HoGent

BUSINESS AND INFORMATION MANAGEMENT

Professional Bachelor in Applied Computer Science
Academic year 2012-2013

Solving CAPTCHA using neural networks

Submitted on 10 June 2013

Student:
Pieter Van Eeckhout

Mentor:
Johan Van Schoor

HoGent Business & Information Management
Professional Bachelor in Applied Computer Science
Academic year 2012-2013

Solving CAPTCHA using neural networks

Submitted on 10 June 2013

Student:
Pieter Van Eeckhout

Mentor:
Johan Van Schoor

Contents

Abstract

TODO

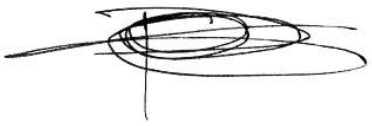
Preamble

First, dear reader, I would like to thank you for taking the time to read this thesis. Without an audience this entire endeavour would not mean as much as it does right now, while you are reading its results. I personally believe this is because I would like my life not to go unnoticed. So if this thesis helps, or influences you in any way, then this work has gained more meaning.
Second I would like to thank the following people who have made it possible for me to arrive at this point. Special thanks and mentions go to:

- my parents, for supporting me and giving me the opportunity and supplying the means for me to pursue my academic career.
- my girlfriend, Anne Charlotte Magdaraog Mendoza. Because she has helped me countless times through the rough spots. Not once did she complain about the time consuming job of writing this work.
- my good friends, willing proof readers and content critics: Wouter Dekens, Patrick Van Brussel and Thijs van der Burgt.
- Johan Van Schoor and Bert Van Vreckem for the support, organisation, guidance and feedback.

Bare in mind that this is not an exclusive list. Finally I would like to thank all the other people who are not mentioned by name: such as the teaching and support staff at University College Ghent.

Ghent BELGIUM, June 2013



Pieter Van Eeckhout

Chapter 1

Solving CAPTCHA using neural networks

The target audience. This thesis was written with an audience in mind that already has some technical understanding of computers and how they operate on hardware level (processor etc.). If you feel that your current knowledge is insufficient, or just want to read up some more, then I refer you to the "How Computers Work - Processor and Main Memory" [Young, 2001] e-book.

The history of SPAM. Ever since the internet found its way into our daily life, there have been people out there who don't always have other people's best interest in mind. I am referring to spammers, people aiming to advertise their product, services, etc . . . in an aggressive manner. The methods of advertising include but are not limited to:

- Sending bulk emails without the recipients permission (SPAM).
- Posting irrelevant links and information on fora and various social media.
- Flooding chat channels with their links and information.

These emails, posts and messages inconvenience the end-users, requiring time to filter out the junk. The economic costs of SPAM has led to a decrease in the Japanese GDP by 500 billion Yen (3.78 billion Euro) in 2004 and were projected to reach a decrease of 1% of the total GDP by 2010 unless adequate countermeasures were taken [Ukai and Takemura, 2007]. [Khong, 2004] researched the economic arguments for regulating junk mails and the efficiency of these regulations.

Birth of CAPTCHA. The two previously mentioned researches signify the importance and impact of SPAM on our daily life. The users of the internet quickly tried to implement methods to prevent spammers from spreading their advertisements to the masses. Several prevention and detection methods and systems were developed successfully. These methods and mechanisms range from hidden text to invalid HTML tags, all used to confuse and interrupt automated programs. One of the methods developed to prevent SPAM is a CAPTCHA test. CAPTCHA is an acronym based on the word "capture" and stands for 'Completely Automated Public Turing test to tell Computers and Humans Apart'. An attempt to trademark the term was made by Carnegie Mellon University on 15 October 2004, but the application was eventually dropped on 12 April 2008

Spammers fight back. All these prevention and detection methods did not stop the spammers from trying to reach an audience as large as possible. The spammers rely on a large target audience because of the return rates being as low as 0.0023% [Cobb, 2003]. The spammers started to device ways to circumvent or break the existing systems in order to reach a large enough audience. One of these methods is solving CAPTCHA tests by making use of the adaptive learning and pattern recognizing capabilities of neural networks. These networks can be used to recognize letters from images with adversarial clutter. This is the area I will focus on in this thesis. This thesis will list some of the difficulties regarding the extraction of relevant data from a CAPTCHA and how to possibly overcome these difficulties. However the main focus will be on understanding the inner workings of neural networks and on searching for the types and configuration of neural networks best used for pattern recognition.

Chapter 2

Premise and research questions

2.1 Premise

The main objective of this thesis is to ascertain whether neural networks are capable of solving the current generation of CAPTCHA images. we will define the premise as following:

"Are neural networks a viable tool for solving the current generation of CAPTCHA?"

2.2 Research questions

The research can be divided into two separate subjects. If one was to develop software for automatic CAPTCHA solving, the following questions and problems would need to be addressed.

CAPTCHA:

- What are the different types of CAPTCHA?
- How can the distorted text be extracted?

Neural networks:

- How do neural networks operate?
- Which types of neural networks are well suited for pattern recognition?
- What network configuration would perform best?

General:

- How future proof would this solution be?
- Is there enough economic incentive to invest in development?

Chapter 3

Methodology

Research philosophy. TODO

Research approach. TODO

Data Analysis. TODO

Chapter 4

Corpus

4.1 CAPTCHA

4.1.1 CAPTCHA, an explanation

A CAPTCHA (pronounced /'kæp.ʧə/) is a type of challenge-response test that aims to make sure the response was made by a human. These tests are designed in such a manner that they should be easy to generate and grade by a computer, and at the same time be difficult for a computer to solve. Yet a human should be able to solve the test without much difficulty. If a test was solved successfully one can assume that the response was entered by a human.
These test are mostly found on sites where one would like to prevent the access to unwanted bots. This is because having lots of spam on a site or in a service can have real detrimental consequences for that site or service. This is because most contemporary interactive sites store and serve their content from a database. When a database gets filled up the site can become slow and sluggish, reducing the customer's experience. This is only one of the many useful applications of CAPTCHAs. On the other hand, legitimate users also need to solve these tests, so it requires them to perform an extra task before they can post their content, create an email or view a certain page. While this 'simple' extra task does not seem like a large barrier, it does inconvenience some people enough to prevent them from posting valid content. This problem becomes even more apparent when dealing with non-native speakers [Banday and Shah, 2011]. Protecting your site with a CAPTCHA can even have a detrimental effect on the conversion rates[1].

[1] http://www.seomoz.org/blog/captchas-affect-on-conversion-rates

4.1.2 The history of CAPTCHA

Moni Naor was the first one to think of the concept of CAPTCHA in 1996. He proposed that reverse Turing testing, as CAPTCHAs are often called, should consist of "tasks where humans excel in performing, but machines have a hard-time competing with the performance of a three year old child." Some of these tasks were [Naor, 1996]:

- gender recognition
- understanding facial expressions
- understanding handwriting
- filling in words

In 1997 'Yahoo!' was having a gargantuan problem with spammers using bots to create free email addresses used to spread a huge amount of unwanted advertisement, giving Yahoo email adresses a bad reputation. 'Yahoo!' contacted Carnegie Mellon University[2] for help, by 2000 the first real CAPTCHA as we know them was invented [Egen, 2009]. These were also the people who first used the term "CAPTCHA" and tried to trademark it.
As computing power increased, so did the amount of CAPTCHA tests being broken. By 2008 there was an 30% to 60% success rate on the most used CAPTCHA systems [Yan and El Ahmad, 2008]. As a response to this Von Ahn and his team at Carnegie Mellon University released reCAPTCHA (Figure 2, page 42) in September 2008, a popular system which is still in use.
CAPTCHAs have always undergone changes once it became clear a certain generation method didn't stop the spammers any more. The first CAPTCHAs generated by EZ-Gimpy for 'Yahoo!' looked completely different from the CAPTCHAs that are currently being generated. A good example of the adaptive nature of CAPTCHAs is reCAPTCHA, were you can see the changes depending on when a CAPTCHA was generated. (Figure 3, page 42)

4.1.3 Types of CAPTCHA

Following is a list and description of the different types of CAPTCHA, courtesy of [Sauer and Hochheiser, 2008].

Character based In this category a string of characters is presented to the user. This string can contain either words or random alphanumeric characters. The task is to identify the string of characters.

[2] http://www.cylab.cmu.edu/research/projects/2008/captcha-project.html

Image based In this category images or pictures are presented to the user. This is normally in the form of an identifiable real-world object, but can also be presented in the form of shapes. The task is to identify the object shown in the picture.

Anomaly based In this category a series of different objects, shapes, characters,... is presented to the user. The task is to determine which object, character or shape does not belong in a set of images displayed on the screen.

Recognition based In this category all previous categories can be used. The user is tasked to determine what is being presented to them and respond accordingly.

Sound based In this category an audio version of a CAPTCHA is presented. The task is to identify the words and letters or image presented to the user.

4.1.4 Data extraction

As previously stated, the data extraction part of solving CAPTCHAs is not the main focus of this thesis. Therefore I will not give in-depth explanations of the algorithms used and described here.

