Detecting Deception in Conversational Speech

Mattias Appelgren

5th Year Project Report
Master of Informatics
School of Informatics
University of Edinburgh

2017

Abstract

Deception detection has the aim of determining if a person is lying. The automatic detection of deception has been done on a corpus of conversational speech. I use a Long Short-Term Memory artificial neural network to perform deception classification. Three different feature types are evaluated with Mel Frequency Cepstral Coefficients being the most successful.

Acknowledgements

Thank you to *Matúš* Falis for finding a bug that saved my project and apologies for putting your name in math mode as that was the quickest way I could find to get your accents right...

Most of all, thank you to Steve Renals and Cathrine Lai for guiding me through two years of this project. Your advice on things big and small has been invaluable and I am very grateful.

Table of Contents

1 Introduction **7**

2 Background **9**
2.1 Literature Review . . . 9
2.1.1 Cues to deception . . . 10
2.1.2 Summary . . . 11
2.2 Deception Classification using Machine Learning . . . 11
2.2.1 Audio Features for Deception Detection . . . 11
2.2.2 Textual Features . . . 12
2.3 N-gram bag of word features . . . 12
2.4 Neural Networks . . . 13
2.5 Learning . . . 14
2.5.1 Gradient Descent . . . 14
2.5.2 RMSProp . . . 15
2.6 Recurrent Neural Networks . . . 16
2.6.1 Long-Short Term Memory . . . 17
2.7 Regularisation . . . 19
2.7.1 Weight Decay . . . 19
2.7.2 Dropout . . . 19
2.7.3 Early Stopping . . . 21
2.8 Applied Neural Networks . . . 21
2.8.1 Sequence to Sequence labeling . . . 21
2.8.2 Sequence Classification . . . 22

3 Methodology **25**
3.1 CSC corpus . . . 25
3.2 Data Pipeline . . . 26
3.2.1 Data segmentation . . . 26
3.2.2 MFCC . . . 27
3.2.3 Phone Features . . . 28
3.3 Neural Network Architecture . . . 29
3.3.1 Justification . . . 29
3.4 Overfitting and Regularisation . . . 30

4 Results **33**
4.1 Comparison of Feature Sets . . . 33

4.1.1 MFCC 34
4.1.2 Phones 34
4.1.3 GEMAPS 35
4.2 Choice of Model 35
4.2.1 Discussion 36

5 Conclusions **37**

Bibliography **39**

Chapter 1

Introduction

The detection of deception has long been of interest to psychologists and law enforcement who have wished to detect lies. Contemporary methods, such as the polygraph, are ineffective and intrusive. The results are unreliable and there are several known techniques for cheating the test (to Review the Scientific Evidence on the Polygraph , National Research Council (US). Recently, there has been an increased interest in automatic deception detection from speech, for example, through the 2016 Interspeech deception detection challenge (Schuller et al., 2016). Detecting deception from speech is much less intrusive as it only requires a recording of the speaker to function. This project explores the use of Recurrent Long short-term memory neural networks using features extracted from audio to detect deception.

Neural Network models have not been used for deception detection. These methods have been effective in many other domains within language processing and speech recognition (Graves et al., 2013; Socher et al., 2011, eg.). Since no previous work exist on explicit deception detection, this work focuses on exploration, considering three different feature types as input to the networks, low level acoustic features, phone labels, and acoustic/prosodic features extracted on the phone level. The experiments are done one the CSC Deception Corpus (Hirschberg et al., 2005), a corpus of deceptive conversations containing recorded in high quality audio, transcribed and tagged with truthfulness on a sentence level.

This is the second year of a two year project. In the first year focus was laid on using lexical features extracted from transcriptions. A set of n-gram features represented as a bag-of-words were used to classify deception beating previous state of the art models. The models used were Naive Bayes and Random Forests, both simple classifiers. This years work focuses on using more expressive models and to use the audio signal as an input source.

Last years work had an extensive overview of psychological theories of cues to deception. This work starts by going over some of the main points covered focusing on the aspects that may be important for audio deception detection. Following this an overview of Neural Networks is given, presenting methods used in related fields such as sentiment analysis and speech recognition. In chapter 3 the feature extraction

and neural architecture used for experimentation are used. Chapter 4 presents and discusses the result of the experiments, showing that Mel-frequency Cepstral Coefficients extracted from windowed speech can perform on par with previous state-of-the-art systems. Chapter 5 summarises the main conclusions.

Chapter 2

Background

People lie. The reason why we lie can vary from the innocent, telling a friend they were right to insult a waiter, or the serious, criminals trying to avoid detection. Being able to tell truth from lie can be very important. Psychologists have tried to understand lying and find ways of spotting when someone is being deceived (DePaulo et al., 2003), while law enforcement has mostly relied on equipment such as the polygraph. The polygraph functions by measuring a subjects heart rate and skin conductivity. It is presumed lying will cause a physical change in these measurements. Whether or not the readings can be trusted at all is debatable, but even if they are there are techniques that can allow you to cheat the polygraph. A series of control questions are used to normalise the readings for each subject, however, if the subject artificially heighten their default response the subsequent readings cannot be relied upon (to Review the Scientific Evidence on the Polygraph , National Research Council (US).

In psychology a common approach is to look for cues to deception. These cues can be indications that a true internal state is leaking through your false exterior or heightened emotional response associated to guilt or fear of getting caught (DePaulo et al., 2003). For example, someone who is angry but does not wish to show it may show tiny expressions of anger that only stay for a short moment, or their voice may be altered, more strained, due to held back emotion. Cues may also stem from the added cognitive load the liar goes through as they need to keep a complicated narrative consistent in their mind while responding to questions (DePaulo et al., 2003).

We attempt to detect deception in conversational speech using modern Neural Network models. This chapter covers deceptive cues found in psychology and introduces and reviews neural networks with particular focus on the Long Short-Term Memory networks that have been used for experimentation.

2.1 Literature Review

An extensive review of deceptive cues and features used for machine learning was done in Appelgren (2016) focused on lexical features. This section will reiterate the

main points but focusing instead on potential audio features. For our machine learning algorithms numeric features must be extracted from the available data. To understand what type of features should be extracted. Further, a review of previous auditory and lexical features are examined.

2.1.1 Cues to deception

The central axiom that deception detection is built on is that lying is different from telling the truth. That some physical change can be detected when someone is lying. That they act differently when they are truthful or lie. It is unlikely that a single behaviour is present in all deceptive actions or in all deceivers (Zuckerman et al., 1981). However, looking at the underlying psychological processes and emotions of deception can give some indication of what deceptive cues might look like.

2.1.1.1 Emotion in deception

First let us consider the emotions of deception. Deception can elicit special emotional responses (DePaulo et al., 2003) or may be the main purpose of the lie (Ekman and Friesen, 1969). Emotion is the main purpose of a lie if the lie attempts to hide a true inner (emotional) state. For example, putting on a false smile when we are upset. Keeping up this type of charade is difficult and people might slip. This can manifest itself as a micro-expression, an expression that plays over the face for an instant before snapping into the false expression the deceiver is trying to hide (Ekman and Friesen, 1969). A careful observer may be able to spot these cues, called leakage cues, and can through this infer a deceptive state of mind (Ekman and Friesen, 1969). When speaking about micro-expressions we generally speak of facial expression. However, emotion is also detectable from our speech. It is not unreasonable to imagine that if we have micro-facial-expression then we may also show micro-vocal-expressions.

The other way emotion pertains to deception is through the speakers emotional response to deception. The emotional response will depend largely on the stakes under which the lie is performed. For example, if there is a dire consequence to getting caught then the speaker might feel fear (Zuckerman et al., 1981). If the deceived is someone the deceiver trusts and cherishes then the emotional response could be guilt (Ekman, 1985). Fear can manifest itself as higher pitch, faster and louder speech, more pauses, speech errors, and indirect speech. Guilt can manifest as a type of sadness, lowered pitch, and softer and slower speech.

2.1.1.2 Cognitive Strain

Deception may also cause a cognitive strain on the speaker. A liar must be more careful with their speech as they do not wish to be caught. If the lie is elaborate this will require a story to be kept consistent and believable. This added strain may cause speech to be slower and contain more pauses for thinking (Zuckerman et al., 1981).

This is assuming that the story is being made up on the spot. Lies can often be pre-planned and rehearsed. This might manifest itself in an opposite fashion with faster speech and less pauses (Ekman, 1985).

2.1.2 Summary

Deceptive speech might manifest itself in many ways. These ways are often contradictory and difficult to distinguish from normal speech. However, a starting point may be given to us. Micro-expressions may manifest themselves as quick changes in speech, indicating a need for low level discrimination of speech. Emotion may cause increases and decreases in pitch, faster or slower speech, and louder or softer speech. Therefore the model we use must be able to classify not only linear relationships, to be able to have complex decision boundaries detecting the deviations from the norm.

