort was made in the third survey, however.

The number of subjects was relatively small, but there were ten sentences and it was possible to count each rating as an observation because of the chosen method, and thus many of the results were still statistically significant (see section 4.7.1, p. 72). Besides, the discussion here relies on not just one, but three surveys.

6.2 The results

The results of the first survey show that in the context of the JavaScript application with all of its post-processing, the ANN model was indistinguishable from the ground truth at the 0.05 level, but distinguishable from the baseline model. The ratings for the ground truth and ANN models were only significantly different in sentence 8. Apart from that, no significant differences were found sentence-wise apart from between the ground truth and baseline models in sentence 5, likely because the number of data points per sentence were relatively few.

The second survey, which was virtually the same, however done using a pair-wise preference test, yielded significant differences in the total means in the same model comparisons, i.e. only in the comparisons of the baseline against the other two models. The ANN model was very close to the ground truth both in the direct comparisons and the respective superiority over the baseline model. The sentence-wise results for the ground truth and ANN models were only significantly different in sentences 5 and 9, but in as many as five sentences when comparing the ANN and baseline models.

The post-processing of these two first surveys can be considered as a way to control for the impact of articulation, as it made them articulate roughly equally. It also ensures e.g. lip closure and other requirements for certain consonants. This means that the first two surveys mainly investigate the mouth shapes and the temporal dy-
namics of vowels, but also consonants to some degree, since the overlaid consonant visemes only override necessary bones. Since the post-processing was identical for the ground truth and the two procedural models, the differences can be attributed to the motion capture data and the learned function imitating it.

On the other hand, the results of the third survey reveal the difference between the learned model and the ground truth more honestly, since the amount of post-processing is very limited. It is therefore more suited to analyzing the performance of the machine learning itself. The third survey found a significant difference between the ground truth and the ANN model, and even so for six of the individual sentences. Likely, this would have been the case for all sentences, given more subjects. Since no efforts were made to make them articulate equally, or to re-record the audio to make it suit both models equally well, the results in the third survey, which is supposed to answer whether the animation is realistic regardless of the degree of articulation or how well it matches audio, might be biased in favor of the ground truth. This means that while the third survey does show that the difference between the ground truth and the ANN model is noticeable, it did not control for the impact of these aspects, and thus it is not clear to what extent this difference is negative in terms of the perceived realism.

The difference in results regarding the ground truth and ANN comparison between the first two surveys and the third survey is more likely explained by the problems regarding the closure of especially bilabials in the ANN model. The post-processing in the third survey was not consonant- or phoneme-specific and was, as in the other surveys, the same for all models. However, solving this problem using consonant-specific overlays like in the first two surveys should not impair the coarticulation effects to any great extent, as requirements such as lip closure are fairly context-independent.

The objective study, done in the context of the JavaScript application with the post-processing of the first two surveys, most importantly gives an idea of the temporal dynamics that were learned. They are difficult to replicate using conventional rule-based models. The trajectory of the baseline model is useful since it shows how well a rather simple model can attempt to replicate them, but also shows how, in contrast, the ANN model found the subtler details due to coarticulation when encountering specific sequences of phonemes. No-
tably, however, neither captured the aspiration in the final phoneme, /t/, denoted by the ground truth’s descent at approximately frames 165 through 173, at least not in the y-coordinate of the jaw bone plotted in figure 5.11. The trajectory of the ANN model is sometimes more extreme than the ground truth, but can usually, as expected, be explained as a sort of average of it.

6.3 Ethical and societal aspects

There are many ethical issues related to the progress in the machine learning area regarding automatic animation, especially within speech synthesis, as the requirements to be able to forge videos of e.g. politicians are being met (Suwajanakorn et al. 2017). Therefore, it is of great importance that the methods available are known throughout society, so that citizens of any social status are critical of any information they encounter.

