Bilingual End-to-End Deep Learning Speech Recognition System

Nikola Tonev

15119311

BSc (Hons) Sound Engineering and Production

Supervisor: Jason Hockman

May 2019

ABSTRACT

Automatic speech recognition refers to the task of converting raw speech into a human-readable text by a computer. It has been a heavily researched topic for many years and researchers have come up with various approaches to the problem, all building upon each other. This project focuses on one of the most modern approaches used by companies like Google and Apple – deep learning. The main aim of the project is to develop, test and evaluate an end-to-end deep learning system that has the ability to recognize both English and Bulgarian speech with no changes in the model and no need for specialist language knowledge, inspired by Baidu's Deep Speech 2.

The model consists of two convolutional neural network (CNN) layers, followed by three bi-directional recurrent neural network (RNN) layers, followed by a fully connected layer. The output of the fully connected layer is fed into a Connectionist Temporal Classification (CTC) layer.

The data used for training and testing was collected from three commercial data sets – LibriSpeech, BulPhonC and Google Speech Commands. The training process was split into three parts – large-vocabulary training, small-vocabulary training and single file overfitting.

The training processes resulted in label error rates (LER) of 67% for large-vocabulary training on LibriSpeech, 74% for small-vocabulary training on Google Speech Commands and 1.23% and 0% for single file overfitting respectively on LibriSpeech and BulPhonC audio files. Important relationships were observed between certain hyper-parameters, the quantity and distribution of the training data and the performance of the system.

The results from this study show that the system has the ability to adapt to input data successfully in both languages and has the potential for larger-scale training and recognition in the presence of powerful enough computational resources and longer training times.

ACKNOWLEDGEMENTS

“identifying those from whom assistance has been received.” (General Guidance)

Example One

I would like to thank all the staff from the Centre for Academic Success, Brian Fanning at CA Associates, Sasha Johnson, John Kempson, and Kate Chandler. I should also like to thank: Julie, Tracy, Maria, Meredith, Charlie, Kaz, Geoff Phillips, Ian Urguhart, Kit Wallace, Luke Brennan, Garry Lunn, Richard Cliff, Mum, Dad, Ginnie and everyone else who helped.

CONTENTS PAGE

GLOSSARY

Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
ARSG	Attention-Based Recurrent Sequence Generator
ASR	Automatic Speech Recognition
BiRNN	Bi-Directional Recurrent Neural Network
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CTC	Connectionist Temporal Classification
DBN	Deeb Belief Network
DCT	Discrete Cosine Transform
DNN	Deep Neural Network
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
LER	Label Error Rate
LSTM	Long Short-Term Memory
LVSCR	Large-Vocabulary Continuous Speech Recognition
MFC	Mel Frequency Cepstrum

MSE	Mean Square Error
OS	Operating System
PC	Personal Computer
RAM	Random Access Memory
RMSProp	Root Mean Square Prop
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
STT	Speech-to-Text
WER	Word Error Rate

LIST OF FIGURES AND TABLES

List of Figures

List of Tables

1.0 INTRODUCTION

Speech recognition is the technology that allows a computer of any kind to capture human speech in the form of a sound wave and convert it into written text. Also known as Automatic Speech Recognition (ASR) or Speech-to-Text (STT), this technology can have various sets of characteristics that depend on the purpose and needs that the system is built to serve. For example, such systems can be large-vocabulary for general purpose speech recognition or small-vocabulary for a specific application where only a certain set of commands is being used. Also, they can be speaker-dependent or speaker-independent based on whether the system was intended to be used respectively by a single user or by the wider public.

Despite being one of the most natural and simple functionalities in people, ASR has always been a task of high complexity for a computer, due to the unpredictability and variance of speech. As a result of many years of extensive research and experimentation, numerous different approaches to ASR exist, some of which will be explored and discussed in greater detail in the following chapters.

1.1 Problem Definition

For many years the development of functional speech recognition systems relied on specialized knowledge of the language of interest and all of its specifics, requiring the recruitment of linguists and other specialists to build accurate acoustic and language models. All grammatical rules, pronunciation specifics, etc. had to be hardcoded in the system, which was time consuming, made those systems large in size, therefore impractical to incorporate in mobile applications, and inflexible to different pronunciations and dialects and unusable on another language.

In 2014, Alex Graves and his team introduce the innovative idea of end-to-end speech recognition deep learning model which would automatically learn the pronunciation and acoustic models and would remove the need for specialized language knowledge and hand modelling, given enough training examples (Graves et al., 2014). Shortly after that, Baidu Labs created the Deep Speech (1 & 2) system to build on Graves' idea and prove that an ASR system can be trained to recognize two different languages with little or no changes in the system (Amodei et al., 2016).

This project will explore the essence of end-to-end speech recognition and will take the idea of Deep Speech further to demonstrate that a single deep learning model can be trained to recognize and transcribe both English and Bulgarian language, regardless of the significant differences in the two languages, including the use of different alphabets (Latin and Cyrillic).

1.2 Scope

Over the provided seven-month period for the completion of this project, three main stages can be identified – research of the existing technologies, design of the proposed model and implementation and evaluation of the system.

In the research stage, the project briefly reviews the history and advancement of speech recognition, evaluates the strengths and weaknesses of each approach and justifies the choice of the end-to-end approach to ASR. At this stage, an overview of the basics of machine learning and its relevant areas is presented as well.

The relevant sections to the design stage present a proposed architecture for a large-vocabulary, speaker independent deep learning model which has the ability to convert both English and Bulgarian speech into text. All of the model's components and their functionality are described in details.

The sections on the implementation and evaluation proceed to present the process of implementing of the system, including the hardware and software tools and resources used for the completion of the project. All testing scenarios and results are laid out and a critical discussion of the project results and limitations is carried out.

Finally, a short conclusion summarises the key findings and proposes possible further improvements to the system.

1.3 Rationale

With the rapid evolution of smart technologies nowadays and specifically the AI and IoT, which are seen as two of the focus areas of development in what is called The Fourth Industrial Revolution, the demand for more advanced and accurate speech recognition algorithms and systems is growing exponentially.

For people with various severe physical impairments, speech recognition is a new opportunity for easier and better communication and interaction with the modern world. The choice of this topic was motivated by the creation of Baidu's Deep Speech 2 system as well as the increasing interest and necessity for speech recognition systems in the modern society. Nowadays, technology takes an increasingly big part of people's everyday lives and communicating efficiently with it is crucial.

1.4 Aims

The overall aim of this project is to develop and implement a bilingual end-to-end deep learning ASR system, based on Baidu's Deep Speech 2 system, which will work in English and Bulgarian.

1.5 Objectives

- Research the history and fundamentals of speech recognition.
- Examine and contrast all existing algorithms and methods for achieving speech recognition and justify the final choice of approach.
- Design a model architecture for the ASR system.
- Obtain speech corpora in both languages for training and testing.
- Formulate a methodology and develop a suitable application based on the available resources.
- Test the system with different configurations and sets of hyper-parameters and evaluate the label error rate and efficiency of the designed system.
- Provide statistical analysis of the chosen training and testing data.
- Critically evaluate the results and project limitations.
- Propose further improvements to the system.

2.0 REVIEW OF EXISTING KNOWLEDGE

2.1 Speech Recognition – History and Importance

Automatic speech recognition has been an area of interest since the 1950s. Initially, researchers were exploiting the acoustic-phonetic features characterising speech sounds. The first attempt for creating an ASR system was made in Bell Laboratories in 1952 by Davis, Biddulph and Balashek. They created an isolated digit recognizer that operated by analysing the spectral content in the vowel region of each digit in order to detect specific spectral resonances (Davis et al, 1952).

By the mid-1970s, the new basic concept of pattern recognition was introduced. It was based on the newly developed methods for spectral analysis – Linear Predictive Coding (Itakura and Saito, 1970; Atal and Hanauer, 1971).

In the 80's, template-based pattern recognition, which in its essence did not incorporate any automatic learning, was quickly replaced by the newer concept of the statistical-based, probabilistic modelling. In the following three decades, most ASR systems were relying mainly on Hidden Markov Models (HMM) combined with Gaussian Mixture Models (GMM). During that time, the concept of artificial neural networks (ANN) was already introduced and used in many systems (Waibel et al., 1989; Bourlard and Wellekens, 1989; Bengio et al., 1991, 1992; Robinson and Fallside, 1991; Konig et al., 1996). However, it did not gain much popularity among researchers, due to its performance being similar to the one of HMM-GMM models, and were only used for separate small tasks inside the HMM-GMM model.

Over the years, researchers slowly started gaining more interest in deep learning models after their successful implementation in numerous applications and once the performance of GMM-HMM models reached a plateau regardless of the growing data sets, the focus of researchers quickly shifted onto deep learning (Goodfellow et al., 2017).

Over the years, researchers have overcome many limitations like speech segmentation, temporal non-uniformity, background noise, etc. However, ASR still faces various imperfections and therefore remains a research topic of high interest. The following section will explore the evolution of deep learning for speech recognition over the years in greater depth through critical analysis of the existing research.

2.2 Critical Review of Previous Research

Baker (2009) states that arguably the most important advancement in the historical progress of speech recognition systems was the introduction of stochastic processing with HMMs as part of the acoustic model in the new for the time statistical approach to ASR that deals with the temporal variability of speech (Baker, J. K., 1975; Jelinek, F., 1976). *Figure 1* shows a high level representation of the typical statistical speech recognition system. With the introduction of the expectation maximization (EM) algorithm (Dempster et al., 1977), training HMMs for real-world speech recognition applications became possible. The EM algorithm uses GMMs to form a probabilistic representation of the relationship between acoustic input and the HMM as a mixture of Gaussian probability distributions.

However, Hinton et al. (2012) explain that despite the many advantages of GMMs, they are statistically inefficient when it comes to modeling data that lies close to or on a nonlinear manifold in the data space. He argues that the underlying structure of speech has lower dimensionality than initially observed in a window with numerous coefficients. Models with the ability to exploit embedded information of lower dimensionality in a big window of frames would operate more efficiently on such data. When trained discriminatively by backpropagating error derivatives, artificial neural networks (specifically feed-forward deep neural networks) can process such manifold data much more efficiently, given enough computational power. In their paper, Hinton and researchers from four prestigious organizations explain and demonstrate a two-stage procedure for training a deep neural network (DNN) for acoustic modeling. They tested both the GMM and the DNN training procedures on six different large-vocabulary tasks (hours of training data ranging from 24 to 5870) and found the DNN-HMM model's performance to be equal to or exceeding the one of highly tuned GMM-HMM even in cases where the latter is trained with much more data.

The major conclusion from their study is that applying purely generative pretraining as a the first stage would prevent overfitting and speed up the discriminative fine-tuning process later by training each feature detection layer based on its preceding layer using Deep belief networks (DBN), instead of designing all feature detectors at once and by hand. The DBN would initialize weight values that are close to the good global solution and the discriminative process of back-propagation through the DNN would only need to adjust the weights very slightly (Mohamed et al., 2011; Mohamed et al. 2012).

In the second stage of fine tuning, however, DNNs have the disadvantage of lacking the ability to use parallelism, which can slow down the process as data sets grow larger.

Even though different kinds of neural networks have replaced many key components of the typical complex ASR system (*see Fig. 1*) over the years, some of the base components have remained the same for decades and most of the existing systems are HMM-based. In traditional speech recognition systems, features are extracted from the raw input acoustic waveform and a neural network is trained to classify frames of this acoustic data, the output distributions of which are then transformed into emission probabilities for the HMM. Such systems also have to include a language and a pronunciation model, specifically designed for the language of interest, in order to predict an output sequence.

In their paper, Graves and Jaitly (2014) propose a new and simplified end-to-end approach to speech recognition systems that directly transcribes audio data with little or no hand-engineered language and pronunciation modelling, removing the necessity for an intermediate phonetic representation. They argue that existing HMM-based ASR systems require significant human expertise and complex modeling which can often slow down the development and training process and could be avoided by this end-to-end approach.

Despite the fact that end-to-end speech recognition can be achieved without any preprocessing of the raw input waveform (Graves, 2012), it proved to significantly decrease the system's performance and add to the computational cost of the model. Graves and Jaitly (2014), therefore, decided to use spectral audio representation as a minimal preprocessing. Their proposed system combines a deep bi-directional Long short-term memory (LSTM) recurrent neural network (RNN), which processes the spectrograms, with a Connectionist Temporal Classification (CTC) output layer which deals with aligning input data to text transcripts of different sizes and training the RNN model. Decoding is performed with a beam search algorithm. To improve the decoding speed and efficiency, Miao et al. (2015) suggest embedding individual decoding components (language models, lexicons and CTC labels) into weighted finite-state transducers.

The findings of this study show that the proposed system works successfully even in the absence of a language model, provided it is trained with a large data set. However, using simple language models helps reducing the word error rate (WER) even further. Using a trigram language model brings the WER down to 8.2% from the original 27.3% obtained from training without any prior linguistic information (Graves and Jaitly, 2014).

Maas et al. (2015) suggest a further improvement to Graves and Jaitly's system by performing character-level decoding of the RNN outputs as opposed to word-level ones, which give the system the ability to transcribe new words and fully eliminates the need for a lexicon.

In their paper, explaining Deep Speech – a similar end-to-end speech recognition system, Hannun et al. (2014) outline that traditional speech recognition systems tend to show poor performance in noisy and non-isolated environments, unless they incorporate specifically designed components to model such interferences. An end-to-end deep learning system, on the other hand, can learn and adjust itself to effects like reverberation, background noise or speaker variations without the need for such components, given that a large labeled training set, which includes enough examples with the above-mentioned effects, is provided. While it is possible to collect and label large amounts of data containing the distortions of interest, it is extremely inefficient in practice and it can take incomprehensible amounts of time and resources. As an alternative, Hannun et al. suggest generating the necessary data by modifying the existing "clean" data set. This process, also called superposition of the source signals, consists of adding a noise track $\xi^{(i)}$ to the original signal $x^{(i)}$ in order to simulate audio captured in a noisy environment without having to physically record it. Further processing can be applied as well like adding echoes, reverberations, delay or other effects that need to be handled by the system.

It is important to note, however, that for synthesizing noisy speech tracks from *n* hours of clean recordings, roughly *n* hours of non-repetitive noise recordings will be necessary as well. If the noise track is repeating numerous times over the training set, there is a risk for the RNN to memorize it and subtract it out of the newly synthesised track. For the purpose, Hannun suggests collecting and using many shorter audio clips, which can be easily extracted from public video sources, instead of sourcing an *n*-hour long one.

The results from the Deep Speech system showed a great improvement in noisy speech recognition, compared to various popular systems with 19.06% WER on noisy speech as opposed to 43.76% WER for Apple Dictation. For a combination of noisy and clean speech, the Deep Speech system performed with only 11.85% WER, compared to 26.73% for Apple Dictation. This proves that superposition of the training signals as a means for generating noisy speech to add to the training set leads to significantly better performance of the system, compared to hand-engineering those distortions in the sound.

Amodei et al. (2016) take the idea of the end-to-end deep learning approach further by developing Deep Speech 2 in order to demonstrate how such a system can recognize two entirely different languages with little or no expert knowledge on any of the languages. The architecture of this system resembles the one of the original Deep Speech – 20ms windows of spectral content serves as an input for a recurrent neural network, having 1-3 convolutional input layers and a number of uni/bi-directional layers, followed by a fully connected and a softmax layer (consisting of the alphabet of the respective language). The end-to-end training is achieved by using the CTC loss function. The CTC model is combined with a language model and beam search is used for estimating the most probable transcription.

However, for the RNN units Amodei et al. use gated recurrent units (GRUs) instead of LSTMs. Introduced in 2014 (Cho et al.), GRUs have the same concept as LSTMs. However, GRUs have a simpler structure with less parameters and do not include a memory cell, which makes them much faster to train (Chung et al., 2014).

To further improve the performance of the system, Amodei et al. apply batch normalization throughout the whole network and SortaGrad to the training set – a technique for classifying minibatches of training examples in terms of difficulty, which is determined by the length of the longest utterance in a minibatch.

Like in (Hannun et al., 2014), the training set is a combination of collected data and an augmented version of it with added noise and effects. The results of the training show a 40% relative decrease in WER for each factor of ten increase in the size of the training set.

From the results it can be concluded that 2D convolutional layers provide a significant improvement in the WER for noisy data, as opposed to 1D layers that provide only a small benefit both for the clean and the noisy data. The transition from a single 1D to three 2D convolutional layers showed improvement of 23.9% in the WER on noisy data.

The adaptation to Mandarin required minimal changes, all depending on the Chinese character set. Instead of outputting probabilities for 29 characters (letters a-z, space, apostrophe, blank), the network outputs probabilities for 6000 characters. Then a character level language model is used, since Mandarin is not segmented by words in text. Similarly, the system that recognizes Bulgarian language will output probabilities for 32 characters (30 letters from the Cyrillic alphabet, space, blank). Either word or character level language model can be used, based on availability.

An innovative alternative for sequence-to-sequence encoding-decoding to CTC was introduced recently, called attention mechanism. First applied in the fields of machine translation (Bahdanau et al., 2015), visual object classification (Mnih et al., 2014), image caption generation (Xu et al., 2015), etc., the attention mechanism was later applied to speech recognition as well (Chorowski et al., 2015). It provides the output sequence generator in a network the ability to access any part of the input sequence, instead of a compressed version of the whole input in the form of a single vector. Thus, the model can focus its attention to a specific part of the input signal based on a set of weights associated with each element of the input sequence.

In their work on an end-to-end attention-based LVSCR system, Bahdanau, et al. (2016) use an Attention-Based Recurrent Sequence Generator (ARSG) as a decoder network. The ARSG outputs a sequence of elements ($y_1,\ldots, y_t$) generated from the sequence ($h_1,\ldots,h_n$), which is just the encoded representation of the input signal x. The ARSG is composed of an RNN and an attention mechanism which selects specific locations from the input sequence and use them to update the hidden state of the neural network based on their attention weights and to estimate the following output value.

The results from this experiment (Bahdanau et al, 2016) show that attention-based systems perform better than CTC systems in the absence of an external language model with 18.6% WER for ARSG compared to 35.8% WER for a CTC system (Hannun et al., 2014). However, when an external language model is used, CTC systems still outperform attention-based ones. In the presence of an extended trigram language model, the ARSG system produces 9.3% WER while a CTC system produces 7.3% WER (Miao et al., 2015). Therefore, the choice of model type should be made based on the language model availability.

In the past few years, most of the research in this field has been centered around the idea of end-to-end systems. Many researchers keep developing the attention-based approach (Toshniwal et al., 2017; Kim and Seltzer, 2017), others still work with CTC systems (Liptchinsky et al., 2017; Zeghidour et al., 2018) and there are a few research teams looking into hybrid CTC/attention-based systems (Hori et al., 2018) .

This project will aim for the implementation of the CTC approach, similar to the one proposed by Amodei et al. (2016) due to its established success and the wide range of existing research on the topic.

The following section will provide a brief overview of the base theory behind machine learning, necessary for the understanding of deep learning systems.

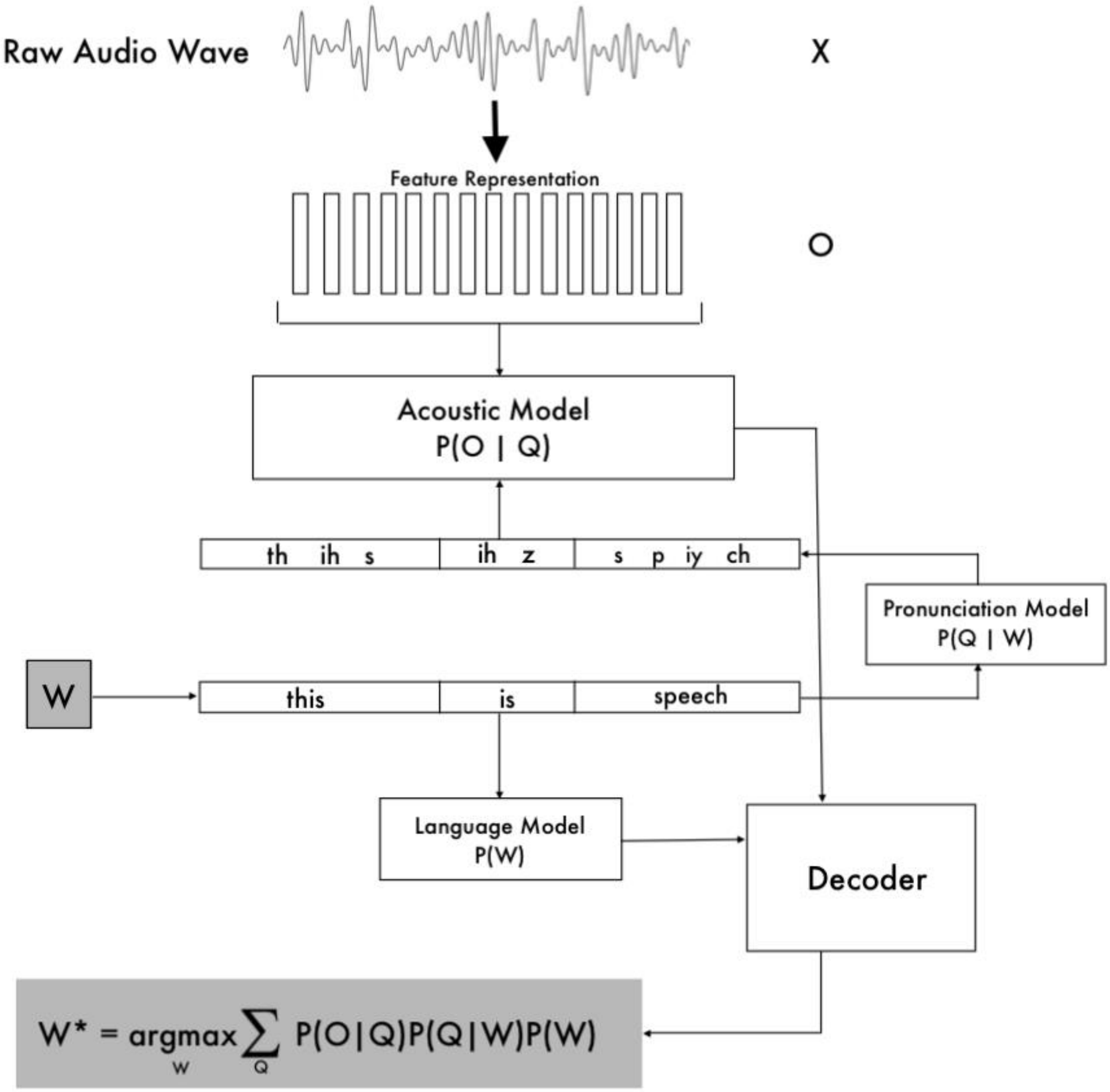


Figure 1: A high level statistical speech recognition system diagram, showing the computation of the maximum value of the probability P(W|X) of word sequence W given the acoustic input X. The diagram was composed based on a similar one provided by Steve Young (1996) in 'Large Vocabulary Continuous Speech Recognition: a Review' and the lecture slides from Adam Coates' Speech recognition and Deep Learning presentation in Stanford.