2.2 Deception Classification using Machine Learning

Work on deception detection is limited. The main focus has been on engineering features and analysing their effectiveness. The main classifiers used are Decision Trees (DT) (Hirschberg et al., 2005), Support Vector Machines (SVM) (Graciarena et al., 2006), and Gaussian Mixture Models (GMM) (Graciarena et al., 2006). More recently language modeling and a variety of other techniques have been used for classification (Levitan et al., 2016, 2015).

In the remainder of this section different features and approaches used for deception detection are reviewed.

2.2.1 Audio Features for Deception Detection

The main features used have focused around extracting statistical properties of pitch, energy, and duration. In the psychology literature many of the emotional states of deception have impact on the pitch of the speaker. For example, fear of discovery is associated with higher pitch. For this reason minimum, maximum, and average pitches are a feature that was used a lot (Levitan et al., 2016; Hirschberg et al., 2005). Loudness can also indicate fear, which means that energy features can be useful. Finally, the cognitive effort could lead to slower speech, or faster speech if the lie was prepared. Therefore, the duration features were also used. In addition to these features statistics about individual speakers can be extracted and used to modify the features. For example, the duration features can be the difference in max pitch from the average max pitch for a speaker. These speaker dependent features were found to be very helpful, increasing classification accuracy greatly.

Higher level features have also been used. These usually involve an intermediate processing step that creates a higher level representation, or extracts some kind of labels from the signal. A clear example of this is a system that used an emotion recogniser

to extract labels of emotion on the deceptive speech. These labels were then fed into a deception classifier (Amiriparian et al., 2016). Another example uses phone recognisers to recognise phones in the sequence. These have been used in different ways. The first is simply to use the phone labels, these were used to create a phone-level language model, one for deceptive and one for non-deceptive speech (Levitan et al., 2016). These were used to predict how likely that the speech was deceptive or non-deceptive. The other method for using phones was to use them as acoustic events. The main feature extracted from them was the phone durations, which could help determine the speed of the speech.

2.2.2 Textual Features

Features extracted from the transcriptions have been mainly three different types: word categories, pauses, and sentiment (Hirschberg et al., 2005). The word category features were extracted using Linguistic Inquiry and Word Count (Pennebaker et al., 2001). The features are made up of categories with psychological significance. It finds counts for a varied group of word categories, such as personal pronouns and positive sentiment words. These are relevant since the psychology literature deceivers are often more negative or try to remove themselves from their lies and this can present itself as speech using less personal pronouns and more negative sentiment words. They also looked for negativity through sentiment analysis. These types of algorithms generally give a score that is positive for positive speech and negative for negative speech (Socher et al., 2011).

Filled pauses such as “um” and “uh” may also be an indication of truthfulness/lies (Benus et al., 2006). Since deception may cause higher cognitive load the use of filled pauses could indicate added stalling for time when the deceiver attempts to think what to say.

2.3 N-gram bag of word features

In (Appelgren, 2016) features were extracted from transcribed speech. The features tested were n-grams, pos-tags, and sentiment features. Naive Bayes and Random Forests were used to classify into “truth” or “lie”. Extensive feature selection was done to pick out the top features for predicting these classes. The top features were 27 n-grams classified with a Random Forest (RF+FS). None of the pos-tags or sentiment features were included in this final set.

For deception detection text features have often overfit to the task (Levitan et al., 2016). The performance will depend very much on the way each individual speaks normally and under pressure and what subject is being spoken about. For example, in this case interviews about people’s performance is being done. Therefore words such as “good” and “poor” are unusually significant. This could clearly be seen when feature selection and classification was run on each individual speaker. The most significant features for

each speaker was very different, as was the performance of classification (Appelgren, 2016).

A main criticism of the work is that the models used are simple. Since deception can manifest itself in many different ways the use of a model that can express more powerful relationships between features would be preferred. This years work focuses on addressing this issue by the use of Recurrent Neural Networks

Two main criticisms of the work are that the models used were very simple and that only text features were used. The Naive Bayes and Random Forest classifiers are not well used nowadays and are generally seen as inferior to other techniques.

This years work has focused on addressing these two main points by using audio features and by using more modern techniques in the form of Recurrent Neural Networks.

2.4 Neural Networks

In this work we will be using a type of Neural Networks (NN). The basic NN, also known as a Mulit-layer perceptrion (MLP) consists of layers of hidden units $\mathbf{h}^{(l)}$. Each hidden layer is calculated through matrix multiplication between a weight matrix $\mathbf{W}^{(l)}$ and the previous hidden layer $\mathbf{h}^{(l-1)}$ before being put through a non-linear function f:

$$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \tag{2.1}$$

$$\mathbf{h}^{(l)} = f(\mathbf{a}^{(l)}) \tag{2.2}$$

The first layer takes as input the feature vector $\mathbf{x}$

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b} \tag{2.3}$$

$$\mathbf{h}^{(1)} = f(\mathbf{a}^{(l)}) \tag{2.4}$$

The final, output layer, functions like the other hidden layers, except a different function g is used instead of f.

$$\mathbf{y} = g(\mathbf{a}^{(L)}) \tag{2.5}$$

The choice of g depends on the expected output of the network and is often different from f. For classification problems $softmax$ and the sigmoid function $\sigma(a)$ are the most common choices:

$$softmax(\mathbf{a})_c = \frac{\exp(a_c)}{\sum_{c'=1}^{C}\exp(a_{c'})} \tag{2.6}$$

$$\sigma(a) = \frac{1}{1+\exp(-a)} \tag{2.7}$$

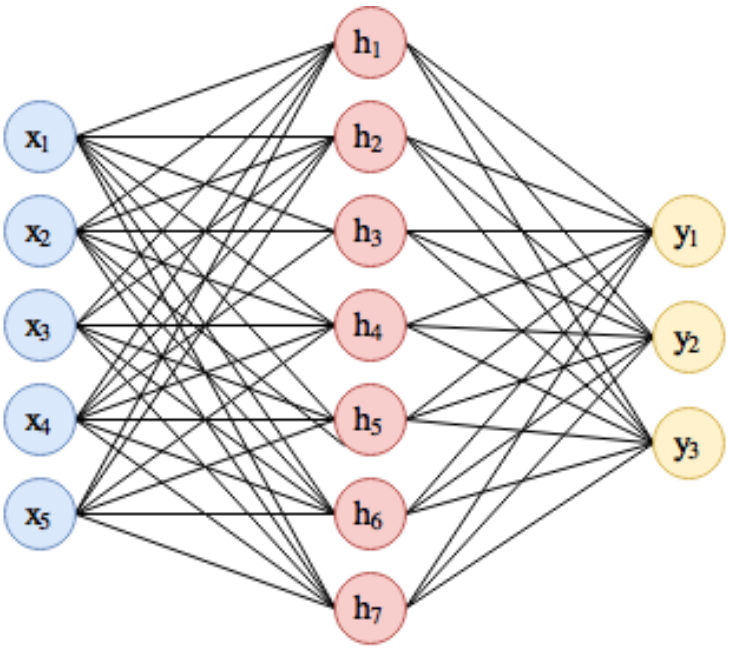

Figure 2.1: A basic neural network with one hidden layer and tree outputs. Each hidden unit is the weighted sum of the inputs with an activation function. The outputs in turn are the weighted sum of hidden layers applied to an activation function.

$\sigma(x)$ is generally interpreted as a binary probability distribution, true or false, for example, the presence or absence of an apple in a picture. A large positive a will lead to a high probability and negative to a probability close to 0. Softmax is interpreted as a probability distribution over C classes. The denominator is a normalisation constant that ensures that the softmax probabilities sum to 1.

The two major hyper parameters that need to be set when designing an MLP are the number of hidden layers and number of hidden units. Figure 2.1 shows a neural network with 1 hidden layer and 7 hidden units in that layer. There are 6 input features and three outputs.

2.5 Learning

Machine Learning must have a way of fitting the model to data. The variables that are fit here are the weights $\mathbf{W}^{(l)}$. These are trained by a gradient update function aiming to minimise a cost function. The gradient approach is used since there is no analytical solution to find the best setting of the weights.

2.5.1 Gradient Descent

Given an error function $E(\mathbf{D},\mathbf{W})$ where $D = \{(\mathbf{x_1},\mathbf{y_1},...,(\mathbf{x_N},\mathbf{y_N})\}$ is the dataset and $\mathbf{W}$ are the trainable parameters $\mathbf{W}^{(l)}$. We update the weights by finding the partial

derivate of the error given each weight: $\nabla E(\mathbf{D}, \mathbf{w})$ and changing the weight by a factor of the learning rate η in the direction of the minimum.

$$\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{D}, \mathbf{w}) \tag{2.8}$$

Where $\nabla E(\mathbf{D}, \mathbf{w})$ is a matrix of partial derivatives.

To successfully learn a pick a good learning rate η must be picked. The learning rate determines how large a step is taken down the gradient. If the step is too large we may step over the optimal value meaning an optimal value will never be reached. If η is too small then learning will take a very long time. We may step in the right direction but we will never reach the end.

An additional problem with the learning rate is that the optimal learning rate may be different at different times in training. Early on large learning rates that bring us close to the optimum quickly are preferred. While a smaller learning rate, ideal for learning finer details, is preferred in later stages. Thus, some people use learning rate schedulers to change the learning rate at different times, starting high and reducing as the number of epochs increases (Darken and Moody, 1990).

