KU LEUVEN
FACULTEIT
INGENIEURSWETENSCHAPPEN

Building a state of the art speech recogniser

Moritz Wolter

Thesis submitted for the degree of
Master of Science in Mathematical
Engineering

Thesis supervisors:
Prof. dr. ir. Patrick Wambacq
Prof. dr. ir. Hugo van Hamme

Assessor:
Prof. dr. ir. Johan Suykens

Mentor:
Ir. Vincent Renkens

Academic year 2016 – 2017

Preface

I would like to thank everybody who kept me busy the last year, especially my promotor and my assistants. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my family for supporting me trough my studies.

Moritz Wolter

Contents

Abstract

Speech recognition is concerned with transcribing what is said in a recoding of spoken language. In machine learning terms this process is called sequence labeling. A recoding consists of a chain of frames, this chain can be split up into several sequences, these make up words or phonemes, which must be labeled. The sequence of labels forms the transcription.

The meaning of speech depends on context, therefore a good system needs to take it into account. Classical feed-forward networks fail to do that, which is why this system will mainly consist of recurrently connected Long Short Term Memory (LSTM) blocks. Inspired by the recurrent connects of neurons in the human brain, LSTM-RNNs have the ability to store information over long time periods, an important requirement in take context into account.

In order to train machine learning systems, speech and transcription text pairs are used. The text contains the exact information of what is said in the recording, but where in the recoding which word or sound is said is unknown. In other words text to speech alignment is missing. Aligning the data will be an important issue in this thesis.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

ConvNet	Convolutional neural network
MSE	Mean Square error
PSNR	Peak Signal-to-Noise ratio

Symbols

42	“The Answer to the Ultimate Question of Life, the Universe, and Everything” according to [?]
c	Speed of light
E	Energy
m	Mass
π	The number pi

Chapter 1

Literature Study

1.1 Preprocessing and feature extraction

Filter-Bank features

Filter-banks are collections of filters. These filters can be spread out over audible frequencies[1]. Filter-bank output is commonly used as input for speech analysis [17][6]. The number of filter-banks depends on the required resolution, 32 is a common choice [18]. The energy within the part of the signal spectrum described by all individual filters is measured. Figure 1.1 shows the resulting energy measurements using 23 filters, for a sentence recording contained in the *TIMIT* data set. The general argument for filter banks in speech recognition is that the cochlea, in the human ear, resembles a filter bank [17, page 30]. Humans do not perceive frequency linearly. Experimental evidence suggests, that our perception of is scaled according to the Mel-Scale [17, page 34]:

$$B(f) = 1125ln(1 + f/700) \tag{1.1}$$

A normalized plot of this function is shown in figure 1.2 on the left. According to the Mel-scale, humans are able to distinguish more lower frequencies than higher frequencies. In the plot the first four thousand Herz occupy roughly eighty percent of the scale. The band from four thousand to eight thousand Herz is left with only about twenty percent of the scale, even tough half of the considered frequencies are in this band. Mel spaced filter-banks are an attempt to include the human perception

[1] Approximately 16 to 16000 Hz.

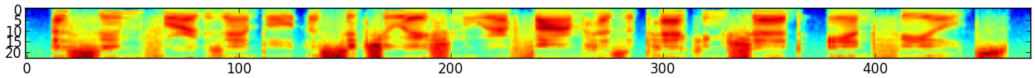

Figure 1.1: Frequency Bank input computed from a sentence contained in the *TIMIT* dataset. Time is shown on x and Frequency on the y-Axis.

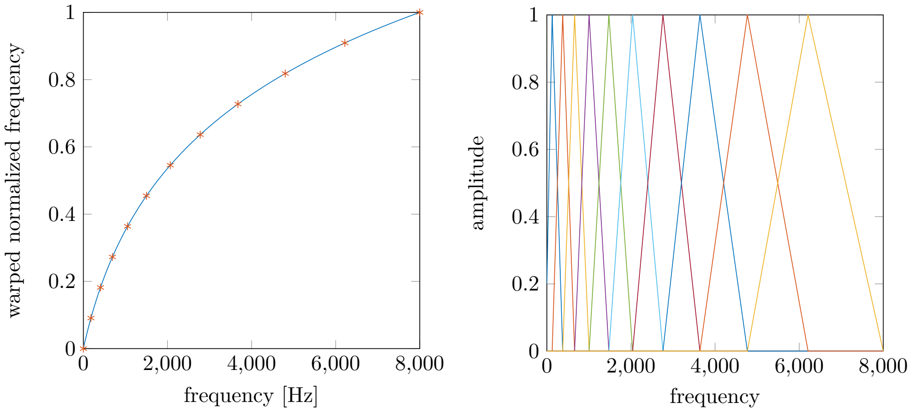

Figure 1.2: The Mel-scale (blue) with Mel-Frequency Cepstrum Coefficients (red) on the left. Filterbank with Mel-spaced filters (right).

in speech recognition. The filter functions are defined by [17, page 317]:

$$H_m = 0 \qquad \text{if } k < f[m-1] \tag{1.2}$$

$$H_m = \frac{k - f[m-1]}{f[m] - f[m-1]} \qquad \text{if } f[m-1] \leq k \leq f[m] \tag{1.3}$$

$$H_m = \frac{f[m+1] - k}{f[m+1] - f[m]} \qquad \text{if } f[m] \leq k \leq f[m+1] \tag{1.4}$$

$$H_m = 0 \qquad \text{if } k > f[m+1] \tag{1.5}$$

In the equations above H_m denotes the magnitude of filter m with a total of M filters. The frequency is denoted by k, the vector f contains $M + 2$ linearly spaced filter border values. These are the red stars on the left of figure 1.2. The right plot shows the triangular filter banks. These banks are spaced according to the same values. Roughly speaking using mel-filter banks means using a high filter resolution where human hearing is good and a low resolution where it is bad.

In addition to perceptually motivated instantaneous features, derivative information can be appended to each feature vector. These so called delta features add information regarding the speed of change taking into account $2 * N$ feature vectors. N denotes the number of frames back and forward in time. The derivative information can be computed using [22]:

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-1})}{2 \sum_{n=1}^{N} n^2} \tag{1.6}$$

The formula above is a central difference, because the value at time t is not taken into account, in contrast to forward or backward differences, where the value at t is part of the difference. A common choice for N is two. In this case the formula above simplifies to:

$$d_t = \frac{2c_{t+2} + c_{t+1} - c_{t-1} - 2c_{t-2}}{10} \tag{1.7}$$

In this case two future frames are used to compute the derivative at time t, which implies, that at computation time a full recording is available. Should that not be the case a backward difference scheme should be considered instead. To further augment the features second derivative or acceleration information can be added as well. These can be computed by simply applying the central difference to the deltas one more time.

Before using the features it is best practice to standardize the components of the input vectors. It is desirable to use features with an overall mean of zero and a standard deviation of 1 over the entire training set [10, page 30]. The feature normalization process starts by computing the mean,

$$m_i = \frac{1}{\|S\|} \sum_{x \in S} x_i \tag{1.8}$$

and standard deviation

$$\sigma_i = \sqrt{\frac{1}{|S| - 1} \sum_{x \in S} (m_i - x_i)^2} \tag{1.9}$$

Standardized input vectors can then be computed using:

$$\hat{x}_i = \frac{x_i - m_i}{\sigma_i} \tag{1.10}$$

Input standardization is beneficial to network performance, because it puts the input values in rages, which standard activation functions such as sigmoid or hyperbolic tangent can handle better [10, page 30].

Mel-Frequency banks are considered high level feature inputs. When the recognition system is found to work with these, features on a lower level or even raw data could be used as input. The idea behind doing fewer preprocessing, is that the network might be able to come up with something better on its own.

1.2 Neural Networks

Artificial neural networks where originally designed to model biological neurons. Even tough it is now known that artificial neural networks have little in common with their counterparts in biology, the models are still popular due to their useful pattern recognition properties [**?**, page 13]. All neural networks consist of elementary functions with weighted and biased inputs, the operations necessary to evaluate them form a computational graph. In a nutshell the idea is to choose the weights and biases such that the network output corresponds to patterns seen in the input data. The process of finding good weights is called training. Training in turn is done using gradient descent.

