

**Bilingual End-to-End Deep Learning**

**Speech Recognition System**

**Nikola Tonev**

**15119311**

BSc (Hons) Sound Engineering and Production

**Supervisor: Jason Hockman**

May 2019

# ABSTRACT

“a **summary** of the report (100-200 words), which should fully encapsulate the content of the project, being informative, interesting and for appropriate aspects, quantitative” (General Guidance)

**Further Information**

Read Learning Centre tip sheet: Writing up the report: Writing an Abstract

**Example**

Birmingham based registered youth charity St Basils’ required an Information System (IS) that could be used as an interesting medium for communicating relevant resident information on the World Wide Web. Rapid Application Development (RAD) Methodology – Throwaway Prototyping, Computer Aided Software Engineering (CASE) tools helped to develop this system at a fast rate and to accomplish the project in the short time scale given. With the design stage of the Systems Development Life Cycle (SDLC) complete, a flexible, powerful yet easy to use server-side scripting language was required. PHP was chosen as it fulfilled the previous criteria. To complement PHP, MySQL database server was chosen to store the relevant data for the website, because of the speed at which PHP and MySQL are able to interact. A highly accessible web site was created to meet the charity’s needs.

# 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

[ABSTRACT ii](#_Toc6749491)

[ACKNOWLEDGEMENTS iii](#_Toc6749492)

[CONTENTS PAGE iv](#_Toc6749493)

[GLOSSARY vi](#_Toc6749494)

[LIST OF DIAGRAMS AND TABLES vii](#_Toc6749495)

[1.0 INTRODUCTION 1](#_Toc6749496)

[1.1 Problem Definition 1](#_Toc6749497)

[1.2 Scope 2](#_Toc6749498)

[1.3 Rationale 2](#_Toc6749499)

[1.4 Aims 3](#_Toc6749500)

[1.5 Objectives 3](#_Toc6749501)

[1.6 Background Information 4](#_Toc6749502)

[2.0 REVIEW OF EXISTING KNOWLEDGE 5](#_Toc6749503)

[2.1 Speech Recognition – History and Importance 5](#_Toc6749504)

[2.2 Critical Review of Previous Research 6](#_Toc6749505)

[2.3 Machine Learning 12](#_Toc6749506)

[2.3.1 Basics 12](#_Toc6749507)

[2.3.2 Neural Networks 13](#_Toc6749508)

[3.0 METHODOLOGY 18](#_Toc6749509)

[3.1 Introduction to the Methodology 18](#_Toc6749510)

[3.2 Literature Search Methodology 19](#_Toc6749511)

[3.3 Software and Hardware Resources 19](#_Toc6749512)

[3.3.1 Programming Language and Libraries 19](#_Toc6749513)

[3.3.2 Hardware 21](#_Toc6749514)

[3.4 Model Architecture 22](#_Toc6749515)

[3.4.1 Feature Representation 24](#_Toc6749516)

[3.4.2 Neural Network 24](#_Toc6749517)

[3.4.3 Connectionist Temporal Classification (CTC) 29](#_Toc6749518)

[3.5 Data Set 31](#_Toc6749519)

[3.5.1 Large-vocabulary 31](#_Toc6749520)

[3.5.2 Small-vocabulary 32](#_Toc6749521)

[3.6 Testing and Evaluation Methodology 33](#_Toc6749522)

# 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**

[Figure 1: A high level statistical speech recognition system diagram 11](file:////Users/Kokovich/Desktop/FYP_DRAFT.docx#_Toc6752015)

[Figure 2: A generic artificial neural network structure 13](#_Toc6752016)

[Figure 3: A generic recurrent neural network structure with one hidden layer 15](#_Toc6752017)

[Figure 4: A generic example of 2-D convolution 16](#_Toc6752018)

[Figure 5: Illustration of 'same' padding.. 17](#_Toc6752019)

[Figure 6: A high-level diagram of the model architecture for all three modes of the system: 23](#_Toc6752020)

