Master Thesis

Optimization of Neural Network

Author:
Martin Bulín, MSc.

Supervisor:
Ing. Luboš Šmídl, Ph.D.

A thesis submitted in fulfillment of the requirements for the degree of Engineer (Ing.)

in the

Department of Cybernetics

May 18, 2017

Declaration of Authorship

I, Martin Bulín, MSc., declare that this thesis titled, "Optimization of Neural Network" and the work presented in it are my own. The main methods follow on my work presented in (Bulín, 2016), however, the workload of this thesis is different.

I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

"*Look deep into nature, and then you will understand everything better.*"

A. Einstein

UNIVERSITY OF WEST BOHEMIA

Abstract

Faculty of Applied Sciences

Department of Cybernetics

Engineer (Ing.)

Optimization of Neural Network

by Martin Bulín, MSc.

Neural networks can be trained to work well for particular tasks, but hardly ever we know why they work so well. Due to the complicated architectures and an enormous number of parameters we usually have well-working black-boxes and it is hard if not impossible to make targeted changes in a trained model. In this thesis, we focus on network optimization, specifically we make networks small and simple by removing unimportant synapses, while keeping the classification accuracy of the original fully-connected networks. Based on our experience, at least 90% of the synapses are usually redundant in fully-connected networks. A pruned network consists of important parts only and therefore we can find input-output rules and make statements about individual parts of the network. To identify which synapses are unimportant a new measure is introduced. The methods are presented on six examples, where we show the ability of our pruning algorithm 1) to find a minimal network structure; 2) to select features; 3) to detect patterns among samples; 4) to partially demystify a complicated network; 5) to rapidly reduce the learning and prediction time. The network pruning algorithm is general and applicable for any classification problem.

Acknowledgements

I would like to express my gratitude to my advisor Ing. Luboš Šmídl, Ph.D. for his support of my work, valuable comments and supervision of this master thesis. I also thank him for providing the speech data.

Next, I would like to thank Dr. Tomas Kulvicius for helping me start with this topic, for sharing his ideas and willing comments whenever I asked for them.

Finally, I have to thank all members of my family for their continuous support of my studies.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
DCT	**D**iscrete **C**osine **T**ransform
DFT	**D**iscrete **F**ourier **T**ransform
GDA	**G**radient **D**escent **A**lgorithm
HMMs	**H**idden **M**arkov **M**odels
MNIST	**M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology
MFCCs	**M**el **F**requency **C**epstral **C**oefficients
MSE	**M**ean **S**quared **E**rror
NN	**N**eural **N**etwork
OBD	**O**ptimal **B**rain **D**amage
PA	**P**runing **A**lgorithm
RPE	**R**ule **P**lus **E**xception
UFI	**U**nbalanced **F**eature **I**nformation
XOR	e**X**clusive **OR**
WSF	**W**eight **S**ignificance **F**actor

Chapter 1

Introduction

The very first model of an artificial neuron dates back to 1943, when two scientists, neurophysiologist Warren S. McCulloch and mathematician Walter Pitts, tried to imitate key features of biological neurons. The highly simplified model was further elaborated, which led to the concept of a perceptron, published by Frank Rosenblatt in 1958, and over the time to teachable systems nowadays known as artificial neural networks.

Some people hold the view that deep neural networks are close to produce the highly complex behaviour similar to what humans can do and solemnly call their conducts "artificial intelligence". In my opinion, today's AI designates a bunch of methods that have nothing to do with a general intelligence, but, yes, it can produce a human-like behaviour when solving one particular problem. To give one example speaking for all, we can come with Google translator, which started to use neural networks to increase fluency and accuracy when translating longer sentences. Similarly, NNs are becoming the state-of-the-art classification method in many other domains and one must admit that the results are often fascinating.

Well, we know that NNs can be trained to work very well for several tasks, however, the problem is that hardly ever we know why they work so well. The architectures are complicated and the number of parameters is enormous. In other words, we usually have a well-working black box.

When we deal with a real (not academical) problem, one often comes to a point when his or her network works well (let's say with the accuracy of 90%), but a customer asks for an accuracy of 98% for example. Then we can either keep trying and spend months on tuning the black box mostly in a random manner, or, if we demystify what is going on inside the network, we can suggest reasonable and targeted improvements.

In this work, we focus on understanding the behaviour of feedforward neural networks classifying particular data. We do it by optimizing the structure, specifically by pruning parts of the network that are unimportant for the classification. The hypothesis is that networks are often oversized and many synapses are redundant. We also think that we can find some rules or patterns in the network if it consists of important synapses only.

This effort could possibly lead to a general knowledge of how to design networks and tailor them for the challenged problem. In effect, the dimensionality of the parameters is significantly reduced, which speeds up both learning and prediction.

1.1 State of the Art

Optimization is a term of a broad meaning. In (Orhan, 2017), they try to break symmetries in a network in order to improve its performance. They do it by adding so-called "skip connections", which can also be considered as a kind of optimization. From another point of view, the problem could rest in an optimization of the crucial network learning algorithm (GDA).

In this study we rather focus on making networks small and simple. Having the smallest model that perfectly fits the classified data has two crucial advantages: 1) good generalization; 2) good chance that we will understand how the classification works.

There are two ways of how to end up with a small model that fits the data:

1. build a network from scratch by adding single parts (neurons, synapses) until a required performance is reached;
2. train a network that is larger than necessary and then remove the parts that are not needed.

The first approach is left out for the future work, and we focus on the second course of action, which is called network *pruning*. The general approach of a pruning procedure consists of these steps:

1. choose an oversized network architecture;
2. train the network until a reasonable solution is obtained;
3. delete a part of network (usually a synapse);
4. if the classification error has not grown, go to step 2), otherwise finish.

The key question is how to identify the parts that can be deleted without an increase of the error. A good survey of published pruning methods is provided in (Reed, 1993). The author starts with hypothetical calculations of what will happen if we use a brute force and remove the elements one by one. It ends up with a complexity of $O(MW^3)$, where M is the number of samples and W is the number of network elements - slow for large networks.

The pruning methods described below take a less direct approach and they basically differ one from each other in how they identify the unimportant network parts. In (Reed, 1993), the methods are divided into two groups:

- *sensitivity calculation methods*;

  These methods estimate the sensitivity of the error function to removal of an element; the elements with the least effect can then be removed.

- *penalty-term methods.*

  These methods modify the cost function so that backpropagation based on the function drives unnecessary parameters zero and, in effect, removes them during training.

Since the cost function could include sensitivity terms, there is some overlap in these groups and as our method would better fit to the first group, we focus on three published methods based on sensitivity calculations.

Skeletonization

In (Mozer and Smolensky, 1989) they introduce a measure called relevance ρ of a synapse, which is an error when the synapse is removed minus the error when the synapse is left in place.

The value is approximated using a gating term α for each unit such that

$$o_j = f(\sum_i w_{ji} \cdot \alpha_i \cdot o_i) \tag{1.1}$$

where o_j is the activity of neuron j, w_{ji} is the weight from neuron i to neuron j and $f(\cdot)$ is the *Sigmoid* function. If $\alpha = 0$, the synapse has no influence on the network; if $\alpha = 1$, the synapse behaves normally. The relevance estimation is then given by the derivative from backpropagation

$$\hat{\rho}_i = - \left. \frac{\partial E^l}{\partial \alpha_i} \right|_{\alpha_i = 1} \tag{1.2}$$

Rather than the usual sum of squared errors, the error E^l (Eq. (1.3)) is used to measure relevance, because it works better when the error is small.

$$E^l = \sum |t_{pj} - o_{pj}| \tag{1.3}$$

The authors claim the method works well for understanding the behaviour of a network in terms of "rules", which is shown on the RPE problem and on Michalski's trains (both these examples are also presented in this study for comparison).

Optimal Brain Damage

In (LeCun, Denker, and Solla, 1990) they use this ambitious title for a study that also tries to identify the unimportant weights and remove them from a network. Their measure is called "saliency" of a weight and it is estimated by the second derivative of the error with respect to the weight.

They compute the Hessian matrix H containing elements h_{ij}.

$$h_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \tag{1.4}$$

Since H is a very large matrix, they make a simplifying assumption that the off-diagonal terms of H are zero. This leaves

$$\delta E \approx \frac{1}{2} \sum_i h_{ii} \cdot \delta \cdot w_i^2 \tag{1.5}$$

It turns out that the second derivatives h_{kk} can be calculated by a modified back-propagation rule. The "saliency" s_k of weight w_k is then

$$s_k = h_{kk} \cdot \frac{w_k^2}{2} \tag{1.6}$$

In each pruning step, weights with low saliencies are deleted. The method is tested on the MNIST dataset (LeCun and Cortes, 2010), which is also used to show the developed method in this study.

Karnin's Measure

The measure published in (Karnin, 1990) is the most similar to the one used by the developed PA. The author also used the change in weight during the pruning process to compute a measure called "sensitivity" for each synapse. The sensitivity S_{ij} of weight w_{ij} is given as

$$S_{ij} = -\frac{E(w^f) - E(w^i)}{w^f - w^i} \cdot w^f \tag{1.7}$$

where w^f is the final value of the weight after training and w^i its randomly chosen initial value. Rather than removing every weight and calculating the errors directly, the author approximates S by monitoring the sum of all the changes experienced by the weight during training. The estimated sensitivity is

$$\hat{S_{ij}} = \sum_{n=0}^{N-1} \left[\Delta w_{ij}(n)\right]^2 \frac{w_{ij}^f}{\eta \cdot (w_{ij}^f - w_{ij}^i)} \tag{1.8}$$

The Δw values are calculated by backpropagation, hence, each weight has an estimated sensitivity after training. The lowest sensitivity weights are then deleted.

Contribution of This Work

We introduce own measure for the determination of how important individual synapses are. It is believed to work equally well or better than the others, while the principle is based on a nice and simple idea. It leads to a better performance in terms of computational demands, which is discussed and compared to the other listed methods in chapter 4.

Moreover, this study also suggests some ideas of how to take the advantages of pruned networks. We show on several examples, that the developed PA is capable of: 1) finding a minimal network structure for a given classification problem; 2) detection of feature importance; 3) distinguishing important samples from less important ones; 4) basic feature selection; 5) partial demystification of complicated networks.

1.2 Master Thesis Objectives

The objectives of this study are:

1. to design a neural network framework capable of learning a general classification problem;
2. to develop a pruning proceeder equipped by a tool for dimensionality reduction after pruning;
3. to demonstrate the developed methods on appropriate examples and suggest possible applications for pruned networks;
4. to implement state-of-the-art pruning methods and compare them to the developed method.

1.3 Thesis Outline

The thesis consists of 5 chapters following the standard skeleton of scientific publications. Chapter 2 details the instruments and operations we performed.

At first, in section 2.1 we give a general description of the used classification method and highlight some important design choices and conventions (a detailed description of the established conventions is then given in appendix A1).

Then the developed network pruning algorithm is introduced in section 2.2, which also contains the recipe of how to reduce the dimensionality of weight matrices after pruning. Then we put a section called "Insight of Neural Network" containing some ideas of how to use a derived minimal structure to understand the workflow in the network. Section 2.4 is also included among the methods, because it describes the approach of how the speech data was collected.

In chapter 3, six examples are presented. Each of the examples shows the pruning algorithm from a different point of view and each basically finds a new application for it.

Chapter 4 comes with the discussion about the results. It also contains a comparison of the developed pruning method to the presented state-of-the-art methods from section 1.1. Then, ideas for the future work are suggested. The study is concluded in chapter 5.

As mentioned above, appendix A1 gives a detailed view on the mathematical notation used in section 2.1. Appendix A2 contains figures and tables that did not fit to the main text, but still can be interesting for some readers. Then, the structure of the workspace is provided in appendix A3 and the attached code is documented in appendix A4.

Chapter 2

Methods

This chapter opens a collection of recipes of how the work in this study is done. Section 2.1 describes the classification, learning and evaluation procedures. In section 2.2 the developed network pruning algorithm is introduced and also the dimensionality reduction in pruned networks is shown. Section 2.3 comes with the feature selection method using a procedure called *pathing* and, additionally, some ideas of how to use remaining weights in minimal structures are suggested. Finally, section 2.4 is devoted to the process of how the speech data was gathered.

Section 2.1 moreless specifies a generally known approach. Sections 2.2 - 2.3 are partially based on (Bulín, 2016), but the methods are further elaborated.

2.1 Classification Method

The classification in this study is performed by dense feedforward neural networks. An illustration of a general feedforward network structure is in Fig. 2.1. Note that the structure is fully connected, meaning that each neuron is connected to all neurons in the next layer.

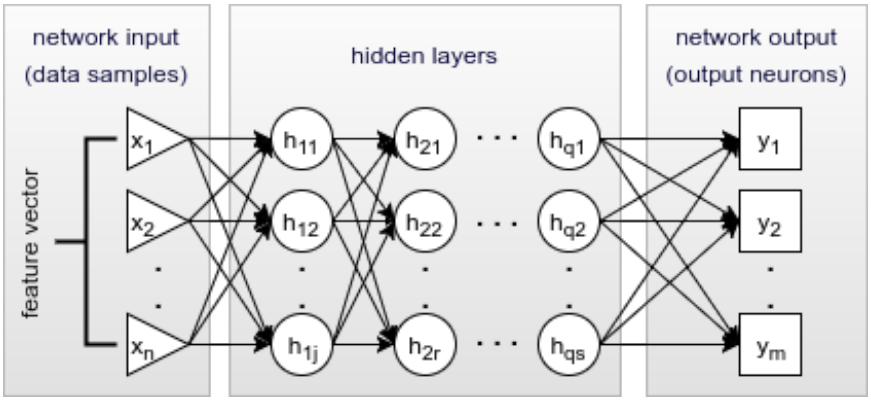


FIGURE 2.1: A general dense feedforward neural network.

The number of input units n is determined by the problem dimension. The number of classes m gives the number of output units (see the established conventions in appendix A1). In this study, we mostly use a simple hidden structure, usually with one hidden layer ($q = 1$).

Neuron Principle

The behaviour of artificial neurons follows our understanding of how biological neurons work. One unit has multiple inputs and a single output (Rosenblatt, 1958). A model of neuron is shown in Fig. 2.2. The diagram complies with the following notation:

$neuron_k^{(i)}$: k^{th} neuron in i^{th} layer

$a_k^{(i)}$: activity of k^{th} neuron in i^{th} layer

$w_{k,l}^{(i)}$: weight of synapse connecting l^{th} neuron in $(i-1)^{th}$ layer with k^{th} neuron in i^{th} layer

$b_k^{(i)}$: bias connected to k^{th} neuron in i^{th} layer

$z_k^{(i)}$: activation of k^{th} neuron in i^{th} layer

$f(\cdot)$: transfer function (Eq. (2.3); Fig. 2.3)

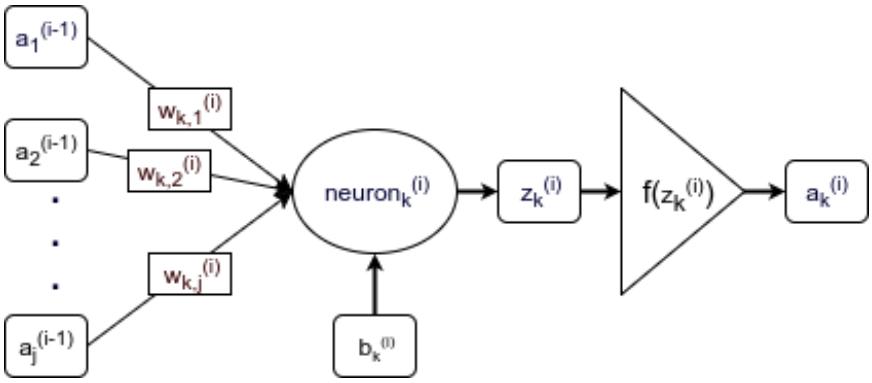


FIGURE 2.2: A model neuron

Assuming j being the number of neurons in $(i-1)^{th}$ layer, the activation of $neuron_k^{(i)}$ is computed as in Eq. (2.1)

$$z_k^{(i)} = \sum_{l=1}^{j} [a_l^{(i-1)} \cdot w_{k,l}^{(i)}] + b_k^{(i)} \tag{2.1}$$

Then we get the neuron activity by mapping its activation into a finite interval using a transfer function $f(\cdot)$ - see Eq. (2.2).

$$a_k^{(i)} = f(z_k^{(i)}) \tag{2.2}$$

The *Sigmoid* function (Eq. (2.3)) keeps neuron activities in the $\langle 0, 1 \rangle$ interval and is used by default in this work. Alternatively, one could use the hyperbolic tangent ($tanh(\cdot)$) function which maps the input into the $\langle -1, 1 \rangle$ interval.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Two basic transfer functions are shown in Fig. 2.3.

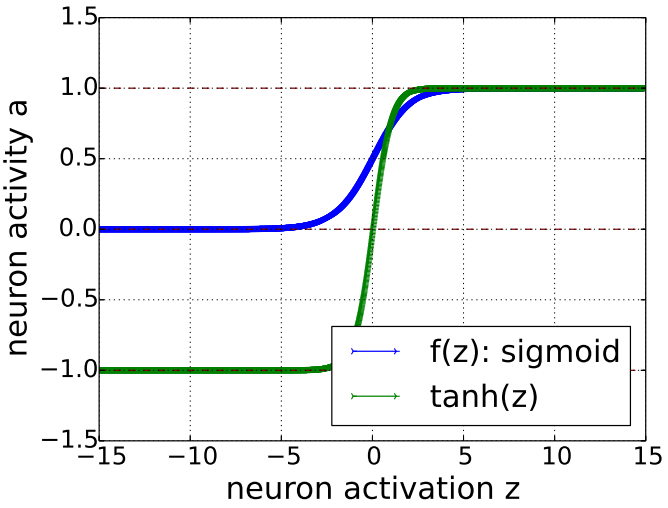


FIGURE 2.3: Transfer functions: *Sigmoid* and *Tanh*

Notation

The work of a neural network is all done by matrix calculations. Therefore we need to introduce a notation used in this study. Detailed examples of the itemized matrices can be found in appendix A1.

n : number of input neurons (problem dimension; size of one sample);

m : number of output neurons (classes);

p : number of samples;

q : number of hidden layers;

X : network input: n-by-p matrix;

$W^{(i)}$: r-by-s matrix of weights for synapses connecting s neurons in $(i-1)^{th}$ layer to r neurons in i^{th} layer;

$B^{(i)}$: vector of r biases for r neurons in i^{th} layer;

$Z^{(i)}$: r-by-p matrix of activations for r neurons in i^{th} layer for all samples;

$A^{(i)}$: r-by-p matrix of activities of r neurons in i^{th} layer for all samples;

$\Delta^{(i)}$: r-by-p matrix of errors on r neurons in i^{th} layer for all samples;

Y : predicted network output for all samples: m-by-p matrix ($Y = A^{(q)}$)

U : desired network output (targets) for all samples: m-by-p matrix

Learning Algorithm

Feedforward networks are trained by the common *Backpropagation* method illustrated by the flowchart in Fig. 2.4.