1.2.1 Gradient descent

The optimization process of neural networks works with a training data set $\{\{\mathbf{x}_1, \mathbf{t}_1\}, \dots, \{\mathbf{x}_p, \mathbf{t}_p\}\}$ [21, page 156]. The elements of this set are the input- and output-patters $\mathbf{x}$ and $\mathbf{t}$

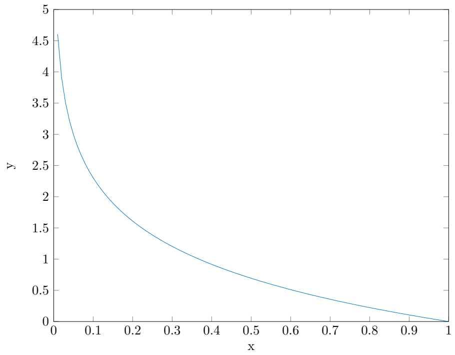

Figure 1.3: Plot of $y = -\ln(x)$ for $x \in (0, 1)$.

respectively. Ideally the network output should be the same as the desired one for all data pairs. The difference between target and current outputs could be measured by the cost function [21, page 156]:

$$E = \frac{1}{2} \sum_{i=1}^{p} \|\mathbf{o}_i - \mathbf{t}_i\|^2. \tag{1.11}$$

However generally the more complex cross-entropy is used as a cost function. Cross entropy is defined as [5, page 245]:

$$E = -\sum_{i=1}^{p} \mathbf{t}_i \ln(\mathbf{o}_i) + (1 - \mathbf{t}_i) \ln(1 - \mathbf{o}_i) \tag{1.12}$$

In order to understand why the cross entropy is a measure of output error, the two scalar cases $t = 0$ and $t = 1$ are considered. For $t = 0$ the cross entropy simplifies to $-\ln(1 - o)$. Looking at figure 1.3, this expression turns out to be a good cost function, because for values close to the desired output $o = 0$, $-\ln(1 - o)$ will be close to zero. If the output moves away from the desired value zero, the cost grows asymptotically. If $t = 1$ is considered the cost function simplifies to $-\ln(o)$ which will be zero for the desired output and once more displays asymptotic growth for undesired values far away from one. If the target probability is somewhere between these two extreme cases the cost will be the sum of the two cases considered earlier. Cross entropy is statistically motivated, the output vectors $\mathbf{o}$ describe probability distributions over the possible labels, their elements must therefore be $\in (0, 1)$ and sum op to one.

During the training process a local minimum of the error function E is sought. At this minimum the difference between the network output $\mathbf{o}$ and the target values $\mathbf{t}$ is small. This value might not be the smallest possible value, termination can happen at the global or a local minimum. After the training process has completed the network is expected to identify similarities to data seen during the training process and produce a similar output. In order to reach a local minimum, the gradient of the error function is needed. The key idea of gradient descent is to follow the negative gradient until a local minimum is reached. Neural networks can be considered as large composite functions, which are made up of elementary operations. The evaluation of the network can be written as a graph. Computations are done at each node and information travels trough the network along directed edges from node to node. In order to create a computational graph for the training process each of the output units of the network under consideration are connected to a new node which computes $\frac{1}{2}(o_{ij} - t_{ij})^2$[21, page 157] or the cross-entropy term or one data target pair. These new nodes in turn are connected to one more node, which sums up all error values and produces E_i. The process described above must be repeated for all training data pairs. One final node is added, which sums up all values E_i. Its output gives the value for the error function E which is now in the form of a large graph.

Reverse mode algorithmic differentiation or back-propagation is an algorithm to compute the gradient of a graph consisting of basic elementary operations. As an example its operation is now illustrated using the function [8, page 69]:

$$f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_1 x_2 x_3) \tag{1.13}$$

This function written in terms of five elementary operations as [8, page 70]:

$$\begin{align} x_4 &= x_1 x_2 \tag{1.14}\\ x_5 &= sin(x_4) \tag{1.15}\\ x_6 &= x_4 x_3 \tag{1.16}\\ x_7 &= exp(x_6) \tag{1.17}\\ x_8 &= x_5 + x_7 \tag{1.18}\\ y &= x_8 \tag{1.19} \end{align}$$

Computing the gradient means computing the partial derivatives of f with respect to all inputs. In this case this means finding:

$$\begin{align} \frac{\partial f(x_1, x_2, x_3)}{\partial x_1} &= x_2(\cos(x_1 x_2) + x_3 \exp(x_1 x_2 x_3)) \tag{1.20}\\ \frac{\partial f(x_1, x_2, x_3)}{\partial x_2} &= x_1(\cos(x_1 x_2) + x_3 \exp(x_1 x_2 x_3)) \tag{1.21}\\ \frac{\partial f(x_1, x_2, x_3)}{\partial x_3} &= x_1 x_2 \exp(x_1 x_2 x_3) \tag{1.22} \end{align}$$

Above the derivatives have been found by hand using the chain rule. Now these will be computed using back-propagation. Figure 1.4 shows a graphical representation of equation 1.13. The partial derivatives needed for the backwards sweep can be found on the edges.

The gradient is computed using a forward and backward sweep. During the forward sweep the inputs are fed into the network and the functions at each node are evaluated layer by layer, until the output at the last node is known. In figure 1.4 this means computing x_4 to x_8.

After the forward sweep the gradient is found by going back trough the network from

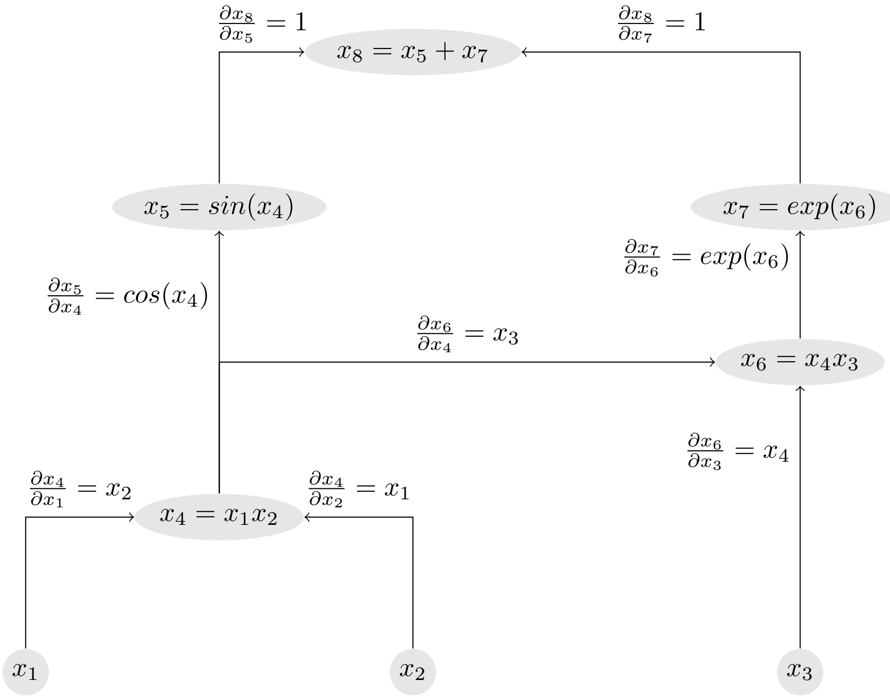
$\frac{\partial x_8}{\partial x_5} = 1$
$x_8 = x_5 + x_7$
$\frac{\partial x_8}{\partial x_7} = 1$
$x_5 = sin(x_4)$
$x_7 = exp(x_6)$
$\frac{\partial x_7}{\partial x_6} = exp(x_6)$
$\frac{\partial x_5}{\partial x_4} = cos(x_4)$
$\frac{\partial x_6}{\partial x_4} = x_3$
$x_6 = x_4x_3$
$\frac{\partial x_6}{\partial x_3} = x_4$
$\frac{\partial x_4}{\partial x_1} = x_2$
$x_4 = x_1x_2$
$\frac{\partial x_4}{\partial x_2} = x_1$
x_1
x_2
x_3

Figure 1.4: Example funciton network with partial derivatives.