[Figure 7: A high-level diagram of all layers in the neural network. 25](#_Toc6752021)

[Figure 8: A generic model of a bi-directional recurrent neural network. 27](#_Toc6752022)

[Figure 9: Gated Recurrent Unit (GRU). 28](#_Toc6752023)

[Figure 10: A basic example of calculating edit distance 33](#_Toc6752024)

**List of Tables**

[Table 1: Pros and Cons of the most popular Python libraries used for ML 20](#_Toc6752233)

[Table 2: Details of the software resources used for this project. 21](#_Toc6752234)

[Table 3: Specifications of the server computer. 21](#_Toc6752235)

[Table 4: Details for large-vocabulary data sets that will be used in the project. 32](#_Toc6752236)

[Table 5: Details for small-vocabulary data sets that will be used in the project. 32](#_Toc6752237)

# 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.

## 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).

## 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.

## 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.

## 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.

## 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. based on Baidu’s Deep Speech 2 system. (🡨 should I add the text in red to the sentence?)
* 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.
* Critically evaluate the results and project limitations and propose further improvements.
* Develop a web application for easy and straightforward user interaction with the system. (if I have time; I’ve worked a lot with node.js, html and css, so that shouldn’t take long if I have a few spare days)

## Background Information

Should I add any information about the cyrillic alphabet and Bulgarian language here and maybe explain the major differences from English or should I avoid this section as a whole? Background knowledge on the history of speech recognition is included in Review of Existing Knowledge (next section).

# REVIEW OF EXISTING KNOWLEDGE

## 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.

## 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 to the original signal  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 (*y1,…, yt*) generated from the sequence (*h1,…,hn*), 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.