2.3 Machine Learning

2.3.1 Basics

Machine learning is a subfield of artificial intelligence that aims to replace the explicit programming of a system with automatic acquisition of knowledge based on a given set of data or as Mitchell (1997) describes it, it is the improvement in performance P with experience E on a task T.

Depending on the type of the training data set, the following 3 main types of machine learning algorithms can be identified:

1. **Supervised Learning Algorithms**
   An algorithm is considered supervised when all examples from the training data set are associated with their respective expected outputs, also called label. Given the vector *x* and its associated vector *y*, the algorithm attempts to learn how to predict unknown values for *y* from a given *x* by estimating the conditional probability distribution p(*y* | *x*) by using maximum likelihood estimation. Linear regression is considered the simplest supervised learning algorithm.

2. **Unsupervised Learning Algorithms**
   An algorithm is considered unsupervised when the training set contains elements with many features, but no labels associated. The model is expected to identify common patterns and useful properties in the structure of the training data, without having any expected output values. In other words, the algorithm attempts to determine the probability distribution p(***x***) or a distinctive property of it given a random vector *x*.
   The k-means clustering is a popular unsupervised learning algorithm, which divides the training data set into a number of clusters of similar data points (MacQueen, 1967).

3. **Representation algorithms**
   A representation algorithm is not given a fixed data set to work with. Instead, it experiences an interaction with its environment and receives a feedback after each of its actions as a form of a reward. The purpose of a representation algorithm is to learn to perform in way that will maximize the total rewards signal over a certain amount of iterations (Sutton and Barto, 1998).

2.3.2 Neural Networks

2.3.2.1 Artificial Neural Networks

An ANN is simply a network of nodes structured in layers – usually an input layer, one or more hidden layers and an output layer. Each hidden layer consists of units that perform nonlinear operations to the input signal:

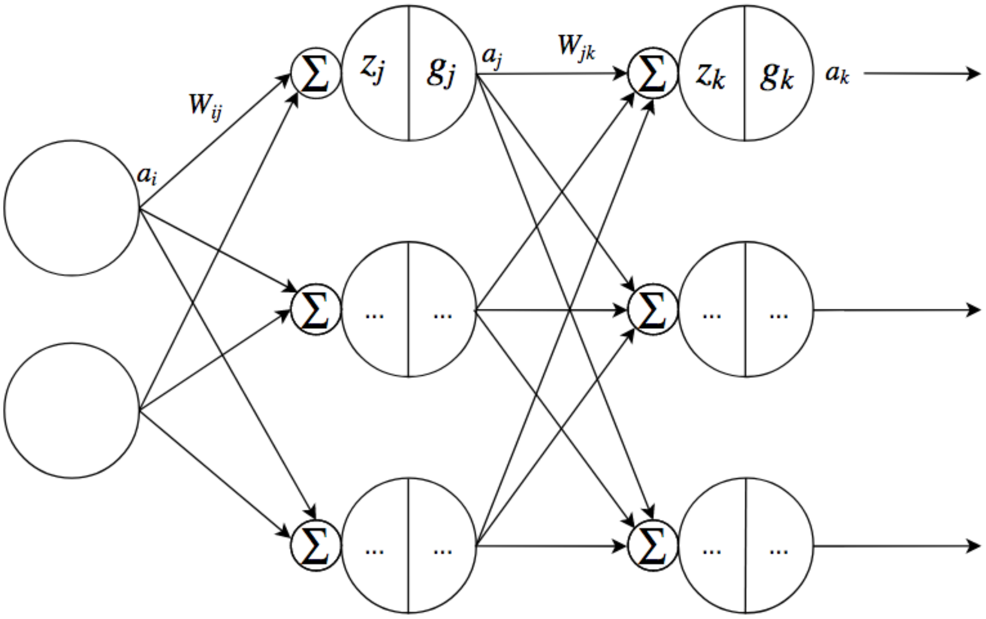


Figure 2: A generic artificial neural network structure.

where

$$a_k = \overbrace{g_k(\overbrace{b_k + \sum\nolimits_j \underbrace{g_j\left(\underbrace{b_j + \sum\nolimits_i a_i W_{ij}}_{z_j}\right)}_{a_j} W_{jk})}^{z_k}}^{a_k}$$

In a generic feed-forward artificial neural network, the following calculations can be observed:

I. The input signal a_i for each node in the input layer is multiplied by a weight matrix W_{ij} which connects the input layer with the hidden layer.
II. Those products are then summed and a bias value is added. This operation forms the input of each node of the hidden layer z_j .
III. The input of each hidden node is then multiplied by the layer's nonlinear activation function g_j to produce the output of the layer - a_j .
IV. Similar to the previous steps, the output of each hidden node is multiplied by the weight matrix W_{jk} , a bias b_k is added and the signals are summed to produce the input of the output node – z_k **.**
V. Z_k is then multiplied by its activation function g_k to produce the neural network's output values $a_{k..}$

Artificial neural networks are used for modelling very complex relationships in data that occur in tasks like image labelling, handwriting recognition or speech recognition. They achieve this by constantly updating their internal parameters in order to fit to the input data.

A deep neural network is simply an artificial neural network with multiple hidden layers. There is no formal definition for the number of layers beyond which a neural network is considered deep.

2.3.2.2 Recurrent Neural Networks

Recurrent neural networks are a subset of neural networks that have at least one recurrent connection in their hidden layers, meaning that each hidden node takes as inputs not only the current input value (x_t), but also the value of the hidden node for the previous time step (h_{t-1}) in order to produce an output value (*see Eq. 1 & Fig. 3*).

$$h_t = Ux_t + Wh_{t-1} + b \qquad (1)$$

where *U* and *W* are the respective weight matrices and *b* is the bias matrix.

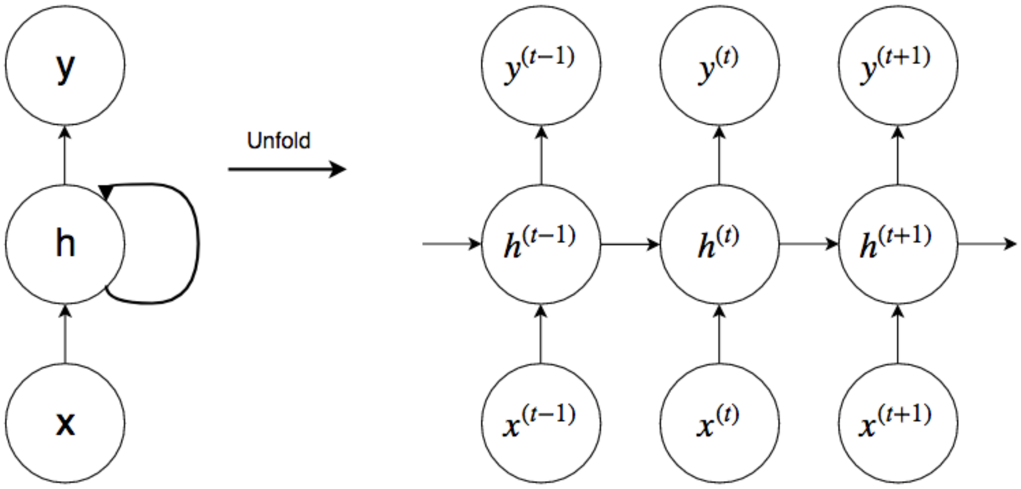


Figure 3: A generic recurrent neural network structure with one hidden layer

2.3.2.3 Convolutional Neural Networks

Convolutional neural networks are a special type of ANN which is used for processing data with grid-like structure. Their main objective is to extract high-level features from given data (most widely used for image processing). As the name indicates, CNNs make use of the linear mathematical operation called convolution (add info in appendix), as opposed to the general matrix multiplication used in regular ANNs. A kernel (i.e. a small window of values) of certain size is being convolved with different sections of the input structure to produce an element of the output matrix. CNN makes use of the so called weight sharing, meaning that the same set of weight values (in the kernel) are used for all sections of the input, thus preventing overfitting and reducing the number of weights to learn, which contributes to the model's robustness (Abdel-Hamid et al., 2014; *see Fig. 4*).

If no padding is added to the inputs, such networks reduce the dimensionality of the input data structure. Alternatively, 'same' padding can be applied in order to keep the same dimensionality from the input to the output (*see Fig. 5*).

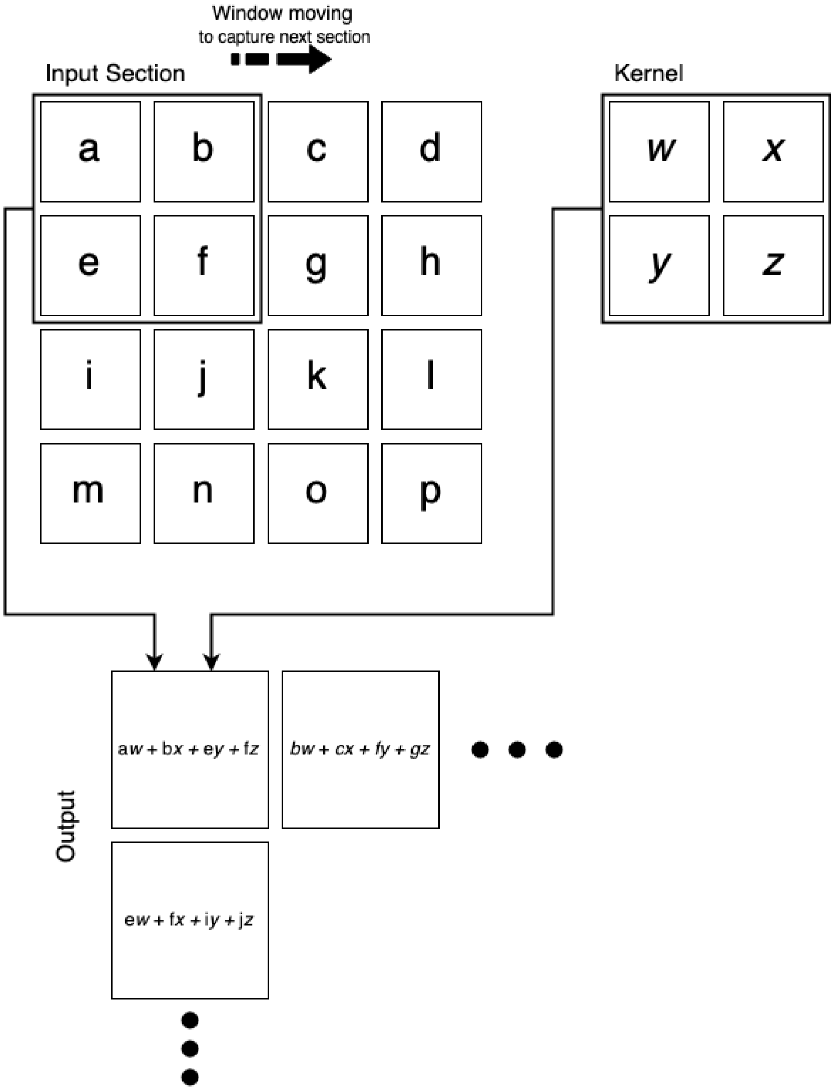


Figure 4: A generic example of 2-D convolution with no padding and kernel of shape [2, 2].

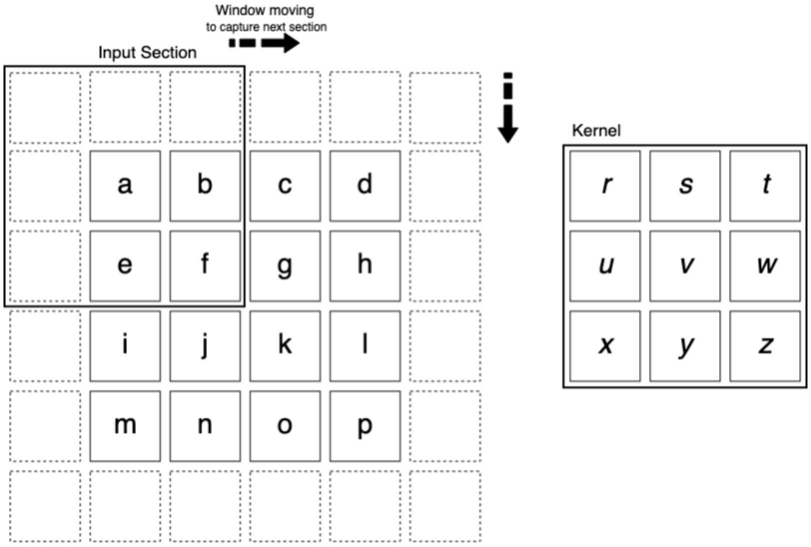


Figure 5: Illustration of 'same' padding. The output matrix will have the same dimensionality as the input one - 4x4.

2.3.2.4 Training the Neural Network

Training a neural network typically consists of two main steps – calculating the cost function and minimizing it.

The cost function (also known as *loss*) is simply a measure of how accurate the model is when predicting. A higher cost value means lower accuracy of prediction. The cost is expressed as the distance (i.e. difference) between the predicted output value and the actual value provided as part of the training data. The most widespread cost function is the Mean Square Error (MSE; *see Eq. 2*).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \qquad (2)$$

To minimize the cost function, an optimization algorithm is introduced, which adjusts the parameters (weight matrices) inside the neural network until the cost function reaches its lowest value. The most basic and most popular optimization algorithm is the gradient descent. It calculates the gradient of the function for a set of parameters with respect to the parameter being updated, multiplies it by the learning rate and subtracts it from the parameter's value. Thus, if the gradient is positive, the parameter will be updated to a smaller value, taking it closer to the function minimum and if the gradient is negative, the parameter value is increased. This process is repeated until all gradients converge to 0 and the cost function reaches its minimum (*see Eq. 3*).

$$\theta_j = \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) \qquad (3)$$

where α is the learning rate and θ_j is the parameter being updated. The learning rate controls the size of each step of descent, thus controlling the speed of the learning process.

3.0 METHODOLOGY

3.1 Introduction to the Methodology

The following section will present the chosen design, implementation, testing and evaluation methodologies and the reasoning behind those choices. Elements such as data sets choice and collection, software and hardware resources will be discussed as well.

This project undertakes the end-to-end deep learning approach to automatic speech recognition. As discussed in the previous section, this approach provides a simplified way of transcribing audio data without many of the complex intermediate components of the traditional ASR system. One of the main reasons behind the choice of this approach is that it is not relying on much hand-engineering and expert knowledge in the languages of interest, which makes it the most efficient approach to multi-lingual speech recognition. This is particularly important for developing a system that can transcribe audio with little or no language-specific modelling in both English and Bulgarian, which are two very different languages that even use different alphabets.

3.2 Literature Search Methodology

A broad range of physical and online literature sources were used for the retrieval of the literature referenced in this report. Those include the BCU-provided *UCEEL Digital Library* and the library located in Curzon building in the City Centre campus, as well as external digital libraries such as *IEEE Xplore, Cornell University's online archive (arxiv.org), ResearchGate* and *ScienceDirect,* which offer hundreds of accredited scientific papers, peer-reviewed academic journals , e-books, technical reports, etc. Google's excellent search engine *Google Scholar* was used to simultaneously search all of those libraries and databases.

All of the obtained reference sources, together with highlights and notes, are stored in Zotero.

The following list presents key terms and phrases was used for searching the above-mentioned sources:

- "speech recognition"
- "automatic speech recognition"
- "end-to-end speech recognition"
- "end-to-end deep learning"
- "attention-based speech recognition"
- "multi-lingual speech recognition"
- "low-resource language speech recognition"
- "voice recognition"
- "voice control"
- "voice commands"

3.3 Software and Hardware Resources

3.3.1 Programming Language and Libraries

The programming language used for this project was Python, being the most widely used programming language in machine learning. Python is an interpreted, high-level programming language, released back in 1991. It is a powerful language that prioritizes code readability and simplicity in order to allow the developer to focus on the development of complex algorithms.

The main reason for choosing Python was the availability of a wide variety of libraries and modules for it aimed towards solving machine learning problems.

TensorFlow is an open-source library for Python, providing tools for performing complex numerical computation. Being developed by Google, TensorFlow is a strongly supported, solid and up to date library. TensorFlow provides very powerful low level programming for modelling specific and unique algorithms. This library was used extensively for the development of this project.

For the available alternative libraries and their comparison, please see *Table 1*.

For more details on the software resources used, please see *Table 2.*

Table 1: Pros and Cons of the most popular Python libraries used for Machine Learning.

Library	**Pros**	**Cons**
TensorFlow	▪ Provides good computational graph visualizations (via TensorBoard). ▪ Programs can be run both on CPU and GPU without having to write at C++ and CUDA level. ▪ Powerful ▪ Frequent updates.	▪ Complex structure. ▪ Steep learning curve. ▪ Only supporting Nvidia GPUs. ▪ Incomplete documentation.
Scikit-learn	▪ Simple to use for developers at all levels ▪ Efficient for simple data analysis tasks.	▪ Not efficient for complex deep learning models. ▪ No visualization tools.
Keras	▪ Fast and easy neural network prototyping. ▪ Extensive documentation. ▪ Capable of running on top of TF/Theano ▪ Suitable for beginners.	▪ Dependent on TensorFlow and Theano. ▪ No focus on other ML algorithms, rather than deep learning ones.
Theano	▪ Takes advantage of the computer's GPU. ▪ A lot of functionality due to being old. ▪ Wider applications (than deep learning).	▪ Functionality can be overcomplicated. ▪ Use of single GPU only. ▪ Lower performance.

Table 2: Details of the software resources used for the development of this project.

Programming Language	Python 3.6.7/3.7.1
External Libraries/Packages	▪ **TensorFlow** 1.13.0-rc1 ▪ **NumPy** 1.15.4 ▪ **SciPy** 1.1.0 ▪ **SoundFile** 0.10.2 ▪ **tqdm** 4.31.1
Internal Libraries/Packages	▪ os ▪ time ▪ shutil ▪ pickle ▪ argparse
Text Editor	SublimeText 3.1.1
OS(s)	▪ MacOS Mojave 10.14 ▪ Ubuntu 18.04
Server Connection	▪ Tunnelblick 3.7.8 (OpenVPN client-server connection interface) ▪ VNC Viewer 6.19.107 (graphical desktop sharing)

3.3.2 Hardware

For training the model a remote server computer in Bulgaria was used. The PC has the following parameters:

Table 3: Specifications of the server computer.

Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU	Intel(R) Core(TM) i7-2600 @ 3.40GHz
On-line CPU(s) list	0-7
Thread(s) per core	2
Core(s) per socket	4
Socket(s)	1
Total thread(s)	8
CPU MHz	3800.00
RAM	16 GB @ 1600 MHz
OS	(Linux) Ubuntu 18.04

The server was only used for uninterrupted training and not for everyday use. During the course of the project, the RAM was upgraded from 2x4 GB to 2x8 GB and the OS was reinstalled from CentOS 7 to Ubuntu 18.04 due to problems with TensorFlow installation.

