# BSP S3 - Speech Recognition for Luxembourgish by a Recurrent Neuronal Network

**Thursday 12th December, 2019 - 17:15**

Le Minh Nguyen
*University of Luxembourg*
*Email: le.nguyen.001@student.uni.lu*

Vladimir Despotovic
*University of Luxembourg*
*Email: vladimir.despotovic@uni.lu*

***Abstract*—Recurrent Neural Networks (RNN) use their internal state to operate on data time-series. One of the domains in which RNNs are applied is speech recognition. In this paper, the focus will be on the task of end-to-end isolated word recognition. We apply RNN and feedforward Neural Networks as the baseline for the recognition and compare their performance.**

## 1. Introduction

This paper presents the Bachelor Semester Project (BSP) made by Le Minh Nguyen together with Vladimir Despotovic as his motivated tutor. The project is divided into two parts. In the first period, we implement our classification model to recognize spoken words. The first part contains the following concepts:

- Data collection and preprocessing.
- Deep Neural Networks architecture: Deep Feedforward Neural Networks and Recurrent Neural Networks, which are presented in the popular Deep Learning textbook [1].

In the second part, we explain the deep Neural Networks architecture and compare their performance obtained during our classification experiment.

## 2. Project description

### 2.1. Domains

- Speech Recognition
- Artificial Neural Networks
- Deep Learning
- Data preprocessing
- Training set and testing set
- Python
- Keras

**2.1.1. Scientific.** The scientific aspects covered by this Bachelor Semester Project are the concepts of speech recognition and Deep Learning. Different Artificial Neural Networks architectures are investigated for the end-to-end isolated word recognition task.

**Speech Recognition.** The objective of speech recognition is to map an audio signal which contains a set of spoken natural language expressions to the matching sequence of words produced by the speaker. In the past, Automatic Speech Recognition (ASR) was made up of different modules such as complex feature extraction, acoustic models, language and pronunciation models. [2] Sequential models, such as Hidden Markov Models (HMM), in combination with a pre-trained language model, were used to map sequences of phones to output words. [3] A different approach is to build ASR models end-to-end. With deep learning, it replaces most of the modules with a single module. This alternative method will be the main task of this report.

**Artificial Neural Networks (ANN).** ANNs are computing systems inspired by the biological brain. These systems are based on a set of connected units called artificial nodes. Each connection can transmit *signals* between units. A unit can process the signal and transmit it to another unit.

**Deep Learning.** This field deals with learning by decomposing a task's input into smaller and simpler compositions. With Deep Learning, computing systems can build complex concepts from a composition of simpler concepts.

**2.1.2. Technical.** The technological aspect which is covered in this project is the data collection, feature extraction and implementation of our classification model.

**Data preprocessing.** Data preprocessing is an important phase in machine learning. It ensures the quality of the gathered data by eliminating irrelevant and redundant information. Data preprocessing contains tasks such as cleaning, instance selection, normalization, feature extraction and selection. We will focus on feature extraction in this paper with the presentation of Mel-frequency cepstral coefficients (MFCCs) in Section 4.2.1 which are used as features extracted from the speech signal.

**Training set and testing set.** For a computing system to learn from and make predictions on data, a mathematical

model is built from input data. This input data used to create the model consists of two datasets: The training and testing set. The training set contains pairs of an input vector, which represent features and an output vector often called the labels. With the training set, the model learns to map the input vector to the labels. On the other hand, the testing set evaluates how well the model generalizes the prediction over the dataset previously not seen by the model.

**Python.** This is a programming language which is interpreted, high-level and general-purpose. [4]

**Keras Library.** *Keras* is a high-level Neural Networks API written in Python. It is designed for easy-to-use and efficient experimentation with Deep Learning. [5]

### 2.2. Targeted Deliverables

**2.2.1. Scientific deliverables.** One of the main deliverables is to present how we extract the features from the dataset with Mel-frequency cepstral coefficients (MFFC). Additionally, we present the notions of Deep learning. This paper gives a brief introduction to Artificial Neural Networks (ANN) and their application for classification. Further, we extend this scientific presentation by diving deeper into three different ANNs; Feedforward Neural Networks, Recurrent Neural Networks (RNN) and Long short-term memory (LSTM) as a special case of RNN. Finally, we compare their performances obtained after being trained on the given audio dataset.

