Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Developing an automatic speech recognition system based on Czech spoken language

Bc. Richard Werner

Department of Applied Mathematics
Supervisor: Mgr. Alexander Kovalenko, Ph.D.

May 4, 2020

Acknowledgements

THANKS (remove entirely in case you do not with to thank anyone)

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 4, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Werner, Richard. *Developing an automatic speech recognition system based on Czech spoken language.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

Summarize the contents and contribution of your work in a few sentences in English language.

Keywords Replace with comma-separated list of keywords in English.

Contents

List of Figures

List of Tables

Introduction

Chapter 1

Neural networks

Inspiration to create artificial neural networks (commonly referred to as "neural networks") was taken from the entirely different way in which human brain processes and computes information, compared to conventional digital computers. The human brain is a highly complex, nonlinear, and parallel information-processing system with the capability to organize it is structural components, known as *neurons*, to perform several different kinds of computation (e.g., perception, pattern recognition or motor functions). [1]

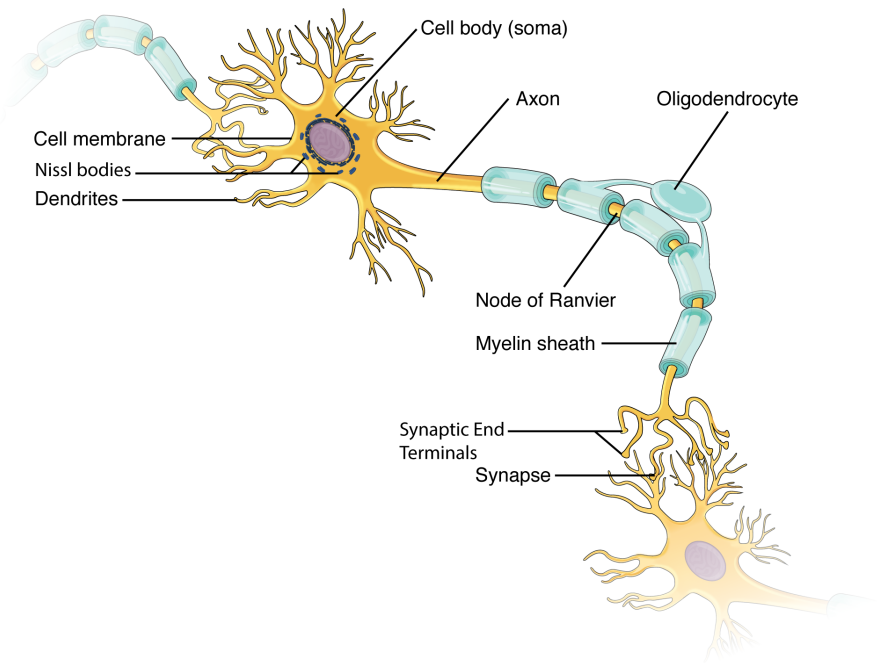


Figure 1.1: Neuron anatomy by [2]

As shown in figure 1.1, every neuron has a branch-like structure of dendrites which serve as receivers of information from connected neurons or another kind of receptor/cell. From dendrites, the information is then passed as an electric signal through the cell body (soma) to the axon. The purpose of the axon is the transportation of the signal to adjacent cells. On the end of the axon is another branch-like structure called synaptic end terminals (or axon terminals). Those are the transfer points for the information to other cells. [3]

An essential feature of the brain, which is called brain *plasticity*, is the ability to *adapt* to its surrounding environment. And just as the plasticity is the key to the functioning of the brain, the same applies to the artificial neural networks and their artificial neurons.

In its most general form, a neural network is a machine that tries to *model* how the human brain performs a particular task or function of interest. The network is usually implemented by using electronic components or is simulated by software on a digital computer. With this in mind, let us define neural networks as follows:

> *A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*
>
> 1. *Knowledge is acquired by the network from its environment through a learning process.*
> 2. *Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge. [1]*

The mentioned learning process is performed by a *learning algorithm*, which is a function that modifies the synaptic weights of the network to fit it to a specific problem or objective. Apart from modifying the internal weights, some networks are capable of even modifying their topology to suit themselves to a problem better.

In the following sections, I will go through the first concepts and history of neural networks, their learning algorithms, and advanced types of NN.

1.1 Basic concept – Rosenblatt's Perceptron

Rosenblatt's perceptron was the first algorithmically described neural network. It was invented by psychologist Rosenblatt who inspired engineers, physicists, and mathematicians alike to devote their research effort to different aspects of neural networks in the 60s and 70s of the 19th century. Moreover, the concept of perceptron in its basic form is as valid today as it was in 1958 when Rosenblatt's paper on the perceptron was first published. [1]

An artificial neural network, in general, can be viewed as an oriented graph, where nodes are neurons, and edges are the inter-neuron connections. Each neuron consists of three basic components [4]:

1. **weight vector**, which stores the weights of the connections between neurons,
2. **summation function**, which computes weighted sum of all inputs, and
3. **activation function** that maps the result of sum function to a number based on the specific type of the activation function.

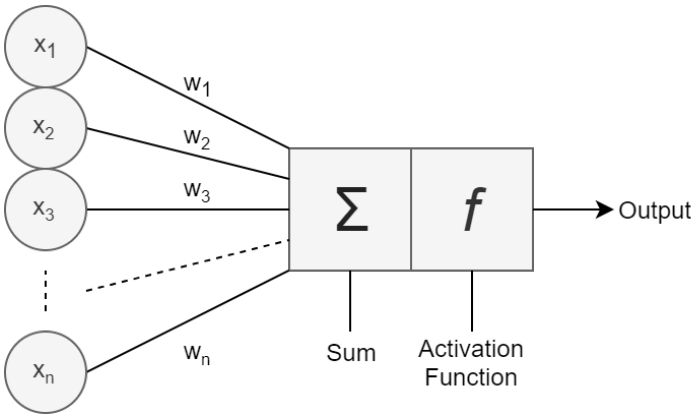


Figure 1.2: Perceptron schema

In the figure 1.2 you can see the general schema of a perceptron cell. Each of the inputs x_i has a corresponding weight w_i. The weighted sum of the inputs a would then be computed as:

$$a = \sum_{i=0}^{n} w_i x_i + b,$$

where b is externally applied bias.

The activation function of the original perceptron was implemented as the signum function. That means the perceptron was able to classify an input vector $X = (x_1, ..., x_n)$ into one of two classes, because the range of this

function is a two-value set $\{-1, 1\}$. In the purest form of the perceptron, there are two decision regions separated by a *hyperplane*, which is defined by:

$$\sum_{i=0}^{n} w_i x_i + b = 0$$

which would for example mean a simple line in a 2D space:

$$w_1 x_1 + x_2 x_2 + b = 0$$

Since the basic perceptron is only able to generate a hyperplane as a decision boundary, a perceptron model is limited to linearly separable problems, thus is unable to classify any more complex tasks correctly. This leads us to a conclusion, that something more robust is needed. [1, 3]

1.2 Multilayer perceptron

The multilayer perceptron (MLP) consists of multiple layers of neurons that interact using weighted connections. After the first input layer, there are usually several more hidden layers, followed by the last output layer. There are no connections between neurons in one layer, while all neurons in one layer are fully connected to neurons in adjacent layers. While this statement tends to create an impression that all the information from one layer is copied to all the neurons of the adjacent layer, it is not necessarily so. Since the connection weights are usually real numbers, some of the connections may effectively be rendered insignificant. [5, 1] There is an example of a multilayer NN in figure 1.3.