2.5.2 RMSProp

A problem that standard gradient descent runs into is learning getting stuck due to too small gradients. What happens is that the weight will change too slowly to make any meaningful progress in a timely fashion. To get around this a fixed update can be done instead of one based on the gradient size.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathit{sign}(\nabla E(\mathbf{D}, \mathbf{w})) \tag{2.9}$$

Where *sign* is $+1$ for positive elements and -1 for negative elements.

Thus we move the weights by η in the direction of the (negative) gradient. The magnitude does not make a difference. This algorithm solves the fact that small gradients will slow learning. However, ignoring the magnitude can be quite damaging to the learning results. If nine mini-batches have gradient -0.1 and one has 0.9 then we would expect the weight to stay at about the same position. However, with this approach the new weight will be $9\eta - 1\eta$, which is not what we desire.

RMSprop is introduced to solve this problem (Tieleman and Hinton, 2012). A decaying sum of previous gradients is kept such that the gradient step can be modified to remember which direction the gradient has gone, effectively increasing the learning rate when the movement is all in one direction and reducing it when going in the wrong direction:

$$r_t = (1-\gamma)\nabla E(\mathbf{D}, \mathbf{w}_t)^2 + \gamma r_{t-1} \tag{2.10}$$

$$\mathbf{v}_{t+1} = \frac{\eta}{\sqrt{r_t}\nabla E(\mathbf{D}, \mathbf{w}_t)} \tag{2.11}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{v}_{t+1} \tag{2.12}$$

2.6 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a neural network that can be applied to sequences of input vectors $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_T\}$. The same network is applied to each input, however, it "remembers" information from the previous time step by taking as input the previous hidden state.

$$\mathbf{a}_t = \mathbf{W}\mathbf{x}_t + \mathbf{b} \tag{2.13}$$

$$\mathbf{h}_t = f(\mathbf{a}_t; \mathbf{h}_{t-1}) \tag{2.14}$$

Again, f is some activation function. The function could be anything from a simple sigmoid function to a long-short term memory cell (Hochreiter and Schmidhuber, 1997) (see section 2.6.1).

There are two main ways that RNNs: sequence-to-sequence (seq2seq) or sequence classification. In seq2seq each input is associated with an output. For example:

$$\mathbf{y}^t = net(\mathbf{h}_t) \tag{2.15}$$

where *net* could be any neural network, including simply a single output layer.

In sequence classification the whole sequence is processed by the network before the final hidden unit is fed to an output network:

$$\mathbf{y} = net(\mathbf{h}_T) \tag{2.16}$$

For more on this see section 2.8.2.

These networks are trained through a special learning algorithm called backpropagation through time (Werbos, 1990). Since the same network is applied to several inputs. Thus the weight update will depend on the gradient of the error with regard to each of the previous input states.

The error is backpropagated through the hidden state that is passed forward to the next stage.

2.6.1 Long-Short Term Memory

Long Short-Term Memory (LSTM) cells are a special recurrent network function that has been designed to solve a problem observed in the classical RNN network experiments (Hochreiter and Schmidhuber, 1997). Originally the sigmoid function (equation 2.7) was most commonly used as activation function. The problem faced by these early networks was that the gradient would quickly either go towards zero or blow up and become really large. This would cause the network to be unable to learn long term dependencies as after a few time steps the gradient was too small to influence the weights in a meaningful way.

The LSTM cell solves this problem by having the internal state not be affected by a non-linear function. This means that the derivative will simply be **1** which means that the error will backpropagate without changing size (Hochreiter and Schmidhuber, 1997).

Each LSTM has a hidden state C_t. Three gates decide what input will be fed to the state, what parts will be remembered or forgotten, and what will be given as output. These are the input gate, i_t, the output gate, o_t, and the forget gate, f_t. Each gate is a value between 0 and 1. If the value is 1 it lets the associated value through, if it is zero it blocks it.

The input gate $\mathbf{i}_t$ decides what input might be fed to the memory cell.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{2.17}$$

$$\hat{\mathbf{C}}_t = \tanh(\mathbf{W}_C[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \tag{2.18}$$

$$\tag{2.19}$$

The forget gate $\mathbf{f}_t$ decides what part of the previous memory cell will be kept, remembered, and which will be forgotten.

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{2.20}$$

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \hat{\mathbf{C}}_t \tag{2.21}$$

$$\tag{2.22}$$

The output gate $\mathbf{o}_t$ decides what part of the memory cell will be given as output in this time step.

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{2.23}$$

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t) \tag{2.24}$$

A graphical representation of the LSTM can be seen in figure 2.2

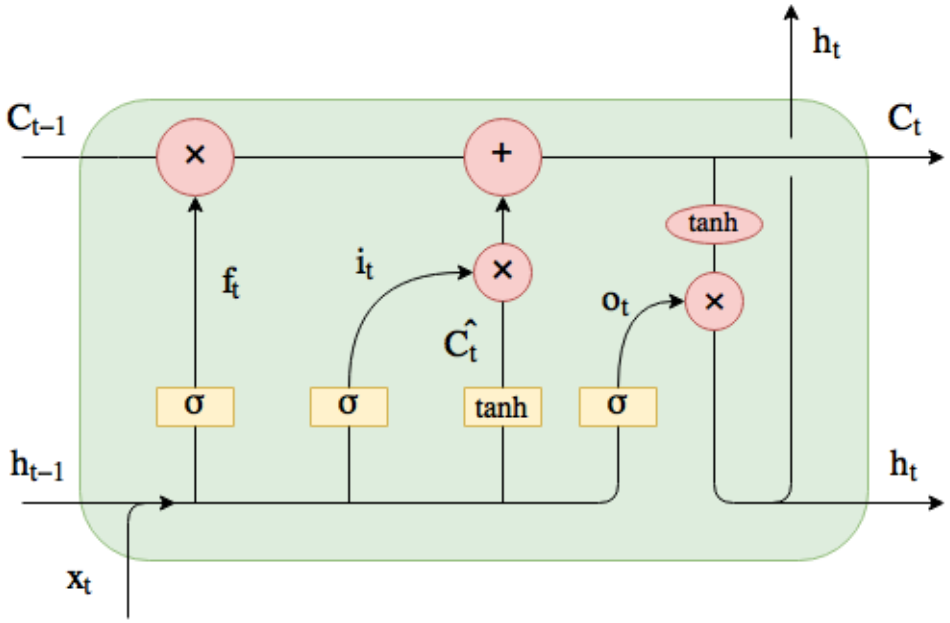

Figure 2.2: An LSTM cell. C_t is the memory cell that is fed forward in time. A new potential cell $\hat{C}$ is created from the inputs x_t and previous time-step outputs h_{t-1}. The input gate i_t decides what part of $\hat{C}$ can be added to the memory cell. The forget gate f_t decides what parts of the previous memory cell will be kept and which will be forgotten. The output gate o_t decides what part of the memory cell will be passed on as the outputs h_t.

2.7 Regularisation

Neural networks often have huge numbers of parameters that need to be trained. The number of parameters can even be larger than the the number of data points available. This means that the function can very easily overfit to the training data. Overfitting happens when the model picks up on noise present in the training data and associating it to the labels. This means that the model will not generalise well to unseen data. Regularisation are techniques for dealing with this problem.

2.7.1 Weight Decay

One way of regularising is to make the model simpler. To use the least amount of information to solve the problem. If the weights are larger then the model be more complicated as a small difference in one feature might make a big difference for the prediction. A simpler model would have small differences in features make small differences in prediction. Thus, if we limit the magnitude of the weights the model may generalise better.

To do this we update our error function to include a term that associates a high cost to big weights.

$$E_{reg}(\mathbf{D}, \mathbf{w}) = E(\mathbf{D}, \mathbf{w}) + \lambda ||\mathbf{w}|| \tag{2.25}$$

When the derivative of this error function is taken the weight magnitude term will push each weight toward zero. If the error derivative does not push harder in the opposite direction then the weights will decay over time.

This is called the l2-cost. There are other ways of associating the cost with the weights, but this is the most common. When using this method we do have a new hyper parameter that needs to be set, the λ which decides how strong the decay will be.

2.7.2 Dropout

A regularisation method used with other classifiers is to train a large number of simple classifiers and average their result. An example of this is the Random Forrest classifiers, where decision trees are created with some randomness inserted into the process. The result of all the decision trees are used to vote for the output. The hope is that these will pick up on different parts of the problem and if some have fit to noise then others will not and will compensate.

