MLP Coursework 4

Eskil Joergensen

March 18, 2017

1 Introduction

Optimizing the training performance of a convolutional neural network (CNN) on image classification task, CIFAR 10. Building on a baseline set of experiments, done with a deep neural network, the presented () 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 (bartch norm, Local Response Norm. and dropout)

2 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.

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

2.1.1 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, the values of the resulting feature map are partially determined by the values of the kernel.

The convolution was implemented using the tf.nn.conv2d method from Tensorflow(tf). stride of 1 and no padding.

2.1.2 Fully Connected Layer

2.1.3 Pooling Layer

tf.nn.max_pool

2.1.4 Norm Layer

tf.nn.lrn

2.1.5 Dropout Layer

tf.nn.dropout

2.1.6 Batch Normalization

2.1.7 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 TenorFlow graph using tf.Variable().

2.2 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.

2.2.1 Stage 1: Initial Training and Model Verification

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, 0.005 and 0.1. 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 activation function and learning rate was chosen for the rest of the trials in Stage 1.

Next a variety 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

2.2.2 Stage 2: Fine-tuning hyper parameters

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

2.2.3 Stage 2: Finding the one and Testing

trained for 100 epochs

3 Results and Discussion

4 Conclusion

5 Future Work

References