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 supervisor:
Prof. dr. ir. Patrick Wambacq

Assessor:
Prof. dr. ir. Johan Suykens

Mentor:
Ir. Vincent Renkens

Academic year 2015 – 2016

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

The `abstract` environment contains a more extensive overview of the work. But it should be limited to one page.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

Con-vNet	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

The first contains a general introduction to the work. The goals are defined and the modus operandi is explained. TODO: describe the problem.

1.1 Preprocessing and feature extraction

Filter-Bank features

Filter-banks are collections of filters. These filters are spread out over the whole frequency band [13, page 251]. Filter-bank output is commonly used as input for speech analysis [13][4]. The number of filter-banks depends on the required resolution, 32 is a common choice [14]. 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 is speech recognitions is that the cochlea, in the human ear, resembles a filter bank [13, page 30]. However humans do not perceive frequency linearly. Experimental evidence suggests, that our perceptions is scaled according to [13, page 34]:

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

A normalized plot of this function is shown in figure 1.2 on the left. Mel-scaling suggests, that human 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 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 in

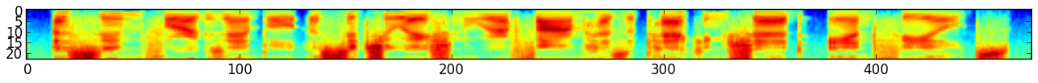


Figure 1.1: Frequency Bank input computed from a sentence contained in the *TIMIT* dataset. Time is shown on x and Frequency on the y-Axis.

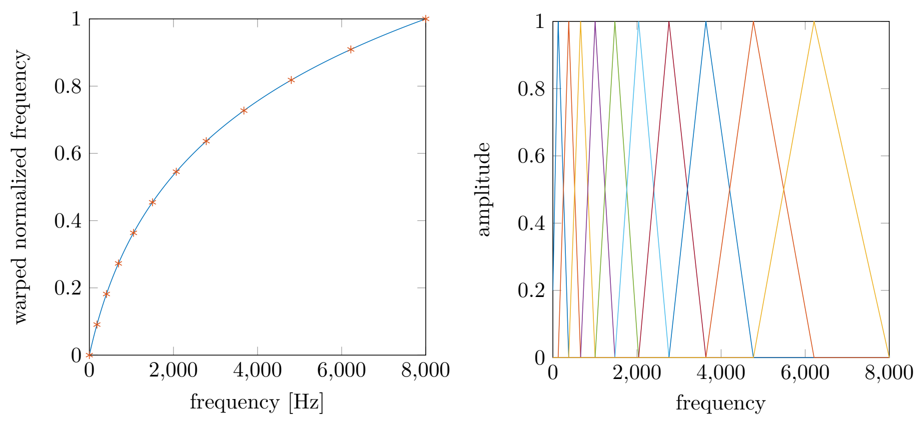


Figure 1.2: The Mel-scale (blue) with Mel-Frequency Cepstrum Coefficients (red) on the left. Filterbank with Mel-spaced filters (right).

speech recognition. The filter functions are defined by [13, 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 right of figure 1.2. The left plot shows the triangular filter bank using, where the spacing is done 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.
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.

1.1.1 Text and Phoneme output

1.2 Deep Neural Networks

1.2.1 Gradient descent

The optimization process of neural networks is done based on a training data set $\{\{\mathbf{x}_1, \mathbf{t}_1\}, \ldots, \{\mathbf{x}_p, \mathbf{t}_p\}\}$ [16, page 156]. The elements of this set are the input- and output-patters $\mathbf{x}$ and $\mathbf{t}$ respectively. Ideally the network output should be the same as the desired one for all data pairs. The difference between target and current performance is measured by the cost function[16, page 156]:

$$E = \frac{1}{2}\sum_{i=1}^{p} \|\mathbf{o}_i - \mathbf{t}_i\|^2. \tag{1.6}$$

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 as small as possible. After the training process is done 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 $\frac{1}{2}(o_{ij} - t_{ij})^2$[16, page 157]. 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. Its operation is now illustrated using the function [6, page 69]:

$$f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_1 x_2 x_3) \tag{1.7}$$

This function written in terms of five elementary operations as [6, page 70]:

$$x_4 = x_1 x_2 \tag{1.8}$$
$$x_5 = sin(x_4) \tag{1.9}$$
$$x_6 = x_4 x_3 \tag{1.10}$$
$$x_7 = exp(x_6) \tag{1.11}$$
$$x_8 = x_5 + x_7 \tag{1.12}$$
$$y = x_8 \tag{1.13}$$