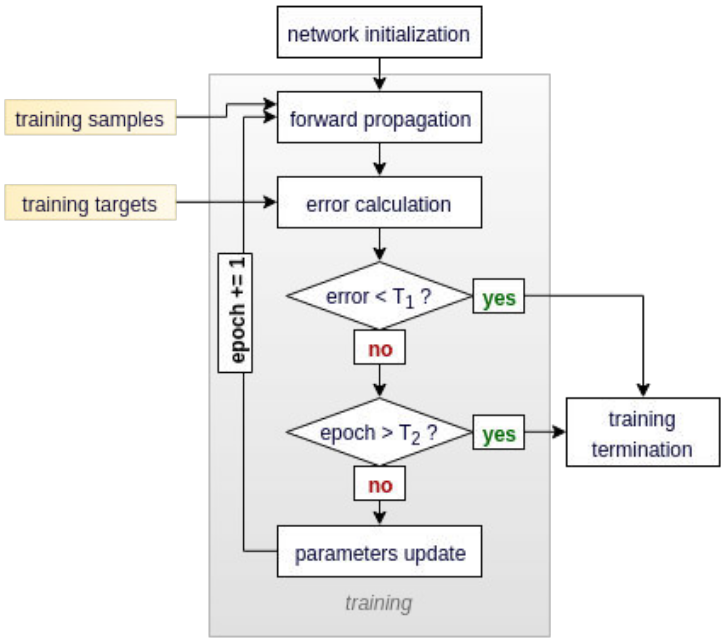


FIGURE 2.4: Training process flowchart. T_1: Threshold for a terminating condition based on the prediction error (Eq. (2.19)). If the error is reduced to be lower than T_1, the learning process is stopped and the model is considered trained. T_2: Threshold for a terminating condition based on the number of epochs. The learning process is stopped after T_2 epochs, no matter how successful the training has been.

In case of feedforward neural networks, the goal is to find optimal values for two groups of parameters - *weights* (W) and *biases* (B). The key idea lies in the optimization method called *Gradient Descent Algorithm (GDA)*.

At first, a batch of samples X is (forward) propagated through a network to get the network's prediction Y.

$$Z^{(1)} = W^{(1)} \cdot X + B^{(1)} \tag{2.4}$$

$$A^{(1)} = f(Z^{(1)}) \tag{2.5}$$

$$A^{(i)} = f(W^{(i)} \cdot A^{(i-1)} + B^{(i)}) \tag{2.6}$$

$$Y = A^{(q)} = f(W^{(q)} \cdot A^{(q-1)} + B^{(q)}) \tag{2.7}$$

Then, having the known targets (supervised learning), we compute a prediction error on every output neuron for every sample and store those errors in the m-by-p matrix E.

$$E = \frac{(U - Y) \times (U - Y)}{2} \tag{2.8}$$

Now it is time to use *GDA* to find such W and B that make E minimal. The details of the *Backpropagation* procedure are well described in (Nielsen, 2017). The envelope of the algorithm includes these steps:

1. *find the derivative of the transfer function (assuming Sigmoid)*;

$$f'(z) = f(z) \cdot (1 - f(z)) \tag{2.9}$$

2. *backpropagate the prediction error through the network;*

$$\Delta^{(q+1)} = (U - Y) \times f'[Z^{(q+1)}] \tag{2.10}$$

$$\Delta^{(i)} = \left[\left[W^{(i+1)}\right]^T \cdot \Delta^{(i+1)}\right] \times f'[Z^{(i)}] \tag{2.11}$$

3. *find the optimal parameter changes;*

   Every sample ξ has a vote $dW^{(i)}_{(\xi)}$ (resp. $dB^{(i)}_{(\xi)}$) on how the parameters $W^{(i)}$ (resp. $B^{(i)}$) should change to get the minimal error and the result is then obtained as a compromise of those votes.

   Index (i) indicates the layer. Consider $\Delta^{(i)}_{(\xi)}$ be the ξ^{th} column of the $\Delta^{(i)}$ matrix, which corresponds to the ξ^{th} sample (see appendix A1). Analogically, $A^{(i-1)}_{(\xi)}$ is the ξ^{th} column of the activation matrix $A^{(i-1)}$ in the $(i-1)^{th}$ layer. Then we get the votes as:

$$dW^{(i)}_{(\xi)} = A^{(i-1)}_{(\xi)} \cdot \left[\Delta^{(i)}_{(\xi)}\right]^T \tag{2.12}$$

$$dB^{(i)}_{(\xi)} = \Delta^{(i)}_{(\xi)} \tag{2.13}$$

4. *update the parameters.*

   At this point we introduce the first parameter of the learning process called `batch_size`. It states how many votes are processed together to make one update of the parameters. If `batch_size = 1`, we are talking about *sequential learning*. In this case, each vote is applied to update the parameters before any other votes are computed (Eq. (2.14)).

$$dW^{(i)} = dW^{(i)}_{(\xi)} \tag{2.14}$$

   In this work, we usually use *batch learning* (`batch_size > 1`).

$$dW^{(i)} = \sum_{\xi}^{batch\_size} dW^{(i)}_{(\xi)} \tag{2.15}$$

   Equations 2.14 and 2.15 work analogically for the biases.

The second parameter of the learning procedure is called `learning_rate` (μ) and is usually set $0 < \mu << 1$ in order to deal with GDA problems (see (Nielsen, 2017)). The update of the parameters is then done as follows (t refers to a moment in time):

$$W^{(i)}(t+1) = W^{(i)}(t) + \mu \cdot dW^{(i)}(t) \tag{2.16}$$

$$B^{(i)}(t+1) = B^{(i)}(t) + \mu \cdot dB^{(i)}(t) \tag{2.17}$$

When the votes of all samples are applied, the learning epoch ends. The maximal number of epochs and the maximal required error (see Fig. 2.4) are also parameters of the learning procedure. To list them all:

- `batch_size`
- `learning_rate` (μ)
- `n_epoch`
- `max_error` (MSE'; Eq. (2.19))

Network Evaluation

We use two measures to evaluate the network training: error and accuracy. The error measure is based on the standard *Mean Squared Error (MSE)* given by Eq. (2.18).

$$MSE = \frac{1}{2p}\sum_{\xi=1}^{p} ||y_\xi - u_\xi||^2, \tag{2.18}$$

where y_ξ is the prediction for sample ξ and u_ξ its corresponding (known) target. Both are vectors of length m (number of classes).

In this study we rather use the measure given by Eq. (2.19), because it makes a fair comparison of problems with a different number of classes.

$$MSE' = \frac{1}{2pm}\sum_{\xi=1}^{p}\sum_{\theta=1}^{m}(y_{\xi,\theta} - u_{\xi,\theta})^2 \tag{2.19}$$

The classification accuracy is computed with Eq. (2.20).

$$acc = \frac{1}{p}\sum_{\xi=1}^{p}\psi, \qquad \psi = \begin{cases} 1, & argmax(y_\xi) = argmax(u_\xi) \\ 0, & \text{otherwise} \end{cases} \tag{2.20}$$

The classification result is often shown by a confusion matrix (well explained in (Buitinck et al., 2013). The testing is usually done on different data samples than used for training - see Fig. 2.14.

2.2 Network Pruning

The rule of thumb in classification with feedforward neural networks is using a fully connected structure - every unit is connected to all units in the next layer.

Pruning methods work with the hypothesis that some of the synapses in fully connected networks do not contribute to the classification and so their removal would not cause a significant accuracy drop. The problem (graphically illustrated in Fig. 2.5) is to distinguish those redundant synapses from the important ones.

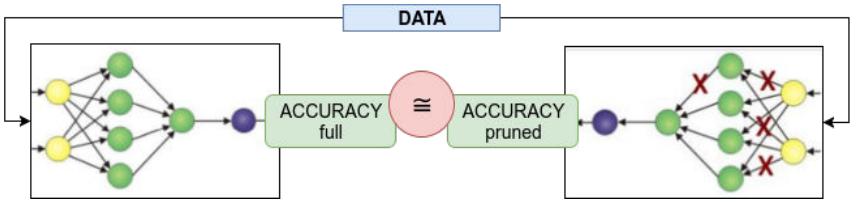


FIGURE 2.5: Pruning Algorithm: problem formulation.

There are several ways of how to estimate the importance of synapses (see section 1.1). In this study we introduce a measure called *weight significance factor* (`WSF`) given by Eq. (2.21).

$$WSF(w_{k,l}^{(i)}) = |w_{k,l}^{(i)}(t_f) - w_{k,l}^{(i)}(0)|, \tag{2.21}$$

where $w_{k,l}^{(i)}(t_f)$ is the weight of the synapse coming to k^{th} neuron in the i^{th} layer from l^{th} neuron in $(i-1)^{th}$ layer after training (t_f: time final). Analogically, $w_{k,l}^{(i)}(0)$ is the initial weight of the same synapse before training. We compare these two values and get the change in weight over the training process. The key idea is that the redundant synapses do not change their weights over the training. Therefore, those synapses with low `WSF` are considered less important than those with high `WSF`.

Realization of the Pruning Proceeder

The general recipe of how to prune a synapse is illustrated in Fig. 2.6.

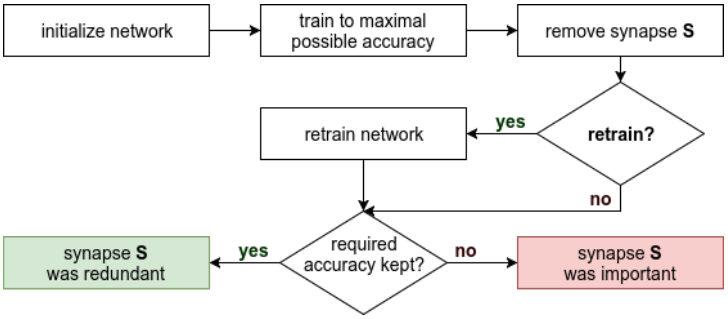


FIGURE 2.6: The flowchart of pruning one synapse, demonstration of parameter `retrain`.

Fig. 2.6 introduced the first parameter of the pruning process called `retrain`. We make it *True* if we want to retrain the pruned network (without the cut synapse) before checking the accuracy drop. In practice, we are able to set the exact number of epochs we want the net to retrain.

The required accuracy (`req_acc`) is the second parameter. In some cases we want to give up some of the accuracy in exchange for a well pruned network in order to see some patterns in it. This parameter sets the minimal accuracy the network must have so that we can treat the lastly cut synapses as unimportant. The accuracy is checked on the development data, while the training is performed on the training data (see Fig. 2.14).

So far we gave a recipe of how to prune one synapse. Of course, we cannot check all the synapses one by one using the approach in Fig. 2.6. Instead, we cut out multiple synapses at once before checking the accuracy drop. In fact, network pruning is an iterative procedure with some guessing. The first guess is the order in which we prune the synapses and the second guess is how many of them we should prune at once.

The order of the synapses is determined by the `WSF` (Eq. (2.21)) - the key idea of our pruning algorithm. Each time before pruning, all the synapses are sorted by their `WSF` values.

To estimate the number of synapses to cut at once we use percentiles - see Fig. 2.7.

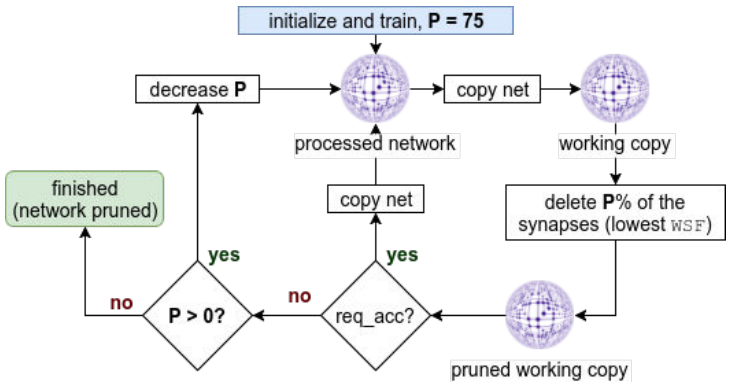


FIGURE 2.7: Network pruning proceeder.

The percentile level is set to $P = 75$ at the beginning. Hence the algorithm tries to prune 75% of the synapses (those with low `WSF`) in the first pruning step. Then the accuracy drop is checked (with or without retraining). If the accuracy dropped, we decrease the percentile level P and try to delete less synapses in the next step.

At this point we introduce another parameter of the pruning procedure called `percentile_levels`, which is an array specifying the levels we try (e.g. $percentile\_levels = (75, 50, 30, 20, 0)$). The last level in this array is always zero. When $P = 0$ we delete only the one synapse; the one with the lowest `WSF`. In this manner, at some point of the pruning process, the algorithm will come to removing only one synapse at once and if only a single synapse has been removed during the last pruning step and the accuracy has been broken, it means that even this single synapse with the least change in

weight is important for classification. Therefore, the pruning is stopped and the current net structure (including this last synapse) is saved as the minimal structure. Therefore, the algorithm is finite and it also guarantees that the classification accuracy does not drop. To sum up the parameters of the pruning procedure:

- `retrain`
- `req_acc`
- `percentile_levels`

Dimensionality Reduction

In practice, neural networks are usually represented by matrices (W and B) and the learning and prediction are performed by matrix calculations. The pruning algorithm cuts out some synapses which leads to the reduction of matrix dimensions if we restructure the matrices after pruning.

If a synapse is pruned, the corresponding weight in the W matrix takes zero value. Consider the following weight matrix $W^{(i)}$, where the i^{th} layer consists of 3 neurons and the $(i-1)^{th}$ layer has 4 neurons.

$$W^{(i)}_{full} = \begin{bmatrix} w^{(i)}_{11} & w^{(i)}_{12} & w^{(i)}_{13} & w^{(i)}_{14} \\ w^{(i)}_{21} & w^{(i)}_{22} & w^{(i)}_{23} & w^{(i)}_{24} \\ w^{(i)}_{31} & w^{(i)}_{32} & w^{(i)}_{33} & w^{(i)}_{34} \end{bmatrix} = \begin{bmatrix} -0.02 & 0.32 & -0.28 & -0.91 \\ -0.72 & 0.9 & 0.81 & 0.54 \\ 0.13 & -0.45 & 0.62 & 0.24 \end{bmatrix}$$

Now, let's assume that synapses corresponding to weights $w^{(i)}_{13}$, $w^{(i)}_{14}$, $w^{(i)}_{21}$, $w^{(i)}_{22}$, $w^{(i)}_{23}$, $w^{(i)}_{24}$, $w^{(i)}_{31}$, $w^{(i)}_{33}$ were pruned. The weight matrix $W^{(i)}$ changes to:

$$W^{(i)}_{pruned} = \begin{bmatrix} -0.02 & 0.32 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -0.45 & 0 & 0.24 \end{bmatrix}$$

We know that each row of the matrix corresponds to inputs of one neuron in the i^{th} layer and each column maps the outputs of one neuron in the $(i-1)^{th}$ layer. Therefore, if we find a row of zeros, we can safely remove the corresponding (2^{nd}) neuron from the network, as it has no inputs. Moreover, we can (better said we must) also remove the 2^{nd} column in the weight matrix $W^{(i+1)}$ responsible for the outputs of the removed neuron.

Analogically, if the 3^{rd} neuron in the $(i-1)^{th}$ layer has no outputs (3^{rd} column of zeros in $W^{(i)}$), it is useless for classification. Hence we can also delete this column in $W^{(i)}$ and must remove corresponding inputs of the removed neuron, which is the 3^{rd} row of the weight matrix $W^{(i-1)}$. We call the process *network shrinking*.

$$W^{(i)}_{shrinked} = \begin{bmatrix} -0.02 & 0.32 & 0 \\ 0 & -0.45 & 0.24 \end{bmatrix}$$

After the removal of a specified neuron, a corresponding bias is also removed. When we shrink the first hidden layer, we must also adjust the feature vectors accordingly.

2.3 Insight of Neural Network

We know that state-of-the-art fully-connected networks can be trained to work very well for several tasks, however, hardly ever we know why they work so well. Pruned networks come with some advantages that help demystify these black-box models.

Pathing in Networks

Based on the PA (section 2.2) we assume that every single synapse left in the pruned network is somehow important for classification. Therefore it makes sense to track these connections from the input to the output of a network to find out some patterns among features and classes. We define $path_f^{(c)}$ as a sorted list of synapses that connects feature f with class c (Fig. 2.8).

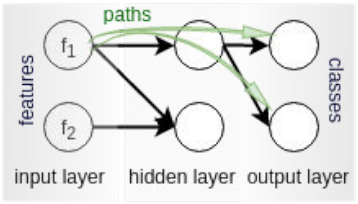


FIGURE 2.8: Explanation of a path.

If there is a path from feature f_1 to class c_1, we assume that f_1 is interesting for the class. Otherwise, if there is no path, the feature is probably not needed for class c_1 at all. This leads to a promising idea of how to do a feature selection.

Feature Energy

Moreover, we also know the weights of the remaining synapses. Hence, besides the information if or if not a feature influences a class, we can also state how big the influence is. We introduce a measure called *feature energy* (E).

$$E(f,c) = \sum_{p=1}^{P} \prod_{s=1}^{S_p} \frac{w_s^{(p)}}{|b_s^{(p)}|} \tag{2.22}$$

where $E(f,c)$ is the energy of feature f for class c, P is the number of paths from feature f to class c, S_p is the number of synapses in the p^{th} path, $w_s^{(p)}$ is the weight of the s^{th} synapse in the p^{th} path and $b_s^{(p)}$ is the bias of the neuron this synapse is connected to.

