Tampere University

Matti Tuhola

ENGLISH LEXICAL STRESS RECOGNITION USING RECURRENT NEURAL NETWORKS

Faculty of Information Technology and Communication Sciences
Master of Science Thesis
September 2019

ABSTRACT

Matti Tuhola: English Lexical Stress Recognition Using Recurrent Neural Networks
Master of Science Thesis
Tampere University
Degree Programme in Information Technology
September 2019

Lexical stress is an integral part of English pronunciation. The command of lexical stress has an effect on the perceived fluency of the speaker. Moreover, it serves as a cue to recognize words. Methods that can automatically recognize lexical stress in spoken audio can be used to help English learners improve their pronunciation.

This thesis evaluated lexical stress recognition methods based on recurrent neural networks. The purpose was to compare two sets of features: a set of prosodic features making use of existing speech recognition technologies, and simple spectral features. Using the latter feature set would allow for an end-to-end model, significantly simplifying the overall process. The problem was formulated as one of locating the primary stress, the most prominently stressed syllable in the word, in an isolated word.

Datasets of both native and non-native speech were used in the experiments. The results show that models using the prosodic features outperform models using the spectral features. The difference between the two was particularly stark on the non-native dataset. It is possible that the datasets were too small to enable training end-to-end models. There was a considerable variation in performance among different words. It was also observed that the presence of a secondary stress made it more difficult to detect the primary stress.

Keywords: lexical stress recognition, computer-assisted pronunciation training, prosodic features, recurrent neural networks

TIIVISTELMÄ

Matti Tuhola: Englannin kielen sanapainon tunnistus takaisinkytkeytyvien neuroverkkojen avulla
Diplomityö
Tampereen yliopisto
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Syyskuu 2019

Sanapaino on olennainen osa englannin kielen ääntämistä. Sen osaaminen vaikuttaa puhujan havaittuun sujuvuuteen, ja se toimii vihjeenä sanojen tunnistamiselle. Menetelmiä, joilla sanapaino voidaan automaattisesti tunnistaa puheesta, voidaan käyttää apuna englannin oppijoiden ääntämisen parantamisessa.

Tämä diplomityö arvioi takaisinkytkeytyviin neuroverkkoihin perustuvia menetelmiä sanapainon tunnistukseen. Tarkoitus oli vertailla kahdenlaisia piirteitä: joukkoa prosodisia piirteitä, jotka hyödyntävät olemassa olevia puheentunnistusteknologioita, ja yksinkertaisia äänen spektriin perustuvia piirteitä. Jälkimmäisten piirteiden käyttö mahdollistaisi päästä-päähän -mallien käyttämisen, mikä yksinkertaistaisi kokonaisprosessia merkittävästi. Ongelma esitettiin muodossa, jossa tarkoitus oli löytää pääpainon sijainti, eli sanan voimakkaiten erottuva tavu, yksittäisestä sanasta.

Tutkimuksessa käytettiin dataa sekä englantia äidinkielenään että ei-äidinkielenään puhuvilta. Tulosten mukaan prosodisia piirteitä käyttävät mallit suoriutuvat tehtävästä paremmin kuin äänen spektriin perustuvia piirteitä käyttävät mallit. Erot olivat erityisen suuria datajoukossa, joka koostui englantia ei-äidinkielenään puhuvien puheesta. On mahdollista, että käytetyt datajoukot olivat liian pieniä päästä-päähän -mallien opettamista varten. Mallien suorituskyvyssä oli huomattavaa vaihtelua eri sanojen välillä. Tutkimuksessa havaittiin myös, että sivupainon läsnäolo vaikeutti pääpainon tunnistamista.

Avainsanat: sanapainon tunnistus, tietokoneavusteinen ääntämisen opetus, prosodiset piirteet, takaisinkytkeytyvät neuroverkot

PREFACE

I would like to thank my supervisors Tuomas Heikkinen and Guangpu Huang for their guidance over the course of this thesis. In addition, I would also like to thank Timo-Pekka Leinonen for his assistance and helpful comments. I would also like to express my gratitude to my colleagues for their help with various aspects of this work. Finally, I would also like to thank my friends and family for their support.

Tampere, 30th September 2019

Matti Tuhola

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

Adam	Adaptive Moment Estimation
ASR	Automatic Speech Recognition
BGD	Batch Gradient Descent
BRNN	Bidirectional Recurrent Neural Network
CAPT	Computer-Assisted Pronunciation Training
FCNN	Fully Connected Neural Network
FFNN	Feedforward Neural Network
GRU	Gated Recurrent Unit
LSR	Lexical Stress Recognition
LSTM	Long Short-Term Memory
MBGD	Mini-Batch Gradient Descent
MFCC	Mel-Frequency Cepstral Coefficients
NN	Neural Network
ReLU	Rectified Linear Unit
RMS	Root Mean Square
RNN	Recurrent Neural Network
SD	Standard Deviation
SGD	Stochastic Gradient Descent

1 INTRODUCTION

Learning pronunciation is an essential aspect of learning English. Speaking with a strong non-native accent can require listeners to make a significant effort to understand the speaker or, in the worst case, render their speech unintelligible.

Non-native accents are often attributed to having difficulties producing English phonemes. Just as important, however, is speaking English with the correct prosodic properties, such as lexical stress, rhythm, and intonation. Some researchers have even suggested that prosody might play a larger role than phonetics [55, pp. 397 – 409].

Pronunciation is known to be notoriously difficult to teach. Little time in classrooms can be afforded to be spent on pronunciation, especially correcting the errors of individual students. One-to-one instruction can be effective but not accessible for many students. Computer-assisted pronunciation training (CAPT) systems aim to alleviate this problem by emulating some aspects of one-to-one instruction, such as providing immediate feedback, while being scalable to large groups of learners.

In recent years there have been great advances in spoken language technology, particularly in speech recognition (e.g. [2]) and text-to-speech systems (e.g. [62]). Much of this can be attributed to deep learning, a set of neural network technologies that make use of the abundance of computation and data resources that are available today [31]. These advances have also lead to rapid development in CAPT systems [8] and some lexical stress recognition systems making use of them have been proposed (e.g. [42, 60]). While neural networks have been applied to the problem of lexical stress recognition before, many of the more recent advances have seen limited application in this area.

In this thesis, various approaches using recurrent neural networks are employed to detect the lexical stress in spoken audio. The main goal is to investigate whether modern end-to-end neural network architectures can be an improvement over the traditional approaches, which often depend on a separate speech recognition system to locate the syllables in the audio. To evaluate the performance of these two approaches, the systems are trained and tested on two datasets with data from native and non-native speakers.

This thesis is organized as follows. Chapter 2 describes the problem of lexical stress recognition, along with background information on English lexical stress and its acoustic correlates. Further background information is provided in Chapter 3, which provides a general description of neural networks, focusing on simple fully connected neural networks and recurrent neural networks. Chapter 4 describes the methodology, including

an overview of the system, a description of the data preprocessing and feature extraction procedures, and the neural network models used to conduct the experiments. Chapter 5 presents the datasets and the experiments conducted for this thesis, along with their results. Finally, Chapter 6 summarizes the results and provides an outlook for future research.

2 ENGLISH LEXICAL STRESS

English is a stress-timed language. This means that syllables in a word are perceived to vary in features such as duration, intensity, and pitch, which results in some syllables appearing more prominent than others. This relative emphasis is known as *lexical stress*, or word stress. Each English word with more than one syllable has a *primary stress* corresponding to the most prominent syllable in the word. Long words may also have one or more secondary stresses, i.e., syllables that are more prominent than the unstressed ones, but not the most prominent syllable in the word.

Aspects of pronunciation are commonly divided into segmental features, i.e., phonemes, and suprasegmental features, i.e., features that extend over syllables or phrases. Lexical stress is a suprasegmental feature. Other suprasegmental features include rhythm, intonation, and pitch accent [41]. Collectively, they constitute *prosody*. Among suprasegmental features, lexical stress is the only word-level phenomenon — the other features occur at the phrase level. Lexical stress is an intrinsic part of the pronunciation of English words that is largely independent of the context [55, p. 106].

The phrase-level suprasegmentals can be used to express emphasis and emotion, and to clarify the speakers tone. Rhythm refers to the regular timing patterns apparent in speech. Intonation is phrase-level pitch variation with many functions such as distinguishing between questions and statements. Pitch accent can be defined as tonal prominence of words that is distinct from the intonation pattern [23]. It provides semantic information such as focus [24]. It is worth noting that pitch accent can also refer to lexical pitch accent, which serves a similar function as lexical stress in some languages.

It is widely agreed that suprasegmental features play an important role in intelligibility and foreign accent [29, 36, 49]. Lexical stress, in particular, serves as a cue to recognize words within sentences and to disambiguate between similar sounding words. Native English speakers expect to hear certain stress patterns and may find it difficult to understand someone who otherwise correctly pronounces the syllables but fails to place the stress on the correct syllables. Moreover, lexical stress can in some cases mark the difference between two phonetically similar words with different meanings, such as *'insight* and *in'cite*, where *'* is used to mark the stressed syllable [55, pp. 109 – 113].

Whether segmental or suprasegmental features are more important in teaching pronunciation has been the subject of much debate. According to Reed and Levis [55, pp. 399–409], the consensus in the literature is that pronunciation instruction is moving

towards a more balanced view where both segmental and suprasegmental features are seen as important. They suggest that the distinction between segmental and suprasegmental features may not be as clear as previously thought, and that these features have to be seen "as part of an integrated and interactive system where the production of one can influence the other."

2.1 Lexical Stress Recognition

The goal of lexical stress recognition (LSR) is to identify the stressed syllables in speech. An LSR system takes features calculated from an audio recording as its input and, for each syllable in the audio, produces an output denoting the stressedness of that syllable. The transcription of what is being said is typically known in advance. LSR is usually considered a supervised learning problem where the model is trained using labeled training data.

There are generally considered to be three levels of lexical stress, namely primary stress, secondary stress, and no stress. The problem can be simplified to binary classification, either recognizing whether a syllable carries primary stress or not, or whether a syllable carries any stress (either primary or secondary) or not. As the difference between primary and secondary stress can be very subtle, difficult for even humans to recognize [45], and for many purposes a binary result is sufficient, the simplification is justifiable.

Like all supervised learning tasks, LSR requires an effective representation of the input that captures the properties that carry information about the problem at hand. Syllable-wise features based on the acoustic correlates of lexical stress are commonly used. They are calculated from the full syllables or the syllable nuclei, which are segmented from the full audio recording using methods such as forced alignment. The acoustic correlates of lexical stress are discussed in more detail in Section 2.2. In other closely related fields, such as automatic speech recognition (ASR), spectral features including Mel spectrograms and Mel-frequency cepstral coefficients (MFCCs) are commonly used (e.g. [4, 9, 38]). In LSR, they have only seen limited use (e.g. [16, 60]).

There is a large body of research on English lexical stress recognition, varying widely in both motivation and methodology. Many early LSR systems were developed to improve ASR systems. This was motivated by the suggestion that stressed syllables carry more robust phonetic information than unstressed syllables, and thus locating them could reduce the search space of possible words [3]. Various models, including Bayesian classifiers [69, 73], hidden Markov models [17], and neural networks [33] have been employed for this purpose. These systems typically work at the syllable level without the context of the word or the surrounding syllables, taking features calculated from the syllable as their input and predicting whether that syllable was stressed or unstressed.

As ASR systems have improved, the focus of research on LSR has shifted to other problem domains, including computer-assisted pronunciation training and speech therapy. LSR systems developed for CAPT have different requirements than those developed for

ASR. CAPT systems need to work on non-native data where the phonetic pronunciation may not be correct. In addition, they need to consider the full word and not just try to predict the stressedness of one isolated syllable. It is common to formulate the problem as one of locating the syllable carrying the primary stress among the syllables in a word [14, 66, 72]. Classifying the overall stress pattern as being correct or incorrect has also been proposed [68]. It has been suggested that neither of these is sufficient for CAPT in all cases as non-natives can stress multiple syllables with equal prominence [16]. The models used in the existing research vary widely, and include Gaussian mixture models [7, 16], neural networks [42, 60, 61], and support vector machines [75]. Unsupervised methods have also been attempted with modest results [15].

Supervised learning requires data that has been labeled with the ground truth. For an LSR system, this means labels denoting the stressedness of each syllable. A natural approach for obtaining the labels is to have the data be annotated by language teachers or other experts. This, however, is a resource-intensive and a challenging problem in and of itself, as distinguishing between stressed and unstressed syllables can be difficult even for experts. The agreement level between two experts who independently annotate which syllable in an audio recording of a word has the primary stress is typically in the range of 80 % to 90 % [16, 33, 44]. When trying to distinguish syllables with primary stress, secondary stress, or no stress as three separate categories, the inter-annotator agreement can be significantly lower [43]. Using automatically generated labels can, in some cases, be an alternative to annotation. For example, if data from native speakers is used to train an LSR system, it can be assumed that their stress patterns are correct, and a pronouncing dictionary can be used to produce the labels automatically. This approach is not possible with non-native speakers, where lexical stress errors are expected.

2.2 The Acoustic Correlates of Lexical Stress

To develop a system for recognizing lexical stress, it is essential to understand which properties of speech distinguish stressed and unstressed syllables. These properties are known as the acoustic correlates of lexical stress.

In a seminal study by Fry [18], one of the earliest studies on this area, it was suggested that lexical stress is perceived as a variation in four psychological qualities apparent in vowels, namely length, loudness, pitch, and quality. These qualities are said to have their corresponding physical features, namely vowel duration, intensity, fundamental frequency (f_0), and the formant structure of the sound waves. The study investigated the effect of the four physical features on perceived lexical stress by varying them in synthesized speech. The results indicated that duration and intensity act as cues to lexical stress, duration producing the greater overall effect. In addition, the result showed that the direction of change of f_0, had a significant effect on stress perception, whereas the magnitude of the change did not appear to be important.

Many similar acoustic studies have been conducted, largely agreeing with Fry's results.

Lieberman [44] studied the acoustic correlates of lexical stress for noun-verb pairs that are primarily differentiated by their stress patterns. The words were recorded by native American English speakers and the stress patterns were annotated by two observers. It was found that a higher fundamental frequency, a greater peak envelope amplitude, and a longer duration were correlated with stressed syllables, with the first two being the most important features. In [48], listeners were asked to judge stress on synthesized non-words consisting of nonsense syllables (e.g. *"sisi"* and *"sasa"*), where f_0, duration, and intensity were varied. All three features were found to be correlates of stress with fundamental frequency being the most important.

A notable shortcoming of these early studies is that they do not consider the effect of phrase-level prominence on the acoustic features of the stressed syllables. If the word of interest is in a focal position in the phrase, it is likely to have a pitch accent on the stressed syllable [54, 64]. Unlike lexical stress, pitch accent is not a structural, linguistic property of the word but a result of the word's position in the phrase.

In studies by Sluijter and van Heuven [63, 64] the difference between lexical stress and pitch accent was controlled for. A major finding was that f_0 may not be a correlate of lexical stress in isolation. Instead, stressed syllables may differ in f_0 by virtue of them having a pitch accent caused by the word being in a focal position in the phrase. The most reliable correlate for lexical stress was found to be duration. Spectral tilt, a measure of the distribution of energy between low and high frequencies, was found to be another reliable correlate. Several different methods for calculating spectral tilt have been suggested [35].