3.4 Model Architecture

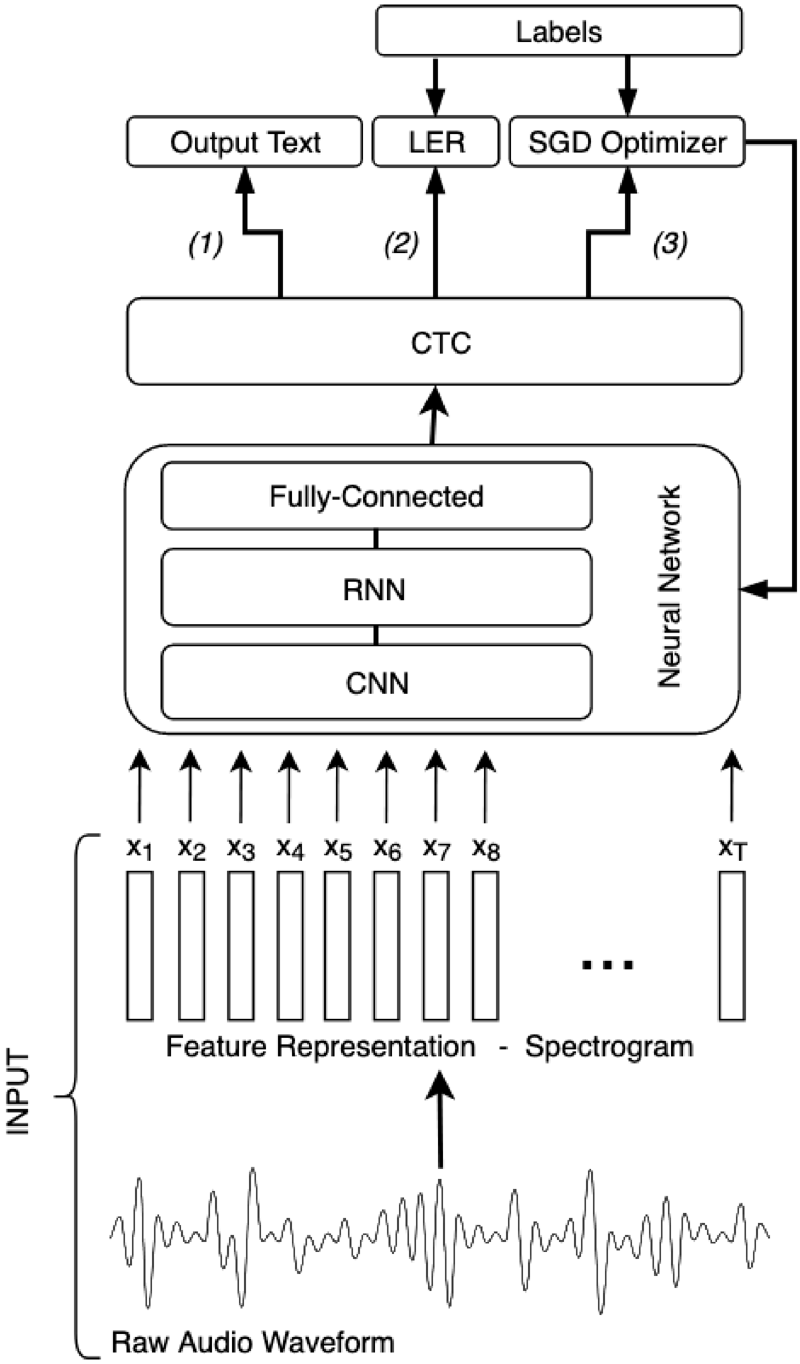


Figure 6: A high-level diagram of the model architecture for all three modes of the system: (1) – Inference; (2) – Evaluation; (3) – Training.

The figure above presents a high-level diagram of the chosen model architecture, which consists of three main parts – feature representation of the input signal, multi-layered neural network and connectionist temporal classification.

3.4.1 Feature Representation

The chosen feature representation of the input signal for this project is the spectrogram, which illustrates the normalized power spectrum of the audio over a set of frequencies. The spectrum is obtained by applying consecutive Fast Fourier Transform functions on the time-domain waveform. The chosen window type was *Hanning* with a length of 20ms and an overlap of 10ms. The spectrogram was computed via the *signal.spectrogram* function as part of the SciPy library.

An alternative to the spectrogram is the Mel Frequency Cepstrum (MFC). It is obtained by taking the discrete cosine transform (DCT) of the log of the mel frequency spectrogram (*see Appendix X*). The MFC is identical to the spectrogram in terms of dimensionality and contributes to the accuracy of the model, compared to inputting raw audio data. However, the spectrogram was chosen due to its simpler computation, incorporating less processing of the raw waveform.

After preprocessing the raw training and testing audio files in order to obtain their feature representations, they are stored as 2D NumPy arrays (i.e. matrices) in the '*.npy'* file format. Their respective transcriptions are mapped to numbers according to the provided output mapping files for each language and are stored in *'.npy'* files as well. Then, when training/testing takes place, those files are read in groups as indicated by the batch size (specified in the config file), converted back to NumPy matrices and fed into the neural network.
At inference time, the batch size is being fixed to one, a raw audio file is read, its features are extracted and being fed directly into the neural network without storing them on the working machine.

3.4.2 Neural Network

The designed neural network consists of 6 layers – two 2D CNN layers, three bi-directional RNN layers and one fully-connected output layer (*see Fig. 7*). The following subsections will look into each set of layers in more details.

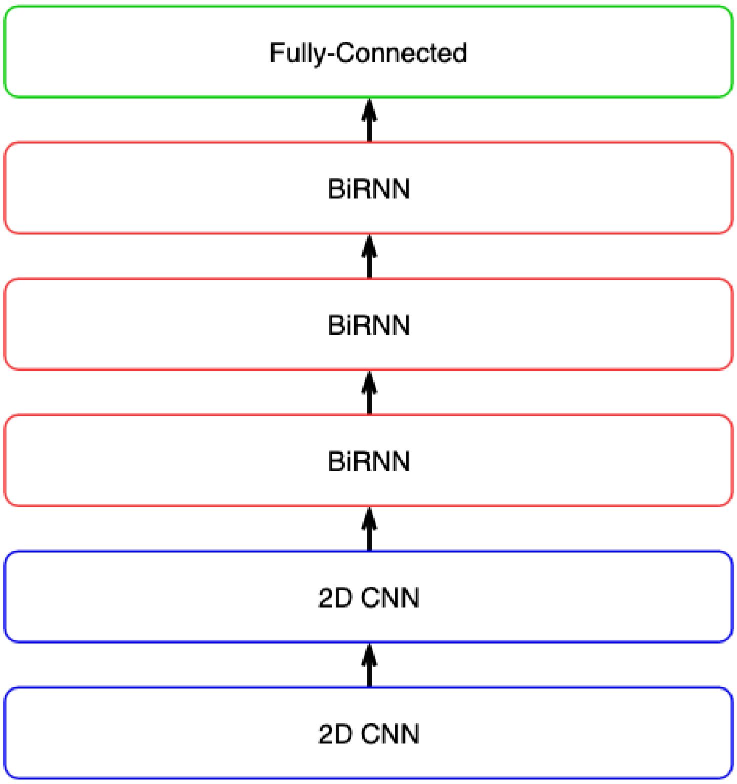


Figure 7: A high-level diagram of all layers in the neural network.

3.4.2.1 CNN Layers

Similarly to Deep Speech 2 (Amodei et al. ,2016), the first two layers of this neural network are CNN layers, acting as feature extractors (*see section 2.3.2.3*). The 2D convolution (both in time and frequency domain) is preferred to the 1D (time domain only), due to its ability to model the spectral variance resulting from the variability of speakers more accurately, thus having a greater contribution to the performance of the whole system.

Initially, there was just a single CNN layer. However, TensorFlow was giving an *AlreadyExists* error, stating that a certain temporary variable already exists and cannot be reinitialized. This problem was solved by increasing the number of CNN layers from one to two. Most probably the error was caused due to TensorFlow using a different wrapper when working with a single layer of CNN as opposed to multiple layers, which may have caused the issue.

For the application of the CNN layers, the function *conv2d* from the neural net (*nn)* operations module of the TensorFlow library was used. 'Same' padding was applied to the inputs in order to keep the same dimensionality in the outputs and the kernel was set to a size of 5x5 with a stride of 1 in each direction.

3.4.2.2 RNN Layers

The core of the neural network consists of three bi-directional RNN layers (*see Fig. 8*). Bi-directional RNNs (BiRNN) are preferred to conventional uni-directional ones due to their ability to make use of future context as well as previous one. Since speech is transcribed in whole utterances where characters are related to each other, this bi-directionality increases the transcription accuracy in prediction.

All values of y_t for t = 1 to T for a single layer BiRNN are calculated as follows:

$$y_t = W_{\overrightarrow{h}y}\,\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_0$$

where

$$\overrightarrow{h}_t = \mathbf{H}\left(W_{x\overrightarrow{h}}\,x_t + W_{\overrightarrow{h}\overrightarrow{h}}\,\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathbf{H}(W_{x\overleftarrow{h}}\,x_t + W_{\overleftarrow{h}\overleftarrow{h}}\,\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}})$$

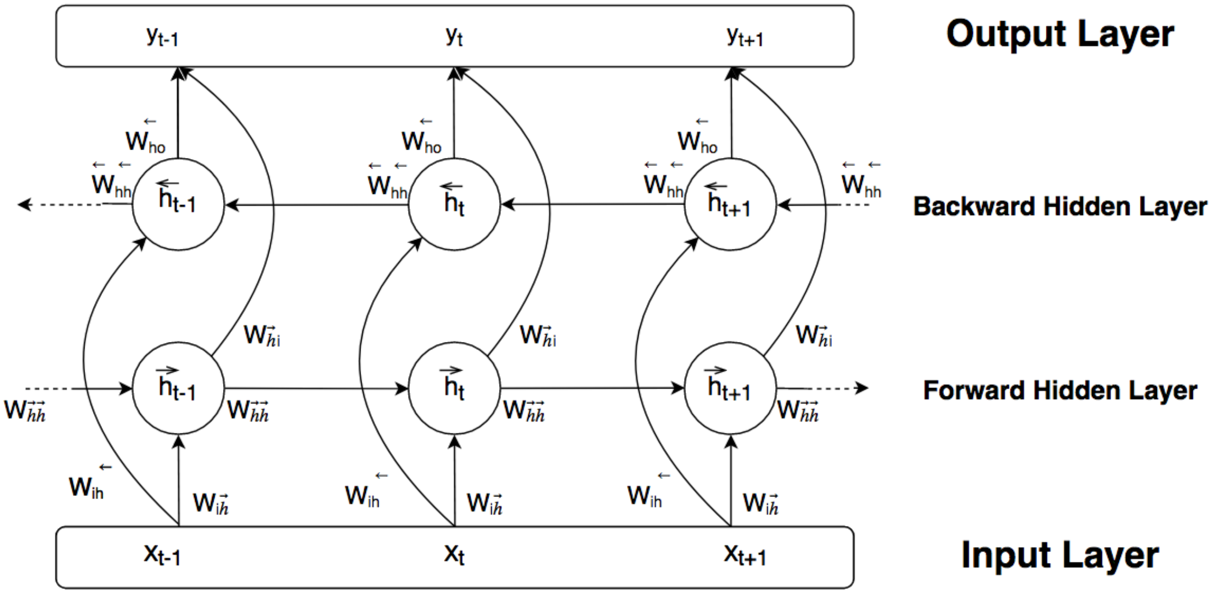
where **H** is the activation function, which usually is a sigmoid function that fits the output in the range (0, 1) and each *b* is the respective bias vector.

A *deep* BiRNN is created by stacking multiple bi-directional hidden layers between the input and the output which operate in the same way.

The preferred type of units for the neural network is the GRU. Those units are designed to overcome the vanishing gradient problem that often occurs when vanilla units are used by deciding what information from the past should be kept, based on relevancy to the model (*see Fig. 9*). This is done by introducing gates that regulate the information flow. This way, relevant information will not be washed away with time.

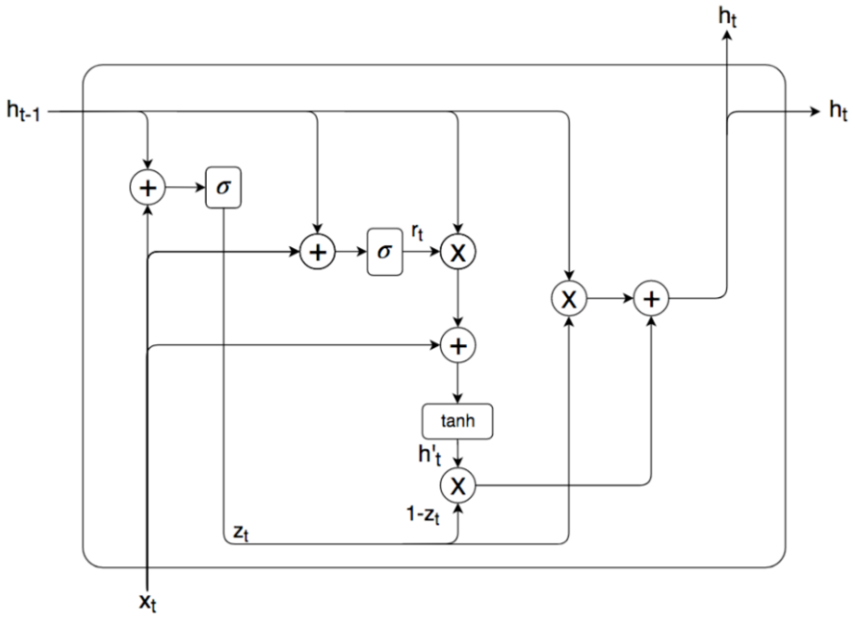
An alternative to the GRU unit is the LSTM unit. Similarly to the GRU, the LSTM unit is designed to overcome the vanishing gradient problem and the short-term memory problem in recurrent neural networks by using gates and the so called ‘memory’ that carries important information from previous time steps. However, the LSTM unit has a more complex structure with more gates and performs a higher number of tensor operations. Therefore, it is computationally heavier and takes more time to train, compared to the GRU.

For the application of the RNN layers, two functions from the *nn* TensorFlow module were used. First, the *rnn_cell.GRUCell* function was used for creating all of the forward and backward GRU cells (the number of units is specified in the configuration file) for all RNN layers. Then, the *bidirectional_dynamic_rnn* function is used to build the independent forward and backward RNNs that comprise the dynamic BiRNN from those cells.



y_{t-1}
y_t
y_{t+1}
Output Layer
$W_{\overleftarrow{h}o}$
$W_{\overleftarrow{h}\overleftarrow{h}}$
$\overleftarrow{h}_{t-1}$
$\overleftarrow{h}_t$
$\overleftarrow{h}_{t+1}$
Backward Hidden Layer
$W_{\overrightarrow{h}i}$
$W_{\overrightarrow{h}\overrightarrow{h}}$
$\overrightarrow{h}_{t-1}$
$\overrightarrow{h}_t$
$\overrightarrow{h}_{t+1}$
Forward Hidden Layer
$W_{i\overleftarrow{h}}$
$W_{i\overrightarrow{h}}$
x_{t-1}
x_t
x_{t+1}
Input Layer

Figure 8: A generic model of a bi-directional recurrent neural network.



$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

$$h'_t = tanh(Wx_t + r_t * h_{t-1})$$

$$h_t = z_t * h_{t-1} + (1 - z_t) * h'_t$$

Figure 9: Gated Recurrent Unit (GRU).

3.4.2.3 Fully-Connected Layer

The last layer in the neural network is a fully connected layer, which means that at all of its nodes receive input from every node in the previous layer. The fully connected layer acts identical to a traditional DNN layer (*see Fig. 2*). It performs the final classification based on the features extracted earlier in the neural network.

This layer is applied by simply multiplying the output of the last BiRNN layer and a weight matrix of shape *[rnn_size, number_of_classes]* and adding a bias vector of length *number_of_classes* to the product.

This gives the output of the whole neural network - at each time step *t,* it outputs a set of probabilities for each character from the Latin or Cyrillic alphabet, including the *space* and *blank* symbols and the *apostrophe,* in the form $p(k_t \mid x)$ where k_t represents each character.

3.4.3 Connectionist Temporal Classification (CTC)

The output of the neural network is fed into the so called Connectionist Temporal Classification (CTC) layer. It interprets the above-mentioned outputs as probability distributions of sequences P(*c*|*x*) across all of the possible sequences of labels, where *c* can be any sequence:

$$P(c|x) = \prod_{i=1}^{N} P(c_i|x) \qquad (4)$$

P(*c*|*x*) is essentially the product of the probabilities for each character of the sequence of interest, which probabilities are taken from the softmax of the output of the NN. The CTC then maps all sequences to their respective transcriptions by removing all duplicates and blanks (e.g. β(HHHH_E__LLL_LO__) = "HELLO").

As already mentioned, the system has three running modes, where CTC is used differently:

a) Training

When training mode is active, the output matrix of the neural network is fed into the CTC loss function together with the respective transcriptions of the current batch of training audio recordings. The loss value is expressed as the negative logarithm of the probability of the transcription given in the training set. This is calculated by summing all label sequences that produce this transcription (e.g. all sequences that are mapped to "HELLO"):

$$P(y|x) = \sum_{c:\beta(c)=y} P(c|x) \qquad (5)$$

$$CTC_{loss} = -\log(P(y|x)) \qquad (6)$$

For this project, the CTC loss is being calculated by the *ctc_loss* function from the *nn* module in TensorFlow. The softmax operation is being applied to the RNN outputs inside this function, removing the necessity to apply it beforehand.

Following the CTC loss calculation, the sum of the probabilities of all transcriptions given in the current batch is being maximized (which would minimize the loss value) by tuning the set of parameters θ:

$$\theta^* = \operatorname{argmax}_\theta \sum_i \log P(y^{*(i)} | x^{(i)}) \tag{7}$$

This operation is done by using the stochastic gradient descent (SGD) optimizer (*GradientDescentOptimizer*) that comes as part of the *train* module in TensorFlow.

Two alternative optimizers that were considered during the development of this project are the Root Mean Square Prop (RMSProp) and Adaptive Moment Estimation (Adam). The RMSProp is an adaptive learning rate optimizer, meaning that as opposed to SGD, its learning rate is not a fixed value, but is being updated in accordance with the average of the squared values of all past gradients. Similarly, the Adam optimizer also computes an adaptive learning rate. However, it builds on top of RMSProp by also applying momentum (a different calculation of the past gradients). Both RMSProp and Adam are designed to adapt to the training process and speed up convergence, however, when tested they showed decreased performance, compared to SGD.

b) Evaluation

When the system runs in evaluation mode, it goes through two steps – decoding the outputs of the NN and calculating the Label Error Rate (LER) of the prediction.

The decoding stage has the purpose of identifying the most probable sequence of numbers from the probability matrix created by the CTC. The most straightforward decoding algorithm is the best path decoder, which simply takes the most probable characters at each time step and concatenates them to make the most probable sequence.

However, this method is prone to errors, since it does not add up probability scores of characters sequences that yield the same transcription, but treats them as separate transcriptions. Instead, the beam search decoder is used. It overcomes the best path decoder problem by creating beams (i.e. text candidates) from the NN output and scoring them, thus obtaining the most probable transcription more accurately.
For the application of the beam search decoder, the *ctc_beam_search_decoder* function from the *nn* module of TensorFlow was used.

Once the most probable transcription is obtained by the decoder, the LER is calculated on the according batch. Further information about the LER calculations can be found in the testing and evaluation section.

c) Inference

When the system operates in inference mode, the outputs of the NN are simply sent to the beam search decoder and the output of the decoder is mapped back to letters using the provided output mapping files for each language (e.g. 0 = ‘a’; 1 = ‘b’, etc.) in order to display a final readable transcription to the user.

3.5 Data Set

For this project, both large-vocabulary and small-vocabulary datasets were being used in order to compare performance.

Normally, the training set makes up 80% of the whole data set and the test set makes up the remaining 20%. It is assumed that the examples in both set are identically distributed. This split ratio was used for this project as well.

3.5.1 Large-vocabulary

For large-vocabulary Bulgarian language recognition, the Bulgarian speech corpus BulPhonC was used (Hateva et al., 2016). The corpus consists of 319 sentences selected out of 21838 to contain most phoneme bigrams in Bulgarian language at least three times. Speakers from various ages and regions of residence participated in the recording (*see Table 4 for more details*).

For large-vocabulary English language recognition, the LibriSpeech corpus was used (Panayotov et al., 2015). The corpus is obtained from audiobooks belonging to the LibriVox project. It consists of 1000 hours of recorded speech, split into 3 groups. For this project, the smallest group of 100 hours will be used (*see Table 4 for more details*). To make the model lighter and faster, a reduced version of the LibriSpeech dataset was also used. This was done by removing all files that are bigger than 2.5 MB in size. This way, all weight and bias matrices were reduced in size, since they are as big as the largest training file, thus allowing the batch size to be increased to 10 without crashing the server.

Table 4: Details for large-vocabulary data sets that will be used in the project.

Name	Language	No. of sentences	No. of sentences spoken	No. of speakers (total)	No. of speakers (female)	No. of speakers (male)	Total length (h)
BulPhonC	Bulgarian	319	21891	147	85	62	32
LibriSpeech	English	---	---	251	125	126	100

3.5.2 Small-vocabulary

For small-vocabulary English language recognition, the Speech Commands dataset developed by Google was used (Warden, 2018). It contains a total of 105 829 one second-long audio recordings of 35 different speech commands (e.g. ‘left’, ‘right’, ‘stop’, etc.). The data was collected via people’s phone and laptop microphones on various locations in order to capture noise coming from everyday surroundings and the non-professional quality of the microphones. No demographic information is provided for this dataset (*see Table 5 for more details*).

Table 5: Details for small-vocabulary data sets that will be used in the project.

Name	Language	No. of Keywords	No. Audio Recordings	No. of speakers (total)	Total length (h)
Speech Commands	English	35	105 829	unkown	30

3.6 Testing and Evaluation Methodology

During the training process, calculations were performed over a set period in order to test and evaluate the performance of the model described in section 3.4. As mentioned in section 3.5, the testing would be performed on 20% of each data set, containing only data which was not observed by the model while training. However, since the cost function did not converge to a desirable amount, such testing was not performed and the evaluation is based on the training evaluation.

As discussed in 3.4.3 b), the main measure used for evaluating the system's performance is the label error rate. It is simply the edit (Levenshtein) distance between the predicted transcription and the ground truth label in a given example. The edit distance represents the minimum number of insertions, substitutions or deletions of single characters that would change the prediction into the actual transcription (*see Fig. 10*). For the calculation of the LER, the *edit_distance* function from TensorFlow was used.