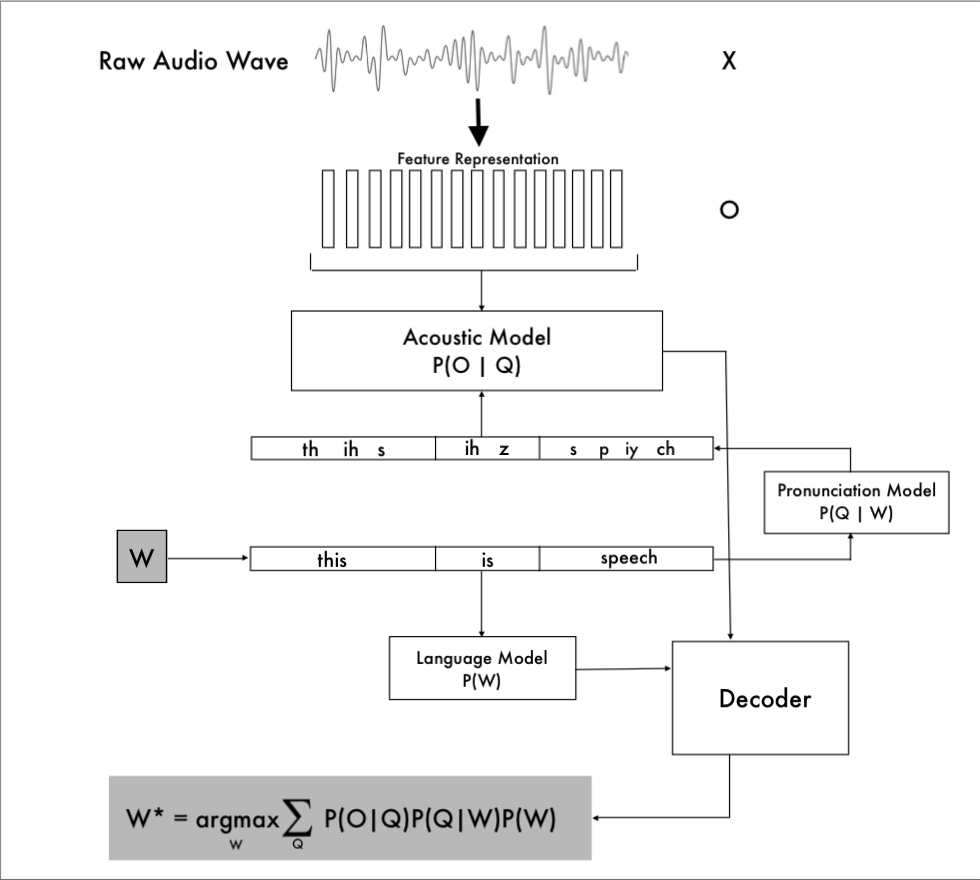


Figure 1: A high level statistical speech recognition system diagram, showing the computation of the maximum value of the probability P(W|X) of word sequence W given the acoustic input X. The diagram was composed based on a similar one provided by Steve Young (1996) in ‘Large Vocabulary Continuous Speech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

## Machine Learning

### 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:

* **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.

* **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).

* **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).

### Neural Networks

#### 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:

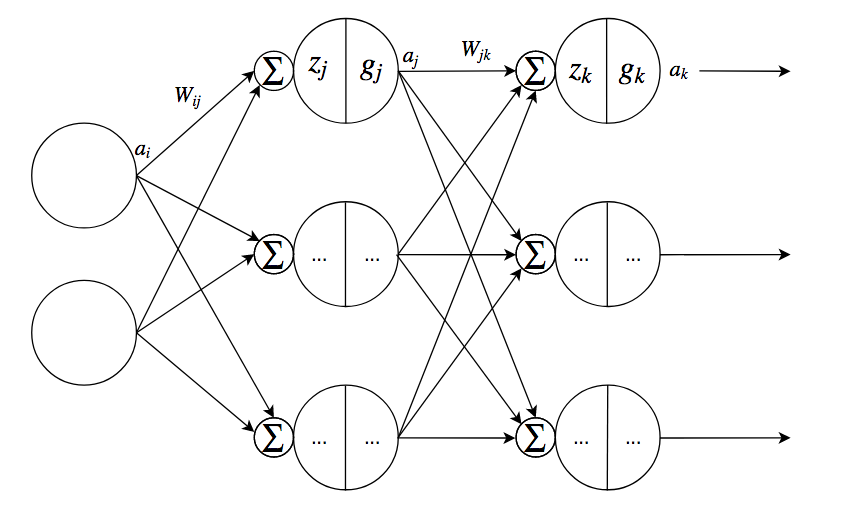


Figure 2: A generic artificial neural network structure.

where

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

1. The input signal *ai* for each node in the input layer is multiplied by a weight matrix *Wij* which connects the input layer with the hidden layer.
2. Those products are then summed and a bias value is added. This operation forms the input of each node of the hidden layer *zj* .
3. The input of each hidden node is then multiplied by the layer’s nonlinear activation function *gj* to produce the output of the layer - *aj* .
4. Similar to the previous steps, the output of each hidden node is multiplied by the weight matrix *Wjk* , a bias *bk* is added and the signals are summed to produce the input of the output node – *zk* .
5. *Zk* is then multiplied by its activation function *gk* to produce the neural network’s output values *ak..*

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.

#### 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 (*xt*), but also the value of the hidden node for the previous time step (*ht-1*) in order to produce an output value (*see Eq. 1 & Fig. 3*).

