NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Recognition of handwritten mathematical symbols and expressions

by

Even Dalen, Håvard Langdal and Torkil Solheim

A bachelor thesis submitted in partial fulfillment for the degree of Bachelor in Computer Engineering

in the
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

May 1, 2018

Preface

The assignment was chosen for several reasons. We were all interested in the contents of the assignment, and we saw this as a great opportunity to explore the machine learning world and learn new technologies. We had little to none prior experience with machine learning, and we knew this would be challenging. This however, made the assignment more interesting as we were motivated to learn and curious about ways to solve the problems related.

Before we chose the assignment, we read about the basics of neural networks and the mathematics behind them. We quickly understood that a lot of the math we have learned in previous courses was relevant, and being able to make use of this knowledge further increased our interest for the assignment.

The process from start to finish was demanding, we have had days, even weeks were we felt no progress was made. This is of course frustrating and made us doubt our ideas very often. In the initial phase of the project time went by much faster than expected, we spent several weeks just researching the most optimal way to solve the product part of our project. After the initial phase of the project we all had basic ideas about how it could be solved in the best way possible. Although we had a lot of different ideas, we all quickly understood that in the first phase of coding, we should all use a method for high interaction and cooperation between us. This need of a solid method led us to pair programming, even though we were three. At first, this was extremely helpful, and it led to some creativity and a way to quickly respond to changes.

First of all, we would like to thank Ole Christian Eidheim for being both an inspiration and our mentor during this project. We would also like to thank our external mentors Tore Forbregd, Trygve Solstad and Hermund Andre Torkildsen for an exciting project and great inputs.

Even Dalen Håvard Langdal Torkil Solheim

1.5.2018
Trondheim

Assignment

In this project, a handwriting recognition system for mathematical signs will be developed. The system will be used in a web application that is being developed at the Department of Teacher Education in collaboration with IDI. The system will use a standard machine learning algorithm such as convolutional neural networks that represent state-of-the-art handwriting recognition. Training data must be collected from students to train the network.

Since the beginning of the project, some changes were made to the assignment. Early on, it was not known if developing a system that would work well and enrich the functionality of the web application was actually achievable. Thus, there was decided to go in the direction of developing a proof of concept, which could be developed further.

The assignment was also limited to focus on the handwriting recognition system, and not how to implement it into the web application.

What makes our schools bachelor's projects somewhat unique is the fact that we are in addition to solving an engineering task, we are also solving a hypothesis. At least that's the overall thought for software engineering projects, our project stands out as more of a research project than a pure software engineering project.

Abstract

In this thesis we researched different approaches for recognizing handwritten mathematical symbols and expressions. Interpreting different typographic styles has historically been a challenging task, however, recent improvements in both hardware and software for machine learning have made the task more approachable.

The purpose of the assignment was to create a module which interprets mathematical symbols and expressions. We created an example system to display how our module can be integrated into an independent system. Combined, the module and example system provide the complete functionality as specified by our project owners.

Through our research, we explored different methods in pre- and post-processing of digital handwriting in order to best classify mathematical expressions. Our two approaches were to process drawings into images (using convolutional networks) and interpreting raw trace data (using recurrent networks). The combination of both approaches gave the best result and is the architecture used in our final model.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Motivation

In education, the usage of digital tools has increased.

In education the usage of digital tools to present and distribute tasks to students has increased [0].

This includes tasks with multiple choice, graphs, geometric models and algebra. Traditionally these tools does not support dynamic input in form of digital ink, which makes the tools somewhat unrealistic from classic math on paper.

1.2 Problem definition

How can proportionality and relativity be used in recurrent neural networks?
Our problem was created after being presented with our project task, which is a task to create a module for interpreting handwritten mathematical symbols. The task itself was focused on creating an generic module which could easily be used in both existing and upcoming projects for the department of teacher education at Norwegian University of Science and Technology.

1.3 Structure of the report

1.3.1 Chapter 1 Introduction

The introduction introduces important thoughts, concepts and motivation behind this project without going in depth.

All of the concepts introduced in this chapter are required to understand both relevance and motivation.

1.3.2 Chapter 2 Theory

The theory consists of relevant theory to understand both approach, limitations, choices and eventually conclusion.
By our project definition we are to expect that the reader is at a level of a 2nd year computer engineering student at our own department.
Even though we know that the 2nd year computer engineering student does have the conceptual knowledge required to understand the theory behind neural networks, some of the mathematics will be explained.

1.3.3 Chapter 3 Method

This chapter has a direct correlation with chapter two. Here the choices and technology in chapter two are justified. In addition, we will also discuss tools that made our project flow optimal.

1.3.4 Chapter 4 Results

The results are divided into at least three parts.

1. Scientific results This section will describe the data/product which will serve as a foundation for answering our problem/hypothesis in chapter 1.1.
2. Engineering results This section will describe the goals set in the start of this project. In a clearly defined product and goal this would be the vision document, in this projects case it would be much more up to the product owners.
3. Administrative results This section will go into the process behind this project, from the planning stage to the finishing touches on the thesis. In addition to the

process we will also talk about our goals set early in the project phase, and how our goals changed according to our progress and knowledge. Eventually we will arrive at methodology and how there was made attempts to follow agile methodologies.

1.3.5 Chapter 5 Discussion

The discussion is also divided into the same parts as Chapter 4. In this chapter we are to answer some questions.

To explain the different sections, the questions linked to each section is listed underneath.

1. Scientific results Why did, or why didn't our hypothesis or scientific problem hold? In addition to discussing the hypothesis we are to discuss how our results can be interpreted in combination with or as an answer to our hypothesis or scientific problem.

2. Engineering results How did the software part of this project turn out? Were the product owners satisfied with the result? Which requirements were met and which were not? Why did the results become what they are? What went good? What went bad?

3. Administrative results What went good or bad as a result of choosing a specific process, approach or technology? What went good or bad regardless of the chosen process, method or technology?

In addition to discussing our results, the weaknesses of this work should is to be discussed, how it could have been done differently and so forth.

1.3.6 Chapter 6 Conclusion

The purpose of this work is to find ways of how to use machine learning to recognize handwritten mathematical symbols. Our product owners have a need, and our goal is to fulfill that need in the best way possible. We hope to accomplish exactly that by examining the different approaches to an issue like this and try to achieve desired results.
In addition to the project

1.3.7 Attachments

This explains itself, our attachments are included here.

1.4 Target audience

Machine learning is an extremely large field, with enormous potential. We encourage the reader to be open to some technical aspects including our thoughts and process from start to finish. Machine learning is quite complex and we hope to inspire students and others new to machine learning. With that said, both our report and code is meant for a reader with some knowledge about software, linear algebra and numerical mathematics.

1.4.1 Report

This project report is meant for everyone curious about an specific introduction to practical examples of machine learning using neural networks. The specific part is linked to this projects assignment, which is to use different approaches to achieve an respectable accuracy.
Even though this report is meant as an introduction to a complicated field, we have based our writing on that the reader has knowledge equivalent to an 2nd year computer engineering student at Norwegian University of Science and Technology in Trondheim. Thus, some of the fundamentals are excluded.

1.4.2 Application

The application which consists of an simple back end and a front end library is available to use under the MIT license [1]. As previously stated, the application is specifically created to fit the needs of our product owners, but the front end library can be used in a more general matter.

1.5 Purpose

The purpose of this project is both to create something useful for our product owners, while exploring the possibilities and limitations of recognition of handwritten mathematical symbols. In addition to creating something useful, we hope to inspire further work on the subject. Choices made during the project were strongly influenced by previous work, for example Martin Thoma, 2015 [2] and Catherine Lu and Karanveer Mohan [3].

1.6 Abbreviations

ANN	Artificial Neural Network
BMP	Bitmap
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CROHME	Competition on Recognition of Online Handwritten Mathematical Expressions
GPU	Graphics Processing Unit
GUI	Graphical User Interface
ICFHR	International Conference on Frontiers in Handwriting Recognition
ILU	Department of Teacher Education
InkML	Ink Markup Language
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptrons
MNIST	Modified National Institute of Standards and Technology database
OCR	Optical Character Recognition
RNN	Recurrent Neural Network

Table 1.1: A list of abbreviations used.

Chapter 2

Theory

2.1 Handwriting recognition

Handwriting recognition systems are computer systems that can recognize characters and symbols written by hand. These systems have been continuously developed and researched ever since the ancient days of computers [4], beginning with the optical character recognition (OCR) systems in the 1950s [5]. The early OCR systems were very limited in terms of hardware, which resulted in a great deal of testing and failing. Researchers had high aspirations for handwriting recognition, but progress was slow and it gained the reputation of being an extremely difficult problem [4]. As hardware and computers became more advanced, the OCR and handwriting recognition systems followed in the same direction. Hardware has usually been the bottleneck for these systems (trenger kilde).

When computational power increased, using artificial neural networks became more popular, also for handwriting recognition. For example, in (TODO kilde fukushima, wake), a neural network was trained to recognize 35 symbols. Since then, neural networks has been used in many solutions, and has proven to be a major contributor to the field, as it is one of the most used tools in the leading solutions today (some of these are presented in section 2.1.3).

This field has for a long time been a worldwide research project [5]. Many competitions on different handwriting recognition problems has been arranged, and researchers from all over the world has cooperated to produce solutions This further confirms the difficulty of a general handwriting recognition system.

2.1.1 Main approaches

Handwriting recognition systems generally has two main approaches, online and offline [6]. Online recognition uses data from pen strokes based on sampled coordinates of pen movements and the time difference between them. These pen strokes are also referred to as traces and is stored separately. This means that strokes can easily be separated, which in many ways can aid in the segmentation problem (see section 2.1.2.2).

While online recognition uses dynamic data, offline recognition uses static data such as images. In this approach the recognition system has to rely solely on pixel values in the images, which introduces some complications. For instance, these systems can't separate traces in the same easy way as online recognition systems since only images are available.