On the other hand, and as for the purpose of this product, it could increase the enjoyment of engaging with automatic customer service for any age. This is especially true for those who take the time to use it, which would likely be children and the elderly. It has been shown, and mentioned in section 2.1.3 (p. 16), that multimodal speech, provided by for example an animated chat bot, can increase comprehension. Thus, it could increase the access to information for all citizens and thereby also the participation in society. The fact that the bones rig can be retargeted to any arbitrary 3D model means that the JavaScript application can also be adapted to any target group. An example of this could be creating speaking anthropomorphic creatures for children.

6.4 Conclusions

Using artificial neural networks for visual speech has been proven to be helpful to easily catch the complex temporal dynamics at play, even when the input is simply phoneme sequences. The amount of data and the complexity of the network was not enough to capture consonants fully, but for vowels and the rest of the temporal dynamics, the ANN still outperformed the baseline model and was indistinguishable from the ground truth. However, the third survey revealed that the differ-
ence between the ANN model and the ground truth is still noticeable for consonants. It is not entirely clear to what extent this difference is negative in terms of perceived realism, since the third survey did not even attempt to control for the impact of articulation. The difference is more likely explained by the fact that the ANN model missed some important requirements such as lip closure for bilabials, handled by the overlay of consonant-specific visemes in the JavaScript application. This should not impair the coarticulation effects to any great extent, as requirements such as lip closure are fairly context-independent, but naturally, even small deficiencies in consonants in the model affect the coarticulatory influence from those consonants on the neighboring phonemes.

The Iphone X was found to be less powerful than expected, but for the purpose of the final product it was still satisfactory with the help of consonant viseme overlays and normalization procedures, used in the first two surveys and the final product.

The network complexity was relatively small, compared to related works. This means that the computation times are feasible for real-time processing, at least when the computations are performed on a GPU like the ones in this project. The bones rig can also be retargeted to any arbitrary 3D model.

### 6.5 Future work

It seemed clear that increasing the complexity of the network also increased the need for data, when investigating the effects of input space reductions (p. 48). With the current amount of data, the network complexity seemed optimal. Future work could investigate whether deeper networks could generate more complex dynamics, just like related works used deeper networks to create more abstract representations, but that requires more data. At the same time, the ANN model is still indistinguishable from the ground truth in the context of the final product, though there is room for further improvement.
Bibliography


95


Fan, Y., Hong, Z., et al. (2014). “TTS synthesis with bidirectional LSTM based recurrent neural networks”. In: INTERSPEECH.


Appendix A

The rendering pipeline

In a typical real-time rendering engine, there will first be a geometry processing stage where 3D space is projected into 2D screen space vertex by vertex, using the vertex shaders. The vertex shaders control how this projection is made, and can manipulate properties such as position and color. This is done on a per-primitive basis, and the computations may demand hundreds of floating point operations per vertex. This is then followed by the scan conversion or rasterization phase, where the color values for each pixel are computed. This is done per pixel, while storing computations in a frame buffer. Since DirectX 10, geometry shaders can be inserted after the vertex shaders, and are capable of altering the geometry and tesselating the meshes (Bao et al. 2011a).

It is also necessary to determine which vertices are occluded by other geometry, in a process called visible surface determination (Salomon 2011).

The light computations can vary a lot depending on the processing power available. Direct illumination is easy to calculate for polygonal objects, since the light for a polygon only has to be calculated once, and the light is assumed even for the polygon. Phong and Gouraud models then help to achieve a smooth lighting across the surface by interpolating the normal vectors of the polygons. Rendering objects one by one like this is referred to as scanline rendering (Salomon 2011).

