# A super-awesome jedi power title

Magnus Halvorsen

October 1, 2014

**Abstract**

# Chapter 1

# Introduction

## 1.1 Theory

In this section we will go through the fundamental mathematics behind the Convolution Neural Network model, in order to be able to recognize which operations that should be hardware-accelerated.

### 1.1.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational model that is used for machine learning and pattern recognition. It is used in The name and basic concept is inspired by how the animal brain uses a network of neurons to recognize and classify objects.

An ANN can intuitively be viewed as a probabilistic classifier. Depending on the input data it will output the probability that the data belongs to a certain *class* (e.g. an object in an image or an investment decision). As the brain it can be trained to recognize different classes by being provided a set of labeled training data, e.g. a set of faces and a set of non-faces. It can then learn to decide whether a image contains a face or not. This is called supervised learning. The network can also be trained unsupervised, by providing it with a set of unlabeled images. It will then learn to recognize a set of classes, but will be unable to label them.

The topology of an ANN is a number of layers containing a set of so-called neurons. A neuron takes as in a set of inputs (e.g. image pixels), where each input is associated with a respective weight. The input and the weight are then multiplied and summed, and the result is used to calculate a non-linear activation function. More formally a neuron is defined as follows:

$$Input : \{x_1, x_2, \ldots, x_n\} = \mathbf{x}$$

$$Output : f(\mathbf{w^T x}) = f(\sum_{i=1}^{n} w_i x_i + b)$$

**w** is the connection weights, b is the neuron bias and *f(...)* is the activation function. *f(...)* tends to be either:

$$Sigmoid : f(z) = \frac{1}{1 - e^{-z}}, \in [0, 1]$$

or

$$Hyperbolictagent : f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \in [-1, 1]$$

An ANN consist of $n_l$ layers, each containing a set of neurons. The first layer is *the input layer*, and the last layer is *the output layer*. The layers in between are called *the hidden layers*. Each layer uses the previous layer output as input. The input layer is provided with the initial input and calculates its activation functions (one for each neuron). The result is propagated to the first hidden layer, and continues up until it reaches the output layer - which provides the final output. This is known as a *feedforward neural network.*

The network takes in two parameters, $(\mathbf{W}, \mathbf{b}) = (\mathbf{w^{(1)}}, \mathbf{b^{(1)}}, \mathbf{w^{(2)}}, \mathbf{b^{(2)}}, \ldots, \mathbf{w^{(n_l)}}, \mathbf{b^{(n_l)}})$. Then $w_{ij}^l$ denotes the weight between neuron j in layer l, and neuron i in layer l+1. $b_i^l$ denotes the bias associated with neuron i in layer l+1.

During the training of the network it is the parameters $(\mathbf{W}, \mathbf{b})$ that are altered in order to adapt the network to the training data. This is done by providing the network with a set of training examples, where we provide an input and an expected output. We then use a cost function to estimate the error of the actual output. Our goal then would be to minimize the cost function, so the actual output is as close as possible to the expected output. This can be done by a technique called gradient decent.

This paragraph must be elaborated!

Let the cost function for a single training example $(x, y)$ be defined as follows:

$$Cost(\mathbf{w}, b; x, y) = \frac{1}{2}(h_{\mathbf{w},b}(x) - y)^2$$

Where $x$ is the input, $h_{\mathbf{w},b}(x)$ is the actual output of the ANN and $y$ is the correct output. Then the cost function for $m$ training examples $((x^1, y^1), (x^2, y^2), \ldots, (x^m, y^m))$ is:

$$Cost(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} Cost(\mathbf{w}, b; x^{(i)}, y^{(i)}) + \frac{\lambda}{2} \sum_{l=1}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l + 1} (\mathbf{w}_{ji}^l)^2$$

The first term is simply the average sum-of-squares error term. The second term is the *text*regularization term, or *weight decay term.*

Based on this we can use *gradient decent* to compute how we should alter the weights in order to reduce the cost function.

1. Perform a feedforward pass, computing the output of every layer.
2. For each output neuron, compute:

$$\delta_k = o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden layer $l = n_l - 1, n_l - 2, \ldots, 2$ compute:

$$\delta_i^l = o_i^l(1 - o_i^l) \sum_{j=1}^{s_{l+1}} w_{ij}^l \delta_j^{l+1}$$

4. Compute the partial derivative for each weight:

# Chapter 2

# Related Work

Related work stuff

# Chapter 3

# Method

My special method.

# Chapter 4

# Discussion

Discuss dis

# Chapter 5

# Conclusion

I HAVE CONCLUDED!