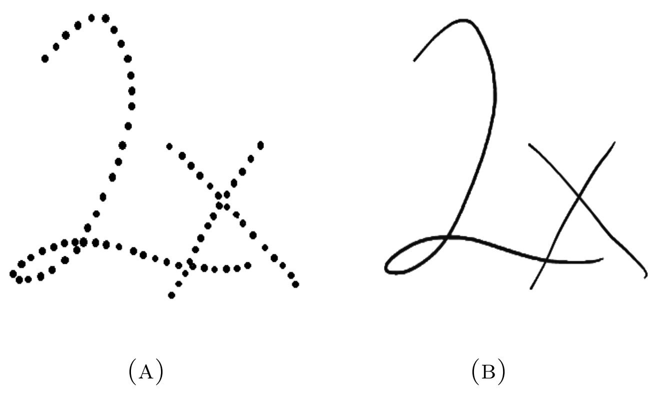


FIGURE 2.1: A: Online recognition data with coordinate samples. B: Offline recognition data (only an image).

With the online input data, there is possible to create an image of all the traces by drawing a line between the coordinates. This means that online recognition systems has access to the same images as offline recognition systems, as well as coordinates for each of the traces. Having more accurate input data and more features to analyze is important since handwriting is different from person to person. In general, online recognition systems has higher performance than offline recognition systems [6].

2.1.2 Common steps in handwriting recognition

In the recognition process there are several problems that must be solved. Usually the most common problems include segmentation and classification of symbols. In addition, the pre- and post-processing steps can demanding.

2.1.2.1 Pre-processing

In the pre-processing step the raw input data is prepared for analysis. Raw input data can come in many formats and sizes, and may contain values that are not valid. This step deals with these problems by converting the data to a format that suits the consecutive steps better, and discards information that is irrelevant for the recognition process [preprocessing kilde - Huang et al].

A common task in pre-processing is to normalize the input data. This simply means that all the data is scaled to the same value range. As a result, the data samples are cleaned for values that are not valid. For handwriting recognition, normalizing generalizes the different writing styles by removing some of the variations [preprocessing kilde - Huang et al].

Since the input can be offline or online, there are many techniques that can be used to prepare the data. Common tasks in pre-processing for offline data includes removal of noise from the images and converting greyscale images to binary images [6]. The images could also be padded to not contain any blank space.

[image of binarization]

Online data can be pre-processed in completely different ways than offline data. The data samples consists of coordinates, which can be used in many ways. For example, the coordinates can be plotted and lines can be drawn between them. Without any pre-processing these plots might have sharp edges and overall be a false representation of handwriting. Smoothing could be applied to the coordinates to fix this by using interpolation of points. The data samples might also need to be parsed to a format that is more suitable for processing, for example matrices or arrays.

[image of the smoothing]

2.1.2.2 The segmentation problem

The segmentation problem deals with segmenting symbols from traces. Finding out if two traces that overlap belongs to the same symbol or not is difficult because they can overlap in countless ways.

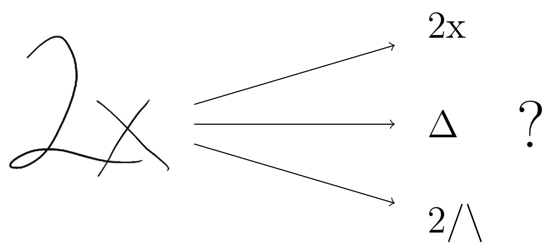


FIGURE 2.2: The segmentation problem deals with segmenting symbols from traces. The sample above has three traces, one for the "2" and two for the "x". How they are segmented can lead to many different outcomes.

2.1.2.3 Classification of symbols

The classification step deals with the actual recognition. Here, each of the segmented symbols will be classified to a truth that a system calculates. There are many tools that could be used for this task. A popular approach is to use neural networks (see section 2.4).

The classification tool needs to have an alphabet of symbols to recognize. If some these symbols look alike, there might be some misclassifications. For instance, the numbers 1 and 7 could look very similar in some cases.

2.1.2.4 Post-processing

After all the symbols have been segmented and classified, there might still be some post-processing that can be done. The results of the classification has to be presented somehow. There are many formats the results could be converted to, for example XML, LaTeX (see section 2.2) or plain text.

The results of the classification might also contain errors. In some cases a contextual analysis of the sequence of characters might correct some of these mistakes. Analyzing the sequence of characters might reveal poor grammar and words that does not exist. These words could then be switched with similar words that is valid.

2.1.3 Recognition of handwritten mathematics

Recognition of handwritten mathematics is a field in handwriting recognition that deals with mathematical symbols and expressions. This field has many of the same problems as recognition of words and characters, however, there is also a new problem that must be dealt with. Just as regular language, mathematics has a set of grammatical or positional rules that should be followed. For example, there are fractions, exponents and square roots that has to be interpreted correctly. This two-dimensional notation requires new processing techniques compared to analyzing regular words.

A context of how all the symbols fit together has to be built by looking at positional values. This context search could be done in the post-processing step, using the results of the classification for guidance. The classification of symbols might also be completely wrong, as the truth to a symbol has many context-dependencies [Kilde - zanibbi, blostein]. For instance, the decimal separator ".” and the multiplication operator "·” must be interpreted based on their position to their surroundings.

[Image of the steps in math recog]

Mathematical notation uses a large amount of different symbols. There are numbers, variables, operators and structural symbols. With a rising amount of symbols to classify, there are room for more mistakes to be made by the classification system.

2.1.4 Previous work and existing solutions

There is a lot of work previously done in recognition of handwritten mathematics, and many solutions has been made. The first work in this field dates back to the 60s and has gradually grown ever since [7]. Researchers from all over the world has published theses and many bachelor- and master-theses from students has contributed with ideas and solutions. For example, in [Kilde - Matsakis 1999] from 1999 a system that uses a gaussian classifier for recognition of handwritten mathematics is presented. [trenger et eksempel til?]

In recent years a competition called Competition on Recognition of Online Handwritten Mathematical Expressions hosted by ICFHR has been held [7] [8]. In these competitions teams from around the world competed in recognition of different math notations. The main task was to recognize formulas. Other subtasks was for example to recognize matrices and isolated symbols. This competition has contributed a lot to the math recognition field, by producing standardized data sets and evaluation metrics.

In the CROHME competition held in 2016 the MyScript corporation won in all categories. To produce the best results, their system handles segmentation, classifying and interpretation concurrently [7]. Their technology is available commercially (1.5.2018).

2.1.5 InkML

InkML is a markup language based on XML to describe digital ink data. The format is specified by the World Wide Web Consortium (W3C) [9]. InkML groups it's content in specific XML-style elements called ink, the ink element often has sub elements which is a trace element.

```
<trace id="0">
    207 136, 199 141, 197 143, 196 144, 195 146, 194 147, 193 149
</trace>
<trace id="1">
    847 201, 847 204, 846 207, 846 211, 846 215, 845 219, 845 222
</trace>
<trace id="2">
    833 235, 822 235, 821 235, 820 235, 819 236, 819 235, 819 236
</trace>

<traceGroup xml:id="3">
        <annotation type="truth">+</annotation>
        <traceView traceDataRef="1"/>
        <traceView traceDataRef="2"/>
        <annotationXML href="+_1"/>
</traceGroup>
```

Listing 2.1: **InkML** example of three traces, each with an file-unique id. In addition to traces, we have listed a tracegroup, which specifies what and where the traces belong to. Truths are used when providing labels to use in supervised learning, which is explained later in this chapter.

2.2 LaTeX

LaTeXis a tool for preparing documents for high-quality typesetting.

It is most regularly used in the creation of technical or scientific documents, but it is versatile. Thus, it's useful in almost any type of publishing.[10]

2.3 Machine learning

We are currently living in a time where big data surrounds us in our daily lives. Take YouTube for example, YouTube has over one billion users, each day several hundred millions of hours are being watched and every second 5 hours of YouTube content is uploaded.

This massive ammount of data could benefit from analysis, which is what machine learning is doing. According to [11], machine learning can be defined as a set of methods that can automatically detect patterns in data. These uncovered patterns can also be used to predict future data or support decision making. For example, in [12], historical data of maintenance is used to predict failure for individual mechanical parts.
[13] explains machine learning as an applied form of statistics, which uses computers to do statistic estimations on complicated functions and decreased attention on proving the related confidence intervals.

2.3.1 Supervised learning

Supervised learning is the machine learning task to map an input to an output. The training data needed has features in addition to a label. For example, to train a model(in this case, a neural network) in the case of supervised learning, one would need to supply the model with a lot of training data, in addition to labels associated with the data.

If you want to classify or use any supervised learning algorithm, the algorithm could study many examples of what it's going to classify to eventually be able to classify to the correct class, which in this case is labels associated with mathematical symbols.
To explain this in another way think of the function p(y | x), this reads the probability of y given x has happened. This is exactly what supervised learning is, the y labels are provided by a supervisor.

On the other hand we have unsupervised learning which is basically observation and after observing data it attempts to learn probability distribution p(x) of them implicit or explicit. An example of unsupervised learning is clustering. In clustering the goal is to divide the data into different clusters based on their attributes. [13]

2.4 Artificial neural networks

2.4.1 Core concepts

A neural network consists of a combination of artificial neurons, also referred to as computational units. These neurons takes a vector as input. The input is multiplied the with precomputed weights, and a bias is added. Then the result is summed and used as input to the activation function (2.4.2). Lastly the weighted sum is returned as output from the neuron. [14] [15]

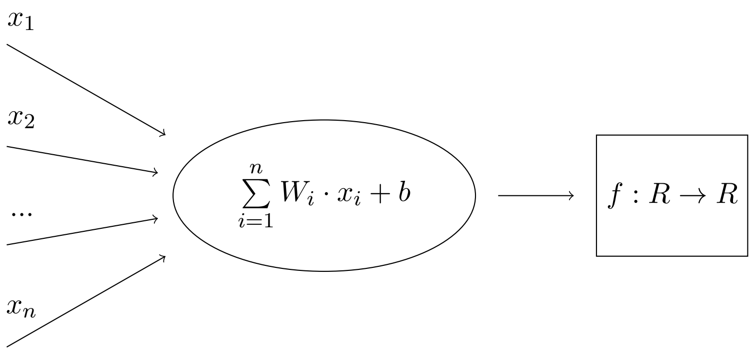


FIGURE 2.3: Diagram of a single artificial neuron.

