# MLP Coursework 4

Eskil Joergensen

March 21, 2017

## 1 Introduction

Fully connected neural networks are trained at a relatively low computational cost, and they have been used in machine learning for quite some time. However, current state-of-the-art networks are all convolutional neural networks (CNNs) [1]. The high performance is (typically) achieved as the kernels in the convolutional layers extract features from the input data via their spacial locality. A common use case is object recognition in images, but can also be used in other tasks such as natural language processing [2].

Preliminary work has been done on the same data set using a fully connected neural network. More details are presented in the Background section (1.1) below.

CNNs introduce a significantly larger number of weights and connections, compared to a fully connected network, which means the parameters of the network must be carefully tuned to ensure adequately successful training.

This report presents a series of experiments set to investigate how some key components of a CNN affect it's overall performance during training. Furthermore, the (optimal) findings from each experiment are integrated into the network for the subsequent experiments, trying to improve the performance of the network as much as possible. The following aspects of the CNN are investigated:

- How does the size of kernels and number of feature maps in the convolutional layers affect the training? Time, performance, computational resources
- How does the choice of activation function impact the performance of the network?
- What type, and magnitude, of regularization is needed to overcome potential over-fitting of the CNN? Without any normalization or regularization significant over-fitting can occur during training of the network. MORE +++

optimize the training performance of a CNN on a image classification task, using the CIFAR 10 dataset [3]. 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.

## 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):

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

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

2. Treatment for the object (what do to object) 3. Procedure to collect data 4. Procedure to analyze data

## 3.1 The Network Architecture

A neural network with two convolutional and two fully connected hidden layers was chosen for this study. The network also had two max pooling layers, one after each convolutional layer, as well as a 10-way softmax output classification.

Why this model?

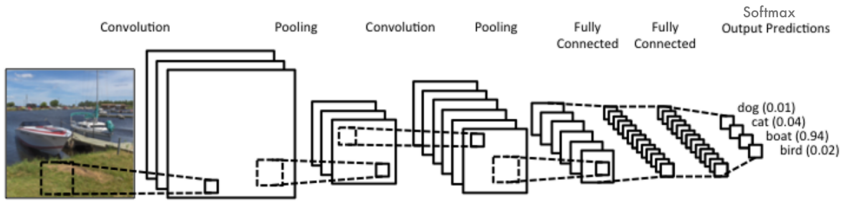


*Figure 1: Showing a visual representation of the model used, from TensorBoard.*

### 3.1.1 Convolutional Layer

The convolution was implemented using the `tf.nn.conv2d` method from TensorFlow (tf) with stride of 1 and no padding (`padding='SAME'`). The convolution operation was executed on the layer inputs together with the kernels (using the specified stride and padding). Both the layer inputs and the kernels were TensorFlow tensors, and the result of the convolutional operation was also a TensorFlow tensor.

The biases were added to the tensor after the convolution. In the convolutional layers where batch normalization was specified, ... . Finally, an activation function was applied to the tensor, before passing the output on to the next layer.

A truncated normal distribution was use to initialize the kernels, and the biases were initialized to zero. See Section 3.1.6, Weight and Biases, for the specific implementation in TensorFlow.

### 3.1.2 Fully Connected and Output prediction Layer

The fully connected and the classification output layers were implemented using the same class, as only a few factors separate the behavior of the two. A constant halving of the number of hidden units were integrated into all the fully connected layers, except the output prediction layer which had a 10-was output.

The weights and biases were initialized in the same as mentioned for the convolutional layers, only the weights for the affine layers were two dimensional.

Finally, the inputs were multiplied with the weights using `tf.matmul` ,before the biases were added to the tensor. For the final classification layer this tensor would be returned, otherwise an activation function would be applied before returning the tensor.

### 3.1.3 Pooling Layer

The max pooling layer was implemented using the `tf.nn.max_pool` method from TensorFlow. Each pooling layer used a 3 by 3 window for the sampling, and a stride of 2. Similarly to the convolutional layer the pooling layer used no padding, `padding='SAME'` in TensorFlow.