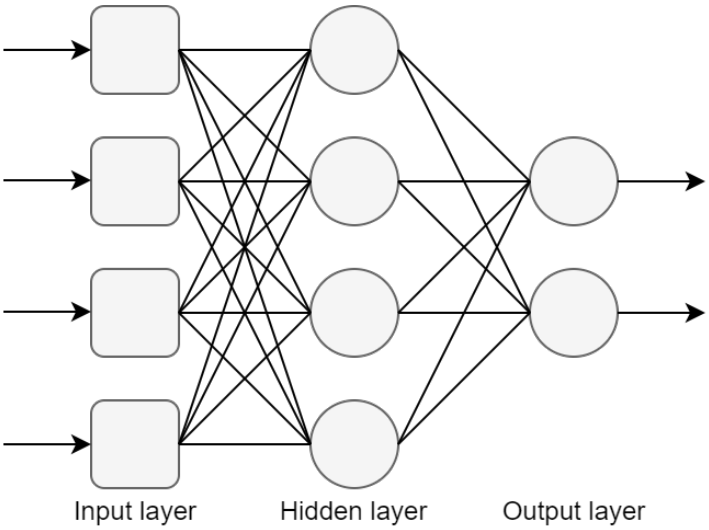


Figure 1.3: Multilayer NN example schema

Another difference over the simple perceptron is the activation function. Each model of neuron includes a nonlinear activation function that is *differentiable*.

Those characteristics, however, are responsible for the deficiencies in our understanding of the behavior of the network. The distributed non-linearity and high connectivity of the network make the analysis of the network quite hard to undertake. Moreover, the use of hidden layers makes the learning process harder to visualize.

1.2.1 Backpropagation

Backpropagation, short for "backward propagation of errors", is an algorithm for supervised learning of ANNs, which is using the *gradient descend*. In this

section, especially in the formal definition, I've drawn from [6]. It proceeds in two phases:

1. In the **forward phase**, the connection weights of the network are fixed, and the input is propagated through the network, layer by layer, until it reaches the output. In other words, the network performs inference.

2. In the **backward phase**, the error is computed by comparing the network output with the desired output. The resulting error is then propagated through the network, but this time from the output layer to the input. In this phase, appropriate adjustments are made to the connection weights based on the amount of error introduced by the corresponding connection. [1]

Formal definition

If we were to define the backpropagation formally, there would be three things required:

1. A **dataset** consisting of input-output pairs $(\vec{x_i}, \vec{y_i})$, where $\vec{x_i}$ is the input and $\vec{y_i}$ is the corresponding desired output. The dataset of size N is denoted $X = \{(\vec{x_1}, \vec{y_1}), ..., (\vec{x_N}, \vec{y_N})\}$.

2. A **feedforward NN**, e.g. our MLP, whose parameters are collectively denoted θ. Parameters of primary interest are w_{ij}^k, which is the connection weight between node j in layer l_k and node i in layer l_{k-1}, and b_i^k, the bias for node i in layer l_k.

3. An **error function** $E(X, \theta)$, which defines the error between the desired output $\vec{y_i}$ and the calculated (inferred) output $\hat{\vec{y_i}}$ of the NN on input $\vec{x_i}$ for a dataset X and parameters θ.

Training a NN with gradient descent requires the calculation of the gradient of the error function $E(X, \theta)$ with respect to the network parameters. Then, according to the learning rate α, each iteration of gradient descend updates the weights and biases θ as:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta},$$

where θ^t denotes the parameters of the network at iteration t in gradient descend.

The following formulation is for a neural network with one output. The algorithm can be applied to a network with any number of outputs by consistent application of the chain rule and power rule. Thus, for all the examples below, input-output pair will be of the form $(\vec{x}, y)$, i.e. the target variable is a number, not a vector.

Preliminaries

Let us use the *mean squared error* as our error function:

$$E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2,$$

where y_i is the target output value and $\hat{y}_i$ is the computed output value for the input $\vec{x_i}$. This function is used for numeric (not a vector) output. If we were to predict vectors, the use of a vector similarity function would be necessary.

Lets us lay down a few preliminaries:

- the sum of weighted inputs for the i-th node in the k-th layer a_i^k is:

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ij}^k o_j^{k-1} = \sum_{j=0}^{r_{k-1}} w_{ij}^k o_j^{k-1}$$

  where b_i^k is bias for node i in layer k, r_k is number of nodes in layer l_k, w_{ij}^k is weight for node j in layer l_k from incoming node i and o_j^k is output for node i in layer l_k. After the standard version with the explicit bias follows the one with the bias incorporated into the weights:

$$w_{i0}^k = b_i^k.$$

- Backpropagation attempts to minimize the error function with respect to the NN weights by calculating the value of $\frac{\partial E}{\partial w_{ij}^k}$ for each weight w_{ij}^k. We can decompose the derivative with respect to each input-output pair as follows:

$$\frac{\partial E(X,\theta)}{\partial w_{ij}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial}{\partial w_{ij}^k}\left(\frac{1}{2}(\hat{y}_d - y_d)^2\right) = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial E_d}{\partial w_{ij}^k}.$$

Error function derivatives

The derivation of the backpropagation algorithm begins by applying the *chain rule* to the error function partial derivative:

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k},$$

where a_j^k is the weighted sum (activation) of node j in layer k before it is passed to the nonlinear activation function. This decomposition simply says the change in the error function due to a weight is a product of the change in the error function E due to the sum a_j^k times the change in the sum a_j^k due to the weight w_{ij}^k.

Mostly, the whole equation is simplified by calling the first term the *error* and denoting it

$$\delta_j^k = \frac{\partial E}{\partial a_j^k}.$$

The second term is calculated from the equation for a_j^k above:

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{l=0}^{r_{k-1}} w_{lj}^k o_l^{k-1}\right) = o_i^{k-1}.$$

And finally the partial derivative of the error function E with respect to a weight w_{ij}^k is

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1}.$$

This is quite intuitive, since the weight w_{ij}^k connects the output of node i in layer $k-1$ to the input of node j in layer k in the computation graph.

The output layer

Backpropagation starts from the final layer, thus it attempts to define the value δ_1^m, where m is the index of the final layer and the subscript is constant 1, because we are concerned with only one-output NN. We are able to express the output value by a neuron from the weighted sum from the previous layer and the activation function of the current neuron. Therefore, the error function can be expressed as

$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(g_0(a_1^m - y)^2,$$

where $g_0(x)$ is the activation function for the single neuron in the output layer.

Applying the partial derivative and using the chain rule gives

$$\delta_1^m = (g_0(a_1^m) - y)g_0'(a_1^m) = (\hat{y} - y)g_0'(a_1^m).$$

Putting this together with the partial derivative of the error function E with respect to a weight in the final layer w_{i1}^m gives

$$\frac{\partial E}{\partial w_{i1}^m} = \delta_1^m o_i^{m-1} = (\hat{y} - y)g_o'(a_1^m)o_i^{m-1}.$$

The hidden layers

Firstly, we need an equation for the error term δ_j^k for layers $1 \leq k < m$. Again, with the help from the chain rule, we get:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{r^{k+1}} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k},$$

where l ranges from 1 to r^{k+1}. l does not take the value of zero because the bias input o_0^k corresponding to w_{0j}^{k+1} is fixed, and its value is not dependent on the outputs of previous layers.

Using the error term δ_l^{k+1} gives the following equation:

$$\delta_j^k = \sum_{l=1}^{r_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}.$$

With definition of a_l^{k+1} as

$$a_l^{k+1} = \sum_{j=1}^{r_k} w_{jl}^{k+1} g(a_j^k),$$

where $g(x)$ is the activation function for the hidden layers, we get:

$$\frac{\partial a_l^{k+1}}{\partial a_j^k} = w_{jl}^{k+1} g'(a_j^k).$$

Using this in the above equation yields a final equation for the error term δ_j^k for the hidden layers called the *backpropagation formula*:

$$\delta_j^k = \sum_{l=1}^{r_{k+1}} \delta_l^{k+1} w_{jl}^{k+1} g'(a_j^k) = g'(a_j^k) \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

And finally, the partial derivative of the error function E with respect to a weight in the hidden layers w_{ij}^k for $1 \leq k < m$ is

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l^{k+1}.$$

The algorithm itself

Assuming a suitable value for learning rate α and a random initialization of the parameters w_{ij}^k, the backpropagation algorithm proceeds in four steps:

1. **Calculate the forward phase** for each input-output pair $(\overrightarrow{x_d}, y_d)$ and store the results and store the NN output $\hat{y_d}$ and inner neuron outputs a_j^k, and o_j^k for each node j in layer k by proceeding from layer 0, the input layer, to layer m, the output layer.