The neurons in a neural network are typically organized into different layers, where each layer has a fixed number of neurons. The diagram below is an example of how these neurons can be connected together.

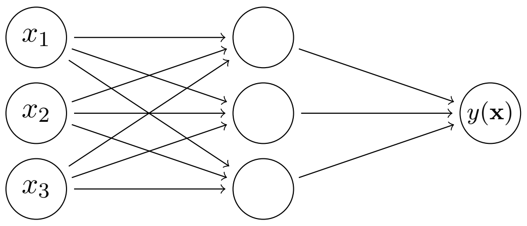


FIGURE 2.4: Diagram of a single layered neural network. It contains three neurons in the input layer, three neurons in the hidden layer, and a single neuron in the output layer.

The leftmost **layer** of neurons in 2.4 is called the input layer. The input layer is a passive layer, meaning the layer does not modifiy the data. Each neuron in the input layer receives a single input and duplicate the input to its multiple output neurons. The hidden layer and the output layer from 2.4 are active neurons. These neurons modify the data before propagating the values to the next layer. The modifications are described in 2.3. The result of the output layer is the approximation done by the neural network. [16] In order for a neural network to be a good approximator, the weights and biases in each neuron has to be optimized to minimize error in the output layer. This tuning process is performed through training. Training a network means giving the network labeled training data, comparing the output label from the network to the label provided, and tuning the weights and biases to best satisfy a loss function (2.4.4). This optimization process is typically done using an optimizer such as gradient decent, and through a concept called backpropogation. This is described in detail in 2.4.5.

2.4.2 Activation functions

Activation functions in artificial neural networks has inspiration from how neurons in our brain works. In a mathematical model of an artificial neuron by McCulloch-Pitt [17], the activation of a neuron was modelled as the unit step function. This model has later been generalized to use functions such as sigmoid, rectified linear unit, softmax and tangens hyperbolicus. [18]

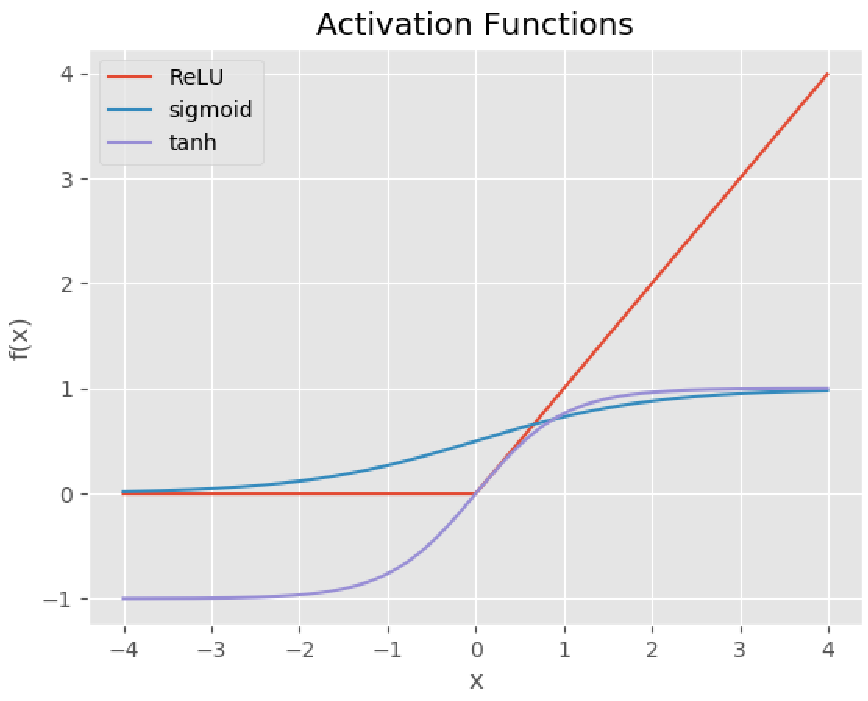
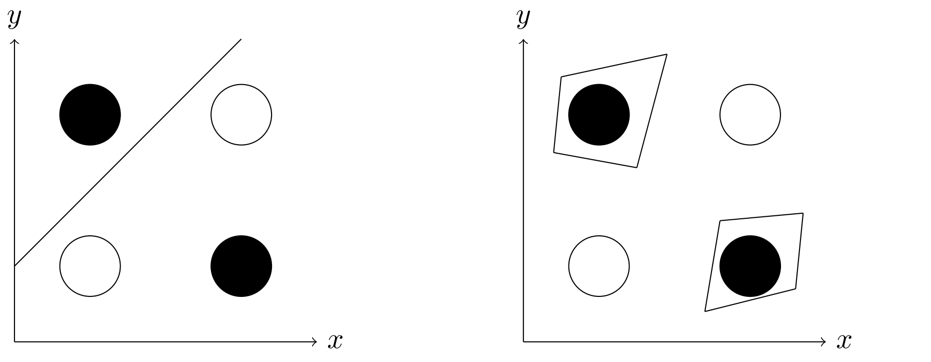


Figure 2.5: Three activation functions plotted on the interval [-4,4].

One purpose of an activation function is to introduce nonlinearity into the model. A network without nonlinearity can not represent functions which are nonlinear, despite having multiple layers. On the other hand, it is proved that. **leshno1993multilayer**

> A standard multilayer feedforward network with a locally bounded piecewise continuous activation function can approximate any continuous function to any degree of accuracy if and only if the network's activation function is not a polynomial.

An example of the importance of nonlinearity in neural networks can shown in the exclusive-OR problem. [18][19]



(A) Using no activation functions (B) Using non-linear activation functions

FIGURE 2.6: Figure of the exclusive-OR problem, by the capabilities of a model with non-linear activation functions versus no activation function.

Activation functions are also important for squashing the output of a neural network into desired bounds. A common use case is for returning class probabilities for a classification model, where softmax is often used as the activation function for the last layer. Softmax is an activation function which transforms its inputs to a combined sum of 1.[19]

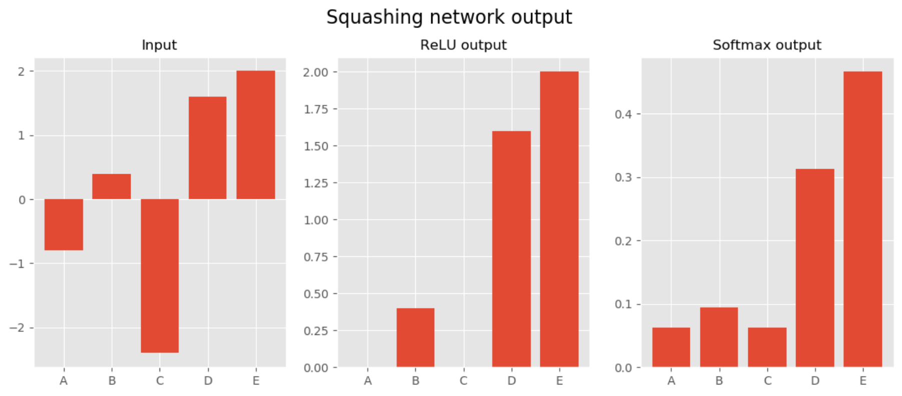


FIGURE 2.7: Figure of how example input data is transformed through two different activation functions. Data flows from left to right, with output from last graph used as input to the next.

2.4.2.1 ReLU

Rectified linear unit or ReLU is a non-linearity introduced by Hanloser et al. in 2000 [16], and has later been found successful as a nonlinearity in deep neural network. It is a function which disallows negative values, and outputs the input if positive, else it outputs 0.

$$f(x) = max(x, 0) \tag{2.1}$$

ReLU is a cheap mathematical operation to perform, as the max operation only includes addition, multiplication and comparison, while other previously popular nonlinearities are dependent on calculating exponential functions.

ReLU has also performed well compared to other nonlinearities in stochastic gradient descent (SGD) 2.4.3. Krizhevsky et al. demonstrated an improvement of a factor of six, when comparing number of epochs needed to reach 25% error rate with a network using ReLU, compared to an equivalent network with tanh as a nonlinearity. [20]

A negative side effect from ReLU is the dying ReLU problem, where neurons can be pushed to a state where the neuron will output the same value regardless of input. This is a variation of the vanishing gradient problem **??**. The concept of dead neurons is described in 2.4.7[21]

2.4.2.2 Softmax

Softmax is an activation which transforms its inputs to a distribution which adds up to one. This makes the softmax function useful to represent probabilities, and is therefore often used on the last layer of neural networks. Combined with the loss function categorical cross entropy 2.4.4.1, softmax can be used to predict multiclass classification.[22]

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.2}$$

2.4.2.3 Sigmoid

Sigmoid has been historically popular as activation function in neural networks. It works by squashing the input to the range $[0, 1]$, where small numbers are returned as a value close to 0, and large numbers are returned as values close to 1.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

An issue with sigmoid is that the outputs are not zero-centered, which means the output of the function will all be of the same sign. When these values are sent to the next layer of the network, the inputs will cause the gradients to be the same sign as well. This can cause undesirable updates of weights during backpropagation, with zig-zag dynamics for weight updates. [23]

2.4.2.4 Tanh

Tanh is an activation function with similar characteristics as sigmoid, and can be represented as a scaled sigmoid (which can be seen in the equation below).