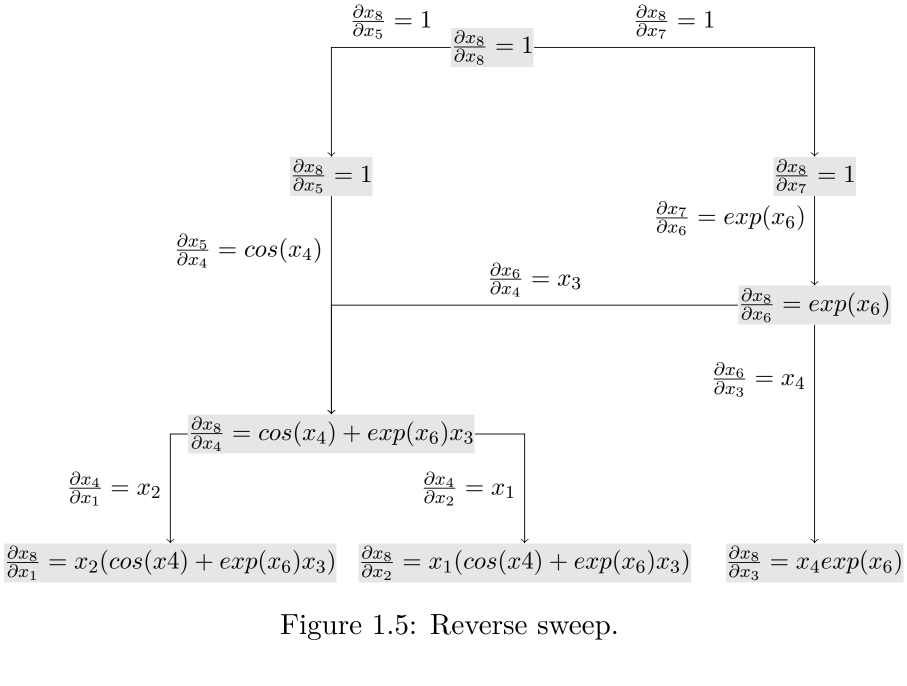
$\frac{\partial x_8}{\partial x_5} = 1$
$\frac{\partial x_8}{\partial x_8} = 1$
$\frac{\partial x_8}{\partial x_7} = 1$
$\frac{\partial x_8}{\partial x_5} = 1$
$\frac{\partial x_8}{\partial x_7} = 1$
$\frac{\partial x_7}{\partial x_6} = exp(x_6)$
$\frac{\partial x_5}{\partial x_4} = cos(x_4)$
$\frac{\partial x_6}{\partial x_4} = x_3$
$\frac{\partial x_8}{\partial x_6} = exp(x_6)$
$\frac{\partial x_6}{\partial x_3} = x_4$
$\frac{\partial x_8}{\partial x_4} = cos(x_4) + exp(x_6)x_3$
$\frac{\partial x_4}{\partial x_1} = x_2$
$\frac{\partial x_4}{\partial x_2} = x_1$
$\frac{\partial x_8}{\partial x_1} = x_2(cos(x4) + exp(x_6)x_3)$
$\frac{\partial x_8}{\partial x_2} = x_1(cos(x4) + exp(x_6)x_3)$
$\frac{\partial x_8}{\partial x_3} = x_4exp(x_6)$

Figure 1.5: Reverse sweep.

the output to each input node. Using a seed value of 1 at the output node the lower unit values are computed by multiplying the associated partial derivative found on each edge. If a node has more then one incoming value, their sum is computed. The process is illustrated in figure 1.5. At the roots of the tree the partial derivatives of the output with respect to each input can be found. Together these root values make up the gradient. To be able to perform the first forward sweep the network weights are initialized at random. The training data pairs are known and can be added as constants to the graph. If the weights of the network are stored in a weight vector $\mathbf{x}$ the value of the gradient after the nth update may be written as [10, page 27]:

$$\triangle\mathbf{x}(n) = -\alpha\frac{\partial f}{\partial\mathbf{x}(n)}. \tag{1.23}$$

Where $\alpha \in [0, 1]$ denotes the learning rate. Unfortunately simple gradient descent tends to get stuck in local optima. In order to increase the chance of gradient descent to escape from such a local minimum a momentum term can be added to the formulation [5, page 267][10, page 27]:

$$\triangle\mathbf{x}(n) = m\triangle\mathbf{x}(n-1) - \alpha\frac{\partial f}{\partial\mathbf{x}(n)}. \tag{1.24}$$

Above $m \in [0, 1]$ denotes the momentum. In order to understand the effect of the momentum term consider a weight space region of very low curvature, where it can be assumed, that the gradient stays constant. Using the momentum formula above yields [5, page 267]:

$$\begin{align} \triangle\mathbf{x}(n) &= -\alpha\frac{\partial f}{\partial\mathbf{x}}(1 + m + m^2 + \dots) \tag{1.25}\\ &= -\frac{\alpha}{1-m}\frac{\partial f}{\partial\mathbf{x}}. \tag{1.26} \end{align}$$

Using the fact that the sum of the geometric series is known and $|m| < 1$ must hold. The momentum term therefore changes the effective learning rate from α to $\frac{\alpha}{1-m}$ in regions of low curvature. Elsewhere the gradient is oscillatory, which leads to cancellation of successive contribution of the momentum term [5, page 267].

1.2.2 Deep Neural Networks

In machine learning increasing the network depth yields better results than the width. Additionally avoiding separately trained systems, in favor of end-to-end systems can lead to deep networks.

Stochastic gradient descent

When training deep networks on very large training sets, working with the full data set to compute the current gradient becomes very inefficient. As a remedy its is good practice in machine learning to work with so called mini-batches. A mini-batch includes a random subset of the training data set. This procedure is known as

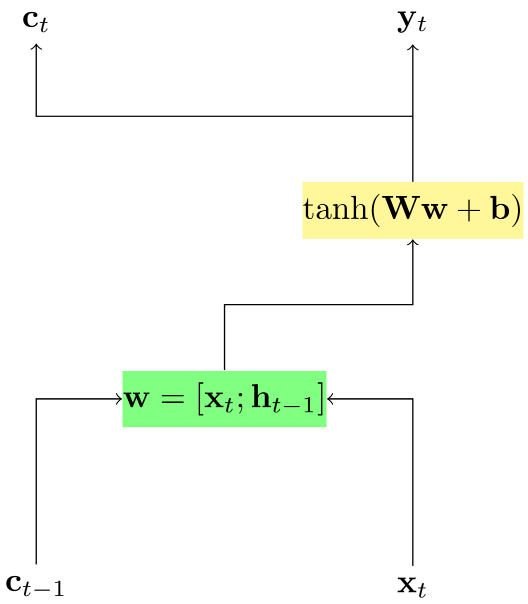
$\mathbf{c}_t$
$\mathbf{y}_t$
$\tanh(\mathbf{Ww}+\mathbf{b})$
$\mathbf{w}=[\mathbf{x}_t;\mathbf{h}_{t-1}]$
$\mathbf{c}_{t-1}$
$\mathbf{x}_t$

Figure 1.6: Visualization of a single recurrent cell.

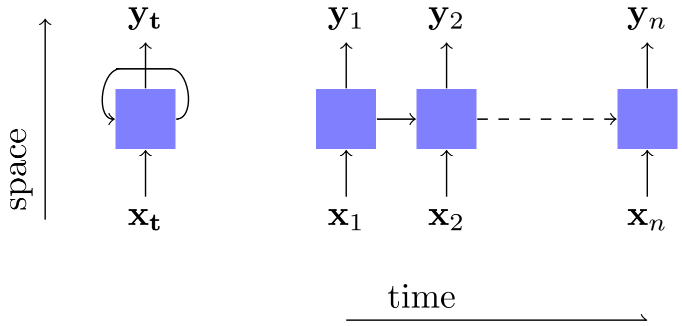

Figure 1.7: Rolled (left) and unrolled (right) recurrent neural net with two units.

randomized gradient descent. During training the stochastic process goes trough the data mini-batch by mini-batch, the gradient is accumulated and applied for the entire batch after all mini-batches have been processed.

1.2.3 Recurrent Neural Networks

When processing speech its is important to take context into account. When spelling the letters which make up a word, it is important to know what the previous letter was, in order to make the right decision. Feed-forward neural nets do not possess memory. These networks make decisions, starting from zero every time. In oder to fix this a cell state variable can be introduced. A simple recurrent cell where the current state is set to the previous output is shown in figure 1.6. Another way to depict the same architecture is shown in Figure 1.7, here the cell state is not labeled explicitly, the single cell on the left is just a simplified version of the figure above.

If the network is unrolled in time the flow of the state becomes apparent, which is done in figure 1.7 on the right. The unrolled form shows a direct dependency of the output at time t on the previous output at $t-1$, which in turn depends on the previous outputs. This causes y_t and y_{t-1} to change together. In other words: The introduction of recurrent connections leads to correlation of the two outputs.

The exploding and vanishing gradient problem

Even tough past information is available in theory, RNNs have only limited access to contextual information in practice [10, page 1]. Due to problems with gradient descent on correlated data, the back-propagated derivative can sometimes become weaker and weaker until it ultimately vanishes [15]. Another problem is that sometimes classical recurrent neural nets produce a gradient that blows up [20]. The exploding gradients can be fixed by clipping, but vanishing gradients require more sophisticated treatment [4].