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

$$\frac{\partial f(x_1, x_2, x_3)}{\partial x_1} = x_2(\cos(x_1 x_2) + x_3 \exp(x_1 x_2 x_3)) \tag{1.14}$$
$$\frac{\partial f(x_1, x_2, x_3)}{\partial x_2} = x_1(\cos(x_1 x_2) + x_3 \exp(x_1 x_2 x_3)) \tag{1.15}$$
$$\frac{\partial f(x_1, x_2, x_3)}{\partial x_3} = x_1 x_2 \exp(x_1 x_2 x_3) \tag{1.16}$$

Above the derivatives have been found by hand using the chain rule. Now these will be computed using back-propagation. Figure 1.3 shows a graphical representation of equation 1.7. 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 1.3 this means computing x_4 to x_8.
After the forward sweep the gradient is found by going back trough the network from 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 1.4. 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.

Stochastic gradient descent

When training 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 randomized gradient descent. With an added momentum term it can be formalized as [10, page 4]:

$$\triangle w_{ij}(t) = \alpha \triangle w_{ij}(t-1) - \epsilon \frac{\partial C}{\partial w_{ij}(t)} \tag{1.17}$$

C is the cost, which is computed by comparing the current network output to the desired output. $\alpha \in (0,1)$ is a momentum coefficient. The weight of a connection from unit i in the layer under consideration to unit j in the layer below is given by the expression w_{ij}.

1.2.2 Convnets

When looking for a signal in different parts of a recoding it is not always advisable to relearn recognition of the signal in different locations. With a conventional fully connected structure every sample on the time axis gets its own input weight. For shifted version of this signal the network will not be able to reuse weights it used to recognize the same signal at another time point in the recording. Convolutional neural nets aim to solve this problem [7, page 6], while preserving essential ordering information.

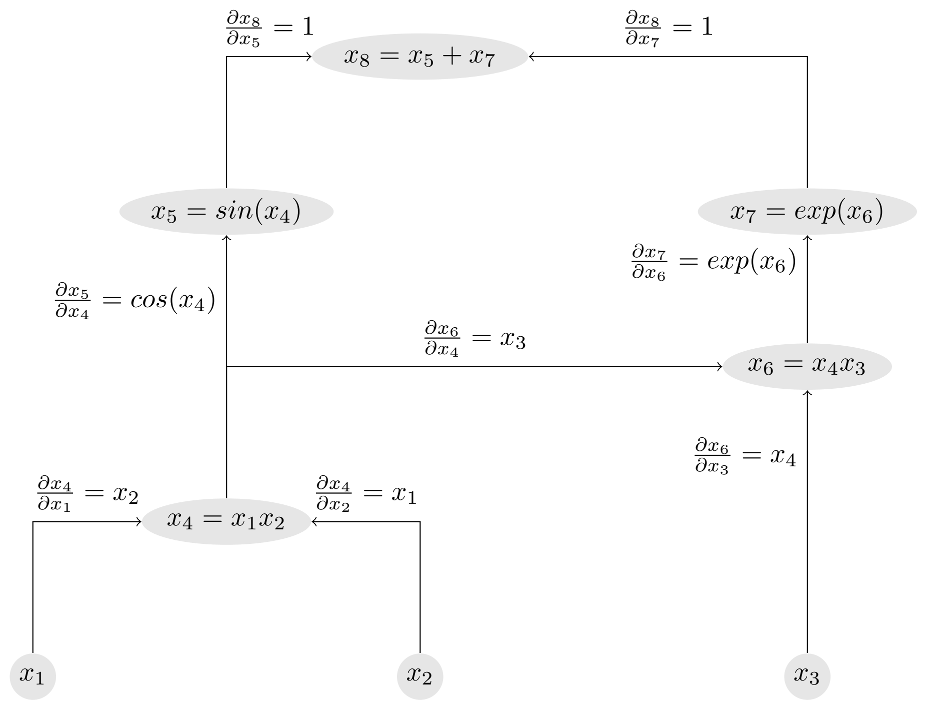

$\frac{\partial x_8}{\partial x_5} = 1$
$x_8 = x_5 + x_7$
$\frac{\partial x_8}{\partial x_7} = 1$
$x_5 = sin(x_4)$
$x_7 = exp(x_6)$
$\frac{\partial x_7}{\partial x_6} = exp(x_6)$
$\frac{\partial x_5}{\partial x_4} = cos(x_4)$
$\frac{\partial x_6}{\partial x_4} = x_3$
$x_6 = x_4x_3$
$\frac{\partial x_6}{\partial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

Figure 1.3: Example funciton network with partial derivatives.