Tanh, however, squashes the inputs to the range $[-1, 1]$, which means it does not inherit the same issues with non zero-centered outputs. Therefore, in practice, tanh is always preferred to sigmoid. [23]

$$tanh(x) = 2 \cdot \sigma(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{2.4}$$

2.4.3 Calculating an output

The output of a neural network a list of numerical values. This vector contains as many values as there are output units in the neural network. To create this output vector the neural network performs a series mathematical operations on the input vector.

For this demonstration the network in figure 2.4 will be used. The first step is to calculate the outputs from the hidden layer, H_o.

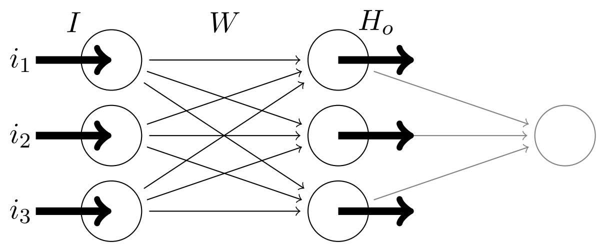


FIGURE 2.8

Each of the input values are multiplied with the weight to each unit in the hidden layer. All the inputs to a unit in the hidden layer is then summed. Matrix multiplication can be used for this, where a matrix containing all the weights is multiplied with a column-matrix containing the input values. The weights matrix is a 3x3 matrix since there are 3 inputs and 3 units in the hidden layer.

$$W * I = \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \\ w_{1,3} & w_{2,3} & w_{3,3} \end{bmatrix} * \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = H_i$$

To calculate the output from the hidden layer the activation function chosen is used on these input values.

$$activation(H_i) = H_o$$

The output from the hidden layer is then sent further in the network.

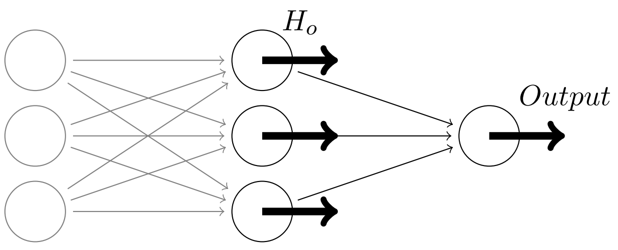


FIGURE 2.9: The final output is calculated

The same process with matrix multiplication and activation function is repeated with the output from the hidden layer and the weights to the output layer. This will calculate the final output from the neural network. Since the output layer consist of only one unit, the final output will be a single value.

For a larger neural network the same process is used but on a bigger scale. If a neural network has more than a single hidden layer, the process is repeated for each of them.

2.4.4 Loss functions

Loss function, also called error function or cost function, is a functions used to calculate the margin between the expected value, and the predicted value in a neural network. The loss function is used in conjunction with gradient descent [kilde?], in order to update the weights and biases.

A commonly used loss function is mean squared error (MSE)

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.5}$$

In MSE, the error is measured by finding the mean error (Expected (labeled) value - Predicted Value) squared. By squaring the error, outliers are punished much more than small errors. The error function can therefore be used to shape what the network should avoid, and what it should do more of.

2.4.4.1 Crossentropy

Crossentropy is a loss function typically used for classification tasks. Binary crossentropy is used when there are only two different classes, while categorical crossentropy is used for multiclass classification.

In order to represent the loss between a predicted class probability distribution and the correct distribution, we use a format called one-hot encoding. An example for predicting letters, we define three vectors; C (classes), $\hat{Y}$ (truth) and Y (prediction)

TODO: Finn gode kilder

$$\begin{aligned} C &= [A, B, C, D] \\ \hat{Y} &= [0.6, 0.1, 0.2, 0.1] \\ Y &= [1.0, 0.0, 0.0, 0.0] \end{aligned} \tag{2.6}$$

Cross entropy can be calculated through the following equation.

$$H(Y, \hat{Y}) = \sum_i Y_i \log \frac{1}{\hat{Y}_i} = -\sum_i Y_i \log \hat{Y}_i \tag{2.7}$$

Using the example from above. We calculate the loss of prediction $\hat{Y}$ as.

$$\begin{aligned} H(Y, \hat{Y}) &= \\ -(1.0 \cdot \log 0.6 & \\ +0.0 \cdot \log 0.1 & \\ +0.0 \cdot \log 0.2 & \\ +0.0 \cdot \log 0.1) & \\ = -1.0 \cdot \log 0.6 \approx -0.51 & \end{aligned} \tag{2.8}$$

Notice how the prediction did on the correct answer A is the only number which influences the resulting loss. If the predicted probability is close to 0, our loss will get very large, and when the predicted probability is close to 1, the loss will get very small.

$$\begin{aligned} \lim_{p \to 0} \quad -\log(p) &= \infty \\ \lim_{p \to 1} \quad -\log(p) &= 0 \end{aligned} \tag{2.9}$$

2.4.5 Training with backpropagation

Backpropagation is an algorithm used in the training of a neural network. The algorithm was introduced by [24], however, it was not until 1986, in [25] that the importance of the algorithm was acknowledged [KILDE - brilliant.org - backpropagation]. This section will lightly explain the algorithm without going into details for the underlying mathematics.

The algorithm has two phases: the forward phase and the backward phase. In the forward phase the output from each layer is calculated as in section 2.4.3. The final output is then compared to a target output specified by the training sample to calculate an error. This error is then sent backwards through the neural network.

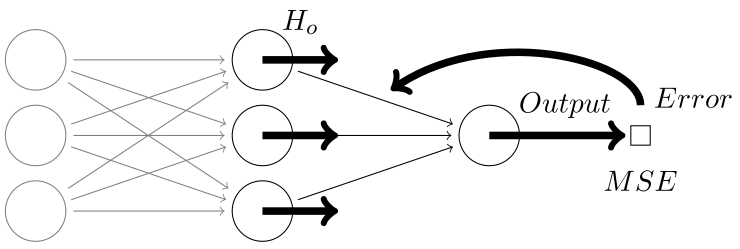


FIGURE 2.10: Finding the error with mean squared error and sending it backwards.

To actually train a neural network the weights between each layer has to be modified to reduce the overall error of the output. Gradient descent is used for this, where the partial derivatives for the error produced by a unit is calculated with respect to each of the connected weights.

For the neural network in figure 2.11, the error from the output layer is first used to calculate the modifications for the weights between the hidden layer and the output layer. Then, the process is repeated for each of the hidden units. Three new errors is calculated, one for each of the hidden units. Finally, these errors are used to calculate the modifications for the weights between the input layer and hidden layer.

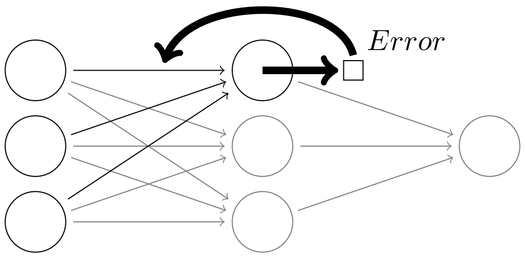


FIGURE 2.11: Using error from the hidden units to calculate modification for the connected weights. The same process is repeated for the other hidden units.

2.4.6 Backpropagating through time

2.4.7 Dropout

Dropout is a technique used during training of a neural network. For a training sample, random units and all their connections are temporary dropped from the neural network [26]. This means that the connections dropped will not be updated for this training sample.

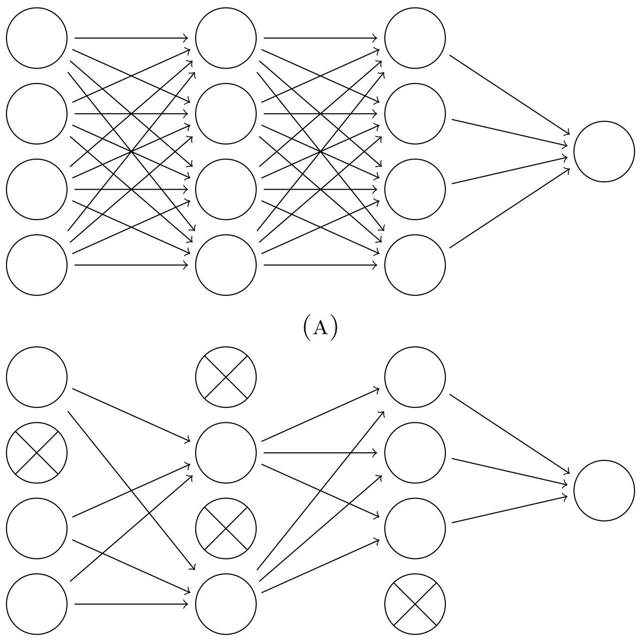


FIGURE 2.12: Neural networks without (A) and with (B) dropout applied. The crossed circles are the dropped units.

The purpose of dropout is to reduce the chance of overfitting a neural network. A neural network is overfitted when it is too closely related to the training samples, and not being as generalized as possible. An overfitted neural network will give good results if tested with the training data, but might fail on new data that the neural network has not seen before. In [26], classifiers and neural networks trained with and without dropout applied is compared using standardized data sets. In every comparison, neural networks with dropout was ranked high, if not highest.