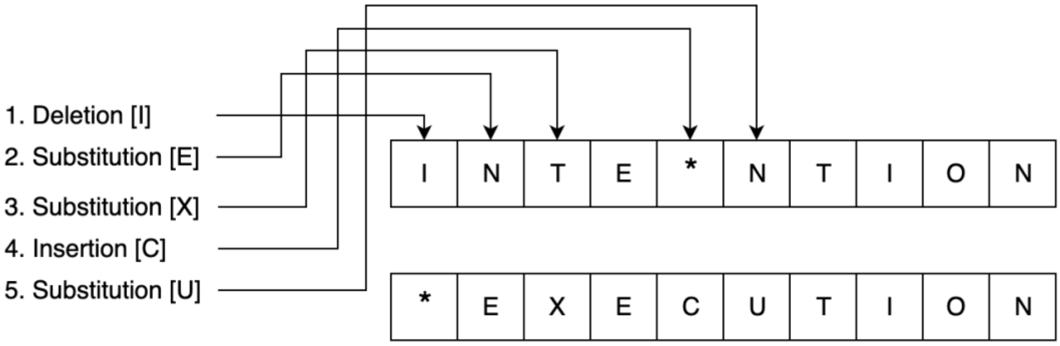


Figure 10: A basic example of calculating edit distance.

4.0 RESULTS

In order to evaluate the system objectively for the given time and resources, the training and testing process was split into three parts – large-vocabulary training, small-vocabulary training and single file overfitting. The following sections cover those parts, the most relevant training and testing configurations and the outcomes from them.

4.1 Large-Vocabulary

The first part of the testing process aims to evaluate the system performance when trained on large-vocabulary data sets with large speaker variability and no constraint on the words being used.

4.1.1 LibriSpeech

The following table presents the initial configurations that were tested:

Table 6: Details on the initial testing scenarios on LibriSpeech.

	Batch Size	Optimizer	Learning Rate	RNN size	Epochs	Num. of audio files	Min. LER value per batch achieved
(1)	6	SGD	0.001	64	3	85 608	72 %
(2)	6	Adam	0.002	64	< 1	15 960	99 %
(3)	6	SGD	0.003	64	< 1	15 000	70 %
(4)	6	SGD	0.005	64	1	29 040	68 %

(1)

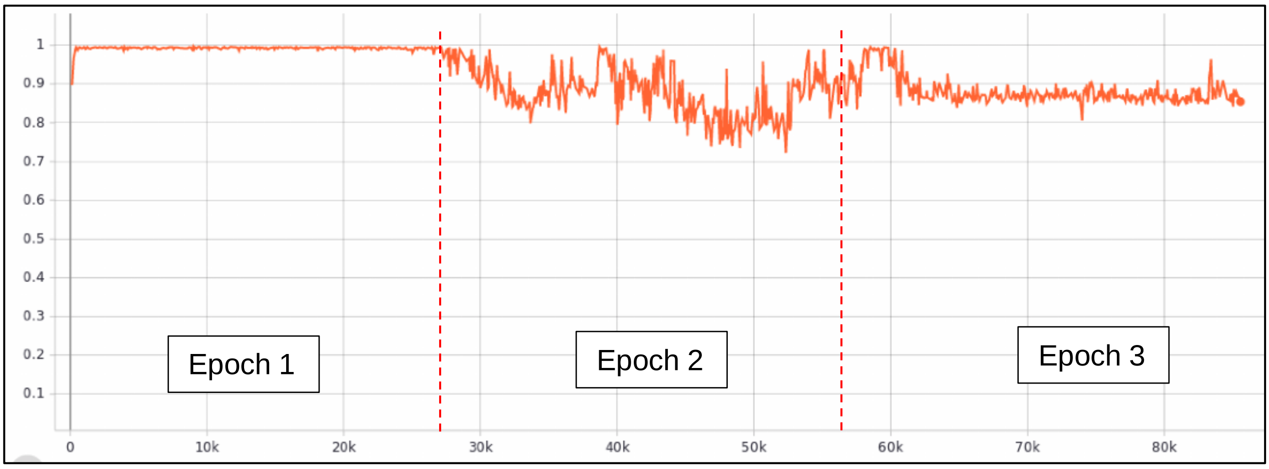


Figure 11: TensorBoard graph of LER over training (1) on LibriSpeech.

As the graphs shows, the LER evaluation acted differently in the three training epochs. In the first epoch, the LER was almost constant with a value of 99%. In the second epoch, it started fluctuating and reached its minimum of 72% at global step 52 320. In the last epoch, the LER hit a plateau and remained the same until the end of the training. Furthermore, the cost function was very noisy and did not converge.

(2)

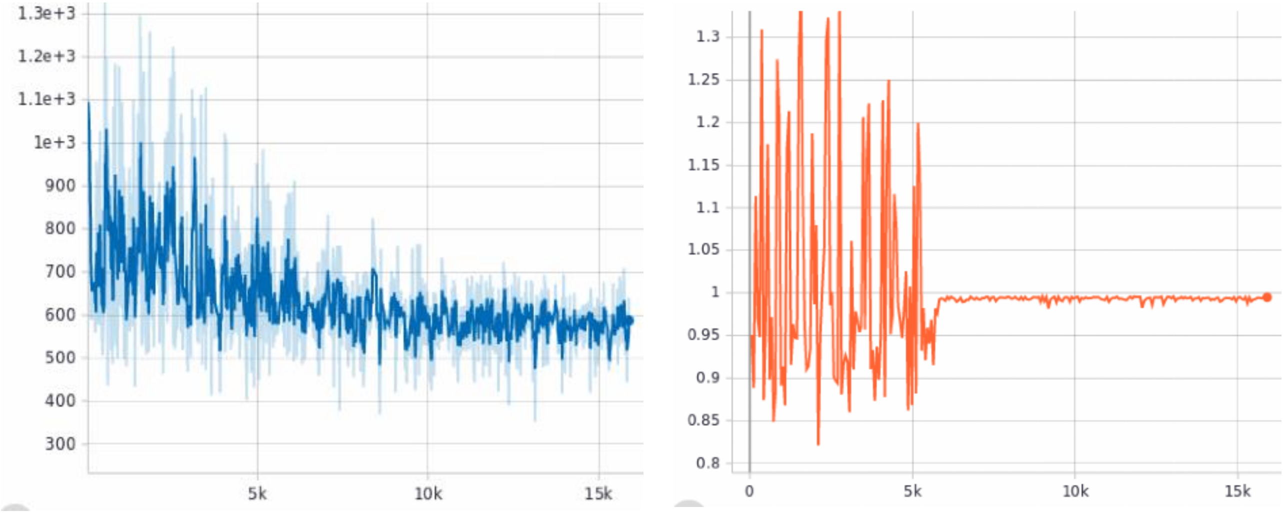


Figure 12: TensorBoard graphs of cost with 0.6 smoothing (left) and LER (right) over training (2) on LibriSpeech.

In the second training scenario, improvements could be noticed in the cost function. It was getting less noisy and slightly dropped its average value (*Fig. 12 left*). However, this did not have an impact on the LER, which was fluctuating with a large amplitude for the first 5000 global steps, after which it hit a plateau of 99% and remained fairly constant until the end of the training.

(3)

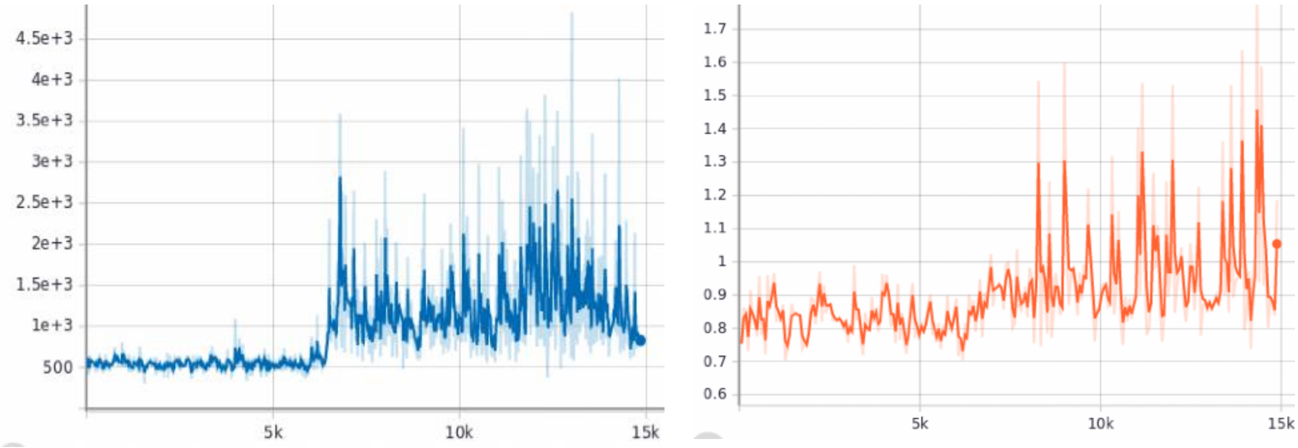


Figure 13: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.4 smoothing (right) over training (3) on LibriSpeech.

In the third training scenario, both the cost function and the LER followed a similar trend – they remained fairly constant for the first 6000 global steps with LER values fluctuating between 80% and 90%, after which their values drastically increased and the LER reached values up to 144%.

(4)

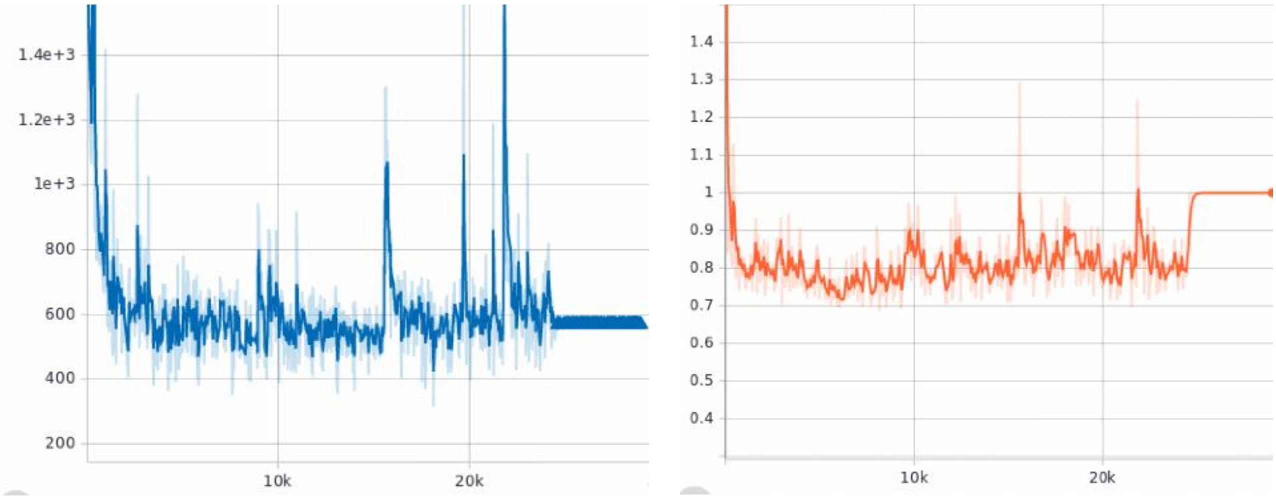


Figure 14: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) over training (4) on LibriSpeech.

In the last training scenario, the LER reached its lowest value of 68% at global step 8160. However, the overall cost and LER values did not drop significantly. At global step 24 590, the cost value was outputted as NaN (not a number) and te LER became 100% and both remained the same until the end of the training. It is believed that this is the vanishing gradient problem, occurring as a result of the high learning rate used.

The following tests were performed on the reduced version of LibriSpeech, as described in section *3.5.1*:

Table 7: Details on the second set of testing scenarios on LibriSpeech

	Batch Size	Optimizer	Learning Rate	RNN size	Epochs	Num. of audio files	Min. LER value per batch achieved
(5)	10	SGD	0.003	64	3	60 200	79 %
(6)	10	SGD	0.005	64	3	60 200	72 %
(7)	10	SGD	0.007	64	3	60 200	69 %
(8)	10	SGD	0.01	64	3	60 200	67 %

(5)

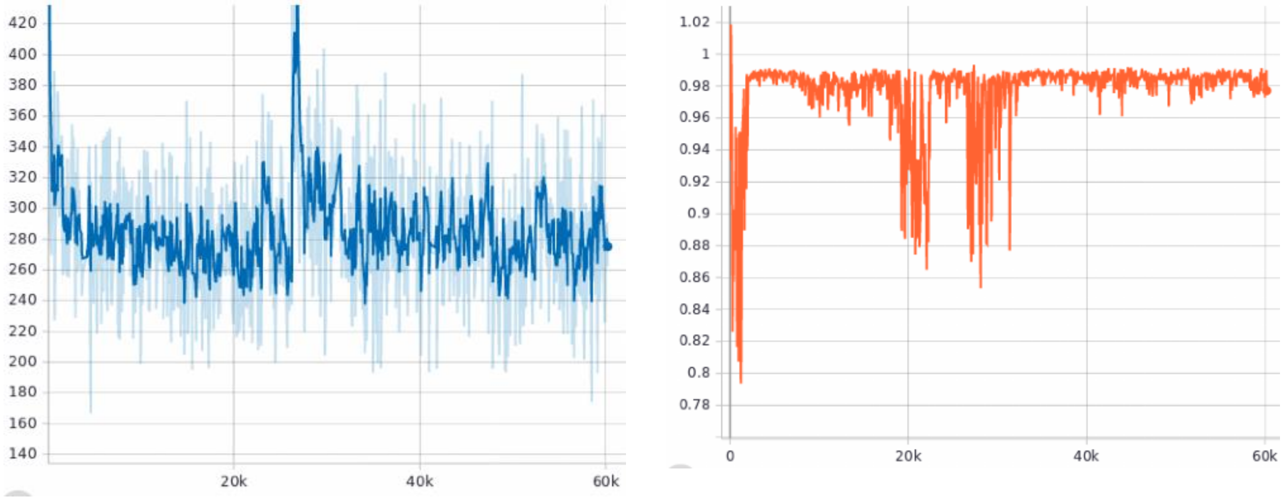


Figure 15: TensorBoard graphs of cost with 0.7 smoothing (left) and LER (right) over training (5) on LibriSpeech

For a learning rate value of 0.003, the cost function did not show significant improvement as shown in the figure above. The LER had an initial drop to 79 % at global step 1250, followed by two more drops in the range of 85-90% between global step 28 000 and 32 000. However, for the rest of the training, the LER remained at 98% with very little fluctuations.

(6)

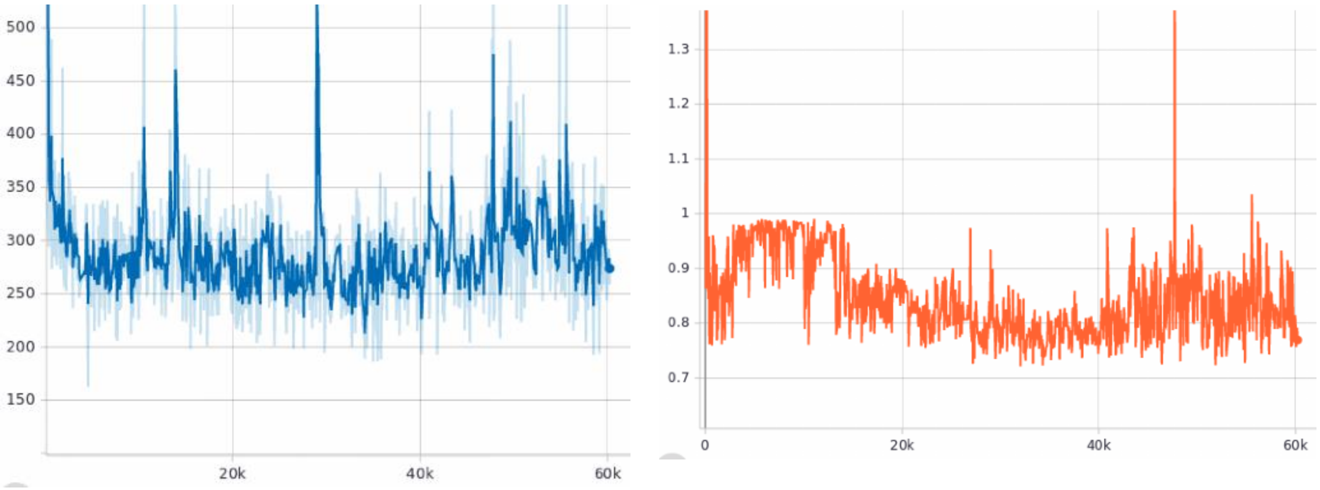


Figure 16: TensorBoard graphs of cost with 0.6 smoothing (left) and LER (right) over training (6) on LibriSpeech

For a learning rate value of 0.005, the LER dropped in the second epoch (after global step 20 000) and reached 72% multiple times, before increasing again and at the end of the training. The cost remained fairly flat with only a few spikes up to global step 45 000, after which it started increasing.

(7)

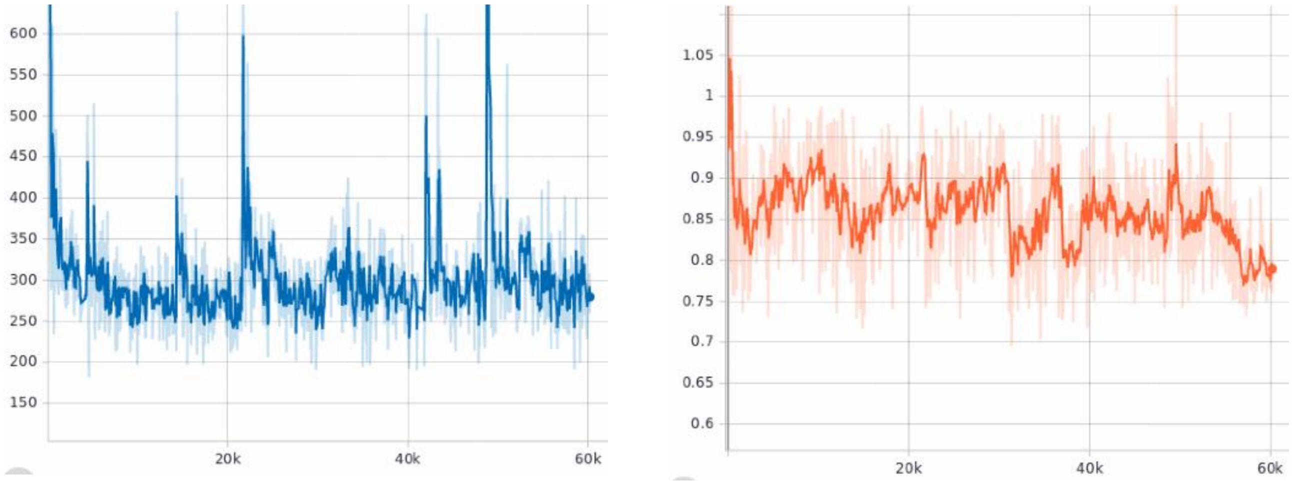


Figure 17: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.8 smoothing (right) over training (7) on LibriSpeech

For a learning rate value of 0.007, the trend of the cost function resembles the ones of (5) and (6), being flat throughout the whole training process with only few occurrences of spikes. The LER in this scenario was noisier than the previous ones, however, it generally started decreasing by the end of the third epoch. It reached its lowest value of 69% at global step 31 350.

(8)

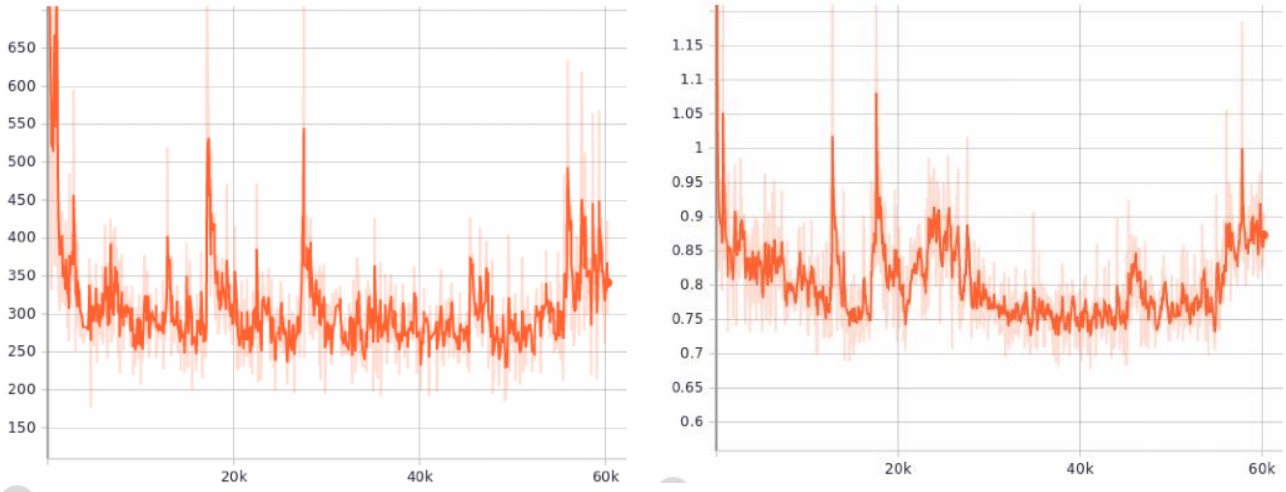


Figure 18: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.7 smoothing (right) over training (8) on LibriSpeech. Coloring is the same for both due to a TensorBoard bug.

For a learning rate value of 0.01, the cost remains flat for most of the training, however, it starts increasing at the end of the third epoch. In contrast, the LER was fluctuating significantly more. It dropped by roughly 15% during the first epoch, followed by a sharp peak in the beginning of the second epoch and another drop by the end of it. During the third epoch, the LER started increasing gradually back to its initial value. It had its lowest value of 67% at global step 41 100.

These tests were performed for equal number of epochs, equal RNN and batch sizes and the same type of optimizer. This way, the impact of the learning rate was tested. The results were quite fluctuating and therefore difficult to compare. However, a general trend can be identified of minimum LER values dropping as the learning rate increases.