Long short-term memory

Initially researchers tried to solve the vanishing gradient problem by making changes to the training algorithm using simulated annealing, time delays or compression [10, page 32]. However a good solution to the problem turned out to be changing the RNN cell architecture. Long short-term memory (LSTM) cells as proposed in [16] are more complex network units. LSTMS are differentiable versions of memory chips in a digital computer. Memory chips generally have read, write, and erase ports which can be set in order to allow the chips state to be read, modified or emptied. In the LSTM case the input, output, and forget gates serve the same function [10, page 33]. Throughout the literature these gates are generally denoted as $\mathbf{i}, \mathbf{f}$ and $\mathbf{o}$. The content of the memory or state is written as $\mathbf{c}$, the output as $\mathbf{h}$, while a t subscript denotes the time step. These memory cells use the differentiable equation system [12, page 5][2]:

$$\begin{align}
\mathbf{i_t} &= \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h_{t-1}} + \mathbf{W}_{ic}\mathbf{c_{t-1}} + \mathbf{b}_i) \tag{1.27}\\
\mathbf{f_t} &= \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h_{t-1}} + \mathbf{W}_{fc}\mathbf{c_{t-1}} + \mathbf{b}_f) \tag{1.28}\\
\mathbf{c_t} &= \mathbf{f_t}\mathbf{c_{t-1}} + \mathbf{i_t}\tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \tag{1.29}\\
\mathbf{o_t} &= \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h_{t-1}} + \mathbf{W}_{oc}\mathbf{c_t} + \mathbf{b}_o) \tag{1.30}\\
\mathbf{h_t} &= \mathbf{o}_t\tanh(\mathbf{c}_t) \tag{1.31}\\
& \tag{1.32}
\end{align}$$

From the definition of the matrix product follows that

$$\mathbf{A}\mathbf{x}_1 + \mathbf{B}\mathbf{x}_2 = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2\end{bmatrix}. \tag{1.33}$$

[2] Various versions of LSTM cells exist. This one is commonly referred to as the “peephole” variant.

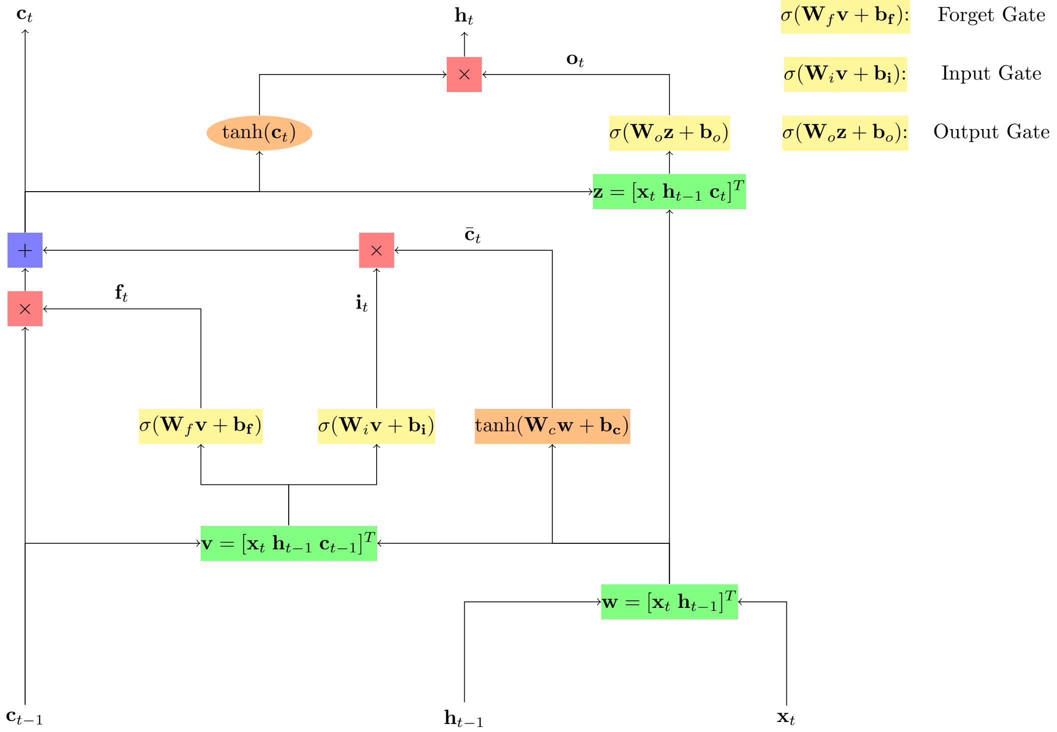
$\mathbf{c}_t$
$\mathbf{h}_t$
$\sigma(\mathbf{W}_f\mathbf{v} + \mathbf{b_f})$: Forget Gate
$\mathbf{o}_t$
$\times$
$\sigma(\mathbf{W}_i\mathbf{v} + \mathbf{b_i})$: Input Gate
$\tanh(\mathbf{c}_t)$
$\sigma(\mathbf{W}_o\mathbf{z} + \mathbf{b}_o)$
$\sigma(\mathbf{W}_o\mathbf{z} + \mathbf{b}_o)$: Output Gate
$\mathbf{z} = [\mathbf{x}_t\ \mathbf{h}_{t-1}\ \mathbf{c}_t]^T$
$\bar{\mathbf{c}}_t$
$+$
$\times$
$\times$
$\mathbf{f}_t$
$\mathbf{i}_t$
$\sigma(\mathbf{W}_f\mathbf{v} + \mathbf{b_f})$
$\sigma(\mathbf{W}_i\mathbf{v} + \mathbf{b_i})$
$\tanh(\mathbf{W}_c\mathbf{w} + \mathbf{b_c})$
$\mathbf{v} = [\mathbf{x}_t\ \mathbf{h}_{t-1}\ \mathbf{c}_{t-1}]^T$
$\mathbf{w} = [\mathbf{x}_t\ \mathbf{h}_{t-1}]^T$
$\mathbf{c}_{t-1}$
$\mathbf{h}_{t-1}$
$\mathbf{x}_t$

Figure 1.8: Visualization of the LSTM architecture.

Which this relation in mind the equations above can be rewritten, by creating column wise concatenated weight matrices for every neuron gate W_i, W_f, W_o, as well as for the state W_c. These matrices can then be multiplied by a row wise concatenated vector $[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c}]^T$, which leads to the slightly simplified system of equations below:

$$\mathbf{i_t} = \sigma(\mathbf{W}_i[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c_{t-1}}]^T + \mathbf{b}_i) \tag{1.34}$$

$$\mathbf{f_t} = \sigma(\mathbf{W}_f[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c_{t-1}}]^T + \mathbf{b}_f) \tag{1.35}$$

$$\mathbf{c_t} = \mathbf{f}_t\mathbf{c_{t-1}} + \mathbf{i}_t \tanh(\mathbf{W}_c[\mathbf{x}_t\ \mathbf{h_{t-1}}]^T + \mathbf{b}_c) \tag{1.36}$$

$$\mathbf{o_t} = \sigma(\mathbf{W}_o[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c_t}]^T + \mathbf{b}_o) \tag{1.37}$$

$$\mathbf{h_t} = \mathbf{o_t} \tanh(\mathbf{c_t}) \tag{1.38}$$

This system of equations is visualized in figure 1.8. Just like diagram 1.6 this depiction is read from bottom to top. The most important part is the line from $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$ [7]. It records operations on the cell state $\mathbf{c_t}$. The cell state contains information from the past which helps the block make decisions regarding the current output $\mathbf{h}_t$. The sigmoid functions $\sigma(\cdot)$ are applied element wise on the input vectors and produce outputs between zero and one. In the case of the forget gate output $\mathbf{f}_t$ these values $\in (0, 1)$ well serve as a measure of how much of the past state the cell would like to remember. One means keep this variable and zero throw it away [7]. The following task is to determine what should be added to the memory. This information can be found in the input gate result $\mathbf{i}_t$. $\mathbf{i}_t$ is multiplied element wise with the candidate values $\bar{\mathbf{c}}_t$. These are computed by a hyperbolic tangent neuron. The $\tanh(\cdot)$ function makes sure all vector elements are between -1 and 1. The neuron computing the candidate state values $\bar{\mathbf{c}}_t$ looks at input data and the past outputs. Both are labeled $\mathbf{w}$ in figure 1.8, $\mathbf{w}$ contains all information that could possibly be included in the new state. Finally the weighted candidate values are added to what was previously stored. This operation leads to the updated memory state $\mathbf{c}_t$. Last but not least the new output value has to be computed, which will be a filtered version of the cell state. The decision of which and how much of each state variable will be send outside is made by the output gate. It's output $\mathbf{o}_t$ is multiplied with a rescaled version of the cell state. The rescaling is done using another hyperbolic tangent, which again sets all values between minus one and one. The product of this rescaled state and the weights found in $\mathbf{o}_t$ then yields the new output $\mathbf{h}_t$.