Okobi [51] studied the effect of a wide array of acoustic features, trying to disentangle the phenomena of lexical stress and pitch accent. The results were similar to those obtained by Sluijter and van Heuven. It was found that spectral tilt, noise at high frequencies, and syllable duration were the most important features for primary stress independent of the presence of a pitch accent. Intensity, f_0, and the amplitude of the first harmonic were found to be correlated with stressed syllables only in accented positions, i.e., when a pitch accent was present.

The distinction between primary and secondary stress is a less studied area of research. Plag et al. [54] investigated this distinction in both accented and unaccented positions in American English. It was found that spectral tilt serves as a correlate for the distinction between primary and secondary stress in both accented and unaccented words. Intensity, and f_0 were found to be strong correlates in accented left-prominent words, i.e., words where the primary stress is located on a syllable before the syllable bearing the secondary stress (e.g. *il'lumi,nate*, where *'* and *,* mark the syllables carrying the primary and secondary stresses, respectively). The correlation is weaker in accented right-prominent words (e.g. *il,lumi'nation*), where the order of the syllables carrying primary and secondary stresses is reversed. It was suggested that the reason for this may be right-prominent words having an additional pitch accent on the unstressed syllable. In unaccented words, intensity and f_0 are only weak correlates. Duration and pitch slope, i.e., the slope of a line drawn between the maximum and minimum f_0 values in a syllable,

were not found to differ between syllables with primary and secondary stress.

The results from these studies indicate that there are many potential acoustic correlates of lexical stress, including intensity, duration, spectral tilt, fundamental frequency, and pitch slope. Duration and spectral tilt are among the most reliable correlates. Some of these features, particularly intensity and fundamental frequency, are only correlated with stressed syllables when the word appears in an accented position.

3 NEURAL NETWORKS

Neural networks (NNs), or artificial neural networks, are a set of computational models that are commonly used in machine learning. They have been applied to a multitude of tasks in various fields, such as speech recognition, computer vision, and machine translation. NNs are most commonly used for supervised learning tasks, namely classification and regression, but they can be used for reinforcement learning and unsupervised learning tasks as well. In this chapter neural networks are be considered exclusively in the context of classification.

The primary inspiration for developing neural network models was originally modeling biological neural systems. Early models from the 1940s and 1950s, such as the neural model by McCulloch and Pitts [46] and Rosenblatt's perceptron [56], were developed with this goal in mind. In the following decades, the focus shifted from accurately modeling the brain to creating computer systems capable of learning in order to solve a variety of practical problems. The field of neural networks advanced with developments such as the backpropagation algorithm [58, 71], recurrent neural networks [34], and convolutional neural networks [19, 40]. However, it is only in the past decade that the data and computation resources required for large-scale neural network systems have become widely available, and that NNs have started to significantly outperform other machine learning models [31].

Following the increased interest in neural networks prompted by the recent advances, the field has come to be known as *deep learning*, named after the multiple levels of hierarchy, or depth, used in modern neural network architectures. Deep learning is an engineering discipline, which only loosely draws inspiration from neuroscience [22, pp. 13–16]. The divergence of these two fields is understandable, as the goal of deep learning is to find efficient, well-generalizing learners, whereas the use of NNs in neuroscience is motivated by understanding the principles of brain function [21].

3.1 Neural Networks as Classifiers

Supervised learning is the task of learning a function that maps an input variable $\boldsymbol{x}$ to an output variable $\boldsymbol{y}$ based on a training set comprising example input-output pairs $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \langle \boldsymbol{X}, \boldsymbol{Y} \rangle$. The input observation is known as the *sample*, and the corresponding output observation is known as the *label*. *Classification* is the subcategory of supervised learning where the output variable is discrete, only taking a finite set of values known

as *classes*. Examples of classification include recognizing handwritten digits, deciding whether or not an email is spam, and predicting whether a medical image contains evidence of cancer.

Neural network based classifiers approximate a function from the input features to the labels $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the network's parameters. Being an approximation, the output produced by the classifier is not the label $\boldsymbol{y}$ but a prediction $\hat{\boldsymbol{y}}$. The process of producing a prediction $\hat{\boldsymbol{y}}$ for a given input is known as *inference*. The prediction is calculated through the *forward pass* of a neural network. Section 3.4 describes the process in more detail.

Training a neural network means finding the parameters that best map the samples in the training set to the corresponding labels. The training process aims to minimize the difference between the predictions produced by the model and the labels. This happens iteratively by changing the network's parameters through a process called *gradient descent*. The difference between the model's predictions and the labels is called the *loss*, and it is measured using a *loss function*. To determine how the parameters should be changed in order to minimize the difference, the *gradient* of the loss function is calculated with respect to the parameters. This is done through the *backward pass* of the neural network. The parameters are adjusted based on the gradient, and the process is repeated, until the model converges. The forward and backward passes are described in Section 3.4. Gradient descent and the training process are discussed in more detail in Section 3.5.

It is not enough for a classifier to make accurate predictions on the samples in the training set. It should also be able *generalize*, i.e., to make accurate predictions on new, unseen data. A model that works well on the training data but performs poorly on unseen data is said to have *overfit* the training data. Generalization and ways to avoid ovefitting are discussed in Sections 3.5.2 and 3.5.3.

3.2 Neurons

The neuron, or the artificial neuron, is the elementary unit of calculation in a neural network. Artificial neurons are modeled after a coarse representation of their biological counterparts [37]. In this representation, a neuron receives and sends electrical signals via its connections to other neurons. The incoming signals can inhibit or excite the neuron, which affects the rate at which it sends out signals. This behavior is influenced by both the strength of the incoming signals and the strength of the connections. An excited neuron can become active and fire, sending out signals at an increased rate in a spike of energy.

Similarly, an artificial neuron receives one or more inputs and — based on the inputs and the neuron's parameters — produces a single value known as the *activation* as its output. Neurons have two kinds of parameters, namely weights and biases. Analogously to the coarse biological model, the weights affect how strong the connections between

the neurons are, and whether the connections are excitatory or inhibitory, i.e., positive or negative.

Another way to think about the inputs and the weights is to consider the neuron to be a system that is trying to detect a particular pattern in the inputs. With this framework, the weights are a representation of this pattern and the inputs serve as evidence of the pattern. If a lot of evidence for this pattern exists in the input, the neuron is more likely to become active.

The bias b is a value that is internal to the neuron and separate from the weights. It regulates how easy it is for the neuron to become active. A high bias value means that the neuron can become active, even if there is only a low amount of evidence for the pattern it is trying to detect. A low bias value has the opposite effect.

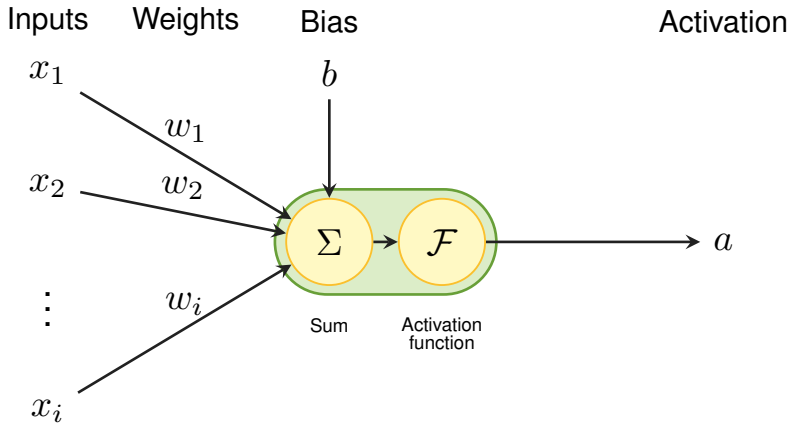


Figure 3.1. *The structure of a neuron.*

Figure 3.1 illustrates the structure of a neuron. The input to the neuron is a vector $\boldsymbol{x}$, where each element x_i is an activation received from preceding neurons. The weights are stored in a weight vector $\boldsymbol{w}$ with the same length as the input vector. The neuron calculates a weighted sum of the inputs and adds the bias to produce the *pre-activation output* of the neuron. It is given by

$$z = \sum_{i=1}^{I} (w_i x_i) + b. \tag{3.1}$$

To produce the activation of the neuron, the pre-activation output is passed through a non-linear *activation function* $\mathcal{F}(z)$. The activation is given by

$$a = \mathcal{F}(z) = \mathcal{F}\left(\sum_{i=1}^{I} (w_i x_i) + b \right). \tag{3.2}$$

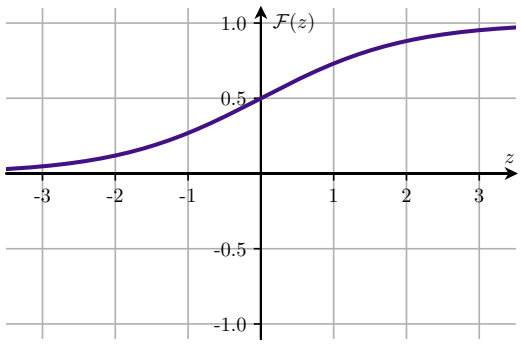
This activation value is the output of the neuron, used as an input by the subsequent neurons. The activation function decides how active the neuron should become based

on the pre-activation output. In the analogy to the coarse biological neural model, the activation function models the firing rate of the neuron.

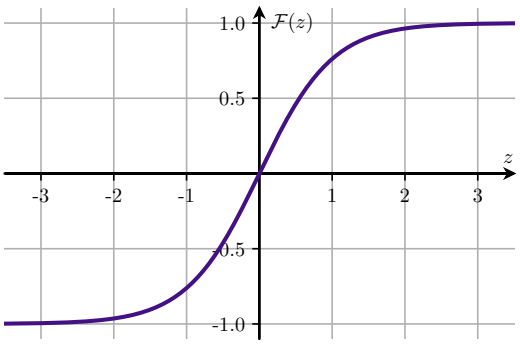
3.3 Activation Functions

The activation function serves an important purpose in enabling the neural network to represent complex, non-linear mappings. In fact, an arbitrarily large neural network with linear activations will always be equivalent to a single neuron with a linear activation, since any linear combination of linear operators is a linear operator.

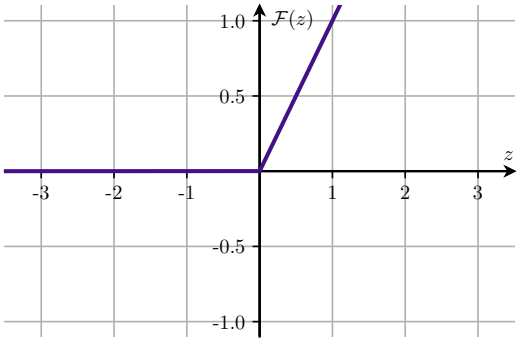
In addition to non-linearity, there are some other properties that are required of the activation function, or otherwise desirable. First, it is preferrable that the activation function be differentiable in order to enable the use of gradient-based methods on training the neural network. Secondly, it is helpful for the activation function to be monotonic. Thirdly, activation functions have the additional purpose of making sure that the neuron output is in a particular range, such as $[-1, 1]$ or $[0, \infty)$. Finally, as the activation has to be frequently calculated for all neurons in the network, it should be efficient to calculate.



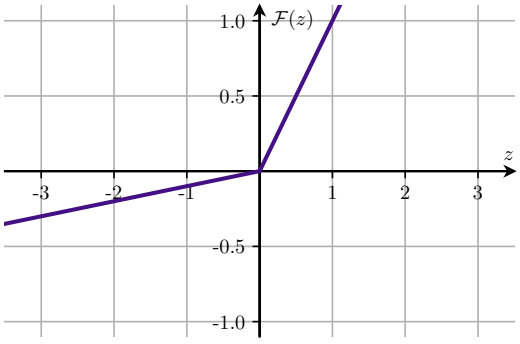
(a) *Logistic function*



(b) *Hyperbolic tangent*



(c) *Rectified linear unit (ReLU)*



(d) *Leaky ReLU*

Figure 3.2. *Activation functions.*

Figure 3.2 shows the plots of common activation functions. The figure shows two classes of functions, namely sigmoid functions characterized by their S-shape and piecewise linear functions.

The sigmoid functions include the logistic function

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{3.3}$$

and the hyperbolic tangent

$$\tanh(z) = \frac{1 - \mathrm{e}^{-2z}}{1 + \mathrm{e}^{-2z}}. \tag{3.4}$$

The main difference between the two functions is the output value ranges, which are $[0, 1]$ for the logistic function and $[-1, 1]$ for the hyperbolic tangent. There exists a relationship between the functions

$$\tanh(z) = 2\sigma(2z) - 1 \tag{3.5}$$

to calculate the output of one from the other. As such, the two functions are largely equivalent outside of the output ranges [25, p. 14].

The sigmoid functions have been found to work poorly in deep neural networks, and piecewise linear functions have been suggested as an alternative that achieves better performance [21]. They include the rectified linear unit (ReLU)

$$\mathrm{ReLU}(z) = \max(0, z)$$

and its variant LeakyReLU

$$\mathrm{LeakyReLU}(z) = \max(0, z) + \min(0, \alpha z),$$

where α is a small positive constant.

One reason that the ReLU activation works better than the sigmoid functions, suggested by Glorot et al. [21], is that it enables sparse representations, where only a small portion of the neurons are active for the same input. This can have desirable effects such as the representation becoming more distributed and less entangled, meaning that it is easier to understand cause and effect.

The flat left half of ReLU can sometimes lead neurons to become inactive, and unlikely to recover, during training. LeakyReLUs try to alleviate this problem with a small slope in the negative range.

The first derivatives of the four activation functions mentioned earlier are as follows:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z)) \tag{3.6}$$

$$\frac{\partial \tanh(z)}{\partial z} = 1 - \tanh^2(z) \tag{3.7}$$

$$\frac{\partial \mathrm{ReLU}(z)}{\partial z} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z \leq 0 \end{cases} \tag{3.8}$$

$$\frac{\partial \mathrm{LeakyReLU}(z)}{\partial z} = \begin{cases} 1 & \text{if } z >\geq 0 \\ \alpha & \text{if } z \leq 0. \end{cases} \tag{3.9}$$

The piecewise linear activation functions are not differentiable at zero. To overcome this in practical use, the derivative at that point is chosen to be some constant, such as 0 or 1. The derivatives are used during the backward pass of the neural network.

3.4 Fully Connected Neural Networks

A fully connected neural network (FCNN), also known as a *multi-layer perceptron*, is one of the simplest and most common types of neural networks. It is a *feedforward neural network* (FFNN), which means that the connections between the neurons in the network do not form cycles. This is distinct from *recurrent neural networks*, discussed in Section 3.6, which can have cycles, allowing the network to use its previous outputs as inputs.

Fully connected neural networks are organized into multiple layers of neurons, in such a way that each neuron is connected to all the neurons in the preceding layer and the subsequent layer. Neurons within a layer are not connected to each other. Each neuron has its own bias, and each connection between two neurons has its own weight. Together, the weights and the biases constitute the parameters θ of the network.