A total energy $E(f)$ of feature f for a classification problem is given as:

$$E(f) = \sum_{k=1}^{m} |E(f,c_k)| \tag{2.23}$$

2.4 Gathering of Speech Data

The presented methods are tested on several examples (chapter 3) and one of them rests in a classification of phonemes. By definition a phoneme is one of the units of sound that distinguish one word from another in a particular language (Wikipedia, 2004). We focus on Czech language and consider 40 phonemes listed in Table 2.1. This section describes the way of gathering such phonemes and building a dataset for the classification.

sound	*phoneme*	*example*	*sound*	*phoneme*	*example*	*sound*	*phoneme*	*example*
a	a	m**á**m**a**	ch	x	**ch**yba	ř	R	mo**ř**e
á	A	t**á**ta	i	i	p**i**vo	ř	Q	t**ř**i
au	Y	**au**to	í	I	v**í**no	s	s	o**s**el
b	b	**b**od	j	j	vo**j**ák	š	S	po**š**ta
c	c	o**c**el	k	k	o**k**o	t	t	o**t**ec
č	C	o**č**i	l	l	**l**oď	ť	T	ku**t**il
d	d	**d**ům	m	m	**m**ír	u	u	r**u**m
ď	D	**ď**ěti	n	n	**n**os	ú (ů)	U	r**ů**že
e	e	p**e**s	n	N	ba**n**ka	v	v	**v**lak
é	E	l**é**pe	ň	J	la**ň**	z	z	ko**z**a
eu	F	**eu**nuch	o	o	b**o**k	ž	Z	**ž**ena
f	f	**f**acka	ou	y	p**ou**to		_sil_	(silence)
g	g	**g**uma	p	p	**p**rak			
h	h	**h**ad	r	r	**r**ak			

Table 2.1: Czech phonetic alphabet.

The generation of a speech dataset consists of the following steps:

1. acquisition of real voice recordings;
2. feature extraction from the sound signals (parameterization);
3. labeling the data;
4. definition of one sample;
5. splitting samples into training/development/testing sets.

Acquisition of Voice Recordings

The phoneme dataset is made of real speech recordings from a car interior environment, provided by (*Škoda auto* 2017). We are talking about simple voice instructions for a mobile phone or a navigation system, many of them are names of people, streets or towns only. In total 14523 recordings (`.wav` files) of various length (and so number of phonemes) were obtained.

Parameterization

The goal of parameterization is to represent each recording by a vector of features. A commonly known procedure based on MFCCs is used. A nice detailed explanation of this method can be found e.g. in (Lyons, 2009).

The idea behind MFCCs originates in the fact that a shape of human vocal tract (including tongue, teeth etc.) determines what sound comes out.

The shape of the vocal tract manifests itself in the envelope of the short time power spectrum, and the job of MFCCs is to accurately represent this envelope.

The parameterization workflow is summarized by these steps:

1. *splitting the signal into short frames*;

   Fig. 2.9 illustrates how a sound signal is divided into short frames. The parameters are

   $$frame\_size = 0.032\,s = 32\,ms$$
   $$frame\_shift = 0.01\,s = 10\,ms$$

   Using the sampling frequency $f_s = 8kHz$, we get frames of length 256.

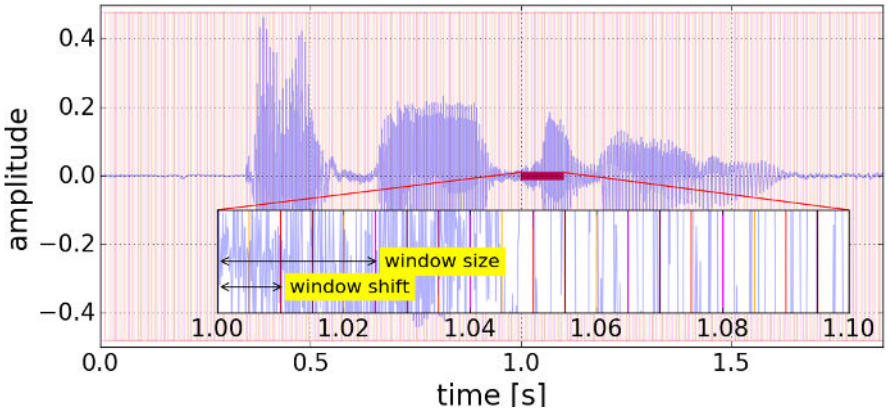


FIGURE 2.9: Framing a sound signal.

   We assume each frame captures one possible shape of the human vocal tract and therefore it is capable of carrying one phoneme only. The next steps are applied for every single frame.

2. *calculation of the periodogram estimate of the power spectrum*;

   The aim is to identify which frequencies are present. In order to do so, we apply the Hamming window and perform the discrete Fourier Transform (DFT; Eq. (2.24)).

   $$S(k) = \sum_{n=0}^{N-1} s_n \cdot e^{-2\pi i \frac{kn}{N}}, \qquad k = 0, ..., N-1, \tag{2.24}$$

   where N (in this case $N = 256$) is the signal length. Then we take the absolute value $|S(k)|$.

3. *application of the mel filterbank to the power spectra, summation of the energy in each filter, taking a logarithm of the result*;

   Next, we use a filterbank of triangle filters (illustrated in Fig. 2.10) predefined on the transmitted band ($bw = \frac{f_s}{2} = 4kHz$) to get a single value per filter. We use 40 filters, therefore, each frame is now described by a vector of 40 numbers.

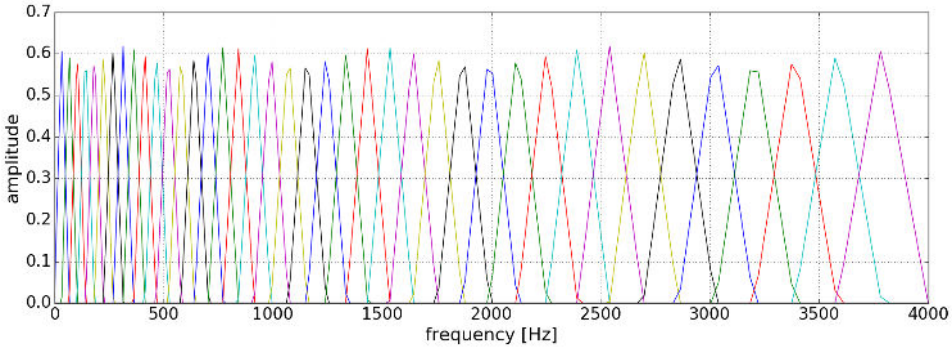


FIGURE 2.10: Mel Filterbank of 40 filters in Hertz-axis.

Finally, a logarithm of the result is taken and considered as a description of the frame (phoneme). Usually, a discrete Cosine Transform (DCT) is applied at the end, however, it is not done in this work. The result of a signal parameterization is a matrix shown in Eq. (2.25).

$$recording\_params = \begin{bmatrix} f_{11} & f_{12} & f_{13} & \cdots & f_{1F} \\ f_{21} & f_{22} & f_{23} & \cdots & f_{2F} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{W1} & f_{W2} & f_{W3} & \cdots & f_{WF} \end{bmatrix}, \tag{2.25}$$

where $F = 40$ is the number of filters and W is the number of frames (windows) depending on the duration of a recording. Value f_{12} then belongs to the feature computed with the second filter in the first frame.

Data Labeling

We perform a supervised learning method, hence the data must be labeled. To do so a speech recognition method based on Hidden Markov Models (HMMs) is used. It labels the frames of each recording as shown on an example in Table 2.2.

recording_name		
frame_in	*frame_out*	*phoneme*
0	16	_sil_
16	25	a
25	32	n
32	44	o
44	65	_sil_

TABLE 2.2: Example of a labeled recording.

It says that features extracted from this recording consist of 16 fourty-dimensional vectors representing a silence, then 9 fourty-dimensional vectors representing phoneme `"a"`, 7 vectors of phoneme `"n"`, etc.

Forming a Sample

The 40 phonemes listed in Table 2.1 are naturally labels of classes, so we have a fourty-class classification problem. Having the information from previous section, one can match the extracted features with corresponding phonemes (classes). Now the task is to define the form of one sample.

Fig. 2.11 goes with the example in Table 2.2. The numbers in the first line are frame indices. The second line contains the known frame labels, where each frame is described by a vector of 40 features.

There is a possibility to take all frames labeled as `"a"` and consider the corresponding vectors directly as samples. However, as the labeling was not done manually and therefore cannot be considered as 100% correct, we introduce a parameter called `border_size`. Fig. 2.11 shows that we omit the frames on borders with another phoneme label and take only those in the middle.

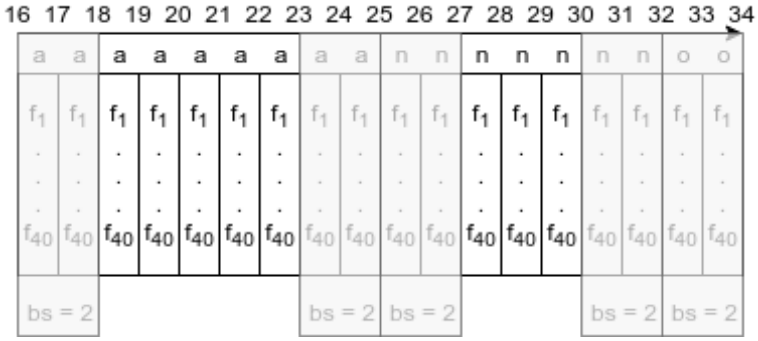


FIGURE 2.11: Forming a sample, illustration of parameter `border_size (bs)`.

Moreover, in Fig. 2.12 parameter `context_size` is introduced. The idea is to consider not only the information of one frame, but also of its context, into one sample.

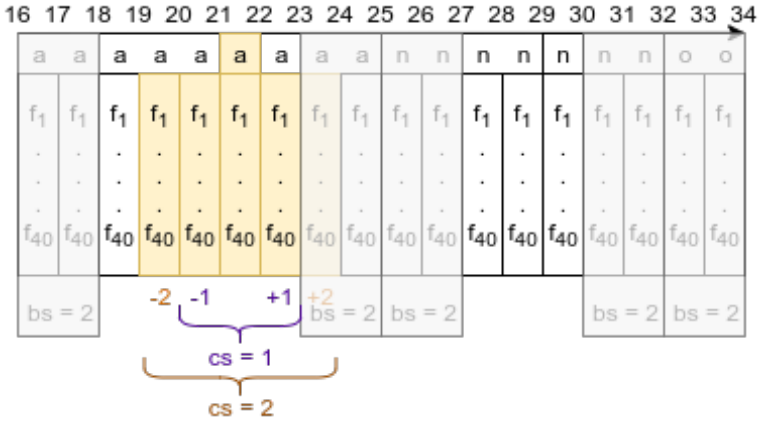


FIGURE 2.12: Forming a sample, illustration of parameter `context_size (cs)`.

Based on the chosen context size cs the previous and subsequent vectors are added one by one and form one feature vector of length $40 \cdot (2cs + 1)$. An example for $cs = 2$ is illustrated in Fig. 2.13.

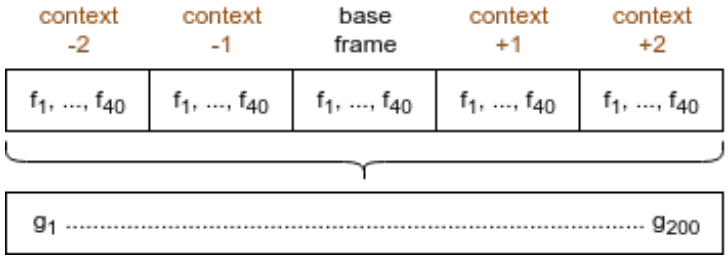


FIGURE 2.13: Example of building a feature vector with `context_size` $cs = 2$.

Talking about Fig. 2.13, features $g_1, g_2, ..., g_{200}$ give the final feature vector of one sample, which takes the label of the base frame.

The last parameter of the speech dataset generation is the number of samples per class (`n_samples`). The rule of thumb is the more samples the better training results, however, getting best possible training results is not the objective of this work. Therefore we often use less samples to speed up the training process.

To summarize this section, we end up with three parameters of the speech dataset generation process:

- `border_size` (`bs`), see Fig. 2.11
- `context_size` (`cs`), see Fig. 2.12
- `n_samples` per class (`ns`)

Splitting data into three disjunctive sets

Fig. 2.14 shows a general approach of data splitting in machine learning. It is used for all classification problems in this work. The training data is used to set up model parameters. The development data is then used for testing during the training process, in order to adjust some learning parameters based on the test results. Finally, a trained model is tested on never-seen testing data. By default, we use splitting: 80% training set; 10% development set; 10% testing set.

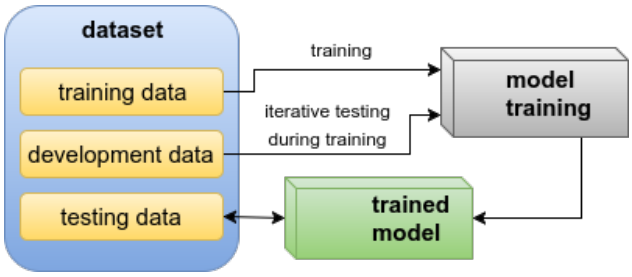


FIGURE 2.14: Using three disjunctive sets of data for a general machine learning process.

Chapter 3

Examples

The pruning algorithm is presented on several examples, where each of them has its purpose of being shown. The XOR problem (section 3.1) should verify the ability of finding an optimal network structure. Section 3.2 comes with another 2D problem, where one feature carries more information than the other one. The Rule-plus-Exception problem in section 3.3 deals with a minority of samples that has to be treated by a different net part than rule-based samples. The train problem (section 3.4) is a working example of the feature selection procedure. The MNIST database (section 3.5) is widely used in machine learning and can be regarded as commonly known, hence it is an ideal example to present new methods on. Finally, in section 3.6 the pruning algorithm is analysed on a large dataset of phonemes.

3.1 XOR Function

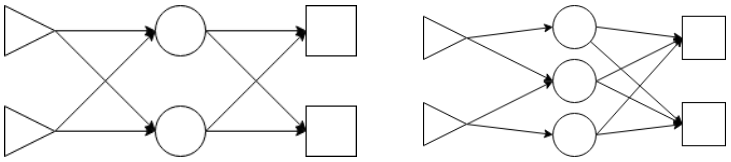
The standard Exclusive OR (XOR) function is defined by truth Table 3.1. Based on this function one can build a classification problem of two features and two classes.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

TABLE 3.1: XOR function.

This problem serves perfectly for a demonstration of network optimization methods, as two optimal architectural solutions producing the XOR function are already known (Fig. 3.1) [1].

[1]The known (e.g. from (Bradley, 2006)) minimal network architectures producing the XOR function $[2, 2, 1]$ and $[2, 3, 1]$ are adjusted to $[2, 2, 2]$ and $[2, 3, 2]$ in Fig. 3.1 in order to comply with the conventions introduced in appendix A1. The number of output neurons always equals the number of classes. The number of hidden-output synapses might not be optimized in this study.



(A) Solution A. (B) Solution B.

FIGURE 3.1: Optimal network architectures producing the XOR function.

With this knowledge we can prove that the pruning algorithm is (or is not) able to find the optimal solution. If the method is correct, it should end up with one of the shown architectures (Fig. 3.1a or Fig. 3.1b).

The truth Table 3.1 ruled the generation of a 2D dataset illustrated in Fig. 3.2. The two classes can be linearly separated by two lines (corresponding to two neurons, see Fig. 3.1a) and each class consists of 1000 samples. Each sample was randomly assigned to one of the two possible points belonging to its class (e.g. (0,0) or (1,1) for class 0) and then randomly placed in the surrounding area within a specified range ($r = \frac{\sqrt{2}}{4}$).

The samples of each class were then splitted into three sets in the following manner: 80% to a training set, 10% to a validation set and 10% to a testing set.

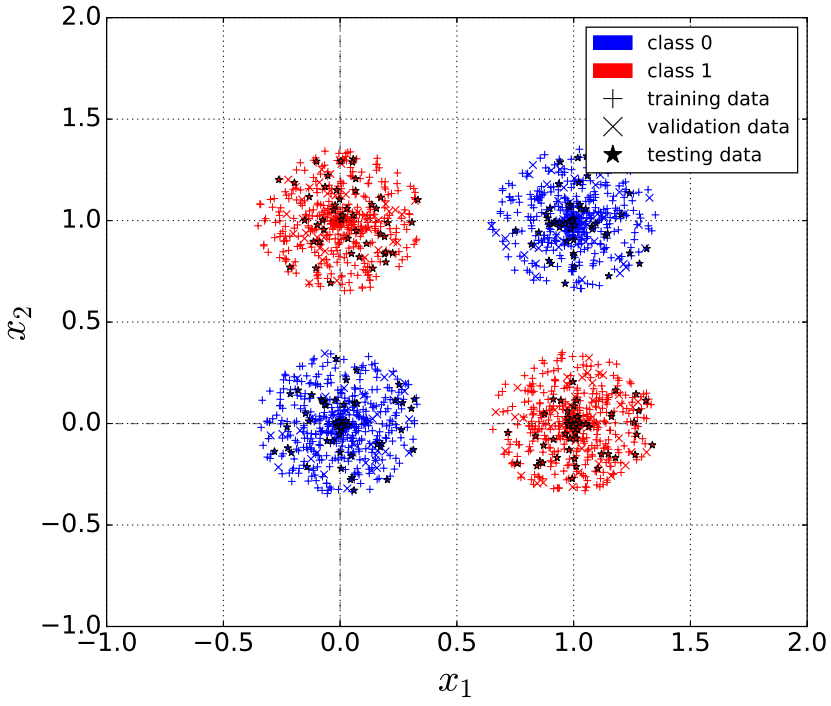


FIGURE 3.2: The XOR dataset.

