

# MLP Coursework 4

Eskil Joergensen

March 19, 2017

## 1 Introduction

Fully connected networks offer good performance at a relatively efficient computational cost, however, current state-of-the-art networks are all convolutional neural networks (CNN). This report presents a series of experiments set to optimize the training performance of a CNN on a image classification task, using the CIFAR 10 dataset [1].

Building on a baseline set of experiments, done with a fully connected neural network,

the presented procedure tries to improve the performance of the more advanced methods used here.

In addition to improving the overall performance of the training of the network, the experiments were guided by a goal of 70% accuracy on the test data set. As a result, the methodology and incremental improvements to the model were made on these grounds.

what is being investigated: activation functions learning rates number of feature maps and kernel sizes regularization (batch norm, Local Response Norm. and dropout)

### 1.1 Background

Preliminary experiments have been conducted on the same data, using a fully connected neural network. The network was trained using a variety of activation functions, hidden layer depths and widths as well as learning rate scheduler.

#### 1.1.1 Activation Functions

Four activation functions were compared. The logistic sigmoid  $f(x) = \frac{1}{1+e^{-x}}$ , the hyperbolic tangent  $f(x) = \tanh(x)$  the rectified linear (ReLU)  $f(x) =$

$\max(0, x)$  and the exponential linear unit (ELU):

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \quad (1)$$

where  $\alpha$  is a hyperparameter that decides the range of negative values for which the activation function saturates.

The results showed that the ELU activation function outperformed the other three, with a final training and validation set accuracy of 66.9 % and 50.5% respectively.

### 1.1.2 Hidden layers

### 1.1.3 Learning Rate Schedule

## 2 Network Architecture

## 3 Methodology

Establishing a baseline model, and choosing the appropriate algorithms to include, can be a daunting task. Even a relatively small CNN has a large number of tunable parameters, and it can be difficult to know where to begin. The section presents the baseline model chose for the experiments, the procedure used to train the network and the parameters used.

### 3.1 The Baseline Model

The default baseline model used has two convolutional layers, two fully-connected layers and finally a softmax classification layer. The network also had two 2x2 pooling layers after each convolutional layer.

Instead of a learning rate scheduler, the Adam adaptive learning rate optimizer was used throughout the investigation. The optimizer was used with varying learning rates, but with  $\beta_1=0.9$ ,  $\beta_2=0.999$  and  $\epsilon = 1e-8$ . (default parameters)

#### 3.1.1 Convolutional Layer

Convolutional neural networks use kernels, or filters, to take advantage of the spatial structure of its inputs. As the kernel is moved across the input image,

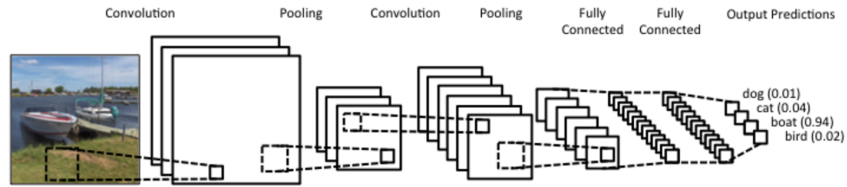


Figure 1: Showing a visual representation of the model used, from TensorBoard.

the values of the resulting feature map are determined by the values of the kernel.

The convolution was implemented using the `tf.nn.conv2d` method from TensorFlow (tf) with stride of 1 and no padding (`padding='SAME'`).