CAPTCHAs are by design tough to solve for a computer. The majority of times a CAPTCHA gets cluttered with noise, or the letters get crowed together. This crowding or noise makes it so that the characters on the image are not separate entities. This is to impede the segmentation of the CAPTCHA. Measures against segmentation are necessary to prevent an OCR[3] algorithm from simply reading and solving the test. This could be possible, as computers can (given the right algorithms) be very efficient at pattern recognition. People trying to solve the CAPTCHA test automatically, have to separate the individual characters first before they can pass the characters to an OCR algorithm for classification.
[Yan and El Ahmad, 2008] described a working segmentation algorithm in 2008, but [Huang et al., 2010] has significantly improved on the performance, so it should be able to segment the contemporary CAPTCHAs.
In the unlikely case that the CAPTCHAs you are trying to solve don't have the segmentation issues, then you can first try to reduce the noise and then segment the characters by using the flood-fill method, as described by [Cai, 2008].

[3] Optical Character Recognition

4.1.5 The future of CAPTCHA

The arms race between the makers of CAPTCHA systems and people trying to break them favoured the defender. This is different from other computer security arms races, where the odds are in favour of the adversary. This is because CAPTCHA has broken the traditional pattern where the attacker's role is to generate new instances while the defender must recognize them, recognizing a problem is almost always harder than generating them. Websites and services using CAPTCHA can easily change the CAPTCHA generation algorithm, creating new unsolvable CAPTCHAs, while the attackers now have the challenging recognition problem. This battle has brought advances to the field of Automated Pattern Recognition and Artificial Intelligence. Some people even believe that eventually the solving algorithms will become so sophisticated they could be classified as a sentient AI[4] (Figure 1, page 42).
All the positive aspects and technological innovations aside, CAPTCHAs are inherently flawed. As the solving agents got better, the CAPTCHAs became harder. We have reached the point where the average user is having difficulties solving the standard CAPTCHAs[5].

CAPTCHA of the future will need to explore completely new test systems. As an example of this, [Sauer and Hochheiser, 2008] and colleges did a small research about how the current CAPTCHA (even the audio CAPTCHA) has serious shortcomings when trying to accommodate for blind or visually impaired users. They suggest a new system were the sound and image part of the test are integrated, opposed to the current system where the audio part and the visual part are on independent development and maintenance paths. With their suggested test all visually impaired and hearing impaired users should be able to solve the test.

But even the newly developed systems will eventually succumb to the ever increasing computing power [Beede, 2010]. The question is whether CAPTCHAs are the right way to prevent spammers, because how many "unsolvable CAPTCHA" (Figure 4, page 43) is a user going to tolerate before giving up? The malcontent of some has even led to the creation of intentionally unsolvable CAPTCHAs (Figure 5, page 44) through a service called "CRAPCHA"[6], aimed at ridiculing the real CAPTCHA services.

> It would seem evident from years of use and research that CAPTCHAs are far from perfect as a solution. Remove spammers from the equa-

[4] http://thenextweb.com/2009/10/15/inevitable-future-captcha/

[5] http://www.internetevolution.com/author.asp?section_id=587&doc_id=259406

[6] http://crapcha.com/

> tion and we remove the need for CAPTCHAs entirely; this is the mentality we should be aiming for. The perfect CAPTCHA is no CAPTCHA at all. [Bushell, 2011]

4.2 Neural Networks

The following section is a synthesis of 2 books describing neural networks, so similarities in structure and wording will be found. However, no malicious copyright infringement is intended. The two books used are "Neural Networks" by David Kriesel [Kriesel, 2013] and "Introduction to neural networks with Java" by Jeff Heaton [Heaton, 2005].

4.2.1 The concept of time

When we describe time in the context of neural networks, we denominate the number of cycles of the neural network. In this context time is split into discrete steps.

Definition 1. (The Concept of time). We reference the current time as (t) the next step in time is referred to as $(t+1)$ and the previous step as $(t-1)$. The other steps further ahead or behind are referenced analogously.

When we refer to a certain point in time for mathematical variables (e.g. $net_j or o_i$) we will do this as following $net_j(t-1) or o_i(t+3)$.

4.2.2 The components of an artificial neural network

An artificial neural network is made up by basic processing units, the *neurons*, and directed weighted connections between these neurons. The strength of the connection is called the weight, this is expressed as $w_{i,j}$ for the connection between neuron i and neuron j

Definition 2. (Neural network). A neural network is a sorted triple (N, V, w) which contains two sets N, V and a function w. N is a set of neurons, V is a set of connections where for each connections between neuron i and j applies $\{(i,j)|i,j \in \mathbb{N}\}$, w is a function $(w : V \to \mathbb{R})$ defining the weights. Where $w((i,j))$ defines the weight for the connection between neuron i and neuron j, shortened to $w_{i,j}$.

Depending on the implementation, a connection int the network does not exist or has no influence when the weight is undefined or equal to zero. The weight can be combined in a square *weight matrix* W, or in a *weight vector* W. The row number indicates where a connection begins and the column number where it ends. In this representation, a zero indicates a non-existing or inactive connection. This kind of representation is also called a *Hinton Diagram* (Figure 6, page 45).

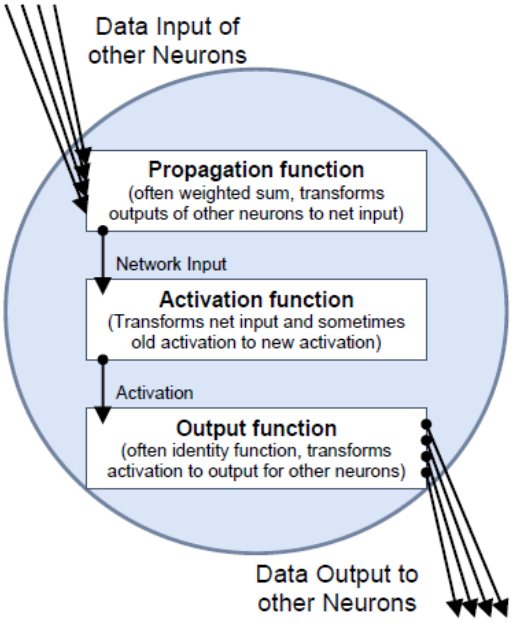


Figure 4.1: Neuron Data processing (Source: [Kriesel, 2013] page 39, Figure 3.1)

Neuron Data Processing

Upon closer inspection, we can see that the neuron and its connections are made up out of several other distinguishable components and processes. which we will discuss more in-depth in the following sections.

Propagation function

In a network there will likely be more then one neuron linking to neuron j. Which means that each neuron one of these neurons will pass its output to neuron j. The propagation function receives all these outputs $o_{i_1}, \ldots, o_{i_n}$ of the neurons $i_1, i_2, \ldots, i_n$ and combines those into a single value, the *network input* net_j, based on the connection weights $w_{i,j}$.

Definition 3. (Propagation function and network input) The network input, or net_j, can be calculated with the propagation function f_{prop} as follows:

$$net_j = f_{prop}(o_{i_1}, \ldots, o_{i_n}, w_{i_1,j}, w_{i_n,j}) \tag{4.1}$$

Where $I = \{i_1, i_2, \ldots, i_n\}$ is the set of neurons so that $\forall z \in \{1, \ldots, n\} : \exists w_{i_z,j}$. The *weighted sum* is a very popular, and the most used propagation function.

$$net_j = \sum_{i \in I} (o_i \cdot w_{i,j}) \tag{4.2}$$

Neuron threshold value

Definition 4. (Threshold value in general). The threshold value Θ_j for neuron j is uniquely assigned to j and defines the position of the maximum gradient of its activation function.

This means that the activation function of a neuron will react very sensitive to changes when net_j is around the threshold value.

Neuron activation function.

Activation status. Neurons can change in activation status. A neuron's activations state depends on the net_j, the previous activation state and the activation function. A neurons activation state is important, because this will influence a neuron's reaction to the net_j.

Activation function. The activation, also called the transfer function, a_j of neuron j depends on on previous states, so this will change overtime. The activation function uses the network input net_j, the threshold value and the previous activation state $a_j(t-1)$ to create a new *activation state*. The neuron activation function is also called the identity.

Definition 5. (activation and activation function). In neuron j the activation function can be described as

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \Theta_j) \tag{4.3}$$

Binary threshold function. the binary threshold function, also called the *Heaviside function*, is the simplest activation function as can only take on two values. The activation will change, depending on whether the input value is above or below the threshold value. In either case the function changes from one value to another. This activation function doesn't have a variable, hence it's derivative is 0. This makes back-propagation learning impossible

Fermi function. The Fermi function, also called the logistic function, uses following equation. from the equation we can derive that the Fermi function maps a range of values between $(0, 1)$. It's function is describe as following:

$$\frac{1}{1+e^{-x}} \tag{4.4}$$

hyperbolic tangent. The hyperbolic tangent function maps to values within the range of $(-1, -)$. It's function is describe as following:

$$\tan(x) \tag{4.5}$$

Fermi function with temperature parameter. The Fermi function can be expanded by adding a temperature parameter. The smaller this parameter becomes the colder the activation function becomes, the steeper the gradient of the activation function becomes. And visa versa. By making the temperature parameter very small, we can emulate the Heaviside activation function's behaviour. The extended Fermi activation is described as following:

$$\frac{1}{1 + e^{\frac{-x}{T}}} \tag{4.6}$$

The bias neuron

The bias neuron (also called the on neuron) is a technical trick. the purpose of a bias neuron is to represent a neurons threshold value as a connection weight. As said before the activation of a neuron depends on its threshold value, however theses threshold values are stored inside the neuron itself. This makes it complicated to access and use in the activation function, or to train the threshold value. This is where the bias neuron comes in.