**2.2.2. Technical deliverables.** The other main deliverable for this paper is the implementation of an end-to-end speech recognition model based on the three Neural Networks mentioned in the section above. Additionally, we collect a small dataset of Luxembourgish spoken words $w \in \{'0', \ldots, '9', 'moien'\}$ to train our model to recognize these words. Since, to the best of our knowledge, there are no publically available datasets for speech recognition of the Luxembourgish language.

### 2.3. Constraints

For this BSP, we set the different constraints for the ANNs, for the Speech recognition for the Luxembourgish vocal dataset and the classification model's implementation.

**Data set.** First, we focus on training our model on a dataset containing English spoken numbers. [6] After having trained the model successfully on this dataset, we train it on the smaller Luxembourgish dataset and analyse its prediction accuracy. Since we aren't able to collect as much voice recordings as the English data set, we should not expect high accuracy.

**Speech recognition.** Since no datasets with continuous sentences of the Luxembourgish Language exists, we choose to work with isolated word recognition instead of continuous speech recognition. In isolated word recognition, words are separated by pauses.

**Speaker dependent.** To simplify the collection of the Luxembourgish dataset, we collect the audio recordings from one person which makes the isolated word recognition speaker-dependent.

**ANNs implementation.** We do not implement the Artificial Neural Networks architecture since existing deep learning libraries such as Keras are already matured. Thereby, we focus on explaining how the APIs work and how to use them efficiently in our model.

## 3. Pre-requisites

To start on the project, certain skills in programming and mathematics are required. In particular, the preliminary requirement of the project is as follow:

- Understanding of vector and matrix algebra.
- Introductory course in Python.
- Software development.
- Knowledge of probability and statistics, but it is not mandatory.

### 3.1. Scientific pre-requisites

**Linear Algebra.** is a sub-field of mathematics, which works with vectors and matrices. Since the input of deep learning is data that are transformed into structures of rows and columns, linear algebra is one of the key foundations of deep learning. It is used to describe the operations of the deep learning algorithms, and implement the algorithms in code. A feedforwad ANN can be represented as a composite function which multiplies some matrices and vectors together. All tasks in deep learning relate to linear algebra, from data preprocessing to the deep learning algorithms. [1]

### 3.2. Technical pre-requisites

**Python.** Python is an interpreted, high-level and general-purpose programming language, which is conceived in the late 1980s and released in 1991 by Guido van Rossum. [7] Its design philosophy accentuates readability of the code. As many high-level programming languages, Python is dynamically typed. Further, it supports multiple programming paradigms such as procedural, object-oriented and functional programming. I pursued many classes based on Python whether in high school or at university. I have a proficient background in programming in Python to work on this project.

**Software development.** Software development is the process of designing and implementation of applications and frameworks. In general, the process of software development is writing and maintaining the source code which is often a planned and structured process. The software

development contains mostly research, prototyping, modification and reuse of existing software. During the technical deliverable section, we use this structured process to design and implement our ANNs.

# 4. A Scientific Deliverable

## 4.1. Requirements

- **FR01** Present the feature extraction with MFCC
- **FR02** Present the notions of deep learning and Artificial Neural Networks (ANNs)
  This should present an introduction to the domain of deep learning. It presents an overview of 3 different ANNs architectures. Every Neural Network presentation will cover a small introduction and theoretical aspect.
- **NFR01** Performance evaluation and comparison
  During this section, we compare and analyse the performance obtained from the three Neural Networks architecture.

## 4.2. Design

### 4.2.1. FR01: Feature extraction with MFCCs.

Before we can train our end-to-end ASR model, we have to process our audio data. The first task is to extract features from the data which describes natural language expressions and excludes background noises and emotions.

Mel Frequency Cepstral Coefficients (MFCCs) are a feature generally used in ASR. MFCCs were presented by Davis and Mermelstein in the 1980s. [8]

To extract the MFCCS from the dataset, the following extraction steps have to be executed on a speech signal sampled at $16kHz$:

1. Frame the signal into $25ms$ frames. The frame length of a $16kHz$ signal:
   $$0.025 * 16000 = 400samples. \tag{1}$$
   With a frame step of $160$ $samples$ or $10ms$, the frames overlap themselves. Such that the first frame containing $400$ $samples$ starts at sample 0 and the second frame starts at sample 160. This pattern repeats until the end of speech signal. If the audio file is not divisible by an even number, it is padded by zeros until it is.