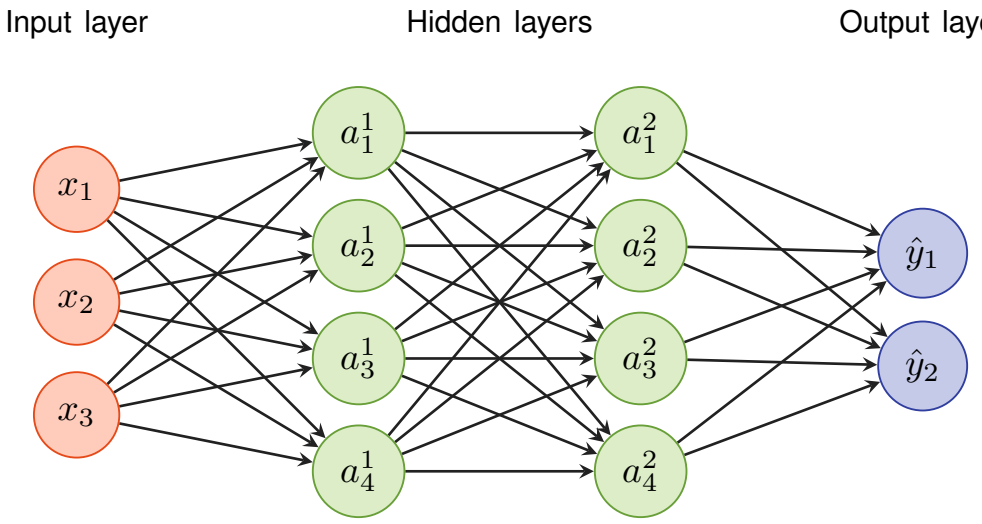


Figure 3.3. *The structure of a fully connected neural network.*

The structure of an FCNN is visualized in Figure 3.3. An FCNN comprises an input layer, an arbitrary number of hidden layers, and an output layer. The dimensions of the data dictate the number of neurons in the input and output layers. The number of input neurons is the same as the number of features in the feature vector. In classification problems, the number of neurons on the output layer is typically equal to the number of discrete classes. Each hidden layer may have an arbitrary number of neurons. Neurons on the hidden layers and the output layer are the kind of artificial neurons discussed in Section 3.2. Neurons on the input layer are a special case: their activation is equal to the corresponding value in the input vector. The number of hidden layers and the number of neurons in each layer is an architectural choice, and there does not exist a general way of determining them.

3.4.1 Forward Pass

The process that yields the predicted output $\hat{\boldsymbol{y}}$ from the input vector $\boldsymbol{x}$ is known as the forward pass of a neural network. In the forward pass, the input data is passed to the input layer, and the activation of each neuron on the following layers is calculated. The activation is calculated in the same way as in Equation 3.2. In order to disambiguate between the different neurons in the network, some additional indices have to be introduced. Let a_j^l denote the activation of the j^{th} neuron on the l^{th} layer of the network and let w_{ij}^l mark the weight from the j^{th} neuron on the $(l-1)^{\text{th}}$ layer to the i^{th} neuron to the l^{th} layer. Using this notation, the activation of a given neuron in the network is given by

$$a_j^l = \mathcal{F}_j(z_j^l) = \mathcal{F}_j\left(\sum_{j' \in J^{l-1}} (w_{jj'}^l a_{j'}^{l-1}) + b_j^l\right), \tag{3.10}$$

where J^{l-1} is the set of neurons on the previous layer.

The last layer in the network is the output layer, which produces the predictions. In classification problems, it is often desirable that the output be a vector that can be interpreted as a probability distribution. This requires an activation function that produces an output that adds up to 1. In binary classification, this can be achieved with a single output neuron and the logistic function. The output of this neuron denotes the probability of one of the two classes being detected, and the complement of the output denotes the probability of the other class. Formally, the predictions are given by

$$p(C_1|\boldsymbol{x}) = \hat{y} = a^L = \sigma(z^L) \tag{3.11}$$

and

$$p(C_2|\boldsymbol{x}) = 1 - \hat{y}, \tag{3.12}$$

where z^L represents the pre-activation output of the neuron on the last layer. The term $\hat{y}$ is often used to mark the activation on the output layer a^L to highlight its role as the

output value produced by the network.

The *softmax* activation function extends this idea to multi-class classification. When using the softmax function, the output layer will consist of a number of neurons equal to the number of classes. The softmax function is used to obtain the class probabilities from the pre-activation outputs of the last layer

$$p(C_i|\boldsymbol{x}) = \hat{y}_i = \text{softmax}(\boldsymbol{z}^L)_i = \frac{e^{z_i^L}}{\sum_{j=1}^{J} e^{z_j^L}} \tag{3.13}$$

where $\boldsymbol{z}^L$ represents the pre-activation outputs of all the neurons on the last layer. The components of the vector produced by the softmax function will add up to 1. The final decision made by the classifier is the class with the highest conditional probability $\arg\max_i p(C_i|\boldsymbol{x})$.

3.4.2 Loss Functions

The loss function is used to evaluate the prediction produced by the forward pass against the true label. The loss is a single value that will be high if the prediction and the label are different from each other, and low if they are close. When training a neural network, the goal is to minimize the loss.

It is common to use loss functions that are derived using the *maximum likelihood principle*. It is based on minimizing the dissimilarity between two probability distributions. In classification, the two distributions are the empirical distribution, defined by the training set, and the probability distribution of the model. The dissimilarity is minimized using the Kullback-Leibler divergence measure. In classification, estimation using the maximum likelihood principle is equivalent to minimizing the negative log-likelihood, also known as the *cross-entropy* [22, pp. 131 – 133]. The resulting loss function for a set of samples is given by

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{j=1}^{I} y_j \log(\hat{y}_j) \tag{3.14}$$

where $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are the predictions and the labels, respectively.

3.4.3 Backward Pass

To enable training neural networks through gradient descent, all operations in the neural network must be differentiable. In gradient descent, the loss function is minimized by calculating the gradient of the loss with respect to the weights and the biases, and then updating them accordingly. The backward pass of a neural network is the process of calculating this gradient, and the algorithm used to do the calculation is called *backprop-*

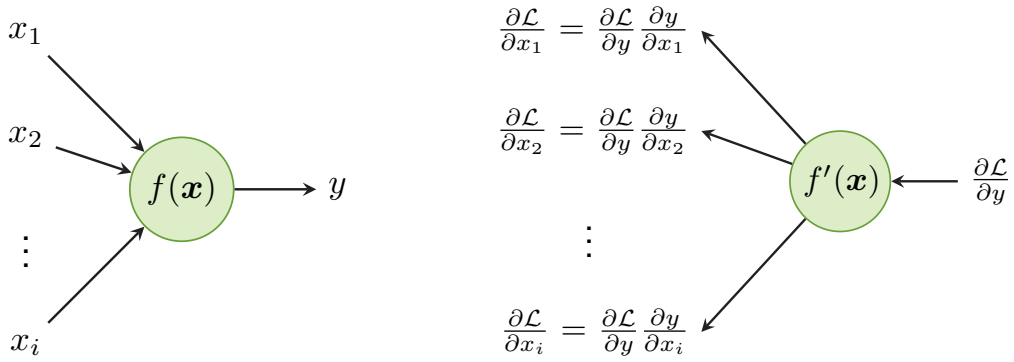
agation.

The first step in the backward pass is to differentiate the loss function with respect to the network prediction. Differentiating (3.14) with respect to the network output gives

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}. \tag{3.15}$$

where $\mathcal{L}$ is shorthand for $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$. This partial derivative indicates how much and in what direction the loss would change if the network output were to change. The network output cannot be changed directly — only the weights and the biases can directly be controlled. Therefore, it is necessary to find out the partial derivatives $\frac{\partial \mathcal{L}}{\partial w_{ij}^l}$ and $\frac{\partial \mathcal{L}}{\partial b_j^l}$ for all the weights and biases in the network, i.e., the gradient of the loss with respect to the parameters $\nabla_{\boldsymbol{\theta}} \mathcal{L}$.

This can be achieved through repeated application of the calculus chain rule for partial derivatives. Figure 3.4 illustrates the application of the chain rule.



(a) *Forward pass* ***(b)*** *Backward pass*

Figure 3.4. *An example of applying the chain rule.*

As an intermediate step, it is helpful to calculate the partial derivatives of the loss function with respect to the pre-activation outputs are, i.e., the weighted sums, of all the neurons in the network. These partial derivatives are denoted by δ_j^l and referred to as *error units*. Taking into account that softmax depends on every pre-activation output of the last layer, the partial derivative of the last layer is given by

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial z_j^L} = \sum_{j'=1}^{J} \frac{\partial \mathcal{L}}{\partial \hat{y}_{j'}} \frac{\partial \hat{y}_{j'}}{\partial z_j^L}. \tag{3.16}$$

To calculate this partial derivative, the derivative of the activation function is needed.

Differentiating the softmax function from Equation (3.13) gives

$$\frac{\partial \hat{y}_j}{\partial z_i^L} = \begin{cases} \hat{y}_i(1-\hat{y}_j) & \text{if } i = j \\ -\hat{y}_i\hat{y}_j & \text{if } i \neq j. \end{cases} \tag{3.17}$$

Now, (3.15) and (3.17) can be substituted into (3.16). Taking into account that $\sum_{j=1}^{J} y_i = 1$, the substitution yields

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial z_j^L} = \hat{y}_j - y_j \tag{3.18}$$

as the error unit, i.e., the partial derivative of the loss function with respect to the pre-activation output of the last layer.

Continuing to apply the chain rule, the network can be traversed backwards, finding all error units δ_j^l. They are given by

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} = \frac{\partial \mathcal{L}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial a_j^l}{\partial z_j^l}\sum_{j' \in J^{l+1}} \frac{\partial \mathcal{L}}{\partial z_{j'}^{l+1}}\frac{\partial z_{j'}^{l+1}}{\partial a_j^l}, \tag{3.19}$$

where J^{l+1} is the set of neurons on the following layer. Substituting in the derivative of the activation function used in each layer, the error unit of the following layer, and the weights, the resulting error unit calculation becomes

$$\delta_j^l = \mathcal{F}_j'(z_j^l)\sum_{j' \in J^{l+1}} \delta_{j'}^{l+1} w_{j'j}^{l+1}. \tag{3.20}$$

This expression can be applied recursively for each layer in the network until the input layer is reached. The derivatives need not be calculated for the input layer as it has no parameters.

Finally, the gradient of the weights and biases can be calculated from these error units. The differential of the bias is simply

$$\frac{\partial \mathcal{L}}{b_j^l} = \frac{\partial \mathcal{L}}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l. \tag{3.21}$$

The bias is a constant added to the weighted sum, and thus its derivative, given the derivative with respect to the pre-activation output, is trivial.

The differential of the weights is

$$\frac{\partial \mathcal{L}}{w_{j'j}^l} = \frac{\partial \mathcal{L}}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{j'j}^l} = \delta_{j'}^l a_j^{l-1}. \tag{3.22}$$

Together, the partial derivatives of all the weights and biases in the network make up the gradient of the loss function with respect to the parameters.

3.5 Training Neural Networks

Training a neural network means finding the weights and biases that best map the input samples to the corresponding outputs. NNs are trained based on a dataset of sample input-output pairs. Training is done through a multi-stage iterative process, where samples in the training set are shown to the network, a loss is calculated, and the network parameters are updated to minimize this loss, using the gradient of the loss function. This process is known as *gradient descent*.

The iterative nature of the training process requires that there be some initial values for the parameters, namely the weights and the biases. The initial values can have a major impact on the training process. This process and the effect the initial values have on it are not yet fully understood. Still, there are some properties that are thought to positively impact the training process. The best understood property is that the initial values need to break the symmetry between the neurons, i.e., different neurons using the same inputs should behave differently. Typically, biases are initialized using a constant, and weights are initialized using small random values [22, pp. 301 – 302].

The training proceeds by iterating over the training set. The iteration happens over *epochs* and *batches*. An epoch is a full pass over the training set. The number of epochs needed to train the network depends on a variety of factors, but is typically in the range of tens or hundreds. The order of the samples is typically shuffled in each epoch. A batch is a subset of the samples in the epoch that are shown to the network between parameter updates. A single batch can contain one sample, the full epoch, or something in between. Choosing the batch size is discussed in Section 3.5.1.

The parameters are updated after each batch. The samples in the batch $\boldsymbol{X}$ are shown to the neural network and the predicted outputs $\hat{\boldsymbol{Y}}$ are calculated based on the current weights and biases, using the forward pass of the neural network. The loss between each predicted output $\hat{\boldsymbol{y}}$ and label $\boldsymbol{y}$ is calculated using a loss function. Finally, the gradient of the cost function with respect to the model parameters is calculated using the back-propagation algorithm. The gradient indicates how the parameters should be changed in order to grow the loss function as quickly as possible. This is indicated in terms of both direction and magnitude. To decrease the loss, the parameters are updated using the negative of the gradient. The simplest update rule $\Delta\boldsymbol{\theta}$ is given by

$$\Delta\boldsymbol{\theta} = -\alpha\nabla_{\boldsymbol{\theta}}\mathcal{L}, \tag{3.23}$$

where α is the learning rate, a small positive constant that controls how much the parameters should change on a single update. The new parameters can then be simply

calculated by

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{3.24}$$

The update rules are often referred to as *optimizers*. Different optimizers are discussed in Section 3.5.1.

The training process continues until any one of its stopping criteria have been met. Typical stopping criteria include stopping after a fixed number of epochs and early stopping (see Section 3.5.3). The full training process is concisely described in Algorithm 1.

Initialize the parameters randomly.
repeat
    Shuffle the order of the samples in the training data.
    foreach *batch in the training data* **do**
        foreach *sample in the batch* **do**
            Calculate the predicted output for the sample with the current parameters.
            Calculate the loss between the predicted output and the label.
            Accumulate the gradient values.
        end
        Use gradient descent to update the parameters.
    end
until *any one of the stopping criteria have been met*

Algorithm 1: Training neural networks with gradient descent.

In addition to the network parameters, there are many *hyperparameters*, i.e., parameters that control the model architecture and some details of the training process. These parameters are chosen before training the network, either by hand or heuristically, e.g. by using a search algorithm like grid search or random search. Hyperparameters include values such as the learning rate, the number of layers, the number of neurons in each layer, the number of epochs, and the batch size.

3.5.1 Variants of Gradient Descent

Gradient descent algorithms vary in two major ways. First, they vary by their batch sizes, i.e., how many samples are shown to the network between each parameter update. Secondly, they vary by their update rules, i.e., the way the parameters are modified by the gradient. As gradient descent is fundamentally an optimization algorithm, the update rules are often referred to as optimizers.

The different variants have been created to solve practical issues when training neural networks. The variations in batch size make a trade-off between accurate gradients and computational cost. The various optimizers aim to increase the speed of convergence

and improve the final accuracy with properties such as momentum and an adaptive learning rate.

Batch Size

In batch gradient descent (BGD), sometimes also called vanilla gradient descent, the gradient for all of the samples in the training set is computed before each parameter update. In other words, the batch is the full epoch. While this approach produces the most correct gradients with respect to the full training dataset, it is computationally expensive, especially if the dataset or the network are large. Furthermore, calculating the gradient for the full epoch repeatedly results in many redundant calculations [57].