Threshold values $\Theta_{j_i}, \ldots, \Theta_{j_n}$ for neurons $j_1, j_2, \ldots, j_n$ can be expressed as a connection weight of a constantly firing neuron. This constantly firing neuron is the bias neuron. To implement this change, an additional neuron, whose output is always 1, will be added tot the network and connected to neurons $j_1, j_2, \ldots, j_n$. The newly made connections' weights will be set to the negative threshold values. $w_{BIAS,j_n} = -\Theta_{j_n}$

Now that the threshold values are implemented as connection weight, the threshold value Θ_{j_n} of neurons $j_1, j_2, \ldots, j_n$ is then set to 0. This also means that the threshold value can be directly trained together with the connection weight. This makes the learning process considerably easier. Since the threshold value is no longer included in the activation function, its effect is now included in the propagation function.

The advantages of the bias neuron are obvious when implementing and training a neural network. However they do have a disadvantage when you want to visually represent the neural network. these extra connection quickly clutter up the visual representation, and this becomes only worse when a lot of neurons are being

used. But since the implementation of a bias neuron is almost a certainty, a lot of people omit the bias neuron from their illustrations for improved clarity. We know it exist and know that threshold values can simply be treated as weights because of it.

Definition 6. (Bias neuron). A bias neuron is a neuron whose output value is always 1 and is used to represent neuron biases as connection weights, which enables any weight training algorithm to train the biases at the same time.

Let $j_1, j_2, \ldots, j_n$ be neurons with threshold values $\Theta_{j_1}, \ldots, \Theta_{j_n}$. By inserting a bias neuron whose output value is always 1, generating connections between the bias neuron and neurons $j_1, j_2, \ldots, j_n$ and weighting these connections $w_{BIAS,j_1}, \ldots, w_{BIAS,j_n}$ with $\Theta_{j_1}, \ldots, \Theta_{j_n}$, we can set $\Theta_{j_1} = \ldots = \Theta_{j_n} = 0$ and receive an equivalent neural network whose threshold values are realized by connection weights.

Neuron activation order.

The sequence in which the neurons of a neural network receive and process the input is critical for the result.

Synchronous activation. In this activation scheme all neurons update at the same time. They simultaneously calculate the input, activation and output. This type of activation scheme closest resembles biological neural networks. However implementing this is only possible on hardware capable of parallel processing. This is the most generic activation scheme and can be used for any network, independent of network topology. However feedforward networks would not benefit from this scheme.

Asynchronous activation. the counterpart of synchronous activation. The neurons won't fire all at once in this scheme, but in different points in time. Several subcategories can be discerned.

Random order activation. In this activation scheme, a random neuron i is chosen from all the neurons and for that neuron the input. For an n neuron network, the activation cycle will end after having update n neurons. During this random cycle some neurons can be update multiple times, while others don't get updated at all. It is for this reason that this order of activation is not always useful.

Random permutation activation. In this scheme a list of each neuron in a randomized order is made. Therefore during one cycle, each neuron will be activated exactly once. But in a random order.
This activation order doesn't get used much, just like random order activation. This is because of the huge overhead generated by having to build a random order list of each neuron for each activation cycle. While Hopfield network topology technically should be updated at random, in practice a fixed order activation is preferred.

Topological order activation. In this scheme the neurons are update in a fixed order. The order is defined by the network topology. First the input neurons get activated then the layer after that, and so on until the output neurons are reached. Due to the definition, this activation order cannot be used for recurrent networks as they don't have a clearly defined start or end. This is the most efficient activation schema when comes to feedforward networks.

Fixed order activation. In this scheme the order of activation gets set during implementation. This can be useful in some cases (again feedforward networks come to mind), but in case a network can change topology during runtime, then this activation order will have detrimental effects, because it cannot and will not change based on the new topology.

Neural output function

The output function calculates what value neuron i will pass to its connected neurons j. The output value o_j of neuron j is calculated from its activation state a_j.

Definition 7. (Output function). For neuron j the output function is formally described as

$$f_{out}(a_j) = o_j \tag{4.7}$$

Generally though the output functions is often the same as the activation a_j, which is directly output.

$$f_{out}(a_j) = a_j, SOo_j = a_j \tag{4.8}$$

Input and output of data

The purpose of a neural network is to receive, process and return data. The input components for a network with n input neurons can be scribed by vector $x = (x_1, x_2, \ldots, x_n)$ and the output components for a network with m output

neurons can be scribed by vector $y = (y_1, y_2, \ldots, y_m)$. this leads us to following definitions:

Definition 8. (Input vector). A network with n input neurons needs n input values $x_1, x_2, \ldots, x_n$ they can be combined in an input vector $x = (x_1, x_2, \ldots, x_n)$.

Definition 9. (Output vector). A network with m output neurons has m output values $y_1, y_2, \ldots, y_m$ they can be combined in an input vector $y = (y_1, y_2, \ldots, y_m)$.

4.2.3 Neural network topologies

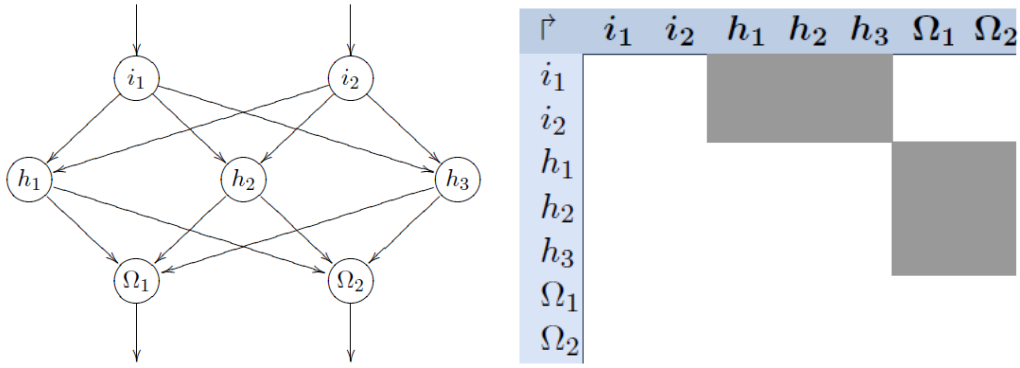
Until now, we have only looked at the different elements that make up a neuron. But we can only do so much with only one neuron. It is time to zoom out and start looking at how several neurons can be organized together to effectively make a neural network. The following is a list of the usual network designs. Every topology described will be accompanied by a visual representation a Hinton diagram, courtesy of [Kriesel, 2013]. The Hinton diagram is supplied so that the characteristics of the network can immediately be seen.

For the transformation from the visual representation tot the Hinton diagram, following legend will be used:

- dotted weight connections will be represented as light gray fields
- solid weight connections will be represented as dark grey fields
- the directional arrows cannot be added. So to express that connections start from the line neurons and end at the column neurons, the ↱ symbol has been introduced in the upper left corner.

Feedforward networks

standard. The first topology we will discuss is the one of the standard feedforward network. In a feedforward network the neurons are divided into layers. There layers can be grouped in three types of layers: one *input layer*, n *hidden layers* and one *output layer*. The hidden layers cannot be seen from the outside (hence the name hidden), and the neurons in those layers are referred to as hidden neurons. The hidden layers are also often referred to as the processing layers. In a feedforward network, a neuron has only connections towards the next layer. These connections all are directed towards the output layer. In many cases we will find that each neuron i is connected to every neuron j of the next layer. These networks are called completely linked feedforward networks. As a naming convention, the output neurons will often be referred to as Ω. (Figure 4.2)



(a) Standard feedforward topology **(b)** Standard feedforward Hinton diagram

Figure 4.2: Standard feedforward network; Source:[Kriesel, 2013]

Definition 10. (Feedforward network).

A feedforward network consists of three types of layers, one input layer, one output layer and optionally one or more processing layers. These layers are clearly separated. Connections only go from one neuron layer to the next, directed towards the output layer.