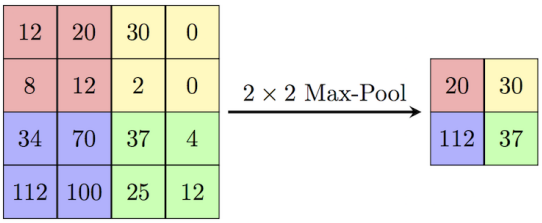


*Figure 2: Showing an example of a 2x2 max pool operation. The 3x3 equivalent operation was used throughout this investigation.*

### 3.1.4 Dropout Layer

The dropout layer was implemented using `tf.nn.dropout(inputs, keep_prop)`, and will keep each element in the input tensor with a probability of `keep_prob`.

### 3.1.5 Batch Normalization

The batch normalization of the convolutional layer was achieved with the `tf.nn.max_pool` method from TensorFlow. Batch normalization transforms its input to zero mean and unit variance. By statistical analysis the batch normalization normalizes the inputs of a transformation layer. Equation 2 shows how the inputs $x_i$ are normalized

$$BN(x_i) = \gamma_i \Big( \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \Big) + \beta_i \tag{2}$$

where $\mu_i$ is the batch mean, $\sigma_i^2$ is the batch variance, $\gamma_i$ is a scale factor and $\beta_i$ is a shift factor. $\gamma_i$ was set to 1.0, $\beta_i$ to zero and $\epsilon$ to 1e-3. The batch mean and variance was obtained from the inputs using `tf.nn.moments`.

#### 3.1.6 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.

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 beta1=0.9, beta2=0.999 and epsilon = 1e-8. (default parameters)

#### 3.2.1 Activation Functions

The model was trained using three learning rates, with the four activation functions used in the preliminary experiments: ReLu, Elu, Tahn and Sigmoid. During the first round of trials the network was trained for 5 epochs. A relatively large, non-uniform spread of the learning rate was chosen intentionally to observe how the activation functions would react to the different rates.

After the initial trials, the best two activation functions were kept for further experimentation, and the other two were discarded. The learning rates were also narrowed down, closer to the best performing learning rate. All activation functions, learning rates and training epochs used during the investigation can be seen in Table 1.

Again, after the second round, the learning rates were modified before the last round of training. As the two last activation functions performed similarly, both were tested in the final trials. The combination of learning rate and activation functions, that resulted in best performance, was then used for the during the next set of experiments, see Section 3.2.2.

*Table 1: Showing the combinations of activation functions, learning rates and training duration in Epochs, for the three rounds of experimentation.*

| Round | Learning Rates | Activation Fn. | Training Epochs |
|---|---|---|---|
| 1 | 5e-4, 3e-3, 0.01 | ReLu, Elu, Tanh, Sigmoid | 5 |
| 2 | 1e-4, 9e-4 | ReLu, Elu | 10 |
| 3 | 4e-4, 7e-4 | ReLu, Elu | 15 |

The effect of the learning rates and non-linear activation functions on the aforementioned network were tested with 3x3 kernels and 24 feature maps. The network did not use any kind of L2 regularization, batch normalization or drop out.

The procedure used to train the network is presented in Section .

### 3.2.2 Filter Sizes and Feature Maps

During the second part of the experimentation, the network was trained with nine combinations of kernel sizes and feature maps. The size of kernels were 3, 5 and 7, whereas the number of feature maps were 16, 24 and 32. During the first round the network was trained for 5 epoch, using the ELU activation function was used with a learning rate of 0.0004.

The network was supposed to be trained in three rounds, incrementally determining the best combination of kernel size and number of feature maps. However, due to exceedingly long training times and swap memory warnings (during the two most expensive trials) the two later rounds were not completed.

The three combinations with 32 feature maps all performed similarly, but at very different completion times and computational cost. See Table 3 for execution times. The 3x3 kernel and 32 feature maps were used for the remaining experimentation.

### 3.2.3 Regularization and Normalization

The last set of experiments

As with the previous experiments, short first round trials were run using coarse set of parameters. The network was trained using eight different combinations of batch normalization, drop out and L2 weight decay. The regularization and normalization parameters used in all three rounds can be seen in Table 2. The batch normalization only applied to the convolutional layers and the dropout

Using three nested loops, the network was trained using eight different combinations of batch normalization, drop out and L2 weight decay. The batch normalization added to the convolutional layers were either true or false, the L2 decay was none or 0.0005 and the keep probability of the dropout layers were either 1.0 (keep all) or 0.5. The network was trained for 5 epochs during the first set of trials.