In stochastic gradient descent (SGD), the gradient is calculated and the parameters are updated for one sample at a time. This is much faster computationally but causes heavy fluctuations. According to [57], the fluctuations can be enable the training process to escape from a local minimum to a different, possibly better local minimum. However, if the learning rate is decreased slowly over the course of training, the convergence behavior is similar to that of vanilla gradient descent.

Mini-batch gradient descent (MBGD) is a compromise between these two approaches. It calculates the gradient for a small subset of the samples at a time and then updates the weights. This is more stable than SGD, and more computationally efficient than BGD, and can make use the highly optimized matrix operations available on modern hardware [57].

In practice, MBGD is the most commonly used variant used today. Confusingly, the terms gradient descent and SGD are sometimes used when referring to mini-batch gradient descent.

Optimizers

The simplest way of updating the parameters based on the gradient is to multiply the gradient with the learning rate, and subtract it from the previous parameters, as given by Equation 3.23. This vanilla update rule is not without its limitations. One major issue is that choosing the learning rate can be difficult, as it involves making a trade-off between the speed of convergence and the amount of fluctuation. Another issue is the training process getting stuck in sub-optimal local minima and so-called saddle points [13], which are surrounded by areas of plateau in the gradient, making it difficult for the training process to improve the parameters.

A large variety of optimizers have been suggested to overcome many of these limitations. The review by Ruder [57] provides an overview of these optimizers. One commonly suggested change is adding a *momentum* term to the gradient update rule, which works similarly to the physical quantity, resulting in faster convergence and a decrease in fluctuation. Another common change is to have an adaptive learning rate that changes during the training process, and further, to have different learning rates for each parameter. The

review purports that while these modified optimizers do not always outperform the vanilla update rule, it is generally recommended to use them for deep or complex neural networks. The differences between adaptive learning rate optimizers are small, but Adaptive Moment Estimation (Adam) [39] is said to be overall the best choice.

3.5.2 Generalization

Generalization refers to a model's ability to produce accurate predictions on data that is outside of the training set. During the training process, the model parameters are being updated based on its performance on the training set alone. As the training process continues, it is possible for the model to start learning properties that are not inherent to the problem at hand, but specific to the data in the training set, resulting in decreased performance for other data. This phenomenon is known as *overfitting*. It is a particularly common issue when the size of the available dataset is small compared to the number of parameters that have to be trained. The opposite phenomenon, *underfitting*, occurs when the model lacks the capacity to capture the relevant properties of the data.

To measure a model's generalizability, some data is typically set aside for validating the model performance. Ideally, separate datasets are created for *validation* and *testing*. During the development, when the neural network model and its hyperparameters are subject to change, the validation set is used for evaluation. The test set is set aside during the development phase and is only used once the final system has been fully trained. The purpose of this is to make sure that the test set remains an objective measure of generalization and does not influence the decisions made during the development.

Unfortunately, data is often scarce, which means that it is not possible to split the data into three parts while still retaining enough data in the training set to train the system properly. One common way to solve this problem is to use K-fold cross-validation, where the dataset is split into K parts. One part is left out for validation, and the other $K-1$ parts are used for training. The training process is repeated K times, using a different part for validation at each iteration. Finally, the results are averaged. The result is a more accurate indicator than simple validation accuracy, at the cost of significantly increased computational complexity.

3.5.3 Regularization

Regularization refers to methods used to reduce overfitting and thus improve the model's ability to generalize. In particular, regularization methods attempt to penalize extraneous complexity without compromising the model's ability to learn complex relationships.

Adding a weight decay term to the loss function is one of the most common regularization methods. This results in the loss function preferring smaller weights and biases. The assumption behind this method is that overfitting is caused by extreme weight or bias values, and thus preferring small values should reduce overfitting. Another common reg-

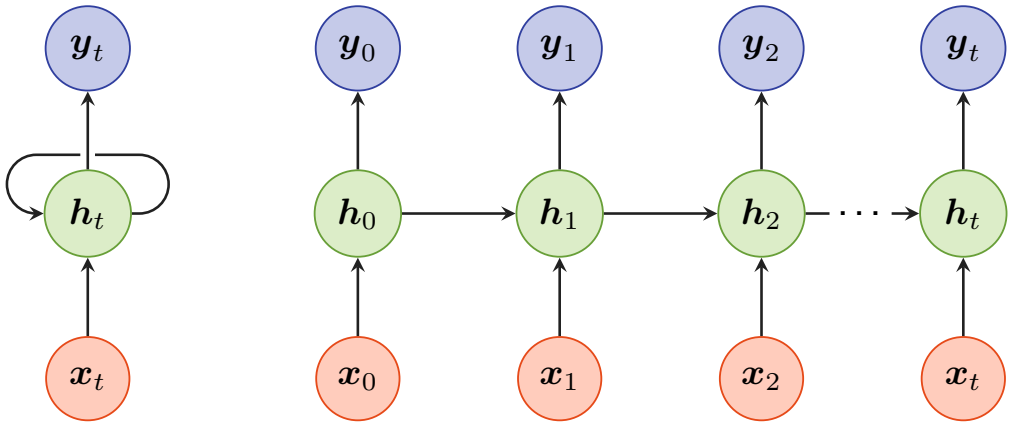
ularization method, used particularly with deep neural networks, is dropout [65], where randomly chosen neurons are temporarily disabled during each iteration in the training process. This penalizes the model from depending too much on particular neurons and connections, and thus reduces overfitting.

There are many other methods to combat overfitting that are not regularization methods per se, but may have a regularizing effect. One such method is to stop the training process when the model's loss on the validation set stops improving, or when some other similar condition is triggered. This is called *early stopping*. Another method to reduce overfitting is to use *data augmentation* to artificially expand the training set. This is done by adding noise and various transformations to the training samples, resulting in new, different samples. It is vital to make sure that the new samples are still recognizable as members of the original class. Data augmentation is particularly useful with image data but can also be done with other types of data.

3.6 Recurrent Neural Networks

One limitation of fully connected neural networks and other feedforward neural network architectures is that they expect both the input and the output dimensions to be fixed in size. This limits their usefulness in problems involving sequential data such as text, audio, and sensor data. Recurrent neural networks (RNNs) remove this limitation by introducing cyclical connections, i.e., allowing neurons to use their previous output as an input. With these connections, information about the past inputs gets stored in the *hidden state* of each neuron. This changes the network from being a simple mapping from a domain of inputs to a domain of outputs to a more flexible model that can draw from the entire history of a variable-size input to produce a variable-size output.

The simplest RNN models, sometimes referred to as *vanilla recurrent neural networks*, add a simple weighted connection from the neuron to itself. The input x^t is different for each timestep t, but the weights are shared across the timesteps. The architecture of a simple RNN model is visualized in Figure 3.5 in two representations. The folded representation shows the neuron's connection to itself as a recurrent loop. When unfolding the network, the input at each timestep creates essentially a new copy of the network with connections to the hidden states of the previous timestep. This representation is not only useful for understanding how RNNs work but also essential for the way the backward pass is calculated.



***(a)** Folded* ***(b)** Unfolded*

Figure 3.5. *A recurrent neural network in its folded and unfolded representations. Each node represents a layer of neurons.*

Layers comprising recurrent neurons can be stacked just like the layers in a fully connected neural network. Moreover, recurrent layers and fully connected layers can be used in the same network. This allows for a variety of configurations with variable- and fixed-size inputs and outputs. A common setup for classifying sequential data is to have a variable-to-fixed architecture, with recurrent layers after the input, and fully connected layers before the output.

The flexibility provided by recurrent neural networks has resulted in a range of different architectures employed in various problem domains beyond classification. Many problems require both the input and the output to be of variable sizes. Examples of such problems include automatic speech recognition and machine translation. ASR is often formulated as a sequence learning problem, where the goal is to assign labels to parts of the input sequence [25, pp. 7–9]. This can be achieved using a large RNN with a variable-to-variable architecture (e.g. [2]). In machine translation it is common to use encoder-decoder models, which consist of a variable-to-fixed encoder that creates an intermediate representation from the original string, and a fixed-to-variable decoder that produces a translated string from the intermediate representation.

3.6.1 Forward Pass

The forward pass of a vanilla RNN is very similar to that of a fully connected neural network. The difference is that the input sequence $\boldsymbol{x}$ consists of vectors $\boldsymbol{x}^t$ at each timestep $t \in T$, and the hidden states need to be recursively calculated considering not only the current input but the previous hidden states as well. The pre-activation output of a recurrent neuron is given by

$$z_j^t = b_j + \sum_{i \in I^{l-1}} w_{ij} h_i^t + \sum_{j'=1}^{J} w_{j'j} h_{j'}^{t-1} \tag{3.25}$$

where I^{l-1} is the set of neurons on the preceding layer, h^t is the hidden state at time t and the upper index is used to mark the timestep. The layer index is omitted for clarity.

The activation function is applied in exactly the same way as it is applied for FFNNs. The hidden state is given by

$$h_j^t = \mathcal{F}_j(z_j^t). \tag{3.26}$$

An initial value for the hidden state h_j^0 is required in order to calculate the later hidden states. These initial values can be initialized as zeros or as random values, or learned like any other parameter in the network.

It is common to use a sigmoid activation function, in particular the hyperbolic tangent, as the activation function in an RNN. The hidden state of an RNN is updated repeatedly during the forward pass, and the limited range of a sigmoid function keeps the hidden state from growing uncontrollably. Using an activation function with an unbounded range could cause a numeric overflow when applied repeatedly over multiple timesteps. The hyperbolic tangent is preferred over the logistic function as its zero-centric range $[-1, 1]$ facilitates the training process [22, p. 195].

3.6.2 Backward Pass

The training process of a recurrent neural network is largely the same as that of a feed-forward neural network. The algorithm for finding the gradient of the loss function with respect to the parameters of an RNN is called *backpropagation through time*. It is a natural extension of the backpropagation algorithm. It involves applying backpropagation to the unfolded representation of an RNN, illustrated in Figure 3.5. This representation does not contain cycles, which allows for the steps to calculate the gradient to be well-defined. Considering the unfolded graph, the error unit calculation from Equation (3.20) becomes

$$\delta_j^t = \frac{\partial \mathcal{L}}{\partial z_j^t} = \mathcal{F}'(z_j^t)\left(\sum_{i=1}^{I} \delta_i^t w_{ij} + \sum_{j'=1}^{J} \delta_{j'}^{t+1} w_{j'j}\right). \tag{3.27}$$

Calculating these error units involves starting at the last timestep T and recursively calculating the error units, decrementing t until the first timestep is reached. Since the loss function has no value beyond timestep T, $\delta_j^{T+1} = 0 \, \forall j$.

The derivatives of the weights and the biases with respect to the loss function can be calculated by summing the derivatives at each timestep. The derivative of the biases becomes

$$\frac{\partial \mathcal{L}}{b_j} = \frac{\partial \mathcal{L}}{\partial z_j^t}\frac{\partial z_j^t}{\partial b_j} = \sum_{t=1}^{T} \delta_j^t \tag{3.28}$$

and the derivative of the weights becomes

$$\frac{\partial \mathcal{L}}{w_{j'j}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial z_j^t} \frac{\partial z_j^t}{\partial w_{j'j}} = \sum_{t=1}^{T} \delta_{j'}^t a_j^{t-1}. \tag{3.29}$$

Notice that the same weights and biases are shared at each timestep.

3.6.3 The Problem of Vanishing and Exploding Gradients

While RNNs may appear to be a straightforward extension of FFNNs, training them can be very challenging in practice. This is particularly true for long input sequences and deep architectures. Training RNNs using gradient based methods involves calculating the gradient of the loss function with respect to the parameters. The gradient on the early layers of the network is calculated through the repeated application of the chain rule, working backwards from the loss function to the earlier layers and earlier timesteps. This long chain of calculations can result in either very small or very large gradient values that make it difficult to adjust the parameters on the early layers [22, pp. 289–290]. This is known as the problem of *vanishing and exploding gradients*, one of the major issues encountered when training recurrent neural networks.

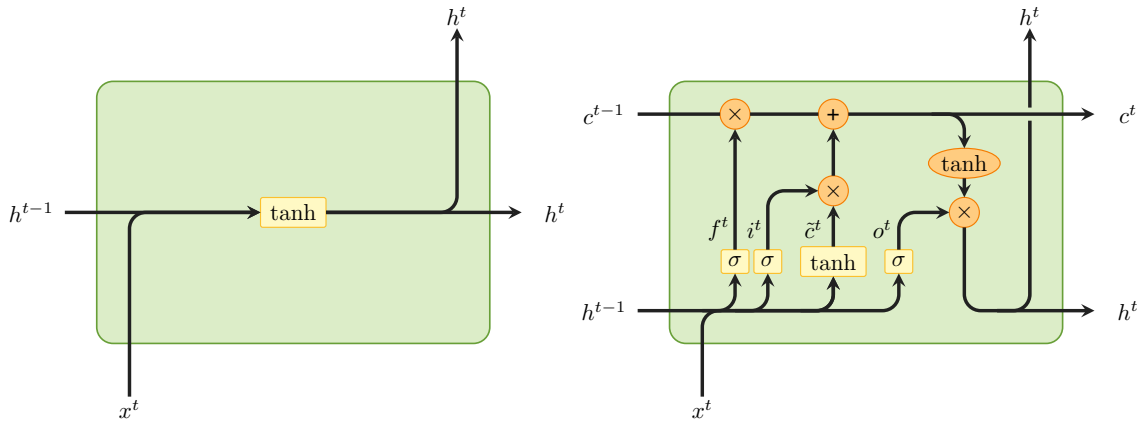
Vanishing and exploding gradients are also a problem for feedforward neural networks but to a lesser extent. There are two main reasons for this. First, the computational graphs in the backward pass of an FFNN tend to be shorter than those of an RNN. RNNs apply the same operations at each timestep, exacerbating the vanishing gradient problem. Secondly, FFNNs can use activation functions such as ReLU, which result in sparse representations and piecewise gradients that avoid the vanishing gradient problem for active neurons [21].

Various solutions for mitigating the vanishing and exploding gradient problems have been proposed. Pascanu et al. [53] explored the vanishing and exploding gradient problem in RNNs. They found that clipping the gradient values when the norm of the gradient grows above some threshold may be a sufficient solution for exploding gradients. The vanishing gradient problem is more difficult to solve. Suggested solutions include leaky units, skip connections [22, pp. 406-408], and a regularizing parameter to limit the growth of gradients [53]. The most efficient solution, however, is to use a different type of RNN altogether.

3.6.4 Long Short-Term Memory

In practice, vanilla recurrent neural networks are not capable of storing long-term information. Moreover, training them is difficult due to vanishing and exploding gradients, even with the solutions discussed in Section 3.6.3. Long short-term memory networks (LSTMs) were explicitly designed to tackle these problems.

Vanilla RNNs use simple neurons that apply an activation function on the weighted sums from the previous layer and the previous timestep, producing a hidden state value. An RNN neuron is visualized in Figure 3.6a. LSTMs replace these neurons with LSTM units, illustrated in Figure 3.6b, that contain a *cell state* in addition to the hidden state. The cell state is controlled by gates that allow the LSTM unit to add or remove information. This allows it to selectively remember the most important aspects of the past inputs.