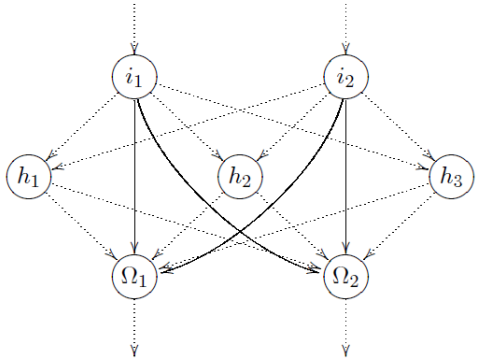
Shortcut connections. This is essentially a standard feedforward network (see 4.2.3) where connections exist that skip one or more levels. these so-called shortcut connections may only be directed towards the output layer. (Figure 4.3)

Definition 11. (Shortcut feedforward). A shortcut feedforward network consists of three types of layers, one input layer, one output layer and optionally one or more processing layers. These layers are clearly separated. Connections only exist directed towards the output layer, the connections however may span over one or more layers.

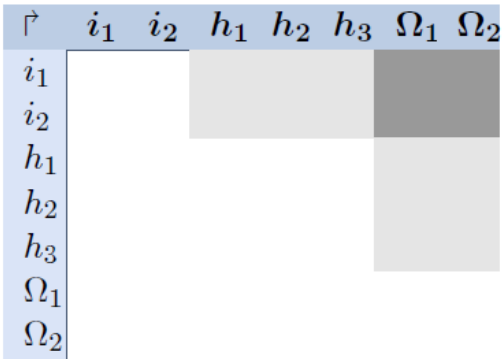
Recurrent networks

Recurrence is the event where a neuron influences itself. This can be done through any means or by any connection. It is because of this that recurrent networks often don't have explicitly defined input and output neurons.

Direct recurrent. Direct recurrence happens in a network when neurons feed their output back to themselves as input. This is also often called self-recurrence.

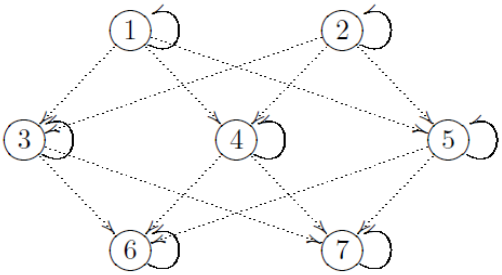


(a) Shortcut feedforward topology

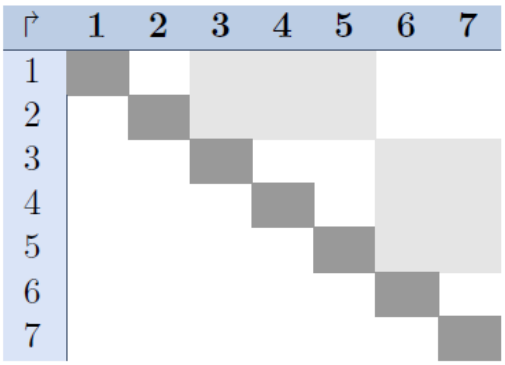


(b) Shortcut feedforward Hinton diagram

Figure 4.3: Shortcut feedforward network; Source:[Kriesel, 2013]



(a) Direct recurrent topology



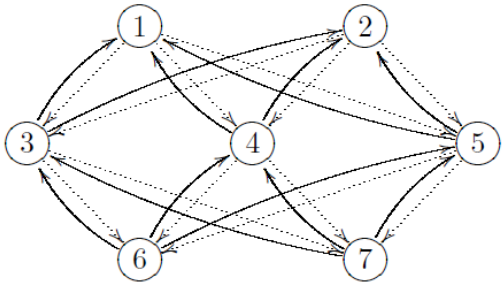
(b) Direct recurrent Hinton diagram; Source:[Kriesel, 2013]

Figure 4.4: Direct recurrent network

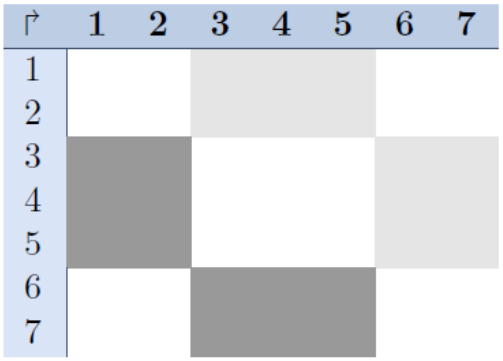
The neurons inhibit themselves, and as a result of this strengthen themselves in order to reach their activation limits. (Figure 4.4)

Definition 12. (Direct recurrence). A direct recurrence network is a network where a neuron j is connected to itself. the connection weight is then defined as $w_{j,j}$. The diagonal of weight matrix W will no longer be zero as a direct result from this.

Indirect recurrent. Indirect recurrence happens when in a network connections towards the input layer are allowed. A neuron can influence itself by passing its output to one of the previous or one of the next layers. (Figure 4.5)



(a) Indirect recurrent topology



(b) Indirect recurrent Hinton diagram; Source:[Kriesel, 2013]

Figure 4.5: Indirect recurrent network

Definition 13. (Indirect recurrence). An indirect recurrent network is a network where connections forward and backward in the network may exist. The neurons influence themselves by influencing the preceding layers.

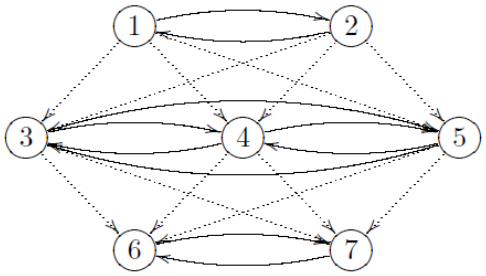
Lateral recurrent. Lateral recurrence happens when connections between neurons of the same layer exist. These connections often make it so that each neuron inhibits the other neurons and strengthens itself. This often means that only the strongest neuron becomes active (*winner-takes-all scheme*). (Figure 4.6)

Definition 14. (Lateral recurrence). A lateral recurrent network is a network where connections between the neurons of the same layer are allowed.

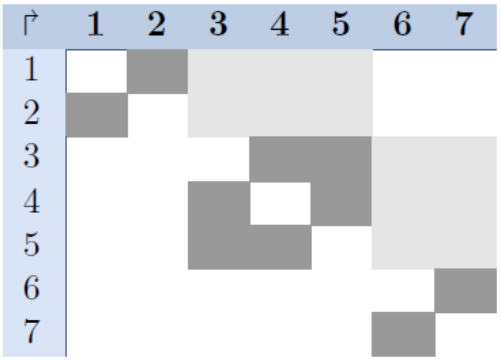
Completely linked networks

A completely linked network happens when all neuron connections are allowed, except for direct recurrent connections. All neuron connections must be symmetric, this means that $w_{i,j} = w_{j,i}$. As a direct result of the complete linking, input and output neurons can no longer be strictly defined and clearly defined layers do not longer exist. Another result is that the weight matrix W can be different from zero everywhere, except along the diagonal. (Figure 4.7)

Definition 15. (Complete interconnection). A complete interconnected network is a network where each neuron is always connected to every other neuron. As a result ever neuron can become an input neuron.

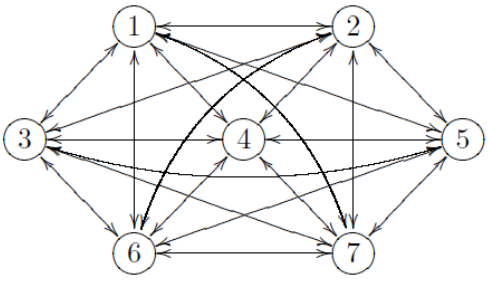


(a) Lateral recurrent topology

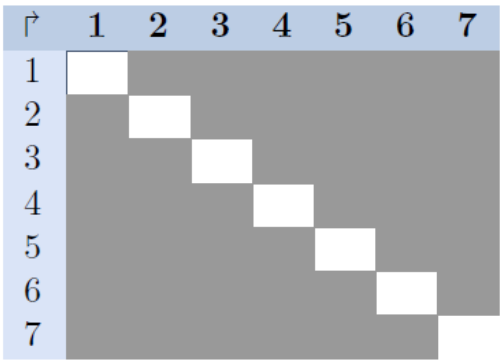


(b) Lateral recurrent Hinton diagram; Source:[Kriesel, 2013]

Figure 4.6: Lateral recurrent network



(a) Completely linked topology



(b) Completely linked Hinton diagram; Source:[Kriesel, 2013]

Figure 4.7: Completely linked network

4.2.4 Neural network learning

One of the most interesting features of a neural network is the ability to familiarize themselves with a problem by means of training. This means that after sufficient training, the network will be able to solve unknown problems of the same type. This is called generalization. Training a network involves some form of leaning procedure. A learning procedure is a rather complex thing, so a brief explanation will be giving first.

Learning paradigms. We must be careful with the term "learning" as it is a very comprehensive term. A system learns when it changes itself in order to adapt to changes in the systems environment. A neural network as a learning systems changes, learns, whenever its components change. So any change in the components previously mentioned due to environmental changes will result in a changing network. this process is classified as neural network learning. Looking ad the previous components capable of changing we can form a list of all theoretically possible means of neural network learning.