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

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

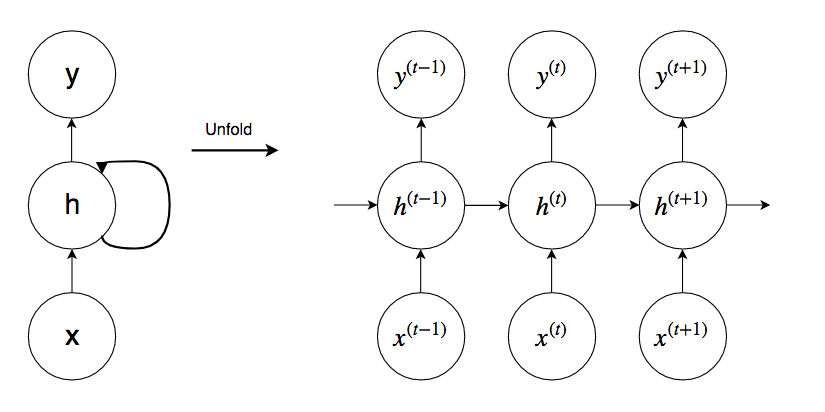


Figure 3: A generic recurrent neural network structure with one hidden layer

#### 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*).

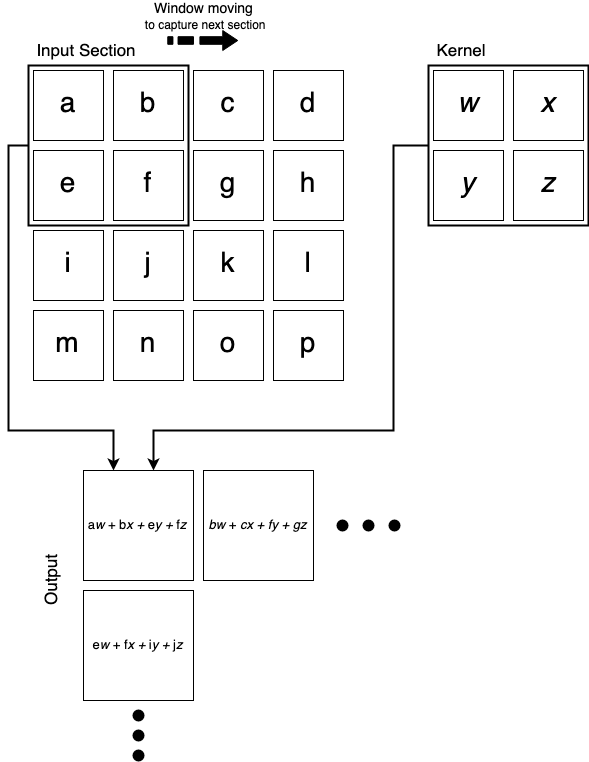


Figure 4: A generic example of 2-D convolution with no padding and kernel of shape [2, 2].

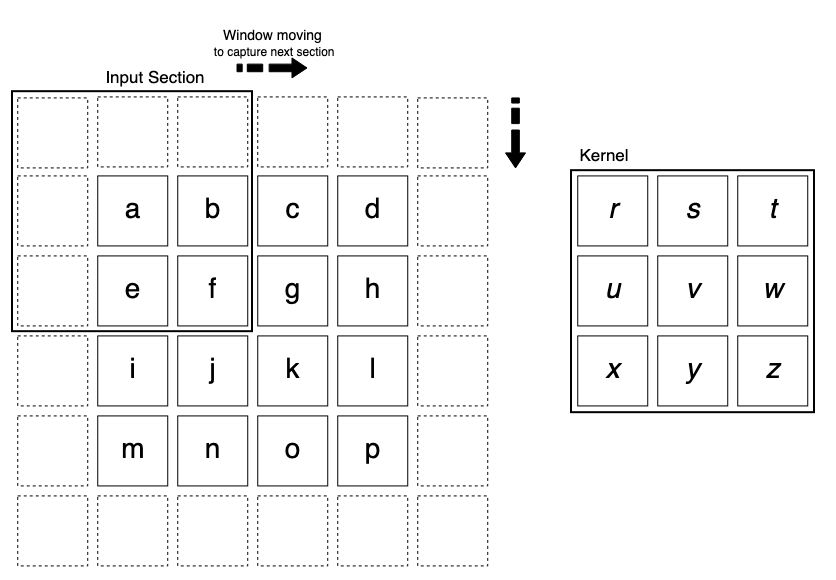


Figure 5: Illustration of 'same' padding. The output matrix will have the same dimensionality as the input one - 4x4.

#### 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*).

|  |  |  |
| --- | --- | --- |
|  |  | (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*).