***(a)** Vanilla RNN* ***(b)** LSTM*

***Figure 3.6.** A vanilla RNN unit and an LSTM unit.*

An LSTM unit has three cells, namely the forget gate, the input gate, and the output gate. Each gate works like an RNN neuron, taking the values of the previous hidden state and the current input, and producing a weighted sum that is passed through an activation function. The logistic function is used as the activation function for its $[0, 1]$ value range, which allows the gate to be open (1), closed (0), or something in between. In Figure 3.6b, these gates are represented by the σ blocks. All gate values are calculated essentially in the same way, but each gate has its own set of weights and biases.

The forget gate, decides whether the current cell state should be forgotten or remembered. The forget gate value for the j^{th} LSTM unit in a layer is given by

$$f_j^t = \sigma\left(b_j^f + \sum_{i=1}^{I} w_{ij}^f x_i^t + \sum_{j'=1}^{J} w_{j'j}^f h_{j'}^{t-1}\right), \tag{3.30}$$

where the upper index f on the weights and the bias denotes that they are specific to the forget gate.

The second gate is the input gate. It controls whether the cell state should be updated with a new value. Similarly to the forget gate, the value of the input gate is given by

$$i_j^t = \sigma\left(b_i^i + \sum_{i=1}^{I} w_{ij}^i x_i^t + \sum_{j'=1}^{J} w_{j'j}^i h_{j'}^{t-1}\right). \tag{3.31}$$

The new candidate cell state $\tilde{c}^t$ is calculated in the same way as the hidden state in an

RNN, and is given by

$$\tilde{c}_j^t = \tanh\left(b_i^c + \sum_{i=1}^{I} w_{ij}^c x_i^t + \sum_{j'=1}^{J} w_{j'j}^c h_{j'}^{t-1}\right). \tag{3.32}$$

With these three values, it is possible to calculate the new cell state c^t. The old cell state c^{t-1} is multiplied by the forget gate value to decide how much information should be forgotten. The new value, produced by multiplying together the input gate value and the candidate cell state, is added to the result. The new cell state is given by

$$c_j^t = f_j^t c_j^{t-1} + i_j^t \tilde{c}_j^t. \tag{3.33}$$

The output gate controls how much of the cell state should be given as the output of the LSTM, i.e., the hidden state. It is given by

$$o_j^t = \sigma\left(b_i^o + \sum_{i=1}^{I} w_{ij}^o x_i^t + \sum_{j'=1}^{J} w_{j'j}^o h_{j'}^{t-1}\right). \tag{3.34}$$

Finally the hidden state is calculated. To produce the hidden state, the cell state is passed through a hyperbolic tangent to keep the output in range $[-1, 1]$ and is multiplied by the output gate value. This is given by

$$h_j^t = o_j^t \tanh(c_j^t). \tag{3.35}$$

The backward pass of an LSTM is similar to that of a vanilla RNN. See, e.g., [25, p. 38] for the backward pass equations.

LSTMs have been found to perform much better than vanilla RNNs in many tasks, especially those requiring long-term contextual information [25, p. 32]. Variants of LSTMs and alternative gated RNN models have also been suggested. The Gated Recurrent Unit (GRU) is a popular alternative to LSTMs. The main difference between LSTMs and GRUs is that GRUs omit the output gate and do not store a separate cell state. The resulting model is simpler and thus faster than LSTMs with a performance comparable to LSTMs in many tasks [12]. Other alternatives have been researched in various works, such as that by Greff et al. [28], but no variant that consistently outperforms LSTMs has been found.

3.6.5 Bidirectional Recurrent Neural Networks

The recurrent neural networks discussed thus far in this chapter are causal in their architecture. This means that they only produce an output based on the current and the previous inputs as opposed to the full sequence. Many real-world problems require additional context to make accurate predictions. For example, in ASR, the interpretation of the

phoneme does not only depend on the current and the previous phonemes; it can also depend on the following phonemes through co-articulation and context [22, pp. 394 – 395].

A common solution for this problem is to use bidirectional neural networks (BRNNs). BRNNs are RNNs that have been extended to process the input sequence both forwards and backwards. This is done by presenting the input sequence to two separate recurrent layers, one for each direction. Both layers are connected to the same output layer, providing it with context from both the past and the future. The two layers are not connected to each other, which means that the unfolded representation of the network remains acyclic. The unfolded representation of a BRNN is illustrated in Figure 3.7. The recurrent layers may be vanilla RNNs, LSTMs, or any other kind of RNN layers.

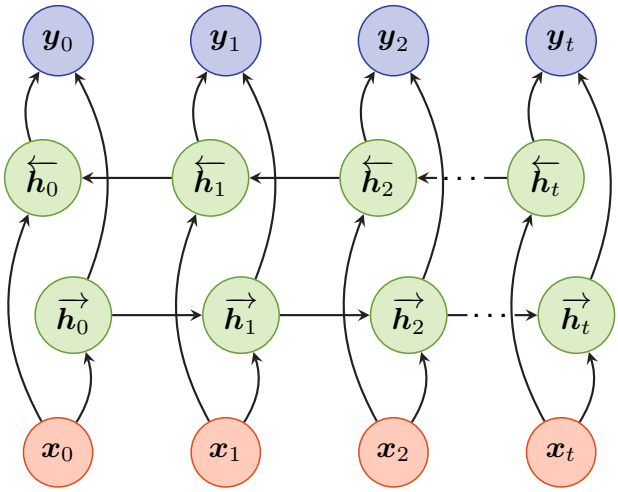


Figure 3.7. *A bidirectional recurrent neural network in its unfolded representation. Each direction is processed using a separate layer, making the unfolded graph acyclic.*

BRNNs have been found to give improved results in domains such as handwriting recognition, ASR, and bioinformatics [22, p. 395]. The additional context provided by the bidirectional structure is helpful in these problems. On the other hand, requiring access to future inputs constrains where BRNNs can be used in practice. In real-time systems it may not be acceptable to wait for the entire input to be recorded before making predictions. This problem can be mitigated by segmenting the input, e.g, processing one word at a time in handwriting recognition. Furthermore, using BRNNs is clearly not feasible in problems that are causal by their nature, such as predicting chess moves or stock market prices.

4 METHOD

This chapter presents a proposed method for lexical stress recognition. On a high level, the system takes an audio recording containing the utterance of an isolated word as its input, and predicts a probability distribution indicating the probability that a given syllable in the word carries the primary stress. A neural network classifier is used to make the prediction. One neural network is used across multiple words, but an extra input such as the number of syllables in the word, or a unique identifier of the word is used.

The data preprocessing procedure consists of three parts, namely feature extraction, normalization, and label encoding. The input features used for the neural network classifier are calculated from the raw audio recordings. Two sets of features are compared: a set of features based on the prosodic properties of speech, and Mel-scaled spectrograms, which are commonly used in speech recognition tasks. Data preprocessing is discussed in more detail in Section 4.1.

The system is trained in a supervised manner. The problem is formulated as a classification problem where only one syllable in a word can carry the primary stress. Secondary stresses are ignored. As discussed in Section 2.1, this approach has its limitations, as it is possible for non-natives to stress multiple syllables in a word with equal prominence. Taking this into account would, however, significantly increase the complexity of the problem and call for a different set of methods, such as formulating the problem as one of sequence labeling (see [25]) rather than classification. Section 4.2 presents details on the training process, including the neural network architectures and the hyperparameter search.

4.1 Data Preprocessing

Before the data can be fed to the neural network, various steps have to be taken to preprocess it into the proper format. This includes feature extraction and normalization for the inputs, and encoding the labels to produce the target outputs.

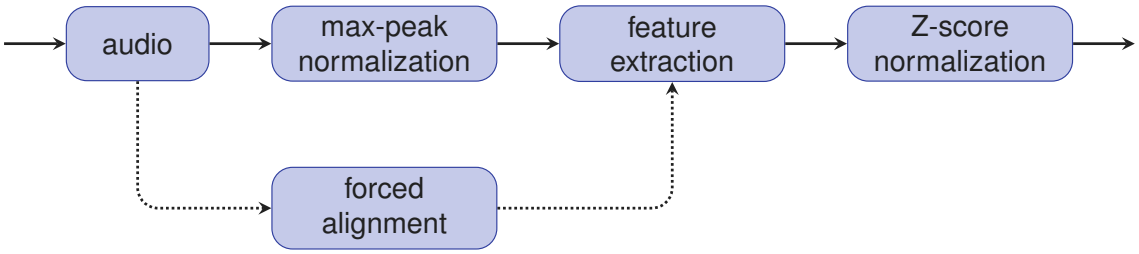


Figure 4.1. *An overview of the input data preprocessing.*

An overview of the data preprocessing procedure used in the experiments is presented in Figure 4.1. The following sections give more details on each preprocessing step.

4.1.1 Feature Extraction

Feature extraction is the process of turning raw data into a representation that can be used as an input to the learning model. The purpose is to find a representation that captures the aspects of the data that differentiate the classes.

Feature extraction can be done explicitly through a process called *feature engineering*, where domain knowledge is used to extract the most useful features for the task. This process can be time-consuming and difficult, but it can lead to very effective features that work even when data is scarce. The prosodic features presented in Section 4.1.1 are an example of this approach.

Alternatively, feature extraction can be simplified or even forgone entirely using *feature learning*, where raw data is used as the input, and the model learns the features as a part of its training process. While this approach is simple, it requires a large dataset and a large model to be effective. In this experiment, feature learning is not done from the raw audio data, but from Mel spectrograms. They are specifically designed for audio processing tasks, but they do not contain any specific information regarding lexical stress. As such, the neural network model needs to discover the representations needed for lexical stress recognition automatically.

Prosodic Features

As discussed in Section 2.2, it is widely agreed that a stressed syllable implies a higher fundamental frequency, more loudness, a longer duration, and a difference in spectral tilt. As such, it makes sense to develop features that are representations of these variables.

These features are calculated separately for each syllable or syllable nucleus, i.e., vowel, in the audio recording. The syllable locations are obtained using a pre-existing system for *forced alignment*, i.e., a system for determining the temporal location of each phoneme in the audio. The duration features are also calculated from these values.

Calculating the fundamental frequency can be done using various pitch tracking algo-

rithms. In this experiment, the YAAPT algorithm [74] was used. It has been commonly stated that f_0 is more likely to be a correlate of pitch accent than lexical stress. Nonetheless, for the purpose of a practical language learning application and, therefore, for the purpose of this system, the distinction is not particularly important — what is important is to recognize the stressed syllable, whether or not it has a pitch accent. Moreover, as pitch accent is a phrase-level feature of speech, this problem does not exist on datasets containing isolated words. Fundamental frequency is not calculated for the full syllable segment, but frame-wise, resulting in a vector of values.

Two loudness measures are used. Both are frame-wise features. The first feature is root mean square (RMS) energy. The second one is peak-to-peak amplitude, calculated as the difference between the highest and lowest amplitude values in the audio segment.

Spectral tilt is another feature that has been found to be a reliable correlate of lexical stress. It measures the distribution of energy between high and low frequencies, but it does not have an exact definition. In this experiment, spectral tilt was calculated as the difference of energy between on the first harmonic frequency and the second harmonic frequency, like in [6]. Spectral tilt is also calculated frame-wise.

Feature	Calculated For		Statistics			
	Nucleus	Syllable	Mean	SD	Max	Min
RMS Energy	✓	✓	✓	✓	✓	✓
Fundamental Frequency	✓	✗	✓	✓	✓	✓
Spectral Tilt	✓	✗	✓	✓	✓	✓
Peak-to-Peak Amplitude	✓	✓	✓	✓	✓	✓
Duration	✓	✓	N/A			

Table 4.1. *Prosodic features used in the experiments.*

All of the features are syllable specific, but some of them are frame-wise feature vectors, while others are scalar values. Four scalar statistics, namely the mean, the standard deviation (SD), the maximum, and the minimum, were calculated for each frame-wise feature. Furthermore, some features can only be meaningfully calculated for the syllable nucleus, i.e., the vowel, as opposed to the whole syllable. Table 4.1 indicates which features were calculated for the syllable nuclei, or the full syllable, and what statistics were calculated from the features, if applicable. The end result is a vector of 26 elements for each syllable.

Mel Spectrograms

The Mel spectrogram comprises the other feature set used in the experiments. While a normal spectrogram, a time-frequency representation of sound, uses a linear frequency scale as its y-axis, a Mel spectrogram uses the Mel scale, which is based on the human perception of frequencies. Human hearing has a better resolution at low frequencies than high frequencies. The Mel scale makes use of this fact, making it suitable for speech processing tasks [30, p. 776]. A widely used approximation for converting frequencies to the Mel scale is

$$m = 2595 \log_{10} \left(1 + \frac{f}{700}\right), \tag{4.1}$$

where f is the frequency in Hz.

Mel spectrograms are calculated directly from the audio without the need for additional metadata. This greatly simplifies the feature extraction process compared to the prosodic features, as computationally expensive steps such as the forced alignment can be omitted. The calculation consists of separating the signal to windows, computing the Fast Fourier Transform for each window, and converting the resulting frequency spectrum to the Mel scale. The conversion to the Mel scale is done using a number of overlapping Mel filters.

Features in Mel spectrograms are highly correlated. Mel-Frequency Cepstral Coefficients (MFCCs) are often used to obtain a more concise, decorrelated representation of the signal. However, this decorrelated representation discards information regarding the harmonic frequencies, which can potentially be useful for lexical stress recognition. Thus, the experiments in this thesis opt for using Mel spectrograms instead of MFCCs. 128 Mel filters are used.

4.1.2 Normalization

To make sure that the features from different audio recordings are comparable with each other, it is important to normalize the data. Normalization can be applied at many different stages of preprocessing.

The first kind of normalization used in this system is *max-peak normalization*. Its purpose is to scale the amplitude range in each audio recording to be $[-1, 1]$. Having the same amplitude range for each sample improves the accuracy of various features, particularly the loudness measures. The max-peak normalized audio signal is given by

$$\bar{\boldsymbol{s}} = \frac{\boldsymbol{s}}{\max(|\boldsymbol{s}|)}, \tag{4.2}$$

where $|\cdot|$ denotes the absolute value operation, applied element-wise to a vector.

Further normalization is commonly applied to the input vectors after feature extraction. One common way to do this is to apply Z-score normalization, also known as feature standardization, where each feature in the data is scaled to have zero-mean and unit-variance. A Z-score normalized feature vector is given by

$$x' = \frac{x - \mu}{\sigma}, \tag{4.3}$$

where x is a value in the feature vector, and μ and σ are the mean and the standard deviation of that feature across the training set, respectively. Feature standardization is known to increase the speed of convergence [32].

Finally, *batch normalization* [32] can be applied during the training process. It is meant to fix a problem known as the *internal covariate shift*, where the input ranges of each layer change over the course of training. Batch normalization normalizes this input range, which improves the training process in various ways. It allows for higher learning rates, as this normalization prevents activations for becoming very large or small. It has also been found to have a regularizing effect. In some cases, it can even make other regularization methods such as dropout unnecessary. The normalization is done per batch, hence the name.

4.1.3 Labels

As discussed in Section 3.5, all training data in a supervised neural network needs to have an associated label. Labels can be represented in a variety of ways, depending on the problem and the method.