Dropout takes this idea to neural networks. Training many neural networks can be costly and take a lot of time. Instead, a single network is trained, but each training iteration some of the hidden units are dropped from the computation graph. This means

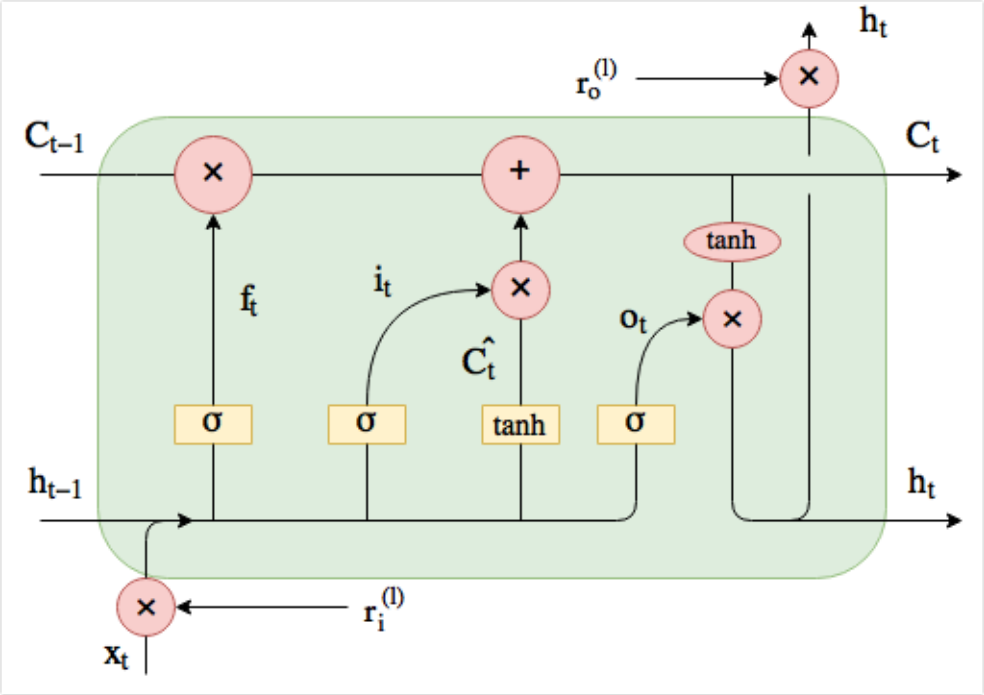

Figure 2.3: To the basic LSTM is added the dropout variables r_i and r_o (i and o stand for input and output). These regulate whether or not an input connection will be used or not. If the value is 0 then that value will be dropped while if it is 1 it is kept.

that it is as if a new network is trained in each iteration, each one slightly different from the others. However, these networks all share weights.

The dropping is done by sampling a mask $m \sim Bernoulli(p)$, taking values from $\{0,1\}$. The mask is multiplied with the hidden units. This will drop some of the weights so they will not impact this learning stage.

$$\mathbf{m}^{(l)} \sim Bernoulli(p) \tag{2.26}$$

$$\hat{\mathbf{h}}^{(l)} = \mathbf{m}^{(l)} * \mathbf{h}^{(l)} \tag{2.27}$$

$$\mathbf{a}^{(l+1)} = \mathbf{w}_i^{l+1}\hat{\mathbf{h}}^{l} + \mathbf{b}^{l+1} \tag{2.28}$$

$$\mathbf{h}^{l+1} = f(\mathbf{a}^{(l+1)}) \tag{2.29}$$

This method has worked very well in practice allowing more complicated, larger architectures to be used while still keeping good generalisation (Srivastava et al., 2014).

2.7.2.1 Dropout in RNN

Dropout was originally developed for straight forward MLPs, not for RNNs. This meant that dropping any hidden unit would not damage the output in a harmful way. However, in RNNs, randomly removing some of the recurrent connections would interfere with learning long term dependencies as the network would have no way of guaranteeing that what it puts in one of the memory cells will still be there when it is

needed. Therefore, one of the first way that Dropout was applied to RNNs and LSTMs was by only dropping only output connections but leave the recurrent connections untouched (Pham et al., 2014).

2.7.3 Early Stopping

Early stopping is very simple. Stop training when the validation error stops improving. If the weights start small then they will be small for a large number of iterations before they grow large. This means that earlier iterations will have simpler models than later ones. Thus, if we train and the validation error is improving then we train until the time that it does not. When it stops improving we will have started overfitting.

This works well because a complex model will first explore simple solutions before using more complex, higher weight, solutions (Caruana et al., 2000).

2.8 Applied Neural Networks

Deep learning has been very effective in a number of fields from machine vision (Sermanet et al., 2012; Tompson et al., 2015; Osadchy et al., 2007, eg.) to speech (Sak et al., 2015; Abdel-Hamid et al., 2012) and language processing (Socher et al., 2011; Mikolov et al., 2010). These models have beaten previous state of the art in these areas and keep improving.

As of yet, no-one has used neural networks to do deception detection.

When we deal with language we are usually dealing with a sequence. With text a sequence of words makes up a sentence. Speech is a continous wave. It is possible to use raw waveforms as input (Sainath et al., 2015) but generally the wave is discretized by using windows of about 20-30ms ever 10ms. This is much easier for our models to deal with as input. This is the manner that we will deal with our speech data. Therefore we shall look at how sequence data can be dealt with through neural networks. The two main model types we will look at are sequence to sequence (seq2seq) and sequence classification models.

2.8.1 Sequence to Sequence labeling

Seq2seq problems are those where a sequence of inputs is processed to create a sequence of inputs. A straight forward example is part-of-speech tagging. Here a sentence of words will be translated into a sequence of part-of-speech tags. This can be modeled in a straight forward way with RNNs. Figure 2.4 shows a basic recurrent seq2seq model. The hidden layer is updated using the previous state and current input and an output is returned every time step. LSTM networks are commonly used for this type of task such as for speech recognition (Graves et al., 2013), part-of-speech tagging (Plank et al., 2016), and language modeling (Sundermeyer et al., 2012). A common

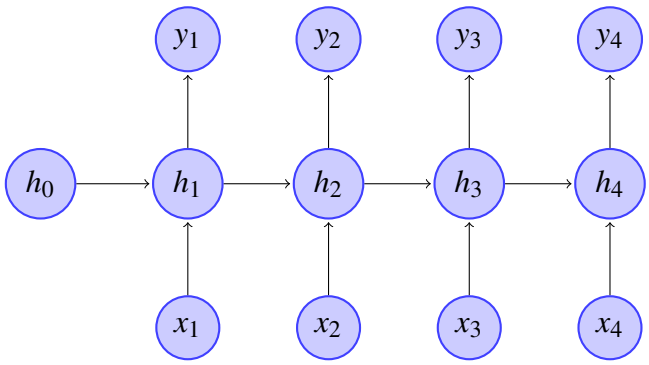

Figure 2.4: A basic sequence to sequence model. For each input an output is given. The hidden state is passed on such that the future outputs will depend not only on the current input but also the previous outputs.

extension above using the straight forward LSTM architecture is using bidirectional LSTMs (Graves et al., 2013; Plank et al., 2016). Bidirectional networks process input in both the forward direction and backward direction. This results in two separate hidden units for each time step that are used to produce an output. This type of network is very effective for this type of problem as it allows the output labels to be conditioned on the whole sequence, not just the previous t outputs.

2.8.2 Sequence Classification

The second type of sequence data problem is sequence classification. Whereas in seq2seq problems both the input and the output are sequences, in sequence classification the output size does not depend on the input size. For example, in sentiment analysis the output might be whether or not the sentence is positive or negative. Using an RNN architecture perform this task will look something like figure 2.6. The input sequence is read into the recurrent hidden unit. However, the output of each hidden unit is ignored until the last time step. The output of this stage is fed into a classification layer that does the final prediction. This can be seen as creating a vector representation of the sentence that captures all of the information contained in all inputs in the final hidden state.

Using RNNs is not the only way to create a sentence representation. Two other approaches have been popular - tree structured Recursive Neural Networks (Socher et al., 2011) and Convolutional Neural Networks (CNNs) (Kalchbrenner et al., 2014). While Recurrent Neural Networks build up a sentence representation by reading words one by one in order these other types do not follow this restriction. Recursive networks almost function as a type of parsing. The inputs are arranged in some form of tree. Two adjacent inputs can be combined into a new vector that is meant to represent the meaning of both inputs (and are explicitly trained to do so (Socher et al., 2011)). This new vector can further be combined with other adjacent inputs. This process is repeated recursively until the whole sentence has been represented. The final vector is used to classify overall sentence (see figure 2.7) (Socher et al., 2011). Originally the

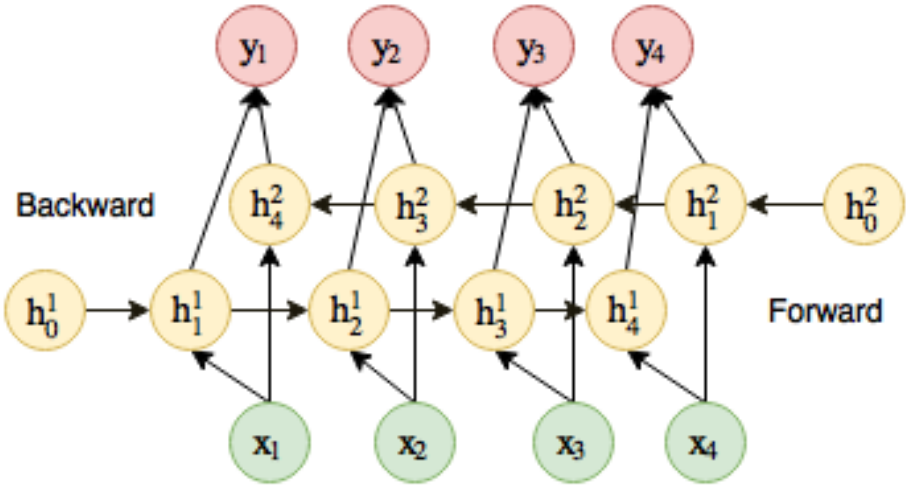

Figure 2.5: A bidirectional Recurrent Neural Network. A forward and backward pass are made over the inputs. Outputs will be calculated using a hidden unit from both the forward and backward pass.