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

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

# METHODOLOGY

## 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.

## 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” |

## Software and Hardware Resources

### 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) |

### 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.

## Model Architecture

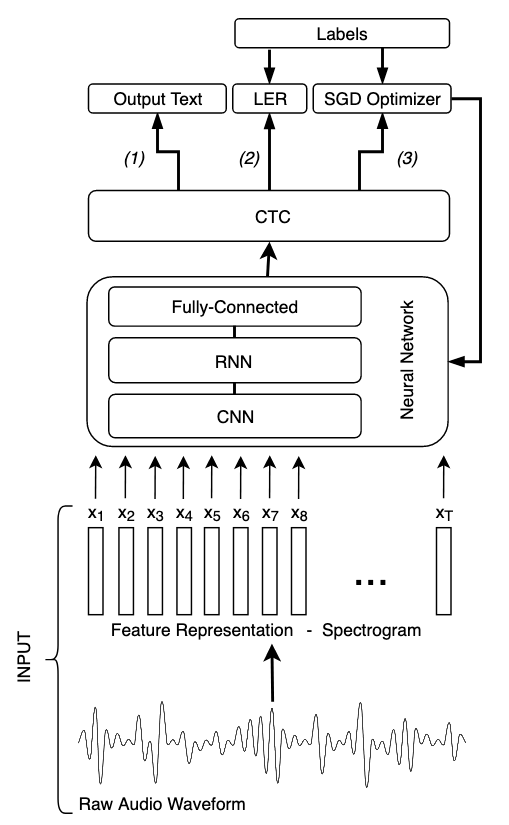


Figure 6: A high-level diagram of the model architecture for all three modes of the system:

(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.

### 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.

### 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.

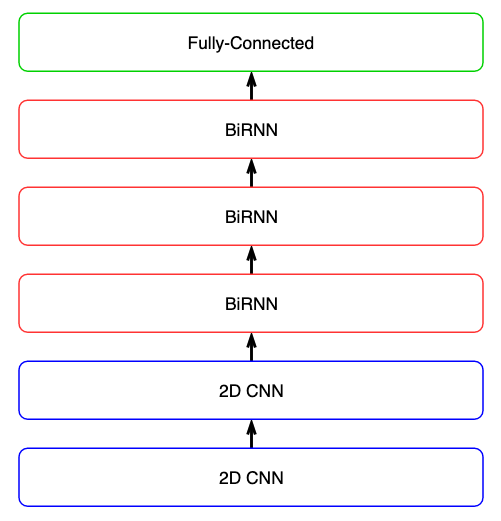


Figure 7: A high-level diagram of all layers in the neural network.

#### 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.

#### 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 *yt* for t = 1 to T for a single layer BiRNN are calculated as follows:

where

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.