Computers since at least 2010 usually have hardware to implement perspective projection, visible surface determination and direct illumination. However, for shadows and indirect illumination (sometimes referred to as global illumination), it is necessary to incorporate algo-
algorithms such as ray tracing or radiosity. Ray tracing is too complex for real-time applications since it is view dependent and has to be recalculated with every view change. It involves casting a ray for each pixel of the 2D screen into the 3D scene, and following each ray for a certain number of reflections - in some cases all the way back to their light sources, depending on the reflectivity of the materials. In the end, a color for the pixel is deduced from this process. It is easy to see that it is a computationally cumbersome process. Even more advanced methods, such as photon mapping, are also available. Radiosity, however, stores radiosity data for the surfaces and can thus compute the light intensity from there, making it feasible for real-time applications (Salomon 2011).

Shadow mapping, the computation of where simple shadows fall, is often a separate module. It does not necessarily require the more complex indirect illumination algorithms, but is in effect a visibility determination problem from the lights’ points of view since shadows fall on points that are occluded from the light sources (Bao et al. 2011b). The difference between using shadow mapping and using indirect illumination is that shadow mapping simply renders shadowed areas darker, whereas indirect illumination calculates the precise color resulting from reflections between objects.
Appendix B

Acoustics

The oscillation of a sound can be described as a waveform, where the air pressure is described as a function of time. It can be further described as the combination of several sinus waves (spectral components), where each has its own amplitude and frequency. The component with the lowest frequency is called the fundamental frequency. In speech, it corresponds to the rate of the vocal cord vibrations. The other components have frequencies corresponding to multiples of the fundamental frequency, and are called harmonics. The vocal tract then amplifies some frequencies and suppresses others. Hence, it acts as an acoustic filter. This property of the vocal tract, that parts of it start to oscillate to frequencies near their natural frequencies, is called resonance. The resonant frequencies are called formants. This description of the system is referred to as the source-filter theory (Case et al. 2012).

Various aspects, called the prosody, of how the speech is delivered can affect how it is interpreted. Prosody includes pitch, stress and timing. The relative pitch of the various syllables can often change the meaning and emphasis. Stressed syllables are emphasized by a louder pronunciation, a higher pitch and so on.

The sound can be displayed in a spectrogram, where the frequency energies (on the y-axis) are plotted over time. In this way, it is possible to distinguish the various formants and how they are altered over time as markings on this plot, as well as the fundamental frequency which is reflected in the periodicity (stripes) of the formant markings. In fact, the vowels can be distinguished by the frequency distribution of the first three formants alone (Case et al. 2012). Examples of its usefulness can be seen in Öhman (1966).
Appendix C

Convolutional and recurrent neural networks

C.1 Convolutional neural networks

Convolutional neural networks (CNNs), a category of DNNs, were inspired by the Hubel and Wiesel model (Hubel et al. 1962) of the visual system in cats, who discovered that the signals of more complex cells were found to be downsampled to more abstract representations in order to be processed on a higher level. In the same manner, CNNs have convolutional and pooling layers that downsample the signal, allowing different layers to process different representations of the data. Early examples are the LeNet networks, the first of which was made to recognize numbers and is described in Cun et al. (1990). AlexNet, due to Krizhevsky et al. (2012), outperformed the previous state of the art in image classification, using five convolutional layers with max-pooling in between some of them, followed by three fully connected layers and a 1,000-way softmax, in order to be able to output 1,000 class labels. CNNs are now used everywhere in pattern recognition tasks.

The assumption made is that for pictures and many other types of data that can be represented as matrices, elements that are close to each other are more related than those further away - neighboring pixels are correlated. This makes the amount of needed parameters fewer, and for photographs the dimensions can otherwise quickly become extremely large and require an unfeasible amount of weights without this subsampling. Moreover, convolution is powerful when processing data that should be invariant to translation, rotation, scal-
ing and so on, which is typically the case when, for instance, classifying photographs. The filters end up learning important patterns that are encoded in them, and will detect a pattern even if it is translated to another area of the matrix (Bishop 2006).