2. **Calculate the backward phase** for each input-output pair and store the results $\frac{\partial E_d}{\partial w_{ij}^k}$ for each weight w_{ij}^k connecting node i in layer $k-1$ to node j in layer k by proceeding from layer m to layer 1. There are three steps to compute the partial derivatives:

   a) Evaluate the error term for the output layer δ_1^m.

   b) Backpropagate the error terms to the hidden layers δ_j^k, working from the final hidden layer.

   c) Evaluate the partial derivatives of the individual error E_d with respect to w_{ij}^k.

3. **Combine the individual gradients** $\frac{\partial E_d}{\partial w_{ij}^k}$ for each pair to get the total gradient $\frac{\partial E(X,\theta)}{\partial w_{ij}^k}$ for the entire set of pairs $X = (\overrightarrow{x_1}, y_1), ..., (\overrightarrow{x_N}, y_M)$ by using the above mentioned (section 1.2.1) simple average of the individual gradients.

4. **Update the weights** according to the learning rate α and the total gradient by using

$$\Delta w_{ij}^k = -\alpha \frac{\partial E(X,\theta)}{\partial w_{ij}^k},$$

   which is moving the weights in the direction of the negative gradient – gradient descend.

1.3 Recurrent neural networks

A standard neural network takes in a fixed size vector as input, which limits its usage in situations that involve a *series* type input with no predetermined size. In other words, vanilla neural networks are unable to take into account any context or history, even if the current input is in a way based on the previous one. That is the reason why the *recurrent neural networks* (RNN) were designed and implemented.

The RNNs are made to take a series of input with no predetermined size. A single input item from the series is related to others, and likely influences its neighbors. So while they remember information from the training phase, they also learn things from prior inputs while generating outputs.

A recurrent network can take one or more input vectors and produce one or more output vectors. The outputs are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior inputs/outputs. That means the same input could produce a different output depending on previous inputs in the series. [7]

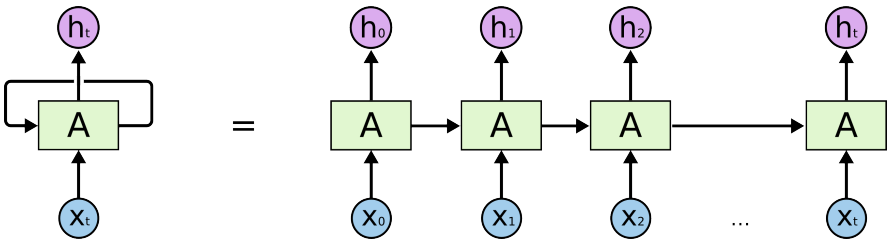


Figure 1.4: A recurrent neural network layer unfolding by [8]

The above-described concept is pictured in figure 1.4. On the unfolded neural network (the right side of the picture), we can see that part of the information from the NN in time 0 is passed to the network in time 1. That continues throughout the whole series of inputs, and in theory, the whole history of inputs and calculations will influence every other computation of the network.

There are many varieties, solutions, and constructive elements of recurrent neural networks. The difficulty of a recurrent network is that if we take into account each time step, each one of them must create its layer of neurons. That causes severe computational complexity. Besides, multilayer implementations are computationally unstable, since they tend to disappear or go off the scale. Restricting the computation to a fixed time window resolves those problems, but the model will not reflect the long-term trends. Various approaches are trying to improve the model of RNN memory and the mechanisms of remembering and forgetting. [9]

1.3.1 The problem of long-term dependencies

Sometimes, for a task, we have enough recent information to be able to compute the output with certainty. For example, imagine a language model that predicts the next word based on the previous inputs/outputs. If it tries to predict the last word in the sentence "Clouds in the sky," we do not need any context other then the rest of the sentence – it is quite apparent what would the last word regard (the sky/heaven). In such cases, where there is only a small gap between the necessary information, the standard RNNs can learn to use the recent information.

However, there are also times when we need a broader context. If a model were to predict the last words in the sentence "I grew up in France ... I speak French fluently," the recent context would probably suggest a name of a language. If it were to clarify which one, it would need the previous context, up to the beginning of the sentence. It is not rare that the gap between the necessary information and the place where it is needed becomes very large. Unfortunately, as the gap grows, the standard recurrent networks become unable to learn how to connect these pieces of information. As was mentioned above, in theory, the RNNs should handle such long-term dependencies. However, in practice, RNNs are not able to learn this.

1.3.2 The vanishing gradient problem

The problem lies in many layers using certain activation functions, a sigmoid function, for example, which squishes a large input space into a small input space between 0 and 1. That means a significant change in the input leads to a small change on the output; hence its derivative becomes small.

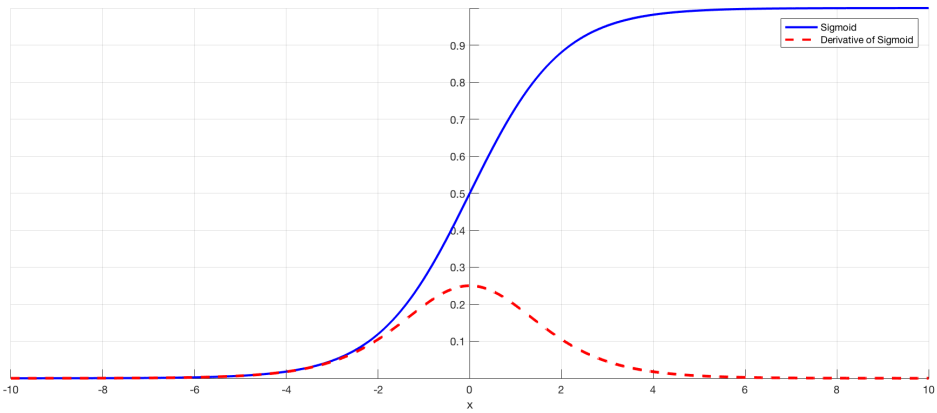


Figure 1.5: Sigmoid function and its derivative by [10]

There is an example in figure 1.5 with the sigmoid function and its derivative. When the input of the sigmoid becomes either larger or smaller, the derivative becomes close to zero. For "shallow" networks with only a few lay-

ers with this type of activation function, it is not a big problem. However, when more layers are added, it can cause the gradient to be too small to have any significant impact on the learning process.

This is caused by the chain rule used in the backpropagation, which is described for feedforward networks in section 1.2.1. It computes the gradients by moving layer by layer from the final one to the initial one. By the chain rule, each subsequent derivative is multiplied by the already computed value. Therefore, when there are n hidden layers using sigmoid-like activation function, n small derivatives are multiplied together. Thus, the gradient value decreases exponentially as the backpropagation algorithm advances to the initial layers. [10]

There are a few well-known solutions [10]:

- Probably the simplest solution is to use a different activation function, such as ReLU (Rectified Linear Unit), which does not cause a small derivative.
- Another solution is *residual networks* (ResNets). They provide residual connections (without weight parameters) straight to the next layers, effectively skipping the activation functions. That results in overall higher derivatives, and therefore the ability to train much deeper networks. [11]
- The last one is *batch normalization* layers. As mentioned before, the problem appears when a large input is mapped to small output, causing the derivatives to disappear. The batch normalization method normalizes the input on a predefined scale, where the sigmoid derivative is not too small.

1.3.3 Long-short-term memory networks

Long-short-term memory (LSTM) networks are a special type of recurrent neural networks capable of learning long-term dependencies as well as short-term ones. They work very well on a large variety of problems and are currently widely used. LSTMs are specifically designed to avoid the problem mentioned above of long-term dependencies, and they also manage to avoid the vanishing gradient problem.