At this point it is interesting to note that the graphs considered in this thesis will almost exclusively consist out the exponential, logarithmic, sigmoid and hyperbolic tangent functions. These functions are intimately related. It is well known that the exponential and logarithmic functions are connected by:

$$\exp(\ln(x)) = x \quad \text{if } x > 0. \tag{1.39}$$

The sigmoid and hyperbolic tangent functions in turn consist out of exponentials, and are related trough [10, page 15]:

$$\tanh(x) = 2\sigma(2x) - 1. \tag{1.40}$$

The equation above can be proven by using the definitions of the sigmoidal $\sigma = \frac{1}{1+\exp(-x)}$, and hyperbolic tangent $\tanh = \frac{\exp(2x)-1}{\exp(2x)+1}$:

$$\begin{aligned}
\tanh(x) &= \frac{\exp(2x) - 1}{\exp(2x) + 1} && (1.41)\\
&= \frac{2\exp(2x) - (\exp(2x) + 1)}{\exp(2x) + 1} && (1.42)\\
&= \frac{2\exp(2x)}{\exp(2x) + 1} - \frac{\exp(2x) + 1}{\exp(2x + 1)} && (1.43)\\
&= \frac{2\exp(2x)}{\exp(2x) + 1} - 1 && (1.44)\\
&= \frac{2}{\frac{\exp(2x)+1}{\exp(2x)}} - 1 && (1.45)\\
&= \frac{2}{1 + \frac{1}{\exp(2x)}} - 1 && (1.46)\\
&= 2\frac{1}{1 + \exp(-2x)} - 1 = 2\sigma(2x) - 1 && (1.47)
\end{aligned}$$

These two non-linear functions are thus very similar, in the LSTM-cell their different function value ranges decide how they are used. Sigmoids determine how much of something should be used, stored, or deleted, because its output values range between zero and one, while computations done on the cell state use the hyperbolic tangent, due to its wider range of function values.

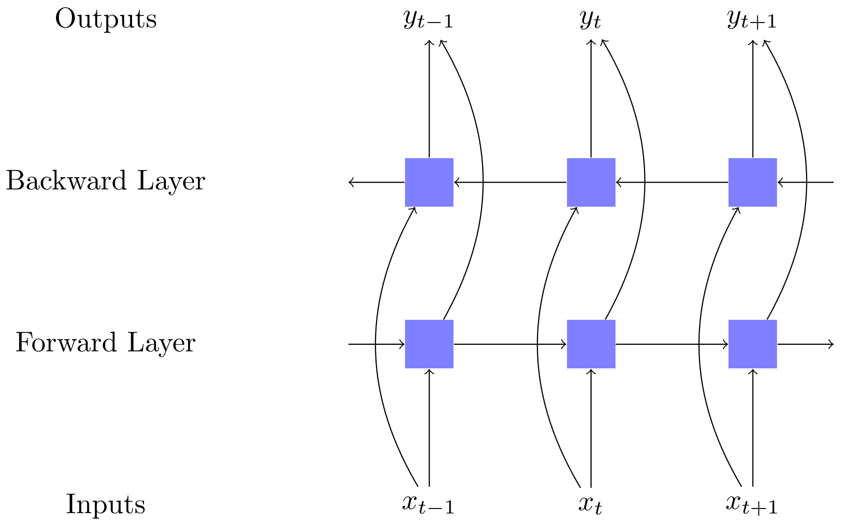

Figure 1.9: A bidirectional Long short term memory layer, according to [14]

Bidirectional Long Short Term Memory

With the advent of LSTMs deep recurrent networks became feasible in speech recognition [14]. RNNs are always deep in time, because their hidden state depends on past inputs. To enable abstraction their structure must also be deep in space. A bidirectional LSTM layer is shown in figure 1.9. It is important to note, that several LSTM layers are often followed by a single linear layer, as is the case in [14]:

$$\overrightarrow{\mathbf{h}}_t = \mathrm{LSTM}(\mathbf{W}_{\overrightarrow{\mathbf{h}}_t}[\mathbf{x}_t \; \mathbf{h}_{t-1}]^T + \mathbf{b}_{\overrightarrow{\mathbf{h}}_t}) \tag{1.48}$$

$$\overleftarrow{\mathbf{h}}_t = \mathrm{LSTM}(\mathbf{W}_{\overleftarrow{\mathbf{h}}_t}[\mathbf{x}_t \; \mathbf{h}_{t+1}]^T + \mathbf{b}_{\overleftarrow{\mathbf{h}}_t}) \tag{1.49}$$

$$\mathbf{y}_t = \mathbf{W}_y[\overrightarrow{\mathbf{h}}_t \; \overleftarrow{\mathbf{h}}_t]^T + \mathbf{b}_y \tag{1.50}$$

If stacked on top of each other, these bidirectional LSTM layers form a deep recurrent network. Defining $\mathbf{h}^0 = \mathbf{x}$, $\mathbf{h}^N = \mathbf{y}$ looking at time from $t = 1$ to T and taking N layers leads to:

$$\overrightarrow{\mathbf{h}}_t^n = \mathrm{LSTM}(\mathbf{W}^n_{\overrightarrow{\mathbf{h}}_t}[\mathbf{h}_t^{n-1} \; \mathbf{h}_{t-1}^n]^T + \mathbf{b}^n_{\overrightarrow{\mathbf{h}}_t}) \tag{1.51}$$

$$\overleftarrow{\mathbf{h}}_t^n = \mathrm{LSTM}(\mathbf{W}^n_{\overleftarrow{\mathbf{h}}_t}[\mathbf{h}_t^{n-1} \; \mathbf{h}_{t+1}^n]^T + \mathbf{b}^n_{\overleftarrow{\mathbf{h}}_t}) \tag{1.52}$$

$$\mathbf{h}_t^N = \mathbf{W}_y^N[\overrightarrow{\mathbf{h}}_t \; \overleftarrow{\mathbf{h}}_t]^T + \mathbf{b}_y^N \tag{1.53}$$

In this setting each LSTM cell has access to information from before or after it. For this to work the speech sequence, which is analyzed has to be recorded completely. In this case future information is available and should be used for recognition purposes.

1.3 Connectionist temporal classification

A bidirectional LSTM layer establishes access to past and future input information, but the alignment of the speech inputs and it's transcription remains unknown. Connectionist temporal classification is an output layer, which in tandem with the BLSTM layer enables the network to work around this problem, while at the same time allowing the gradient to propagate trough it. The resulting network can be trained end-to-end. This entire section is based on [10, chapter 7].

CTC consists of a softmax layer with one output per alphabet element plus one for blank our no label. The idea is to allow the network to output a label at any input time step, and post process these outputs to get the overall sequence of labels correct, while ignoring the timing of each label. More formally L denotes the used alphabet and $L' = L \cup \oslash$ the alphabet with the empty label. The network output vectors are denoted by $\mathbf{y}^t$ and will have length $|L'|$. Given the training set S and the input sequence $\mathbf{x}$ the probability for observing a sequence π consisting of elements from L' with length T is given by [10, page 56]:

$$p(\pi|\mathbf{x}, S) = \Pi_{t=1}^{T} y_{\pi_t}^{t}. \tag{1.54}$$

In a first step all a mapping $\mathcal{B}$ is introduced, which simplifies the allowed paths trough the label probabilities, by removing duplicates. For example $\mathcal{B}(x \oslash xy\oslash) = \mathcal{B}(\oslash xx \oslash xyy) = xxy$. In other words a new label is only produced if the output label changed in the path. The probability of a labeling l can be computed using [10, page 57]:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}). \tag{1.55}$$

Generally speaking this collapsing process is what makes it possible to use CTC with unaligned data.

CTC takes into account only the features and (depending on how the beam search is done) ignores the label it previously produced. Additionally the network output is coupled to the input frames. This shortcoming lead to the development of so called transducers. These models decouple the decoding time from the input time and use past labels to improve performance.

1.4 Listen, Attend and Spell [6]

The Listen Attend and Spell architecture is a deep neural network, designed to jointly learn to align and transcribe speech data. In contrast to CTC it belongs to the transducer family of speech recognition algorithms. All transducers have in common that they consist of an encoder and a decoder, with decoupled time steps. The idea also appeared in the automatic translation literature [2]. Working on the source language text, an annotator or encoder produces high level annotations. These annotations are then given to the translator or decoder, which using its own output time step works with the annotations to compute a context, which it then uses to produce a translation. In the speech application the encoder is referred to as the listener. The decoder as the speller. Combined these two form the las-network. The listener is a pyramidal recurrent neural net. It accepts filter bank spectra $\mathbf{x}_n$ as inputs and produces compressed high level output features $\mathbf{h}_m$. Compression reduces the computational load during feature processing later. The speller in turn accepts the features as input and outputs distributions over character sequences $\mathbf{y}_p$. Due to the uncoupling of the input and decoding time step, it requires a state, as well as an attention mechanism. The state provides a memory of what happened in the decoder in the past. The attention function determines, which listener-features are relevant at a given decoding time step. Combining attention and state makes it possible to label the input data. An overview of the las-achrcitecture is given in figure 1.10.