1. developing new connections
2. deleting existing connections
3. changing connecting weights
4. changing the threshold values of neurons
5. varying one or more of the three neuron functions (activation function, propagation function and output function)
6. developing new neurons
7. deleting existing neurons (and so its existing connections)

We can safely assume that changing the connection weights will be the most commonly used method of training. Especially because of the simple fact that deleting a connection (no longer training a connection with weight) and creating new connections (changing the connection weight from 0) in the connection matrix can be done by connection weight adjustment training. With the introduction of the bias neuron (4.2.2) the threshold values can also be changed by using connection weight training. We can perform any of the first four of the learning paradigms by just training synaptic weights.

Changin neuron functions is difficult to implement and not very intuitive. Because of this, those paradigms aren't really popular and will not be discussed

any further. The option of generating new, or removing old neurons have a few advantages. This method not only provides a way to keep the connection weights well adjusted during the training of a neural network, it also optimizes the neural network topology. For this reason this type of training is gathering a growing amount of interest. A method of achieving this functionality would be evolutionary algorithms. However we will not elaborate any further on this since this is outside the scope of this thesis, and since we already covered that the majority of learning methods can be achieved by weight training.

To summarize, neural network learn by modifying the connecting weights according to rules formulated as algorithms. Therefore a learning procedure is always an algorithm that can easily be implemented by means of a programming language.

Training patterns, teaching input, desired output and error vector

Definition 16. (Desired output). The expected result of the output layer when presented with a training pattern.

Definition 17. (Teaching input). For an output neuron j the teaching input t_j is desired and correct value j should have after the input of a training pattern. The teaching inputs $t_1, t_2, \ldots, t_n$ van be combined into a vector t. t always refers to a specific training pattern p. this correlation is contained in the training set.

Definition 18. (Training pattern). A training pattern is an input vector p with the components $p_1, p_2, \ldots, p_n$ whose desired output is known. By entering the training pattern into the network we receive an output that can be compared with the desired output.

Definition 19. (Training set). A training set (named P) is a finite set of pairs of training patterns (p) and teaching inputs (t). The finite ordered pairs (p, t) of training patterns and teaching inputs are used to train our neural network.

Definition 20. (Error vector).The error vector (also called the difference vector) for output neurons $\Omega_1, \Omega_2, \ldots, \Omega_n$ is the difference between the actual output vector and the teaching input for a training input p. Depending on whether you are learning offline or online, the difference vector refers to a specific training pattern, or to the error of a set of training patterns which is normalized in a certain way.

$$E_p = \begin{pmatrix} t_1 - y_1 \\ t_2 - y_2 \\ \vdots \\ t_n - y_n \end{pmatrix}$$

Unsupervised learning. Unsupervised learning is the the type of learning closest related to how biological neural networks operate, yet this type of learning is not suitable for all problems. In this learning type only the training patterns are given, the network itself is taxed with trying to identify and classify similar patterns.

Definition 21. (Unsupervised learning). The training set only consists of input patterns, the network tries by itself to detect similarities and to generate pattern classes.

Reinforcement learning. Reinforcement learning uses only the training patterns, but after each cycle a logical or real value is returned to it. This value indicates whether the output was correct or not.

Definition 22. (Reinforcement learning). The training set consists of input patterns, after completion of a sequence a value is returned to the network indicating whether the result was right or wrong and, possibly, how right or wrong it was.

Supervised learning. Supervised learning uses a training set P consisting of both the trainings pattern (p) and teaching input (t) in form of the precise activation of all output neurons Ω. The network is fed with the trainings patterns, and the connection weights then are adjusted based on the error vector obtained from subtracting the actual output with the teaching input. The goals is to change the weights in such a matter that the network cannot only associate input and output patterns independently after the training, but that is can also provide plausible results to unknown, similar input patterns. after training the network should be able to generalise.

Definition 23. (Supervised learning). The training set consists of training patterns and teaching inputs so that the network can calculate a precise error vector to adjust the network with.

Offline learning or online learning. Offline learning is a method that stands in stark contrast with online learning.
In offline learning you present all training patterns at once and the total error is calculated with an error function, or simply accumulated. Only after all patterns have been presented will the weights be changed. Offline training procedures are also called batch training procedures, since a batch of results is corrected in one go. on such iteration of a batch training process is called epoch.
In online learning the weights are changed after every pattern presented.

Each method method has advantages and disadvantages, and it isn't always clear what type is best to be used.

Definition 24. (Offline learning). Several training patterns are entered into the network at once, the errors are accumulated and it learns for all patterns at the same time.

Definition 25. (Online learning). The network learns directly from the errors of each training sample.

Training samples We should take a look at selecting the training data and the learning curve now that we know how to train a network. We should be particularly interested in whether our network has merely memorized the our training data (can it produce the right output for our trained inputs but to provide wrong answers for all other problems of the same class), or whether it can generalize the problem.

It can be that the network has sufficient storage capacity to store the training data, but only only recognizes the input data. This implies that the network is oversized and has to much free storage capacity. On the other it can be that the network has insufficient storage capacity, this in turn leads to bad generalization. It is important to find a balance.

For this purpose that the training set is often divided into a set used for training the network, and a set used for verification. Provided of course that there are enough training samples not to affect the initial quality of training. Th usual used ration is 70% for training data and 30% for verification. These sets should be randomly chosen. We can conclude the training of the network when both the results of the training data and the verification data are good. The verification data is included in the training, eventhough it is not directly used in training the network. What this means is that you do not modify the network to accommodate for the poor results of the verification data. If you do this, you run the risk of tailoring the network to the verification data. A possible solution is a third set of validation data used only for validation after a supposable successful training. The downside to keep in mind is that by raining less patterns, we obviously withhold information from the network and risk to worsen the learning performance.

Learning curve and error measurement A learning curve indicates the variations in the error. The motivation for creating a learning curve is to see whether

the network is progressing or not. The error must first be normalized before we can use it to plot out the learning curve. Normalizing the error is to represent the distance between the correct and current output of the network. There are four ways of calculating the normalized error. In the following sections Ω will be the output neurons and O the set of output neurons.

Specific error. The specific error Err_p is based on single training sample p. This means it is generated during online learning.

$$Err_p = \frac{1}{2}\sum_{\Omega \in O}(t_\Omega - y_\Omega)^2 \tag{4.9}$$

Root mean square. The root mean square (also referred to as RMS) of two vectors t and y is a error normalization method that can be used both for online and offline.

The online formula:

$$Err_p = \sqrt{\frac{\sum_{\Omega \in O}(t_\Omega - y_\Omega)^2}{|O|}} \tag{4.10}$$

The offline formula:

$$Err = \sum_{p \in P}\sqrt{\frac{\sum_{\Omega \in O}(t_\Omega - y_\Omega)^2}{|O|}} \tag{4.11}$$

Euclidean distance. Just like the root mean square the Euclidean distance calculates the error based on the two vectors t and y, and can be used both online and offline.

The online formula:

$$Err_p = \sqrt{\sum_{\Omega \in O}(t_\Omega - y_\Omega)^2} \tag{4.12}$$

The offline formula:

$$Err = \sum_{p \in P}\sqrt{\sum_{\Omega \in O}(t_\Omega - y_\Omega)^2} \tag{4.13}$$

This is just a small grasp of he most used error measuring methods, many more can be used. I you are interested in finding out more about error measurement please read the technical report of Prechelt [**?**]

Depending on the the method used our learning curve will look different. Normally a perfect learning curve will look like a negative exponential function. Since a learning curve will be proportional to e^{-t} can best be illustrated by using a logarithmic scale. In this scale a descending line indicates an exponential decrease of the error.

When to stop learning? Generally learning is stopped when the user "thinks" the error became small enough. There is no clear-cut answer for this question, but the analysis and comparison of learning curves can bring more clarity. You can also assume greater confidence, when a network reaches nearly the same final error-rate for different random initializations.

A learning curve descending fast can be overtaken by another curve. This can indicate that either the learning rate of the worse curve was too high or the worse curve itself simply got stuck in a local minimum, but was the first to find it. It is also important to plot the verification data on a second learning curve. This will often provide worse result, but with good generalization this curve can decrease too.

Preventing memorization of the sample data can be achieved by early stopping. An indication that early stopping might be beneficial is when the learning curve of the verification samples suddenly rises rapidly while the learning curve of the verification data is continuously falling

Hebbian rule The Hebbian learning rule acts as basis for most other complex learning rules. It was first described by *Donnald O. Hebb* [Hebb, 1949] and currently exists in 2 forms, the original rule and the generalized form.

The original rule.

Definition 26. (Hebbian rule). If neuron j receives an input from neuron i and if both neurons are strongly active at the same time, then increase the weight $w_{i,j}$, strengthening the connection.

$$\Delta w_{i,j} \sim \eta o_i a_j \tag{4.14}$$

With $\Delta w_{i,j}$ being the change in weight from i to j. This change in weight is simply added to the weight $w_{i,j}$. $\Delta w_{i,j}$ is dependent on following factors.

- the output o_i of predecessor neuron i
- the activation function of successor neuron j

- the constant η, the learning rate

Notice that the definitions speaks about the activation twice, while in the formula the activation is only used once. instead of the activation a_i of neuron i and the activationa_j of neuron j the formula uses the activation a_j of neuron j and the output o_j of neuron i. This is possible because the identity or activation of a neuron is often used as the output (4.2.2). Therefore a_i and o_i can be considerate as being the same value.