Recurrent neural networks are in the form of a chain of repeating elements (neurons). In standard RNNs, those elements are usually of a straightforward structure – for example, one layer of a hyperbolic tangent (tanh). LSTM networks have that same chain structure, but each of the elements implements a much more complex structure. Instead of one layer, they implement four layers that uniquely interact with each other.

The difference between the inner structures of standard and LSTM recurrent networks can be seen in figures 1.6 for the standard approach and 1.7 for the LSTMs. In the second diagram, each black line transmits a whole

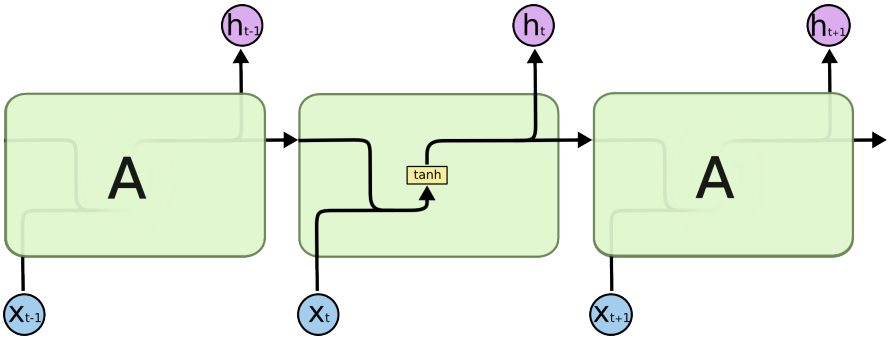


Figure 1.6: Standard RNN chain structure by [8]

vector from one node to another. Merging lines mean vector concatenation, and branching lines are copying content without any alteration. Pink circles represent pointwise operators – vector addition or multiplication. Lastly, the yellow rectangles are the trained layers of the neural network. [12]

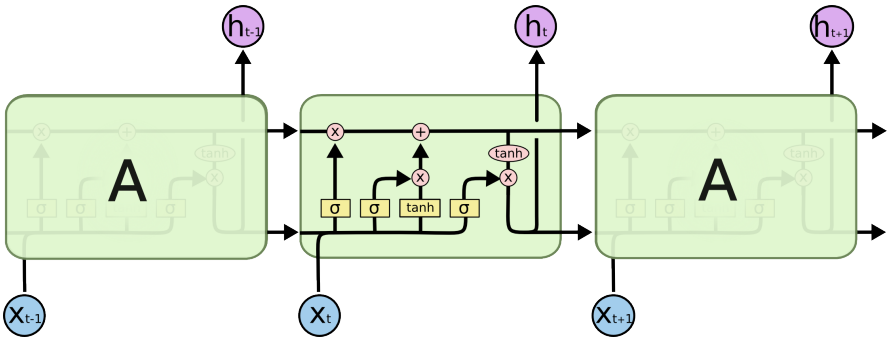
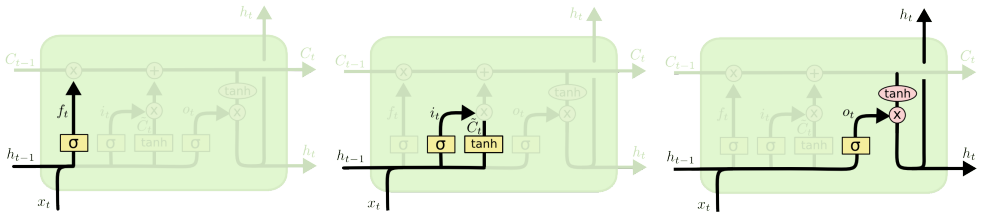


Figure 1.7: LSTM chain structure by [8]

The main feature of LSTMs is the *cell/cellular state*. That is the top horizontal line in the network cell. The neurons can remove or add information to a cellular state, while so-called *gates* carefully regulate the changes. They are composed of a sigmoid function and a pointwise multiplication operation. The sigmoid function returns values between 0 and 1, and it determines the magnitude of the change to the cellular state, whether it is information removal or addition.

To explain the information flow process through the LSTM cell, let us go step by step through it [8]:

1. The first step is to decide what information we are going to remove or keep in the cell state. It is decided by the first gate called the *forget gate layer* displayed in figure 1.8a. It combines values h_{t-1} and x_t, sends it to the sigmoid function, and the function returns a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 means "keep as is," while a



(a) The first gate layer (b) The second gate layer (c) The third gate layer

Figure 1.8: Three LSTM gate layers by [8]

0 represents "completely remove." Mathematically, this step would look like this:

$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right),$$

where W_f are the weights of forget gate layer and b_f is the the bias.

If we were to use the language model example again, the LSTM network could be deciding whether to keep information about the gender currently being spoken of or information about the subject.

2. The next step is deciding which information the network will add to the cellular state. That consists of two parts. Firstly, a sigmoid layer called the *input gate layer* decides which values will be updated. The next step is performed by the *tanh* layer, which creates a vector of new candidate values $\tilde{C}_t$. The equations for this step are:

$$i_t = \sigma\left(W_i\left[h_{t-1}, x_t\right] + b_i\right),$$

and

$$\tilde{C}_t = tanh\left(W_C\left[h_{t-1}, x_t\right] + b_C\right),$$

where W_i/b_i and W_C/b_C are weights/biases for the input gate layer and the *tanh* layer respectively.

In the language model example, the network would add new information about gender or a new subject. This step, or components active during this step, is displayed in figure 1.8b.

3. Now it is time to update the old cellular state C_{t-1} into the new cellular state C_t. This step is only the execution of the two steps described above.

The old state is multiplied by f_t, forgetting, or keeping the information we decided in step 1. Then, as described in step 2, new information $i_t\tilde{C}_t$ is added, where $\tilde{C}_t$ is the information and i_t the scale deciding the magnitude of an update for each cell state value. The equation is then:

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t.$$

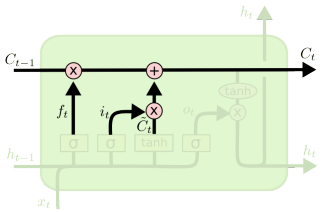


Figure 1.9: LSTM cell state operators by [8]

4. Finally, we decide what the output will be. The output is a filtered version of the cell state. It is again a combination of sigmoid values that decide which parts of information go through, and cell state values "activated" by a *tanh* layer. This step is displayed in figure 1.8c. The final two equations for this process are:

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right),$$

and

$$h_t = o_t \otimes tanh\left(C_t\right),$$

where W_o are the weights for the output gate layer and b_o is the bias.

As for the language model example, since the network just saw a subject, it might output information regarding a verb. For example, information about singularity or plurality so that we know what form of a verb should be used.

1.3.4 Gated recurrent units

Gated recurrent unit (GRU) is a simplification of LSTM-like units. It contains only two gates [13]:

- an **update gate** z_t, whose role is similar to the LSTM forget gate, and
- a **reset gate** r_t , whose role is somewhat similar to the input gate.