1.4.1 The Listener

The listener, consists of Long Short Term Memory blocks. These blocks are arranged in layers. The inputs are first fed into a recurrent bidirectional layer (BLSTM). This choice gives the system access to future data, therefore only fully recorded data can be analyzed. This system is restricted to applications, which do not require transcriptions in real time or where its acceptable to wait with decoding until a sentence has been recorded completely. When going up in figure 1.10, pyramidal layers (pBLSTM) follow the initial BLSTM layer. The pyramidal structure concatenates the hidden values computed previously, such that their time dimension is halved:

$$\mathbf{h}_t^n = \text{BLSTM}(\mathbf{h}_{t-1}^n, [\mathbf{h}_{2t}^{n-1}, \mathbf{h}_{2t+1}^{n-1}]) \tag{1.56}$$

Technically instead of two, three or more previously computed feature vectors could be concatenated, which increases the compression factor per pyramidal layer. This operation reduces the length U of the high level features $\mathbf{H}$. Without this compression the following attend and spell operation has a hard time extracting the relevant information, because a longer time span has to be considered to decode a single character. Additionally the compression reduces the problem complexity, which speeds up the training process significantly.

1.4.2 Attend and spell

The speller takes the features and produces a distribution over Latin character sequences as output. The computation of this output involves the context vector $\mathbf{c}_i$, the decoder state $\mathbf{s}_i$, the features $\mathbf{H}$ and the previous output $\mathbf{y}_i$. The index i denotes decoding time, $i-1$ is used to refer to results from the last time step. The last decoding step I, at which the system terminates is a learned quantity. While the last input step U, depends on the input features, lower case u denotes the input step.

During operation the Attend and spell functions keep track of previous output labels and previously important features, which are contained in the context. This information is stored in the state $\mathbf{s}_i$. To function the network must determine, which part of the computed features is $\mathbf{H}$ are relevant at any given decoding time step i. The context vector $\mathbf{c}_i$ contains a linear combination of relevant features, weighted according to their importance. The AttentionContext function determines these weights based on the state. Finally the speller function finds a probability over possible labels using the on the relevant features in the context and the state. The computing steps are therefore [6, page 4]:

$$s_i = \text{RNN}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_{i-1}) \tag{1.57}$$

$$\mathbf{c}_i = \text{AttentionContext}(\mathbf{s}_i, \mathbf{H}) \tag{1.58}$$

$$P(\mathbf{y}_i|\mathbf{x}, \mathbf{y}_{<i}) = \text{CharacterDistribution}(s_i, \mathbf{c}_i) \tag{1.59}$$

The state follows from a recurrent neural a multilayer LSTM. In contrast to the listener the speller is causal, meaning that it makes decisions only based on information computed during previous decoding steps. LSTMs are necessary here, because past states must be remembered. The attention mechanism, called AttentionContext above, computes a new context vector once every time step. This computation starts with the determination of the scalar energy $e_{i,u}$, which will be used as weight for its corresponding feature vector h_u. The computation starts with two feedforward neural networks or multilayer perceptrons (MLP), ϕ and ψ [6, page 5]:

$$e_{i,u} = \phi(\mathbf{s}_i)^T \psi(\mathbf{h_u}) \tag{1.60}$$

$$\alpha_{i,u} = \frac{\exp(e_{i,u})}{\sum\limits_u \exp(e_{i,u})} \tag{1.61}$$

$$\mathbf{c}_i = \sum_u \alpha_{i,u} \mathbf{h}_u \tag{1.62}$$

α is produced by running $\mathbf{e}$ trough a softmax function, which scales $\mathbf{e}$ such that all elements are within $(0, 1)$ and add up to one. These scaled weights, can then be used to form the context vector $\mathbf{c}_i$. When the training process converges the α_is typically follow a distribution with sharp edges[6, page 5]. Thus it is justified to think of the alphas as a sliding window. This window contains only the currently relevant parts of the condensed input data set.

1.4.3 Training

For end-to-end speech recognition all networks must be trained jointly. The objective is to maximize the logarithmic probability:

$$\max_\theta \sum_i \log P(y_i|\mathbf{x}, y_{<i}; \theta). \tag{1.63}$$

Here y_i denotes the current output distribution, x the input, θ the various network parameters and finally $y_{<i}$ the ground truth, which is the known true desired output. In practice the objective is minimized by working with a cross entropy loss function. Using the known output during training creates a situation, where the past outputs are always right. In practice however the situation will be different, as the network is going to make mistakes. As it is desired to create a robust model it is necessary to sometimes include the character distribution generated by the networks being

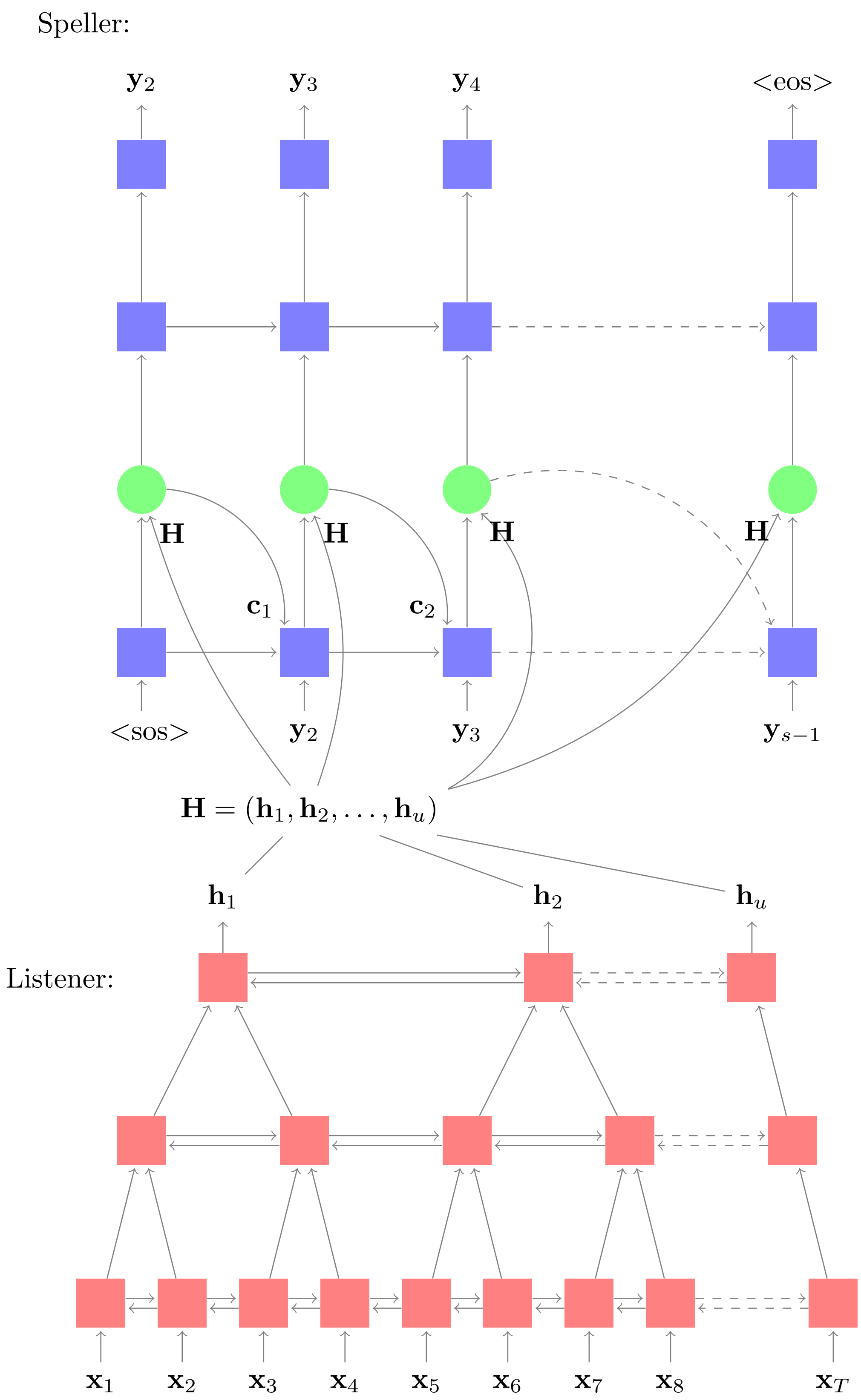

Figure 1.10: The LAS architecture [6, page 3]. BLSTM blocks are shown in red. LSTM blocks in blue and attention nets in green.

trained. Which leads to the objective [6, page 5]:

$$\hat{y}_i = \text{CharacterDistribution}(s_i, \mathbf{c}_i) \tag{1.64}$$

$$\max_{\theta} \sum_i \log P(y_i|\mathbf{x}, \hat{y}_{<i}; \theta) \tag{1.65}$$

The novelty in comparison to the previous expression is that $\hat{y}_{<i}$ is sometimes taken from the past network outputs instead of the ground truth. An idea which Chan et al. found in [3].