2. For each frame calculate the periodogram estimate of the power spectrum. To obtain the Discrete Fourier Transform (DTF) from the frame $i$, we perform:
   $$S_i(k) = \sum_{N}^{n=1} s_i(n)h(n)e^{-j2\pi kn/N} \tag{2}$$
   For each speech frame $s_i(n)$, the periodogram-based power spectral estimate is calculated with:
   $$P_i(k) = \frac{1}{N} \mid S_i(k) \mid^2 \tag{3}$$
   This is called the Periodogram estimate of the power spectrum which identifies for every frame which frequencies are present. The first 257 coefficients are kept from the 512 points fast Fourier transform which is an algorithm to compute the DTF.

3. Apply the Mel filterbank to the power spectral and sum the energy in each filter. The Mel filterbank is a set of 26 triangular filters. The filterbank energies are calculated by multiplying every filterbank with the power spectrum and adding the coefficients kept in the previous step.

4. Take the logarithm of the 26 filterbank energies.

5. Take the Discrete Cosine Transform (DCT) of the 26 log filterbank energies resulting in 26 cepstral coefficients. We only used the lower 12-13 of the 26 coefficients for ASR.

In section 4.3.1 we will use the Python library *Librosa* to show how we extracted the MFCCs feature from the dataset.

### 4.2.2. FR02: Deep Learning and Artificial Neural Networks.

The notions of Deep Learning and examples of ANNs presented in this section are based on the textbook *Deep Learning* [1] written by *Ian Goodfellow, Yoshua Bengio and Aaron Courville*. These presentations should only give a high-level introduction to these notions.

### 4.2.3. Deep Learning.

Artificial Intelligence (AI) is a field with many practical applications such as understanding speech, etc. AI deals with challenging problems which are easy to perform but hard to describe formally by humans. Deep Learning is a solution to these intuitive problems. This approach to AI allows computer systems to study from experience and understand the world through a hierarchal stack of concepts. The computing system gathers knowledge from experience which eliminates the need to describe formal rules. The computer understands complex concepts with this stack of concepts by decomposing them with simpler ones. The representation of this hierarchy of concepts shows a deep graph with many layers, hence the name for Deep Learning. [1]

### 4.2.4. Feedforward Neural Network.

Feedforward neural networks or multilayer perceptrons (MLPs) are very important deep learning models. The objective of MLPs is to approximate a given function $f^*$.

A feedforward network defines a mapping $\hat{\boldsymbol{y}} = f(\boldsymbol{x};\boldsymbol{\theta})$ and learns the values of the parameter $\boldsymbol{\theta}$ with a given input vector $\boldsymbol{x}$ which yields the best approximation of $f^*$. This function can be for example a classifier, $y = f^*(\boldsymbol{x})$ which maps an input $\boldsymbol{x}$ to a class $y$.

These structures are called **feedforward** since there is a progress of information. It is first evaluated as input $\boldsymbol{x}$ through the function $\boldsymbol{f}$, then goes through the computations which define $\boldsymbol{f}$ and ends up at the output $\boldsymbol{y}$.

MLPs are called **networks** because they are made up of composed functions. Let $f_1$, $f_2$ and $f_3$ be three function forming a chain $f(\boldsymbol{x}) = f_3(f_2(f_1(\boldsymbol{x})))$. The chain structures are commonly used in ANNs. In the example, $f_1$ is called the first layer of the network, $f_2$ is called the second layer, etc. The lenght of the chain represents the depth of the network. The final layer is also called the output layer.

During the training of the neural network, we try to find $f(\boldsymbol{x})$ to approximate $f^*(\boldsymbol{x})$. The training data contains examples of $f^*(\boldsymbol{x})$ obtained with different training points where every example $\boldsymbol{x}$ is referring to a label $y \approx f^*(\boldsymbol{x})$. These examples determine how the output layer should approximate a value close to $y$ with each example $\boldsymbol{x}$. The other layers are called the **hidden layers** because the training examples are not determining their behaviour and it doesn't contain the desired output of each layer. Therefore, a learning algorithm has to determine how to utilize the hidden layers to obtain a good approximation of $f^*(\boldsymbol{x})$.

The **width** of the feedforward model is defined by the dimensionality of each hidden layer which is usually a vector containing values. These vector layers can be thought of containing many **units** which process in parallel. Each unit refers to a vector-to-scalar function, by taking input from other units and calculating its proper activation value.