Convolutional layers convolve the image to transform it to a lower resolution feature map or response map. For instance, a filter of size $5 \times 5 \times 3$ can be moved across an image of size $32 \times 32 \times 3$, and at each position an operation is performed, it outputs an “element” of size $1 \times 1 \times 1$, resulting in a $28 \times 28 \times 1$ size response map from the layer. For each out of $28 \times 28$ positions, the convolution has output a one-dimensional “pixel” in the resulting output. A possible interpretation is that each neuron in the output layer is, by the convolution operation, connected to several neurons in the input layer. It is possible to use several filters to get a response map of greater depth as well.

Upon training, the gradient of the loss with respect to the convolution parameters has to be calculated in order to train them. Instead of having as many weights to be learned as there are features in the input matrix, a convolutional layer will only have as many weights as there are neurons in the filter plus one bias. This filter is applied to all areas of the input matrix, drastically reducing the number of weights to train (Bishop 2006). Thus, in the example layer mentioned above, the number of weights would be $5 \times 5 \times 3 = 75$ plus a bias, rather than $32 \times 32 \times 3 = 3072$. Small filters can be stacked to make the computations more efficient than fewer larger filters, since, for example, each neuron in the last of two stacked $3 \times 3$-filter convolutional layers would in fact have the input of a $5 \times 5$ area in the original input image, and this decreases the number of parameters (Li et al. 2016a). The result is then passed through a nonlinear transfer function as usual. Each filter and the resulting feature map detects the same feature but in different areas of the input, and because of this, many filters are often used in each layer in order to detect many different patterns at each representation level (Bishop 2006). It is possible to apply the filters at different strides as well. If a stride and filter size is not compatible with the dimensions of the input matrix, it is possible to use zero-padding. This means that the input is simply padded with zeroes at the border (Li et al. 2016b).

To subsample to a smaller and more abstract representation for the next layer, max-pooling is often used. It outputs, for contiguous and non-overlapping areas of the feature map, the result of some operation of the neurons, e.g. computing the average or maximum value. It also
multiplies this with a weight and adds a bias. This means that if this max-pooling filter is chosen to be e.g. $2 \times 2$ in size, the output representation size will be a quarter of the input representation (half the number of rows and columns). Convolutional layers are often used in conjunction with max-pooling layers. With each such pair, the representation is more invariant to small translations and other transformations. It is not uncommon to increase the number of response maps while decreasing the size of each response map (Bishop 2006). One example of this can be found in Karras et al. (2017).

Finally, fully connected layers then serve to scale the representation to the desired output dimensions, such as class labels using a softmax operation (Bishop 2006).

C.2 Recurrent neural networks

Recurrent neural networks (RNNs) are artificial neural networks that, when fed a sequence of data elements, use a time delay to incorporate some of the information of the previous time steps in the current calculation.

Upon inference time, they can be used for many different purposes. It is possible to generate an output sequence from several iterations out of just one input element at the first time step. This can be used to generate a text, after having trained the network on literature, since the distribution of words or tokens in language are conditioned on the previous elements. Another use is to predict an output element at the last time step from an input sequence, or to predict an entire output sequence continuously, with one output element for each time step, from an input sequence. Finally, it is possible to encode and decode a sequence for e.g. machine translation purposes, by having the RNN receive an input sequence of length $n$ and then, using all of the context provided, outputting a sequence of length $n$.

A simple RNN can be described by three weight matrices $W$, $U$ and $V$, and two bias vectors $b$ and $c$. First, the activation for time step $t$, $a_t$, is calculated:

$$a_t = Wh_{t-1} + Ux_t + b$$

It is a linear combination of the past hidden state $h_{t-1}$ ($h_0$ is initialized to some given state), the current input $x_t$ and the bias $b$. Then, the next hidden state, which is also later used to calculate the output for
the current time step, is calculated by putting it through a nonlinear transfer function, e.g. a hyperbolic tangent function, like so:

\[ h_t = \tanh(a_t) \]

And the output for the current time step \( t \), \( o_t \), is:

\[ o_t = Vh_t + c \]

\( o_t \) can then be put through e.g. a softmax transformation if required.