Generalized form.

Definition 27. (Hebbian rule,generalized). The generalized form of the Hebbian Rule only specifies the proportion of change in the weight based on the product of two undefined functions, but with defined input values.

$$\Delta w_{i,j} = \eta \cdot (h(o_i, w_{i,j}) \cdot g(a_j, t_j)) \tag{4.15}$$

With $\Delta w_{i,j}$ being the change in weight from i to j. This change in weight is simply added to the weight $w_{i,j}$. $\Delta w_{i,j}$ is dependent on following factors.

- $g(a_j, t_j)$: this function uses the actual activation a_j and the expected activation found in the teaching input t_j
- $h(o_i, w_{i,j})$: this function uses the output of the predecessor neuron o_i and the connection weight between the two neurons $w_{i,j}$
- the constant η, the learning rate

Since this is a generalized form, the functions g and h are not specified.

The learning rate. The speed and accuracy of a learning procedure are be controlled by and are always proportional to a learning rate (η). If this value is chosen too big, it is possible to miss possibly favourable narrow local minima in the error. However, setting it very small is not a good solution either as this would dramatically increase the time and computing cost of training a network. Experience has shown that learning rates between 0.01 and 0.9 are mostly optimal. A popular method for finding an optimal η is starting out high ans lowly decreasing it.

Definition 28. (Learning rate). The learning rate (η) dictates the speed and accuracy of a learning procedure.

4.3 Neural networks for pattern recognition

Now that we have a clear understanding of how neural networks work and can be trained, can we take a look at several neural network implementations. This thesis will focus on the network implementations most frequently found in the literature to be used for pattern recognition.

4.3.1 Perceptron networks

This is the classic example of neural network, often when the term "neural network" is used a perceptron network (or variation thereof) is meant. Perceptron networks were first described by Frank Rosenblatt in 1958 [**?**].

There is no set definition of a perceptron network, but it is often described as being a feedforward network with shortcut connections. The input layer of the the perceptron network is called the retina, and is only used for data input, no processing happens in the input layer as the weights for the input layer are static.

Single layer perceptron network

A single layer perceptron (or SLP for short) is a perceptron network that has only one layer (except the input neurons). This means that the trainable connection weights going from the input layer go straight towards one or more output neurons Ω. A single layer perceptrons with multiple outputs can simply be considered as several separate single layer perceptrons with the same input. The single layer perceptrons is because of the limitation to a single layer only able to represent linear data.

Single layer perceptrons and the delta rule. The delta rule is an extension on the Hebbian rule. It has the advantage that it can be used networks without binary activation functions, and that the network will automatically learn faster as it is farther away from the target. I refer you to [Kriesel, 2013] page 90-95 in case you want to read the whole mathematical and logical construction of the delta rule.

Definition 29. (Delta rule). The delta rule (also called the Widrow-Hoff rule) is based on the generalized Hebbian rule (4.2.4), where the function h only uses the output o_i of the predecessor neuron i, and where the function g is the difference between the desired activation t_Ω and the actual activation a_Ω of the

output neuron Ω. If we substitute this into the Hebbian rule we get he following:

$$\Delta w_{i,\Omega} = \eta \cdot o_i \cdot (t_\Omega - a_\Omega) = \eta o_i \delta_\Omega \tag{4.16}$$

If we were to use the desired output instead of the desired activation the formula would look like this:

$$\Delta w_{i,\Omega} = \eta \cdot o_i \cdot (t_\Omega - o_\Omega) = \eta o_i \delta_\Omega \tag{4.17}$$

These formulas can be used for online learning, in the case of batch processing you simply let the error accumulate

$$\Delta w_{i,\Omega} = \eta \cdot \sum_{p \ inP} o_{p,i} \cdot (t_{p,\Omega} - o_{p,\Omega}) = \eta \cdot \sum_{p \ inP} o_{p,i} \cdot \delta_{p,\Omega} \tag{4.18}$$

Multilayer perceptron network

Contrary to single layer perceptron can the multilayer perceptron (MLP for short). This achieved by taking a single layered perceptron with multiple output neurons and attaching other single layer perceptrons to this, to further process the already linear processed output from the output neurons of the base single layer perceptron. Each added layer supposedly adds the capability to process an extra dimension.

It has been mathematically proven that even a multilayer perceptron with only one layer of hidden neurons can approximate a functions with finitely many discontinuities as well as their first derivatives to an arbitrary precision. This proof can unfortunately not be used in the reverse way, so it still is up to us to find the correct amount of neurons to use in each situation.

When describing a multilayer perceptron comprising 7 input neurons, a hidden layer of 10 neurons, another hidden layer of 5 neurons and an output layer of 6 neurons the following abbreviation is often used: 7-10-5-6.
The numbers indicate the amount of neurons for that layer. The first layer is the input layer (retina) and the last is the output layer. The layers in between are the hidden layers, these are always order going from closest to the input layer (and farthest from the output) to farthest from the input layer (and closest to the output).

Word of warning for reading through the literature. Not all sources use the same way of counting the network layers, some include the retina, some count the trainable weight layers and others exclude the output neuron layer.

Multilayer perceptrons backpropagation. The backpropagation of error learning rule is used to train multi-staged neural perceptrons. Backpropagation is a gradient descent procedure with error function $Err(W)$ receiving all the connection weights as arguments and links them to the output error. This system is an extension of the delta rule. The output function is kept the same for each neuron, but the variable δ_i for each neuron i needs to be generalized from one trainable weight to several ones. Again, if you want to read the whole mathematical and logical construction, please read [Kriesel, 2013] pages 104-108.

Definition 30. (Backpropagation). Backpropagation is based on the generalized Hebbian rule (4.2.4), where the function h only uses the output o_i of the predecessor neuron i, and where the function g is the difference between the desired activation t_Ω and the actual activation a_Ω of the output neuron Ω. If we substitute this into the Hebbian rule we get he following:

$$\Delta w_{i,j} : \eta \cdot o_i \cdot \delta_j \text{with}$$
$$\delta_j = \begin{cases} f'_{act}(net_j) \cdot (t_j - y_j) & \text{(when } j \text{ is an output neuron)} \\ f'_{act}(net_j) \cdot \sum_{l \in L}(\delta_l w_{h,l})) & \text{(when } j \text{ is a hidden neuron)} \end{cases} \quad (4.19)$$

Backpropagation and the learning rate. It might be opportune to be able t change the learning rate over time during the training of a network. A larger learning rate might fork fine at the start, but nearing the end of the training procedure, we would to be able to increase the learning accuracy by decreasing the size of the learning rate. It is a bad idea to reduce the learning rate continuously, as the algorithm will get stuck when the learning rate is reduced faster than the error is reduced. The solution is to reduce the error gradually.

It might also be beneficial for different layers to have different learning rates, as the layers farthest away from the output will be least affected by the backpropagtion. It is for this reason wise to assign greater learning tot the layers closest to the input layer.

eliminating the learning rate. Because of the previous two backpropagtion-specific learning rate properties can a chosen learning rate be detrimental for the network learning. To overcome this, resilient backporpagation is often used. Hovere, resilient backpropagtion is not necesary better. We will not go deeper into comparing normal backpropagtion with resilient backpropagtion. Just keep in mind that resilient backpropagtion is based on these two principles:

- Weight changes are not proportional to the gradient thereof.
- Many dynamically adjusted learning rates instead of one static learning rate.

4.3.2 Hopfield networks

A Hopfield network is a completely linked network developed by John Hopfield [Hopfield, 1982]. He was inspired by how the magnetic particles organize themselves into a state of least energy. You can compare the magnetic exerted by the particles with the connection weights between neurons and position or rotation of the particles with the activation of a neuron. To simulate this, all neurons must affect each other, but not itself, hence the choice for a completely linked network. A standard Hopfield network uses a binary activation method so the neuron has only 2 possible states.

A hopfield network is made up out of a set of H completely linked neurons. each of these neurons can only have two possible states, namely $\{-1, 1\}$. The state of the network of $|H|$ neurons can then be expressed as a string value $x \in \{-1, 1\}^{|H|}$. As said in previously in this thesis, is virtually impossible to determine a set of input and output neurons for a completely linked network.

Training a Hopfield network. Again, we will use a set P of training patterns $p \in \{1, 1\}^{|H|}$ each pattern representing a minimum for out network. In contrast to other networks, we already know the minima for the error function, so we don't need to look for them any more. Instead we can just define them, so that the network will converge to the closest minimum when input is presented.

Hopfield network training happens offline and is a done in a single iteration. this means that all patterns to recognize will be learned at once.

$$w_{i,j} = \sum_{p \in P} p_i \cdot p_j \tag{4.20}$$

The influence of connection weights. A neuron can change their state in function of the current state the other neurons and connection weights. This means that the connection weights are able to influence the change of the whole network. There are three possibilities for the connection weights

$w_{i,j} > 0$ This will force the two connected neurons to become equal in state. The bigger this value is, the harder the two neurons will try to get in the same state.