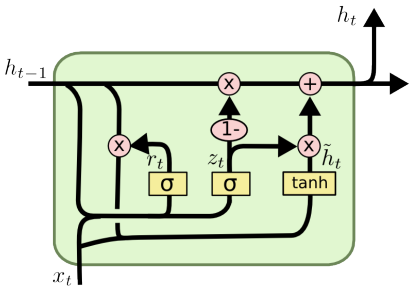


Figure 1.10: GRU unit by [8]

Other than removing the output gate from LSTM, GRU also removed the cellular state C_t. The schema of this unit is displayed in figure 1.10. To represent the data flow through the GRU unit mathematically, we will start with the update gate:

$$z_t = \sigma\left(W_z\left[h_{t-1}, x_t\right] + b_z\right).$$

The reset gate is quite similar:

$$r_t = \sigma\left(W_r\left[h_{t-1}, x_t\right] + b_r\right).$$

As for the computation of the preliminary output $\tilde{h_t}$:

$$\tilde{h_t} = tanh\left(W_h\left[r_t \otimes h_{t-1}, x_t\right] + b_h\right).$$

Finally, the output:

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h_t}.$$

Evaluations found that when LSTM and GRU have the same amount of parameters, GRU slightly outperforms LSTM. [13]

Chapter 2

Automatic speech recognition

Automatic speech recognition (ASR) has been undergoing active research for more than fifty years. It is an essential milestone in both human-human and human-machine communication. Due to the insufficient performance of technologies in the past, ASR has not become a desirable part of human-machine communication. It was because the lack of computing power did not allow passing the usability bar for real users, and other means of communication, such as keyboard and mouse, significantly outperformed speech in most aspects of communication efficiency with computers. [14]

This all changed in recent years. Speech technologies started to change how we live and work and, for some devices, became the primary means of interaction with them. By [14], there are several key areas of which progress allowed this trend:

- the first area is *Moore's law*. The law states that approximately every two years, the number of transistors in a dense integrated circuit doubles roughly every two years. [15] That leads to the computational capabilities of CPU/GPU clusters also being doubled every two years. That makes training of more complex and powerful models possible, and thus the error rates of ASR systems lower.

- The second area is the access to more data due to the continued advance of the Internet and cloud computing. By using and training models on big data collections, it is possible to build much more robust and assumption-less models than before.

- The third is that mobile, wearable, and intelligent living room devices and in-vehicle infotainment [16] became quite popular. Since the use of alternative interaction means, such as keyboard and mouse, is mostly not possible in these cases, speech communication, which is natural for humans, becomes more convenient.

There are several approaches to the ASR with different models, such as Gaussian mixture models or hidden Markov models. Since this thesis deals with neural networks, we are going to go through deep neural network (DNN) models in ASR.

Aside from diverse models, there are several ways to preprocess the raw audio sound. In the following sections, I am going to introduce some of the methods of preprocessing.

2.1 Introduction

Both in human and electronic communication, the speech information is encoded in the form of a continuously varying (analog) waveform that can be transmitted, recorded, manipulated, and ultimately decoded by a human listener. The primary analog form of the message is an acoustic waveform, which we call the *speech signal*. Those can be converted to an electrical waveform by a microphone, further manipulated by analog and digital signal processing, and then converted back to acoustic form by a loudspeaker or another electronic device.

Before we can apply any (digital) processing techniques, we must convert to the acoustic waveform, analog signal, to a sequence of numbers – digital signal. System or tool able to do such converting is called A-to-D converter, which creates digital representation by sampling the waveform in a very high rate, applies filters to preserve a prescribed bandwidth, and then reduces the sampling rate to the desired sampling rate. This discrete-time representation is the starting point for most digital signal processing applications.

Several areas that work with the digital representation of the speech signal follow. [17]

Speech coding

Perhaps the most widespread applications of digital speech are the A-to-D and D-to-A converting mentioned briefly above. It is commonly referred to as *speech coding* or *speech compression*, and its goal is to compress the digital waveform representation of speech into a lower bit-rate representation. The D-to-A decoder part of the system is often called the *synthesizer* because it must reconstruct the speech waveform from the discrete digital data.

Text-to-Speech

A Text-to-Speech analysis is another area that uses the digital signal. The goal of these systems is to start with text and automatically produce speech. The input is an ordinary text such as a newspaper article or an e-mail message. The system is depicted in figure 2.1. The first block named *Linguistic Rules*

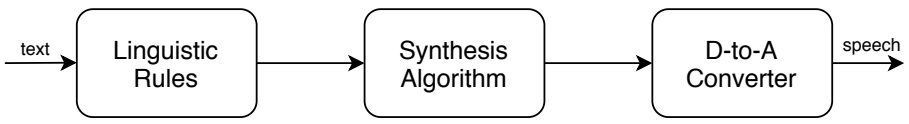


Figure 2.1: Text-to-Speech synthesis system inspired by [17]

has the job of converting the ordinary text into a set of sounds, which must be synthesized afterward. The system should include a set of linguistic rules that must determine appropriate sounds such as emphasis, pauses, or rates

of speaking. Moreover, pronounce acronyms and ambiguous words like *read, bass, or object*, how to pronounce abbreviations like *St.* (street or Saint?), *Dr.* (Doctor or drive?), and adequately pronounce specialized terms, and names. After the system builds the pronunciation set of sounds, it is time to synthesize the speech – to create the appropriate sound sequence to represent the text message in the form of speech.

Text-to-Speech synthesis systems are used to do things like to provide voice output from GPS systems, handle call center help desks, or providing information from handheld devices such as foreign language phrasebooks and dictionaries. They are also being used in announcement machines that provide information – stock quotes and airline schedules. Finally, perhaps the most important application is in reading machines for the blind, where an optical character recognition system provides the text input.

Speech recognition

Quite the opposite of the Text-to-Speech problem described above is the *speech recognition* or *Speech-to-Text* problem. It is concerned with the extraction of information from the speech signal. Figure 2.2 shows a diagram of a generic ap-

Figure 2.2: Generic pattern matching system inspired by [17]

proach to pattern matching problems in ASR. There are several sub-problems in this class, such as

- **speech recognition**, where the goal is to extract the message from the speech signal,
- **speaker recognition**, where the goal is to identify the speaker,
- **speaker verification**, which verifies a speakers claimed identity from analysis of their speech signal,
- **word spotting**, which involves monitoring a speech signal for the occurrence of a specified word or phrase, and
- **automatic indexing** of speech recordings based on the recognition of spoken keywords.

The first block in the pattern matching system diagram 2.2 converts the analog speech waveform to the digital signal using an A-to-D converter. The *feature analysis* module, the second block, converts the sampled speech signal to a

set of feature vectors. The third and final block in the system called *pattern matching* block dynamically time aligns the set of feature vectors representing the speech signal with a concatenated set of stored patterns. Then it chooses the identity associated with the pattern, which is the closest match to the time-aligned set of feature vectors of the speech signal. The symbolic output consists of a set of recognized words in the case of speech recognition. In the case of speaker recognition, it would be the identity of the best matching talker.

Probably the most common use of speech recognition is in portable communication devices. Spoken name speech recognition in cell phones enables voice dialing, and today basically every smart phone has its own voice assistant that is capable of multitude functions including speech-to-text when writing messages, the device options management, or information retrieval from the internet. [18]

The long-time dream of speech researchers are *automatic language translation* systems. The goal of such systems is to convert spoken words in one language to spoken words in another language to facilitate natural language voice dialogues between people speaking different languages.

2.2 Feature extraction

In this section, we are going to introduce two popular methods of speech signal feature extraction – namely, the Mel Frequency Cepstral Coefficients (MFCC) and the Wavelet approach.

The MFCC approach is computing the short-term *power spectrum*, which is based on an *inverse Fourier transform* of a log *power spectrum* on a nonlinear *mel scale* of frequency. [19] There are several definitions of the cepstrum, and they sometimes deviate from the steps used in this thesis. The most notable difference could be the use of a discrete cosine transform (DCT) or the lack of a mel scale filter bank. [20]

The wavelet approach substitutes the Fourier transform in MFCC by the *wavelet transform*. This transform does not expect a stationary signal and is well localized both in time and frequency using a *multiresolution* approach. The following section introduces both Fourier and wavelet transforms, their differences, and their advantages.

2.2.1 Wavelet vs. Fourier transform

Let us start by stating that the Fourier transform can be viewed as a particular case of wavelet transforms. They are both expanding the input signal onto a set of basis functions and transforms that will give an efficient and informative description of the signal.

We are going to start by defining generic transforms and pinpointing the main differences between wavelet and Fourier transforms. After that, we will build an intuitive idea around the transforms to better understands what is hiding behind the mathematics. This section draws from [21, 22].