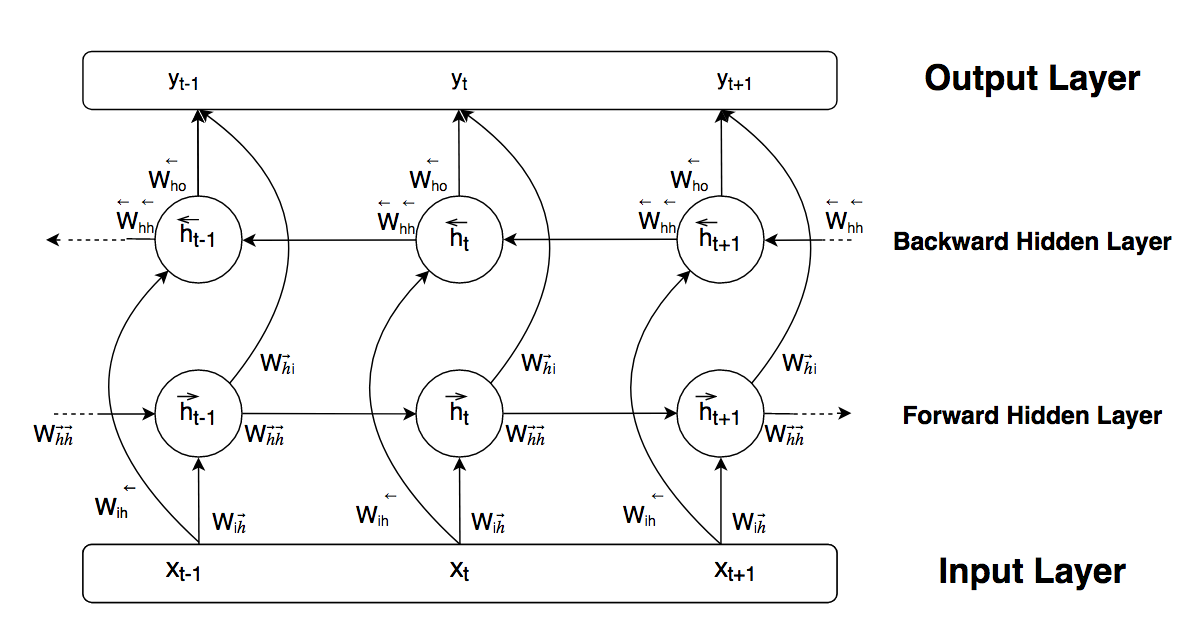


Figure 8: A generic model of a bi-directional recurrent neural network.