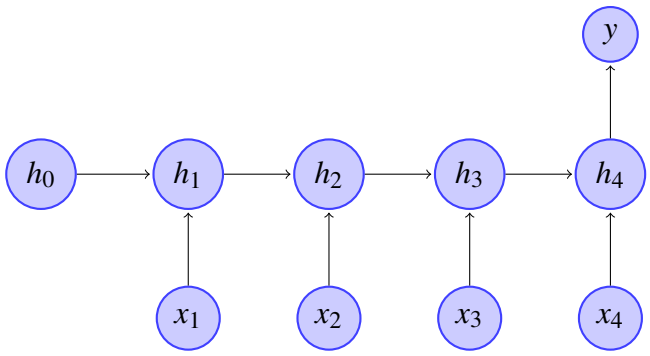

Figure 2.6: A recurrent neural network for sentence classification. Each input is fed to the hidden units that combines a previous hidden state and the inputs. The final hidden state h_4 is interpreted as a representation of the whole sentence. The output of this hidden state is fed to a classification network resulting in label vector $\mathbf{y}$.

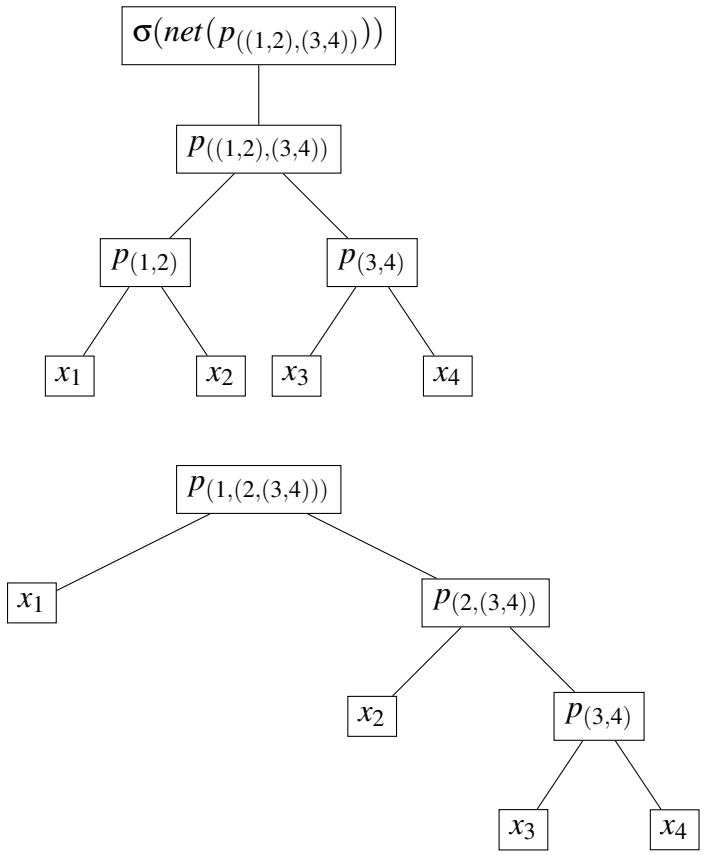

Figure 2.7: Here are two examples of recursive neural networks. Each leaf x_i is a word represented as a real valued vector. The network is applied on two words creating a new representation $p_{(i,j)}$. This can then be used as input together with a new word or phrase representation. The final representation is passed to an output network with a sigmoid output layer represented as $\sigma(\cdot)$ in the first network. The recursion can be applied in different ways. A heuristic search function can be used to pick the best tree.

combination was done using a fairly straight forward MLP (Socher et al., 2011), however, it has since been extended to use LSTMs to store the sentence representation (Tai et al., 2015).

The CNN networks work quite differently from RNNs. The exact details have no baring upon this project, so this is included as a possible alternative. A good overview of these networks can be found in Kim (2014). The overall effect is that we can create a sentence representation that works like a more powerful bag-of-words model. There is some structure information available but it is not as important here as it is for seq2seq problems. These models have successfully been applied to problems such as sentiment analysis (Kalchbrenner et al., 2014). Sometimes CNNS and RNNs are also combined. This is especially true if three are two levels of sequence that are being dealt with. For example, if a whole document is being represented as a vector, a sentence representation can first be created using CNNs. Each sentence representation can then be input to a RNN model that combines these into a document representation.

Chapter 3

Methodology

The purpose of this work is to explore the possibility of using LSTM neural networks to classify deception. The focus will be on using features extracted from the audio signal, rather than the transcription. Since there has been no previous work on the topic, the best choice of feature representation is not known. Where work could have been done exploring different possible architectures we will instead focus on extracting different sets of features and comparing their effectiveness. We will then compare the results against previous state of the art approaches and comment on the validity of these approaches.

In this chapter I will describe the data used for this experiment. The steps taken to extract numeric features from the audio and describe the neural network architecture used to do experiments.

3.1 CSC corpus

The CSC corpus (Hirschberg et al., 2005) consists of 7.2 hours of subject speech, tagged with truthfulness on two levels. It was gathered in an interview scenario where subjects were given a financial incentive to lie in parts of the interview. The data consists of both speech in two channels, interviewer and subject, and a hand transcription that has been time aligned to the speech.

In the study each subject was scored on six different tasks. The scores were compared to a profile based on "the 25 top entrepreneurs of America". However, the results were manipulated such that the subject score was better than the profile in two tasks, worse in two tasks, and exactly the same in two. They were told that they would get a $100 cash prize if they could convince an interviewer that they had scored exactly as well as the profile. This meant they were motivated to tell the truth in two areas and lie in four. The interviewer had to work out what their actual score was and could ask any question except any exact question asked during the tasks. The subjects indicated if what they said was true or false by pressing a pedal that was hidden from the interviewer.

In total 32 speakers of "Standard American" were interviewed and each interview was

25-50 minutes long. In total, 15.2 hours of interview was recorded of which 7.2 hours was subject speech. The interview scenario with monetary reward can be seen as a low stakes reason for lying. It may not invoke as many emotional responses as lying from fear or shame would do. However, the lies are spontaneous, rather than planned, and each subject can largely use their own judgment when lying is appropriate for their goal.

The speech is transcribed by human annotators and the transcription is automatically force aligned to the speech using an automatic speech recognition system. There is a separate audio file for the speaker and for the interviewer and the audio quality is good such that there is no noticeable cross-talk or noise.

The corpus is tagged in two respects. The "big lie/big truth" tag indicates the overall truthfulness of an interview segment, for example when the interview is about a task where the subject performed worse than the target profile and she is trying to talk herself up, then, that whole segment will be tagged as "big lie". However, individual sentences within these segments may not be lies but the aim is still to deceive. The second type of tagging is "little lie/little truth". This comes from the data gathered by the pedal. Here individual sentence-like units (SUs) are tagged with truthfulness. For example, if I were a subject and I had said "My name is Bob." that would be tagged as "little lie". The big lie/little lie tagging scheme has nothing to do with the magnitude of the lie, only with the level on which it was performed. For this project I have been looking at the "little truth/lie" tags and from now on they will be referred to simply as "truth" and "lie".

3.2 Data Pipeline

3.2.1 Data segmentation

I use the same data segmentation form as last year. Segmentation is based on the truth-lie labeling. The labels has been aligned to the transcription as can be seen in figure 3.1. The data is split up along the truth-lie borders. As we can see the segmentation will not always contain a single sentence or sentence-like unit. This should make sense as the lie will be the whole statement not just an individual sentence.

The audio files are split upon the borders of the truth/lie tag, which is aligned to the audio with start and end times. Additionally, the tag will often encompass periods when the interviewer is speaking. During this time the subject will generally be silent (marked <SIL>). These silences are removed if they appear at the start or end of the audio segment. When the silence occurs is determined by looking at the transcriptions.

Thus, for each label an audio file was created amounting to roughly 3700 individual files used to extract features.