A generic discrete linear decomposition of a signal $f(t)$ can be expressed as:

$$f(t) = \sum_{l} a_l \psi_l(t),$$

where a_l are the real-valued expansion coefficients, and $\psi_l(t)$ is a set of real-valued functions of t called the expansion set. If the expansion set is unique, it is called a *basis* for the class of functions that can be so expressed.

This is how a discrete Fourier transform looks like:

$$\hat{f}_n = \sum_{k=0}^{N-1} f_k e^{-\frac{2\pi i n k}{N}},$$

with its inverse counterpart defined as:

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} \hat{f}_n e^{\frac{2\pi i n k}{N}}$$

For a Fourier transform, the basis functions are sine and cosine functions – generally the complex exponential, which we can see in the definitions above.

The problem with the Fourier transform is that it expects a stationary signal in time. This presumption of a stationary signal comes from the basis this transform uses – sine and cosine. Those are periodic, infinite functions, and any linear combination of such functions creates, again, a periodic, infinite one. In other words, this transformation discards all the information about time but greatly captures the frequency information. We can say that the Fourier transform has great *frequency localization* and weak (none) *time localization*. That, however, does not go along with the properties of human speech.

Human speech is very far from being stationary. To be able to recognize words, syllables, and phonemes, we need to have both time and frequency localization. The first approach to tackle this problem is the *Gabor* transform named after Dennis Gabor.

The Gabor transform

Gabor transform is a special case of short-time Fourier transform (STFT). STFT splits the signal into short segments of equal length and computes the Fourier transform for each segment separately. Gabor transform is additionally using a Gaussian window function to separate the desired part of the signal from the rest, squishing undesired parts of the signal (close) to zero.

MFCC approach to the feature extraction is using a very similar process with the signal splitting and windowing. The whole process is described in section 2.2.2.

As hinted above, the Gabor transform is still not ideal. Increasing the window size lowers the time resolution for better frequency information. Smaller window, on the other hand, increases the time resolution, but we lose the low-frequency content available in longer time intervals. How to avoid such trade-off and keep both frequency and time resolution as high as possible? Let us first introduce mother wavelets.

A mother wavelet

A *mother wavelet* is a fundamental parametric function defined as:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)$$

This is a generic description of a wavelet, without a specific wavelet function given. It has two real parameters:

- parameter a, which is a non-zero scaling parameter, and
- parameter b – a translation parameter.

Now, we have a function that is both scalable and translatable in time. In other words, we have got a sliding window with dynamic size. Now, the only

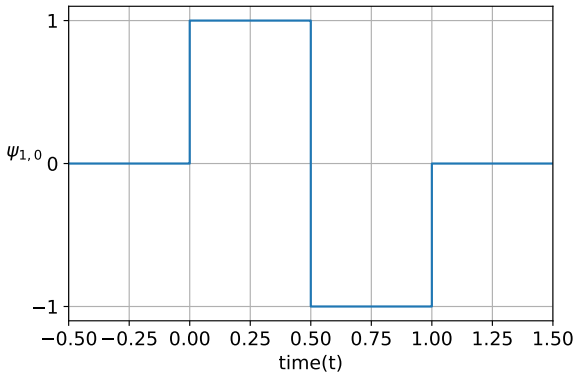


Figure 2.3: Haar wavelet

thing that remains is to define the wavelet function itself. There is a wide variety of wavelet functions that can be defined. Depending on the type of problem, we choose the one with the best properties for us. In the end, the wavelet is simply another expansion basis for the signal representation.

The Haar wavelet

The first recognized wavelet function was the Haar wavelet in 1910, meaning the idea of wavelets is more than a century old. It can be described as:

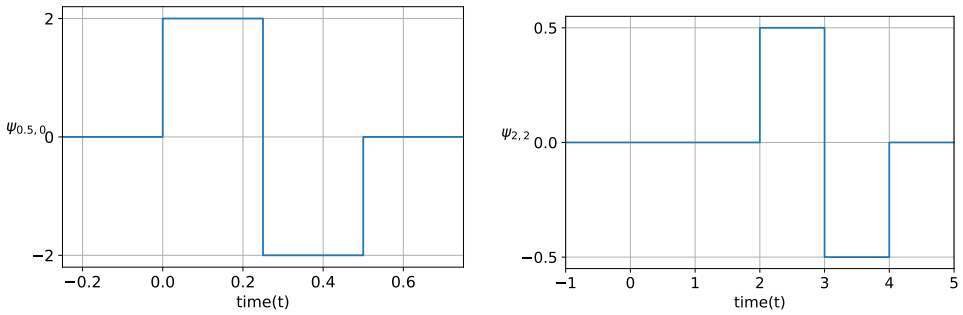
$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2} \\ -1 & \frac{1}{2} \leq t < 1 \\ 0 & else \end{cases} \tag{2.1}$$

Figure 2.3 shows the Haar wavelet step function. It is an ideal wavelet for describing localized functions because is has *compact support*. Compact support means that a function returns values on a defined interval and returns zero outside the interval. That leads to Haar function having a strong time localization. However, its frequency localization is weak, because it decays like a sine function in powers of $\frac{1}{\omega}$ in the frequency domain (Fourier transform of the wavelet). This wavelet has two more properties, that are important for us. The sum of the "area under the curve" equals zero:

$$\int_{-\infty}^{\infty} \psi(t)dt = 0, \tag{2.2}$$

and the area of the squared absolute value of the curve equals one:

$$\int_{-\infty}^{\infty} |\psi(t)|^2 \, dt = 1. \tag{2.3}$$



(a) Compressed (b) Dilated and translated

Figure 2.4: Haar wavelet with different parameters

The algorithm

To be precise, figure 2.3, and the wavelet description 2.1 are both the wavelet $\psi_{1,0}(t)$. Meaning its translation is zero, and its scaling is unity. Figure 2.4 shows other parameter configurations for Haar wavelets. Plot 2.4a depicts wavelet $\psi_{\frac{1}{2},0}$, which is a compressed function without translation, and plot 2.4b depicts $\psi_{2,2}$, which is a dilated wavelet with positive translation.

That allow for large scale structures in time in the signal to be captured by wide time-domain Haar wavelets. At this broad scale, the time resolution is pretty bad. However by rescaling the wavelet in time, we get a finer and finer time resolution along with high frequency information. Thus all the information at the low and high scales is preserved so that a complete reconstruction of the original signal can made. In the end, the only limit in this process is the number of rescaling levels.

2.2.2 MFCC vectors

The goal of this section is to describe the transformation process of the digital signal into a sequence of acoustic feature vectors. Each of the vectors represents the information in a small time window of the signal. This section draws from [23]. Most of the figures in this section are my own, created with help from [24]. The original wave file, which is the whole process of feature extraction demonstrated on, is my recording saying “yellow dandelion.”

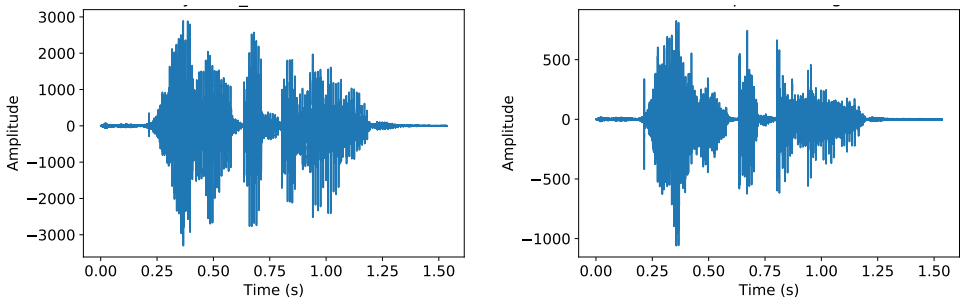
Preemphasis

The first step in this process is to boost the amount of energy in the high frequencies. If we look at the frequency spectrum of voiced segments like vowels, there is more energy at the lower frequencies than the higher frequencies. This drop of energy is called the *spectral tilt*.