$w_{i,j} < 0$ This will force the two connected neurons to become opposite in state. The bigger this value is, the harder the two neurons will try to get in the opposite state.

$w_{i,j} = 0$ This means that the two neurons do not influence each other.

Converging a Hopfield network.

input and output of a Hopfield network. A Hopfield network automatically looks for a minimum for the state it is in. We will use this, and define an input pattern as being exactly that, a state of a Hopfield network. f we use the binary string representation again, then a Hopfield input pattern can be described as string $x \in \{1, 1\}^{|H|}$.

The network will converge to a minimum state after the network has been initialized with the input pattern. But how an we now that such a minimum has been reached? Simple, the network stops. It can be mathematically proven that a regular completely linked (hence only zeros on the diagonal) hopfiled network will always converge to a minimum. We can then build another binary string from the reached equilibrium, $y \in \{1, 1\}^{|H|}$ is then the hopfield output.

4.3.3 Self-organizing maps

TODO

4.4 Implementation

4.4.1 Captcha builder

TODO TO USE: [Browning and Kolas, 2007]; [Chellapilla and Larson, 2005]; [Fletcher, 2009]; [Gang and Zheyuan]; [Jiang and Kam Sie Wah, 2003]; [Sathasivam Saratha, 2011]; [Törmä, 1995]

4.4.2 Neural networks

TODO

Chapter 5

Conclusion

TODO

Bibliography

M Tariq Banday and NA Shah. A Study of CAPTCHAs for Securing Web Services. *IJSDIA International Journal of Secure Digital Information Age*, 1(2):66–74, 2011. URL http://adsabs.harvard.edu/abs/2011arXiv1112.5605T.

Rodney Beede. *Analysis of reCAPTCHA effectiveness*. PhD thesis, University of Colorado at Boulder, 2010. URL https://www.google.be/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&sqi=2&ved=0CD0QFjAB&url=http%3A%2F%2Fwww.rodneybeede.com%2Fdownloads%2FCSCI5722%2520-%2520Computer%2520Vision%2C%2520Final%2520Paper%2C%2520Rodney%2520Beede%2C%2520Fall%25202010.pdf&ei=poOGUebeNsanO8LsgIAE&usg=AFQjCNHg9HPwDNUm7Qvv1tYXZ7YLhUToqg&sig2=-AYq-OR-Qnr6Qu6rXpVGNw&bvm=bv.45960087,d.d2k.

Adam Browning and Dave Kolas. Defeating CAPTCHAs : Applying Neural Networks. Technical report, 2007. URL http://filebox.vt.edu/users/brownad/CS5804/abrowning-dkolas-project.pdf.

David Bushell. In Search Of The Perfect CAPTCHA, 2011. URL http://coding.smashingmagazine.com/2011/03/04/in-search-of-the-perfect-captcha/. accesed 2013-05-28.

Tianhui Cai. CAPTCHA Solving With Neural Networks. Technical report, TJHSST Computer Systems Lab, 2008.

Kumar Chellapilla and Kevin Larson. Computers beat humans at single character recognition in reading based human interaction proofs (HIPs). *Proceedings of the ...*, 98053, 2005. URL http://www.researchgate.net/publication/220271810_Computers_beat_Humans_at_Single_Character_Recognition_in_Reading_based_Human_Interaction_Proofs_(HIPs)/file/79e4150d119fa42dff.pdf.

Stephen Cobb. The Economics of Spam, 2003. URL http://spamhelp.whybot.com/articles/economics_of_spam.pdf.

Dennis Egen. A Proposal For Improvements of Image Based CAPTCHA. Technical report, Rutgers University, Camden, 2009.

Tristan Fletcher. Hopfield Network Learning Using Deterministic Latent Variables. Technical report, University College London, London, 2009. URL http://www.academia.edu/283069/Hopfield_Network_Learning_Using_Deterministic_Latent_Variableshttp://science5.net/h/hop-eld-network-learning-using-deterministic-latent-variables-w5362.html.

Wei Gang and Yu Zheyuan. Storage Capacity of Letter Recognition in Hopfield Networks. Technical report. URL FacultyofComputerScience, DalhousieUniversity,Halifax,N.S.,CanadaB3H1W5.

Jeff Heaton. *Introduction to neural networks with Java*. 1 edition, 2005. ISBN 097732060X. URL http://tocs.ulb.tu-darmstadt.de/185177875.pdf.

D O Hebb. *The Organization of Behavior: A Neuropsychological Theory*, volume 44 of *A Wiley book in clinical psychology*. Wiley, 1949. ISBN 0805843000. doi: 10.2307/1418888. URL http://www.amazon.com/dp/0805843000.

J. J. Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982. ISSN 0027-8424. doi: 10.1073/pnas.79.8.2554. URL http://www.pnas.org/cgi/content/long/79/8/2554.

SY Huang, YK Lee, Graeme Bell, and Zhan-he Ou. *An efficient segmentation algorithm for CAPTCHAs with line cluttering and character warping*. PhD thesis, Ming Chuan University, 2010. URL http://link.springer.com/article/10.1007/s11042-009-0341-5.

Xudong Jiang and Alvin Harvey Kam Sie Wah. Constructing and training feed-forward neural networks for pattern classification. *Pattern Recognition*, 36:853–867, 2003. URL http://www.sciencedirect.com/science/article/pii/S0031320302000870.

Dennis W K Khong. An Economic Analysis of Spam Law. *Erasmus Law and Economics Review*, 1(February):23–45, 2004. URL http://www.eler.org/viewarticle.php?id=2.

David Kriesel. *Neural Networks*. 2013 edition, 2013. URL http://www.dkriesel.com/en/science/neural_networks.

laughingsquid.com/. Crapcha, impossible captcha tests that pranks web users. http://laughingsquid.com/crapcha-impossible-captcha-tests-that-prank-web-users/.

Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon Mccoy, Geoffrey M Voelker, and Stefan Savage. *Understanding CAPTCHA-Solving Services in an Economic Context*. PhD thesis, University of California, San Diego, 2010. URL https://www.google.be/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&sqi=2&ved=0CEEQFjAA&url=http%3A%2F%2Fwww.usenix.org%2Fevent%2Fsec10%2Ftech%2Ffull_papers%2FMotoyama.pdf&ei=8I6GUa-YA6ar0QWAvoDoDg&usg=AFQjCNGw19XqLpU9H07GzrwXiZzp7AUSHw&sig2=n_TgEyAkII33doIB1VJFDw&bvm=bv.45960087,d.d2k.

municipal cooperation.org. Help:adding content. http://www.municipal-cooperation.org/index.php?title=Help:Adding_content, unknown. Accessed: 2013-05-28.

Moni Naor. Verification of a human in the loop or Identification via the Turing Test. *. . . . wisdom. weizmann. ac. il/˜ naor/PAPERS/human . . .*, 1996. URL http://www.wisdom.weizmann.ac.il/~/naor/PAPERS/human.pdf.

Google reCAPTCHA. recaptcha. http://www.google.com/recaptcha, 2013. Accessed: 2013-05-28.

Sathasivam Saratha. Learning Rules Comparison in Neuro-Symbolic Integration. *International Journal of Applied Physics and Mathematics*, 1(2):129–132, 2011. URL http://www.ijapm.org/papers/025-P0025.pdf.

Graig Sauer and Harry Hochheiser. Towards a universally usable CAPTCHA. *. . . the 4th Symp. On Usable . . .*, pages 2–5, 2008. URL http://www.researchgate.net/publication/228957237_Towards_a_universally_usable_CAPTCHA/file/d912f50d10c2235ccc.pdf.

Markus Törmä. *Kohonen self-organizing featuremap in pattern recognition*. PhD thesis, Helsinki University of Technology, 1995.

Yasuharu Ukai and Toshihiko Takemura. Spam mails impede economic growth. *The Review of Socionetwork Strategies*, 1(1):14–22, March 2007. ISSN 1867-3236. doi: 10.1007/BF02981628. URL http://link.springer.com/10.1007/BF02981628.

Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a Microsoft captcha. *Proceedings of the 15th ACM conference on Computer and communications*

security - CCS '08, page 543, 2008. doi: 10.1145/1455770.1455839. URL http://portal.acm.org/citation.cfm?doid=1455770.1455839.

Roger Stephen Young. *How Computers Work Processor and Main Memory*. 2001. URL http://www.fastchip.net/howcomputerswork/bookbpdf.pdf.