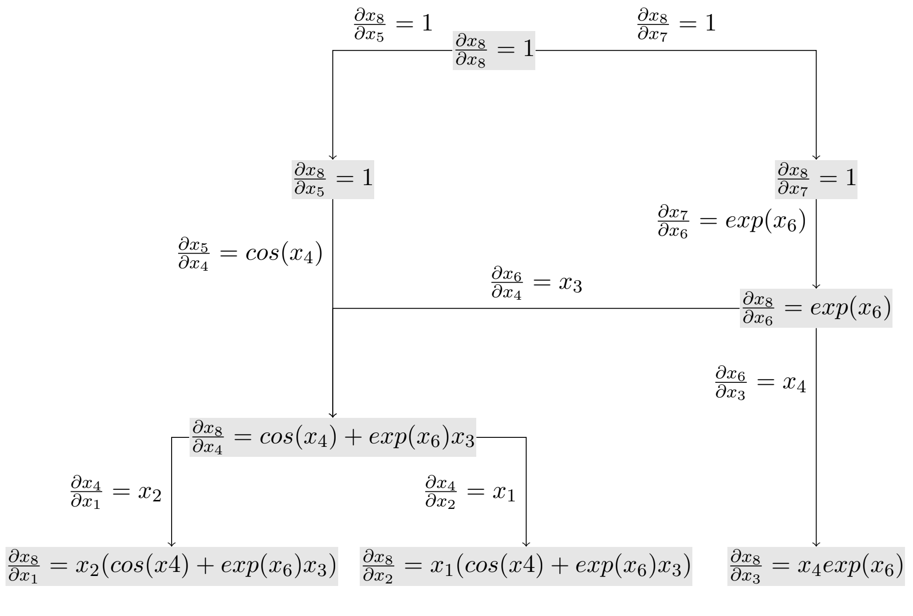

$\frac{\partial x_8}{\partial x_5} = 1$
$\frac{\partial x_8}{\partial x_8} = 1$
$\frac{\partial x_8}{\partial x_7} = 1$
$\frac{\partial x_8}{\partial x_5} = 1$
$\frac{\partial x_8}{\partial x_7} = 1$
$\frac{\partial x_7}{\partial x_6} = exp(x_6)$
$\frac{\partial x_5}{\partial x_4} = cos(x_4)$
$\frac{\partial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

Figure 1.4: Reverse sweep.

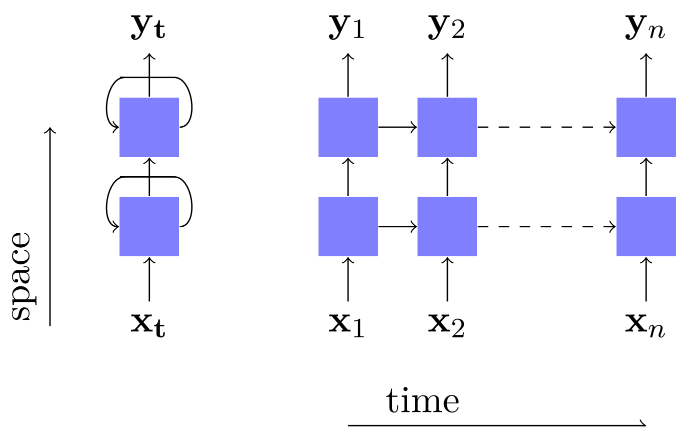


Figure 1.5: Rolled (left) and unrolled (right) recurrent neural net with two units.