As briefly mentioned above, more advanced RNNs can be used for machine translation, such as Cho et al. (2014), where the final hidden state of an English input sequence, i.e. the context \( c \), was first encoded, and then fed into a decoder RNN which generated the translated sequence in French. In the decoder, each hidden state was dependent on \( c \), the previous hidden state and the previous output element.

Another example of usage is to generate image captions by feeding the representation from a CNN into the context of an RNN that generates a sequence of words (Mao et al. 2014).

Because of the requirement of RNN loss functions to take long sequences into account, gradients tend to either explode or vanish. This problem is present in back-propagation through time (BPTT) approaches, where the RNN is unfolded and the gradients are computed over each time step, distributing the error over them. It is thus local in space but not in time, i.e. the computational complexity does not depend on the network size but on the sequence length. Real-time recurrent learning (RTRL) approaches, where the gradients are calculated while going through the sequence, i.e. the network is not unfolded over time (it is instead local in time but not in space), also faces these problems (Hochreiter et al. 1997). An easy solution to exploding gradients is to simply clip the gradient if its magnitude exceeds a certain threshold (Pascanu et al. 2012). Another solution is to use long-term short memory (LSTM) neurons, introduced in Hochreiter et al. (1997), which helps with vanishing gradients by having a constant error flow. This is done by each memory cell having a constant error carousel (CEC) which "traps" the error signal inside the memory cell, and also multiplicative unit gates for the input and output to regulate how much of the input from the network is to affect the unit and how much the output of the unit is to affect the rest of the network at each time step. Upon back-propagation, the error is propagated back through time in the CEC.
It is not further propagated back to other cells once it has entered a cell, but only out to the cell inputs to train their weights. This constant error flow makes very long time delays possible, unmatched by any other previous approaches.
Appendix D

Optimization algorithms

To set the parameters during training, more advanced optimization algorithms than SGD are warranted. This is because of a number of issues that arise as the models become more complex.

In ravines, where the gradient has different magnitudes in different dimensions, it would be necessary to have different learning rates in these different dimensions. Since the gradient descent algorithm steps in the direction of the steepest descent, having the same learning rate for all dimensions means that the model would end up fluctuating across the local minimum in one dimension and approach it in another. With all dimensions considered, it would miss the local minimum. Ravines are naturally formed in networks where weights have different magnitudes, and other techniques have surfaced to deal with this problem (Sutton 1986).

Another issue is that in saddle points, i.e. where the gradient is zero in one dimension but non-zero in others, the model could get stuck when coming down to the saddle point instead of stepping downwards in the other dimensions.

Momentum dampens the oscillations of gradient descent by introducing a momentum term for each parameter. If \( W_t \) is the weight matrix for some layer at epoch \( t \), then we introduce \( v_t \) which has the same dimensions as \( W_t \), and update as such:

\[
\begin{align*}
    v_t &= \rho v_{t-1} + \eta \frac{\partial l}{\partial W} \\
    W^{(t+1)} &= W^{(t)} - v_t
\end{align*}
\]

where \( \rho \in [0, 1] \) is a constant, typically 0.9 or 0.99. Thus, the parameter change retains some of the previous momentum when updating,
and since different momentum terms are stored for different parameters (or dimensions), the parameter changes will oscillate slower in dimensions with a lot of sudden gradient changes, and accelerate in dimensions with consistent gradients. However, this also means that the model could still miss the local optimum, and the parameter changes accelerate too quickly. Nesterov accelerated gradient descent (NAG) instead uses an estimate of the parameters of the next step when determining $v_t$, and this has been shown to improve RNN performance (Bengio et al. 2012).