In lexical stress recognition, labels are representations of the stress patterns present in the original audio. In this experiment, the problem is limited to predicting the syllable carrying the primary stress with the assumption that there can only be one such syllable in a word. As such, a natural way to represent the stress patterns is by using one-hot vectors, where the stressed syllable is denoted by a one, and the unstressed syllables are denoted by zeros. The system is designed to handle words up to five syllables in length. This is a reasonable cut-off point, as longer words in English are rare. In the Carnegie Mellon University Dictionary [67], fewer than 1 % of the words have more than five syllables.

4.2 Training Process

The training process for the neural networks used in the experiments consists of multiple stages. First, a hyperparameter search is conducted on the training data to determine the optimal hyperparameters. Then, these hyperparameters are used to train the final neural network. The following sections provide more detail on this process.

The training process is halted after the accuracy on the validation set has not increased

in 30 epochs, or after a total of 300 epochs, whichever comes first. The test set is used for the purpose of early stopping for the final model. The weights that produced the best accuracy on the validation set are restored upon finishing the training process. The Adam optimizer was used for all neural network experiments.

4.2.1 Neural Network Architectures

Finding the most optimal neural network architecture for lexical stress recognition is not the main purpose of this thesis. As such, only a limited number of neural network models are evaluated. The model architectures used are loosely modeled after to those used successfully to solve related problems in other speech processing related literature, such as [2] and [27].

The neural network architecture consists of a number of bidirectional LSTM layers, followed by a number of fully connected layers. The number of neurons in each layer, and the number of layers of each type, are determined by a hyperparameter search. The model also accepts an arbitrarily sized vector as an extra input. This second input is connected to the first fully connected layer, and it is used to give extra metadata about the words to the model.

Hyperparameter Optimization

Neural network models have a variety of so-called hyperparameters that control the model architecture and the training process. Hyperparameters can be optimized using any of various search algorithms and other optimization methods.

Hyperparameter search algorithms are methods that perform a search over a predefined hyperparameter space. This space is known as a parameter grid, and it contains all possible values for each hyperparameter.

The most simple, and at the same time the most comprehensive, method for hyperparameter search is called *grid search*. It involves training the model repeatedly, exhaustively trying out all possible combinations of hyperparameters and saving the result. Increasing the size of the parameter grid quickly increases the computational cost of grid search. As such, it is typically only used when the parameter grid is small or when the model is fast to train.

A less computationally expensive option is to use *random search*. It involves running the experiment for a randomly chosen subset of all possible combinations. This is dramatically more efficient than grid search, and often leads to comparable if not better performance [5]. The experiments in this thesis use a random search for hyperparameter optimization. The experimental setup is discussed in more detail in the following chapter.

5 EXPERIMENTS

This chapter presents the experiments conducted for this thesis. The datasets, the evaluation procedure, the experiment setup, and the results are discussed.

5.1 Datasets

The method proposed in Chapter 4 was evaluated using two different datasets. The first one only contains utterances of individual words from non-native English speakers. The second one contains full sentences from native American English speakers.

The following sections provide more detail on each dataset. This includes discussion on the characteristics of the data, along with information on how this data was collected and annotated.

5.1.1 Custom Dataset

The dataset referred to as the *custom dataset* is a purpose-built dataset containing utterances of individual English words by native speakers of either Finnish or German. The level of English proficiency of the users is not known. The data was collected through a mobile application that instructed the user to record themselves saying certain English words. The users were informed that the speech samples would be used for the development of automatic pronunciation evaluation systems. The users were not informed about lexical stress in particular. Each user used their own mobile device to provide the speech samples. The samples were recorded at a sampling rate of 44.1 kHz and a bit depth of 16 bits. The environment where the users used this application was not controlled for in any way. These conditions are a close match to the intended real-world use case of this system.

The English words were chosen by English teachers in such a way that they would represent all the common stress patterns that exist in English. Another important consideration was to choose words where the users could be expected to make stress errors. This meant either choosing difficult words, or words where the stress pattern is different from what Finnish and German native speakers are accustomed to.

The data was labeled by at least two expert annotators who were instructed to mark the syllable that carried the primary stress according to their interpretation. Only one syllable

was allowed to be marked as having the primary stress. If the annotators were uncertain about a particular sample, they could mark that sample uncertain, either together with their assessment of the location of the primary stress, or without including it at all. In addition, the annotators were given the choice of discarding samples that did not feature the correct word, had excessive background noise, or had other major issues that rendered the sample unusable. The annotators were instructed not to pay attention to the phonetic realization of the word as long as it was still recognizable. The annotators were not the same throughout the dataset but they were mostly the same for all samples of any given word, with few exceptions. There were two annotators for most of the data but a small subset of the data had between three and five annotators.

Only samples that the annotators agreed upon were included in the final dataset. A sample was considered to be agreed upon if the annotators had not marked the primary stress on conflicting syllables. Samples marked as uncertain were included unless the primary stress assessment was missing from all annotators. A sample weight for each sample was calculated from the uncertainty values using the formula

$$w_{sample} = \max\left(\left(1 - \frac{\#marked\_uncertain}{\#annotations}\right), 0.5\right). \tag{5.1}$$

The purpose of this weight is to give more importance to samples that the annotators agreed on during model training, while also making sure that all samples have at least some weight. When creating the train and test split for this dataset, the test set was created such that it only included fully certain samples. An 80 %/20 % split was used for the training and test sets.

The agreement level for all words was 72.6 %, when including the cases where the annotators are uncertain, but only 61.4 % without including them. These numbers are not directly comparable to the agreement levels found in the literature, discussed in Section 2.1, as the annotation procedures are different.

The full, unprepared dataset comprised 6,863 annotated audio recordings from 914 speakers. Samples were filtered out from this data based on the annotations. First, the 851 files that at least one of the annotators had marked for discardment, were discarded. Secondly, samples where the annotators disagreed were removed. Having left out the discarded and disagreed samples, the resulting dataset comprised 4,983 samples from 848 speakers, including 772 samples marked uncertain by at least one annotator.

There were 12 unique words in the dataset. The relatively high amount of samples per word makes it possible to use this dataset to evaluate how well the system can be trained to distinguish between different lexical stress realizations in a word.

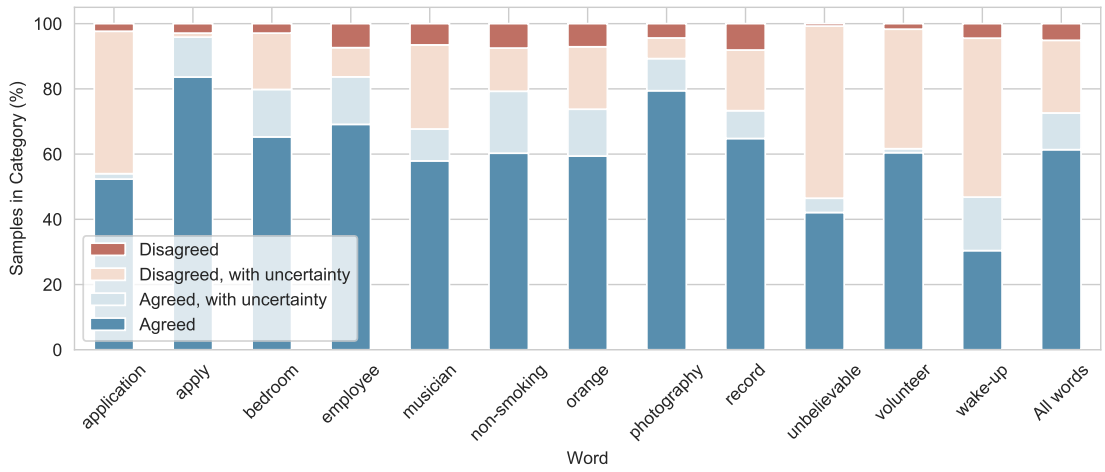


Figure 5.1. *The agreement levels between the annotators for each word in the dataset.*

The word specific agreement levels between the two annotators are shown in Figure 5.1. As the figure shows, the agreement level varies significantly between different words. The agreement levels between the different pairs of annotators were not found to differ significantly.

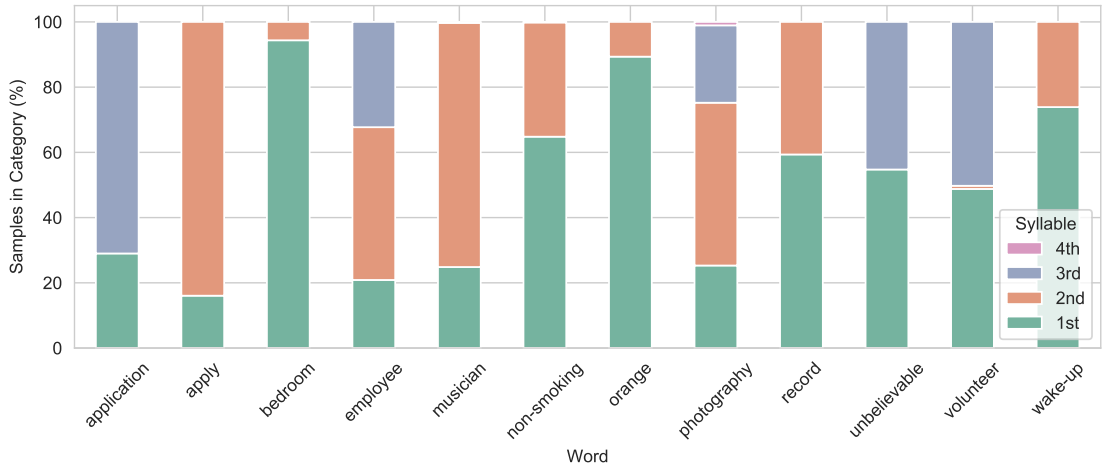


Figure 5.2. *The class distribution for each word in the dataset.*

The distribution of samples to different classes is shown in Figure 5.2. In most words, there are two or three frequently occurring stress patterns, with the correct stress pattern typically being the most common one.

To gauge the degree to which annotators agree with their own annotations, two annotators were asked to reannotate a set of samples that they had annotated a few weeks earlier. The reannotation set was a superset of a set of samples they had previously annotated, in random order. The annotators were not informed that they would be reannotating the same samples. The result was that the agreement level the annotators had with themselves averaged 85 %.

There are many possible hypotheses for the relatively low agreement levels. One potential reason for this could be the presence of a secondary stress, which may increase the difficulty of determining which syllable carries the primary stress. According to the Carnegie Mellon University dictionary, many of the words with the poorest agreement levels, such as *application*, *unbelievable*, and *wake-up* have a secondary stress. Another possible reason is the nature of the utterances. The users were instructed to pronounce the words in isolation. This lack of a surrounding carrier phrase may have affected their realization of lexical stress.

5.1.2 TIMIT Corpus

The *TIMIT corpus* [20] is a publicly available dataset of American English. It contains 6300 sentences spoken by 630 speakers with various dialects of American English. The dataset was created for acoustic-phonetic studies, and it comes with transcriptions and temporal alignments for each spoken phoneme.

In order to use the dataset to train the lexical stress recognition system, individual polysyllabic words were extracted from the sentences using the provided temporal alignments. Words that were longer than five syllables were filtered out.

As the dataset is comprised of native speech, all of its audio samples can be considered correctly stressed without the need for manual annotation. This makes it possible to use a pronouncing dictionary to create the labels. The labels for this experiment were created using the Carnegie Mellon University dictionary. The location of the primary stress in each polysyllabic word was retrieved from the dictionary, ignoring secondary stresses. Words, where the location of the primary stress could change depending on the context, were discarded from the dataset.

The resulting dataset contains 16,489 samples. The dataset contains a large number of unique words, 3,968, but each word in the dataset only occurs an average of four times. While the dataset only contains examples of correct realizations of lexical stress, the large variability in words makes it possible to evaluate how well the system can be trained to recognize the stress pattern regardless of the word. The train/test split provided in the original dataset was used.

5.2 Evaluation Procedure

One of the simplest and most common metrics for measuring classifier performance is accuracy, which is defined as the percentage of samples that were assigned the correct class labels. This metric is dependent on the underlying distribution of labels, which means that accuracy scores for different datasets can not be expected to be comparable. For the same reason, the accuracy score may be a misleading metric. Consider a dataset where 95 % of the samples belong to one class. A zero-rule classifier that always predicts the label of this majority class would reach 95 % accuracy despite not providing any

information. In spite of this major shortcoming, accuracy is a valid metric for comparing results between different models on the same dataset.

Another way to evaluate the classifier is to build metrics that consider all the different possible success or failure cases. A classifier used for automatic pronunciation evaluation can either *reject* the user's input, informing them that they made a mistake, or *accept* the input, if it was deemed correct. This framework results in the following cases:

True acceptances (TA)
: The user's input was correct, and this was correctly recognized.

False acceptances (FA)
: The user's input was incorrect, but the system incorrectly recognized it as being correct.

True rejections (TR)
: The user's input was incorrect, and the system correctly rejected it.

False rejections (FR)
: The user's input was correct, but the system incorrectly rejected it.

Similar nomenclature has been used in some earlier work related to LSR [16]. In other machine learning literature, these terms are usually known as *true positives*, *false positives*, *true negatives*, and *false negatives*, respectively.

False acceptances decrease the usefulness of the system. From the user's point of view, however, false rejections are the worst type of error, as it can be discouraging to be corrected despite being correct.

Using these four categories, the metrics *precision* and *recall* can be defined. They are given by

$$precision = \frac{TA}{TA + FA} \tag{5.2}$$

and

$$recall = \frac{TA}{TA + FR}, \tag{5.3}$$

respectively. TA, FA, and FR refer to the number of samples in the respective categories.

In the context of LSR, precision measures what fraction of the samples that the system accepted were actually correct. It penalizes false acceptances. Conversely, recall measures what fraction of the samples that the system should have accepted were indeed accepted. It penalizes false rejections. A system that accepts every sample would, therefore, have a perfect recall but a poor precision, as many of the accepted samples should have been rejected. Conversely, a system that could accurately accept just one sample as correctly stressed, and reject the rest, would have a perfect precision, but a poor re-

call. Precision is ill-defined when all samples are rejected, as not having any acceptances results in division by zero. In these instances, precision is treated as a zero.

Clearly, neither precision nor recall alone is enough to give a useful estimate of the model's performance. Both of them need to be examined in order to assess the model. However, having a single figure as a metric is often desirable. The F_1-score is the harmonic mean of the two metrics, and it is commonly used for this purpose. It is given by

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}. \tag{5.4}$$

The F_1 score gives equal weight to precision and recall. It is a specific instance of the more generic F_β-score, where β can be adjusted to give more weight to either of the two metrics.

Precision, recall, and consequently, the F_1-score, are metrics for binary classification. In a multi-class context, the metric can be calculated class-wise by binarizing the labels. The class-wise results can then be aggregated. The binarized labels denote whether the sample is a member of the target class or a member of some other class.

There are many strategies for averaging the class-wise results, including *micro average* and *macro average*. Micro average calculates the metric globally across all classes, without favoring any class in particular. Macro average calculates the metric separately for each class, and averages the result. It gives equal weight all to classes regardless of their size, and thus, the samples in the minority classes are given more weight. This can be desirable when working with an imbalanced dataset, and when performance on the minority classes is of particular interest. As the class distributions in Figure 5.2 suggest, the problem at hand is an imbalanced one. Thus, the macro average of F_1 scores are used in this thesis.

