Deep Learning Course 2017 **Ilia Dobrusin**
Autonomous Intelligent Systems
University Freiburg Due Date: 20.11.2017

Deep Learning Lab Course 2017 - Exercise 2

The second exercise of the lab course has the goal of implementing a small convolutional neural network (CNN) in TensorFlow[1]. As in the first exercise, the MNIST data set is used for training and evaluting the network. The code of this exercise can be found at Github[2].

Convolutional Neural Network in TensorFlow

The network for classifying the MNIST data set is based on LeNet and consists of two convolutional layers with 3x3 filters (with a stride of 1), where each layer is followed by a ReLU activation function and a max pooling layer. After the convolutions, the output of the last max pooling layer is used as the input of a fully connected layer with 128 units followed by a softmax layer with 10 units. The optimization is performed with stochastic gradient descent with cross-entropy.

The exercise consists of two experiments. In the first experiment, the network is trained several times, while the learning rate for each training run is varied. All other paremters such as number of training epochs, number of filters in the convolutional layers, and batch size in the stochastic gradient descent are fixed and shared between training runs. The first experiment was conducted with TensorFlow running on the GPU.

The second experiment examines the influence of the number of filters in the convolutional layer on the duration of the training and shows the performance differences of running TensorFlow on the GPU and the CPU during training.

Experiments and Results

Varying Learning Rates

The network was trained using the following learning rates during training:
$0.1, 0.01, 0.001, 0.0001$. The validation accuracy for each training epoch is displayed in figure 1. Each convolutional layer used 16 filters, resulting in a total number of 104250 trainable parameters. The network was trained for a total of 2000 epochs.

Result Figure 1 shows the results of the training runs of the different learning rates, displaying the accuracies on the validation set in the first 1000 learning epochs. The highest learning rate of 0.1 results in the best performance during training on the validation set with an accuracy of 98.930 %. With decreasing learning rate the performance on the validation set decreases. With the lowest learning rate of 0.0001 we obtain a validation accuracy of 98.160 %. The plot in figure 1 shows how the learning rate influences the learning curve: with increasing learning rate, the learning curve gets steeper resulting

[1] `https://www.tensorflow.org/`
[2] `https://github.com/idobrusin/DeepLearningLab2017`

in a faster convergence. In figure 2 a detailed view of learning curve is displayed where the differences between the curves are more pronounced. The figure shows the first 350 training epochs and the resulting accuracies on the validation set between 0.8 and 1.

The resulting performance on the test set is similar to the results of the validation set, with the highest learning rate performing best on the data set. The resulting accuracies on the test set are listed in table 1.

For the MNIST data set, a high learning rate performs best, which allows the the training to be shorter compared to low learning rates. The results show that the network performance for a high learning is good after a small number of training epochs compared to a low learning rate.

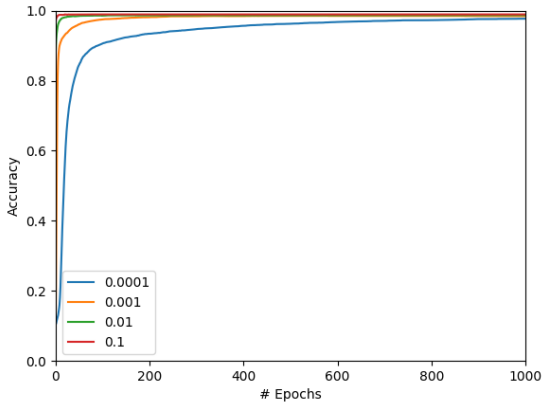

Figure 1: Learning curve of the network during training.

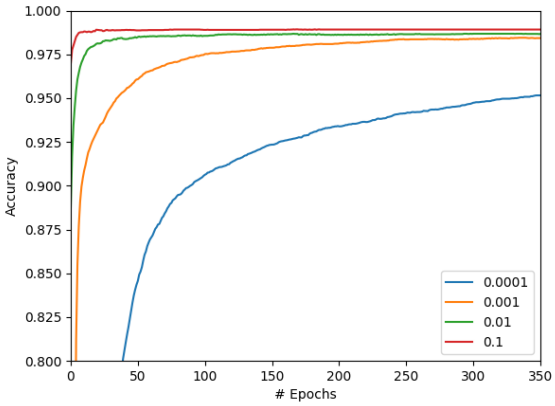

Figure 2: Detailed plot of the learning curves during training

Table 1: Accuracies of the network trained with different learning rates on the test set of MNIST.

Learning rate	**Accuracy**
0.1	0.99120
0.01	0.98620
0.001	0.98480
0.0001	0.98110

Runtime analysis for varying filter sizes on GPU and CPU

In this experiment the network was trained several times with different filter numbers while measuring the runtime of the training. The number of parameters increases for an increasing number of filters in the convolution layers. In table 2 the number of filters and the respective number of trainable parameters in the network is displayed. The network was trained for 250 epochs with a fixed learning rate of 0.01 and a batch size of 100 in the stochastic gradient descent.

The network was trained on the GPU and CPU respectively. On the GPU, the following number of filters where used for training the network: $8, 16, 32, 64, 128, 256$. For the experiments on the CPU the last two number of filters where not used. The

resulting runtimes are displayed in figure 3, where the number of trainable parameters are displayed in millions.

An increasing number of parameters in the network increases the runtime of a training run on the network. As TensorFlow is optimized to run on the GPU, the runtime of the training on the CPU are significantly slower. For an increasing number of parameters the runtime on the CPU increases more between runs compared to the GPU.

Table 2: Number of filters in the convolution layers and the respective number of trainable parameters in the network.

No. of filters	**No. of parameters**
8	52 258
16	104 250
32	211 690
64	440 394
128	953 098
256	2 199 690

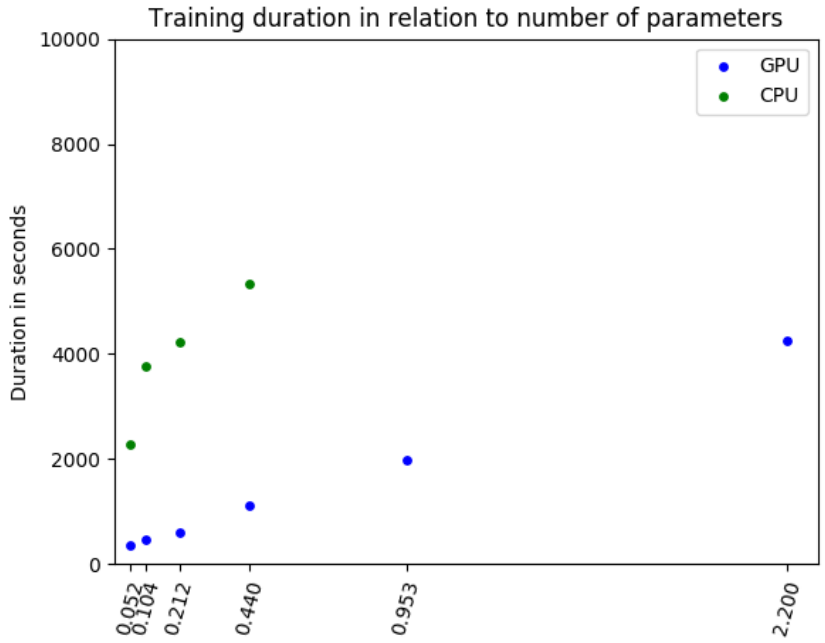

Figure 3: The plot shows the different numbers of trainable parameters in the network and their respective runtimes during traing on the GPU and CPU.