4.1.2 BulPhonC

Due to the inability of the cost function to converge and the LER to reach a desirable percentage when the system was trained on LibriSpeech, it was decided to omit the large-vocabulary training on BulPhonC as well in order to proceed with simplifying the training process.

4.2 Small-Vocabulary

Sourcing a small-vocabulary speech corpus has the same idea as the reduction of LibriSpeech discussed in the previous section. The aim is to decrease the size of the whole model, which would allow larger batch sizes and faster training.

4.2.1 Google Speech Commands

The system was tested on the Google Speech Commands dataset with the following configurations:

Table 8: Details on the testing scenarios on Speech Commands

	Batch Size	Optimizer	Learning Rate	RNN size	Epochs	Num. of audio files	Min. LER value per batch achieved
(1)	64	SGD	0.005	64	3	233 920	96 %
(2.1)	32	SGD	0.01	128	10	846 400	74 %
(2.2)	32	SGD	0.01	128	17	1 443 808	74 %

(1)

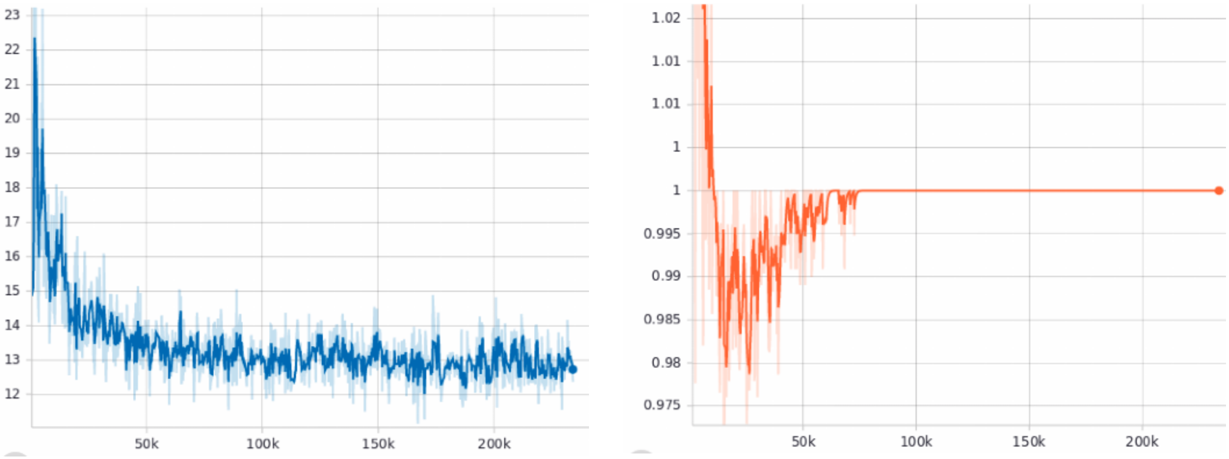


Figure 19: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of training (1) on Speech Commands.

In the first testing scenario, the cost generally follows a decreasing curve with significantly less noise than the examples discussed in section 4.1. However, after the end of the first epoch (global step 84 K), the cost plateaus and remains relatively the same until the end of the training loop. The LER values show a small drop in the first epoch, achieving the lowest value of 96% at global step 15 040, followed by an increase to 100%. The LER remained exactly 100% for the rest of the training.

(2.1)

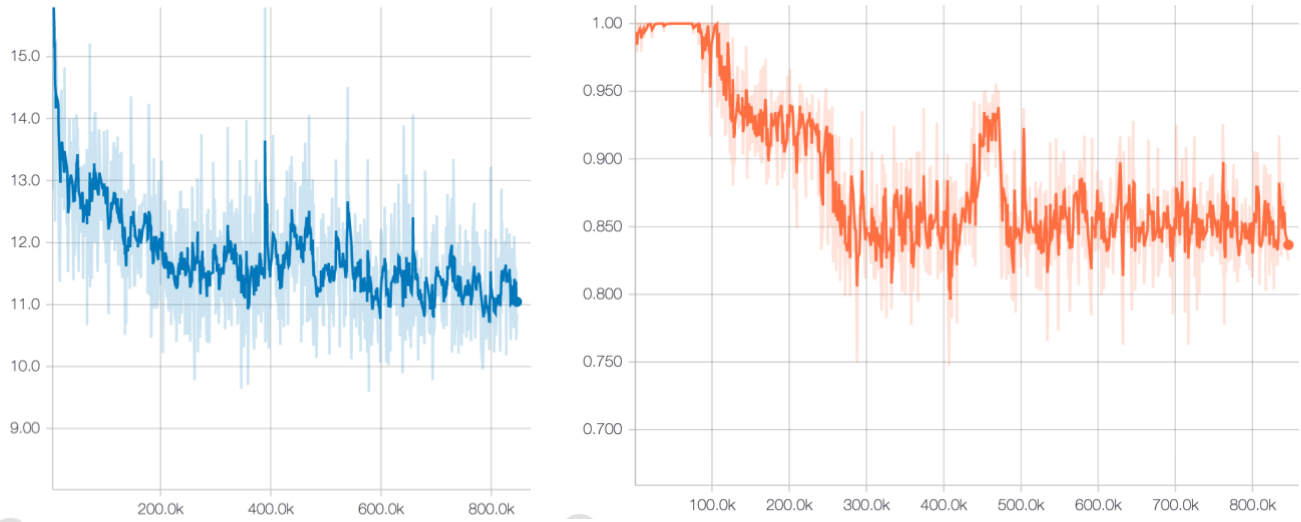


Figure 20: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of training (2) on Speech Commands.

In the second testing scenario, when the batch size was reduced to 32 and the rnn size increased to 128, the LER value had two large drops in the first four epochs, reaching its lowest value of 74 % at global step 407 040. From there on, the LER remained fairly flat with one big peak. The cost value decreased quickly in the first two epochs and then continued decreasing very slowly until the end of the training.

(2.2)

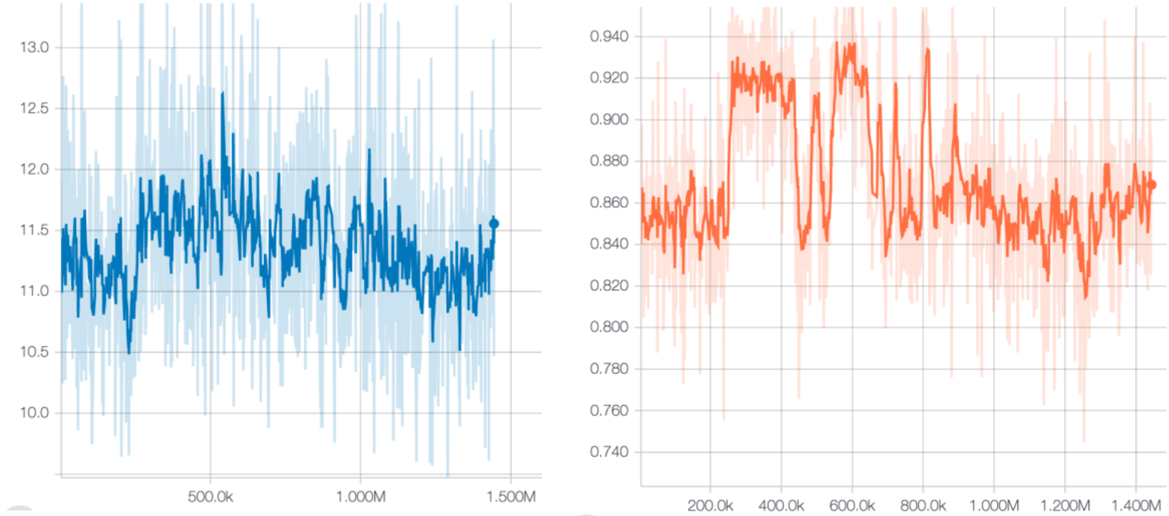


Figure 21: TensorBoard graphs of cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of training (3) on Speech Commands.

The final testing scenario was a continuation to the previous one, starting from checkpoint 846 400. As the figure above shows, the cost and the LER did not decrease further from their starting values. Both measurements were very noisy and the LER had two large peaks, shown in the figure. It reached its lowest value of 74% at global step 1.235 M.

4.3 Single File Overfitting

The last part of the testing process was aiming to show that the proposed system has the ability to adapt successfully to input data in both languages. The approach to this was to oversimplify the process by overfitting the model to a chosen set of single audio files. The RNN size was set to 64 for all training/testing scenarios.

4.3.1 Choice of Sentences

In order to make a fair choice of sentences, statistical analysis was performed on both data sets. First, the distribution of letters in each data set was calculated in order to identify the most commonly used letters and evaluate the effects of the letter distribution on the prediction accuracy.

The following graphs plots the average number of occurrences of each letter from the English alphabet per sentence in LibriSpeech:

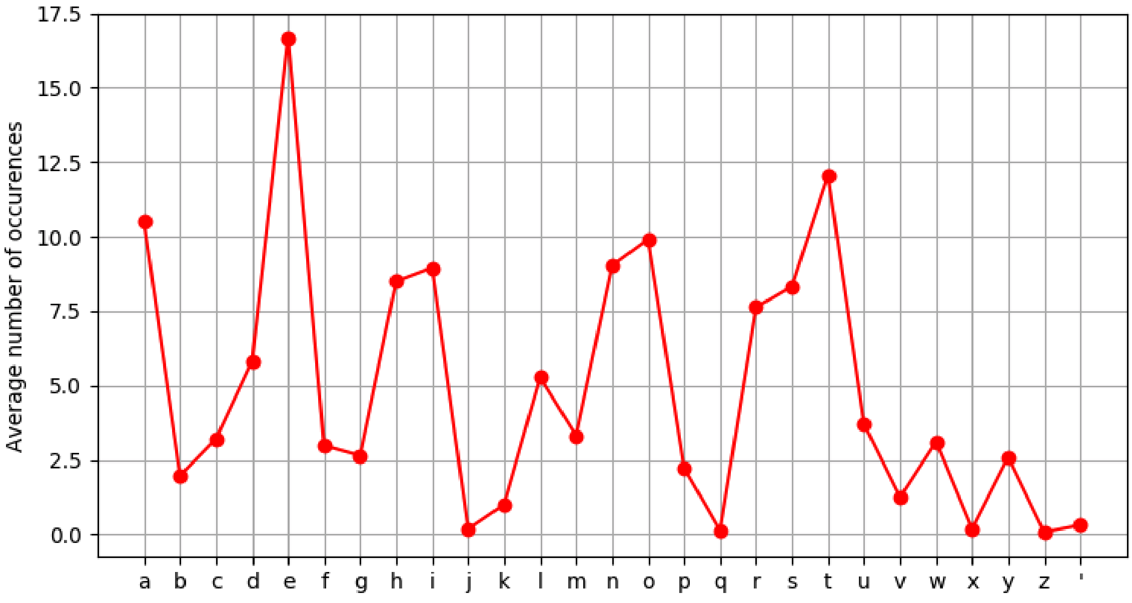


Figure 22: A line graph of the average distribution of letters per sentence in LibriSpeech.

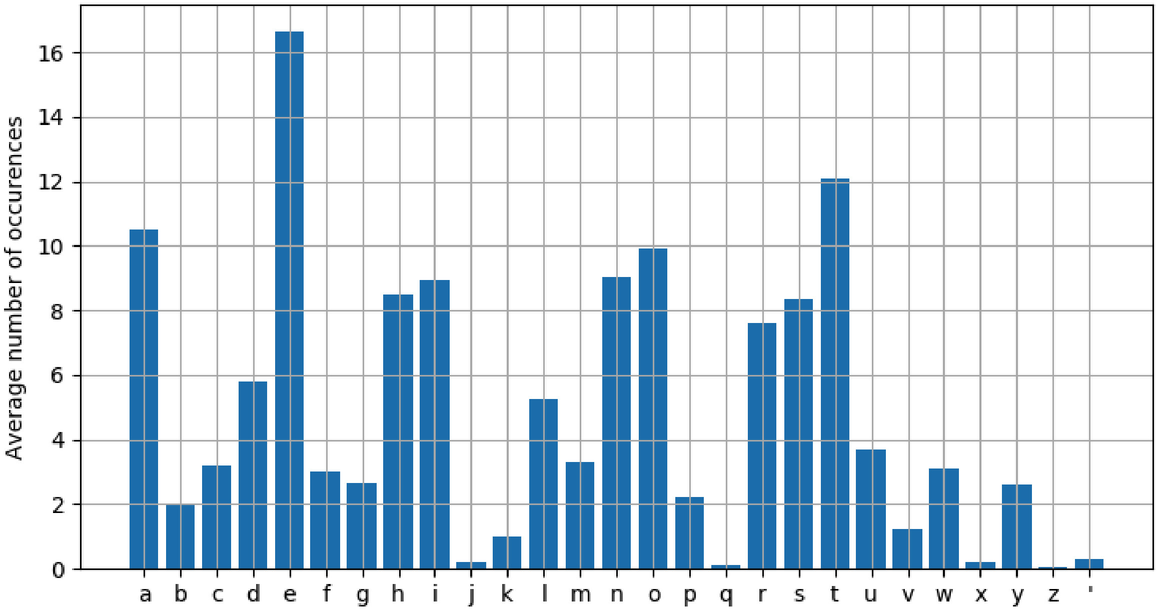


Figure 23: A histogram of the average distribution of letters per sentence in LibriSpeech.

The results from the analysis show that the most common letters in the data set are '*e*', '*t*' and '*a*', respectively appearing on average 16.5, 12 and 10.4 times per sentence. As opposed, the letters *'j', 'q', 'x', 'z*' and the apostrophe appear less than once in a sentence on average. The average sentence length was found to be 160 characters.

Similarly, the distribution of letters from the Bulgarian alphabet in BulPhonC are plotted on the following graphs:

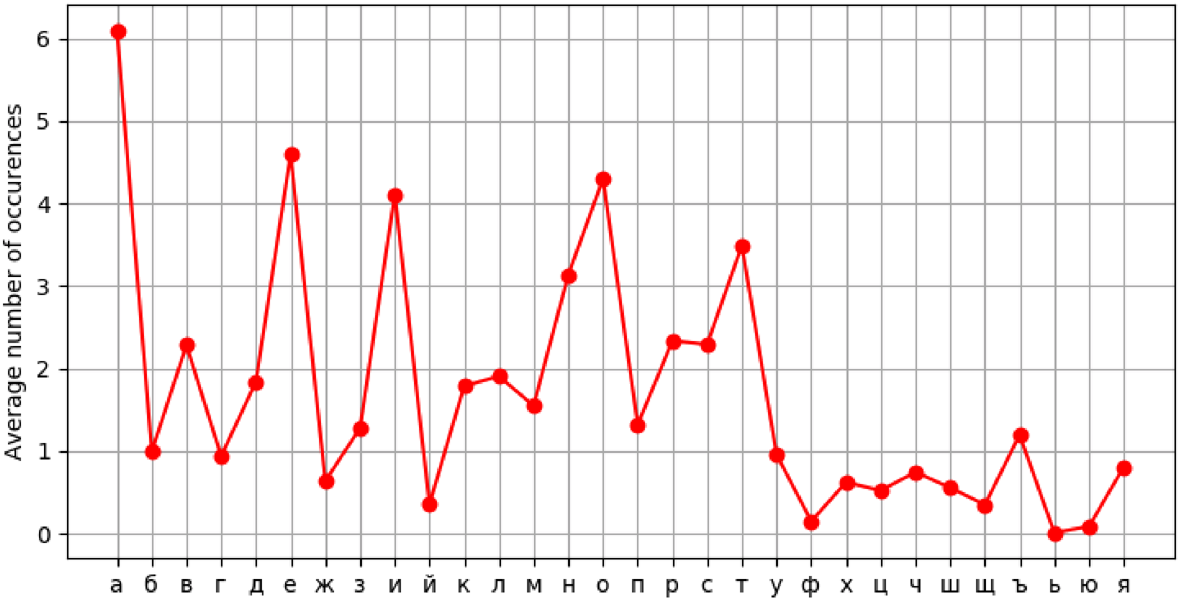


Figure 24: A line graph of the average distribution of letters per sentence in BulPhonC.

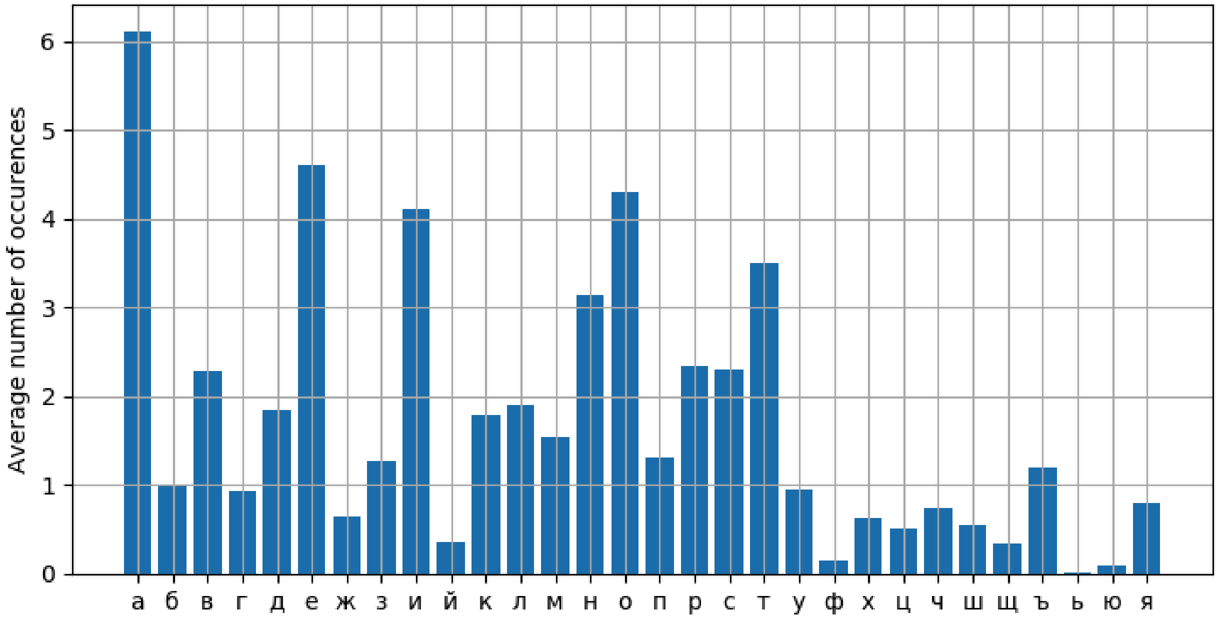


Figure 25: A histogram of the average distribution of letters per sentence in BulPhonC.

The analysis of BulPhonC shows that the most commonly used letters in the data set are 'а', 'е', 'о', 'и'(equivalent to 'i'), and 'т', respectively appearing on average 6, 4.5, 4.25, 4 and 3.5 times per sentence. As opposed, the letters 'ь', 'ю', 'ф' and 'й' appear on average less than 0.5 times per sentence. The average length of sentence in the data set was calculated as 61 characters.

The next step was calculating the standard deviation for each sentence from both data sets, thus identifying how flat are their distributions. This calculation was done by using the following formula:

$$SD = \sqrt{\frac{\sum(x_i - m)^2}{N}} \tag{8}$$

where x_i represents the number of occurences of each letter in the alphabet, m represents the mean of all those values and N is the number of letters in the alphabet. Once calculated, the four sentences with the smallest standard deviation value (meaning flattest distribution) and length of at least the average length for each data set were picked for the process of overfitting.

4.3.2 LibriSpeech

4.3.2.1 Training Files

The four sentences from LibriSpeech, chosen through statistical analysis as described in the previous section, are presented in the following table:

Table 9: Details on the sentences chosen for overfitting from LibriSpeech

	ID	Transcription	Length (chars.)	Standard Deviation
(1)	7800-283492-0003	we can only give a guess at that frank told him whew exclaimed bluff as he grasped the meaning back of those few words after all I wouldn't put it past him frank	161	3.695
(2)	6848-252322-0000	said craggs pushing billy towards them as he spoke faix and ye might have got worse muttered a very old man billy traynor has the lucky hand how is my lord now nelly	165	3.701
(3)	200-126784-0029	I have no cold in my head i fancied it then from seeing you had covered such handsome black locks with that ugly old wig it was my mistake you will please to pardon it	167	3.867
(4)	8324-286682-0038	well how will you put my tail in place of yours asked the bachelor i don't know answered the young mink but you are so wise that i thought you might know some way	162	3.871

The following graph shows the distribution of each sentence mapped against the average distribution in the dataset:

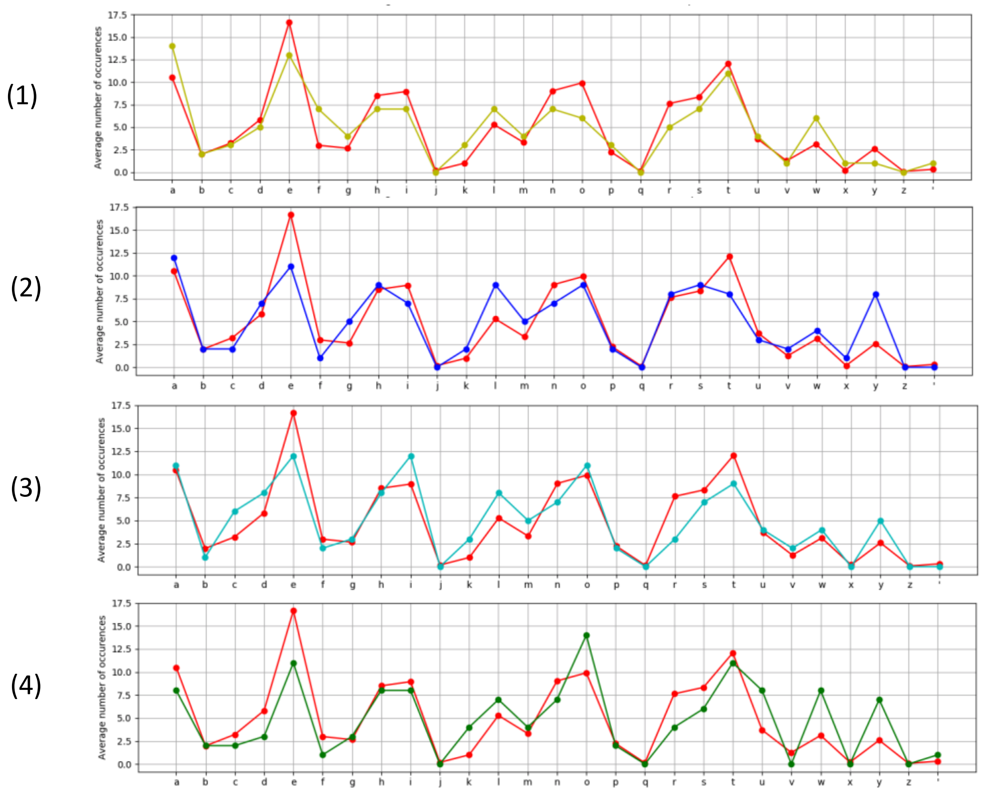


Figure 26: Sentence distributions against average LibriSpeech distribution (in red).