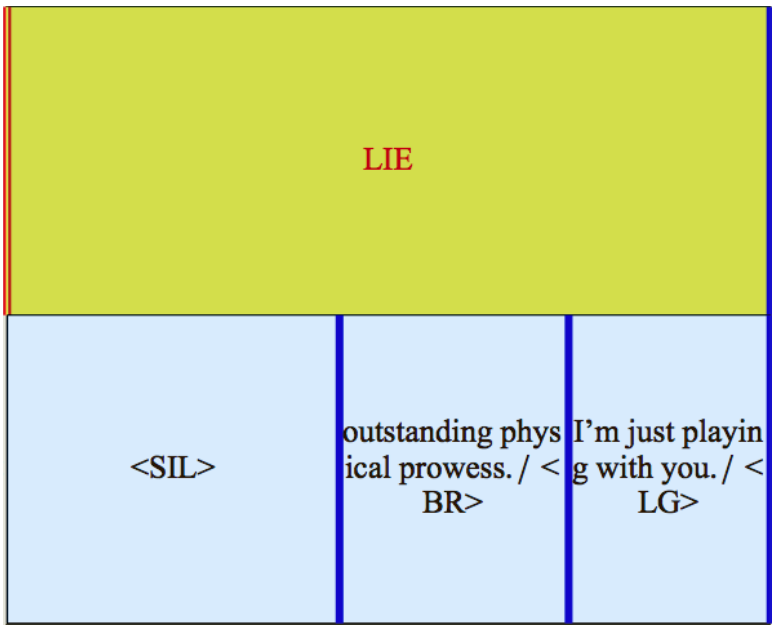

Figure 3.1: A single data point with label "lie". The label (yellow) is time aligned to the transcript (blue) and to the audio file (not shown).

3.2.2 MFCC

The Mel Frequency Cepstrum Coefficients (MFCCs) are a popular feature representation in speech recognition (Rabiner and Juang, 1993). They give a low level description of audio that models human perception of hearing.

The most common way to use these features is to first split the audio into a number of short windows, 20-30ms long. The features are extracted from each of these windows in four stages.

1. The audio is transformed from the time domain to the frequency domain through a discrete Fourier transform (DFT), which means a power spectrum $P(f)$ - indicating the power of the wave of frequency f
2. The power spectrum is warped along the frequency axis into the mel-frequence: $M(f) = 2595 \log 10(1 + f/700)$ giving us the mel-power spectrum $P(M)$. This approximates the perception of humans
3. Next we take the log of the square of the power spectrum: $Y = \log P(M)^2$. This is called the filterbanks and is sometimes used as a feature instead of the full MFCC.
4. Finally an inverse DFT (iDFT) puts everything back in the time domain.

Twelve filterbanks plus the logged energy of the frame are used as features. The first and second derivatives are taken for each of these values resulting in a total of 39 features.

I have extracted my MFCCs using the openSmile software (Eyben et al., 2013). I have used a configuration file provided with the software that extracts the 39 MFCC features

from Hamming windows of length 25ms that are extracted every 10ms. The sequences are anywhere between 100 and 9000 steps in length.

3.2.3 Phone Features

In three of the papers responding to the 2016 Interspeech Compare Deception challenge (Schuller et al., 2016) phone features were used. In each a speech recognition system was used to automatically extract the phones. They were then used in three different ways. Levitan et al. (2016) created a phone level language model for deceptive and non-deceptive speech. These were used to score the likelihood that some speech was deceptive or not. Gosztolya et al. (2016) simply counted the number of occurrences and duration of each phone in their phoneset. Herms (2016) categorised the phones into vowlels, non-vowels, filled pauses, and silent pauses. Duration and count statistics were then extracted for these different groups. For example, the speaker rate and duration of silence. Each of these techniques helped improve classification above the baseline with the phone language model being the most effective on the dev set but performing badly on the test while the other two were slightly better on both dev and test set.

There are phone labels available for a third of the CSC deception corpus, about 1300 data points. Since this is such a small portion this is a bit too little data availability to use on its own. Therefore I decided to extract the phones using the Brno automatic phone recogniser[1] (Schwarz et al., 2006). The recogniser is used to extract Hungarian phone labels. This gives a time-aligned sequence of labels for each audio file. These phones can be used in one of two ways. The first is by using the labels as input to the classifier similar to Levitan et al. (2016) and Gosztolya et al. (2016). The second is to extract features on the phone level similar to what was done by Herms (2016). I use both of these methods.

3.2.3.1 Phone labels

Using the phone labels as input is done by assigning each label a unique id. A phone is represented in vector form with a 1-hot vector. This is a vector of length equal to the number of phones. All values are zero except for the value at the index of the phone being represented. This encoding is commonly used for words in NLP tasks.

3.2.3.2 GEMAPS

The time-aligned phones are used as time-steps. To do this, the audio files are split upon the start and end time of each phone. This results in a number of audio files equal to the number of phones. A feature extraction method is applied to every one of these audio files resulting in sequences of feature vectors where each feature vector represents one phone. The features were the GEMAPS feature set (Eyben et al., 2016).

[1] http://speech.fit.vutbr.cz/software/phoneme-recognizer-based-long-temporal-context

It is a set of acoustic and prosodic features that have been developed to be a reasonable baseline on emotion recognition and other paralinguistic tasks. The set contains 88 features including low level descriptions related to pitch, jitter, formant frequency, and energy, means and averages of the low level descriptions, and MFCCs.

The dataset is intentionally kept small as opposed to the dataset released as part of the Interspeech ComParE (IS) that contains thousands of features. Whereas the IS feature set is meant to be an exhaustive set of features from which the most useful for an individual task can be picked out the GEMAPS set is a small reasonable baseline that can be used largely as is.

I used the OpenSMILE software to extract GEMAPS features for each phone. Using one of the configuration files available in the downloaded software package.

3.3 Neural Network Architecture

The Neural Network architecture I used for this problem was an LSTM network for sequence classification (see section 2.8.2). For each variable length sequence a single output label was predicted using a single softmax output layer. The problem is a binary classification problem so there are two output nodes for the softmax layer - truth and lie. Each output indicates the probability of the output being true or false.

The network was trained to minimise the cross entropy:

$$E(D,w) = CE(D,w) = \sum_{n=1}^{N} \mathbf{y}_n lg(\mathbf{y}'_n) \tag{3.1}$$

Where y_n is the expected label and y'_n is label predicted by the network for data point n and $D = (x_n, y_n)$ is the dataset.

The learning update I used was RMSprop using the default values for $\eta = 0.001$ and $\gamma = 0.9$ which were found to give satisfactory learning.

Additionally, I ran experiments using regularisation with dropout on the output and early stopping.

This is implemented using Google's open source neural network framework TensorFlow(Abadi et al., 2016). The implementation uses the built in versions of LSTMs (Hochreiter and Schmidhuber, 1997, based on), and RMSProp (Tieleman and Hinton, 2012, based on). Built in to their optimisers are backpropagation and automatic differentiation that were used to train the model.

3.3.1 Justification

The LSTM architecture was chosen for its success in various other sequence related tasks. It has been shown to be much easier to train than a standard RNN and has

been able to learn long term dependencies. This is especially important for the MFCC features as the sequences are very long. A different possible hidden layer would be the GRU unit. This is a recurrent unit very similar in concept to the LSTM. It was developed as a simpler version of the LSTM. The performance of the two types of units are very similar and I found no compelling reason to choose GRU over LSTM (Chung et al., 2015). I could also have tried to use the recursive networks of Socher et al. (2011). However, these were developed for text input and leverage a parsing behvaiour. With audio features this does not seem to make a lot of sense.

A different method would be using CNNs to create a sentence representation. This is a reasonable thing to do as CNNs have been applied to speech before. This will be left as potential further work.

Cross entropy as the error function is the most commonly used error function. There were thoughts of looking into other functions that could act to prefer different solutions, such as preferring high precision over high recall. However, nothing would really be more effective than simply using cross entropy and let any precision/recall trade-offs be determined with a decision function picking a higher decision threshold than a standard $p(a) > p(b)$ decision, such as $p(a) > 0.75$.

RMSprop as chosen as the optimiser for its power and ease of use. Standard SDG requires extensive fitting of the learning rate which does not have a simple or obvious solution. RMSprop on the other hand usually works well with a default learning rate of 0.001 and does not require as much hyper parameter optimisation. This means that valuable experimentation time can be spent on other hyperparameters and on real experimentation rather than dealing with the uncertainty of whether the architecture is bad, the learning rate is incorrect, or if the overall approach is incorrect.

The network is regularised with dropout. The two main choices for regularisation is adding a regularisation term into the error function, such as l1 or l2 loss or using dropout. Recently dropout has become the most popular way to regularise deep learning functions. The reason for this is that it seems to regularise much more effectively than the loss functions and is also easier to apply. Finding a good balance between too much l2 loss, causing weights to go to zero, and too little, meaning ineffective regularisation is difficult. It is yet another hyper parameter to fit. Additionally, even when optimal, dropout seems to outperform l2 loss as it allows for the building of bigger, especially wider networks, which l2 does not. Dropout often works well with a rate of 0.5 however, modifying this rate can often improve matters. Another possibility I tried was annealing the dropout rate, starting at a high rate of dropping and going towards keeping everything.

3.4 Overfitting and Regularisation

This section will examine the effects that regularisation has on the different models. First, we look at the learning curves of some unregularised networks. Subsequently, we inspect the learning curves of models regularised with dropout and see how these improve the learning curves.