5.3 Experiments

Both datasets were evaluated using models with both prosodic and spectral features, resulting in four experimental setups. A hyperparameter search was conducted to find the best hyperparameter configuration for each setup using 5-fold cross-validation on the training set. This yielded five sets of hyperparameters. The hyperparameters that occurred most often among the top results were chosen as the final hyperparameters. The final models were trained on the full training set using the best hyperparameters, and evaluated on the test set.

The hyperparemeters that were varied in the hyperparameter search included the number of neurons and layers, the activation function, the dropout rate, the batch size, and whether to use batch normalization. The parameter grid had between two and four possible values for each hyperparameter. Running the full grid search over the entire parame-

ter grid would have been prohibitively expensive computationally, so the hyperparameter search was done using randomized search with 50 iterations per fold. The basic framework for the network architecture discussed in Section 4.2.1 is used for all experiments.

The TensorFlow library [1] and, in particular, its built-in version of the Keras interface [10], was used to build the neural network models. Various other libraries, mainly Librosa [47] and AMFM Decompy [59], were used to calculate the features. The forced alignment was done using the Gentle forced aligner [50].

5.3.1 Results

This section presents the results of the experiments on the two datasets. The key metrics are presented and the results are briefly discussed.

Hyperparameter	Custom Dataset		TIMIT Corpus	
	Prosodic	Spectral	Prosodic	Spectral
LSTM neurons per layer	24	96	24	128
LSTM layers	2	4	2	3
FCNN neurons per layer	64	64	64	128
FCNN layers	3	3	2	2
activation on FCNN layers	ReLU	LeakyReLU	ReLU	ReLU
dropout rate on FCNN layers	0.25	0	0.25	0.25
batch size	128	32	64	32
batch normalization	off	off	off	off

Table 5.1. *Hyperparameter search results on both datasets.*

The results of the hyperparameter search are presented in Table 5.1. While both prosodic and spectral features, namely Mel spectrograms, are time series, the spectral features have much larger dimensions and thus require a significantly larger neural network architecture. This is reflected by the hyperparameter search results. Each feature set appears to have a similar set of optimal hyperparameters for both datasets.

Custom Dataset

The model was trained on the custom dataset using both prosodic and spectral features. A unique identifier for each word was given as an extra input, formatted as a one-hot vector. The purpose of this second input was to help the neural network focus on the different classes within each word.

The accuracies and the F_1-scores of the two models are reported in Table 5.2. Results on the zero rule classifier, i.e., a classifier that always predicts the majority class, are also

Word		Zero Rule		Feature Set			
				Prosodic		Spectral	
	N	Acc.	F_1	Acc.	F_1	Acc.	F_1
application	32	0.84	0.46	**0.84**	**0.73**	0.78	0.44
apply	126	0.89	0.47	**0.93**	**0.78**	0.92	0.77
bedroom	89	0.89	0.47	0.88	0.54	**0.90**	**0.63**
employee	96	0.56	0.24	**0.70**	**0.65**	0.59	0.52
musician	88	0.88	0.31	**0.90**	**0.44**	0.89	0.38
non-smoking	91	0.55	0.24	**0.71**	**0.48**	0.66	0.44
orange	102	0.82	0.45	**0.84**	0.59	0.82	**0.60**
photography	102	0.57	0.18	**0.73**	**0.50**	0.71	0.46
record	193	0.60	0.37	**0.92**	**0.92**	0.84	0.84
unbelievable	38	0.66	0.40	**0.71**	**0.71**	0.29	0.29
volunteer	51	0.57	0.24	**0.84**	**0.57**	0.75	0.50
wake-up	59	0.69	0.41	**0.80**	**0.77**	0.69	0.65
Average	1067	0.71	0.51	**0.83**	**0.61**	0.77	0.55

Table 5.2. *Results on the custom dataset. Best results are in bold.*

presented for each word. Since the dataset is imbalanced, this information is particularly valuable for the interpretation of the results. It is especially important to consider the zero rule when interpreting the accuracies.

The classifier using prosodic features outperforms the classifier using spectral features almost universally, often with a large margin. It is possible that the size of the dataset is simply insufficient for the end-to-end classifier.

The results also show that there is a large discrepancy of performance between different words. Some words, such as *record* and *wake-up* show a sizable increase in performance over the zero rule, while others such as *musician* and *bedroom* do not seem to work in any meaningful way, and only show slight improvement over the zero rule. Many of these poorly performing words have a high zero rule accuracy to begin with, indicating that there are very few samples representing the minority classes.

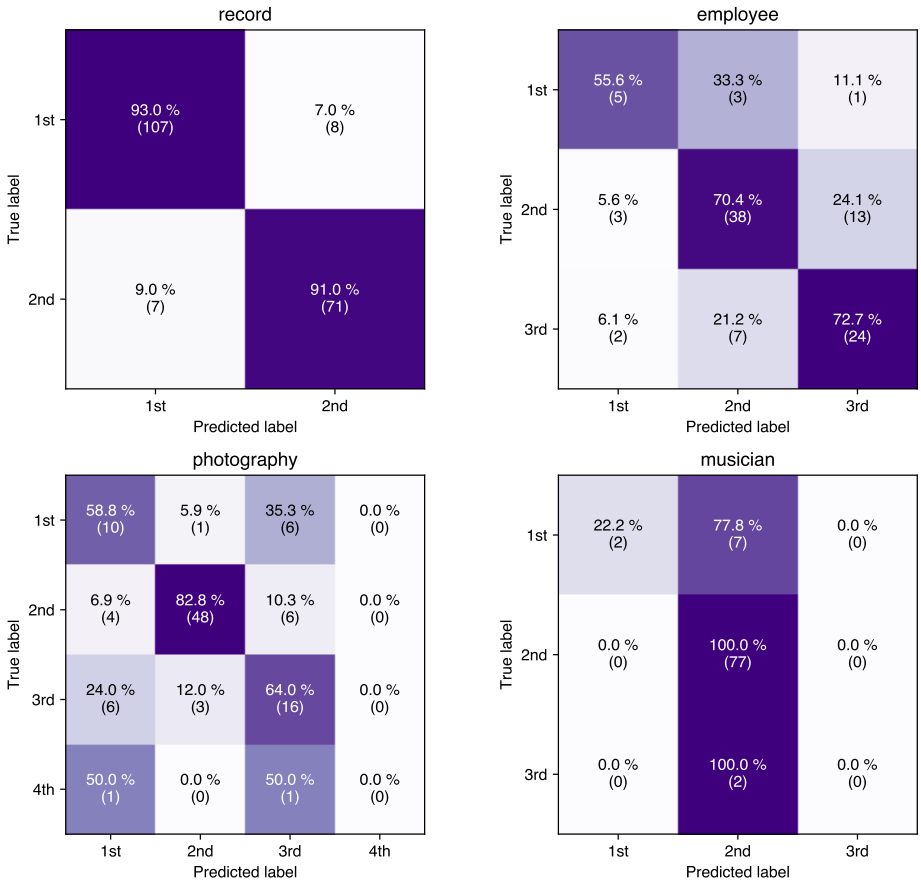


Figure 5.3. *Confusion matrices of different words from the classifier using the prosodic feature set.*

Figure 5.3 shows the confusion matrices of select words in the dataset. In the confusion matrix, each row corresponds to the samples that were labeled as having a particular label, and each column corresponds to the samples that were predicted to have a particular label. A perfect classifier would, therefore, have all of the samples on the diagonal of the matrix. This matrix well illustrates the large performance discrepancy, where the system performs well for some words, and hardly produces meaningful predictions for others.

TIMIT Corpus

The model was trained on the custom dataset using both prosodic and spectral features. The model was also given the number of syllables in the word as an extra input. The purpose of this input was to orient the neural network toward finding the location of the stressed syllable without having to determine the number of syllables in the word from the features.

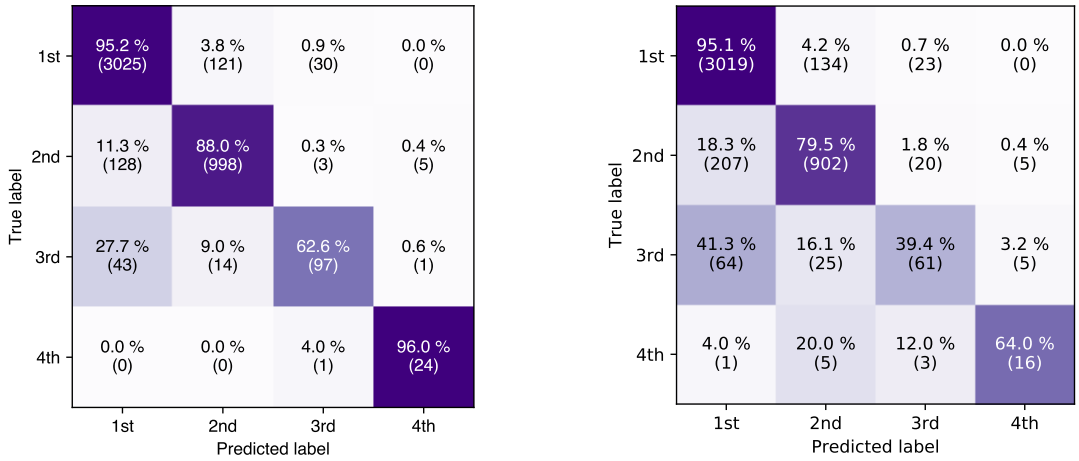
The accuracies and the F_1-scores of the two models are presented in Table 5.3. The dataset contains examples of words with the primary stress on the first, second, third, and fourth syllable. Results on each of these categories are shown separately, but as the dataset only contains correct stress realizations, the zero rule is not meaningful and is

Canonical Stress		Feature Set			
		Prosodic		Spectral	
	N	Acc.		Acc.	F_1
1st	3176	**0.95**	**0.95**	0.95	0.93
2nd	1134	**0.88**	**0.88**	0.80	0.82
3rd	155	**0.63**	**0.68**	0.39	0.47
4th	25	**0.96**	**0.87**	0.64	0.63
Average	4490	**0.92**	**0.85**	0.89	0.71

Table 5.3. *Results on the TIMIT Corpus. Best results are in bold.*

thus omitted.

The results show that the classifier using the prosodic features outperforms the one using the spectral features for all types of words. The difference is very notable on the minority classes, but not as pronounced on the majority class. Both systems appear to have particularly large problems on words, where the primary stress is on the third syllable.



(a) *Model using prosodic features.* ***(b)*** *Model using spectral features.*

Figure 5.4. *Confusion matrices on the two models.*

Figure 5.4 shows the confusion matrices of the two models on the full test set. It can be seen that both models frequently mistakenly predict the primary stress to be on the first syllable when it should be on the third. This is likely due to the presence of a secondary stress on the first syllable. In fact, 91 % of the words in the test set, whose primary stress is on the third syllable, have a secondary stress on the first syllable, according to the Carnegie Mellon University dictionary. Similarly, 34 % of the words that have the primary stress on the first syllable, and have at least three syllables, have a secondary stress on the third syllable. As discussed in Section 2.2, the distinction between primary and secondary stresses can be subtle. Therefore, it is not surprising that the models have difficulty distinguishing between the two, especially when the secondary stresses are not marked in the labels.

5.3.2 Discussion

The results suggest that models using the forced alignment-based prosodic features still outperform the end-to-end approach, particularly on the smaller custom dataset. With the larger TIMIT corpus, the differences were less significant. This suggests that lexical stress recognition without an existing ASR model to provide the alignments is feasible but may require substantially more data than the alignment-based approach.

The experiments were done on a limited set of neural network architectures and, therefore, few conclusions can be drawn about their suitability to the task. Trying more varied network architectures and methods could be a potential topic for future research.

Overfitting was a major issue encountered while training the networks, particularly with the end-to-end model. Methods including dropout and early stopping were employed to mitigate its effects. Other methods such as data augmentation could potentially further improve the generalization performance.

Another major issue, especially with the custom dataset, was the scarcity of the data, and to some extent, the unreliability of the labels. As the agreement level between the annotators was low, many of the samples had to be discarded from the final dataset. Even among the samples in the final dataset, many of the labels were uncertain. Furthermore, there was a notable imbalance in the distribution of the classes in most words.

There are many factors that may have brought about the relatively low agreement levels. For one, annotators were required to choose one syllable as the one bearing the primary stress, or not provide any decision at all. As discussed in Section 2.1, this approach may be insufficient. It is possible that non-native speakers stress multiple syllables in the word with equal prominence. Therefore, letting the annotators label each syllable as stressed or unstressed could lead to a better agreement level, at the cost of increased annotation effort.

It might also make sense to include the secondary stress in the annotation process. Distinguishing between primary stress and secondary stress is a difficult problem, but representing the problem as one of distinguishing between stressed, whether it be primary or secondary, and unstressed syllables, could be a viable option. This is supported by the results on the TIMIT corpus, which showed that the system often had difficulty predicting the location of the primary stress when a secondary stress was present.

The choice of words may also play an essential role. As shown in Section 5.3.1, the results on the custom dataset vary considerably among different words. A respectable level of performance is achieved for some words, given the size of the dataset, while the system fails to produce meaningful results for others. One reason why some words perform better than others may be that stress errors are more common in some words, resulting in a less imbalanced dataset. There are many potential factors affecting the prevalence of stress errors, such as the speaker's unfamiliarity with the word, the existence of similar but differently stressed English words, and the existence of cognates with different stress

patterns in the speaker's native language. Moreover, the realization of the lexical stress could potentially be more natural if it were embedded in a carrier phrase, instead of being spoken in isolation.

6 CONCLUSIONS

The purpose of this thesis was to evaluate various recurrent neural network based methods for recognizing English lexical stress. In particular, the goal was to see whether end-to-end models can provide an improvement over the models using specialized features.

The motivation behind this was that end-to-end systems could allow for a faster and a simpler system. A major practical shortcoming of lexical stress recognition systems using specialized features is that they often depend on external speech recognition systems to find the temporal alignments of the syllables in the audio recording. Such systems are complex in their own right, and may not always work reliably on data from non-native English speakers. End-to-end systems would allow replacing such features with generic audio features, such as Mel spectrograms, that can be efficiently calculated for any audio file.

The experiments were conducted using two datasets. The first dataset contained non-native data, and it was collected specifically for this purpose. The other dataset, the TIMIT corpus, is a dataset of native American English commonly used in speech recognition and other related fields.

The problem was formulated as one of locating the syllable carrying the primary stress among all the syllables in an isolated word. The results suggest that methods based on finding the temporal alignments and calculating syllable-wise features still outperform end-to-end systems. The difference was particularly stark with the non-native data. There were notable differences in the performance among different words. Furthermore, it was discovered that the presence of a secondary stress could lead to decreased performance in recognizing the primary stress.

Annotating audio data with regard to lexical stress turned out to be a more difficult problem than anticipated. During the annotation process, it was found that there was a high degree of ambiguity and uncertainty in the annotations, and as a result, the resulting labels contained a substantial amount of uncertainty. The annotators were restricted to select one syllable as the one carrying the primary stress. While this limitation simplified the annotation process, it may have lead to less trustworthy labels than the alternative of annotating each syllable separately as stressed or not stressed. Additional research on the annotation process would be needed to better understand it.