2.5 Common problems

In artificial neural networks there are issues that optimization algorithms must overcome. The first issues to address is **vanishing** and **exploding** gradient. Typically they occur when the feedforward neural network is extremely deep and in recurrent networks. The common factor here is that both types of artificial neural network generate long computational graphs.
To explain this problem further, some mathematics is required.
A square matrix A with n linear eigenvectors $q_i(i = 1..n)$ can be factorized as

$$A = Qdiag(\lambda)Q^{-1} \tag{2.10}$$

In equation 2.10 A is our original matrix, Q is an nxn matrix where each column is an eigenvector q_i and $diag(\lambda)$ is a diagonal matrix of eigenvalues. If we do this multiplication t times, it only changes the number of times the diagonal matrix is multiplied. [27]

$$A^t = Qdiag(\lambda)^tQ^{-1} \tag{2.11}$$

Both the vanishing and exploding gradient issues both originate from the scaling according to the diagonal matrix with eigenvalues $diag(\lambda)$, with repeated multiplication by A. Vanishing gradient makes it difficult to find out in which direction parameters should move in order to move the cost function. While exploding gradients makes learning unstable. [13]

Another issue with artificial neural networks is "dead" neurons, this is a direct cause of using the rectified linear unit also known as ReLU. ReLU 2.1 is widely used to not get caught in between gradient issues, but they may cause weights to be updated in a way that makes the neuron unable to activate again. This is an irreversible consequence of setting the learning rate too high. [14]

2.6 Feedforward neural networks

Feedforward neural networks is also often referred to as deep feedforward networks or multilayer perceptrons (MLPs) are the most typical of deep learning models. The ultimate goal of feedforward neural networks is to approximate some function **f**, for example a task to classify (map) an input x to a specific class y. y = $\mathbf{f^*(x)}$ in mathematical notation. The networks mapping is defined as y = $\mathbf{f(x,}\theta\mathbf{)}$ where θ is learned in the way that **f** is approximated in the best way possible.
[13][p. 163] explains why networks such as these are called feedforward.

> These models are called **feedforward** because information flows through the function being evaluated from **x**, through the intermediate computations used to define f, and finally to the output **y**. There are no **feedback** connections in which outputs of the model are fed back into itself. ([13])

2.6.1 Convolutional Networks

Convolutional networks is also often referred to as convolutional neural networks or CNNs. CNNs are specialized for processing data that has a known typology. Examples includes images, which is an 2D-grid of pixels and say time series data. In this attempt to get some accurate classifications for handwritten mathematics we are both using images and sequential data. The reason that CNNs are actually called a CNN is because they use a convolution, which is a mathematical operation.

> *Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.* ([13])

To understand this further, the convolution operation needs to be explained. A convolution can be explained as an operation on two functions which has real-valued arguments.

The example [13] uses to explain a convolution is as follows; we are tracking the location of a spaceship with a laser. Our laser is feeding us a single output x(t), which reads the position of the spaceship at time t. Now if the laser is noisy we may read data that could be unrepresentative. Thus, we could need some sort of average function and in addition to averaging we would also like to specify that the most recent recordings are most important. This is solvable using weights, w(a) is our weighting function, where a is the age of a measurement.

$$s(t) = \int x(a)w(t-a)da \tag{2.12}$$

FIGURE 2.13: [13][p. 322] Example convolution which averages sensor readings and weights the most recent sensor readings. w(a) is the weighting function, x(t) is the position of the spaceship at time t. The result of this is a new function s, which gives us a smoothed estimate

$$s(t) = (x * w)(t) \tag{2.13}$$

FIGURE 2.14: Asterisk notation of a convolution.

The first argument is called input, the second argument is often called kernel and the output of this is often called the **feature map**. In the example with lasers x would be our input and w would be the kernel.
In a perfect world the laser would be able to provide real-time measurements, but that is not realistic. Often readings are discretized, that means that a new reading will arrive at a given interval, an example of this could be a new reading every second. This would eventually lead to t being an integer and since both x and w is dependent on t we end up with the discrete convolution. [13]

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.14}$$

FIGURE 2.15: [13][p. 323] definition on the discrete convolution.

For two-dimensional inputs, there is two-dimensional kernels, which is commutative:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.15}$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.16}$$

FIGURE 2.16: [13][p. 323] the two dimensional kernel in both it's original form in addition to the version with the flipped kernel. The last version is a result of **flipping** the kernel relative to it's input.

The actual function that usually exists in machine learning libraries is **cross-correlation**. **Cross-correlation** is the same as a convolution without flipping the kernel and it is often referred to as convolution.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \qquad (2.17)$$

FIGURE 2.17: [13][p. 324] the **cross-correlation** function.

Sparse interactions, connectivity or weights is a result of making the kernel smaller than the input. Even if an image has many thousands of pixels it is possible to extract small crucial features using smaller kernel sizes.

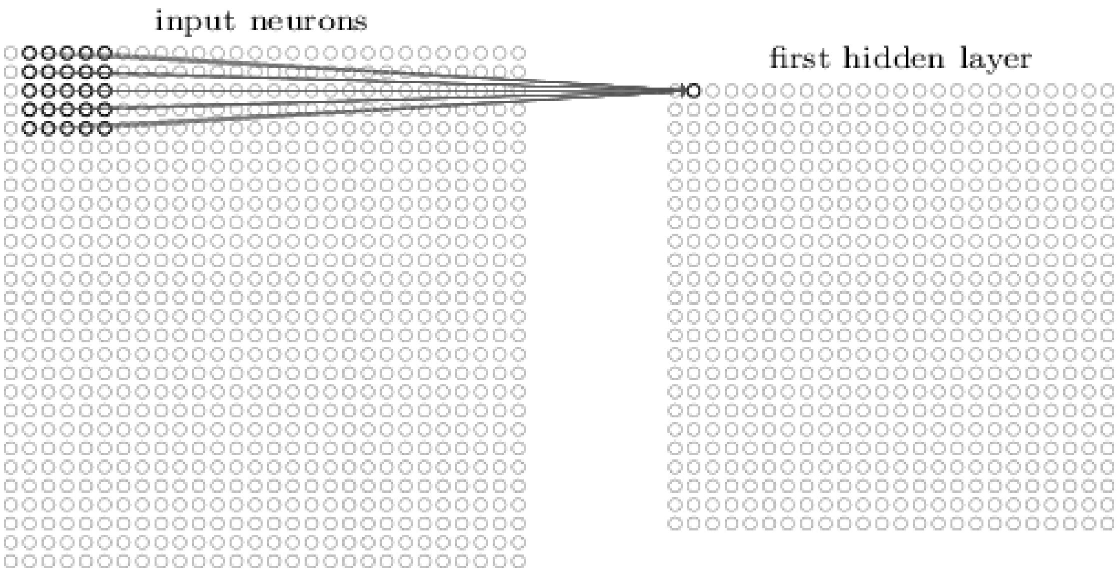


FIGURE 2.18: This figure from [28] shows how a convolution layer works with a 5x5 kernel, also referred to as a filter or feature detector. In this example, our input is 28x28, if we apply a 5x5 kernel we will end up with 24x24 neurons in the hidden layer. Available from http://neuralnetworksanddeeplearning.com/images/tikz45.png, 23.04.2018

A convolutional layer has typically three stages, the first is to do the convolutions as illustrated in 2.18, these are linear activations. When all the convolutions are done, they are passed through a nonlinear activation function. An example of a nonlinear activation function is the sigmoid function 2.3. Furthermore this stage is often referred to as the **detector** stage.

The last stage of the a typical convolutional layer is when we apply a **pooling function**[29] to further modify the output. A **pooling function** is able to replace the output of a small area with a summary of the nearby outputs. There are multiple pooling functions, in our project and for our purpose we went with **max pooling**. The max pooling function uses a pooling unit which has a specified size, for example 2x2. The pooling unit in a max pooling function simply outputs the maximum activation of it's region(2x2), a max pooling function with a pooling unit region of 2x2 would turn 24x24 neurons to 12x12 neurons. [13] [28]

2.7 Recurrent neural networks

Recurrent neural networks, also often referred to as RNNs [25] are neural networks which can process sequential data. CNNs are sort of specialized on taking and analyzing grid like data, such as images or videos. RNNs on the other hand are more specialized on processing data that is sequential. The idea behind RNNs is to share parameters in different parts of a model. Sharing parameters makes it possible to use the model on data with variable length.

> *Such sharing is particularly important when a specific piece of information can occur at multiple positions within the sequence.* ([13])

Furthermore an artificial neuron or computing unit is not only determined by it's activations in previous layers, it is now possible for an artificial neuron or computing unit to be determined by it's own activation earlier.
[13] [28]

Data provided by the CROHME competitions is sequential ink coordinates as illustrated in listing 2.1. Thus, the principle of information sharing is quite significant in this projects case as well.

2.7.1 Long short-term memory

Long short-term memory unit is a type of computational unit often used in recurrent neural network. LSTM networks were introduces in 1997 by Sepp Hochreiter and Jürgen Schmidhuber**??**, and has been later improved in further research, for instance by adding a forget gate**??**. These units can be utilized to keep important information over time, as opposed to traditional recurrent units, by adding new information on top of old state and not overriding it for each timestep. This is described in **Empirical evaluation of gated recurrent neural networks on sequence modeling**

> Unlike to the traditional recurrent unit which overwrites its content at each time-step (see Eq. (2)), an LSTM unit is able to decide whether to keep the existing memory via the introduced gates. Intuitively, if the LSTM unit

detects an important feature from an input sequence at early stage, it easily carries this information (the existence of the feature) over a long distance, hence, capturing potential long-distance dependencies.

An LSTM layer typically consist of a several computational units. These will be referred to as *memory cells*, and a the collection of one or more cells will be referred to as *memory blocks*. The input at the current timestep t is notated with x, the output from of the cells are notated by h, and the block's state is notated as C.

A key part of an LSTM memory block is the Constant Error Carousels (CEC) **??**, also known as state. This is visualized as the horizontal line in the top part of the figure below. The state flows through all cells in a memory block, and the state eventually determines the cell's output.

Each memory cell includes several gates, which are used to decide what the network should remember. These gates determines how the input should influence the state of the memory block. This is done through filters, which are combinations of a sigmoid activation function and a pointwise multiplication. These filters are known as gates. A typical LSTM memory cell has three gates, an input gate, output gate and a forget gate.