The goal of this example is to show that the pruning algorithm finds one of the known minimal network structures (Fig. 3.1). An oversized network $[2, 50, 2]$ is used as the starting point. The following Table 3.2 shows all the experiment settings.

initial network		*learning parameters*		*pruning parameters*	
structure	[2, 50, 2]	learning rate	0.3	required accuracy	1.0
n synapses	200	number of epochs	50	retrain	True
transfer fcn	sigmoid	minibatch size	1	retraining epochs	50

TABLE 3.2: Experiment settings for the XOR example.

Results: XOR Function

Fig. 3.3 describes the pruning process. We can see the number of synapses, the network structure and the classification accuracy for single pruning steps. When the required accuracy (1.0) was not reached, the corresponding steps are transparent in the figure, indicating they were forgotten.

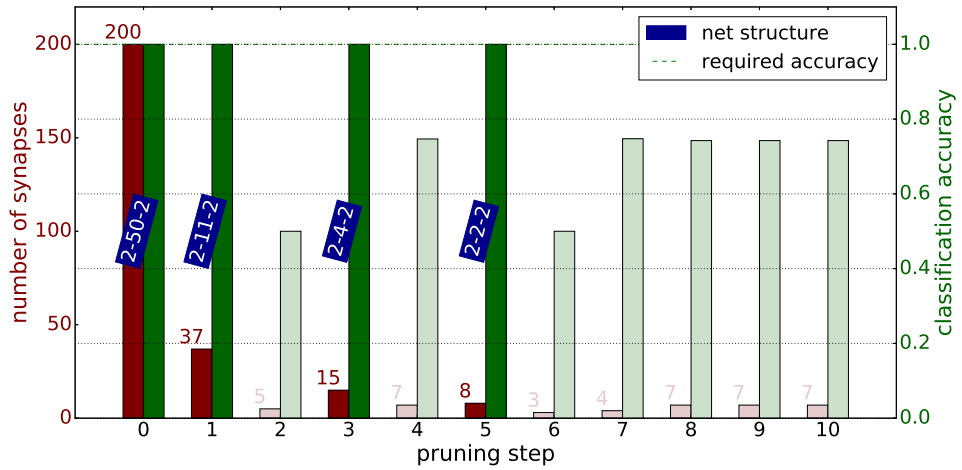
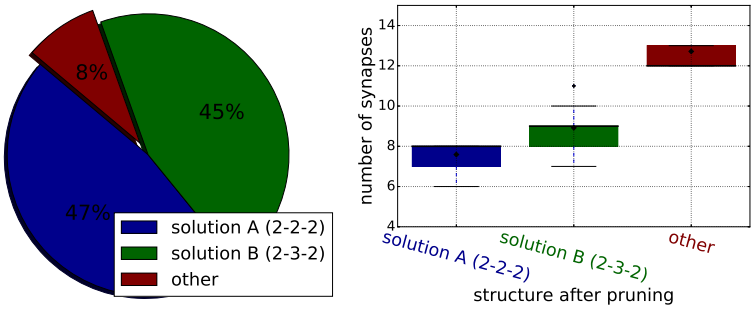


FIGURE 3.3: Illustration of the pruning procedure applied on XOR dataset (selected observation).

In Fig. 3.4 the hypothesis of this experiment is confirmed. We ran 100 observations of the experiment. In Fig. 3.4a we can see that in 47 out of 100 cases the pruning algorithm changed the network to $[2, 2, 2]$ architecture (Fig. 3.1a), in 45% of the cases it resulted with $[2, 3, 2]$ (Fig. 3.1b) and only in 8% it failed to find the optimal architecture. Fig. 3.4b gives statistics for the final number of synapses in these three cases.



(A) Network structure after pruning (100 observations).

(B) Number of synapses after pruning (100 observations).

FIGURE 3.4: Pruning results for XOR dataset.

3.2 Unbalanced Feature Information

This example is adopted from (Karnin, 1990). The problem is again two-dimensional having two non-overlapping classes as depicted in Fig. 3.5. The samples are uniformly distributed in $[-1, 1]$ x $[-1, 1]$ and the classes are equally probable, separated by two lines in 2D space ($x_1 = a$ and $x_2 = b$, where $a = 0.1$ and $b = \frac{2}{a+1} - 1 \approx 0.82$). Clearly, the problem can be solved by two neurons, similarly as the previous one.

What is interesting about this two-classes layout is that feature x_1 is much more important for the global classification accuracy than feature x_2. Having x_1 information, based on Fig. 3.5 one could potentially classify more than 90% of the samples. Opposite of that, we cannot say much with information from feature x_2 only. And this is something that also the pruning algorithm should find out.

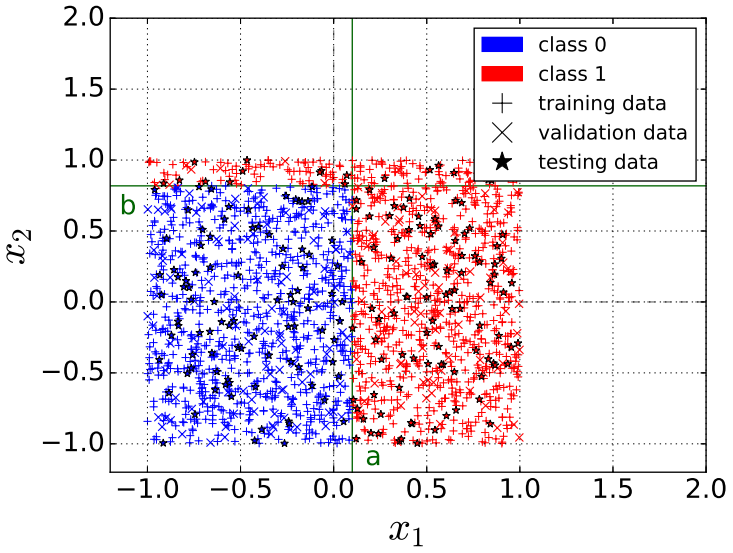


FIGURE 3.5: The UFI dataset.

Hence, we focus on synapses connecting the input and the hidden layer (shortly input-hidden synapses). We know the required network structure is $[2, 2, 2]$, as two lines are needed to separate the data in 2D space. Actually, we even know the lines must be parallel to coordinate axes, which means that each of the hidden units needs one of the features only. Therefore, the first hypothesis here is that pruning of input-hidden synapses should result in one of the cases in Fig. 3.6.

(A) Pruning result 1. (B) Pruning result 2.

FIGURE 3.6: Expected pruning of input-hidden synapses (UFI problem).

To prove this behaviour, we ran an experiment with settings in Table 3.3.

initial network		*learning parameters*		*pruning parameters*	
structure	[2, 2, 2]	learning rate	0.7	required accuracy	0.98
n synapses	8	number of epochs	50	retrain	True
transfer fcn	sigmoid	minibatch size	1	retraining epochs	50

TABLE 3.3: Experiment settings for the UFI example.

The second hypothesis is that the synapse connected to the first feature (x_1) is more important and therefore, the other synapse (the one connected to feature x_2) should always be removed first.

Results: Unbalanced Feature Information

In Fig. 3.7 the first hypothesis is confirmed. We ran the experiment 100 times. In 48 cases, the pruning of input-hidden synapses finished with the result shown in Fig. 3.6a and it finished with the result shown in Fig. 3.6b in 44% of the cases.

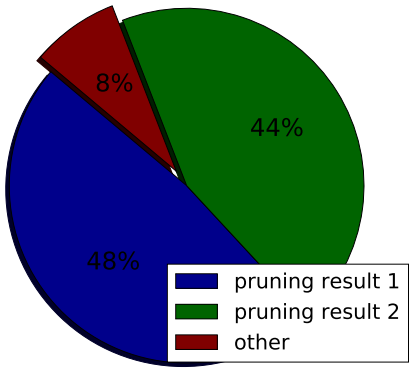


FIGURE 3.7: Results of pruning (see Fig. 3.6) input-hidden synapses (100 observations, UFI example).

In other words, with a probability of 92% the algorithm is able to find the axis-parallel lines and reveals that each of the lines needs the information from corresponding feature only. In the remaining 8% of the cases the pruning resulted with more than two input-hidden synapses.

The second hypothesis is confirmed in Fig. 3.8. The `WSF` was always (100 observations) greater for the synapse coming from feature x_1 than for the synapse connected to x_2. By definition (see section 2.2), the pruning method eliminates the synapses with low significance factors first, therefore the information coming from feature x_1 would live longer in the network than the x_2 information.

Let's try to explain this result. Consider w_{r1} to be the weight of the synapse connecting the x_1 feature and r^{th} hidden neuron (with bias b_r) and w_{s2} to be the weight of the synapse coming from feature x_2 to s^{th} hidden neuron (with bias b_s), then by neuron definition (Rosenblatt, 1958) we created two

lines, perpendicular one to each other, as follows.

$$w_{r1} \cdot x_1 + 0 \cdot x_2 + b_r = 0 \tag{3.1}$$

$$x_1 = -\frac{b_r}{w_{r1}} \tag{3.2}$$

$$0 \cdot x_1 + w_{s2} \cdot x_2 + b_s = 0 \tag{3.3}$$

$$x_2 = -\frac{b_s}{w_{s2}} \tag{3.4}$$

In Fig. 3.5 we see that $a < b$. To generalize the problem (assuming normalised feature vectors) we state $|a| < |b|$, meaning we want:

$$|-\frac{b_r}{w_{r1}}| < |-\frac{b_s}{w_{s2}}| \tag{3.5}$$

Hence we expect:

$$|w_{r1}| > |w_{s2}| \tag{3.6}$$

Out of this we expect a weight magnitude to be greater for more important synapses.

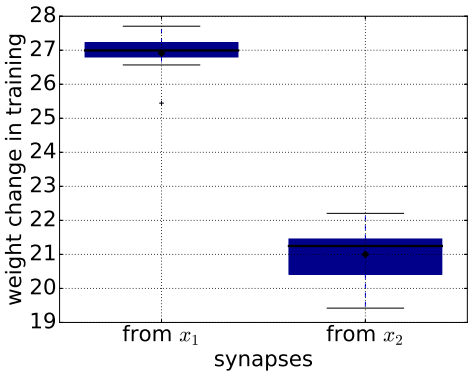


FIGURE 3.8: Weight change in training for the remaining input-hidden synapses (100 observations).

The weight magnitude seems to be a perfect measure to find feature significance factor. However, as we do not use a weight decay (see the learning approach in section 2.1), in general the more epochs we learn the greater weight magnitudes we get. Therefore small initial weight values do not affect the result significantly and so we can state:

$$|w_{ji}(t)| \approx |w_{ji}(t) - w_{ji}(0)| \tag{3.7}$$

where $w_{ji}(0) \in N(0, 1)$ is the initial value of weight w_{ji} and t is time. Summing it up we can say that the `WSF` measure based on weight change is equally good as the magnitude measure for feature selection, assuming enough training epochs (e.g. 50).

3.3 Rule-plus-Exception

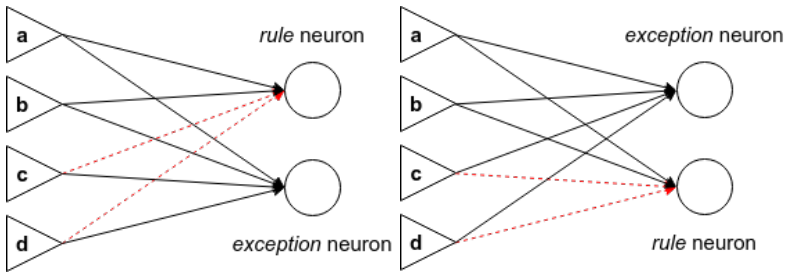
This four-dimensional problem is originally adopted from (Mozer and Smolensky, 1989) and is also used in (Karnin, 1990). The task is to learn another Boolean function: $AB + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$. A single function output should be on (i.e. equals 1) when both A and B are on, which is the *rule*, and it should also be on when the *exception* $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ occurs.

Clearly, the *rule* occurs more often than the *exception*, therefore the samples corresponding with the *rule* should be more important for the global classification accuracy. The hypothesis is that the pruning method should suggest the part of network, which deals with the *exception*, to be eliminated first - before network elements dealing with the *rule*.

To test this hypothesis, a dataset of 10000 samples was generated. Each sample consists of four features: $[a, b, c, d]$. Each of these features (for every sample) was randomly set to be *on* (1) or *off* (0). Then, whenever the *rule* occurred ($a = 1 \wedge b = 1$), the sample was assigned to class 1 (as a *rule* sample). If the *exception* occurred ($a = 0 \wedge b = 0 \wedge c = 0 \wedge d = 0$), the sample was also labeled as 1 (as an *exception* sample). Otherwise, the sample was assigned to class 0. Thereby the generated dataset consisted of:

- 2511 *rule* samples (class 1);
- 649 *exception* samples (class 1);
- 6840 samples in class 0.

The function is expected to be learned by two hidden neurons, one dealing with the *rule* and the other one with the *exception*. We focus on input-hidden synapses again. Two possible expected pruning results are shown in Fig. 3.9.



(A) Pruning result 1. (B) Pruning result 2.

FIGURE 3.9: Expected pruning of input-hidden synapses (RPE problem).

We ran the learning-pruning procedure with the settings listed in Table 3.4.

initial network		*learning parameters*		*pruning parameters*	
structure	[2, 2, 2]	learning rate	1.0	required accuracy	1.0
n synapses	8	number of epochs	50	retrain	True
transfer fcn	sigmoid	minibatch size	1	retraining epochs	50

TABLE 3.4: Experiment settings for the RPE example.

Results: Rule-plus-Exception

Fig. 3.10 supports the hypothesis that one hidden neuron forms the *rule* and the other one the *exception*. With a probability of 97% the pruning finished with one of the structures in Fig. 3.9.

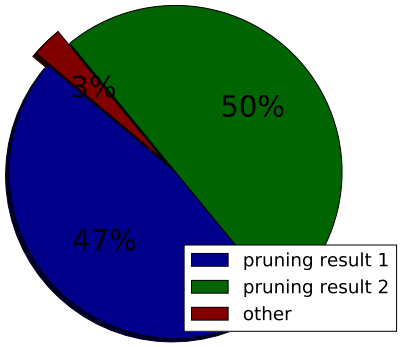


FIGURE 3.10: Results of pruning (see Fig. 3.9) input-hidden synapses (100 observations, RPE example).

The same thing is confirmed by Fig. 3.11. It shows weight change in training (`WSF`) for all 8 input-hidden synapses. We can see that synapses connecting the *rule* neuron with feature c (s_{rc}) and feature d (s_{rd}) were suggested as least important (resulted in structures in Fig. 3.9).

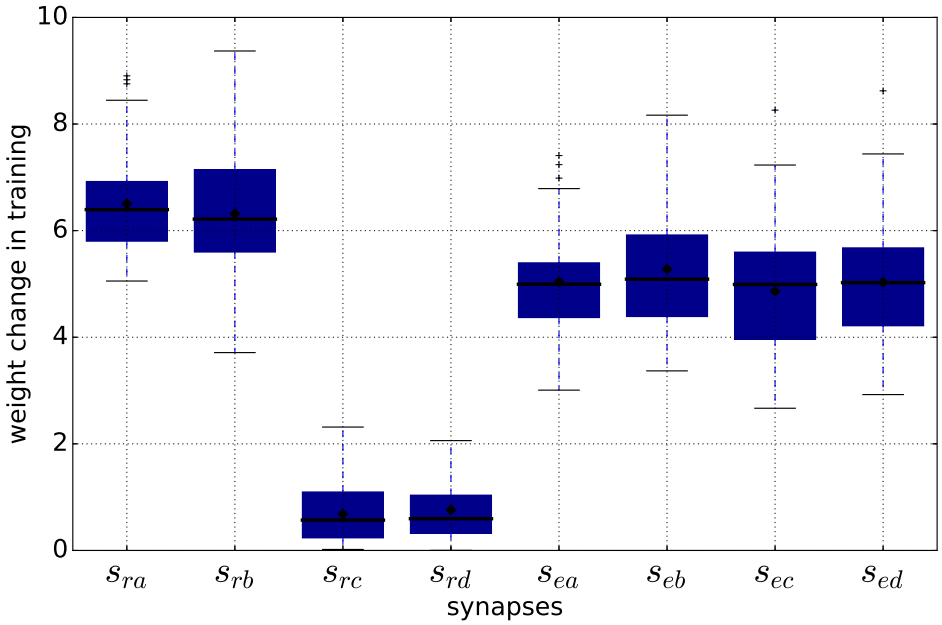


FIGURE 3.11: Weight change in training for input-hidden synapses (100 observations, RPE example).

Additionally, synapses responsible for *rule* (s_{ra} and s_{rb}) have a greater mean significance than the synapses connected with the *exception* neuron (s_{e*}).

3.4 Michalski's Trains

The train problem was originally introduced in (Larson and Michalski, 1977). The task was to determine concise decision rules distinguishing between two sets of trains (Eastbound and Westbound). In (Mozer and Smolensky, 1989), they presented a simplified version illustrated in Fig. 3.12.

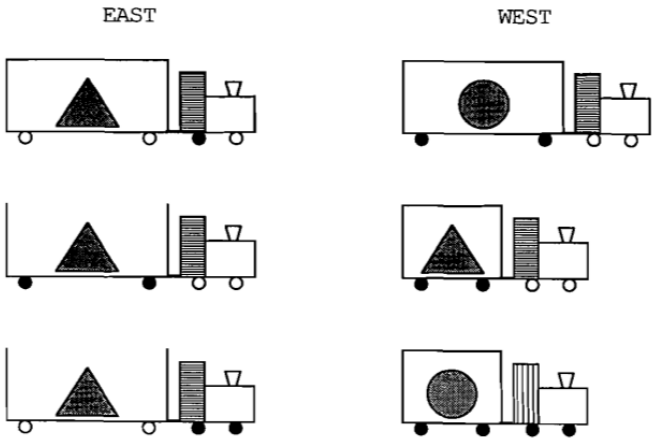


FIGURE 3.12: Michalski's train problem.

Each train is described by 7 binary features listed in Table 3.5.

feature		*encoded as* ***0***	*encoded as* ***1***
0	car length	long	short
1	car type	open	closed
2	cabin pattern	vertical lines	horizontal lines
3	load shape	triangle	circle
4	color of trailer wheels	white	black
5	color of first car wheel	white	black
6	color of second car wheel	white	black

TABLE 3.5: Features describing a train.