This preemphasis is done by applying a filter which goes as

$$y_n = x_n - \alpha x_{n-1},$$

where signal x and parameter $0.9 \leq \alpha \leq 1.0$ are the inputs and y is the signal after preemphasis. Figure 2.5 shows a digital signal before and after the preemphasis is applied, where the value of α is set to 0.97.



(a) Before (b) After

Figure 2.5: Digital signal before and after the preemphasis

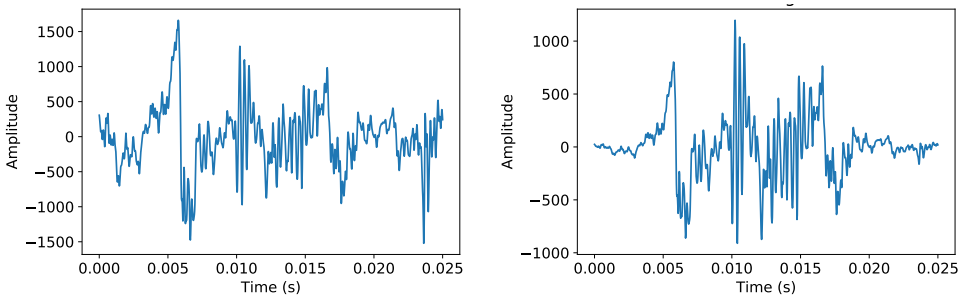
Windowing

Since the goal of this feature extraction is to get spectral features that help us build a phone or sub-phone classifier, we do not want to extract our features from an entire recording of a sentence or whole conversation. In such long pieces, the spectrum changes very quickly. Technically, speech in a *non-stationary* signal, meaning that its statistical properties are not constant across time. Therefore, we need to extract spectral features from a small *window* of speech that characterizes a single sub-phone. For this window, we can make a rough assumption that the signal is stationary (i.e., its statistical properties are constant within the window).

The windowing process takes three parameters:

- the **width** of the window (in milliseconds),
- the **offset** between successive windows, and
- the **shape** of the window.

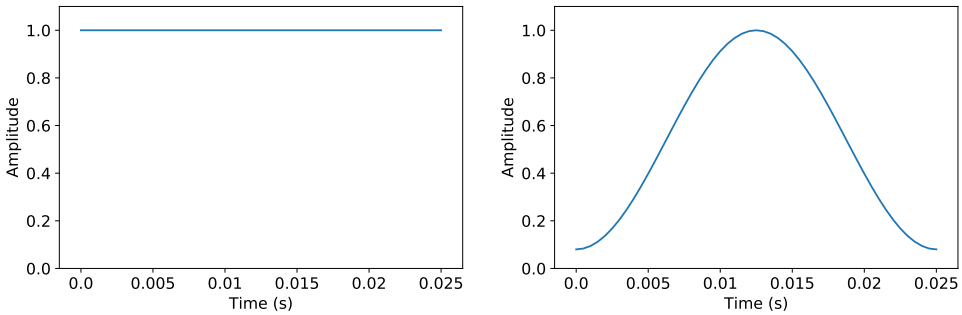
Each piece of speech extracted by the window function is called a *frame*, and its length in milliseconds the *frame size*. A number of milliseconds between the left edges of two consecutive frames (their overlap) is *frame shift*. Figure 2.6 shows the 40th frame of the original signal before and after the application of the Hamming function. We can see that the length of each frame is 25 ms.



(a) Before (b) After

Figure 2.6: A frame before and after the Hamming window function application

The third parameter, the shape of the window, can provide additional transformation to the frame. The most basic type of window is the *rectangular* window, which keeps the signal as is. That can cause problems, however, because it abruptly cuts the signal at the window edges. Such discontinuities create issues when we use the Fourier transform. That is the reason a more typical window used in MFCC extraction is the *Hamming* window. It shrinks the values of the signal near the boundaries toward zero, avoiding discontinuities. Figure 2.7 shows both the rectangular window function and the Hamming window function. We can see how are the left and right edges being squished close to the zero when the function is applied in the case of the Hamming window.



(a) Rectangular window (b) Hamming window

Figure 2.7: Window function plots

Discrete Fourier Transform

The next step is extracting the spectral information from our windowed signal – we need to know the magnitude of energy the signal contains at different frequency bands. The tool used to extract spectral information for discrete

frequency bands for a discrete-time (sampled) signal is the *Discrete Fourier Transform* (DFT).

The input for the DFT is signal $x_m ... x_n$, and the output is a complex number X_k which represents the magnitude of that frequency in the original signal (frame). Plotting the magnitude against the frequency visualizes the *spectrum*. Figure 2.8 shows such spectrum computed for the same frame, on which was the hamming window function demonstrated.

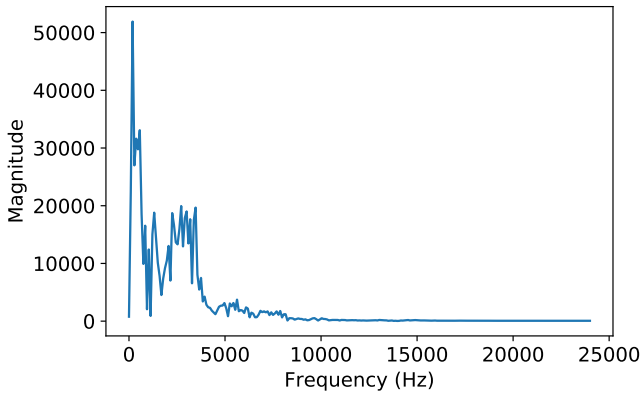


Figure 2.8: A frame spectrum computed with DFT

We are not going to go into detail of the DFT in this thesis; mere definition should be sufficient: TODOTODOTODOTODOTODOTODOTODOTODOTODOTODOTODOTODOTOD

Popular algorithm for the computation of the DFT is the Fast Fourier Transform (FFT). This implementation is very efficient with on constraint – N must equal to values which are powers of two.

Mel filter banks

This part of the extraction process tries to simulate the human hearing, which is not equally sensitive at all frequencies. It is, in fact, less sensitive at higher frequencies. Modeling this property of human hearing in the feature extraction process turned out to improve the speech recognition performance. It is done by warping the output from DFT onto the *mel* scale. A *mel* is a unit of pitch defined so that an equal number of mels separates a pair of sounds that are perceptually equidistant in tone. In essence, two pairs of adjacent notes in different octaves would have different distances in hertz, yet the same distance in mels. Mels can be computed from raw frequency as follows:

$$mel(f) = 1127ln\left(1 + \frac{f}{700}\right),$$

and therefore computing frequency from mels:

$$f(m) = 700\left(e^{\frac{m}{1127}} - 1\right).$$

There are ten filters spaced linearly below 1000 Hz, and the remaining filters logarithmically above 1000 Hz. Figure 2.9 depicts such a bank of triangular filters.

In figure 2.10, there is visualized a spectrogram of the original signal. It is 152 feature vectors (one vector for each frame) of length 26 (number of triangular filters in the bank). The spectrogram is normalized by mean normalization to make the image clearer.

Cepstrum

It would be possible to use the mel spectrum by itself as a feature representation for phone detection. Yet, there is still a lot of unimportant information in the so-far extracted features. One way to further refine the features is the computation of the so-called *cepstrum*.

The cepstrum can be thought of as a *spectrum of the log spectrum*. We take the computed spectrum and apply a logarithmic function to it. Now, we visualize the logarithmic spectrum as if it were an ordinary waveform – its domain is still frequency, but we pretend the domain is time. And the last step is computing the spectrum of this pseudo-waveform we have just created. Generally, we only take the first 12 cepstral features, that solely represent the information of the human vocal tract without unnecessary information. Figure 2.11 visualizes the extracted cepstral features from the original signal.