We will present a simple MLPs with one hidden layer which contains hidden units. The hidden layer represents a vector of hidden units $\boldsymbol{h}$ which are calculated by a function $f_1(\boldsymbol{x};\boldsymbol{\theta})$. $\boldsymbol{\theta}$ consists of $\boldsymbol{W}$ and $\boldsymbol{B}$ which are the $weights$ and $biases$ repectively such that:

$$\begin{aligned} h &= f_1(\boldsymbol{x};\boldsymbol{W},\boldsymbol{B}) \\ \Leftrightarrow h &= \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{B} \end{aligned} \tag{4}$$

$\boldsymbol{h}$ is then used as input for the second layer which is the output layer; $f_2(\boldsymbol{h};\boldsymbol{w},\boldsymbol{b})$.

$$\begin{aligned} y &= f_2(\boldsymbol{h};\boldsymbol{w},\boldsymbol{b}) \\ \Leftrightarrow y &= \boldsymbol{w}^T\boldsymbol{h} + \boldsymbol{b} \end{aligned} \tag{5}$$

The network can be illustrated as a chain of two functions:

$$f(\boldsymbol{x};\boldsymbol{W},\boldsymbol{B},\boldsymbol{w},\boldsymbol{b}) = f_2(f_1(\boldsymbol{x})) \tag{6}$$

After using an affine transformation, the hidden units are computed by a nonlinear function called an activation function:

$$h = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{B}) \tag{7}$$

The default activation function is the **rectified linear unit** or ReLU defined as:

$$g(z) = \max\{0, z\} \tag{8}$$

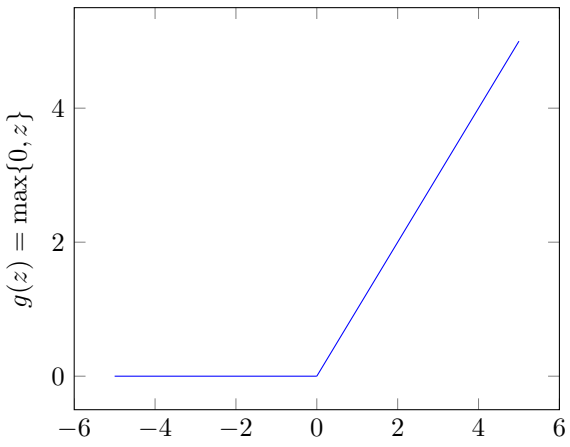


Figure 1. The rectified linear activation function.

The complete feedforward network looks as follow:

$$f(\boldsymbol{x};\boldsymbol{W},\boldsymbol{B},\boldsymbol{w},\boldsymbol{b}) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{B}\} + \boldsymbol{b} \tag{9}$$

A representation of a single perceptron and its position in an MLP is given in figure 4.2.4.

**Gradient-Based Learning.** During the training of the model, deep learning algorithms solve an optimization problem. The Problem is to minimize some function $f(x)$ by changing its input $x$. This function is referred to as the objective function also called the loss function. With the loss function the deep learning model optimizes the approximation of $f^*$. An example of a loss function for our MLP can be the mean squared error (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(f^*(\boldsymbol{x}_i) - f(\boldsymbol{x}_i;\boldsymbol{\theta}))^2 \tag{10}$$

ANNs usually used gradient-based optimizers to obtain a minimal loss function.

**Back-propagation.** We call forward propagation when an MLP takes $\boldsymbol{x}$ as input and computes $\hat{\boldsymbol{y}}$ as output since the information passes through the network. The back-propagation algorithm passes information from the loss function backwards through the network. This algorithm computes the gradient while passing back the network. With this gradient, another algorithm called the *stochastic gradient descent* performs the learning of the $weight = \{W, w\}$

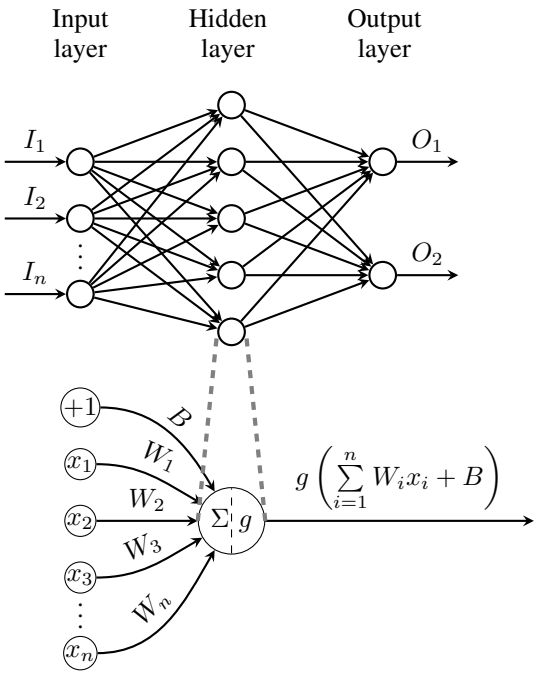


Figure 2. A figure of a single perceptron with its position in a large-scale MLP.

and the $biases = \{B, b\}$ and adjusts them to minimize the lost function.