Having Table 3.5 we can encode the trains shown in Fig. 3.12 into feature vectors as follows in Table 3.6.

class EAST		*class WEST*	
east 1	$[0,1,1,0,0,0,1]^T$	west 1	$[0,1,1,1,1,0,0]^T$
east 2	$[0,0,1,0,1,0,0]^T$	west 2	$[1,1,1,0,1,0,0]^T$
east 3	$[0,0,1,0,0,1,1]^T$	west 3	$[1,1,0,1,1,1,1]^T$

TABLE 3.6: Feature vectors for different train types.

The task is to determine the minimal number of input features capable of the east-west classification based on the six possible types in Table 3.6 (or in Fig. 3.12).

The hypothesis is that the pruning algorithm should select the needed features by eliminating unimportant input-hidden synapses. Looking at Fig. 3.12 one of the solutions could be keeping features $(0, 3)$, because the shape of the load together with the length of the car is enough to distinguish *west* trains from *east* trains. Another solution, for example, is keeping the car length, car type and color of the second car wheel - features $(0, 1, 6)$.

To test our pruning algorithm on this feature selection task, a dataset of 6000 samples (3000 west and 3000 east trains) was generated. The three possible train types for each class (Fig. 3.12) are equally distributed among the samples, meaning we have 1000 samples of each train type.

As shown in (Mozer and Smolensky, 1989), one hidden neuron is enough to learn this problem, hence we started with the network structure $[7, 1, 2]$. The experiment parameters are listed in Table 3.7.

initial network		*learning parameters*		*pruning parameters*	
structure	[7, 1, 2]	learning rate	0.3	required accuracy	1.0
n synapses	9	number of epochs	100	retrain	True
transfer fcn	sigmoid	minibatch size	1	retraining epochs	10

TABLE 3.7: Experiment settings for the train example.

We ran 100 observations of the experiment and considered the features that were not cut out, as a result of a single experiment.

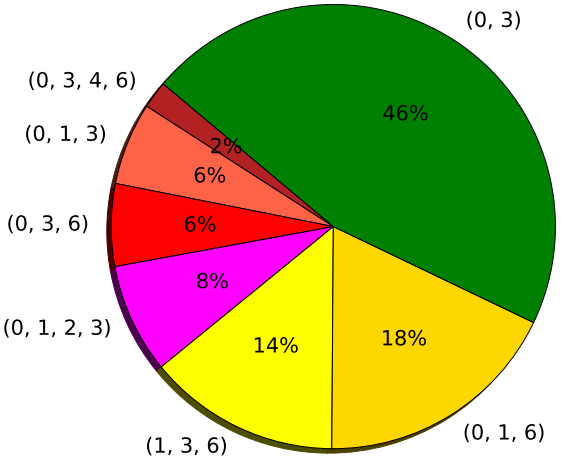


FIGURE 3.13: Results of feature selection by the pruning algorithm (train example). The labels corresponds with feature indices in Table 3.5.

The result pie in Fig. 3.13 shows that the pruning algorithm found the best possible solution ($(0, 3)$ - the car length and the load shape) in 46% of the cases. We can regard the $(0, 1, 6)$ and $(1, 3, 6)$ as another (not best but also good) solutions. The rest we consider as fail cases, as all of them include features $(0, 3)$ and the other features are redundant. To sum it up, we got a perfect solution: 46%; a good solution: 32%; a bad solution: 22%.

3.5 Handwritten Digits (MNIST)

The MNIST (Modified National Institute of Standards and Technology) database (Wikipedia, 2004) is a large database of handwritten digits that is widely used for training and testing methods in the field of machine learning.

The dataset was downloaded from (LeCun and Cortes, 2010). Some of the digits were written by employees of American Census Bureau (*United States Census Bureau* 2017) and some by students of an American high school. In total 70000 samples were collected. Examples are shown in Fig. 3.14.

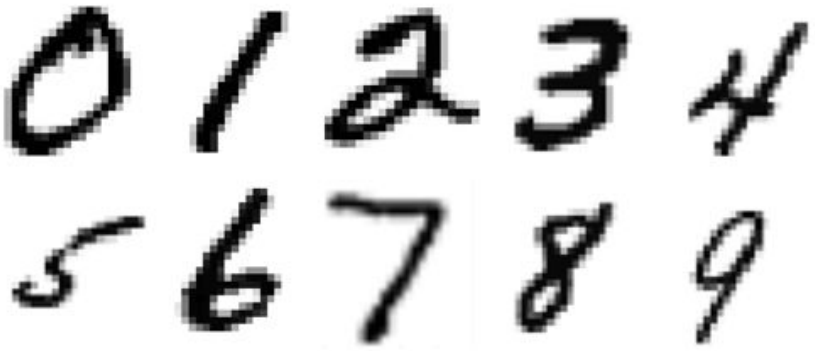


FIGURE 3.14: Examples of MNIST dataset.

Each sample is a grayscale image (normalised to $[0, 1]$) of size $28x28$ pixels. This gives row-by-row a vector of 784 features. The data was splitted into a training set of 50000 samples, a validation set of 10000 samples and a testing set of 10000 samples.

From (LeCun and Cortes, 2010) we know the problem can be learned by a feedforward network with one hidden layer up to high accuracy ($98 - 99\%$). The first task is to achieve similar results with the implemented neural net framework. We tested the following learning settings (Table 3.8).

network parameters		*learning parameters*	
structure	[784, 20, 10]	learning rate	0.3
n synapses	15880	number of epochs	100
transfer function	sigmoid	batch size	10

TABLE 3.8: Settings for training a dense feedforward net on the MNIST dataset.

The training results are summarized in Table 3.9. A confusion matrix for the testing data is given in Fig. 3.15.

	accuracy	*MSE*
training data	97.2%	0.526
testing data	94.3%	1.025

TABLE 3.9: Training results on MNIST dataset.

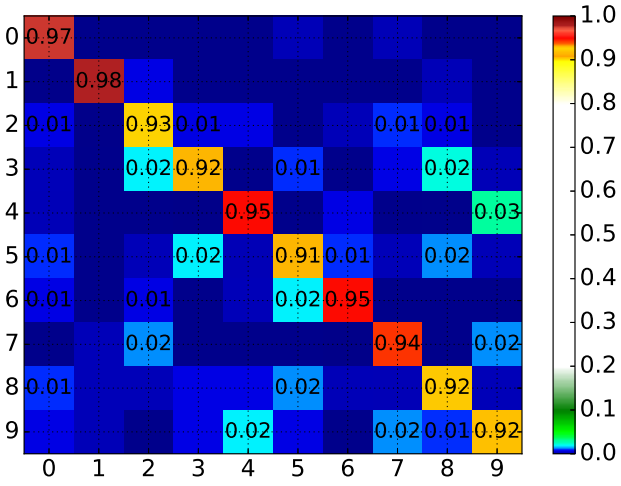


FIGURE 3.15: Confusion matrix (MNIST, testing data).

In the following, the pruning method is analysed on networks trained on the MNIST database. Parameters of the learning-pruning procedure are listed in Table 3.10.

initial network		*learning parameters*		*pruning parameters*	
structure	[784, 20, 10]	learning rate	0.3	required accuracy	0.97
n synapses	15800	number of epochs	30	retrain	True
transfer fcn	sigmoid	minibatch size	10	retraining epochs	10

TABLE 3.10: Experiment settings for the MNIST example.

The hypothesis is that the initial number of synapses in the network (15800) is redundant, as well as the number of features (784). In Fig. 3.16 we can see a selected observation of the pruning process. The number of synapses was reduced to 1259 and the number of used features to 465, while the classification accuracy was kept on 97%. The pruning procedure finished in 424 pruning steps (explained in section 2.2).

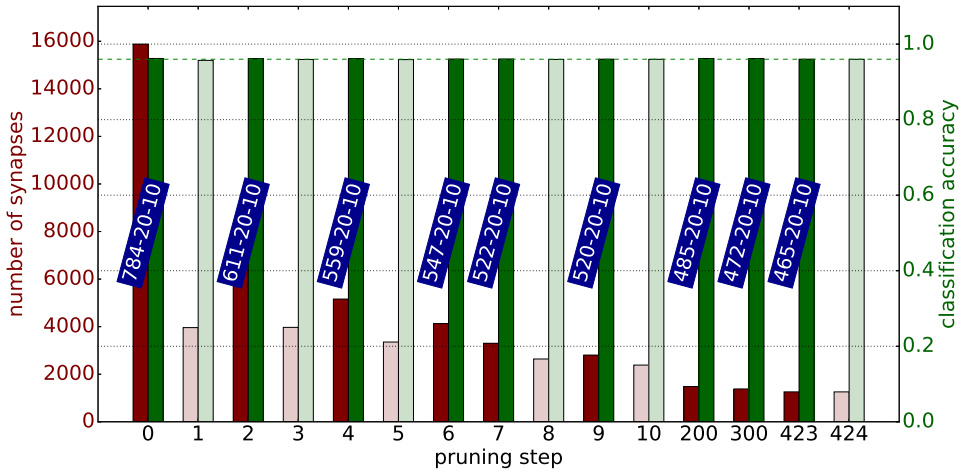


FIGURE 3.16: Illustration of the pruning procedure applied on MNIST dataset (selected observation). Required accuracy: 97%.

In Fig. 3.17, we can see a comparison of the evaluation time. We compare a fully-connected (initial) network to the pruned one. The bars are given for three data groups (training: 50000 samples, validation: 10000 samples, testing: 10000 samples). The pruning reduced the dimensions of weight matrices, which led to the reduction of processing time by nearly a half.

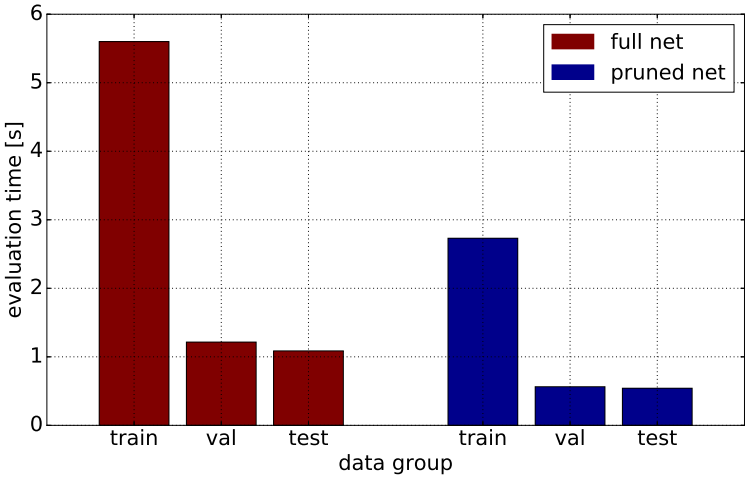


FIGURE 3.17: Evaluation (accuracy and error computation) time for all data groups (pruned vs. full net).

Fig. 3.18 gives the statistics by running 10 observations of the pruning procedure for several values of a required classification accuracy. We observed the number of synapses (red axis) and used features after pruning (blue axis).

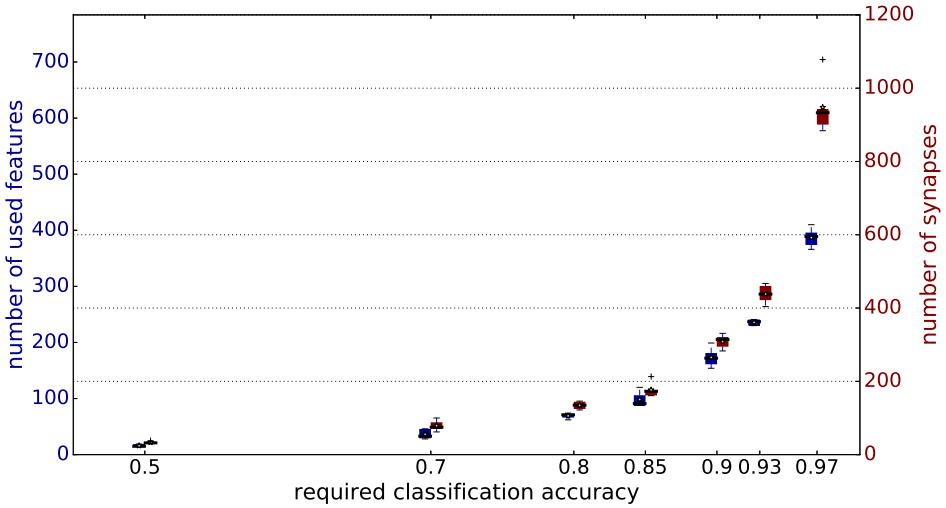


FIGURE 3.18: Minimal number of features and synapses to get required classification accuracy (MNIST data).

The results show that *less than a tenth* of the synapses and *about a half* of the features are needed to keep the maximal classification accuracy (97%). It is also worth saying that the MNIST dataset can be learned to 50% using only 20 features and a network with 38 synapses. In the following, these two results are further analysed.

Minimal MNIST network

At first, we focus on a pruned network capable of MNIST classification with accuracy of 50% (Fig. 3.19). This example is simple enough to show the feature selection method described in section 2.3.

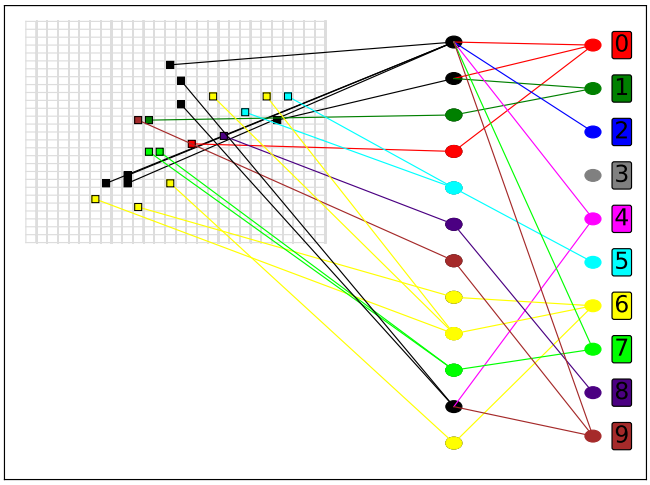


FIGURE 3.19: Result of network pruning and path tracking, MNIST data, accuracy: 50%.

Each class (digit in this case) has its color. If a hidden unit has one output connection only, it inherits the color of the class it is connected to. The features (pixels of the $28x28$ image) are then colored in the same way. If a hidden unit influences more than one class, it is blacked. All features connected to a black hidden unit are then blacked as well, as they also affect more than one class.

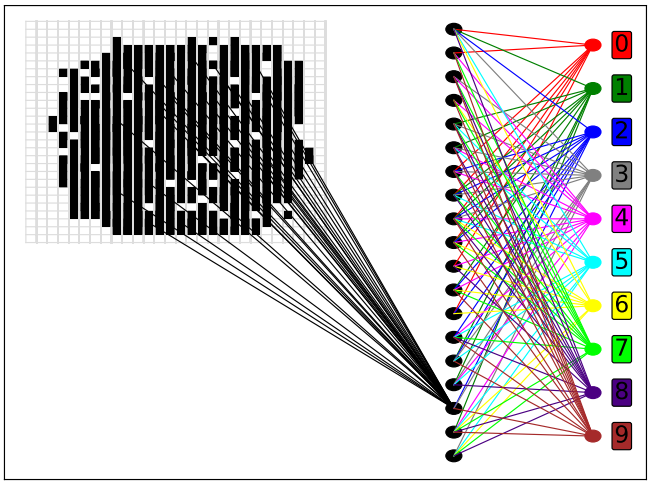


FIGURE 3.20: Result of network pruning and path tracking (shown 17^{th} hidden neuron only), MNIST data, accuracy: 97%.

A pruned network capable of 97% accurate classification is visualized in Fig. 3.20. To make the figure clearer, only the synapses coming to the 17^{th} hidden unit are drawn between the input and the hidden layer.

We can see that each of the features affects more than one class in this case. Therefore we better use the visualization in Fig. 3.21.

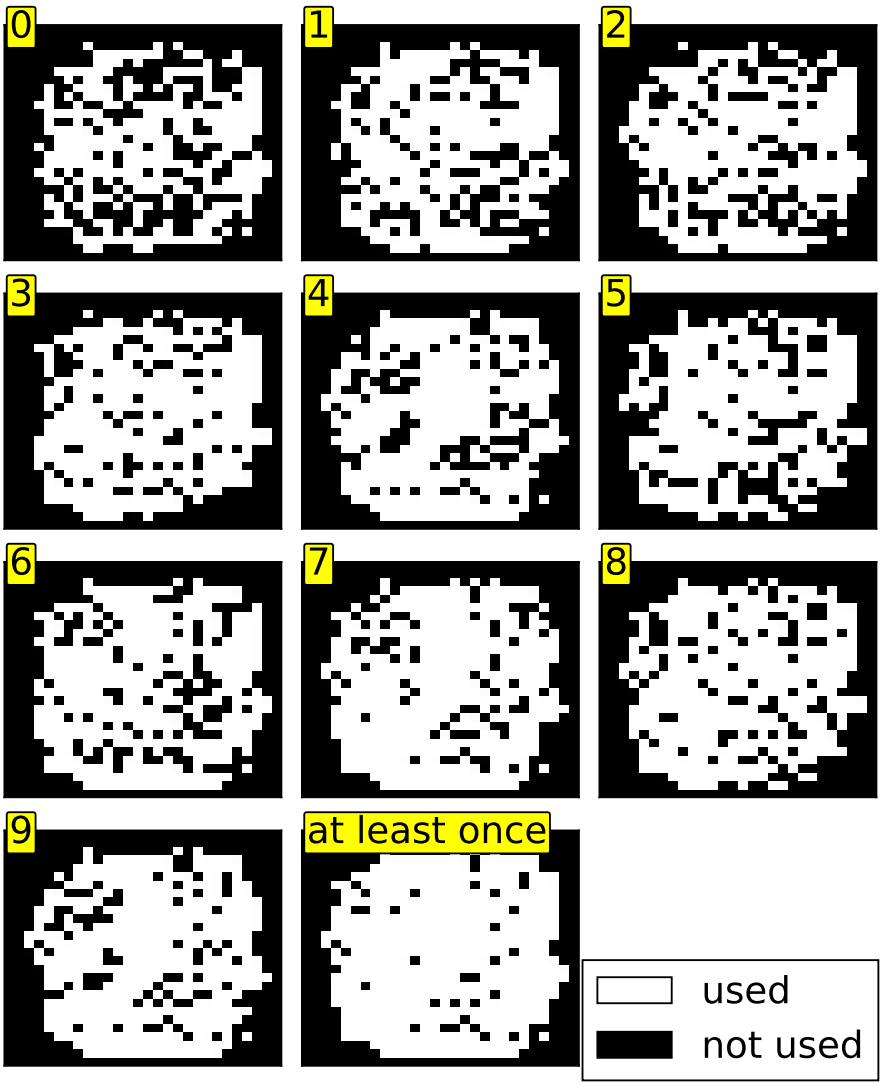


FIGURE 3.21: Used features for individual classes, MNIST data, accuracy: 97%.