The formal definition for the cepstrum is *inverse DFT of the log magnitude of the DFT of a signal*. Mathematically:

$$c_n = \sum_{n=0}^{N-1} log \left(\left| \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi}{N} kn} \right| \right) e^{\frac{i2\pi}{N} kn}$$

It turns out that cepstral coefficients have the handy property that the variance of the different coefficients at different frequency tends to be uncorrelated. That is not true for the spectrum. Spectral coefficients at various frequency bands are correlated. That significantly eases the job for the acoustic model.

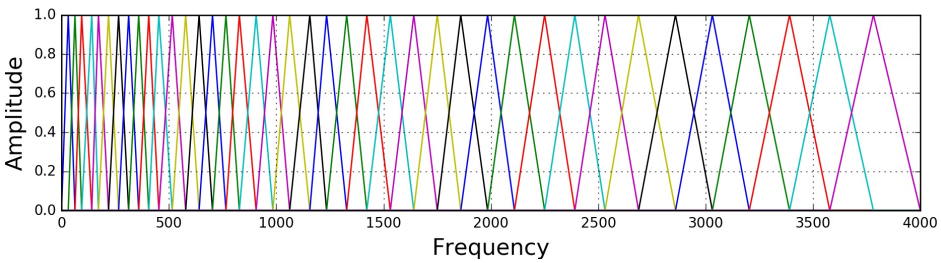


Figure 2.9: Mel filter banks by [24]

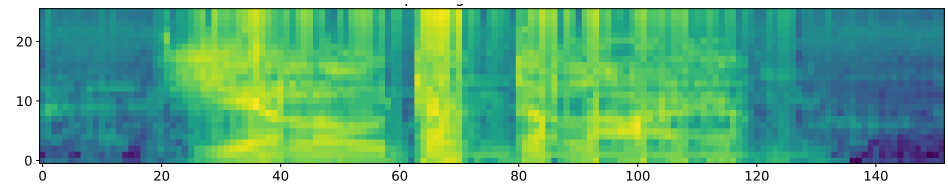


Figure 2.10: Spectrogram of the original signal

Deltas and energy

We have got 12 features extracted from each frame. Now, we next add the thirteenth feature: the energy from the frame. The energy correlates with phone identity and therefore is useful for phone detection (vowels and sibilants have more energy than stops, etc.). It is the power sum over time in the frame. Thus, for a signal x in a window from time t_1 to time t_2:

$$E = \sum_{t=t_1}^{t_2} x_t^2.$$

In the next step, since the speech signal is not constant from frame to frame, we are adding features related to the change in cepstral features over time. We do this by adding for each of the current 13 features a delta or velocity feature, and a double delta or acceleration feature.

Each of the 13 delta features represents the change between frames in the corresponding cepstral/energy feature. Each of the 13 double delta features represents the change between frames in the corresponding delta features. A simple way to compute the deltas is by calculating the difference between frames. Thus, the delta value d_t for a particular cepstral value c_t at time t is:

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

Usually, a much more sophisticated estimate of the slope, which uses a broader context of frames, is used.

Summary

Now, when we have all the features computed, we are at 39 MFCC features for each frame. In table 2.1, we can see the summary of the features with counts for each category.

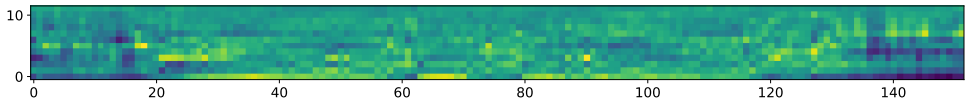


Figure 2.11: Cepstrum features of the original signal

Again, the most useful fact about MFCC features is that the features are uncorrelated, which turns out to make the acoustic model much simpler.

2.3 Wavelets

Table 2.1: MFCC features summary

12	cepstral coefficients
12	delta cepstral coefficients
12	double delta cepstral coefficients
1	energy coefficient
1	delta energy coefficient
1	double delta energy coefficient
39	MFCC features

Test [25, 26]

2.4 Linguistics

2.4.1 Language models

Chapter 3

State-of-the-art in speech recognition

Chapter 4

Analysis and design

Chapter 5

Realization

Conclusion

Bibliography

[1] Haykin, S. *Neural Networks and Learning Machines (3rd Edition)*. Pearson Education India, 2010, ISBN 0131471392.

[2] Biga, L. M.; Dawson, S.; et al. Anatomy & Physiology, Nervous tissue. Online; accessed 18-February-2020. Available from: https://open.oregonstate.education/aandp/chapter/12-2-nervous-tissue/

[3] Kužela, O. *Neural networks with memory*. Master's thesis, CTU in Prague, FIT, Thákurova 9, May 2016.

[4] Patel, N. Data Mining – Artificial Neural Networks. Online; accessed 25-February-2020. Available from: https://ocw.mit.edu/courses/sloan-school-of-management/15-062-data-mining-spring-2003/lecture-notes/NeuralNet2002.pdf

[5] Pal, S. K.; Mitra, S. Multilayer perceptron, fuzzy sets, classifiaction. 1992.

[6] Brilliant.org. Backpropagation. Online; accessed 10-March-2020. Available from: https://brilliant.org/wiki/backpropagation/

[7] Venkatachalam, M. Recurrent neural networks. Online; accessed 16-March-2020. Available from: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[8] Olah, C. Understanding LSTM Networks. Online; accessed 18-March-2020. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[9] learn neural networks. Recurrent neural networks. Online; accessed 16-March-2020. Available from: https://learn-neural-networks.com/recurrent-neural-networks/

[10] Shorten, C. Introduction to ResNets. Online; accessed 17-March-2020. Available from: https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4

[11] Wang, C.-F. The Vanishing Gradient Problem. Online; accessed 17-March-2020. Available from: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484

[12] learn neural networks. Long-short-term memory (LSTM) networks. Online; accessed 17-March-2020. Available from: https://learn-neural-networks.com/lstm-networks/

[13] Zhou, G.-B.; Wu, J.; et al. Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, volume 13, no. 3, 2016: pp. 226–234.

[14] Yu, D.; Deng, L. *AUTOMATIC SPEECH RECOGNITION*. Springer, 2016.

[15] Schaller, R. R. Moore's law: past, present and future. *IEEE spectrum*, volume 34, no. 6, 1997: pp. 52–59.

[16] Dictionary.com. Definition of Infotainment. Online; accessed 25-March-2020. Available from: https://www.dictionary.com/browse/infotainment

[17] Rabiner, L. R.; Schafer, R. W.; et al. Introduction to digital speech processing. *Foundations and Trends® in Signal Processing*, volume 1, no. 1–2, 2007: pp. 1–194.

[18] Hoy, M. B. Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical reference services quarterly*, volume 37, no. 1, 2018: pp. 81–88.

[19] Oppenheim, A. V. Speech analysis-synthesis system based on homomorphic filtering. *The Journal of the Acoustical Society of America*, volume 45, no. 2, 1969: pp. 458–465.

[20] Davis, S.; Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, volume 28, no. 4, 1980: pp. 357–366.

[21] Burrus, C.; Gopinath, R.; et al. Introduction to Wavelets and Wavelet Transform—A Primer. *Recherche*, volume 67, 01 1998.

[22] Kutz, N. Time Frequency Analysis & Wavelets. Available from: https://www.youtube.com/watch?v=ViZYXxuxUKA

[23] Pearson Education, Inc. *Chapter 9, Automatic Speech Recognition*. Pearson Prentice Hall, 2008, available from: http://www.cs.columbia.edu/~julia/courses/CS6998-2019/[09]AutomaticSpeechRecognition.pdf.

[24] Fayek, H. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. 2016, online; accessed 6-April-2020. Available from: https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[25] Hannun, A.; Case, C.; et al. Deep Speech: Scaling up end-to-end speech recognition. 2014, 1412.5567.

[26] Abadi, M.; Agarwal, A.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available from: https://www.tensorflow.org/

Appendix A

Acronyms

GUI Graphical user interface

XML Extensible markup language

NN Neural network

MLP Multilayer perceptron

ANN Artificial neural network

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format