#### 4.2.5. Recurrent Neural Network.

Recurrent Neural Networks (RNNs) are a particular architecture of ANNs which can process sequential data $S = x^{(1)}, \ldots, x^{(\tau)}$. To explain how RNNs are implemented, we have to introduce the notion of computational graphs. A computational graph is a formal structure representing a set of computations. We obtain a chain of events by unfolding a recurrent computation into a computational graph with a repetitive structure.

Consider a recurrent system,

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{11}$$

with $s$ being the state of the system. The graph can be unfolded considering the time step $\tau = 3$, we have,

$$\begin{aligned} s^{(3)} &= f(s^{(2)}; \theta) \\ &= f(f(s^{(1)}; \theta); \theta) \end{aligned} \tag{12}$$

This expression can be illustrated by a directed acyclic computational graph. See figure 3

Now we consider recurrent system accepting an external signal $\boldsymbol{x}^{(t)}$,

$$s^{(t)} = f(s^{(t-1)}, \boldsymbol{x}^{(t)}; \theta) \tag{13}$$

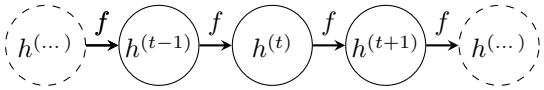


Figure 3. The recurrent system described by equation 12, illustrated as an unfolded computational graph.

where the state $s$ is containing information about the past sequence $S$. The hidden units can be describe with equation 13 using $\boldsymbol{h}$ to denote the state,

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \theta) \tag{14}$$

represented in figure 4.

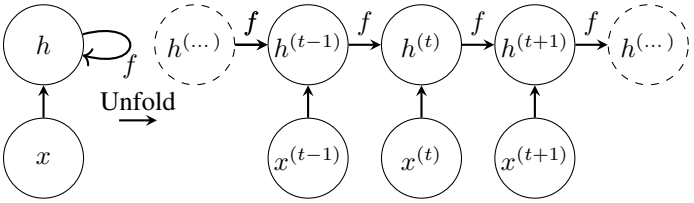


Figure 4. This is an RNN with no outputs, it processes the input $x$ and passes it into the state $h$ which is then passed through time. On the right graph, we have, the same network unfolded as a computational graph. [1]

With graph unfolding, we can now present some common example of an RNNs. Here are some important designs of RNNs:

- RNNs which produce an output at every time step $\tau$ and are connected recurrently between hidden units.
- RNNs which produce an output at every time step $\tau$ and are connected recurrently only from the output from one time step between hidden units at another time step.
- RNNs which are connected recurrently between hidden units, read an entire data sequence $S$ and produce only one single output.

We pick the RNN represented in figure 5 to develop a forward propagation equations. This RNN takes in a data sequence $S$ and outputs for every time step $\tau$ an output. In this figure, we didn't include an activation function for the hidden units $h$. However, for the equations, we assume the activation function $\mathcal{H}$ to be the hyperbolic tangent.

Let $h^{(0)}$ be the initial state. For every time step from $t = 1$ to $t = \tau$, we apply the following equations:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{15}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}) \tag{16}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(V)} \tag{17}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{o}^{(t)}) \tag{18}$$

where $\boldsymbol{b}$ and $\boldsymbol{c}$ are the bias vectors and $\boldsymbol{U}$, $\boldsymbol{V}$ and $\boldsymbol{W}$ are the weight vectors.