Knowing all the remaining synapses are important for classification, we can track the paths from individual classes to features. This way we distinguish features connected to a selected class from those that do not affect that class. Fig. 3.21 shows important features for each class (digit) separately. Note that the *at-least-once* subplot corresponds to the features shown in Fig. 3.20. It shows all features used by at least one class.

3.6 Phonemes (Speech Data)

The process of speech data gathering is described in section 2.4. The dataset generation process has three parameters: `border_size` (bs), `context_size` (cs) and `n_samples` (ns).

In this example, we first try to find optimal parameter settings, which would lead to a maximal trainability. The general rule is the more samples the better trainability, therefore we fix $ns = 1000$ and determine the other parameters at first. See Table A2.1 for details of all generated datasets differing in bs and cs. It reveals that phoneme `"F"` does not have enough occurrences (less than $ns = 1000$) in the data, and of course the number of occurrences decreases with growing bs. For $bs >= 6$ even more phonemes (`"D"`, `"F"`,`"N"`, `"Q"`, `"R"`, `"T"`) have less than 1000 occurrences.

Table 3.11 shows the experiment settings.

experiment settings		learning parameters	
n observations	5	learning rate	0.1
observed value	MSE' (Eq. (2.19))	n epochs	50
network structure	$[40 \cdot (2cs + 1), 50, 40]$	batch size	10

TABLE 3.11: Speech dataset: experiment settings for determination of optimal bs and cs.

We ran 5 observations of a simple network training for every combination of $bs \in [0, 5]$ and $cs \in [0, 9]$. Fig. 3.22 shows average MSE' (see Eq. (2.19)) values.

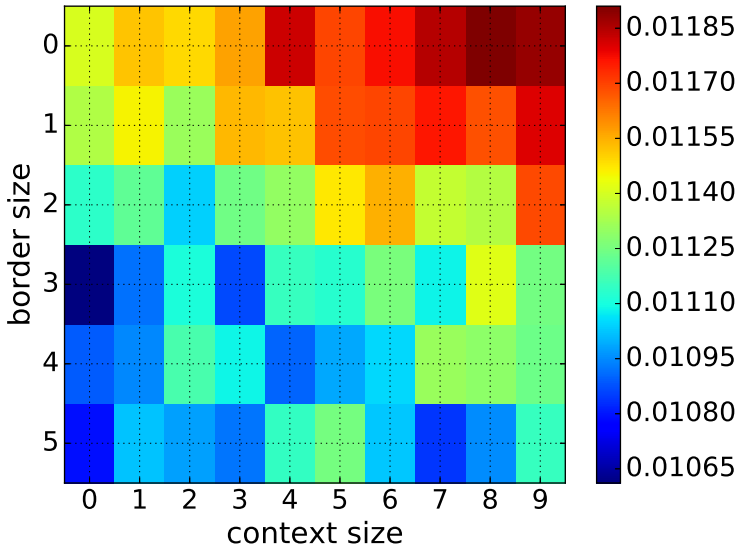


FIGURE 3.22: Test MSE' (Eq. (2.19)) for various parameters bs and cs ($ns = 1000$, 5 observations, see Table A2.1).

The experiment result says that a bigger `context_size` ($cs > 2$) does not go well together with a low `border_size` ($bs \leq 2$). We can state that the `border_size` should better be greater than two. Then the `context_size`

does not influence the trainability much. We must keep in mind that the experiment is too simple to give a reliable estimation (simple network structure, few training epochs), however, it gives an initial idea of a general trend, which is enough for the purposes of this work. For the experiments below we consider these settings:

- `border_size`: $bs = 3$
- `context_size`: $cs = 3$
- `n_samples`: $ns = 10000$

Analysis of the Generated Speech Dataset

The goal of this section is to show what kind of data we actually work with. In Fig. 3.23 we can see one randomly selected sample for each phoneme. For a more illustrative view the context is cut out ($cs = 0$), hence we see 40 features corresponding to 40 frequency filters (see Fig. 2.10).

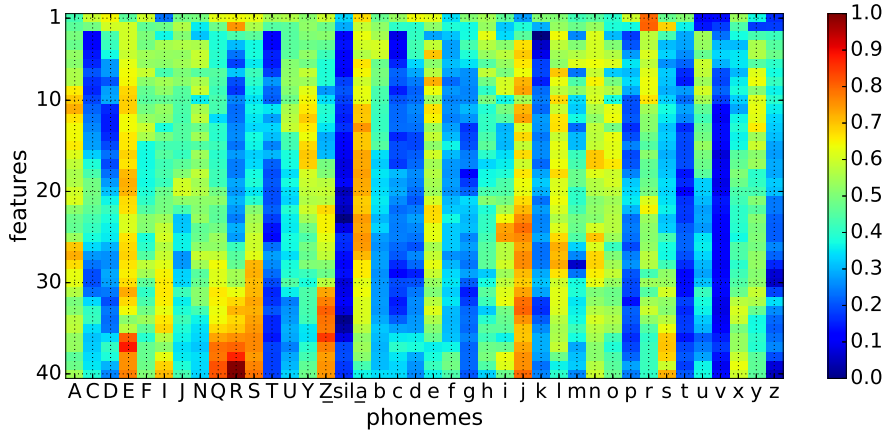


FIGURE 3.23: Randomly selected sample for each phoneme, $cs = 0$.

Fig. 3.24 shows an average feature vector out of 10000 samples for each phoneme. Here we can find some patterns.

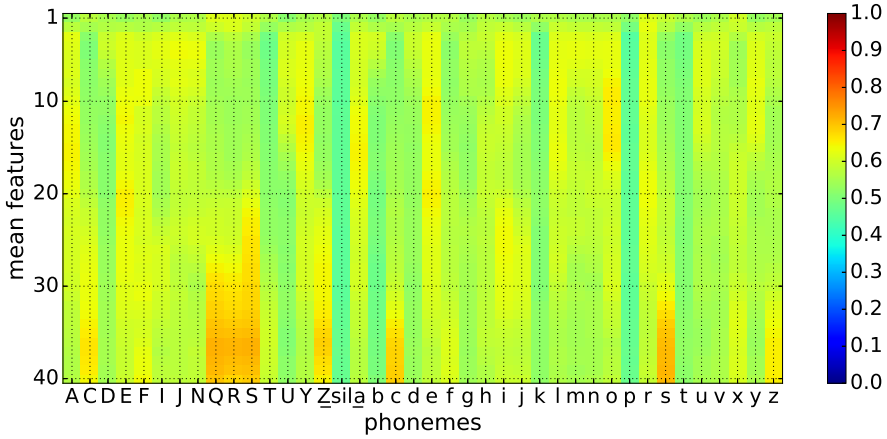


FIGURE 3.24: Average sample for each phoneme, $cs = 0$.

For example, the pause (`"_sil_"`) is very similar to phoneme `"p"` having low values for all frequencies. Phonemes `"A"` and `"a"` also take a similar course, as well as high-frequency groups (`"Q"`, `"R"`, `"S"`) or (`"c"`, `"s"`, `"z"`).

Using the derived parameter settings ($cs = 3$) we end up with a feature vector of length 280. An example for each class is shown in Fig. 3.25. Average values can be found in appendix A2, Fig. A2.1.

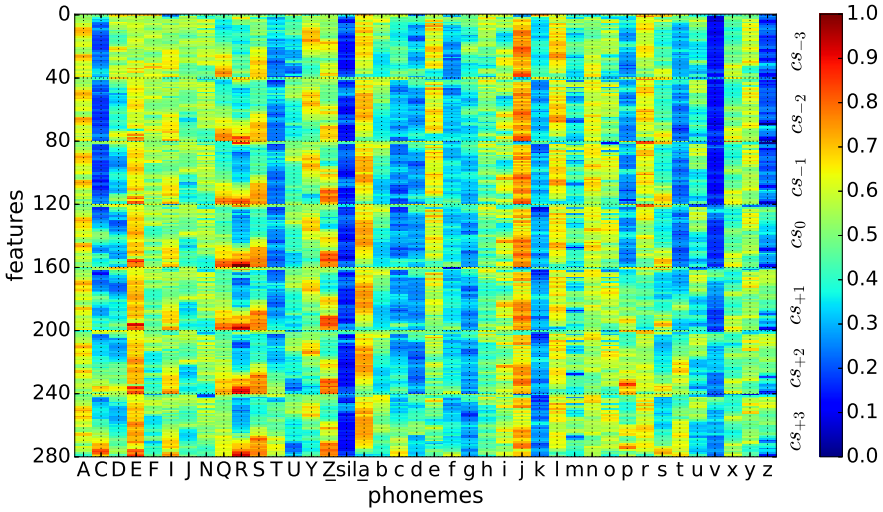


FIGURE 3.25: Randomly selected sample for each phoneme, $cs = 3$.

Classification Results: Speech Dataset

At first we must keep in mind that getting the best possible classification accuracy is not the goal of this work. Rather than spending months of computation time for training huge networks, we choose a smaller network structure and try to get the best out of it. Two different network structures were trained:

1. Network `[280, 50, 2, 50, 40]` (bottleneck[2] network);
2. Network `[280, 100, 50, 40]`.

The learning parameters for both networks are listed in Table 3.12.

dataset		learning parameters	
n samples / class	5000	learning rate	0.07
border size	3	n epochs	100
context size	3	batch size	10

TABLE 3.12: Phonemes: dataset and learning parameters.

[2] A bottleneck network contains a layer that consists of few nodes compared to the previous layers. It can be used to get a representation of the network input with reduced dimensionality.

1. Network (bottleneck)

The network with the bottleneck layer was trained to test accuracy 30% ($MSE' = 0.0105$). A complete confusion matrix can be found in appendix A2, Fig. A2.2. The confusion matrix helped us find phonemes that had been trained better compared to the others. We focused on phonemes with recall[3] greater than 0.5.

The purpose of training a bottleneck network is the reduction of input's dimensionality. Using a layer with just two neurons usually does not lead to a high classification accuracy (also confirmed here), but the advantage is that it can be illustrated in 2D space.

For the selected phonemes, Fig. 3.26 shows the representation of testing samples in the bottleneck layer. We used the *Sigmoid* transfer function, so all samples lie in $\langle 0, 1\rangle \times \langle 0, 1\rangle$.

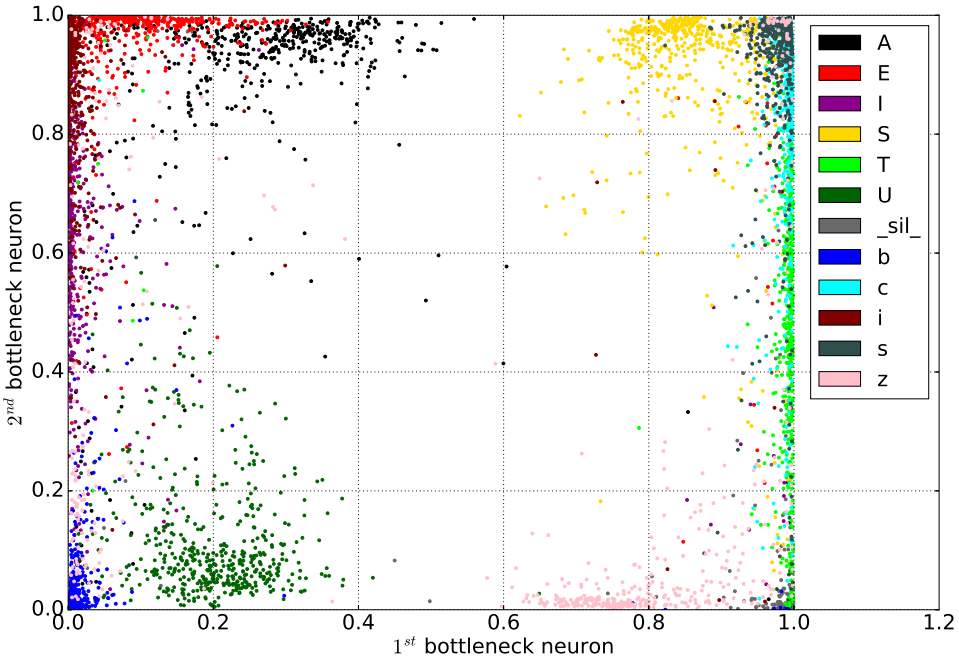


FIGURE 3.26: Representation of individual phonemes in the 2D bottleneck layer (selected phonemes).

The figure confirms some intuitive expectations. For example, samples of phoneme `"I"` are close to representations of phoneme `"i"` and also not far away from `"A"` and `"E"` samples. Another area contains samples of `"S"` together with `"s"` and `"c"` samples.

The bottleneck network with two neurons is a simple example of how the work in networks can be illustrated.

[3]Recall score is the ability of a classifier to find all the positive samples. It is defined as $\frac{tp}{tp+fn}$, where tp is the number of true positives and fn the number of false negatives.

2. Network

The second network with structure `[280, 100, 50, 40]` was trained with the same settings (Table 3.12). The learning ended with accuracy 63.8% and $MSE' = 0.0063$. Fig. 3.27 shows the complete confusion matrix.

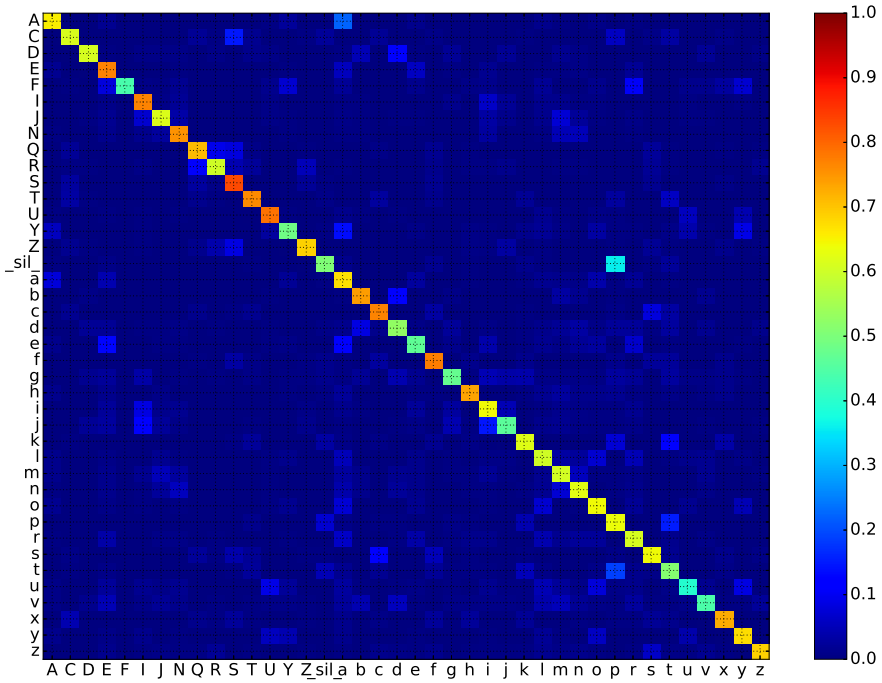


FIGURE 3.27: Confusion matrix of the classification results on the speech dataset.

Pruning Results: Speech Dataset

The second network (`[280, 100, 50, 40]`) was pruned. In this case we sacrificed a few percent of classification accuracy and required $req\_acc = 50\%$ only. The pruning process is tracked in Fig. 3.28.

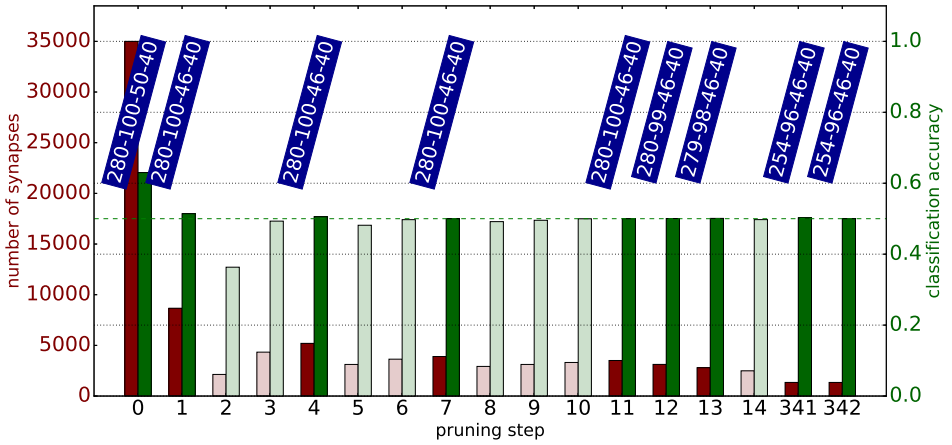


FIGURE 3.28: Illustration of the pruning procedure applied on SPEECH dataset (selected observation). Required accuracy: 50%.

The initial number of synapses 35000 was reduced to 1344. Eight hidden neurons (four in each hidden layer) were cut out. Regarding the input layer 26 neurons were cut out, which reveals that only 254 out of 280 features are necessary to obtain the classification accuracy of 50% on validation data.

The dimensionality reduction is not that significant for this example compared to the others (e.g. MNIST). Also we require a low classification accuracy. However, one must remember that we work with a 40-class problem, which is far from trivial. In the following section we try to find some patterns in the results, which could demystify what is going on in the network.