List of Figures

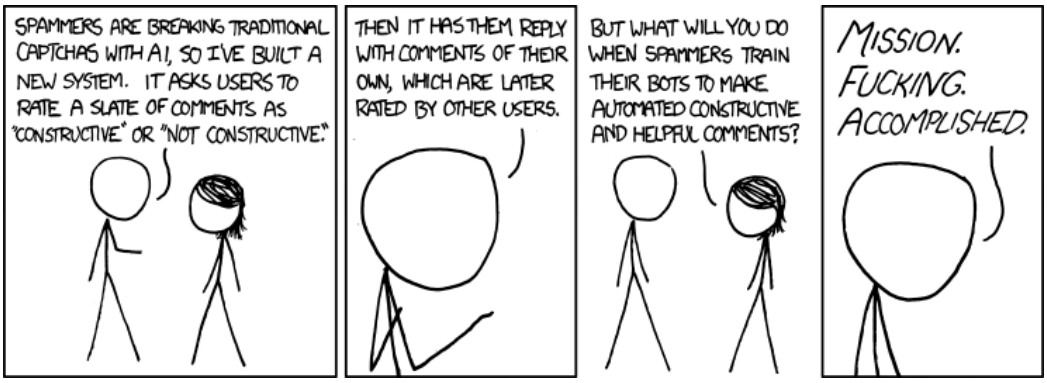
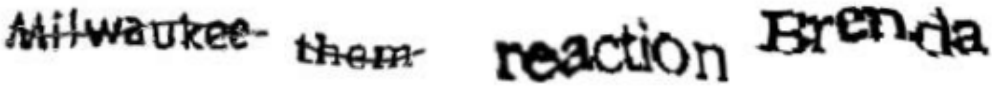


Figure 1: xkcd on the future of CAPTCHA (Source: `http://www.xkcd.com/810/`, accessed on 2013/05/28)

Figure 2: The reCAPTCHA system (Source: [municipal cooperation.org, unknown])



(a) Early 2008

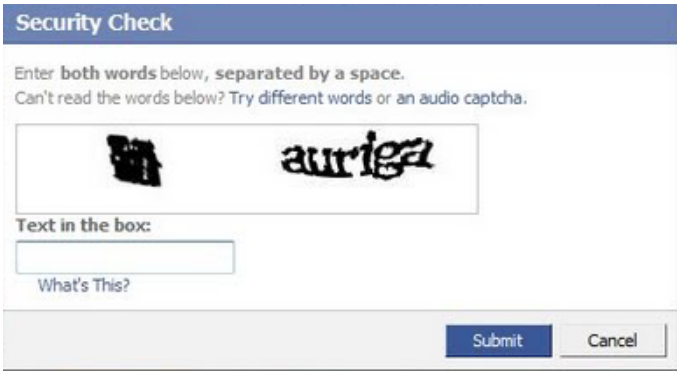
(b) December 16th 2009

(c) January 24th 2010

(d) May 28th 2013

Figure 3: Examples of CAPTCHAs directly Downloaded from reCAPTCHA (Source: [Motoyama et al., 2010] and [reCAPTCHA, 2013])

(a) Impossible CAPTCHA (Source: `http://asmallpieceofgodsplan.blogspot.be/2012/04/captchas.html` Accessed on 2013-0528)



(b) Impossible CAPTCHA (Source: `http://asmallpieceofgodsplan.blogspot.be/2012/04/captchas.html` Accessed on 2013-0528)

(c) Impossible CAPTCHA (Source: `http://oactechnology.com/it-blog/blog/2012/08/31/captcha/` Accessed on 2013-0528)

(d) Impossible CAPTCHA (Source: `http://ragegenerator.com/pages/comic/39222` Accessed on 2013-0528)

Figure 4: Examples of CAPTCHAs nearly impossible to solve

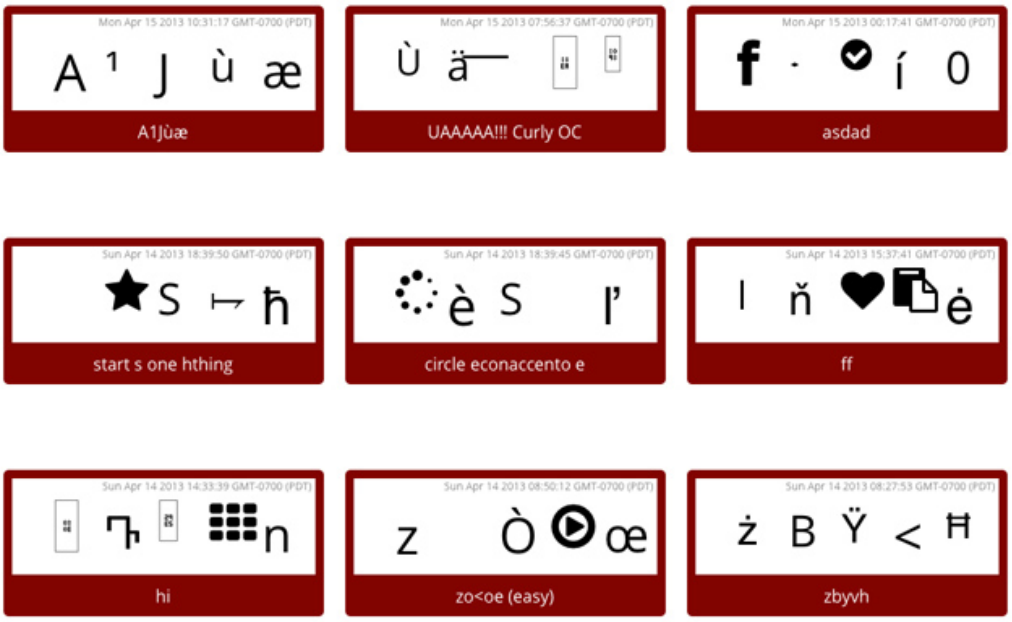


Figure 5: Test generated by the CRAPCHA system (Source: [laughingsquid.com/])

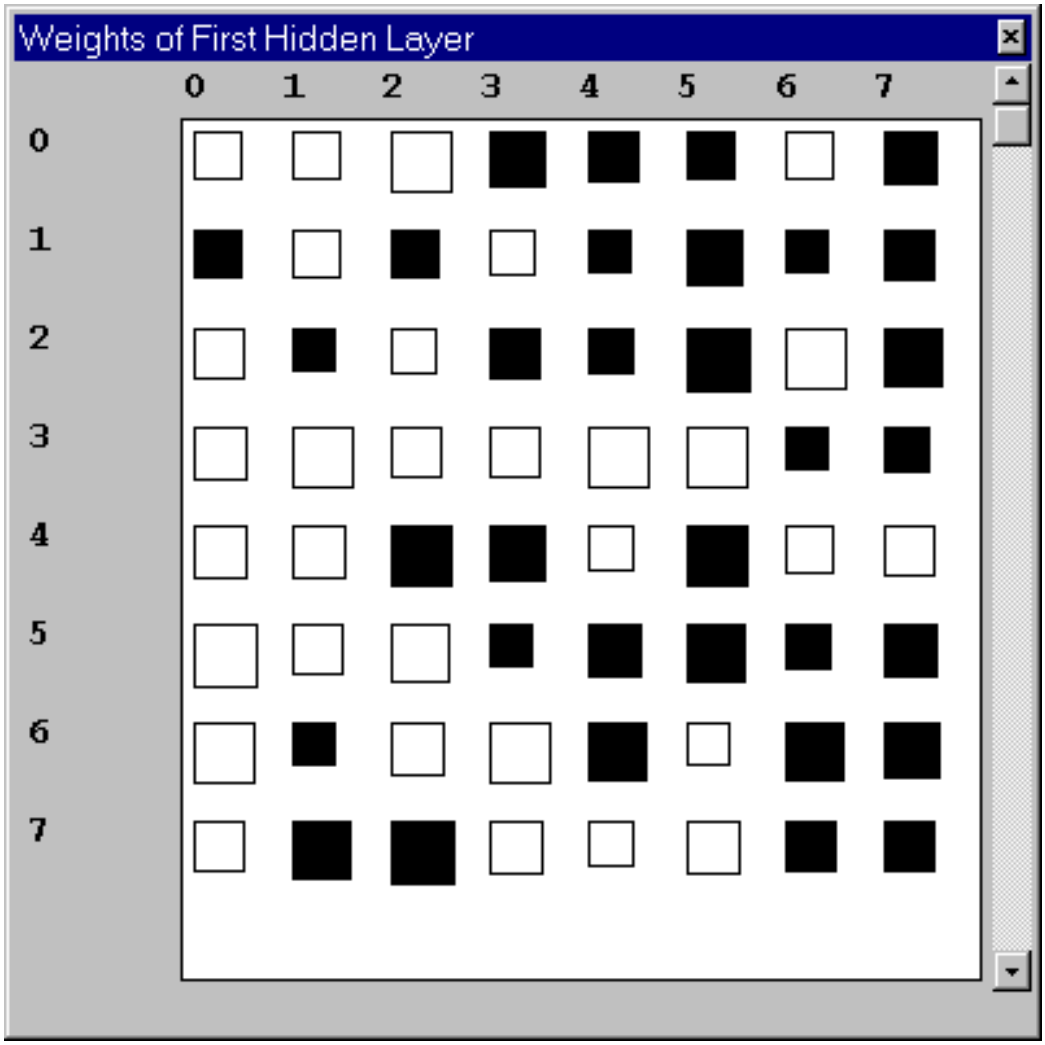


Figure 6: An example Hinton diagram (Source: `http://www.nd.com/products/nsv30/hinton.htm`)