### 3.1.2 Fully Connected Layer

### 3.1.3 Pooling Layer

`tf.nn.max_pool`

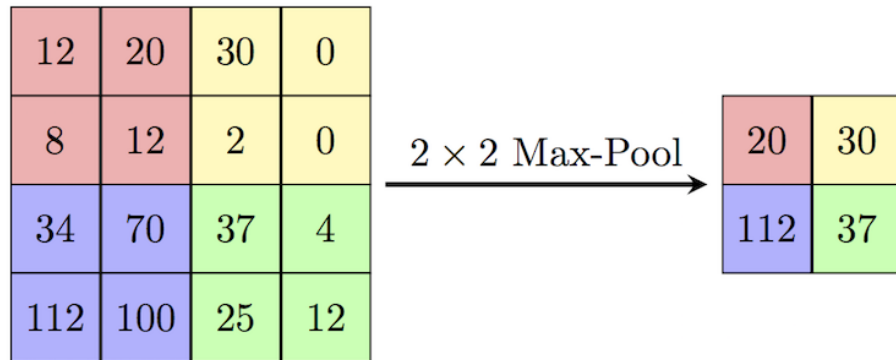


Figure 2: Showing a visual representation of the model used, from TensorBoard.

### 3.1.4 Norm Layer

`tf.nn.lrn`

### 3.1.5 Dropout Layer

`tf.nn.dropout`

### 3.1.6 Batch Normalization

### 3.1.7 Weights and Biases

The weights used in both the convolutional and fully connected layers were initialized using a truncated normal distribution. The values were distributed with zero mean and a standard deviation of 0.1. Values more than two standard deviations away from the mean were dropped. The distributions were initialized with the `tf.truncated_normal` method.

The biases used were all initialized to zero using the `tf.zeros` method. Additionally, the weights and biases were added to the TensorFlow graph using `tf.Variable()`.

## 3.2 Experimental Procedure

The procedure used to tune the performance of the network involved three stages. During the initial stage the network was trained for few epochs, with a coarse spread of hyper parameter and multiple algorithms were tested during this phase. The overall results from the initial trials informed the finer range of parameters used for additional training and further tuning of hyper parameters. The second stage involved longer training. The third stage, before the final testing, the model was trained for significantly longer, and only three varieties of the model were used.

### 3.2.1 Activation Functions

Initially the model was trained using three learning rates, with the four activation functions used in the preliminary experiments: ReLu, Elu, Tahn and Sigmoid. These were tested with a learning rate of 0.0005, 0.003 and 0.01, and the network was trained for 5 epochs. The relatively large, non-uniform spread of the learning rate was chosen intentionally to observe how the activation functions would react to the rates.

After the initial trials, the best two activation functions were kept for further experimentation, and the other two were discarded. The worst learning rate for each of the remaining activation functions was also replaced with another learning rate, trying to narrow down the region of learning rates for which the training of the network performs better. For the second round of trials, the network was trained for 10 epochs.

The second round informed the activation function to be used for the last round, and again the learning rates were narrowed down further. The last set of trials, for the activation functions, the network was trained for 20 epochs. The learning rate that resulted in best performance was then used for the during the later experiments.

The activation functions and learning rates were tested using two convolutional layers with 3x3 kernels and 24 feature maps. The model also had a 3x3 pooling layer with stride of two after each convolutional layer. After the tensors were flattened, the network had two fully connected layers before a final softmax classification layer. The network used the Adam optimizer.

### **3.2.2 Filter Sizes and Feature Maps**

The kernels are responsible for extracting features from the input data, and their sizes as well the the number of feature maps output will affect the performance of a network.

Next of combinations of kernel sizes and feature maps were tested. The size of kernels were 3, 5 and 7, whereas the number of feature maps were 4, 24 and 32. Here as well, the best combination was chosen for the following trials.

with and without batch normalization and l2 loss

trained for 10 epochs

changed max pooling <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>

replaced batch norm with lrn

introduced dropout and data augmentation

trained for 40 epochs

### **3.2.3 Stage 2: Finding the one and Testing**

trained for 100 epochs

## **4 Results and Discussion**

## **5 Conclusion**

## **6 Future Work**

## References

- [1] The cifar-10 and cifar-100 dataset. [Online]. Available:  
<https://www.cs.toronto.edu/~kriz/cifar.html>