1.3 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. This state is fed back into the cell together with new inputs every time step. Such a layout is shown in figure 1.5 on the left. Another way to depict the same network is to not only consider the spacial dimension, but add the time axis as well. Figures, which show the spacial and time dimension are called unrolled network diagrams, shown in figure 1.5 on the right. When looking at the

1.3.1 The exploding and vanishing gradient problem

Even tough past information is available in theory, learning long time dependencies is problematic with classical neural nets. The back-propagated derivative can sometimes become waker and weaker until it ultimately vanishes [11]. Another problem is that sometimes classical recurrent neural nets produce a gradient that blows up [15]. The exploding gradients can be fixed by clipping, but vanishing gradients require more sophisticated treatment [3].

1.3.2 Long short-term memory

Research seemed to be focused on solving the problem by making changes to the back-propagation algorithm. However a good solution to the problem turned out to be changing the network instead. Long short-term memory (LSTM) cells as proposed in [12] are more complex network units. These cells use the equation system [8, page

5]:

$$\mathbf{i_t} = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h_{t-1}} + \mathbf{W}_{ic}\mathbf{c_{t-1}} + \mathbf{b}_i) \quad (1.18)$$
$$\mathbf{f_t} = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h_{t-1}} + \mathbf{W}_{fc}\mathbf{c_{t-1}} + \mathbf{b}_f) \quad (1.19)$$
$$\mathbf{c_t} = \mathbf{f_t}\mathbf{c_{t-1}} + \mathbf{i_t}\tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h_{t-1}} + \mathbf{b}_c) \quad (1.20)$$
$$\mathbf{o_t} = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h_{t-1}} + \mathbf{W}_{oc}\mathbf{c_t} + \mathbf{b}_o) \quad (1.21)$$
$$\mathbf{h_t} = \mathbf{o}_t\tanh(\mathbf{c}_t) \quad (1.22)$$
$$(1.23)$$

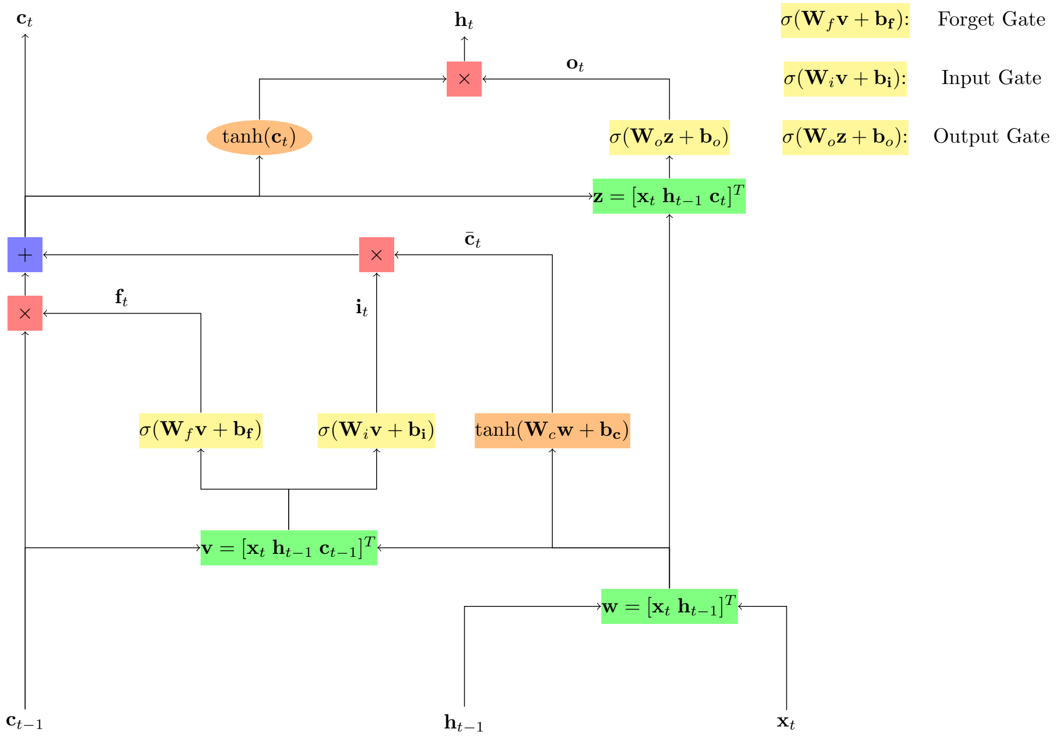
From the definition of the matrix product follows that

$$\mathbf{A}\mathbf{x}_1 + \mathbf{B}\mathbf{x}_2 = \begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2\end{bmatrix}. \quad (1.24)$$