1.4.4 Decoding with beam search

In order to generate a readable text, it is necessary to choose characters from the generated character distributions. One way to do this is to simply pick the most likely letter from each distribution. This method ignores the possibility of generating better results by also considering less likely options. Therefore a broader search trough the most likely options is considered. Unfortunately memory limitations make it impossible to search trough all possible combinations. Therefore only the n most likely options are explored and the rest is disregarded. Adding the most likely options for each letter produces a tree of possible transcriptions. The different routes along this tree are called hypotheses. A score for each hypothesis can be computed, by multiplication of the probability values the las-network assigned to each branch along its path. In beam search only the m most likely hypotheses are kept. Using only las-probabilities would be equivalent to picking the most likely letter at every node. Therefore a language model is required to pick the best hypothesis. Such models can be trained on text data only. A selection can then be made according to[6, page 6]:

$$s(\mathbf{y}|\mathbf{x}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{|\mathbf{y}|_c} + \lambda \log P_{LM}(\mathbf{y}) \tag{1.66}$$

Here P_{LM} denotes the weight the language model assigns to each hypothesis. And λ is a weight factor, which determines the language model importance.

1.5 Tensor-flow

In this section is devoted to the toolbox, which will be used to implement the Listen Attend and spell, architecture. According to the Tensor-flow authors [1]: “TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms”. It was released by Google in 2015 and can be used from within Python or C++.

Chapter 2

Experiments

2.1 TIMIT

The *timit* speech corpus [9], contains recordings of ten phonetically rich sentences, for example:

> "She had your dark suit in greasy wash water all year. "

For each sentence a transcription of the spoken phonemes is also available. Phonemes are sets of sounds, which are considered equivalent in a given language. In alphabetic writing systems such as the latin one the phoneme to letter mappings should ideally be one. Due to the fact that the Latin script was devised for classical Latin as well as the fact that when pronunciation changes the spelling often remains the same, the phoneme to letter mappings are often far from one. Therefore the `timit` data set comes with phonetic transcriptions for all sentences. For the sentence considered above the spoken phonemes are:

> "h# sh ix hv eh dcl jh ih dcl d ah kcl k s ux q en gcl g r ix s ix w ao sh epi w ao dx axr ao l y ih axr h# "

The transcriptions contain a total of 64 possible phonetic labels, in the literature the full set and foldings with 48 and 39 labels are considered [19]. In all following experiments the 39 labels shown in 2.1 will be considered.

For all experiments the timit data set is split into a training, validation and test set. Containing 3696, 400 and 192 sentences [11, page 80].

2.2 BLSTM-CTC

This section explores bidirectional long short term memory layers with CTC output on the timit corpus [11, 13]. Training began after transforming of the speech data into Mel frequency cepstral coefficients (*MFCCC*s) and phoneme folding as described above. Two *bLSTM* layers where stacked on top of each other. The logits computed by the two layers where then fed into a CTC output layer, finally beam search decoding with a beam width of 100 was used for find the phoneme predictions. The

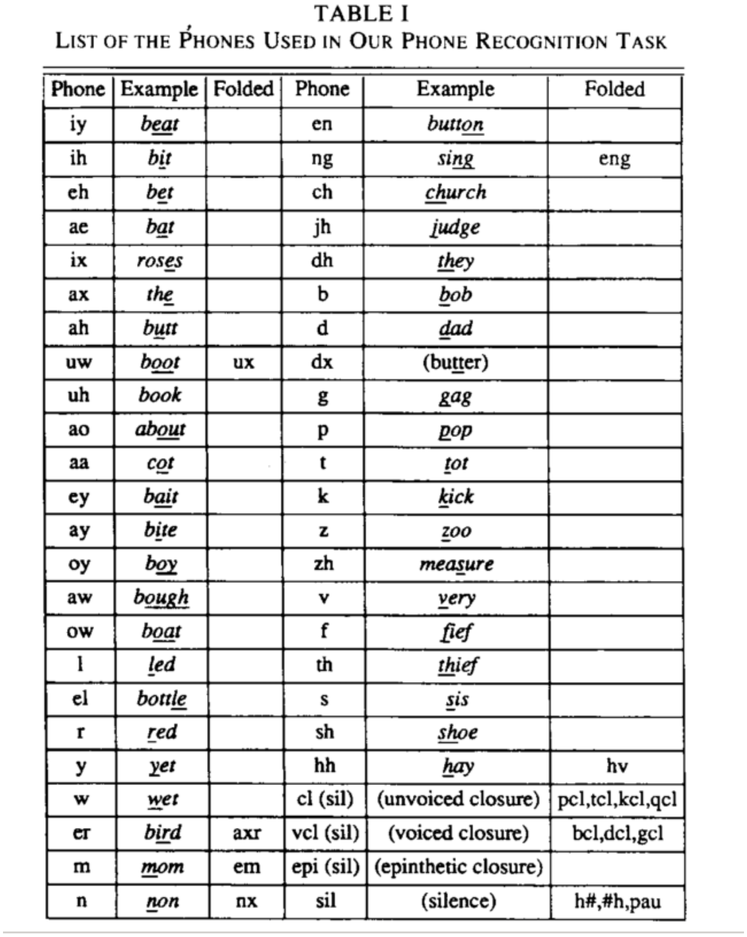

TABLE I
List of the Phones Used in Our Phone Recognition Task

Phone	Example	Folded	Phone	Example	Folded
iy	*beat*		en	*button*	
ih	*bit*		ng	*sing*	eng
eh	*bet*		ch	*church*	
ae	*bat*		jh	*judge*	
ix	*roses*		dh	*they*	
ax	*the*		b	*bob*	
ah	*butt*		d	*dad*	
uw	*boot*	ux	dx	(butter)	
uh	*book*		g	*gag*	
ao	*about*		p	*pop*	
aa	*cot*		t	*tot*	
ey	*bait*		k	*kick*	
ay	*bite*		z	*zoo*	
oy	*boy*		zh	*measure*	
aw	*bough*		v	*very*	
ow	*boat*		f	*fief*	
l	*led*		th	*thief*	
el	*bottle*		s	*sis*	
r	*red*		sh	*shoe*	
y	*yet*		hh	*hay*	hv
w	*wet*		cl (sil)	(unvoiced closure)	pcl,tcl,kcl,qcl
er	*bird*	axr	vcl (sil)	(voiced closure)	bcl,dcl,gcl
m	*mom*	em	epi (sil)	(epinthetic closure)	
n	*non*	nx	sil	(silence)	h#,#h,pau

Figure 2.1: 48 to 39 phoneme folding as shown in [19].

whole network was optimized using momentum gradient descent with a momentum term of 0.9 [11, page 78] and a learning rate of $10^{(} - 3)$. To ensure generalization normal input noise $\mathcal{N}(\mu = 0, \sigma = 0.65)$ was added to the inputs. All weights where initialized using another Gaussian $\mathcal{N}(0, 0.1)$. Gradients where clipped such that all gradient elements are within $(-1, 1)$. Results of the training process are shown in figure 2.2.

2.3 PBLSTM-CTC

2.4 Las-Experiments

2.4.1 Greedy Decoding

2.4.2 Beam-search Decoding

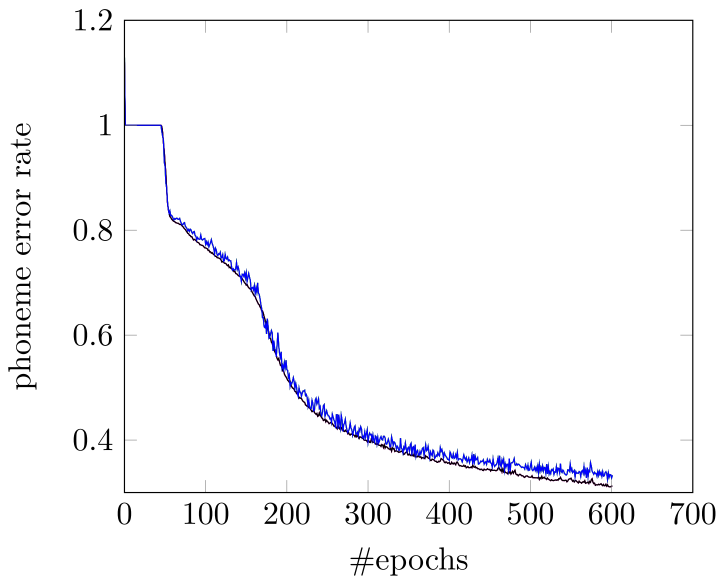

Figure 2.2: Validation and Test set error while training a two layer BLSTM network with CTC output layer on the TIMIT speech corpus with 39 folded phonemes.