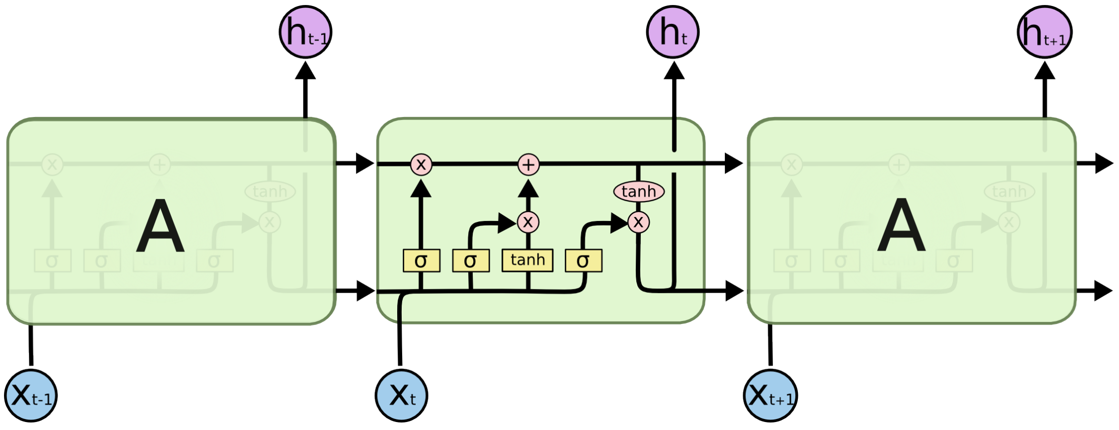


FIGURE 2.19: Model of a memory block with three memory cells. The center cell includes a visualization of the different parts of the cell, and how the information flows inside the cell. The cell's input values are the current state of the memory block C_{t-1} (top left), the output of the previous cell h_{t-1} (bottom left), and the current input value x_t. The upper line is the memory block's state, which flows through each cell of the block. The yellow squares are neural network layers with a specified activation function. (From **Understanding LSTM Networks – colah's blog**).

The first gate which is encountered in the memory cell is the input gate. This gate is notated as the combination of the sigmoid layer and the pointwise multiplication top left of the cell. This gate decides how much the previous cell's output h_{t-1} and this timestep's input x_t should influence the current state. The output of the sigmoid function is multiplied with the previous state C_{t-1}. If the output is 0, everything should be forgotten, and if the output is 1, everything should be remembered.

The next layer with sigmoid activation decides how much of the current input should influence the the memory state. This process is similar to the input gate. The layer noted by the *tanh* activation function decides what should be remembered. The output of the pointwise multiplication between the sigmoid and tanh layer, is then added to the state. The output of the memory cell is a filtered version of the current state, as seen top right in 2.19. The cell's state is run through a *tanh* function, to squish the values to the range $[-1, 1]$, then the output is filtered through the output of the sigmoid function, similar to how state was removed and added in the previous steps. The cell's output is also stored, and will be used as input in the next iteration.

The sequential nature of these memory blocks makes LSTM networks good for predicting sequencial data. In pratice, most of the information we encounter is not needed for future predictions. By using gates, the LSTM network can decide which information should be remembered and to which scale the previous remembered information should influence future decisions. **????**

2.7.2 Gated Recurrent Unit

Gated Recurrent Unit is often referred to as GRU. It is was recently proposed in 2014 by Cho et al. **??**, as an alternative to LSTM. GRU has the same capabilities of keeping important information over time using gates, however without use of the LSTM's memory cells. This makes a GRU unit cheaper computationally, and has shown to outperform LSTM on models with fixed parameters **??**.

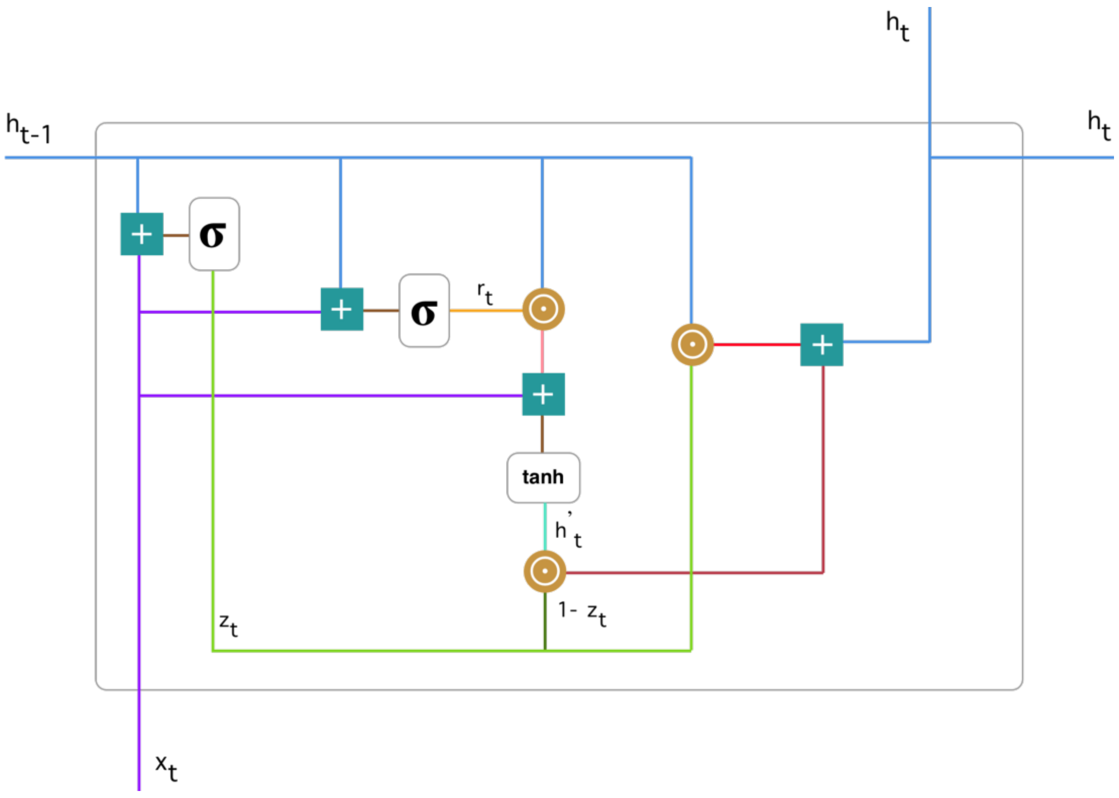


FIGURE 2.20: A model of a Gated Recurrent Unit. h_{t-1} is the output of previous cell, h_t is the output of current cell. z_t is the output from the update gate. r_t is the output of the reset gate. x_t is the input at current timestep. (From **Understanding GRU networks**

The GRU cell calculates a temporary state from a combination of the previous state and the current input. The temporary state will be notated as h_{tmp}.

h_{tmp} is calculated by adding the input at this timestep x_t to a part of the previous state h_{t-1}. The influence of h_{t-1} on h_{tmp} is decided by the reset gate, r_t. This addition is then the input to a layer with *tanh* activation function, which results in h_{tmp}.

To calculate the output of the GNU cell, we need to decide how much of the previous state we are going to keep. This is decided by the update gate z_t. By a element-wise multiplication of z_t with h_{t-1}, we include only the parts of the previous state we wish to keep. h_{tmp} is multiplied element-wise with $1 - z_t$, and thereby we fill the state removed from previous state, with the state calculated in the current cell.

It is important to remember that the different gate's output is a value in the range $[0, 1]$, therefore if $z_t = 0$ then $1 - z_t = 1$

If the update gate is 0, $z_t = 0$, none of the previous state should be kept. Then $1 - z_t = 1$, and the all the current state will be kept. On the other hand, if $z_t = 1$, the cell's output will equal the previous cell's output $h_{t-1} = h_t$, and the current input will not influence the output of the cell $(1 - z_t = 0)$

Chapter 3

Technology and method

3.1 Introduction

In this chapter the technology and methods applied in this project is reasoned. The project was at no point limited to follow Matistikk, but for the sake of simplicity and engineering flexibility tools which both provided good results in short time and if it were to be implemented in Matistikk, a Python solution was preferable.

3.2 Technology

The choice of technology was highly influenced by the nature of the project assignment, if the goal was to create a module for an existing Django application, Python was already the obvious choice. Eventually the project drifted in the way of a proof of concept, Python came out even further as the obvious choice. The reason why Python is the obvious choice in this case, is the ability to go from idea to a working prototype in the fastest way possible. In addition to productivity Python has a lot of mature libraries, in the nature of working towards a proof of concept, one would need stable, tested and well documented libraries. We found especially Keras [30] to be a perfect fit for our task, which is a library that makes TensorFlow or some other machine learning library easier to use.
The choice of back end and processing tool was now chosen, Keras backed by TensorFlow is an extremely feature-rich combination. Since the project went in the way of a proof of concept we needed some graphical user interface in which we could collect data, present statistics and so forth. The selection of front end was determined by the project assignment. Since Matistikk is web-based, we found it quite natural to go with some

simple HTML, Css, JavaScript and Tornado. Tornado is a simple library to build web servers in Python.

3.3 Communication and dataflow

The foundation was quickly made, that led to an classic client/server approach. Already in the first weeks of the project Keras and deep learning was our focus.The progress made was not only based on our own experimentation. Articles and projects posted was quickly digested and learned from. CROHME gave the project it's desirable data and insight into the assignment. CROHME provided both tools and sequential data in InkML files, see listing 2.1 for an InkML example file.
Even with the new discovered data, the research still continued. Lu and Mohan [3] and Thoma [2] inspired the work made in this project. The majority of data at that stage was sequential, a decision was made to use a WebSocket to handle front to back end communication. For the sake of simplicity, it was also implemented a way to do this over a simple http post request as an alternative to the WebSocket implementation. Doing it this way created a simple, but solid foundation for the engineers behind Matistikk to do it the way they wanted.
After discussion with our project owners, we were drawn to the direction of using a http post request. The test application also does not think about the amount of data it sends per classification request. If the canvas is changed, it still re-sends the entire buffer. This is both to easy re-evaluate the buffer, but in addition this can trigger changes in the segmentation process, which can eventually lead to a different classification.