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:

$$\mathbf{i_t} = \sigma(\mathbf{W}_i[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c_{t-1}}]^T + \mathbf{b}_i) \quad (1.25)$$
$$\mathbf{f_t} = \sigma(\mathbf{W}_f[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c_{t-1}}]^T + \mathbf{b}_f) \quad (1.26)$$
$$\mathbf{c_t} = \mathbf{f}_t\mathbf{c_{t-1}} + \mathbf{i}_t\tanh(\mathbf{W}_c[\mathbf{x}_t\ \mathbf{h_{t-1}}]^T + \mathbf{b}_c) \quad (1.27)$$
$$\mathbf{o_t} = \sigma(\mathbf{W}_o[\mathbf{x}_t\ \mathbf{h_{t-1}}\ \mathbf{c_t}]^T + \mathbf{b}_o) \quad (1.28)$$
$$\mathbf{h_t} = \mathbf{o_t}\tanh(\mathbf{c_t}) \quad (1.29)$$

This system of equations is visualized in figure 1.6. The diagram is read from bottom to top. The most important part is the line from $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$ [5]. 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 $\sigma()$ 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 [5]. 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() function makes sure all vector elements are between -1 and 1. The neuron looks at input data and the past outputs. Both are labeled $\mathbf{w}$ in figure 1.6, $\mathbf{w}$ 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 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 $\mathbf{h}_t$.



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

Figure 1.6: Visualization of the LSTM architecture.

1.3.3 Bidirectional Long Short Term Memory

With the advent of LSTMs deep recurrent networks became feasible in speech recognition [9]. 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 1.7. It is important to note, that linear neurons are used to compute the LSTM input as well as the outputs according to the equations [9]:

$$\overrightarrow{\mathbf{h}}_t = \mathrm{LSTM}(\mathbf{W}_{\overrightarrow{\mathbf{h}}_t}[\mathbf{x}_t\ \mathbf{h}_{t-1}]^T + \mathbf{b}_{\overrightarrow{\mathbf{h}}_t}) \tag{1.30}$$
$$\overleftarrow{\mathbf{h}}_t = \mathrm{LSTM}(\mathbf{W}_{\overleftarrow{\mathbf{h}}_t}[\mathbf{x}_t\ \mathbf{h}_{t+1}]^T + \mathbf{b}_{\overleftarrow{\mathbf{h}}_t}) \tag{1.31}$$
$$\mathbf{y}_t = \mathbf{W}_y[\overrightarrow{\mathbf{h}}_t\ \overleftarrow{\mathbf{h}}_t]^T + \mathbf{b}_y \tag{1.32}$$

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:

$$\overrightarrow{\mathbf{h}}_t^n = \mathrm{LSTM}(\mathbf{W}^n_{\overrightarrow{\mathbf{h}}_t}[\mathbf{h}_t^{n-1}\ \mathbf{h}_{t-1}^n]^T + \mathbf{b}^n_{\overrightarrow{\mathbf{h}}_t}) \tag{1.33}$$
$$\overleftarrow{\mathbf{h}}_t^n = \mathrm{LSTM}(\mathbf{W}^n_{\overleftarrow{\mathbf{h}}_t}[\mathbf{h}_t^{n-1}\ \mathbf{h}_{t+1}^n]^T + \mathbf{b}^n_{\overleftarrow{\mathbf{h}}_t}) \tag{1.34}$$
$$\mathbf{h}_t^n = \mathbf{W}_y^n[\overrightarrow{\mathbf{h}}_t\ \overleftarrow{\mathbf{h}}_t]^T + \mathbf{b}_y^n \tag{1.35}$$