Pathing and Feature Energy: Speech Dataset

Section 2.3 presents a method called *pathing*. It finds paths from features to classes in pruned networks. We applied this method on the pruned network from the previous paragraph and got the following results. The input layer of the network consists of 280 neurons. The context of size 3 was used (see forming of a feature vector in Fig. 2.12). Therefore we can split the input into 7 moments in time and at each of these moments we have 40 features representing frequency filters (described by Fig. 2.10).

Fig. 3.29 shows the number of phonemes (classes) influenced by individual features, where the features are illustrated in the time-frequency space. For example, if there is no path from a feature to any of the classes, it got zero score (dark blue). In contrary, if it is connected to all possible phonemes (40), it is dark red in the figure.

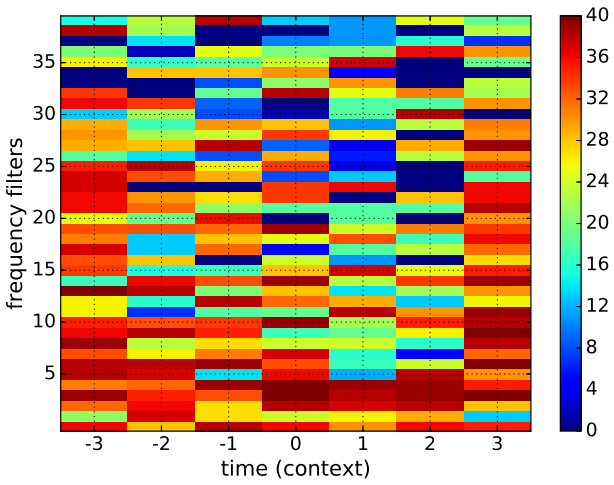


FIGURE 3.29: Number of phonemes affected by individual features. Features are displayed in a time-frequency space.

One can note that low frequencies are used more in general. Interestingly, there also might be a hint that the context features $cs_{(-3)}$ and $cs_{(+3)}$ uses also some higher frequencies compared to the "middle" of the feature vector.

In the following, we consider no context size ($cs = 0$) and analyse just the 40 features corresponding to the frequency filters. Fig. 3.30 shows energies

of these features based on Eq. (2.22). It tells us how much (and in which way) a feature (frequency) influences the class.

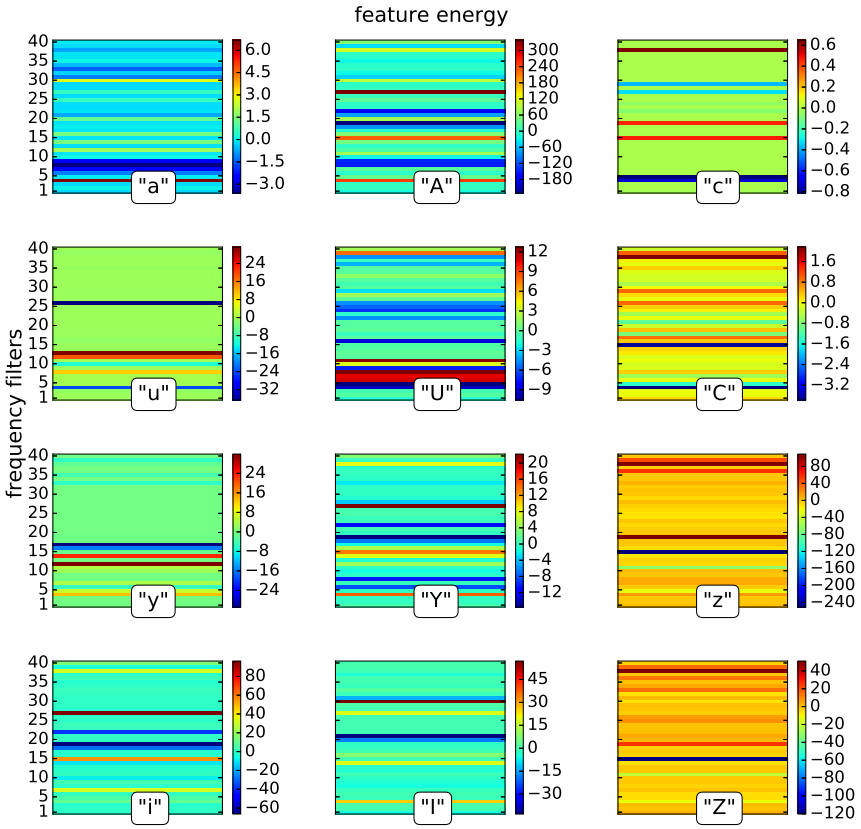


FIGURE 3.30: Feature energies for selected phonemes, phoneme set 1.

For example, phonemes `"z"` and `"Z"` have a similar pattern and differ just in the amount of the energy (see colorbars). The same figures for the other phonemes are in appendix A2.

Fig. 3.31 comes with the total energy of individual features (frequencies in this case) based on Eq. (2.23). Additionally, Fig. A2.5 shows number of active features using each frequency filter for `cs = 0`.

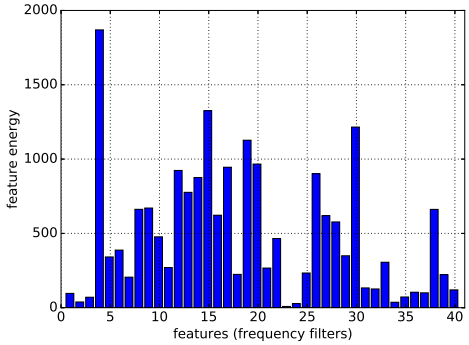


FIGURE 3.31: Total feature energy, speech dataset, $cs = 0$.

Chapter 4

Discussion

The overall objective of the thesis consisted of four subtasks:

1. to design a neural network framework capable of learning a general classification problem;
2. to develop a pruning proceeder equipped by a tool for a dimensionality reduction after pruning;
3. to demonstrate the developed methods on appropriate examples and suggest possible applications for pruned networks;
4. to implement state-of-the-art pruning methods and compare them to the developed method.

4.1 Recapitulation of Methods

At first, we presented the design choices of our classification framework based on feedforward neural networks. We detailed how the learning algorithm was implemented and listed the used evaluation measures.

The developed pruning algorithm was described in section 2.2. The key measure for the identification of redundant synapses called *weight significance factor* (`WSF`) was introduced in Eq. (2.21). Then we described how exactly we proceed when pruning a network and, finally, the dimensionality reduction of weight matrices, titled as *network shrinking*, after pruning was shown.

A new method for working with pruned networks was proposed in section 2.3. The key idea was to track remaining synapses of a minimal network structure and to find some patterns between the network's input and output (features and classes).

Finally, we described the acquisition of the speech dataset. The source recordings were voice commands to control a mobile phone or a navigation in a car. We ended up with three parameters (`bs`, `cs` and `ns`) to be tuned.

4.2 Summary of Results

The methods were tested on six classification problems. Here we sum up the results and observations.

XOR Function

This example from section 3.1 is interesting because of the known minimal network structures (Fig. 3.1) capable of learning the XOR function.

The pruning algorithm was applied on a highly oversized network with a goal to end up with one of the known minimal structures. The desired network architectures were produced in 92% of 100 cases (Fig. 3.4).

The 2D Problem with Unbalanced Feature Information

The example adopted from (Karnin, 1990) comes with two features, where one of them is obviously more important for the classification accuracy than the other one (see Fig. 3.5). Moreover, the separating lines between two classes are parallel to the coordinate axes.

Two hypotheses have been put forward and then confirmed.

1. Two synapses can be deleted from the input-hidden layer, because, due to the axes parallelism to the discriminatory lines, each hidden neuron needs the information from one feature only.

   Result: this behaviour was observed in 92% of 100 cases (Fig. 3.7).

2. The synapses coming from the feature with less information would be the next candidate for deletion, before the synapse carrying more information.

   Result: confirmed in 100% of 100 cases (Fig. 3.8).

The Rule-plus-Exception Problem

This four-dimensional problem adopted from (Mozer and Smolensky, 1989) has been included because it contains samples of two kinds ("rule" and "exception"). The "rule" samples occur more often in the training set compared to "exception" samples. Therefore fitting the "rule" pattern to the model is more important for the classification accuracy than learning the "exception" samples.

Again, two hypotheses have been put and subsequently confirmed.

1. Having two neurons in the hidden layer, one deals with the "rule" samples and the other one with the "exception" samples (Fig. 3.9).

   Result: confirmed in 97% of 100 cases (Fig. 3.10).

2. The synapses connected to the "exception" neuron would be suggested for deletion before the synapses connected to the "rule" neuron.

   Result: generally confirmed by a mean value out of 100 cases (Fig. 3.11).

The Michalski's Trains

Section 3.4 presents a simplification of the well known problem of a feature selection from (Larson and Michalski, 1977). There are two classes and six-dimensional samples, where some of the features are obviously redundant for the classification.

The pruning algorithm does the feature selection by pruning synapses coming from features. Fig. 3.13 shows that out of 100 observations we got:

- a perfect solution (2 features left) in 46% of cases;
- a good but not optimal solution (3 features left) in 32% of cases;
- an unsatisfactory solution (more than 3 or "wrong" features left) in 22% of cases.

Handwritten Digits

The MNIST dataset from (LeCun and Cortes, 2010) was included in order to present the pruning process on a network with many synapses (see Fig. 3.16). The results are summarized in Table 4.1.

	fully-connected	*pruned*
structure	[784, 20, 10]	[465, 20, 10]
n features	784	465
n synapses	15880	1259
accuracy	97%	97%
evaluation time [s]	5.64	2.85

TABLE 4.1: Summarized pruning results on MNIST dataset.

We also performed an alalysis of how many synapses we need to produce a particular classification accuracy - see Fig. 3.18. For example, a network with 38 synapses is able to reach a test classification accuracy of 50% using only 20 features (see Fig. 3.19). The feature selection procedure was then also done for every digit (class) separately (Fig. 3.21).

Classification of Phonemes

Regarding the speech data, the first task was to determine optimal parameters, which led to the maximal trainability of the dataset. Then we showed how the classified samples look like (Fig. 3.23).

Next, we trained a network with a 2D-bottleneck layer and plotted this layer's activity for several phonemes (see Fig. 3.26). Interestingly, phonetically similar phonemes (e.g. `"a"` and `"A"` or `"s"` and `"c"`) took places close together.

Then, we trained a "classical" (`[280, 100, 50, 40]`) network and performed the pruning process (Fig. 3.28). Finally, we computed feature energies (see Eq. (2.22)) based on *pathing* in the pruned network. Results for all phonemes are given in Figures 3.30, A2.3 and A2.4.

4.3 Comparison to Other Pruning Methods

The key part of the network pruning rests in identification of unimportant synapses. In section 1.1, we summarized three studies using similar measures to ours. In the following, we compare them in terms of left synapses and active features after pruning. Additionally, we add two more measures to the comparison:

- *random* : the synapses to prune are chosen randomly
- *magnitude* : weights close to zero are considered less important than those with greater magnitude

All known pruning measures are listed in Table 4.2.

source	*measure name*	*measure*
this study	random	$N(0,1)$
this study	magnitude	$\lvert w_k \rvert$
this study	`WSF`	$\lvert w_k(t_f) - w_k(0) \rvert$
(Mozer and Smolensky, 1989)	relevance	$-\frac{\partial E^l}{\partial \alpha_k}\Big\rvert_{\alpha_k=1}$
(LeCun, Denker, and Solla, 1990)	saliency	$\frac{\partial^2 E}{\partial w_k^2} \cdot \frac{w_k^2}{2}$
(Karnin, 1990)	sensitivity	$\sum_{n=0}^{N-1} \left[\Delta w_k(n)\right]^2 \frac{w_k(t_f)}{\eta \cdot (w_k(t_f) - w_k(0))}$

TABLE 4.2: Known measures of how important synapse corresponding to w_k is.

The experiment was performed on the MNIST dataset. The initial network's structure was `[784, 20, 10]` and the required classification accuracy was 95%. After each pruning step we performed 5 retraining epochs. We ran 10 observations (results in Fig. 4.1)

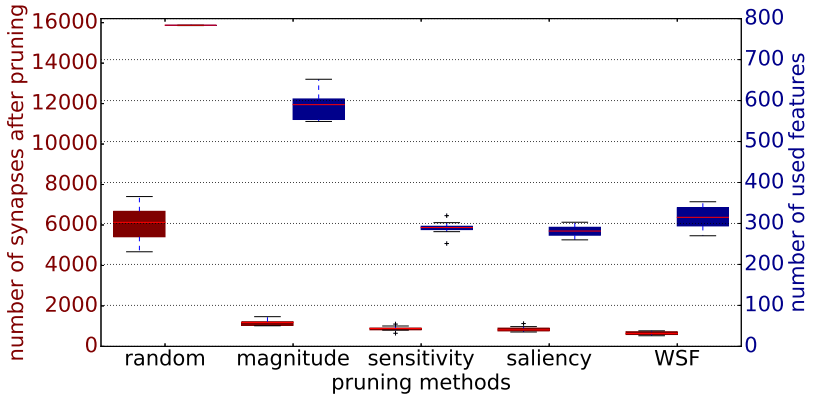


FIGURE 4.1: Comparison of different pruning methods, req_acc = 0.95, retraining: 5 epochs

We can see that using our measure (`WSF`) led to deletion of even more synapses than using *sensitivity* or *saliency*, even though the formula is less demanding in terms of computation time.

Chapter 5

Conclusion

This thesis is about pruning synapses in fully-connected feedforward neural networks. The purpose is to understand individual network parts in order to increase capabilities of using NNs for classification.

We introduced a new measure called `WSF` (*weight significance factor*) to identify unimportant synapses in a fully-connected network. We found out that generally over 90% of the synapses are redundant for classification.

Next, we implemented a network pruning proceeder and tested it on six examples. The experiments confirmed its ability to drive a network to a structure that is minimal for the given data. The minimal structures then lead to partial demystification of even complicated networks. As a side effect of network pruning we got a rapid (over a half) reduction of computation time, which could possibly be useful for low-cost embedded systems.

To sum up the thesis, it is a good start and some of the results are promising, however, I think we have opened some new questions and so there is still a lot of space to work on in this field.

5.1 Future Work

Some of the ideas for the future work are listed here.

- *Shrinking layers.* In this study, we usually work with one hidden layer only. We have not tried to reduce a network in terms of layers yet.
- *Building a network.* As mentioned in the introduction, there are two ways of getting the optimal network structure. In this study, we trained oversized networks and then reduced them. The other way around would be to start from zero and build a network step by step.
- *Tailoring a network.* This could be the way how to take advantages of pruned networks. The idea is to develop a method that would connect individual network parts (possibly differently trained) into one network in order to perfectly fit the given data. The goal would be to get (almost) 100% accuracy for any classification problem.
- *Finding applications.* All the methods are general. It is up to our fantasy to find a good application.

Bibliography

[1] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[2] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[3] James Larson and Ryszard S Michalski. "Inductive inference of VL decision rules". In: *ACM SIGART Bulletin* 63 (1977), pp. 38–44.

[4] Michael C Mozer and Paul Smolensky. "Skeletonization: A technique for trimming the fat from a network via relevance assessment". In: (1989).

[5] Ehud D Karnin. "A simple procedure for pruning back-propagation trained neural networks". In: *IEEE Transactions on Neural Networks* 1.2 (1990), pp. 239–242.

[6] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage". In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[7] Russell Reed. "Pruning algorithms-a survey". In: *IEEE transactions on Neural Networks* 4.5 (1993), pp. 740–747.

[8] Wikipedia. *Plagiarism — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-April-2017]. 2004. URL: https://en.wikipedia.org/.

[9] Peter Bradley. *The XOR Problem and Solution*. 2006. URL: http://mind.ilstu.edu/.

[10] James Lyons. *Mel Frequency Cepstral Coefficient (MFCC) tutorial*. 2009. URL: http://practicalcryptography.com/.

[11] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[12] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[13] Martin Bulín. "Classification of terrain based on proprioception and tactile sensing for multi-legged walking robot". MA thesis. Campusvej 55, 5230 Odense M: University of Southern Denmark, June 2016.

[14] Michael Nielsen. *Neural Networks and Deep Learning*. 2017. URL: http://neuralnetworksanddeeplearning.com/.

[15] A Emin Orhan. "Skip Connections as Effective Symmetry-Breaking". In: *arXiv preprint arXiv:1701.09175* (2017).

[16] *United States Census Bureau*. 2017. URL: http://census.gov/.

[17] *Škoda auto*. 2017. URL: http://skoda-auto.cz/.

[18] Luboš Šmídl. personal communication. supervision of the thesis. 2017.

Appendix A1

Conventions

Here we introduce some notation conventions used in this study. It is an extension of the notation presented in section 2.1.