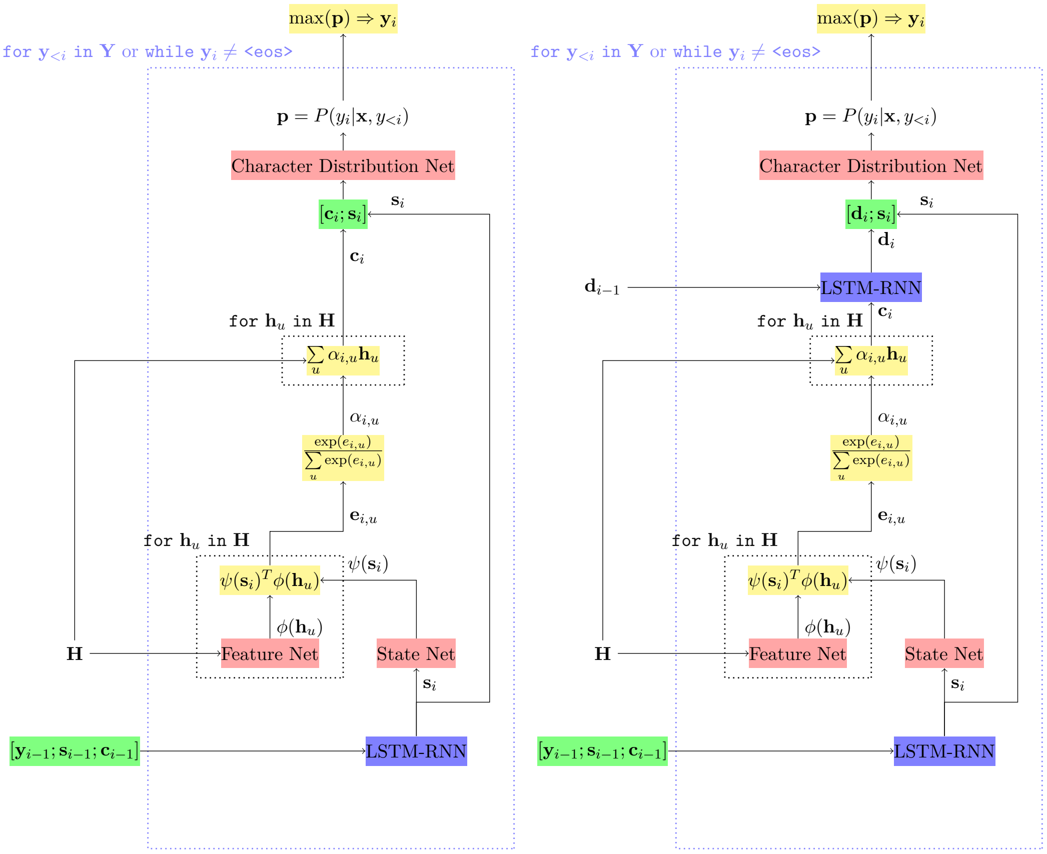
$\max(\mathbf{p}) \Rightarrow \mathbf{y}_i$
for $\mathbf{y}_{<i}$ in $\mathbf{Y}$ or while $\mathbf{y}_i \neq$ <eos>
$\mathbf{p} = P(y_i|\mathbf{x}, y_{<i})$
Character Distribution Net
$[\mathbf{c}_i; \mathbf{s}_i]$
$\mathbf{s}_i$
$\mathbf{c}_i$
for $\mathbf{h}_u$ in $\mathbf{H}$
$\sum_u \alpha_{i,u}\mathbf{h}_u$
$\alpha_{i,u}$
$\frac{\exp(e_{i,u})}{\sum_u \exp(e_{i,u})}$
$\mathbf{e}_{i,u}$
for $\mathbf{h}_u$ in $\mathbf{H}$
$\psi(\mathbf{s}_i)$
$\psi(\mathbf{s}_i)^T\phi(\mathbf{h}_u)$
$\phi(\mathbf{h}_u)$
$\mathbf{H}$
Feature Net
State Net
$\mathbf{s}_i$
$[\mathbf{y}_{i-1}; \mathbf{s}_{i-1}; \mathbf{c}_{i-1}]$
LSTM-RNN
$\max(\mathbf{p}) \Rightarrow \mathbf{y}_i$
for $\mathbf{y}_{<i}$ in $\mathbf{Y}$ or while $\mathbf{y}_i \neq$ <eos>
$\mathbf{p} = P(y_i|\mathbf{x}, y_{<i})$
Character Distribution Net
$[\mathbf{d}_i; \mathbf{s}_i]$
$\mathbf{s}_i$
$\mathbf{d}_i$
$\mathbf{d}_{i-1}$
LSTM-RNN
$\mathbf{c}_i$
for $\mathbf{h}_u$ in $\mathbf{H}$
$\sum_u \alpha_{i,u}\mathbf{h}_u$
$\alpha_{i,u}$
$\frac{\exp(e_{i,u})}{\sum_u \exp(e_{i,u})}$
$\mathbf{e}_{i,u}$
for $\mathbf{h}_u$ in $\mathbf{H}$
$\psi(\mathbf{s}_i)$
$\psi(\mathbf{s}_i)^T\phi(\mathbf{h}_u)$
$\phi(\mathbf{h}_u)$
$\mathbf{H}$
Feature Net
State Net
$\mathbf{s}_i$
$[\mathbf{y}_{i-1}; \mathbf{s}_{i-1}; \mathbf{c}_{i-1}]$
LSTM-RNN

Figure 2.3: Two different attend and spell cell configurations.

Bibliography

[1] A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

[2] D. Bahdanau, K. Cho, and Y. Bengio. NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE. pages 1–15, 2015.

[3] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *arXiv*, pages 1–9, 2015.

[4] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. *IEEE International Conference on Neural Networks - Conference Proceedings*, 1993-Janua:1183–1188, 1993.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1995.

[6] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals. Listen, attend and spell. *arXiv preprint*, pages 1–16, 2015.

[7] Christopher Olah. Understanding LSTM Networks, 2015.

[8] M. Diehl. Script for Numerical Optimization Course B-KUL-H03E3A. Technical report, Optimization in Engineering Center (OPTEC) and Electrical Engineering Department (ESAT-SCD), KU Leuven, Leuven, 2013.

[9] J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren, and V. Zue. TIMIT Acoustic-Phonetic Continuous Speech Corpus, 1993.

[10] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* 2008.

[11] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. 2008.

[12] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, pages 1–43, 2013.

[13] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *Proceedings of the 23rd international conference on Machine Learning*, pages 369–376, 2006.

[14] A. Graves, A.-R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (6):6645–6649, 2013.

[15] S. Hochreiter. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 1998.

[16] S. Hochreiter and J. Schmidhuber. LONG SHORT TERM MEMORY. *Technical Report FKI-207-95*, pages 1–8, 1995.

[17] X. Huang, A. Acero, and H.-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development.* 2001.

[18] B. H. Juang, L. R. Rabiner, and J. G. Wilpon. On the use of bandpass filtering in speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(7), 1987.

[19] K. F. Lee and H. W. Hon. Speaker-independent phone recognition using hidden Markov models. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(11):1641–1648, 1989.

[20] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *Proceedings of The 30th International Conference on Machine Learning*, (2):1310–1318, 2012.

[21] R. Rojas. *Neural Networks - A Systematic Introduction - Backpropagation.* Springer Berlin Heidelberg, 1996.

[22] P. Woodland, G. Evermann, and M. Gales. HTKBook, 2006.

Master thesis filing card

Student: Moritz Wolter

Title: Building a state of the art speech recogniser

UDC: 621.3

Abstract:

In the past machine learning relied heavily on algorithms designed by experts to solve a specific task. Which lead to highly sophisticated algorithms, which could be grasped only by small groups of people. The human brain however does not work this way, although specialized areas exist, these areas consist of similar building blocks. Artificial neural networks attempt to mimic this layout. Similar algorithmic structures are used for a wide variety of tasks. This thesis deals with the application of neural networks in speech recognition. Replacing the various subsystems by one integrated network based approach.

Thesis submitted for the degree of Master of Science in Mathematical Engineering

Thesis supervisors: Prof. dr. ir. Patrick Wambacq
Prof. dr. ir. Hugo van Hamme

Assessor: Prof. dr. ir. Johan Suykens

Mentor: Ir. Vincent Renkens