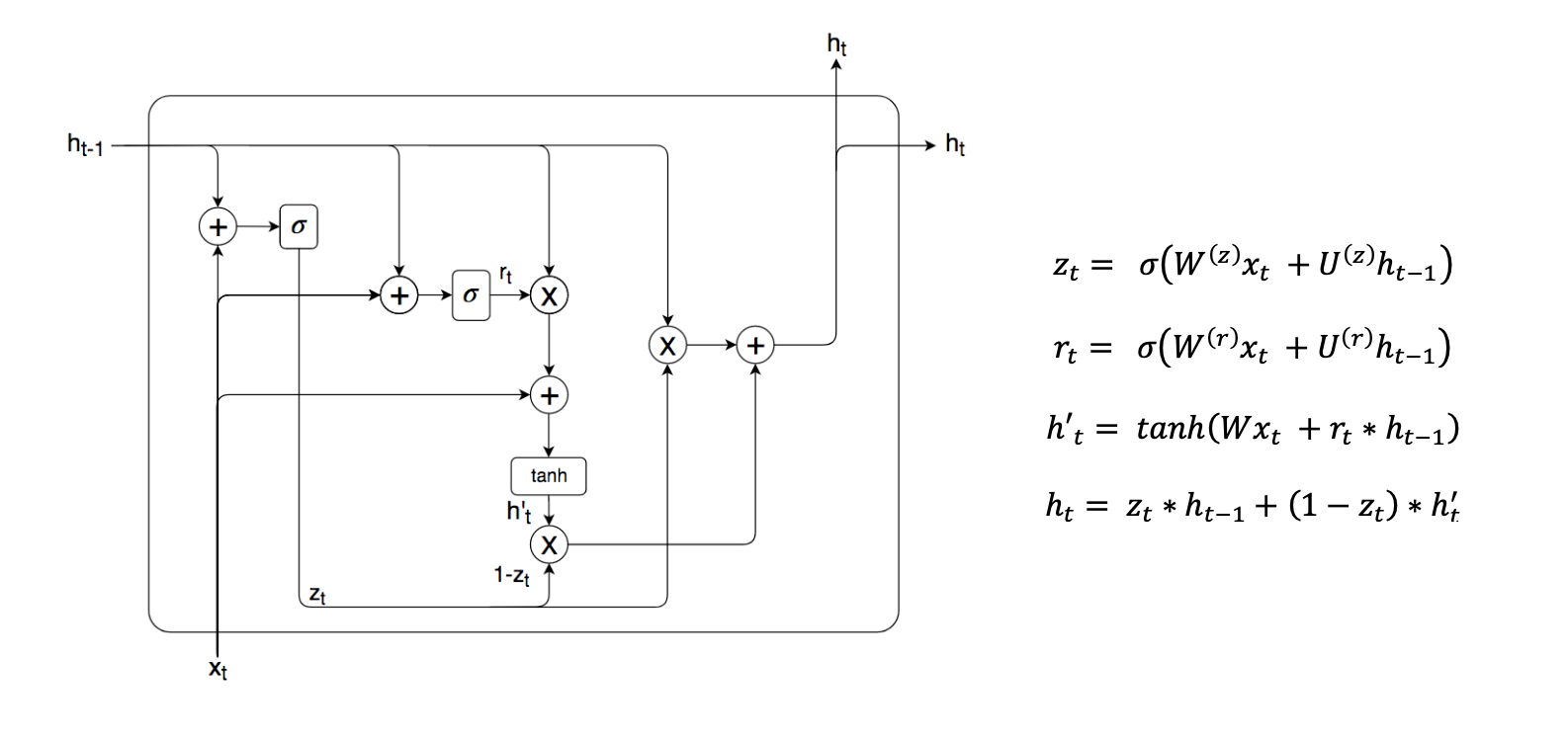


Figure 9: Gated Recurrent Unit (GRU).

#### 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(kt | x)* where  *kt* represents each character*.*

### 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:

|  |  |  |
| --- | --- | --- |
|  |  | (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. (HHHH\_E\_\_LLL\_LO\_\_)  = “HELLO”).

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

1. **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”):

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

|  |  |  |
| --- | --- | --- |
|  |  | (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 :

|  |  |  |
| --- | --- | --- |
|  |  | (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.

1. **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.

1. **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.

## 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.

### 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*).

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 |

### 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 |
| BulPhonC EDITED | Bulgarian | --- | --- | --- | --- |

## Testing and Evaluation Methodology

After the training process was completed, experiments were performed in order to test and evaluate the performance of the model described in section 3.4. As mentioned in section 3.5, the testing will be performed on 20% of each data set, containing only data which was not observed by the model while training.

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.

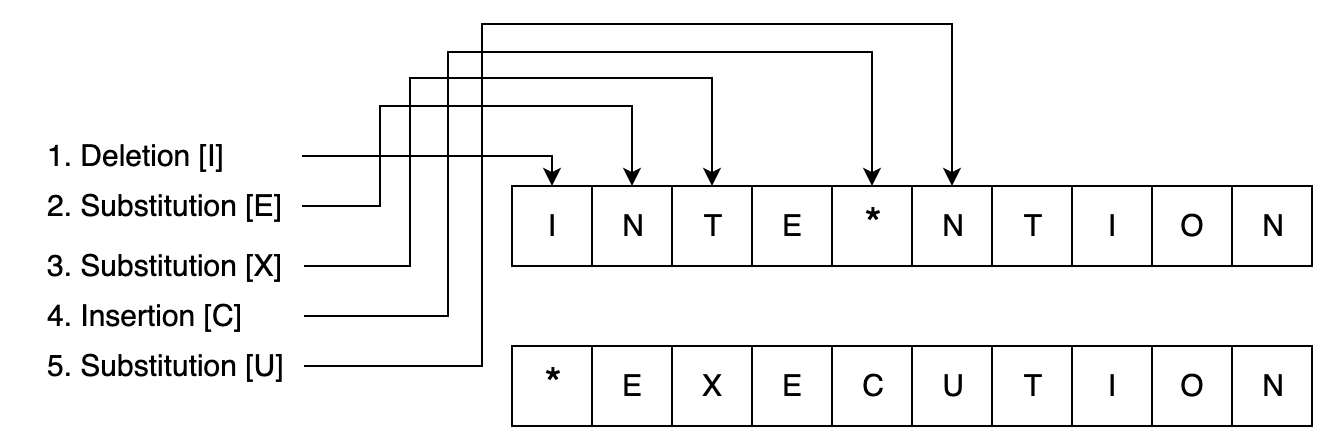


Figure 10: A basic example of calculating edit distance.