First of all, we define a dataset consisting of samples X and labels Y'.

$$\underset{n\times p}{X} = \begin{bmatrix} X_1 & X_2 & \cdots & X_p \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

$$\underset{1\times p}{Y'} = \begin{bmatrix} Y'_1 & Y'_2 & \cdots & Y'_p \end{bmatrix}$$

where X_1 is the first sample, p is the number of samples and n is the problem dimension. Y' is the vector of labels. A label can be represented as a number or a string. For example, we can set $Y'_1 =$ "a" be a label of sample X_1, which is a sample of phoneme `"a"`. To make it work together with our neural network implementation, each label has a transcript, which is unique for every class. The transcript is so called one-hot vector, a zero vector of length m (number of classes), which has the only one `"1"` at the position corresponding to its class. For example, if we classify 5 phonemes and the class `"a"` was assigned to position 2, its transcript Y_1 would be:

$$\underset{5\times 1}{Y_1} = \begin{bmatrix} y_{11} \\ y_{21} \\ y_{31} \\ y_{41} \\ y_{51} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

A general matrix of these transcripts Y is then:

$$\underset{m\times p}{Y} = \begin{bmatrix} Y_1 & Y_2 & \cdots & Y_p \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1p} \\ y_{21} & y_{22} & \cdots & y_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mp} \end{bmatrix}$$

As described in section 2.1 we consider Y to be a predicted output of our neural network. Analogically, we get a general matrix of a desired output of

a network and those two can be item-wise compared.

$$\underset{m\times p}{U} = \begin{bmatrix} U_1 & U_2 & \cdots & U_p \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1p} \\ u_{21} & u_{22} & \cdots & u_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m1} & u_{m2} & \cdots & u_{mp} \end{bmatrix}$$

Moreover, we decipher the matrices of weights and biases. We have a vector W of weight matrices $W^{(i)}$, which is always of length $(q+1)$, where q is the number of hidden layers.

$$\underset{1\times(q+1)}{W} = \begin{bmatrix} W^{(1)} & W^{(2)} & \cdots & W^{(q+1)} \end{bmatrix}$$

Shapes of matrices $W^{(i)}$ then reveals the network structure. For example we itemize $W^{(1)}$, which carries the information about problem dimension n. Let's assume we have j neurons in the first hidden layer.

$$\underset{j\times n}{W^{(1)}} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & \cdots & w_{1n}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & \cdots & w_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j1}^{(1)} & w_{j2}^{(1)} & \cdots & w_{jn}^{(1)} \end{bmatrix}$$

Clearly, the first (row) index indicates the neuron we are going to and the second (column) index indicates the neuron we are coming from. A corresponding bias vector would look as follows.

$$\underset{j\times 1}{B^{(1)}} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_j^{(1)} \end{bmatrix}$$

Finally, to help understand Eq. (2.12), we itemize the error matrix in the output layer of m neurons for p samples.

$$\underset{m\times p}{\Delta^{(q+1)}} = \begin{bmatrix} \delta_{11}^{(q+1)} & \delta_{12}^{(q+1)} & \cdots & \delta_{1p}^{(q+1)} \\ \delta_{21}^{(q+1)} & \delta_{22}^{(q+1)} & \cdots & \delta_{2p}^{(q+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{m1}^{(q+1)} & \delta_{m2}^{(q+1)} & \cdots & \delta_{mp}^{(q+1)} \end{bmatrix}$$

Then for $\xi = 1$, the errors corresponding to the first sample X_1 are:

$$\underset{(1)}{\Delta^{(q+1)}} = \begin{bmatrix} \delta_{11}^{(q+1)} \\ \delta_{21}^{(q+1)} \\ \vdots \\ \delta_{m1}^{(q+1)} \end{bmatrix}$$

Appendix A2

Supplementary Data

This section contains some additional tables and figures that did not fit to the main text.

Generated Speech Datasets

The *Phonemes* classification problem from section 3.6 works with data that are acquired based on three parameters (*bs*: border size, *cs*: context size and *ns*: number of samples per class). Table A2.1 lists all generated datasets differing in those parameters. For big *ns* and/or *bs*, it might happen that some phonemes are not present in sufficient quantity in the source recordings, therefore those classes are incomplete (e.g. class `"F"`).

id	*bs*	*cs*	*ns*	*incomplete classes*	*total*	*train*	*devel*	*test*
ds_00	0	0	1K	F (683)	39683	31747	3968	3968
ds_01	0	1	1K	F (683)	39683	31747	3968	3968
ds_02	0	2	1K	F (683)	39683	31747	3968	3968
ds_03	0	3	1K	F (683)	39683	31747	3968	3968
ds_04	0	4	1K	F (683)	39683	31747	3968	3968
ds_05	0	5	1K	F (683)	39683	31747	3968	3968
ds_06	0	6	1K	F (683)	39683	31747	3968	3968
ds_07	0	7	1K	F (683)	39683	31747	3968	3968
ds_08	0	8	1K	F (683)	39683	31747	3968	3968
ds_09	0	9	1K	F (683)	39683	31747	3968	3968
ds_10	1	0	1K	F (589)	39589	31672	3959	3958
ds_11	1	1	1K	F (589)	39589	31672	3959	3958
ds_12	1	2	1K	F (589)	39589	31672	3959	3958
ds_13	1	3	1K	F (589)	39589	31672	3959	3958
ds_14	1	4	1K	F (589)	39589	31672	3959	3958
ds_15	1	5	1K	F (589)	39589	31672	3959	3958
ds_16	1	6	1K	F (589)	39589	31672	3959	3958
ds_17	1	7	1K	F (589)	39589	31672	3959	3958
ds_18	1	8	1K	F (589)	39589	31672	3959	3958
ds_19	1	9	1K	F (589)	39589	31672	3959	3958
ds_20	2	0	1K	F (498)	39498	31599	3950	3949
ds_21	2	1	1K	F (498)	39498	31599	3950	3949
ds_22	2	2	1K	F (498)	39498	31599	3950	3949
ds_23	2	3	1K	F (498)	39498	31599	3950	3949
ds_24	2	4	1K	F (498)	39498	31599	3950	3949
ds_25	2	5	1K	F (498)	39498	31599	3950	3949
ds_26	2	6	1K	F (498)	39498	31599	3950	3949
ds_27	2	7	1K	F (498)	39498	31599	3950	3949
ds_28	2	8	1K	F (498)	39498	31599	3950	3949
ds_29	2	9	1K	F (498)	39498	31599	3950	3949

id	*bs*	*cs*	*ns*	*incomplete classes*	*total*	*train*	*devel*	*test*
ds_30	3	0	1K	F (410)	39410	31528	3941	3941
ds_31	3	1	1K	F (410)	39410	31528	3941	3941
ds_32	3	2	1K	F (410)	39410	31528	3941	3941
ds_33	3	3	1K	F (410)	39410	31528	3941	3941
ds_34	3	4	1K	F (410)	39410	31528	3941	3941
ds_35	3	5	1K	F (410)	39410	31528	3941	3941
ds_36	3	6	1K	F (410)	39410	31528	3941	3941
ds_37	3	7	1K	F (410)	39410	31528	3941	3941
ds_38	3	8	1K	F (410)	39410	31528	3941	3941
ds_39	3	9	1K	F (410)	39410	31528	3941	3941
ds_40	4	0	1K	F (327)	39327	31462	3933	3932
ds_41	4	1	1K	F (327)	39327	31462	3933	3932
ds_42	4	2	1K	F (327)	39327	31462	3933	3932
ds_43	4	3	1K	F (327)	39327	31462	3933	3932
ds_44	4	4	1K	F (327)	39327	31462	3933	3932
ds_45	4	5	1K	F (327)	39327	31462	3933	3932
ds_46	4	6	1K	F (327)	39327	31462	3933	3932
ds_47	4	7	1K	F (327)	39327	31462	3933	3932
ds_48	4	8	1K	F (327)	39327	31462	3933	3932
ds_49	4	9	1K	F (327)	39327	31462	3933	3932
ds_50	5	0	1K	F (253)	39253	31403	3925	3925
ds_60	6	0	1K	D, F, N, Q, R, T	38049	30441	3805	3803
ds_70	7	0	1K	D, F, N, Q, R, T, Z	36140	28914	3614	3612
ds_80	8	0	1K	D, F, N, Q, R, T, Z	34869	27899	3487	3483
ds_90	9	0	1K	D, F, N, Q, R, T, Z, b	33680	26947	3368	3365
ds_5K	3	3	5K	D, F, N, Q, R, T, Y, Z, g	184750	147803	18475	18472
ds_10K	3	3	10K	D, F, N, Q, R, T, U, Y, Z, g, x	335812	268654	33581	33577

TABLE A2.1: Generated datasets: *Phonemes* problem.

Additional Results: Speech Problem

Fig. A2.1 shows an average sample for every single phoneme computed from training samples of dataset `ds_10K` (see Table A2.1).

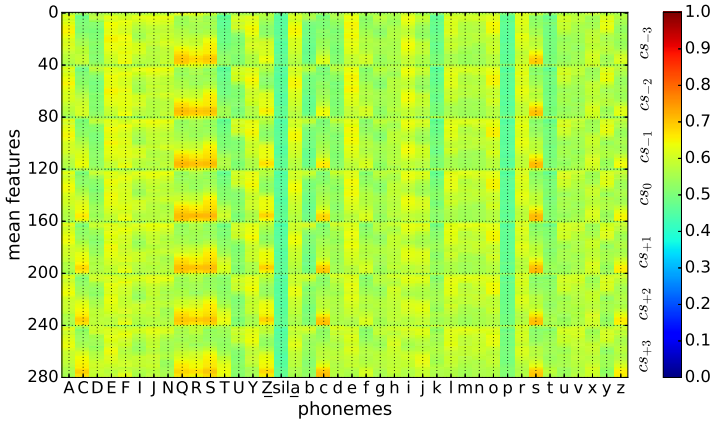


FIGURE A2.1: Average sample for each phoneme, $cs = 3$.

The bottleneck network from section 3.6 was not expected to perform the classification very well. The confusion matrix in Fig. A2.2 confirms this hypothesis. However the goal was to show the representation of selected phonemes in 2D space of the bottleneck layer and we can see that those selected phonemes (e. g. `"A"`, `"E"`, `"S"`, `"_sil_"`, `"z"` or `"i"`) turned out quite well compared to the others.

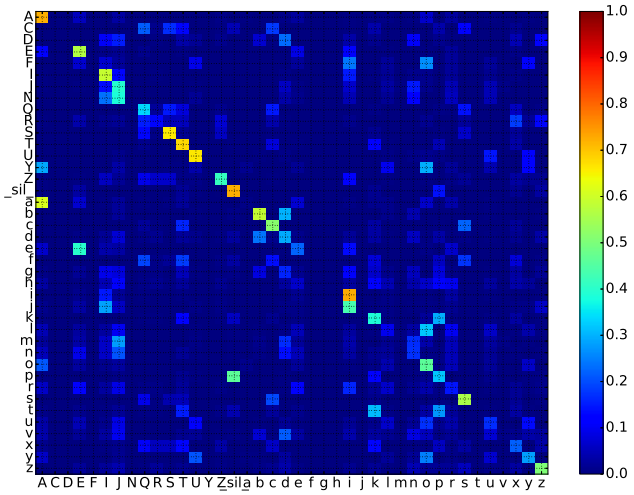


FIGURE A2.2: Trained bottleneck network (speech dataset): confusion matrix.

Fig. A2.3 shows a set of another 12 phonemes as a supplement to Fig. 3.30.

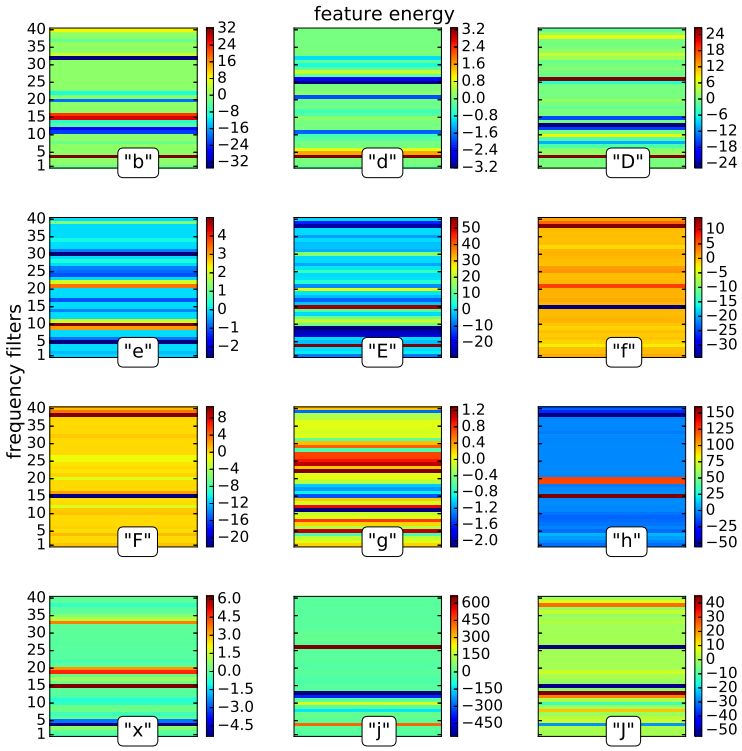


FIGURE A2.3: Feature energies for selected phonemes, phoneme set 2.

And the rest of the phonemes is illustrated in Fig. A2.4.

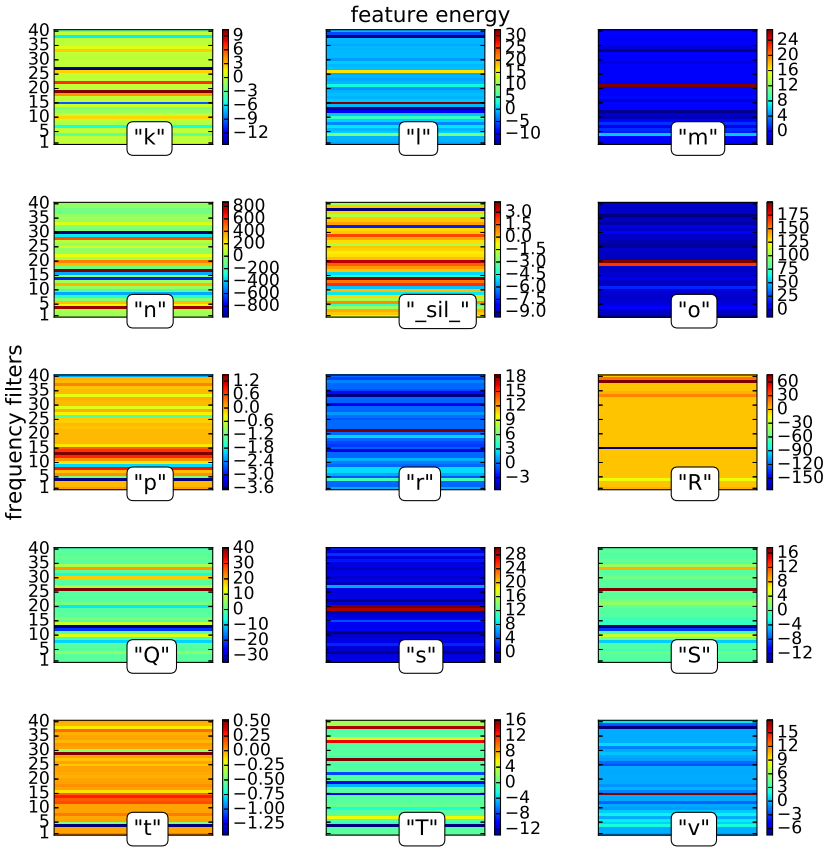


FIGURE A2.4: Feature energies for selected phonemes, phoneme set 3.

One might note the similarity of `"o"` and `"s"` patterns, but remember to check the colorbars. Then we can see that the two phonemes are activated by similar frequencies, but with a different power range. We can also see that `"S"` and `"T"` uses a similar power (energy) as well as phonemes `"p"` and `"_sil_"`.

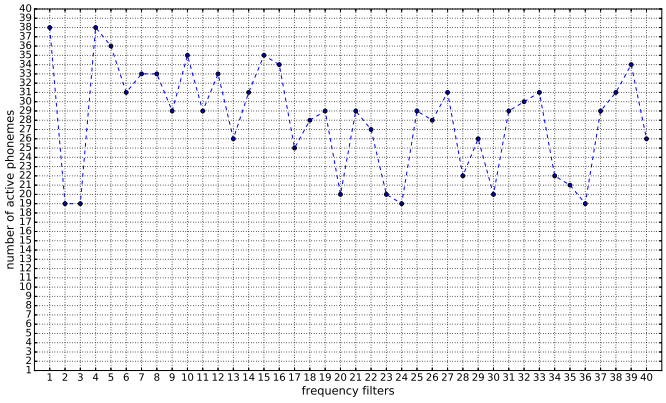
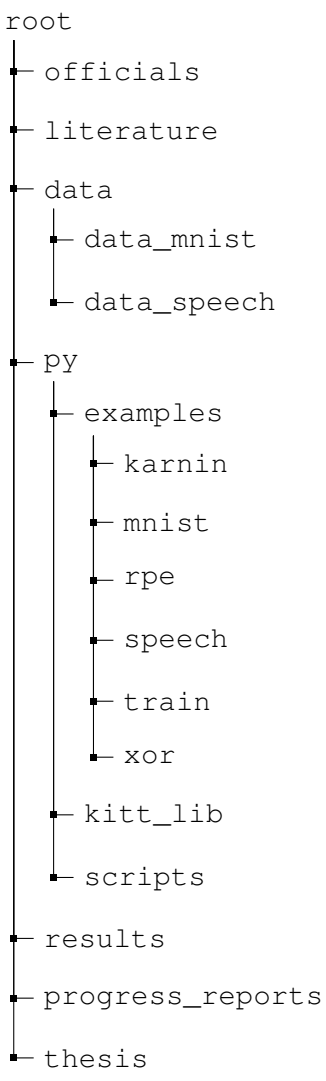


FIGURE A2.5: Number of used phonemes for each frequency filter, `cs = 0`, `bs = 3`.

Appendix A3

Structure of the Workspace



Appendix A4

Code Documentation (API)

The implementation follows the presented methods. The documentation of the main class `FeedForwardNet()` in the developed `kitt_lib` library is given here.

`class kitt_lib.kitt_net.`**`FeedForwardNet()`**

The main class representing a feedforward neural network.

@ **hidden** (array-like) : hidden network structure (e.g. [10, 5]);

@ **tf_name** (str) : transfer function name (e.g. 'sigmoid');

def fit : Fits the network to given data and trains the model.

@ X : array-like, shape (n_features, n_samples)
@ y : array-like, shape (n_classes, 1)
@ val_x : array-like, shape (n_features, n_samples)
@ val_y : array-like, shape (n_classes, 1)
@ learning_rate : learning rate for backpropagation
@ batch_size : mini-batch size for backpropagation
@ n_epoch : number of epochs for backpropagation

def predict : Predicts the probability for each class.

@ x : array-like, shape (n_features, n_samples)
returns y_pred_list : list, sorted tuples (class, prob) by probability

def evaluate : Returns accuracy and error for given data.

@ x : array-like, shape (n_features, n_samples)
@ y : array-like, shape (n_classes, 1)
returns (err, acc) : MSE' error and accuracy

def prune : Prunes the network.

@ req_acc : float, required accuracy to be kept
@ n_epoch : int, number of retraining epochs
@ levels : array-like, pruning levels

def copy : Creates a copy of self.

returns net_copy : kitt_net.FeedForwardNet

def dump : Saves the network.

@ net_filename : str, the path to save network as

def load : Loads a network.

@ net_filename : str, the path to network to be loaded