In the future, these experiments could be expanded upon in various ways. One of the

main limitations of this experiment was the scarcity of data. This could be mitigated by using data augmentation (e.g. [52]). Transfer learning, e.g., from automatic speech recognition systems, could be investigated as another potential way to overcome this limitation. The experiments in this thesis were limited to a small set of recurrent neural network architectures. Different architectures and methods, including convolutional neural networks (see e.g. [70]) and attention based models (see e.g. [4, 11]), could be used in future research.

A different approach that was considered for this thesis, but ultimately left out, would be to formulate the problem as one of sequence labeling, and use methods such as connectionist temporal classification (see [25, 26]). This approach would, in theory, allow for end-to-end lexical stress recognition not just on individual words but on complete phrases.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al. Tensorflow: a system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, 265–283.

[2] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. V. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan and Z. Zhu. Deep Speech 2: end-to-end speech recognition in english and mandarin. *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. event-place: New York, NY, USA. JMLR.org, 2016, 173–182. URL: http://dl.acm.org/citation.cfm?id=3045390.3045410 (visited on 09/27/2019).

[3] A. Aull and V. Zue. Lexical stress determination and its application to large vocabulary speech recognition. *ICASSP '85. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 10. Tampa, FL, USA: Institute of Electrical and Electronics Engineers, 1985, 1549–1552. DOI: 10.1109/ICASSP.1985.1168075.

[4] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel and Y. Bengio. End-to-end attention-based large vocabulary speech recognition. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2016, 4945–4949. DOI: 10.1109/ICASSP.2016.7472618.

[5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13.1 (Feb. 2012), 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2503308.2188395.

[6] N. Campbell and M. Beckman. Stress, prominence, and spectral tilt. *ESCA Tutorial/Research Workshop on Intonation: Theory, Models, Applications*. Athens, Greece, 1997, 67–70.

[7] L.-Y. Chen and J.-S. R. Jang. Stress detection of English words for a CAPT system using word-length dependent GMM-based Bayesian classifiers. *Interdisciplinary Information Sciences* 18.2 (2012), 65–70. ISSN: 1340-9050, 1347-6157. DOI: 10.4036/iis.2012.65.

[8] N. F. Chen and H. Li. Computer-assisted pronunciation training: From pronunciation scoring towards spoken language learning. *2016 Asia-Pacific Signal and Informa-*

tion Processing Association Annual Summit and Conference (APSIPA). Jeju, South Korea: IEEE, Dec. 2016, 1–7. ISBN: 978-988-14768-2-1. DOI: 10.1109/APSIPA.2016.7820782.

[9] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina et al. State-of-the-art speech recognition with sequence-to-sequence models. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, 4774–4778.

[10] F. Chollet et al. *Keras*. 2015. URL: https://keras.io (visited on 09/27/2019).

[11] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho and Y. Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*. 2015, 577–585.

[12] J. Chung, C. Gulcehre, K. Cho and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[13] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems*. 2014, 2933–2941.

[14] O. D. Deshmukh and A. Verma. Nucleus-level clustering for word-independent syllable stress classification. *Speech Communication* 51.12 (Dec. 2009), 1224–1233. ISSN: 0167-6393. DOI: 10.1016/j.specom.2009.06.006. URL: http://dx.doi.org/10.1016/j.specom.2009.06.006 (visited on 10/01/2018).

[15] J. Domokos, A. Stan and M. Giurgiu. An approach to lexical stress detection from transcribed continuous speech using acoustic features. *2014 22nd Telecommunications Forum Telfor (TELFOR)*. Belgrade, Serbia: IEEE, Nov. 2014, 525–528. ISBN: 978-1-4799-6191-7 978-1-4799-6190-0. DOI: 10.1109/TELFOR.2014.7034462.

[16] L. Ferrer, H. Bratt, C. Richey, H. Franco, V. Abrash and K. Precoda. Classification of lexical stress using spectral and prosodic features for computer-assisted language learning systems. *Speech Communication* 69 (2015), 31–45. DOI: 10.1016/j.specom.2015.02.002.

[17] G. Freij and F. Fallside. Lexical stress recognition using hidden Markov models. *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*. New York, NY, USA: IEEE, 1988, 135–138. DOI: 10.1109/ICASSP.1988.196530.

[18] D. B. Fry. Experiments in the perception of stress. *Language and Speech* 1.2 (Apr. 1958), 126–152. ISSN: 0023-8309, 1756-6053. DOI: 10.1177/002383095800100207.

[19] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* 36.4 (1980), 193–202.

[20] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett and N. L. Dahlgren. *DARPA TIMIT acoustic phonetic continuous speech corpus*. NIST, 1993. URL: https://catalog.ldc.upenn.edu/LDC93S1 (visited on 09/27/2018).

[21] X. Glorot, A. Bordes and Y. Bengio. Deep sparse rectifier neural networks. *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, 315–323.

[22] I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (visited on 01/13/2019).

[23] M. Gordon. Disentangling stress and pitch-accent: a typology of prominence at different prosodic levels. *Word Stress*. Ed. by H. van der Hulst. Cambridge: Cambridge University Press, 2014, 83–118. ISBN: 978-1-139-60040-8. DOI: 10.1017/CBO9781139600408.005.

[24] M. Gordon and T. Roettger. Acoustic correlates of word stress: A cross-linguistic survey. *Linguistics Vanguard* 3.1 (2017). DOI: 10.1515/lingvan-2017-0007.

[25] A. Graves. *Supervised sequence labelling with recurrent neural networks*. Vol. 385. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-24796-5 978-3-642-24797-2. DOI: 10.1007/978-3-642-24797-2.

[26] A. Graves, S. Fernández, F. Gomez and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. New York, NY, USA: ACM, 2006, 369–376. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143891.

[27] A. Graves, A.-r. Mohamed and G. Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE international conference on acoustics, speech and signal processing*. 2013, 6645–6649.

[28] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber. LSTM: a search space odyssey. en. *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017). arXiv: 1503.04069, 2222–2232. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2016.2582924.

[29] L. D. Hahn. Primary stress and intelligibility: research to motivate the teaching of suprasegmentals. *TESOL Quarterly* 38.2 (2004), 201–223. ISSN: 0039-8322. DOI: 10.2307/3588378.

[30] W. Hardcastle, J. Laver and F. Gibbon. *The handbook of phonetic sciences*. Blackwell Handbooks in Linguistics. Wiley, 2010. ISBN: 978-1-4051-4590-9.

[31] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597.

[32] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. event-place: Lille, France. JMLR.org, 2015, 448–456. URL: http://dl.acm.org/citation.cfm?id=3045118.3045167 (visited on 09/27/2019).

[33] K. Jenkin and M. Scordilis. Development and comparison of three syllable stress classifiers. *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*. Vol. 2. Philadelphia, PA, USA: IEEE, 1996, 733–736. ISBN: 978-0-7803-3555-4. DOI: 10.1109/ICSLP.1996.607466.

[34] M. I. Jordan. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986, 531–546.

[35] S. Kakouros, O. Räsänen and P. Alku. Evaluation of spectral tilt measures for sentence prominence under different noise conditions. *Interspeech 2017*. ISCA, Aug. 2017, 3211–3215. DOI: 10.21437/Interspeech.2017-1237.

[36] O. Kang, D. Rubin and L. Pickering. Suprasegmental measures of accentedness and judgments of language learner proficiency in oral english. *The Modern Language Journal* 94.4 (2010), 554–566. ISSN: 1540-4781. DOI: 10.1111/j.1540-4781.2010.01091.x.

[37] A. Karpathy. *CS231n convolutional neural networks for visual recognition*. URL: http://cs231n.github.io/neural-networks-1/ (visited on 09/27/2019).

[38] S. Kim, T. Hori and S. Watanabe. Joint CTC-attention based end-to-end speech recognition using multi-task learning. *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2017, 4835–4839.

[39] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[40] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86.11 (1998), 2278–2324.

[41] A. Lengeris. Prosody and second language teaching: lessons from l2 speech perception and production research. *Pragmatics and prosody in English language teaching*. Vol. 15. Dordrecht: Springer, 2012, 25–40. ISBN: 978-94-007-3883-6. DOI: 10.1007/978-94-007-3883-6_3.

[42] K. Li, S. Mao, X. Li, Z. Wu and H. Meng. Automatic lexical stress and pitch accent detection for L2 English speech using multi-distribution deep neural networks. *Speech Communication* 96 (2018), 28–36. DOI: 10.1016/j.specom.2017.11.003.

[43] K. Li and H. Meng. Perceptually-motivated assessment of automatically detected lexical stress in L2 learners' speech. *2012 8th International Symposium on Chinese Spoken Language Processing*. Kowloon Tong, China: IEEE, Dec. 2012, 179–183. DOI: 10.1109/ISCSLP.2012.6423520.

[44] P. Lieberman. Some acoustic correlates of word stress in American English. *The Journal of the Acoustical Society of America* 32.4 (Apr. 1960), 451–454. ISSN: 0001-4966. DOI: 10.1121/1.1908095.

[45] S. L. Mattys. The perception of primary and secondary stress in English. *Perception & psychophysics* 62.2 (2000), 253–265.

[46] W. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5 (1943), 127–147.

[47] B. McFee, V. Lostanlen, M. McVicar, A. Metsai, S. Balke, C. Thomé, C. Raffel, D. Lee, K. Lee, O. Nieto, J. Mason, F. Zalkow, D. Ellis, E. Battenberg, . , R. Yamamoto, R. Bittner, K. Choi, J. Moore, Z. Wei, nullmightybofo, P. Friesch, F.-R. Stöter, D. Hereñú, Thassilo, T. Kim, M. Vollrath, A. Weiss, C. J. Carr and ajweiss-dd. *Librosa*. July 2019. DOI: 10.5281/zenodo.3270922.

[48] J. Morton and W. Jassem. Acoustic correlates of stress. *Language and Speech* 8.3 (July 1965), 159–181. ISSN: 0023-8309. DOI: 10.1177/002383096500800303.

[49] M. J. Munro. Nonsegmental factors in foreign accent: ratings of filtered speech. *Studies in Second Language Acquisition* 17.1 (Mar. 1995), 17–34. ISSN: 1470-1545, 0272-2631. DOI: 10.1017/S0272263100013735.

[50] R. M. Ochshorn and M. Hawkins. *Gentle forced aligner*. URL: http://lowerquality.com/gentle/ (visited on 09/27/2019).

[51] A. O. Okobi. Acoustic correlates of word stress in American English. Thesis. Massachusetts Institute of Technology, 2006. URL: http://dspace.mit.edu/handle/1721.1/37963 (visited on 09/27/2019).

[52] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk and Q. V. Le. SpecAugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779* (2019).

[53] R. Pascanu, T. Mikolov and Y. Bengio. On the difficulty of training recurrent neural networks. *International conference on machine learning*. 2013, 1310–1318.

[54] I. Plag, G. Kunter and M. Schramm. Acoustic correlates of primary and secondary stress in North American English. *Journal of Phonetics* 39.3 (2011), 362–374. DOI: 10.1016/j.wocn.2011.03.004.

[55] M. Reed and J. Levis. *The handbook of English pronunciation*. 1st. Blackwell Handbooks in Linguistics. Wiley-Blackwell, June 2015. ISBN: 978-1-118-31447-0.

[56] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* (1958), 65–386.

[57] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[58] D. E. Rumelhart, G. E. Hinton and R. J. Williams. Learning representations by back-propagating errors. *Nature* 323.6088 (Oct. 1986), 533. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[59] B. J. B. Schmitt. *AMFM decompy*. July 2018. URL: http://bjbschmitt.github.io/AMFM_decompy/ (visited on 09/27/2019).

[60] M. Shahin, J. Epps and B. Ahmed. Automatic classification of lexical stress in English and Arabic languages using deep learning. Sept. 2016, 175–179. DOI: 10.21437/Interspeech.2016-644.

[61] M. Shahin, R. Gutierrez-Osuna and B. Ahmed. Classification of bisyllabic lexical stress patterns in disordered speech using deep learning. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Shanghai: IEEE, Mar. 2016, 6480–6484. ISBN: 978-1-4799-9988-0. DOI: 10.1109/ICASSP.2016.7472925.

[62] R. Skerry-Ryan, E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. J. Weiss, R. Clark and R. A. Saurous. Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. *arXiv preprint arXiv:1803.09047* (2018).

[63] A. M. C. Sluijter and V. J. van Heuven. Acoustic correlates of linguistic stress and accent in Dutch and American English. *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*. Vol. 2. Philadelphia, PA, USA: IEEE, 1996, 630–633. ISBN: 978-0-7803-3555-4. DOI: 10.1109/ICSLP.1996.607440.

[64] A. M. C. Sluijter and V. J. van Heuven. Spectral balance as an acoustic correlate of linguistic stress. *The Journal of the Acoustical Society of America* 100.4 (Oct. 1996), 2471–2485. ISSN: 0001-4966. DOI: 10.1121/1.417955.

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

[66] J. Tepperman and S. Narayanan. Automatic syllable stress detection using prosodic features for pronunciation evaluation of language learners. *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.* Vol. 1. Philadelphia, Pennsylvania, USA: IEEE, 2005, 937–940. ISBN: 978-0-7803-8874-1. DOI: 10.1109/ICASSP.2005.1415269.

[67] *The Carnegie Mellon University pronouncing dictionary*. URL: http://www.speech.cs.cmu.edu/cgi-bin/cmudict (visited on 09/27/2019).

[68] A. Verma, K. Lal, Yuen Yee Lo and J. Basak. Word independent model for syllable stress evaluation. *2006 IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*. Vol. 1. Toulouse, France: IEEE, 2006, I–1237–I–1240. ISBN: 978-1-4244-0469-8. DOI: 10.1109/ICASSP.2006.1660251.

[69] A. Waibel. Recognition of lexical stress in a continuous speech understanding system - A pattern recognition approach. *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 11. Tokyo, Japan: Institute of Electrical and Electronics Engineers, 1986, 2287–2290. DOI: 10.1109/ICASSP.1986.1168788.

[70] Y. Wang, X. Deng, S. Pu and Z. Huang. Residual convolutional CTC networks for automatic speech recognition. *arXiv preprint arXiv:1702.07793* (2017).

[71] P. J. Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences. PhD thesis. Harvard University, 1974.

[72] C. Yarra, O. D. Deshmukh and P. K. Ghosh. Automatic detection of syllable stress using sonority based prominence features for pronunciation evaluation. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA: IEEE, Mar. 2017, 5845–5849. ISBN: 978-1-5090-4117-6. DOI: 10.1109/ICASSP.2017.7953277.

[73] G. Ying, L. Jamieson, Ruxin Chen, C. Michell and Hsin Liu. Lexical stress detection on stress-minimal word pairs. *Proceeding of Fourth International Conference on Spoken Language Processing ICSLP 96 ICSLP-96*. Philadelphia, PA, USA: IEEE,

1996, 1612–1615 vol.3. ISBN: 978-0-7803-3555-4. DOI: 10.1109/ICSLP.1996.607932.

[74] S. A. Zahorian, P. Dikshit and H. Hu. A spectral-temporal method for pitch tracking. *Ninth International Conference on Spoken Language Processing*. 2006.

[75] J. Zhao, H. Yuan, J. Liu and S. Xia. Automatic lexical stress detection using acoustic features for computer assisted language learning. *Proc. APSIPA ASC* (2011), 247–251.