The graph clearly shows that all sentences have a flatter distribution than the average with less prominent peaks. Dips and peaks still exist; however, they are more equally distributed and the peaks representing the most commonly used letters are multiple and have similar values.

4.3.2.2 Results

a) Audio File 7800-283492-0003

The overfitting on the first file was completed in two parts. In the first part the learning rate was set to 0.005, whereas for the second part it was reduced to 0.003. Both parts were being run for 1000 epochs each.
The following figure presents the results from the first part of the training:

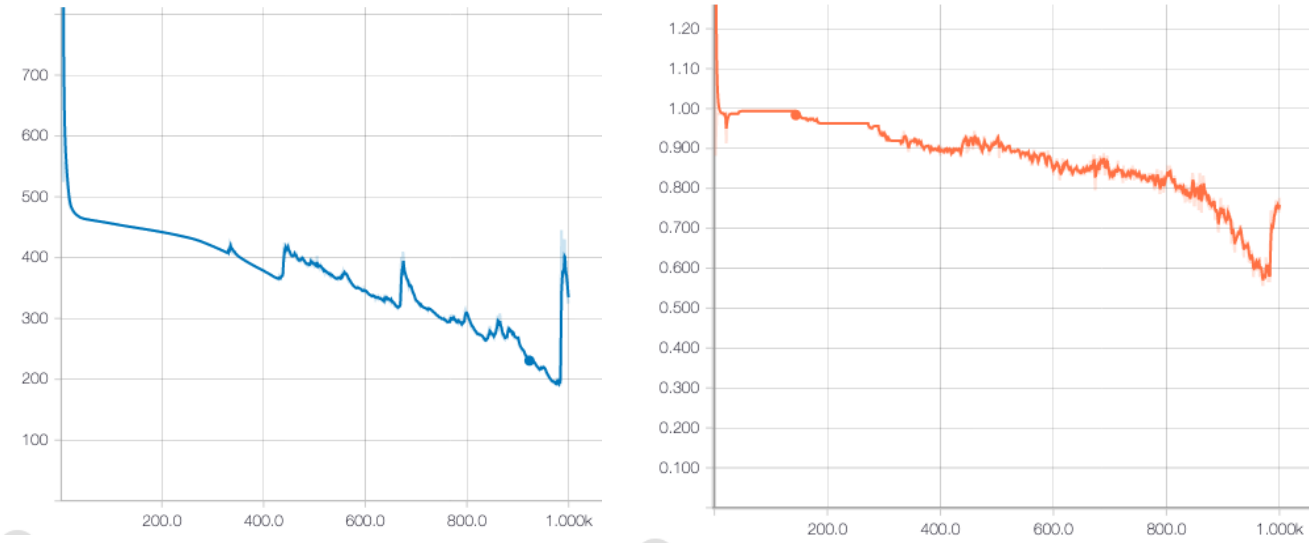


Figure 27: TensorBoard graphs of the cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of the first overfitting example pt.1.

The results from this first part show a very slow decrease in the beginning both in cost value and LER, which gets sharper near the end. The cost dropped to **189** and the LER reached a value of **55%** at global step 983. However, at step 984, there was a sudden rise in both values, possibly due to the size of the learning rate being too big.

Therefore, the learning rate was reduced to 0.003 and the training was continued from checkpoint 983:

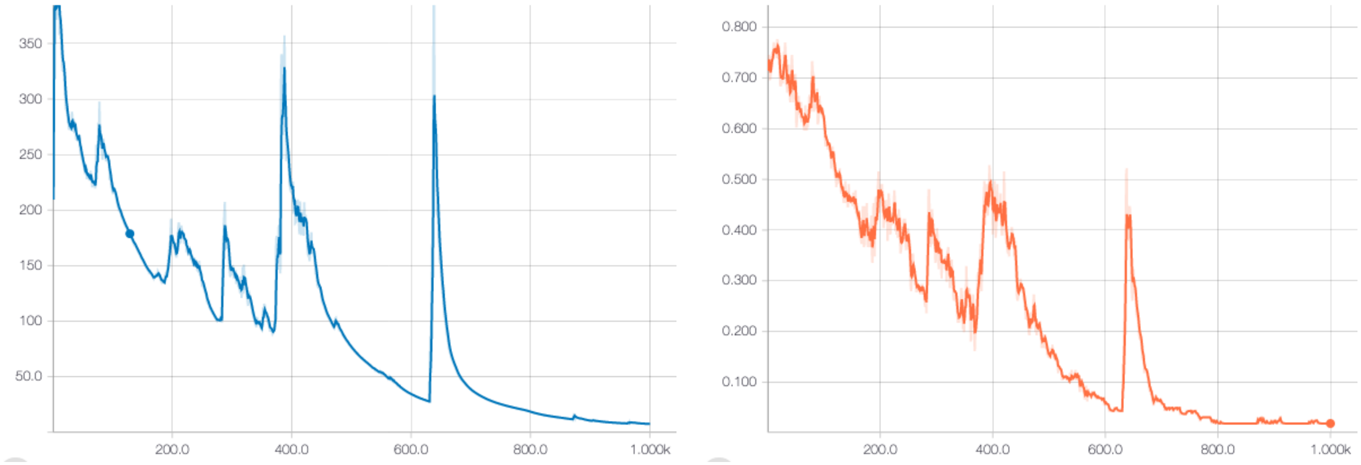


Figure 28: TensorBoard graphs of the cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of the first overfitting example pt.2.

The results from the second part are slightly noisier with two large spikes in both values around global steps 400 and 650. However, eventually the cost converged to its lowest value of **7.4** at global step 1000, while the LER dropped to **1.86%** at step 789 and remained flat with very little fluctuations until the end of the training.

At inference, when using the model from checkpoint 789 with LER **1.86%,** the proposed transcription is:

Original:	we can only give a guess at that frank told him whew exclaimed bluff as he grasped the meaning back of those few words after all I wouldn't put it past him frank
Output from the system:	we can only give a gues at that frank told him whew exclaimed bluf as he grasped the meaning back of those few words after al I wouldn’t put it past him frank

b) Audio File 6848-252322-0000

For the rest of the training examples, it was decided to keep a single learning rate value of 0.003. The overfitting of the second audio file was also performed in two parts of 1000 epochs.

The following figure presents the results from the first part of the training:

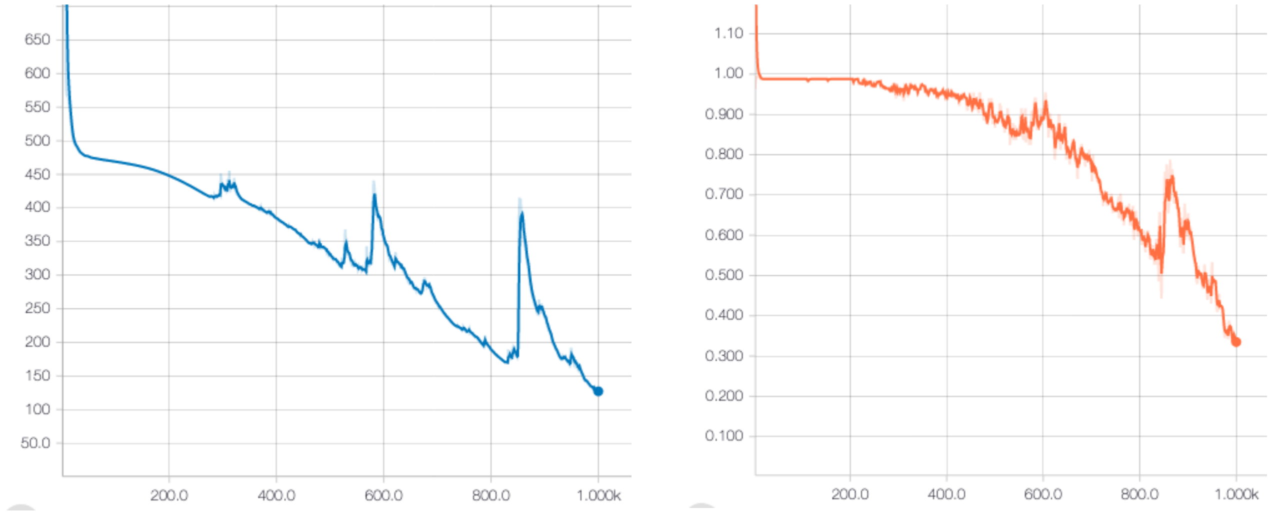


Figure 29: TensorBoard graphs of the cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of the second overfitting example pt.1.

For the first 200 steps, the cost decreased slowly while the LER stayed flat at 98%. For the rest of the training, there was a steady decrease in both values with only two spikes around global steps 600 and 850. The cost value managed to drop to **126**, while the LER reached **32%**. The second part will continue from checkpoint 1000, attempting to bring both values down further.

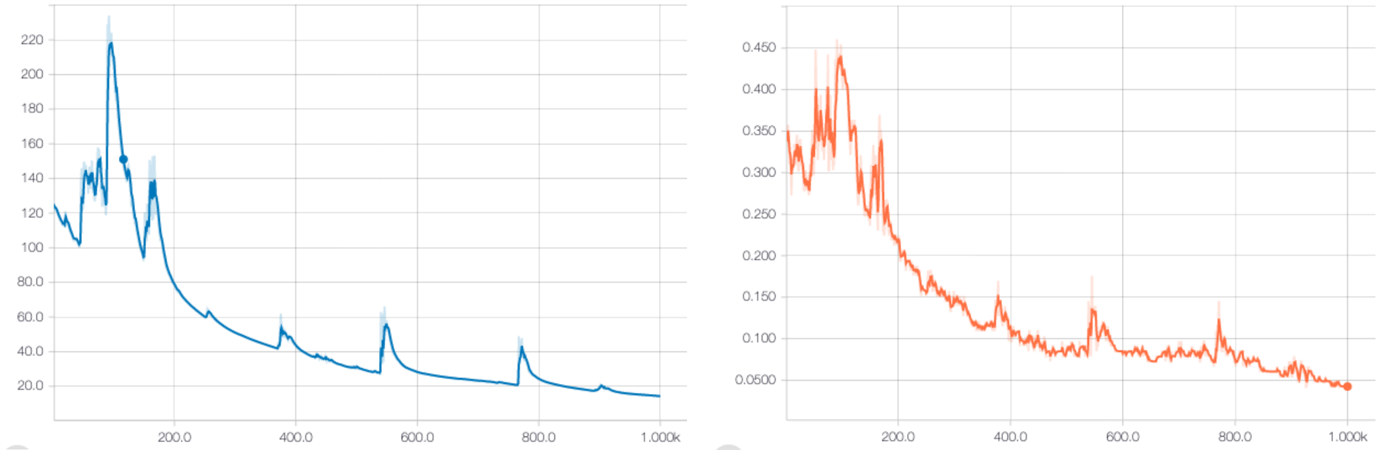


Figure 30: TensorBoard graphs of the cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of the second overfitting example pt.2.

The second part started with a sharp rise in both values with LER reaching over 45%, followed by a sharp decrease. Around global step 500, the decrease slowed down and kept on making small steps until the end of the training loop, when the cost converged to a value of **14.25**, while the LER dropped to **4.24 %**

At inference, when using the model from checkpoint 1000 with LER **4.24%,** the proposed transcription is:

Original:	said craggs pushing billy towards them as he spoke faix and ye might have got worse muttered a very old man billy traynor has the lucky hand how is my lord now nelly
Output from the system:	said crags pushing bily towards them as he spoke faix and ye might have got worse mutered a very old man bily Traynor has the luckhand how is my lord now nely

c) Audio File 200-126784-0029

Since the previous two examples showed that 1000 epochs are not enough for the cost and LER to drop to the desirable level, the training for the rest of the examples will be performed for 2000 epochs.

The following figure presents the results from the training on the third file:

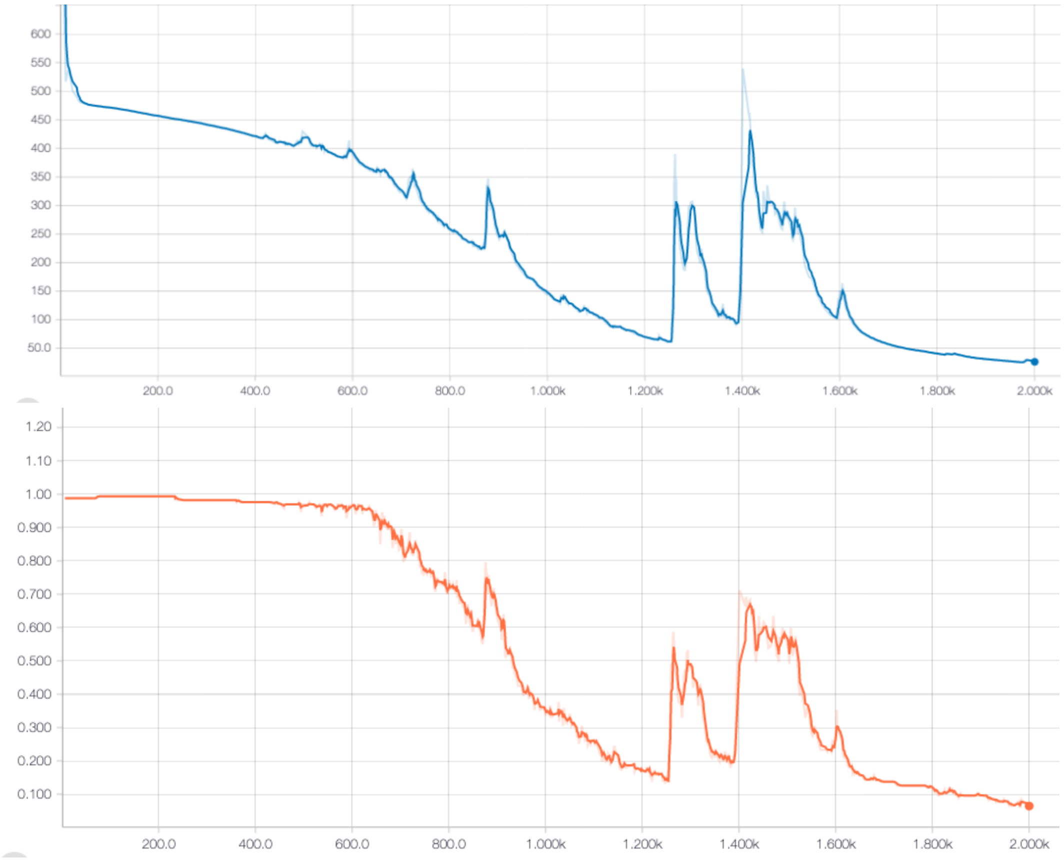


Figure 31: TensorBoard graphs of the cost with 0.6 smoothing (top) and LER with 0.6 smoothing (bottom) of the third overfitting example.

The results show that for the first 600 epochs, there was no significant improvement – the cost was decreasing very slowly and the LER stayed at 95%+. Then there was a sharp decrease in both values and they quickly reached **60** for the cost and **13%** for the LER. However, this was followed by a sharp rise to a peak at epoch 1254, which remained high for another 400 epochs. At the end of the training loop, the cost managed to converge to **25.12** and the LER dropped to **5.98%.**

At inference, when using the model from checkpoint 2000 with LER **5.98%,** the proposed transcription is:

Original:	I have no cold in my head i fancied it then from seeing you had covered such handsome black locks with that ugly old wig it was my mistake you will please to pardon it
Output from the system:	I have no cold my head I fancied it then from seing you had coveuch handsome black locks with that ugly old wig it was my mistake you wil please to pardon it

d) Audio File 8324-286682-0038

This example was trained with the exact same hyperparameters as the previous one (learning rate = 0.003, epochs = 2000).

The following figure presents the results from the training on the fourth file:

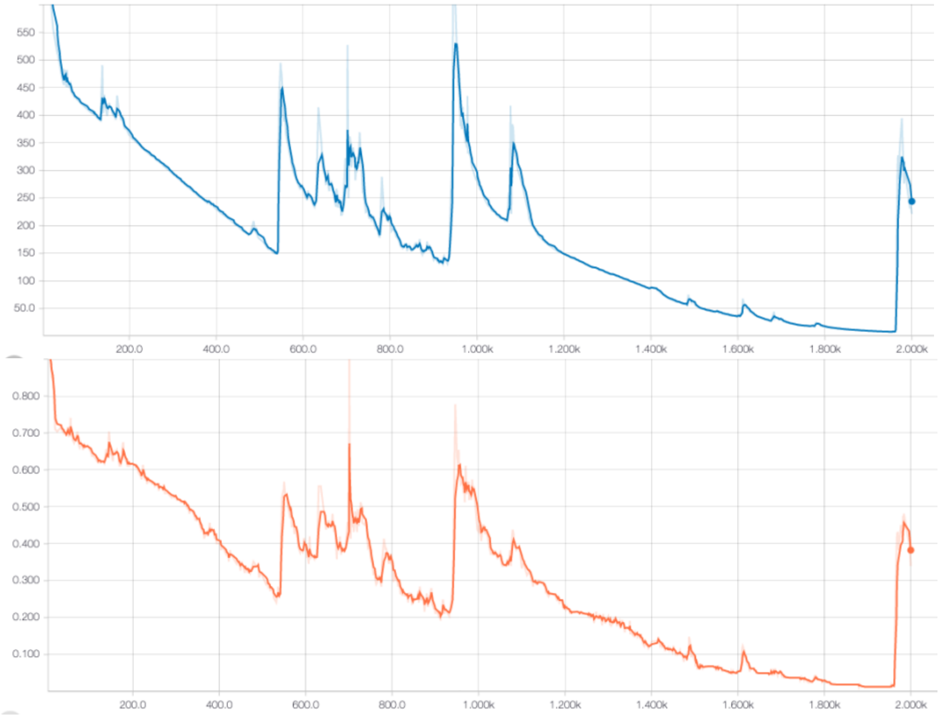


Figure 32: TensorBoard graphs of the cost with 0.6 smoothing (top) and LER with 0.6 smoothing (bottom) of the fourth overfitting example.

For the first 500 epochs, there was a steady decrease of the cost and the LER value, followed by a period of roughly 500 epochs where both values were fluctuating with a large amplitude. After epoch 1000, both values continued with their steady decrease with very little noise until they reached their lowest points of **7.85** for the cost and **1.23%** for the LER.

At inference, when using the model from checkpoint 1945 with LER **1.23%,** the proposed transcription is:

Original:	well how will you put my tail in place of yours asked the bachelor i don't know answered the young mink but you are so wise that i thought you might know some way
Output from the system:	wel how wil you put my tail in place of yours asked the bachelor I don't know answered the younk mink but you are so wise that I thought you might know some wa

4.3.3 BulPhonC

4.3.3.1 Training Files

The four sentences from BulPhonC, chosen through statistical analysis as described in the previous section, are presented in the following table:

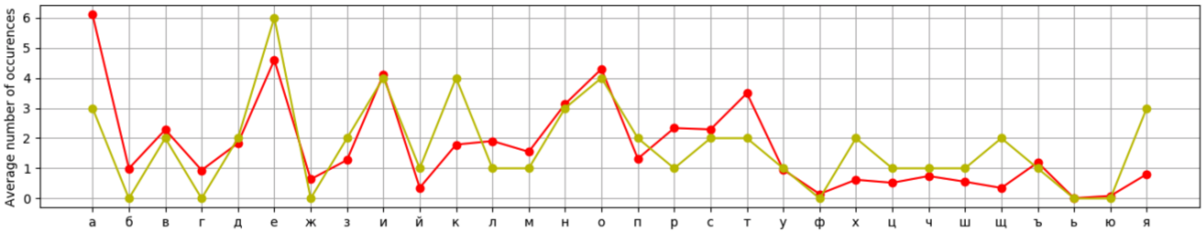
Table 10: Details on the sentences chosen for overfitting from BulPhonC

	ID	Transcription	Length (chars.)	Standard Deviation
(1)	141	през деня не скучаех защото комшийките по цял ден висяха вкъщи	62	1.436
(2)	191	затворих му а той след това се скъса да звъни но аз не му вдигнах	65	1.719
(3)	17	това е редът на планетите спрямо слънцето маркурий венера земя марс юпитер сатурн уран нептун	93	3.027
(4)	24	прилича на акварел в който всичко е загатнато и където не съществува преход между нюансите на охрата	100	2.856

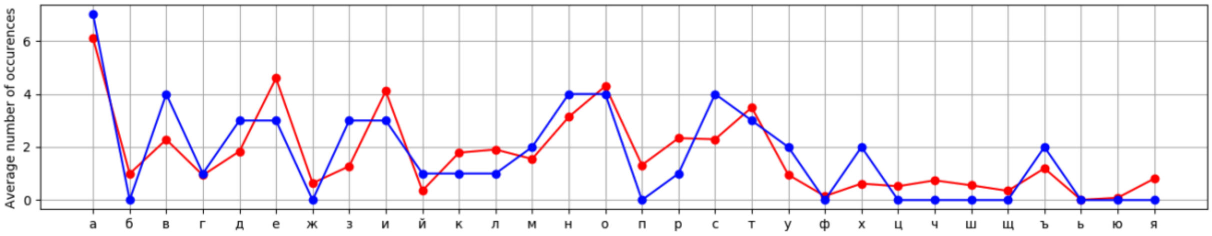
Due to the significantly shorter average length of sentences in the BulPhonC dataset it was decided to pick the two sentences with the smallest standard deviation (1 and 2), as well as two sentences with slightly bigger size and not as low standard deviation (3 and 4), in order to be able to compare the two data sets objectively. Since each sentence is being said by all speakers, the speaker whose audio was used was chosen randomly.