3.4 The recognition system

The recognition system consists of both some "hard-coded" logic and the neural network. As stated in chapter 2.1, handwriting recognition includes several steps required to classify correctly. The "hard-coded" logic solves the segmentation issue in an elementary way, but it is quite vulnerable to noisy input. In addition to segmentation we do a whole lot of prepossessing to make sure our actual data flowing from our front end to our back end is roughly the same size format, data type and so forth.

3.5 Artificial Neural Network

Artificial neural networks in image recognition is often to be found in tutorials. Since the project engineers had no prior experience with artificial neural networks it was necessary to experiment with different types of neural network and data. The MNIST data set consists of handwritten digits which is available in Keras. The amount of thorough tutorials on the MNIST data set inspired the project in new aspects. At that time, there was made a decision to use CNNs to handle the recognition problem. This meant that all the sequential data must be converted into images.
This was a decision that turned out to work as desired, it was not a choice that led to success. But the issue was not more focused on handling segmentation and applying different techniques to obtain good accuracy. After some experimentation with CNNs, attempts at getting it done with RNNs was started. The project went into a hard period, where we were battling variable length input and at the same time we felt that our working solution would not improve.

3.6 Classification model

3.7 Architecture

3.8 Image data

When deciding on the image format it was clear that the best approach was to resemble an already existing data set in order to obtain more data. Experimenting on the MNIST data made the project engineers explore possibilities to convert our InkML data to grey scale images in the same format as MNIST data. This, of course was no straight forward procedure, after several tries over a couple of weeks it was decided to go forward with generating bitmaps. Generating BMPs was a much simpler task, and luckily we found BMPs that were of handwritten mathematical symbols.

3.8.1 Preprocessing

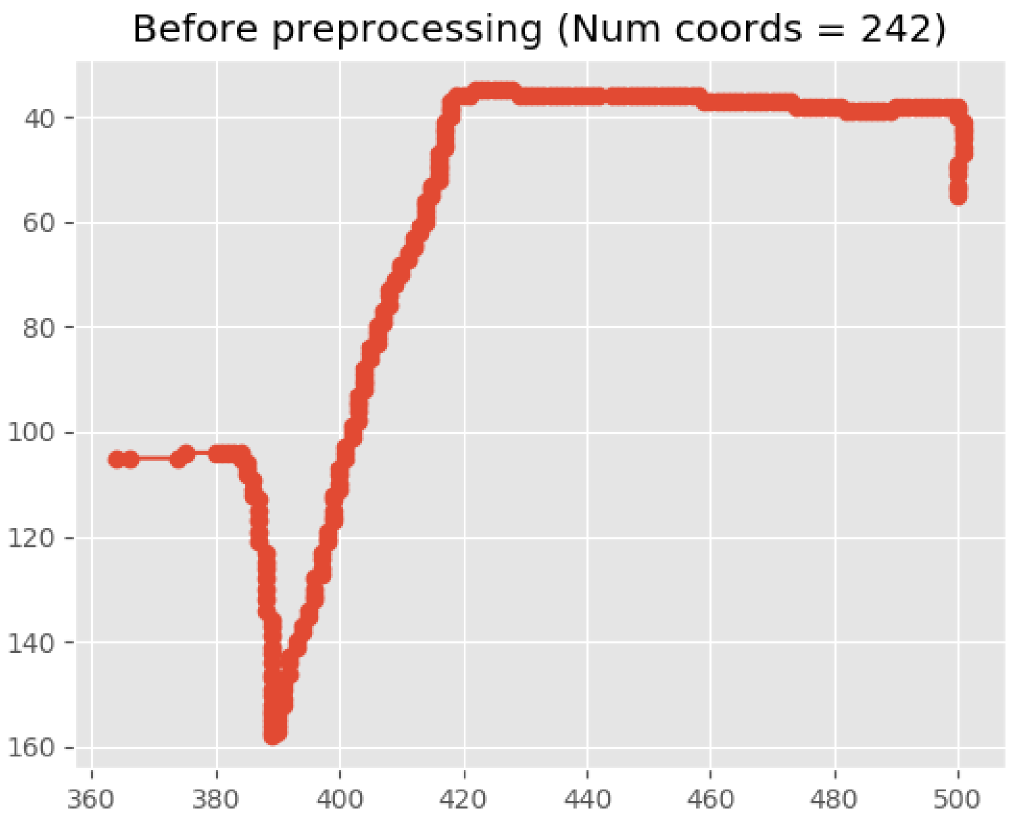


FIGURE 3.1: A square root which has not been thorough preprocessing.

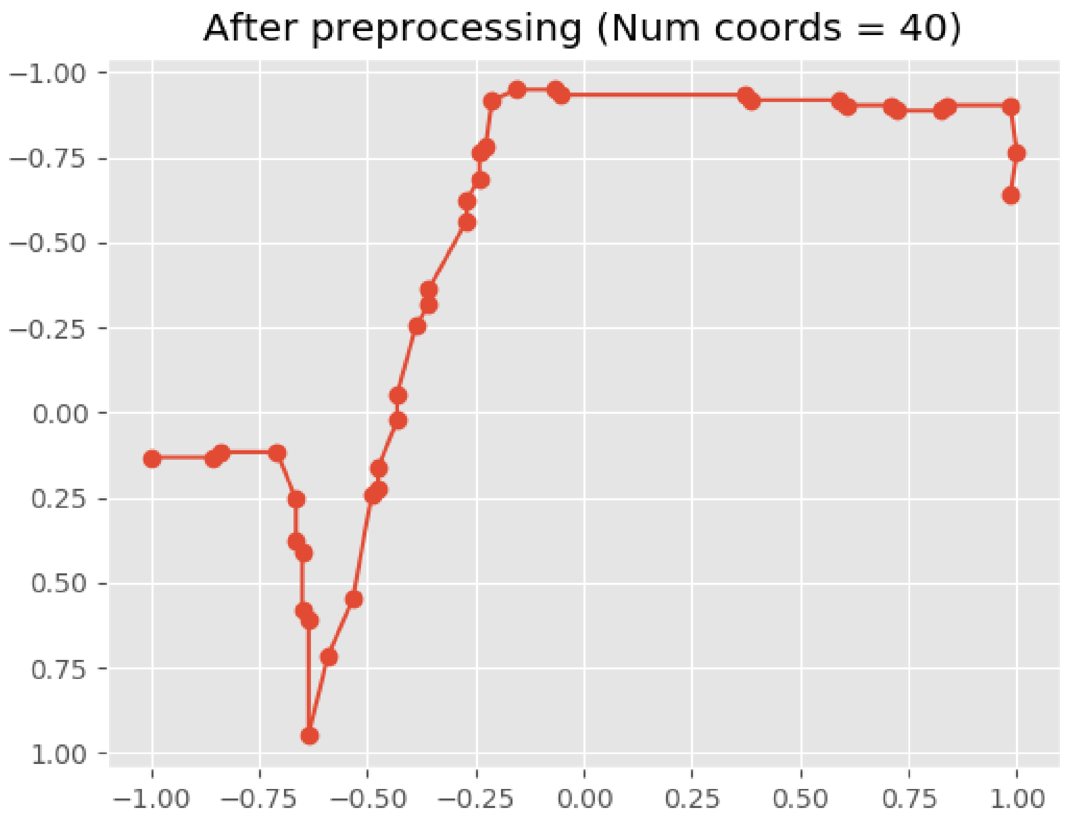


FIGURE 3.2: A square root which has been through preprocessing.

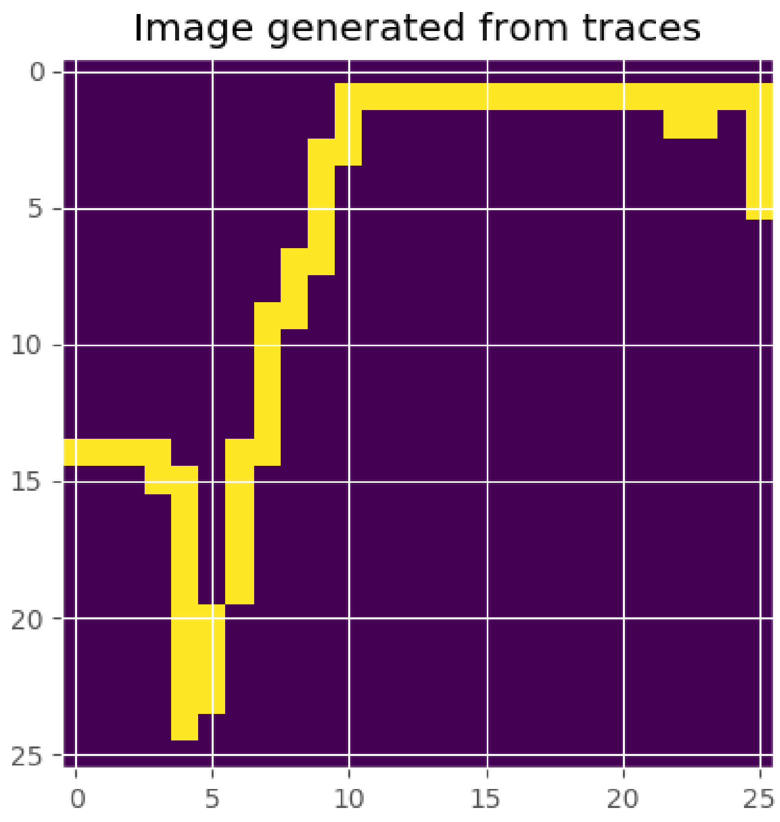


FIGURE 3.3: The resulting square root from the preprocessing done in previous steps. The generated image is 26x26 pixels.