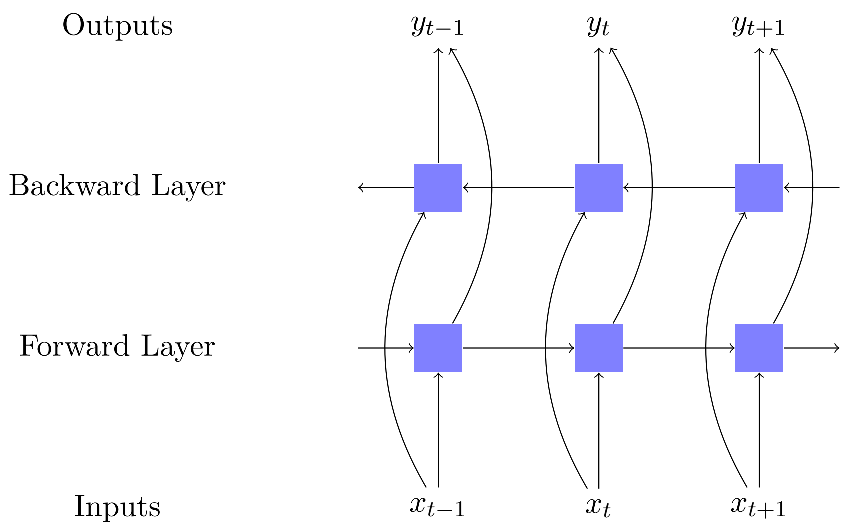


Figure 1.7: A bidirectional Long short term memory layer, according to [9]

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

1.4 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 after installation can be used from within Python or C++.

1.5 Listen, Attend and Spell

The Listen Attend and Spell architecture (LAS) is the main Idea around which this thesis revolves. This entire section is based on [4]. The las-network consists of two mayor parts, the listener and the speller. The listener is a pyramidal recurrent neural net. It accepts filter bank spectra $\mathbf{x}_n$ as inputs and produces high level output features $\mathbf{h}_m$. The speller in turn accepts the features as input and outputs distributions over Latin character sequences $\mathbf{y}_p$. An overview of the las-achrcitecture is given in figure 1.8.

1.5.1 The listener

The listener shown in figure 1.8 on the bottom, consists of Bidirectional Long Short Term Memory RNN (BLSTM) blocks. This choice implies that only fully recorded data can be analyzed. These blocks are arranged in a pyramidal structure, such that the time resolution is cut in half in every 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. Additionally the compression reduces the problem complexity, which speeds up the training process significantly [4, page 4].

1.5.2 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 time, $i-1$ is used to refer to results from the last time step.
These values are computed using [4, page 4]:

$$s_i = \text{RNN}(\mathbf{s}_{i-1}, \mathbf{y}_{i-1}, \mathbf{c}_{i-1}) \tag{1.36}$$

$$\mathbf{c}_i = \text{AttentionContext}(\mathbf{s}_i, \mathbf{H}) \tag{1.37}$$

$$P(\mathbf{y}_i|\mathbf{x}, \mathbf{y}_{<i}) = \text{CharacterDistribution}(s_i, \mathbf{c}_i) \tag{1.38}$$

The state follows from a recurrent neural net (RNN) made of a two layer LSTM. 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 [4, page 5]:

$$e_{i,u} = \phi(\mathbf{s}_i)^T \psi(\mathbf{h_u}) \tag{1.39}$$

$$\alpha_{i,u} = \frac{\exp(e_{i,u})}{\sum\limits_u \exp(e_{i,u})} \tag{1.40}$$

$$\mathbf{c}_i = \sum_u \alpha_{i,u} \mathbf{h}_u \tag{1.41}$$

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[4, page 5]. Thus it is justified to think of the alphas as a sliding window. This window contains only those parts of the condensed input data set, which are currently relevant.

