A super-awesome jedi power title

Magnus Halvorsen

October 6, 2014

Abstract

Chapter 1

Introduction

Let me introduce you to a world of wonders. Pie.

Chapter 2

Related Work

In [1] a CNN was implemented on a Virtex-4 SX35 FPGA from Xilinx. In this implementation all the fundamental computations was hard-wired, and controlled by a 32bit soft processor using macro-instructions. Training was done offline, and a representation of the network was provided to the soft processor. With this implementation they were able to process a 512×384 grayscale image in $100ms$, i.e. 10 frames per second.

I can see you sneaking, Yaman!

Chapter 3

Background

In this chapter we will go through the fundamental mathematics and concepts behind the *Convolutional Neural Network* (CNN) model, in order to be able to recognize which operations that can be hardware-accelerated. It give a basic introduction to both general neural networks and *CNNs*.

3.1 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:

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

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

$\mathbf{w}$ 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 compute the error of the actual output. Our goal is 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.

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*, which tends to reduce *overfitting.*

Based on this we can use *gradient decent* to compute how we should alter the weights in order to reduce the cost function. One iteration of gradient descent updates $\mathbf{w}$ and b as follows:

$$\mathbf{w}_{ij}^{(l)} = \mathbf{w}_{ij}^{(l)} - \alpha \frac{\partial}{\partial \mathbf{w}_{ij}^{(l)}} Cost(\mathbf{w}, b)$$

$$\mathbf{b}_i^{(l)} = \mathbf{b}_i^{(l)} - \alpha \frac{\partial}{\partial \mathbf{b}_i^{(l)}} Cost(\mathbf{w}, b)$$

Where α is the learning rate, which is a predetermined constant. Note that this would only make us able to compute the gradient for the output layer. In order to perform gradient decent on the hidden layers, we need to propagate the error from the output layer backwards, to the hidden layers. For this we use the *backpropagation algorithm*. Let $o_i^{(l)}$ denote the output of the ith neuron in layer l, and $z_k^{(l)}$ is the weighted sum of the inputs plus the bias for the kth neuron in layer l. Then the *backpropagation algorithm* can be formalized as follows:

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

$$\delta_k = \frac{\partial}{\partial z_k^{(n_l)}} Cost(\mathbf{w}, b; x, y) = -o_k^{n_l}(1 - o_k^{n_l})(y_k - o_k^{n_l})$$

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 and bias:

$$\frac{\partial}{\partial \mathbf{w}_{ij}^{(l)}} Cost(\mathbf{w}, b; x, y) = o_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial \mathbf{b}_i^{(l)}} Cost(\mathbf{w}, b; x, y) = \delta_i^{(l+1)}$$

Now, combining *gradient decent* and the *backpropagation algorithm* we can describe an algorithm to train our network:

1. Initialize the weights $\mathbf{w}^{(l)}$ and b^l for all l.
2. Do steps 3 to 5 until the $Cost(\mathbf{w}, b; x, y)$ function is low enough or converges.
3. Set $\Delta \mathbf{w}^{(l)} := 0$ and $\Delta b^{(l)} := 0$ for all l.
4. For i = 1 to m,
   (a) Use the backpropagation algorithm to compute $\nabla_{\mathbf{w}^{(1)}} Cost(\mathbf{w}, b; x, y)$ and $\nabla_b^{(l)} Cost(\mathbf{w}, b; x, y)$.
   (b) Set $\Delta \mathbf{w}^{(l)} := \Delta \mathbf{w}^{(l)} + \nabla_{\mathbf{w}^{(1)}} Cost(\mathbf{w}, b; x, y)$.

(c) Set $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} Cost(\mathbf{w}, b; x, y)$.

5. Update the parameters:

$$\mathbf{w}^{(l)} = \mathbf{w}^{(l)} - \alpha[(\frac{1}{m}\Delta\mathbf{w}^{(l)}) + \lambda\mathbf{w}^{(l)}]$$

$$b^{(l)} = b^{(l)} - \alpha[\frac{1}{m}\Delta b^{(l)}]$$

3.2 Convolutional Neural Network

Convolutional Neural Network[2] (CNN) is a variation of the *Artificial Neural Network* model, which is mainly used for object recognition in images. It addresses three major problems the standard ANN model faces when it comes to object recognition. First, even small images contains a large amount of pixels/inputs, e.g. a 32×32 image contains 1024 pixels/inputs. A fully connected network with 100 hidden units would then end up with 1024×100 weights that needs to be calculated in the first layer! Making it computationally infeasible and not scalable. Second, objects of the same class are seldom exactly alike, something the network has to take into consideration. While possible, the network would have to be very large, would probably contain several neurons with similar weight vectors positioned at different places in the network, and would require a very large number of training samples. Third, a fully connected ANN does not take into consideration the topology of the input. An image has a strong 2D spatial locality correlation, which makes it possible to combine low-order features (edges, end-points etc.) into higher-order features.

The CNN model adds two additional types of layers, in addition to the standard ANN layers: a convolution layer and a pooling/subsampling layer.

The idea behind the two new layers is to exploit the strong 2D local structure of images, i.e. pixels close to each other are highly correlated. By using local correlation one can extract and combine small local features (e.g. edges, corners, points) into higher-order features (e.g. nose, mouths, forehead), which can in the end be recognized as an object (e.g. a face). Intuitively, the convolution layer performs the feature extraction by applying a convolution $n \times n$ filter (kernel) on the whole image, and puts the result in a feature map. If the filter e.g. extracts vertical edges, only the vertical edges from the original image would be represented in the resulting feature map. Thus you can extract different features by having several feature maps with different filters.

Once you have detected a feature, the exact position become less important

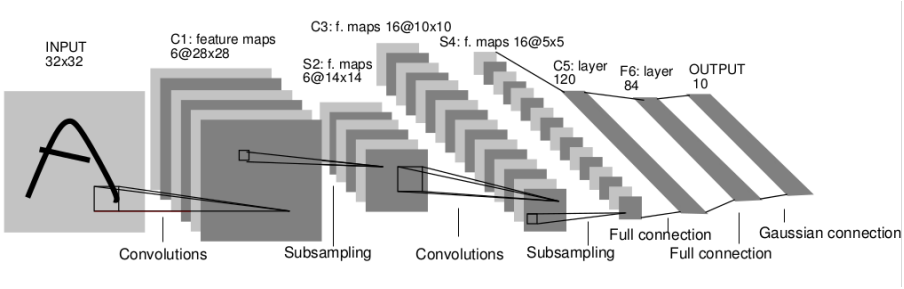

Figure 3.1: An example CNN, the LeNet-5[2].

Chapter 4

Method

My special method.

Chapter 5

Discussion

Discuss dis

Chapter 6

Conclusion

I HAVE CONCLUDED!

Bibliography

[1] Clément Farabet, Cyril Poulet, Jefferson Y. Han, and Yann LeCun. CNP: An FPGA-based processor for Convolutional Networks. *FPL 09: 19th International Conference on Field Programmable Logic and Applications*, 1(1):32–37, 2009.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE International Conference on Computer Vision*, 1998.