The following graph shows the distribution of each sentence mapped against the average distribution in the dataset:

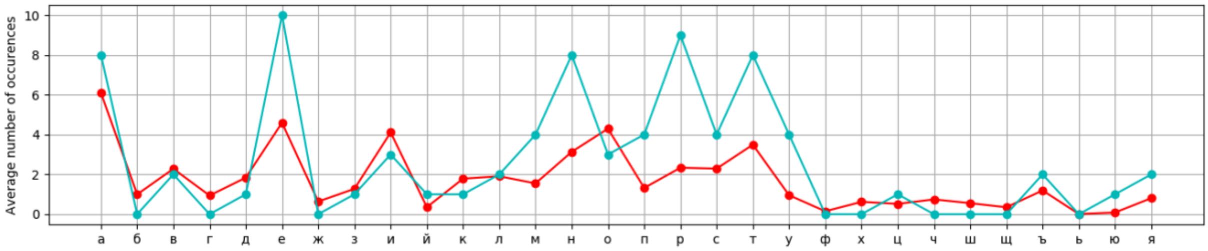
(1)



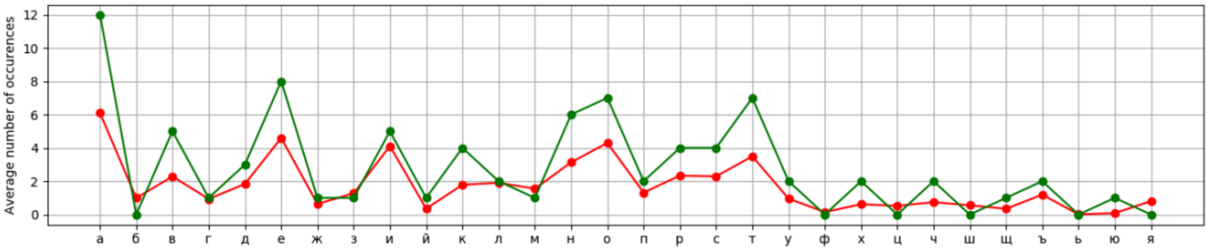
(2)



(3)



(4)



In the plots above, it appears like the sentence distributions are not flat when compared to the average distribution for the data set. This occurs due to the standard deviation of the average distribution of the data set being 1.496, which is only 0.06 more than the one of the first training sentence. Overall, the deviations of these sentences are about three times smaller than those of the sentences from LibriSpeech, which means they are flatter.

4.3.3.2 Results

All examples from the BulPhonC data set were trained with the same hyperparameters – (learning rate = 0.003, epochs = 2000).

a) Audio File S16-141

The following figure presents the results from the training on the first file:

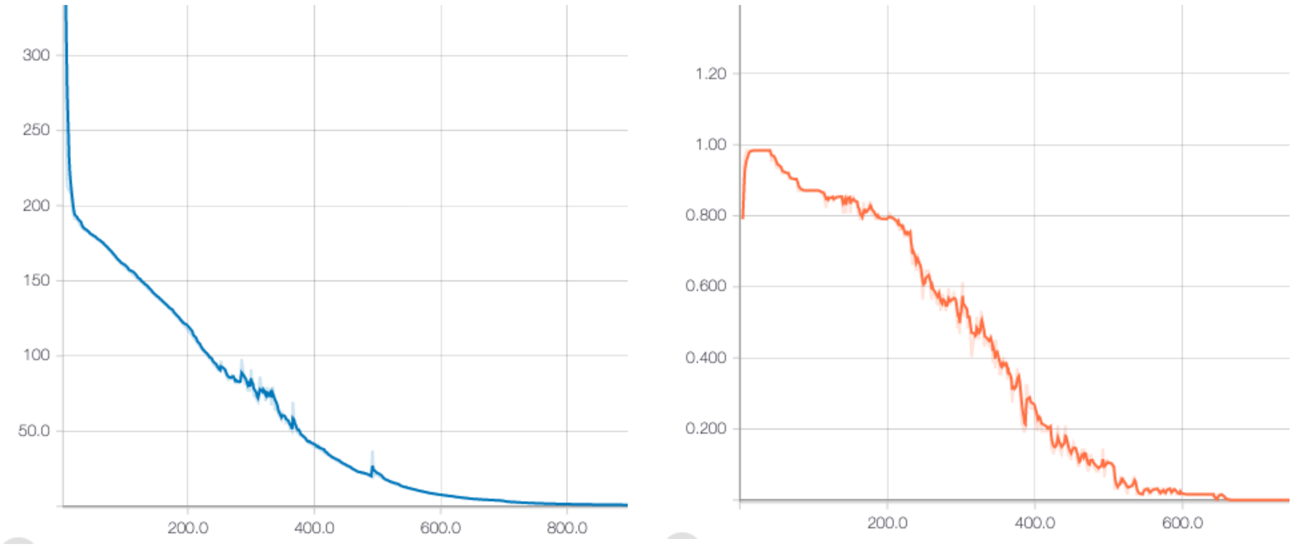


Figure 33: TensorBoard graphs of the cost with 0.6 smoothing (left) and LER with 0.6 smoothing (right) of the first overfitting example (BulPhonC).

Due to the short length of the sentence and the audio file, the cost and LER values dropped very quickly with very little noise and no peaks or spikes. The LER reached **0%** at epoch 643 and remained 0% until the end of the training. At the same epoch, the cost value was **5.2** and it kept on decreasing until it reached **0.2**, even though the LER was already 0%. The training process was interrupted early since the LER had reached its lowest value.

At inference, when using the model from checkpoint 643 with LER **0%,** the proposed transcription is:

Original:	през деня не скучаех защото комшийките по цял ден висяха вкъщи
Output from the system:	през деня не скучаех защото комшийките по цял ден висяха вкъщи

b) Audio File S75-191

The following figure presents the results from the training on the first file:

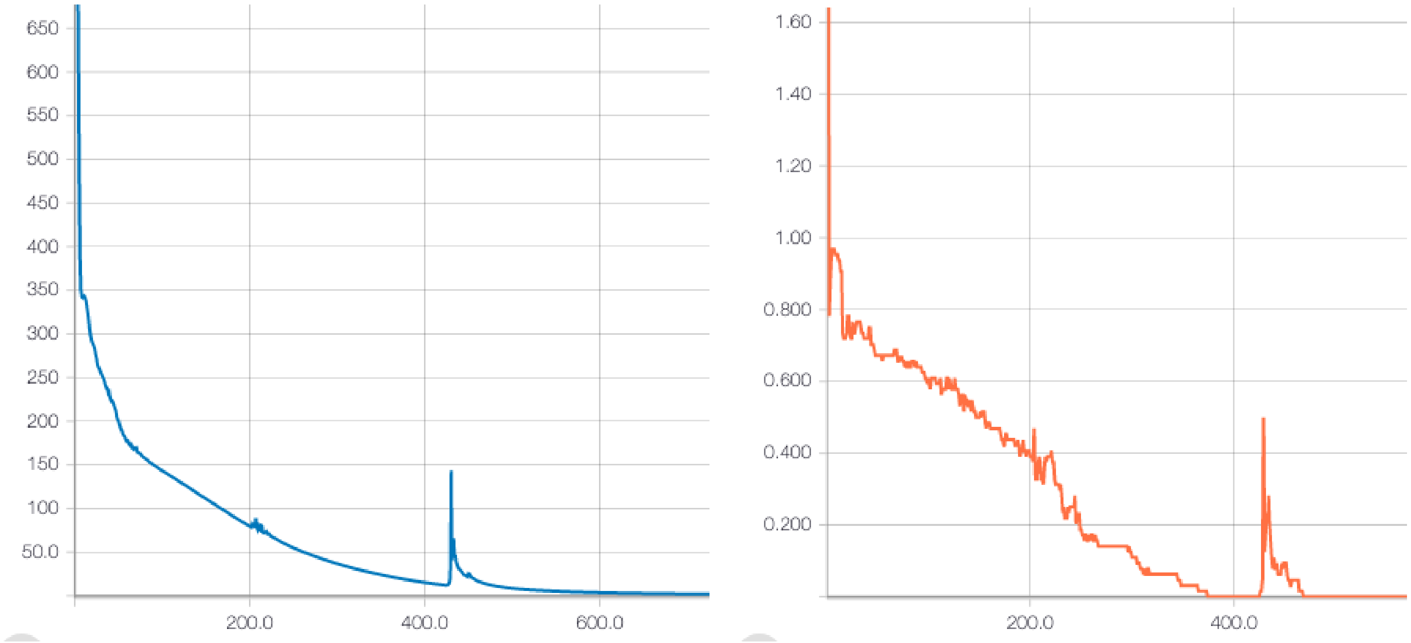


Figure 34: TensorBoard graphs of the cost (left) and LER (right) of the second overfitting example (BulPhonC).

Analogically to the first example from the BulPhonC dataset, the cost and LER here converged quickly. As the graph shows, both values were decreasing steadily with almost no noise and with just one big spike at epoch 429. The LER had reached **0%** before the spike at epoch 374 and dropped back to 0% after the spike as well. The cost kept on decreasing until the end of the training and dropped to **1.07**. As with the previous example, the training was interrupted early due to the fast convergence.

At inference, when using the model from checkpoint 374 with LER **0%,** the proposed transcription is:

Original:	затворих му а той след това се скъса да звъни но аз не му вдигнах
Output from the system:	затворих му а той след това се скъса да звъни но аз не му вдигнах

c) Audio File S30-17

The following figure presents the results from the training on the third file:

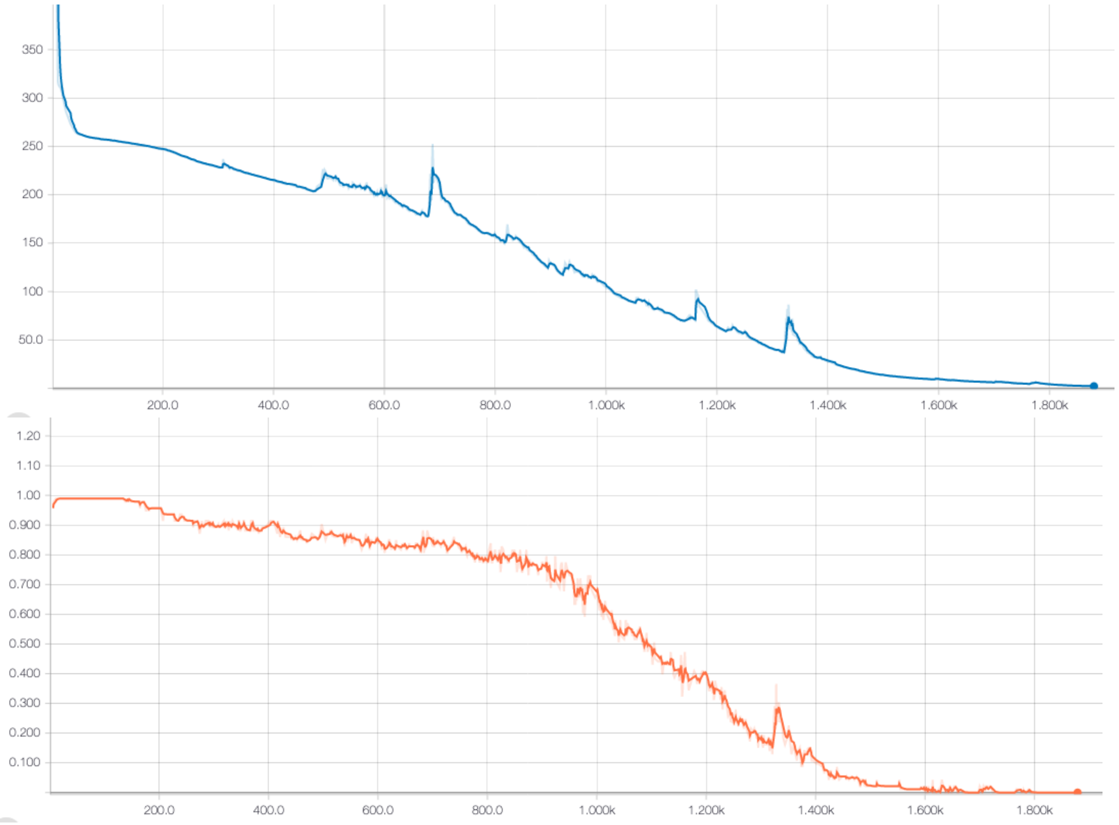


Figure 35: TensorBoard graphs of the cost with 0.6 smoothing (top) and LER with 0.6 smoothing (bottom) of the third overfitting example (BulPhonC).

Both the cost and LER values show constant decrease with very little noise and no significant spikes throughout the training. After epoch 1000, the LER started dropping more rapidly until it reached **0%** at epoch 1600 and remained the same with very little fluctuations between 0 and 2%. The cost kept on dropping even after the LER reached 0%, until it converged to **0.01**.

At inference, when using the model from checkpoint 1600 with LER **0%,** the proposed transcription is:

Original:	това е редът на планетите спрямо слънцето маркурий венера земя марс юпитер сатурн уран нептун
Output from the system:	Това е редът на планетите спрямо слънцето меркурий венера земя марс юпитер сатурн уран нептун

d) Audio File S21-24

The following figure presents the results from the training on the third file:

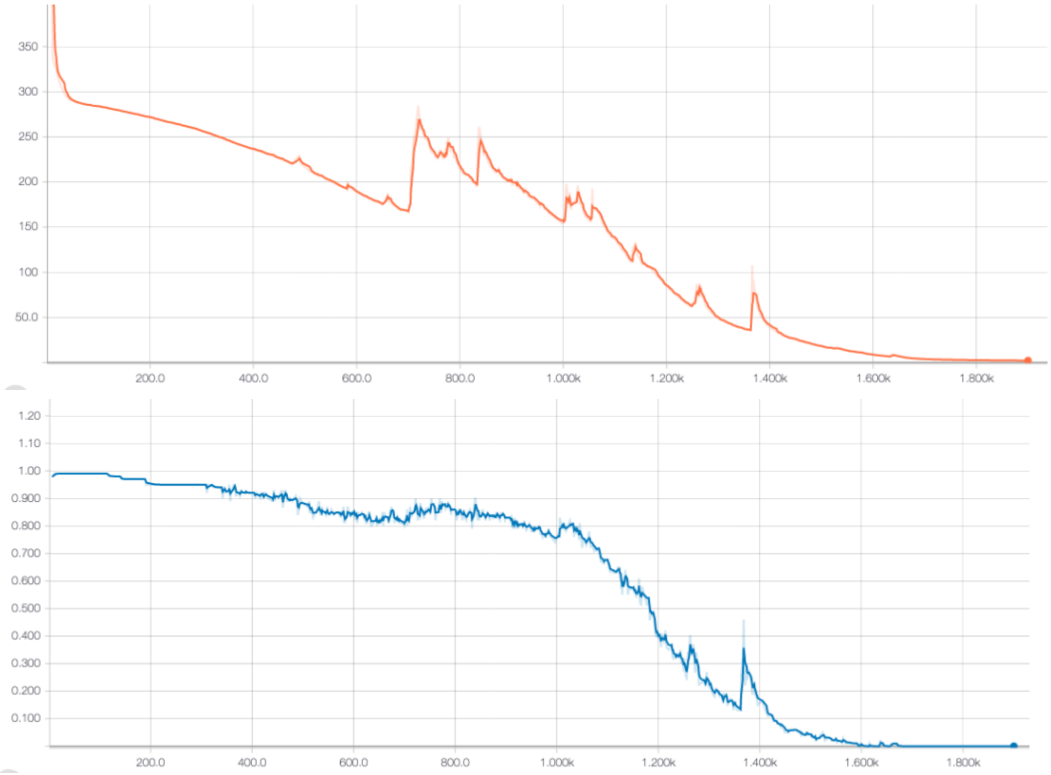


Figure 36: TensorBoard graphs of the cost with 0.6 smoothing (top) and LER with 0.6 smoothing (bottom) of the fourth overfitting example (BulPhonC; colours were swapped due to a TensorBoard error).

For the first half of the training, the LER stayed relatively flat with a very slight decrease, followed by a sharp drop in the second half, reaching **0%** at epoch 1593.

The cost was decreasing with a steady rate up to epoch 715, when it experienced a sudden rise, followed by another period of decreasing with the same rate. Eventually it dropped to **2.21**. In this training example, there is no clear correlation between the performance of the cost and the LER value.

At inference, when using the model from checkpoint 1593 with LER **0%,** the proposed transcription is:

Original:	прилича на акварел в който всичко е загатнато и където не съществува преход между нюансите на охрата
Output from the system:	прилича на акварел в който всичко е загатнато и където не съществува преход между нюансите на охрата

5.0 DISCUSSION

5.1 Single Audio File Overfitting

The following table presents the lowest values reached for all training scenarios as well as the number of epochs it took:

Table 11: Comparison of the results from the overfitting process

	ID	Data set	Standard Deviation	Length (chars)	Lowest LER	Epoch (for lowest LER)	Lowest cost	Epoch (for lowest cost)
(1)	7800-283492-0003	LibriSpeech	3.695	161	**1.86 %**	1772 (983 + 789)	**7.4**	2000
(2)	6848-252322-0000	LibriSpeech	3.701	165	**4.24 %**	2000	**14.25**	2000
(3)	200-126784-0029	LibriSpeech	3.867	167	**5.98 %**	2000	**25.12**	2000
(4)	8324-286682-0038	LibriSpeech	3.871	162	**1.23 %**	1945	**7.85**	1945
(5)	141	BulPhonC	1.436	62	**0 %**	643	**0.2**	1560
(6)	191	BulPhonC	1.719	65	**0 %**	374	**1.07**	910
(7)	17	BulPhonC	3.027	93	**0 %**	1600	**0.01**	1880
(8)	24	BulPhonC	2.856	100	**0 %**	1593	**2.21**	1900

The first identified relationship in the data presented above was the one between the length of the sentences and their standard deviation, showing that the standard deviation generally increases proportionally to the sentence length (*see Fig. 37*).

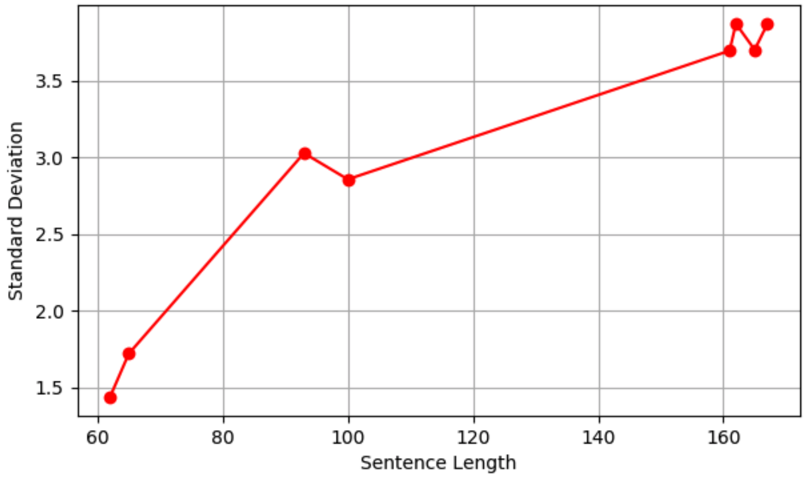


Figure 37: Relationship between sentence lengths and SDs for the eight tested files.

To ensure that this result is not coincidental, the same relationship was examined for the whole data sets (*see Fig. 38, 39*). From the results it can be confirmed that even though fluctuating, the standard deviation generally grows proportionally to the sentence length for both data sets. The most probable reason for this relationship is that the most common letters like '*a*', '*e*' and '*t*' build up in longer sentences and distort the flatness of the distribution.