Instead, there are algorithms with adaptive learning rates. At the update from time step $t$ to $t + 1$, AdaGrad divides the learning rate by the sum of squares of the gradients of all parameters up until time $t$. If the model is about to update the $k$th parameter $\theta^{(k)}$ out of $d$ parameters (dimensions), then the AdaGrad step is defined as: $\theta^{(k)}_{t+1} = \theta^{(k)}_t - \frac{\eta}{\sqrt{G_{t,k}} + \epsilon} g_{t,k}$ where $G_{t,k} = \sum_{i=1}^{t} g_{i,k}^2$, $g_{t,k} = \frac{\partial l}{\partial \theta^{(k)}_t}$, i.e. the gradient of the $k$th parameter at time $t$, and $\epsilon$ is a small constant, e.g. $10^{-8}$ (Duchi et al. 2011). Since the sum of squares increases over time, the learning rate quickly approaches 0. To address this, AdaDelta limits the sum to a time window and introduces momentum, and RMSProp stores an exponentially decaying average of the sum: $G_{t,k} = \gamma G_{t-1,k} + (1 - \gamma) g_{t,k}^2$ where $\gamma$ is typically approximately 0.9. A very commonly used approach is the Adam (adaptive moment estimation) optimizer due to Kingma et al. (2015), which uses exponential decaying averages for both the sum of squares $v_t$ ("variance", which increases and dampens updates in dimensions varying a little and a lot, respectively) and the sum $m_t$ of past gradients. For a parameter $k$ and with constants $\beta_1$ and $\beta_2$, it would then be defined as:

$$m^{(k)}_{t+1} = \beta_1 m^{(k)}_t + (1 - \beta_1) g_{t,k}$$
$$v^{(k)}_{t+1} = \beta_2 v^{(k)}_t + (1 - \beta_2) g_{t,k}^2$$

To counter the bias towards zero from the zero-initialization:

$$\hat{m}^{(k)}_{t+1} = \frac{m^{(k)}_{t+1}}{1 - \beta_1^t}$$
$$\hat{v}^{(k)}_{t+1} = \frac{v^{(k)}_{t+1}}{1 - \beta_2^t}$$
where, in the denominators, $\beta_1$ and $\beta_2$ are raised to the power of $t$. Finally, the update, incorporating another constant, $\epsilon$, is made:

$$\theta_{t+1}^{(k)} = \theta_t^{(k)} - \frac{\eta}{\sqrt{\hat{v}_{t+1}^{(k)}}} \hat{m}_{t+1}^{(k)}$$

The recommended default values for $\beta_1$, $\beta_2$ and $\epsilon$ are $0.9$, $0.999$ and $10^{-8}$, respectively. Adam was shown to outperform many other optimization algorithms, including SGD (with NAG), RMSprop and AdaGrad (Kingma et al. 2015). RMSprop, AdaDelta and Adam are similar, however, and usually do well in the aforementioned tricky circumstances.
Appendix E

Batch normalization

Batch normalization, due to (Ioffe et al. 2015), requires that for each mini-batch, the mean $\mu$ and standard deviation $\sigma$ for the layer and the batch is calculated. Then, normalization of the scores $s$ of the layer is performed to get the normalized scores $\hat{s}$, which are then sent into the transfer functions. Moreover, for each layer, approximations of the total mean and variance (exponential moving/running averages) of the un-normalized scores over all batches are kept and modified along with the batch computations and the learning, and are then used during inference. Having moving averages also means that it is possible to track the validation accuracy and training loss of the model after each epoch during training, which makes it possible to monitor how fast the network is learning and whether it is under- or overfitting. Batch normalization is then simply a manner of subtracting the mean and dividing by the standard deviation, so to get the batch-normalized scores $\hat{s}$ of some layer: $\hat{s} = \frac{s - \mu}{\sigma}$. Batch normalization improves the gradient flow through the network, allows higher learning rates and makes initialization less sensitive, and is very important in deep learning. This is because learning becomes infeasible with a random initialization when the number of layers increases (above two), and is still faster with batch normalization even for just two layers. When doing inference, the fixed mean and variance calculated during training (the running averages over the batches) is used. Batch normalization can decrease the number of training steps by 14 times, and even though its benefit is more obvious for saturated transfer functions, the speed is still increased greatly for networks employing ReLU transfer functions (Ioffe et al. 2015).
Appendix F

Complementary results

This chapter provides various complementary results that are not necessary for the analysis, but could still be of interest. For instance, the results are presented in various other representations, such as number-of-wins tables (where getting the higher score is defined as a "win"), in an effort to present them in a format similar to those of related works such as Karras et al. (2017) and Taylor et al. (2017).