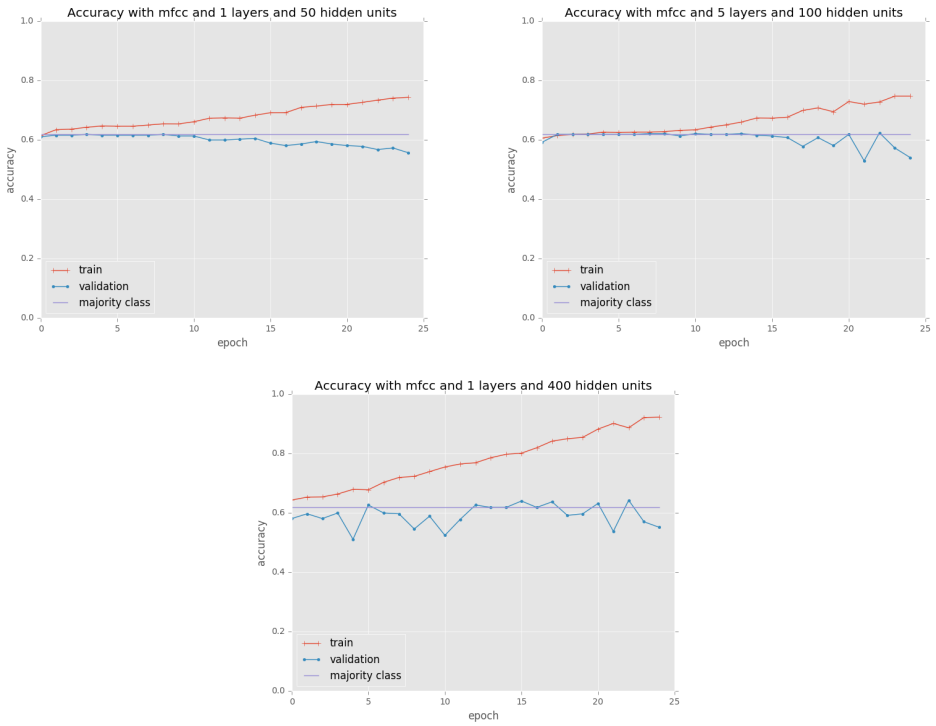

Figure 3.2: Three examples of models overfitting to the training set. The first simply has the validation accuracy go down as training goes up. The other two show a different behaviour where the validation accuracy jumps up and down seemingly at random while train accuracy goes up.

Figure 3.3: Two models regularised with dropout. The jumping behaviour has largely disappeared. At the end of training the networks are still overfitting but earlier epochs look much more promising.

Figure 3.2 shows three different learning curves that seem to have overfit in different manners. First, we have the single layer 50 unit network. We see that the validation accuracy improves slightly at the start together with the training accuracy but after 10 iterations the validation accuracy is quickly going down while train accuracy is increasing. For the 5 unit 100 layer model we see that the accuracies are close at the start but in the final few iterations the validation accuracy starts bouncing up and down. If we look closely we can see that the training set is also going down slightly where there are great dips in validation accuracy. It thus seems that the gradients may be going back and forth over some variable that is important for both datasets. However, the overall result of training are going up. This seems to indicate that some features that are unhelpful are being optimised but that those that are most helpful are being skipped over.

Once we add regularisation (see fig. 3.3) the effects of overfitting are reduced but in no way have they disappeared. It seems like the 5 layer 100 unit model has the validation accuracy follow the training set for slightly longer and slightly higher and that the random oscillation is much reduced or has disappeared completely. We can still see that there is a large discrepancy between the training accuracy and the validation accuracy, especially as time goes on. Overall it seems that the need for regularisation is there. It also seems that dropout is improving things. Since the overfitting is still present, especially as more training iterations were done the two methods of regularisation we will use are dropout and early stopping. These techniques were both easy to apply to the network for fairly painless success.

A possible extension to this work could look into applying different regularisation techniques to the problem. For example, l2-cost was tested early on but was found to be very difficult to fit properly to the model. Since it was difficult to detect if the models were teachable at all, I decided to focus on the methods that were simpler to use.

Chapter 4

Results

We attempt to answer three main questions through our experiments. What feature representation will be most effective for this problem? What is the effect of dropout on the different features? And, can LSTM networks perform on par with previous state of the art results on this task?

We ran similar experiments for each feature set described in chapter 3 attempting to find the best hyper parameter settings for our network. This setting was found using a grid-search method over the number of hidden layers and the number of hidden units. Initially the same values were used for each feature type. If the performance seemed to be going up with the number of units towards the outside bounds of the search then additional experiments were run to see if a better model could be found outside the grid size.

Initial experiments were run without dropout. It was clear that the networks were overfitting and thus the dropout was applied. Dropout was set to 0.5 but other values were also considered. Additionally, experiments were run where the dropout rate was initially set to 0.5 but was slowly decreased until it was set to 0 in the final training epoch.

4.1 Comparison of Feature Sets

To begin we study the relative success of the different features described in section **??**. The best model for each feature set was found using a grid search where the number of hidden units and layers were searched over. Two separate grid search experiments were run for each feature set, one using dropout and one without. It was clear that adding dropout to the best model did not guarantee to create the best possible model (see section 3.4). The most successful models can be seen in table 4.1.

The results indicate that the low level mfcc features do a lot better than the higher level features. A common advantage of neural networks is that lower level features can be used. The many layers of networks allows the network to create its own feature higher level features by combining and weighting the lower level features. A clear example

Features	Layers	Units	Dropout	Accuracy
mfcc	5	50	0.5	**66.39**
phones	5	200	0.5	63.98
phones	1	100	0	63.98
gemaps	1	400	0	62.37
baseline	-	-	-	61.83

Table 4.1: Best results for each feature set. The dropout rate shown is the probability of dropping a connection. Dropout 0 thus means no dropout. The baseline is a simple majority class classifier classifying everything to "truth". Accuracy results are given on the validation set.

of this is machine vision where it is common to give raw pixel values as input. The layers then learn to recognise different higher level features, starting out with lines, then simple shapes, and finally more complicated shapes and objects.

4.1.1 MFCC

We might imagine that this is what is happening in the deep mfcc network. The features contain a lot of low level information that can be leveraged by the architecture to make accurate predictions. Since we are giving it low level information the network can learn its own features that are optimised for our deception detection task. If we design our own features then we might miss some important factor that is picked up by our network but that we have not considered.

For future work, there are a number of other low level features that can be used. A common choice for Neural models for Automatic Speech Recognition is filterbank features. These are based on the MFCC features but the final step of the feature extraction is not performed, the inverse discrete fourier transform (iDFT). Normally the iDFT decorrelates the features, which is very useful for GMM speech recognition, but neural networks have no problem with correlated features, thus retaining information that is lost when doing the iDFT is useful.

4.1.2 Phones

We do still get above chance performance with both other features sets, although gemaps only barely. The fact that the phone features did so well is both surprising and not so. We saw that phone features were used to great effect by Levitan et al. (2016); Herms (2016); Gosztolya et al. (2016). The approach used by Levitan et al. (2016) can be somewhat likened to what we have done. They trained a phone-language model predicting the next phone given the previous phone. These models can be used estimate a score to see how likely it is that the sequence was deceptive or non-deceptive. Neural Language models have used LSTMs to predict the next word given previous words (Mikolov et al., 2010). The models they train are seq2seq models, but are other-

wise quite similar to those I have trained. Levitan et al. (2016) use a more traditional language model, but the theory is similar.

The surprise of a good score comes mainly from the fact that the phones are for the Hungarian language. Since the source material is English we might not expect that the Hungarian phones will give meaningful results from phone labels. What can be said is that any phone set would just be a collection of labels. They will have no pre-existing meaning for the classifier. Thus, which set of labels we use will make little difference as long as they capture similar information about the underlying speech. Since both look for the sounds being made it is not unreasonable to expect the results to be somewhat related which could explain the success of the feature set.

4.1.3 GEMAPS

The GEMAPS features are designed to give a decent baseline for a broad range of paralingistic tasks. This is a bold statement and it is not unlikely that what works well on other problems is not well suited for our deception detection. The best results using these features are barely above chance, only about .5%. This is not a significant increase which seems to indicate that these features extracted on the phone level do not work.

The GEMAPS dataset contains more prosodic information than the MFCCs. The set also includes MFCC bands 1-4. Since prosodic features have been effective in other approaches to deception detection it is surprising that adding this information should do much worse. It is quite likely that the size of time-steps is the culprit here. Since we are taking very big steps we might miss the small changes. The psychology of deception seemed to show that the small things would be very important, that micro-expressions might appear in the speech. To test this theory future work could include extracting GEMAPS features on a lower level, such as directly from the windowed speech.

4.2 Choice of Model

Here we make a comparison between the previously published models and compare their predictive power. In table 4.2 the best published acoustic/prosodic model published for this data, together with the text-based model from Appelgren (2016) are presented. The best model is a combination of two models, an accoustic GMM and a prosodic SVM (Graciarena et al., 2006). The acoustic model trains GMMs on frames of input speech while the prosodic SVM uses features such as pitch and duration. More recent work on deception detection by Levitan et al. (2015) have used a new dataset, so their results are not relevant for comparison.