1.5.3 Training

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

$$\max_\theta \sum_i \log P(y_i|\mathbf{x}, y_{<i}; \theta). \tag{1.42}$$

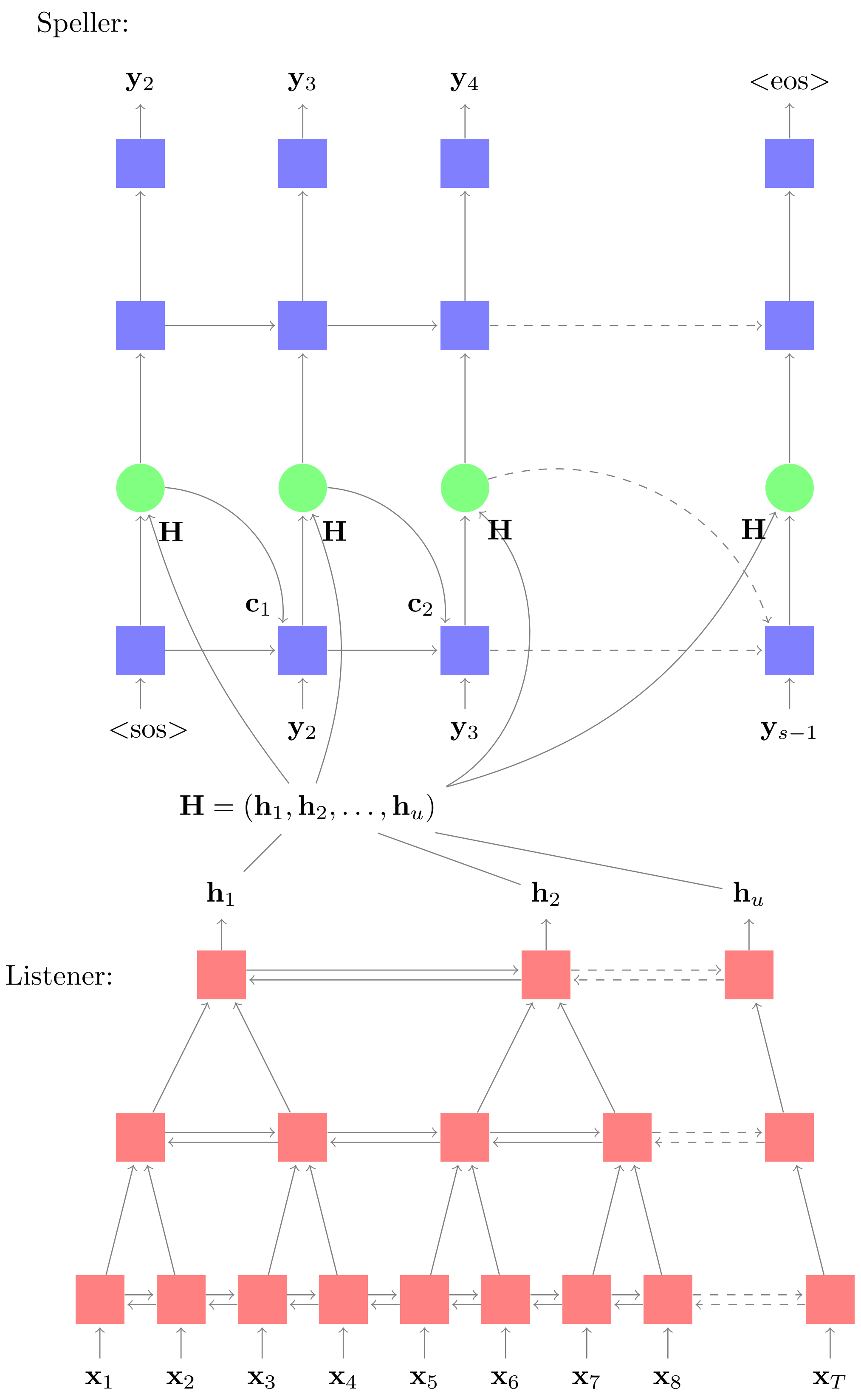


Figure 1.8: The LAS architecture [4, page 3]. BLSTM blocks are shown in red. LSTM blocks in blue and attention nets in green.

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. 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 trained. Which leads to the objective [4, page 5]:

$$\hat{y}_i = \text{CharacterDistribution}(s_i, \mathbf{c}_i) \tag{1.43}$$

$$\max_{\theta} \sum_i \log R(y_i | \mathbf{x}, \hat{y}_{<i}; \theta) \tag{1.44}$$

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 [2].

1.5.4 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 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. 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. Beam search generates a set of possible speech transcriptions. A language model trained on text data only is the used to select on of the beams according to [4, page 6]:

$$s(\mathbf{y}|\mathbf{x}) = \frac{\log P(\mathbf{y}|\mathbf{x})}{|\mathbf{y}|_c} + \lambda \log P_{LM}(\mathbf{y}) \tag{1.45}$$

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

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

[3] 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.

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

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

[6] 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.

[7] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. pages 1–28, 2016.

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

[9] 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.

[10] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[11] 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.

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

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

[14] 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.

[15] 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.

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

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 supervisor: Prof. dr. ir. Patrick Wambacq

Assessor: Prof. dr. ir. Johan Suykens

Mentor: Ir. Vincent Renkens