3.9 Segmentation

Since the project initially went for a solution with a CNN, we had to separate the system into different tasks. The reason behind this is that the CNN itself, in our solution should solely work on the classification of a symbol. Since our assignment is to handle symbols and expressions, we then need to extract single symbols from multiple symbols or expressions.

This specific task was and is the most critical in order to get correct recognition with a CNN. If the segmentation turned out wrong, the classification would be handed bad data which makes a correct classification unlikely.

To solve the segmentation issue different approaches was attempted, among those were object localization and detection, with bounding boxes to easily detect symbols which consists of multiple strokes or traces. This idea of object detection was quite good and would have, if successful made the rest quite easy in comparison.

The attempts made on object detection was unsuccessful, we were getting results, but they were not proving to be better than solving it with a much simpler algorithm which did not require machine learning in the first place.

3.10 Interpretation and context search

The interpretation system consist of a recursive search function and a set of fixed rules to determine the context of how the symbols fit together. This system receives a list of the classified and segmented symbols from the classification step.

[Model of the overall architecture]

3.11 Project process

The project process in terms of software engineering methodology had been a mix between multiple concepts and paradigms. In the early stages of the project methodologies from the agile world were mostly used, such as pair programming. In the early stages of the project, we followed to some extent a methodology called Lean startup. Lean startup focuses on creating a minimal viable product, often referred to as MVP. When the MVP was out, constructive discussions with the product owners about what they liked and what they did not like. This assignment required a lot of curiosity and as project participants, one could never go hungry. This is why, at some stage the project split in different ways, we were continuously working on improving the product, but the main focus was on how we could do the recognition itself better.

3.12 Teamwork and roles

Since the project is more of a research based project, it was not easy to define a simple structure as more common in software engineering projects. Although, a simple structure was created to get the MVP up, this was Even on front end, Torkil and Håvard on back end.
With the MVP up, the project quickly took principles from agile thoughts and methodologies, even though we could not plan an entire sprint to detail, we had a stand-up meeting, or quick recap of the progress and issues. This enabled a dynamic and flexible structure which led us to working together quite effective, when the project stagnated, we would quickly help each other out. When it came to exploring the potential behind RNN it was simply not easy to grasp how to approach this issue in order to classify correctly. This took one of the project engineers several weeks to get going and when the

attempts were up for evaluation and the other project engineers understood the issue, a better solution and RNN was created.

Chapter 4

Results

4.1 Scientific results

This section will contain data, visualizations and results from our work towards our scientific problem.

4.2 Engineering results

This section will contain how our example project/proof of concept has turned out in terms of goals set in earlier stages of the project. Since our project is not a traditional software engineering bachelors project we will try to evaluate our project based on the goals set in some of the meetings with the product owners. The engineering goal in this project was to firstly create a proof of concept application which can display it's usefulness. Although the project owners did not set an lower bound for accuracy they insinuated that the application/module should be a resource for Matistikk. In the true spirit of an agile mindset the product was evaluated with our product owners continuously, to discuss improvements and changes. This also led to a stage in which the proof of concept application worked, looked and performed to be a resource for Matistikk. After becoming quite satisfied with how the application worked, blablabla....... bla bla bla

4.3 Administrative results

This section will contain results from an administrative perspective. This includes the planning, implementation and achievements of project goals. The nature of this assignment led the project first into a software development phase, closely followed by a research phase. Both phases was highly influenced by an agile mindset, although the software development phase was actually planned with sprints and reviews. Attempts were made on planning the research phase, but much of the time went into knowing artificial neural networks.

In the software development phase an agile mindset was always in the background, every critical decision and choice was discussed, this was an important aspect of the teamwork. The project required us to deliver fast and learn from our experiences to improve. If one of the project engineers did not understand or was not present when choices were made, it could lead to confusion and loss of progression. It made stand-up meetings, or a quick status meeting quite important.

Chapter 5

Discussion

5.1 Classification

5.2 Application

Chapter 6

Conclusion and further work

6.1 Conclusion

6.1.1 Application

6.1.2 Deep Learning

6.1.3 Data

6.2 Further Work

6.2.1 Data

6.2.2 Segmentation

Appendix A

System documentation

Glossary

InkML Ink Markup Language, defined by W3C. [9]. x, 11

layer . 13

MVP Minimal viable product. 37

nonlinear The results of a function is unable to be reproduced from a linear combination of the inputs. 25

WebSocket Enables event driven communication without. Client and server can push messages.. 32

Bibliography

[1] (). The MIT license — open source initiative, [Online]. Available: `https://opensource.org/licenses/MIT` (visited on 04/12/2018).

[2] M. Thoma, "On-line recognition of handwritten mathematical symbols", *arXiv:1511.09030 [cs]*, Nov. 29, 2015. DOI: `10.5445/IR/1000048047`. arXiv: `1511.09030`. [Online]. Available: `http://arxiv.org/abs/1511.09030` (visited on 03/06/2018).

[3] C. Lu and K. Mohan, "Recognition of online handwritten mathematical expressions using convolutional neural networks", p. 7, 2015.

[4] J. C. Simon, "Off-line cursive word recognition", *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1150–1161, Jul. 1992, ISSN: 0018-9219. DOI: `10.1109/5.156476`.

[5] S. Mori, C. Suen, and K. Yamamoto, "Historical review of OCR research and development", *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1029–1058, Jul. 1992, ISSN: 00189219. DOI: `10.1109/5.156468`. [Online]. Available: `http://ieeexplore.ieee.org/document/156468/` (visited on 04/10/2018).

[6] A. Priya, S. Mishra, S. Raj, S. Mandal, and S. Datta, "Online and offline character recognition: A survey", IEEE, Apr. 2016, pp. 0967–0970, ISBN: 978-1-5090-0396-9. DOI: `10.1109/ICCSP.2016.7754291`. [Online]. Available: `http://ieeexplore.ieee.org/document/7754291/` (visited on 04/10/2018).

[7] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain, "ICFHR2016 CROHME: Competition on recognition of online handwritten mathematical expressions", in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Oct. 2016, pp. 607–612. DOI: `10.1109/ICFHR.2016.0116`.

[8] H. Mouchère, R. Zanibbi, U. Garain, and C. Viard-Gaudin, "Advancing the state of the art for handwritten math recognition: The CROHME competitions, 2011–2014", *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 19, no. 2, pp. 173–189, Jun. 2016, ISSN: 1433-2833, 1433-2825. DOI: `10.1007/s10032-016-0263-5`. [Online]. Available: `http://link.springer.com/10.1007/s10032-016-0263-5` (visited on 04/10/2018).

[9] Y.-M. Chee, K. Franke, M. Froumentin, S. Madhvanath, J.-A. Magaña, G. Pakosz, G. Russell, S. Muthuselvam, G. Seni, C. Tremblay, and L. Yaeger. (Sep. 20, 2011). Ink markup language (InkML), Ink Markup Language (InkML), [Online]. Available: `https://www.w3.org/TR/InkML/` (visited on 03/06/2018).

[10] (2018). Introduction to LaTeX, [Online]. Available: `https://www.latex-project.org/about/` (visited on 04/11/2018).

[11] K. P. Murphy, *Machine learning: a probabilistic perspective*, ser. Adaptive computation and machine learning series. Cambridge, MA: MIT Press, 2012, 1067 pp., ISBN: 978-0-262-01802-9.

[12] B. Cline, R. S. Niculescu, D. Huffman, and B. Deckel, "Predictive maintenance applications for machine learning", in *2017 Annual Reliability and Maintainability Symposium (RAMS)*, Jan. 2017, pp. 1–7. DOI: `10.1109/RAM.2017.7889679`.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016.

[14] (). CS231n convolutional neural networks for visual recognition - neural networks 1, [Online]. Available: `http://cs231n.github.io/neural-networks-1/` (visited on 04/20/2018).

[15] (). Multi-layer neural network, stanfort.edu, [Online]. Available: `http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/`.

[16] S. W. Smith, "The scientist and engineer's guide to digital signal processing", in *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, 1997, ISBN: ISBN 0-9660176-3-3.

[17] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[18] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial", *Computer*, vol. 29, no. 3, pp. 31–44, Mar. 1996, ISSN: 0018-9162. DOI: `10.1109/2.485891`.

[19] A. Sharma, *Understanding Activation Functions in Deep Learning — Learn OpenCV.* 2018. [Online]. Available: `https://www.learnopencv.com/understanding-activation-functions-in-deep-learning/`.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[21] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton, "On rectified linear units for speech processing", in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 3517–3521. DOI: `10.1109/ICASSP.2013.6638312`.

[22] (). CS231n convolutional neural networks for visual recognition - linear classify, [Online]. Available: `http://cs231n.github.io/linear-classify/` (visited on 04/23/2018).

[23] (Sep. 4, 2018). Neural networks part 1: Setting up the architecture, cs231n.github.io, [Online]. Available: `http://cs231n.github.io/neural-networks-1/`.

[24] P. Werbos and P. J. (Paul John, "Beyond regression : New tools for prediction and analysis in the behavioral sciences /", 1974.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, p. 533, Oct. 9, 1986. [Online]. Available: `http://dx.doi.org/10.1038/323533a0`.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overtting", p. 30, Jun. 2014.

[27] E. W. Weisstein. (). Eigen decomposition, [Online]. Available: `http://mathworld.wolfram.com/EigenDecomposition.html` (visited on 04/25/2018).

[28] M. A. Nielsen, *Neural Networks and Deep Learning.* Determination Press, 2015. [Online]. Available: `http://neuralnetworksanddeeplearning.com` (visited on 03/06/2018).

[29] Y. T. Zhou and R. Chellappa, "Computation of optical flow using a neural network", in *IEEE 1988 International Conference on Neural Networks*, Jul. 1988, 71–78 vol.2. DOI: `10.1109/ICNN.1988.23914`.

[30] F. Chollet *et al.*, *Keras.* 2015. [Online]. Available: `https://keras.io`.