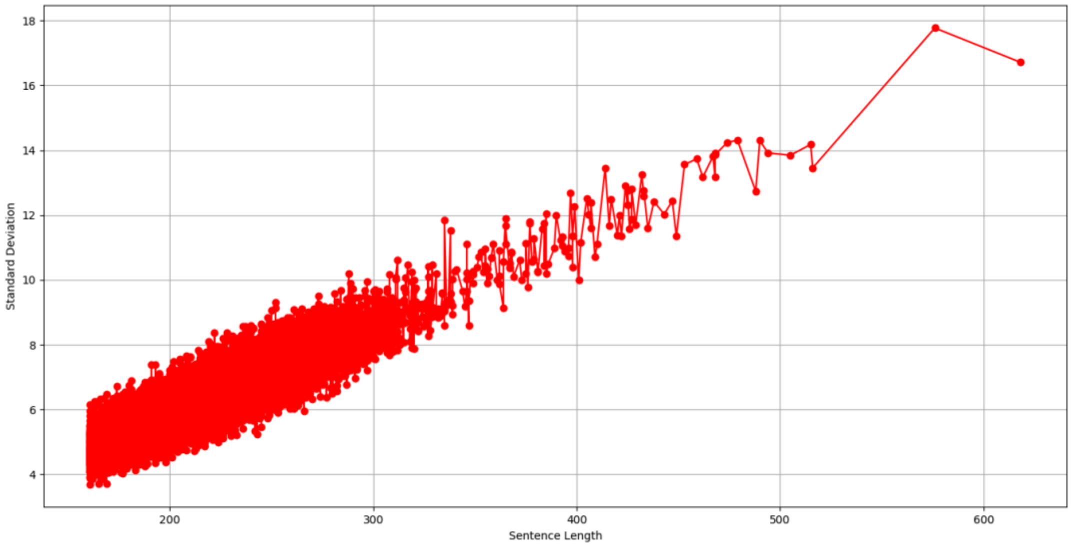


Figure 38: Relationship between sentence lengths and SDs in LibriSpeech

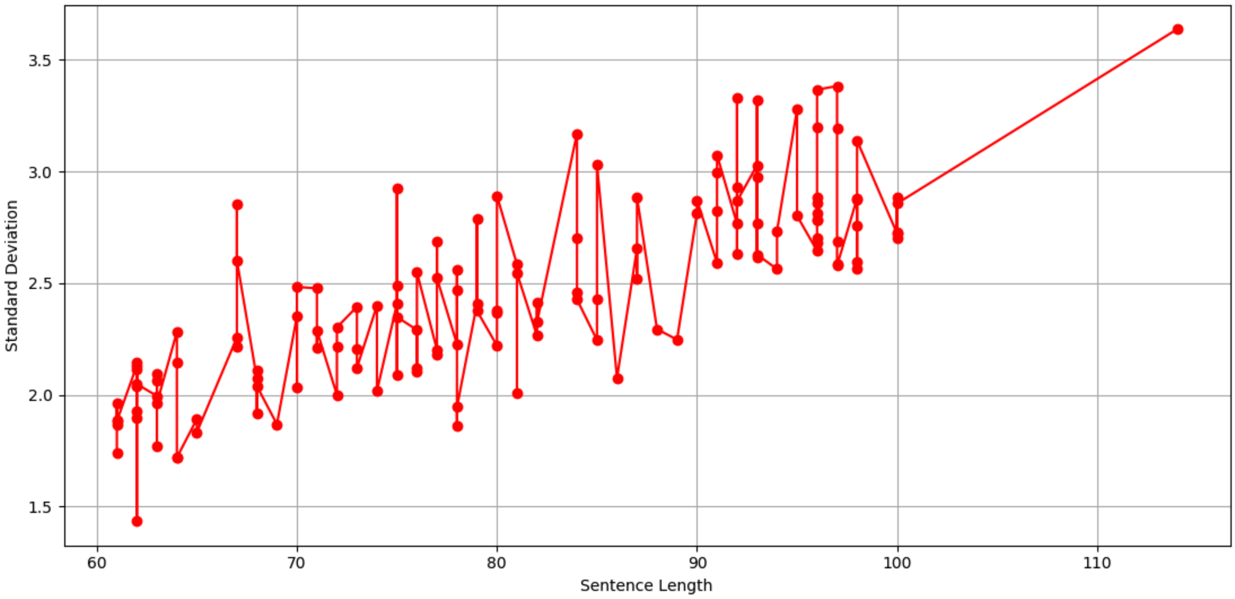


Figure 39: Relationship between sentence lengths and SDs in BulPhonC

An important observation is that for all four audio files from BulPhonC, the LER value managed to converge to 0% before the end of the training loop and all of their costs dropped in the range 0.01 – 2.21. As opposed, none of the trainings on the LibriSpeech files resulted in a LER value of 0% and the lowest cost value was 7.4. The main reason behind those results is that the sentences from LibriSpeech have a bigger length and therefore larger standard deviation, which results in slower convergence. It is assumed that the LER for the LibriSpeech sentences would also converge to 0% if trained for enough epochs. Furthermore, looking at the two examples from BulPhonC with the shortest lengths, it can be noticed that their LER values converge to 0% for only 374 and 643 epochs, which only confirms the relationship between sentence length (therefore size of audio file) and speed of convergence.

Following from that relationship, it can be assumed that for an equal number of epochs in large-vocabulary training, the system would have a greater accuracy of prediction for BulPhonC than LibriSpeech, since the standard deviation of the average distribution of letters in LibriSpeech is 4.272, compared to only 1.496 for BulPhonC, which is almost three times smaller.

The apparent advantage of training on data sets like LibriSpeech, which have longer sentences and bigger audio files is that at inference time, the model would be able to recognize longer speech as a single entity, while models trained on short sentences or single word commands, would only be able to recognize new sentences of the same length. However, this problem can possibly be overcome by having a flexible batch size at inference and in the case of longer input sentences, they would be split into two or more parts and fed into the model as a batch with multiple components. After the prediction is being made, the results would simply be concatenated and passed to the CTC layer to produce readable output.

Most importantly, the overfitting process showed that the system can be trained and can predict equally well in both English and Bulgarian with no changes in the model. The only differences in code are in the training and inference execution scripts and most of them are related to the different ways of storing the data from the different data sets as well as the output mapping, which is different for the two alphabets. The only change that the user would have to make when switching between the languages would be to specify the name of the dataset in the configuration file.

5.2 Additional Observations

Looking back at the results from sections *4.1* and *4.2* (large- and small-vocabulary training)*,* a general trend in the learning curve can be identified. In most of the training scenarios, both the cost and LER values started with a steep initial drop, followed by a plateau. It is seen as a common occurrence, which can either be overcome by fine tuning the hyper-parameters (especially the learning rate), or by running the training for a large number of epochs. Furthermore, the graphs showed very noisy results with a lot of fluctuations, which is a result of the small batch sizes being used for training. Using a larger batch size would smoothen the results and possibly improve the learning curve.

Even though the initial large-vocabulary and small-vocabulary training showed some improvement in the system's prediction accuracy and reached a minimum LER of 67%, the desired level of accuracy was not reached. However, during the process, two important observations were made.

a) Learning rate – file size relationship

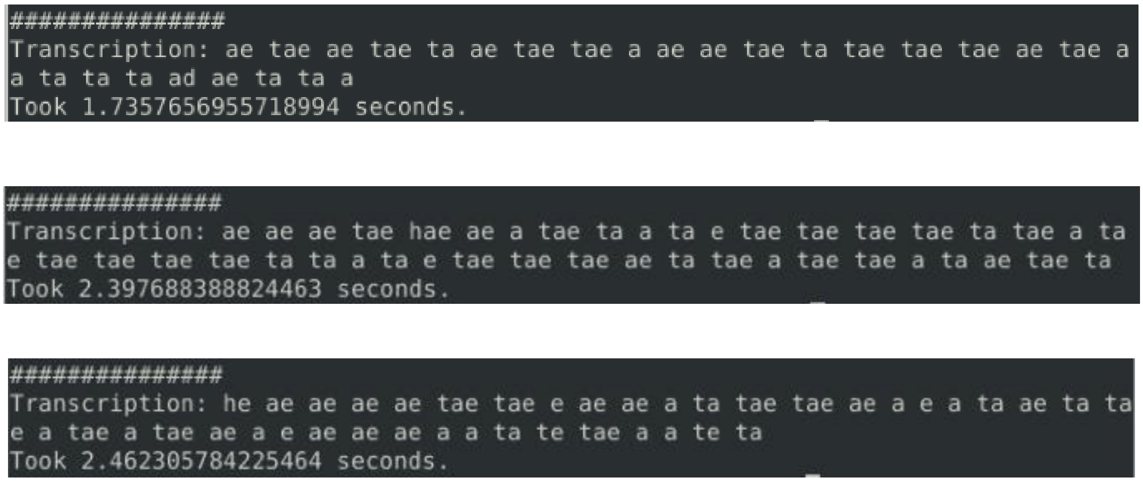
First, a proportional relationship between the learning rate and the maximum audio file size was identified. As described in section *2.3.2.4*, the learning rate is a hyper-parameter which specifies the step size of the cost optimizer. Therefore, the learning rate is directly related to the cost value. When comparing the results from the figures in section *4.1*, *4.2* and *4.3,* a difference in the value ranges for the cost can be noticed. For large-vocabulary training, where audio files are large, the cost starts at 1000+ and a drop to under 200 is considered a huge improvement. As opposed, for small-vocabulary training where all audio files have a length of 1 second, the cost values were starting at only 20 or less and a drop from 20 to under 5 is considered equally as good. These big differences in the cost ranges alter the effect of the learning rate on the optimization process, since the learning rate is multiplied by a function of the cost. Therefore, when a cost with a very small value is being multiplied by a certain learning rate, the resultant step size is small, but when a huge cost value is being multiplied by the exact same learning rate, this results in a big step.

This explains why the exploding and vanishing gradient problems were present during the training process on the full-size LibriSpeech when learning rate was set to 0.005 or more and training on the reduced version of LibriSpeech, Speech Commands or other short sentences worked fine with learning rates of up to 0.02. The exploding gradient problem occurs when the learning rate is too big and the error gradient starts accumulating and resulting in large updates and large cost values (outputting ‘inf’). The vanishing gradient problem works analogically but resulting in infinitesimally small cost values. This observation proves the theory that setting the learning rate is subjective and largely depends on the training data being used.

b) Letter distribution effect on recognition

The second major observation was identified once the statistical analysis on the data sets was made. Looking back at section *4.1,* presenting the results from the large-vocabulary training experiments, it is seen that the LER could not be reduced to a low enough value, which would result in a sensible text output. When inference experiments were made, all the outputs were nonsense. However, a certain repetitiveness of letters could be noticed.

When testing the model from checkpoint 41 100 from the (8) training scenario in section *4.1,* which provides a LER value of 67 %, the predicted transcriptions for three random audio files (for this example: ‘3575-170457-000.flac’, ‘1919-142785-0011.flac’ and ‘1963-142776-0011.flac’) were:



```
###############
Transcription: ae tae ae tae ta ae tae tae a ae ae tae ta tae tae tae ae tae a
a ta ta ta ad ae ta ta a
Took 1.7357656955718994 seconds.
```

```
###############
Transcription: ae ae ae tae hae ae a tae ta a ta e tae tae tae tae ta tae a ta
e tae tae tae tae ta ta a ta e tae tae tae ae ta tae a tae tae a ta ae tae ta
Took 2.397688388824463 seconds.
```

```
###############
Transcription: he ae ae ae ae tae tae e ae ae a ta tae tae ae a e a ta ae ta ta
e a tae a tae ae a e ae ae ae a a ta te tae a a te ta
Took 2.462305784225464 seconds.
```

Figure 40: Transcriptions by model from checkpoint 41 100 (training (8) described in section 4.1)

Those transcriptions consist of only ‘*a*’, ‘*e*’ and ‘*t*’ with only 1 occurrence of ‘*h*’ or ‘*d*’. Figures *22* and *23* from section *4.3.1* show that the most commonly used letters in LibriSpeech are ‘*a*’, ‘*e*’ and ‘*t*’, being used on average 10.4, 16.5 and 12 times per sentence. Therefore, it can be assumed that the system has started learning that these letters are the most commonly used ones and has assigned high probabilities to them. This observation shows that the system has the potential to learn from data.

6.0 CONCLUSIONS

The main goal of this project was to develop, test and evaluate the performance of a bilingual end-to-end speech recognition system as described in the methodology. The results presented in this report demonstrate a number of approaches to testing different aspects of a deep learning system of this type, as well as the effects that various hyper-parameters and other configuration settings have on its performance.

Despite the inability to train a the model to the desired level, this study has identified a number of important findings. The relevance of the choice of learning rate and how this affects the performance of the system have been examined in detail. Furthermore, it was shown that the size, diversity and organization of training data are major factors when it comes to speed of convergence and prediction accuracy. A significant finding that emerged from this study is that the characteristics of the machine used for training can have a great impact on its performance, since they are directly related to some of the hyper-parameters like the maximum batch size or RNN size that can be handled. Most importantly, this study has proven that the modern end-to-end approach to speech recognition eliminates the necessity of specialized language knowledge and ensures the ability of a single model to be trained to recognize many languages.

Taken together, these results suggest that with modern technology, the development of such complex systems requires simpler structures with less components and the focus shifts from the detailed specifics of the current task to the more general techniques and concepts used in deep learning for various applications.
Furthermore, the results implicate that in modern deep learning systems, the quality and quantity of training data is equally as important as the design of the system itself.

Major limitations to this project were the limited available time for the completion of it and the lack of processing power necessary for training a model of this scale.

In conclusion, the testing observations and the success in the single audio file overfitting act as a proof that the training process is executing properly, and the system has the ability to adapt to input data in both Bulgarian and English language with no logical errors. Even though this is **not** a proof that the system will be able to learn successfully from large and complex data, it demonstrates a potential to do it, given enough processing power and the right hyper-parameters.

7.0 RECOMMENDATIONS

This project has a great potential for further development, which was out of its scope due to resource and time constraints. The following are a few recommendations for possible future work on the project:

- Using a more powerful machine with at least 32GB RAM allowing larger models and bigger batch sizes and using a graphics card for faster processing.
- Utilizing distributed parallel training on multiple machines for faster processing.
- Creating a small-vocabulary data set from BulPhonC, by splitting its training examples into separate words to form a small data set. This could be done by an automated script that reads the segmentation files which come with BulPhonC.
- Executing extensive large-vocabulary training on BulPhonC.
- Adding an n-gram word or character level language model in the CTC layer to increase the prediction accuracy.
- Executing the training process for a larger number of epochs.
- Trying more combinations of hyper-parameters. Having many hyper-parameters results in a large number of possible combinations, which require longer time to be tested.

LIST OF REFERENCES

Abdel-Hamid, O., Mohamed, A., Jian, H., Deng, L., Penn, G. and Yu, D., (2014). *Convolutional Neural Networks for Speech Recognition.* Transactions on audio, speech and language processing. **22**(10), 1533-1545

Amodei, D. et al., (2016). Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. Proceedings of The 33rd International Conference on Machine Learning, PMLR 48, 173-182

Atal, B. S. and Hanauer, S. L. (1971) Speech Analysis and Synthesis by Linear Prediction of the Speech Wave, The Journal of the Acoustical Society of America , **50(**2), 637-655

Bahdanau, D. Chorowski, J. Serdyuk, D. Brakel, P. and Bengio, Y., (2016). End-to-end attention-based large vocabulary speech recognition. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* Shanghai, 4945-4949.

Bahdanau, D., Cho, K. and Bengio, Y., (2015). Nerual machine translation by jointly learning to align and translate. In Proc. Of the 3rd ICLR. arXiv: 1409.0473

Baker, J.K., (1975), Stochastic modelling for automatic speech recognition. *Speech Recognition*. D. R. Reddy (editor), New York: Academic Press

Bengio, Y., De Mori, R., Flammia, G., and Kompe, R. (1991), Phonetically motivated acoustic parameters for continuous speech recognition using artificial neural networks. In Proceedings of EuroSpeech’91.

Bengio, Y., De Mori, R., Flammia, G., and Kompe, R. (1992). Neural network-Gaussian mixture hybrid for speech recognition or density estimation. *NIPS 4*. 175–182.

Bhande, A., (2018). [*Examples of Underfitting, Overfitting and Normal Fitting*] [digital image]. [Viewed 28 Nov 2018]. Available from: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

Bourlard, H. and Wellekens, C., (1989), Speech pattern discrimination and multi-layered perceptrons. *Computer Speech and Language*. **3**, 1–19.

Cho, K., Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio Y., (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078 [cs.CL]

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K. and Bengio Y., (2015) Attention-based models for speech recognition. NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1. 577-585

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., (2014) Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv:1412.3555 [cs.NE]

Davis, K.H., Biddulph, R. and Balashek, S. (1952) Automatic Recognition of Spoken Digits, The Journal of the Acoustical Society of America, **24**(6), 627-642

Dempster, A. P.*,* Laird, N. M. and Rubin, D. B., (1977)*,* Maximum Likelihood from Incomplete Data via the EM Algorithm*.* Journal of the Royal Statistical Society, Series B*.* **39** (1), 1–38

Donahue, C., McAuley, J. and Puckette, M., (2018) Adversarial Audio Synthesis. arXiv:1802.04208 [cs.SD]

Ghahramani, Z. (2001) An Introduction to Hidden Markov Models and Bayesian Networks, *International Journal of Pattern Recognition and Artificial Intelligence,* **15**(1), 9-42

Goodfellow, I., Bengio, Y. and Courvile, A., (2017). Deep Learning. Cambridge, MA: MIT Press

Graves, A. and Jaitly, N., (2014). Towards end-to-end speech recognition with recurrent neural networks. In Proceedings of the 31st International Conference on Machine Learning (ICML-14). 1764–1772

Graves, A., (2012), *Supervised Sequence Labelling with Recurrent Neural Networks,* volume 385 of *Studies in Computational Intelligence.* New York: Springer-Verlag Berlin Heidelberg

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014) Deep speech: Scaling up end-to-end speech recognition. http://arxiv.org/abs/1412.5567 [cs.CL]

Hateva, N., Mitankin, P. and Mihov, S., (2016). BulphonC: Bulgarian Speech Corpus for the Development of ASR Technology. Language Resources and Evaluation Conference. Portoroz, Slovenia

Hinton, G. *et al*., (2012), "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". *IEEE Signal Processing Magazine.* **29**(6), 82-97

Hori, T., Cho, J. and Watanabe, S., (2018). End-to-end Speech Recognition with Word-based RNN Language Models. arXiv:1808.02608 [cs.CL]

Itakura, F. and Saito, S. (1970) A Statistical Method for Estimation of Speech Spectral Density and Formant Frequencies, Electronics and Communications in Japan, **53A**, 36-43

J. M. Baker *et al*., (2009), Developments and directions in speech recognition and understanding, Part 1 [DSP Education]. *IEEE Signal Processing Magazine.* **26**(3), 75-80

Jelinek, F., (1976), Continuous speech recognition by statistical methods. *Proc. IEEE*. **64**(4), 532-557

Juang, B. and Rabiner, L. (2005). Automatic Speech Recognition - A Brief History of the Technology Development.

Kiefer, J. and Wolfowitz, J. (1952) *Stochastic Estimation of the Maximum of a Regression Function.* The Annals of Mathematical Statistics. **23**(3), 462-466

Kim, S. and Seltzer, M. L., (2017) Towards Language-Universal End-to-End Speech Recognition. arXiv:1711.02207 [cs.CL]

Konig, Y., Bourlard, H., and Morgan, N., (1996), REMAP: Recursive estimation and maximization of a posteriori probabilities – application to transition-based connectionist speech recognition. In D. Touretzky, M. Mozer, and M. Hasselmo (editors), Advances in Neural Information Processing Systems 8 (NIPS'95). Cambridge, MA: MIT Press

Liptchinsky, V., Synnaeve, G. and Collobert, R., (2017). Letter-Based Speech Recognition with Gated ConvNets. arXiv:1712.09444 [cs.CL]

Maas, A. & Xie, Z. & Jurafsky, D. & Ng, A., (2015). Lexicon-Free Conversational Speech Recognition with Neural Networks. NAACL. 345-354

MacQueen, J. B. (1967). *Some Methods for classification and analysis of Multivariate Observations*. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. University of California Press. 281-297.

Miao, Y., Gowayyed, M., and Metz, F., (2015). EESEN: End-to-end speech recognition using deep rnn models and wfst-based decoding. In ASRU.

Mitchell, T.M., (1997). Machine Learning. New York: McGraw-Hill

Mnih, V., Heess, N., Graves, A., et al., (2014) Recurrent models of visual attention. In Proc. Of the 27th NIPS. arXiv: 1406.6247

Mohamed, A., Dahl, G. and Hinton, G., (2012), Acoustic modeling using deep belief networks. IEEE Trans. Audio Speech Lang. Processing. **20**(1), 14–22

Mohamed, A., Sainath, T. N., Dahl, G. E., Ramabhadran, B., Hinton, E. and Picheny, M., (2011). Deep belief networks using discriminative features for phone recognition. Proc. ICASSP. 5060-6063

Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcouglu, K., (2016). WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499 [cs.SD]

Panayotov, V., Chen, G., Povey, D. and Khudanpur, S., (2015). Librispeech: An ASR corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brisbane, QLD, 5206-5210

Rabiner, L. and Juang, B. (1993). *Fundamentals of speech recognition*. Delhi: Pearson Education.

Robbins, H. and Monro, S. (1951) *A Stochastic Approximation Method.* The Annals of Mathematical Statistics. **22**(3), 400-407

Robinson, A. J. and Fallside, F., (1991), A recurrent error propagation network speech recognition system. *Computer Speech and Language.* **5**(3), 259–274

Sutton, R. S. and Barto, A., (1998). Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press

Toshniwal, S., Sainath, T. N., Weiss, R. J., Li, B., Moreno, P., Wenstein, E. and Rao, K., (2017). Multilingual Speech Recognition With A Single End-to-End Model. arXiv:1711.01694 [eess.AS]

Waibel, A., Hanazawa, T., Hinton, G. E., Shikano, K., and Lang, K., (1989), Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing.* **37**, 328–339.

Warden, P., (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. arXiv:1804.03209 [cs.CL]

Xu, K., Ba, J., Kiros, R., et al., (2015). Neural image caption generation with visual attention. In Proc. of the 32nd ICML. arXiv: 1502.03044

Young, S., (1996), Large Vocabulary Continuous Speech Recognition: a Review. *IEEE Signal Processing Magazine*. **13(**5), 45

Zeghidour, N., Usunier, N., Synnaeve, G., Collobert, R. and Dupoux E., (2018). End-to-End Speech Recognition from the Raw Waveform. arXiv:1806.07098 [cs.CL]

APPENDICES

Will add all of my code here…