The third round of training used 20 epochs instead of 15, as the regularization slowed down the training.

*Table 2: Showing the combinations of parameters used during the normalization and regularization investigative rounds.*

| Round | Batch Norm. [1] | Dropout Prob. [2] | L2 Reg. [3] | Training Epochs |
|---|---|---|---|---|
| 1 | True, False | None, 0.5 | None, 5e-4 | 5 |
| 2 | True | 0.45, 0.55 | 5e-4 | 10 |
| 3 | True | 0.5 | 5e-4, None | 20 |

The training runs were done with 3x3 kernels and 32 feature maps in each of the convolutional layers.

## 3.3 Network Training and Data Collection

placeholder data, batch size 50

TensorBoard Normalized running means - numpy custom model class evaluate validation every 100 step, plus first and last step `tf.softmax_cross_entropy_with_logits`

`tf.train.AdamOptimizer`

[1] Batch Normalization
[2] Dropout keep probability
[3] L2 Regularization Weight Decay

# 4 Results and Discussion

All tests were performed on a macOS Sierra 10.12. operating system, with an Intel i5 2.6 GHz processor and 8 GB DDR3 memory with 1600 MHz. All trials were executed inside a custom mlp python virtual environment.

## 4.1 Activation Functions

After the model was trained with all four transformation functions, the ReLu and ELU performed significantly better than the sigmoid and hyperbolic tangent. The first 5 epoch trial showed that a learning rate of 0.01 was too large, and resulted in high total error and low accuracies around 0.1, which meant the final classifications were merely guesses. The lowest learning rate, 0.0005, resulted in the highest accuracies across all the activation functions, and the ReLu trial had a final validation accuracy of 70.3%.

During the second round of trials, the network was trained with the ReLu and ELU activation functions, with learning rates of 0.0009 and 0.0001. After training the network for 10 epochs, the performance decreased for both activation functions, suggesting the better learning rates were closer to 0.0005. The final trial was then run with learning rates of 0.0004 and 0.0007.

The last set of trials gave very similar results, see Figure 3. The highest validation accuracy was 71.6%, by ELU with the learning rate of 0.0004, and the lowest was 69.7% by ReLu with the learning rate of 0.0007. Although the validation accuracy was over 70%, the training set accuracies were much higher, which indicates a high degree of over-fitting. The over-fitting was also reflected in the validation set errors, as can be seen in Figure.

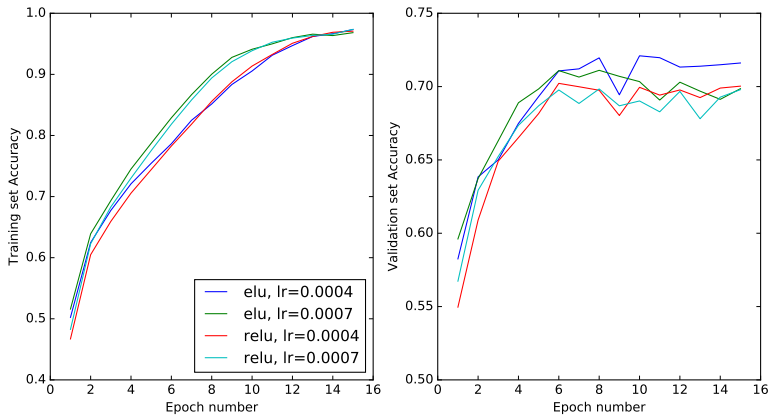


*Figure 3: Showing a visual representation of the model used, from TensorBoard.*

### 4.2 Filter Sizes and Feature Maps

The number of feature maps had a greater impact on the training performance, compared to the size of the kernels. As shown in Figure , the three best performing trials were all using convolutional layers with 32 feature maps.

*Table 3: Showing the training execution time, in minutes, for 5 epochs for different kernel sizes and number of feature maps in the convolutional layers.*

| | Feature Maps | 16 | 24 | 32 |
|---|---|---|---|---|
| Size | 3x3 | 12.5 | 19.1 | 27.0 |
| | 5x5 | 22.5 | 34.6 | 49.1 |
| | 7x7 | 39.9 | 57.9 | 67.0 |

#### 4.2.1 Regularization and Normalization

## 5 Conclusion

## 6 Future Work

# References


[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[2] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 160–167. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390177

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