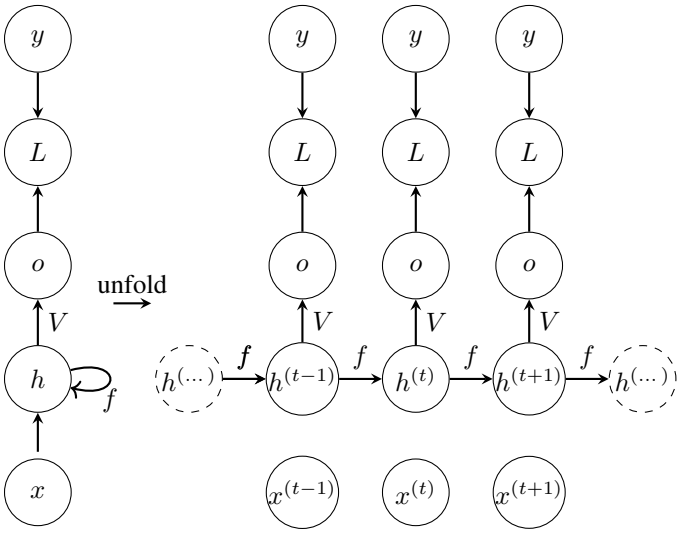


Figure 5. this computational graph is for calculating the training loss of an RNN which labels a sequence of $x$ values to corresponding output $o$ values. A loss function $L$ is calculating the distance from $o$ to the corresponding $y$ target. On the left, we have the network as a recurrent graph. On the right, we have its unfolded computational graph. [1]

**The Challenge of Long-Term Dependencies.**

The challenge of learning long-term dependencies is that gradients, propagated through a very deep RNN, can vanish or explode. Gradients can vanish when the gradients are converging to 0 and therefore the weights won't be updated while learning. Exploding gradients happen if the gradients are too large and the loss function will never reach the optimized minimum. There is a solution to this problem. We will introduce Long Short-term Memory (LSTM) RNNs which are an RNN architecture dealing with this challenge. [9]

#### 4.2.6. Long short-term memory.

### 4.3. Production

Provide descriptions of the deliverables concrete production. It must present part of the deliverable (e.g. source code extracts, scientific work extracts) to illustrate and explain its actual production.

#### 4.3.1. Feature extraction with MFCC in Python.

### 4.4. Assessment (± 15% of section's words)

**4.4.1. NFR01: Accuracy comparison.** Provide any objective elements to assess that your deliverables do or do not satisfy the requirements described above.

## 5. A Technical Deliverable

### 5.1. Requirements (± 15% of section's words)

Functional Requirement (FR) and Non-Functional Requirement (NFR)

- **FR01** Implementation of the three classification models
  We use the Keras library to implement our models with three different Neural Network architecture
- **FR02** Collect a small dataset of Luxembourgish spoken words
  We collect a small dataset containing Luxembourgish audio samples containing the words $w \in \{'0', \ldots, '9', 'moien'\}$

### 5.2. Design (± 30% of section's words)

We explain how to use the Keras Library.

### 5.3. Production (± 40% of section's words)

**5.3.1. Implementation of classification models.** We present the implementation of our classificatin models

**5.3.2. FR02: Luxembourgish dataset collection.** We present how we collect our Luxembourgish dataset.

### 5.4. Assessment (± 15% of section's words)

## Acknowledgment

I would like to thank my tutor Vladimir Despotovic for his constructive feedback and mentorship. His introduction and explanation of neural networks were outstanding. I would recommend fellow BiCS Students interested in this field to work with Vladimir Despotovic. Additionally, I thank him for supervising my paper.

## 6. Conclusion

## References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015. [Online]. Available: http://arxiv.org/abs/1512.02595

[3] W. S. J. Cai, "End-to-end deep neural network for automaticspeech recognition," 2015. [Online]. Available: https://cs224d.stanford.edu/reports/SongWilliam.pdf

[4] "Python software foundation, python language reference, version 3.7. available at," http://www.python.org/.

[5] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[6] P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” *CoRR*, vol. abs/1804.03209, 2018. [Online]. Available: http://arxiv.org/abs/1804.03209

[7] “General python faq,” https://docs.python.org/3/faq/general.html#what-is-python, accessed 19/05/19.

[8] “Mel frequency cepstral coefficient (mfcc) tutorial,” http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#references, accessed 02/12/19.

[9] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

## 7. Appendix

To be changed

```
import librosa
import numpy as np
import pandas as pd
import os
import csv

numbers = '0 1 2 3 4 5 6 7 8 9'.split()

header = ''
for i in range(1, 641):
    header += f'mfcc{i} '
header += ' label'
header = header.split()

file = open('largetrainingset.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)

for number in numbers:
    for filename in os.listdir(f'./recordings/{number}'):
        nfile = f'./recordings/{number}/{filename}'

        y, sr = librosa.load(nfile, sr=None, mono=True, duration=1)

        if y.size < 16000:
            rest = 16000 - y.size
            left = rest // 2
            right = rest - left

            y = np.pad(y, (left, right), 'reflect')

        mfccs = librosa.feature.mfcc(y=y, sr=sr)

        row = mfccs.T.flatten()
        row = np.append(row,number)

        file = open('largetrainingset.csv', 'a', newline='')
        with file:
            writer = csv.writer(file)
            writer.writerow(row)
```