The MFCC model has the highest score out of all the models. It thus seems that there is enough information in the MFCCs and that the model is powerful enough to be able to classify successfully. It should perhaps be pointed out that the MFCCs give

Classifier	Features	Result
Acoustic GMM (A)	A	63.8
Prosodic SVM (B)	P	64.4
A and B	AP	65.7
RF+FS*	L	65.9
MFCC*	AP	66.4

Table 4.2: Key: systems marked with * are developed by me. The other systems are developed by Graciarena et al. (2006) and represent the best acoustic/prosodic models found. The only other, better system was a subject dependent system published by Hirschberg et al. (2005). RF+FS is the feature selected random forest developed by Appelgren (2016) and MFCC is the best network trained by me this year.

mostly acoustic information. We have not added any prosodic features to this, which we can see have been even more successful in the Prosodic SVM (B) system. Thus, if we compare the result against the Acoustic GMM (A) the improvement is even more significant. As LSTMs have outperformed GMM models in speech recognition it is not too surprising that they should also outperform the GMMs here.

4.2.1 Discussion

Several possible extensions exist. Three obvious paths seem to be open. The first is to extend the existing LSTM network, the second to combine the current networks, and the third to explore the use of different neural network models. An effective extension to the current model would be making it bidirectional. This techniques has proven itself several times and would be an obvious next step. The second improvement would be to combine the available networks. This can be done by using the outputs of each of the trained networks as input to a new network. That is, use the probabilities that a sequence is truth or lie as input to a simple network that weights the importance of each networks output, creating a weighted average voting system. Finally, the background mentioned several possible neural architectures that can be used for sentence classification. Experimenting with these architectures to find if any function particularily well for this problem would make sense. The CNN approaches especially are a good next step. Since these can easily be adopted for both speech and text they could easily be incorporated for this problem.

Chapter 5

Conclusions

We have shown that LSTM networks can be trained to perform deception detection. The results gained were on part with previous state-of-the-art systems. Out of the three feature sets tested, MFCCs extracted from windowed speech were found to be the most successful, scoring 66.4% accuracy which is on par with previous systems using acoustic and prosodic features, at 65.7% accuracy, and the system I trained last year at, 65.9% accuracy. At 63.98% accuracy the phone levels performed well above the baseline of 61.8% while the GEMAPS features extracted at the phone level performed badly at only 62.3%. Since the GEMAPS features did not work there is more room to find ways of integrating more prosodic information into the MFCC network by extracting prosodic features from the windowed speech.

Further work could go into improving the networks used and to combine the outputs of each network through a voting procedure to leverage different information available in each set of features. Additionally, several other neural models and architectural improvements could be explored.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P. A., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zhang, X. (2016). Tensorflow: A system for large-scale machine learning. *CoRR*, abs/1605.08695.

Abdel-Hamid, O., Mohamed, A., Jiang, H., and Penn, G. (2012). Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*, pages 4277–4280.

Amiriparian, S., Pohjalainen, J., Marchi, E., Pugachevskiy, S., and Schuller, B. (2016). Is deception emotional? an emotion-driven predictive approach. *Interspeech 2016*, pages 2011–2015.

Appelgren, M. (2016). Detecting deception using natural language. Master's thesis, University of Edinburgh.

Benus, S., Enos, F., Hirschberg, J. B., and Shriberg, E. (2006). Pauses in deceptive speech. Columbia University Academic Commons. http://hdl.handle.net/10022/AC:P:20856.

Caruana, R., Lawrence, S., and Giles, C. L. (2000). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 402–408.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2067–2075.

Darken, C. and Moody, J. E. (1990). Note on learning rate schedules for stochastic optimization. In *Advances in Neural Information Processing Systems 3, [NIPS Conference, Denver, Colorado, USA, November 26-29, 1990]*, pages 832–838.

DePaulo, B. M., Lindsay, J. J., Malone, B. E., Muhlenbruck, L., Charlton, K., and Cooper, H. (2003). Cues to deception. *Psychological bulletin*, 129(1):74.

Ekman, P. (1985). *Telling lies: Clues to deceit in the marketplace, marriage, and politics*. New York: Norton.

Ekman, P. and Friesen, W. V. (1969). Nonverbal leakage and clues to deception. *Psychiatry*, 32(1):88–106.

Eyben, F., Scherer, K. R., Schuller, B. W., Sundberg, J., André, E., Busso, C., Devillers, L. Y., Epps, J., Laukka, P., Narayanan, S. S., and Truong, K. P. (2016). The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing. *IEEE Trans. Affective Computing*, 7(2):190–202.

Eyben, F., Weninger, F., Groß, F., and Schuller, B. W. (2013). Recent developments in opensmile, the munich open-source multimedia feature extractor. In *ACM Multimedia Conference, MM '13, Barcelona, Spain, October 21-25, 2013*, pages 835–838.

Gosztolya, G., Grósz, T., Busa-Fekete, R., and Tóth, L. (2016). Determining native language and deception using phonetic features and classifier combination. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2418–2422.

Graciarena, M., Shriberg, E., Stolcke, A., Enos, F., Hirschberg, J., and Kajarekar, S. S. (2006). Combining prosodic lexical and cepstral systems for deceptive speech detection. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP 2006, Toulouse, France, May 14-19, 2006*, pages 1033–1036.

Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649.

Herms, R. (2016). Prediction of deception and sincerity from speech using automatic phone recognition-based features. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2036–2040.

Hirschberg, J., Benus, S., Brenier, J. M., Enos, F., Friedman, S., Gilman, S., Girand, C., Graciarena, M., Kathol, A., Michaelis, L., Pellom, B. L., Shriberg, E., and Stolcke, A. (2005). Distinguishing deceptive from non-deceptive speech. In *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*, pages 1833–1836.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 655–665.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Levitan, S. I., An, G., Ma, M., Levitan, R., Rosenberg, A., and Hirschberg, J. (2016). Combining acoustic-prosodic, lexical, and phonotactic features for automatic deception detection. In *Interspeech 2016, 17th Annual Conference of the Interna-*

tional Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016, pages 2006–2010.

Levitan, S. I., An, G., Wang, M., Mendels, G., Hirschberg, J., Levine, M., and Rosenberg, A. (2015). Cross-cultural production and detection of deception from speech. In *Proceedings of the 2015 ACM Workshop on Multimodal Deception Detection, WMDD@ICMI 2015, Seattle, Washington, USA, November 13, 2015*, pages 1–8.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

Osadchy, M., LeCun, Y., and Miller, M. L. (2007). Synergistic face detection and pose estimation with energy-based models. *Journal of Machine Learning Research*, 8:1197–1215.

Pennebaker, J. W., Francis, M. E., and Booth, R. J. (2001). Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71:2001.

Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *14th International Conference on Frontiers in Handwriting Recognition, ICFHR 2014, Crete, Greece, September 1-4, 2014*, pages 285–290.

Plank, B., Søgaard, A., and Goldberg, Y. (2016). Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*.

Rabiner, L. R. and Juang, B. (1993). *Fundamentals of speech recognition*. Prentice Hall signal processing series. Prentice Hall.

Sainath, T. N., Weiss, R. J., Senior, A. W., Wilson, K. W., and Vinyals, O. (2015). Learning the speech front-end with raw waveform cldnns. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 1–5.

Sak, H., Senior, A. W., Rao, K., Irsoy, O., Graves, A., Beaufays, F., and Schalkwyk, J. (2015). Learning acoustic frame labeling for speech recognition with recurrent neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 4280–4284.

Schuller, B. W., Steidl, S., Batliner, A., Hirschberg, J., Burgoon, J. K., Baird, A., Elkins, A. C., Zhang, Y., Coutinho, E., and Evanini, K. (2016). The INTERSPEECH 2016 computational paralinguistics challenge: Deception, sincerity & native language. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2001–2005.

Schwarz, P., Matejka, P., and Cernocký, J. (2006). Hierarchical structures of neural networks for phoneme recognition. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP 2006, Toulouse, France, May 14-19, 2006*, pages 325–328.

Sermanet, P., Chintala, S., and LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st International Conference on Pattern Recognition, ICPR 2012, Tsukuba, Japan, November 11-15, 2012*, pages 3288–3291.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 151–161.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM neural networks for language modeling. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pages 194–197.

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1556–1566.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2).

to Review the Scientific Evidence on the Polygraph (National Research Council (US)), C., on Behavioral, N. R. C. U. B., Sciences, S., on National Statistics, N. R. C. U. C., of Behavioral, N. R. C. U. D., and Sciences, S. (2003). *The polygraph and lie detection*. Haworth Press.

Tompson, J., Goroshin, R., Jain, A., LeCun, Y., and Bregler, C. (2015). Efficient object localization using convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 648–656.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Zuckerman, M., DePaulo, B. M., and Rosenthal, R. (1981). Verbal and nonverbal communication of deception. *Advances in experimental social psychology*, 14(1):59.