F.1 The first survey

Complementary results from the first survey are provided in this section. The number of wins (defined as getting the higher of two scores being compared) and draws are presented in tables F.1-F.3. The box plots in figure F.1 illustrate the distributions (including the minimum, lower quartile, median, upper quartile and maximum values) of the ratings for the three models, both for each sentence and in total. Figure F.2 shows the mean ratings for each model and sentence as well as across all sentences. Finally, the inferred probability distributions resulting from the regression, with their 95% confidence intervals, are presented in figure F.3.
**Table F.1:** The number of wins and ties, GT vs. ANN.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>GT</th>
<th>Tie</th>
<th>ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>79</td>
<td>13</td>
<td>58</td>
</tr>
</tbody>
</table>

**Table F.2:** The number of wins and ties, GT vs. baseline.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>GT</th>
<th>Tie</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>98</td>
<td>13</td>
<td>39</td>
</tr>
</tbody>
</table>

**Table F.3:** The number of wins and ties, ANN vs. baseline.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>ANN</th>
<th>Tie</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>89</td>
<td>13</td>
<td>48</td>
</tr>
</tbody>
</table>
Figure F.1: The distribution of the ratings for each model and sentence, as well as across all sentences.
Figure F.2: The means of the ratings for each model and sentence, as well as across all sentences.
Figure F.3: The inferred probability distributions for the ratings of the three models with 95% confidence intervals.
F.2 The second survey

Complementary results from the second survey are provided in this section. The number of wins (defined as getting the higher of two scores being compared) and draws are presented in tables F.4-F.6. Figures F.4-F.6 show the mean points for all pair-wise comparisons for each sentence as well as across all sentences.
Table F.4: The number of wins and ties, GT vs. ANN.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>GT</th>
<th>Tie</th>
<th>ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>37</td>
<td>18</td>
<td>35</td>
</tr>
</tbody>
</table>

Table F.5: The number of wins and ties, GT vs. baseline.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>GT</th>
<th>Tie</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>57</td>
<td>9</td>
<td>24</td>
</tr>
</tbody>
</table>

Table F.6: The number of wins and ties, ANN vs. baseline.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>ANN</th>
<th>Tie</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>56</td>
<td>11</td>
<td>23</td>
</tr>
</tbody>
</table>
Figure F.4: The means of the points gotten for each model and sentence, as well as across all sentences, when comparing GT vs. ANN.
Figure F.5: The means of the points gotten for each model and sentence, as well as across all sentences, when comparing GT vs. baseline.
Figure F.6: The means of the points gotten for each model and sentence, as well as across all sentences, when comparing ANN vs. baseline.
F.3 The third survey

Complementary results from the second survey are provided in this section. The number of wins (defined as getting the higher of two scores being compared) and draws are presented in table F.7. Figure F.7 shows the mean points for each sentence as well as across all sentences. Finally, the inferred probability distributions resulting from the regression, with their 95% confidence intervals, are presented in figure F.8.

Table F.7: The number of wins and ties, GT vs. ANN.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>GT</th>
<th>Tie</th>
<th>ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>71</strong></td>
<td><strong>5</strong></td>
<td><strong>24</strong></td>
</tr>
</tbody>
</table>
Figure F.7: The means of the points gotten for each model and sentence, as well as across all sentences.
Figure F.8: The inferred probability distributions for the points of both models with 95% confidence intervals.