Faculty of Engineering
Albert-Ludwigs-Universitat
Autonome Intelligente Systeme - AIS
Wintersemester 2017/18

Deep Learning Lab Course

2017

Assignment 2

Author: Markus Merklinger
E-mail: merklingermarkus@gmail.com
Tutors: Aaron Klein,
Artemij Amiranashvili,
Mohammadreza Zolfaghari,
Maria Hgle,
Jingwhei Zhang,
Andreas Eitel
Data of submission: November 22, 2017

Deep Learning Lab Course 2017
Assignment 2

November 22, 2017

Abstract

In this Assignment we got familiar with the python Tensorflow framework. This was achieved by implementing a small convolutional neural network (CNN) and training it on the MNIST dataset. With the Architecture of the Network we can perform a classification task of digits. Furthermore the impact different hyperparameter on the performance and runtime was examined. Different learning rates, optimizer dropout and number of filter were evaluated.

Contents

1 Implementation

A scaled-down version of LeNet was implemented in the python Tensorflow (TF) framework. The convolutional neural network (CNN) contains two convolutional layers (16 3 3 filters and a stride of 1), each followed by ReLU activations and a max pooling layer. Then a fully connected layer with 128 units and a soft max layer is added to performe the classification of the digits labeled in the dataset. We optimize the network with minimizing the cross-entropy loss. The implementation allows that the architecture and hyperparameter can be easily changed: The optimizer, learning rate filter depth and the usage of drop out can be easily changed.

The implementation can be found under https://github.com/RedHeadM/dl_lab_2017.

2 Evaluation and Results

2.1 Learning Rate

The learning rate is one of the most important hyperparameter of our CNN. The Figure 1 is illustrating the effect of the different values of the learning rate and how huge the impact is on the performance of the network. Clearly it can be seen that the learning rate of 0.1 performances the best with the SGD optimizer. In contrast the learning rate of 0.0001 really converges very slowly because the fixed steps during the optimization are too small. Since the SCD is used and the wights are randomly distributed the outcome varies each run. Surprisingly the validation and test error are very similar but not exactly the same.

In Figure 2 the effect of dropout is examined. Dropout is used to effective reduce overfitting but as can be seen in the figure that the performance ins nearly identical for this small convolutional network.

The optimizer has also a huge impact on the performance of the CNN. In Figure 3 the Adam Optimizer is used which uses a momentum to update the step size. In the Figure it can be seen that also lower learning rate of 0.001 converges rapidly fast which was not the case with the SGD.

In the runtime experiment we could observe that the learning rate differs also for the architecture of the network. In general a deep network should be trained with a lower learning rate at the beginning to ensure a coverage to good local optimum. Finding a good learning rate of a corresponding model can be a very hard task.

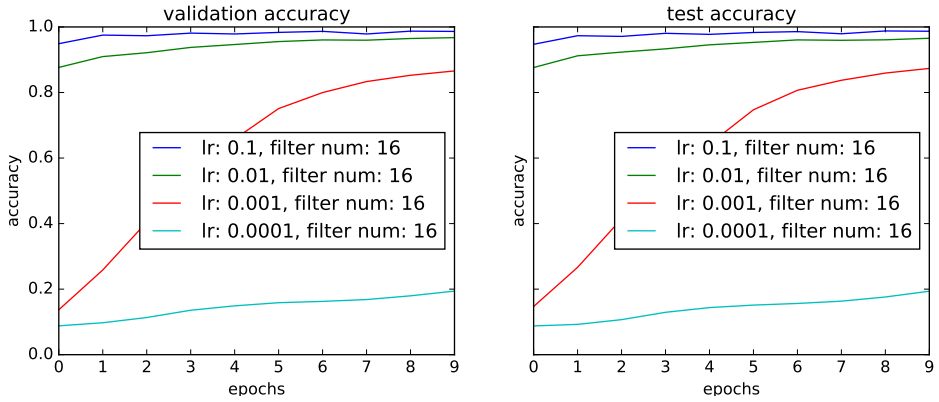

Figure 1: Changing the Learning Rate: SGD, number of filters 16, without dropout, all MNIST training samples are used, batch size 128

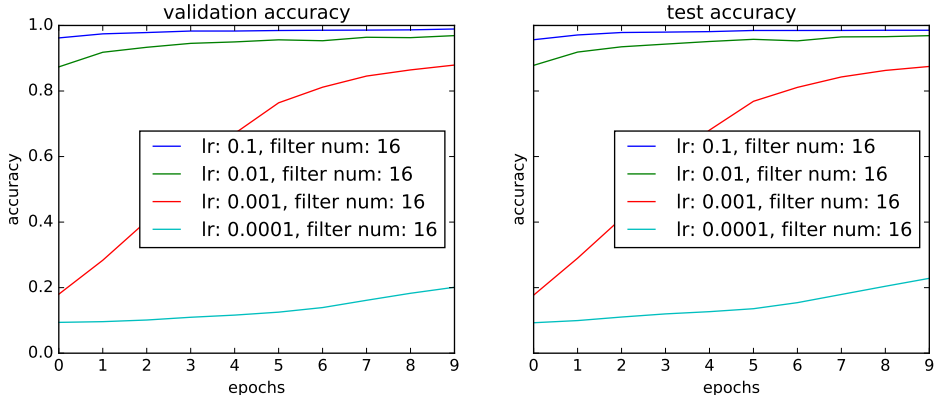

Figure 2: Changing the Learning Rate: SGD, number of filters 16, with dropout, all MNIST training samples are used, batch size 128

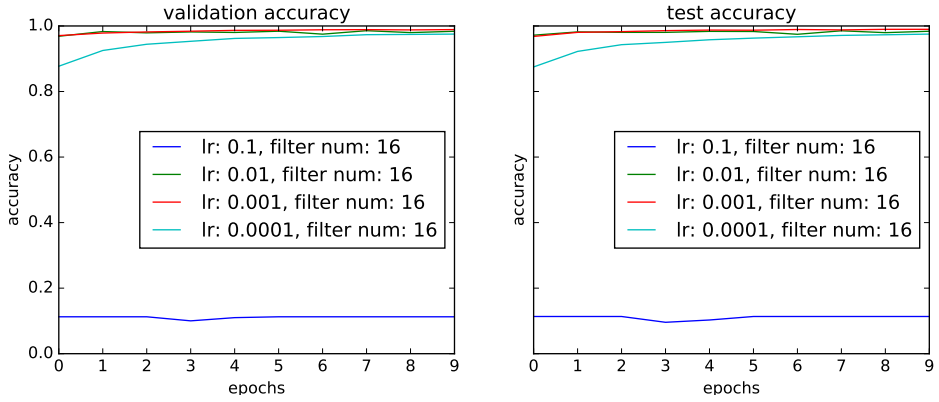

Figure 3: Changing the Learning Rate: Adam Optimization, number of filters 16, with dropout, all MNIST training samples are used, batch size 128

2.2 Runtime

The Training time for a CNN is crucial since they war normally trained with very large data sets. Not only the Hardware but also the network architecture in essential for the runtime. In the flowing we compare the runtime of different number of filters of the CNN on the CPU (3.49 GHz Intel Core i5) and GPU (NVIDIA GEForce GTX 970 4095 MB). The runtime is measured with the training time and the forward pass of the validation. In Figure 4 shows that training on the GPU is way faster compared to the CPU especially for a larger number of filters. In our case the bottleneck was comping the training Images form the CPU to the GPU. Also the usage of dropout increased the runtime.

Also the batch size has am impact on the runtime. In Figure 5 a small batch size is used which lead to a very long training time especially on the CPU.

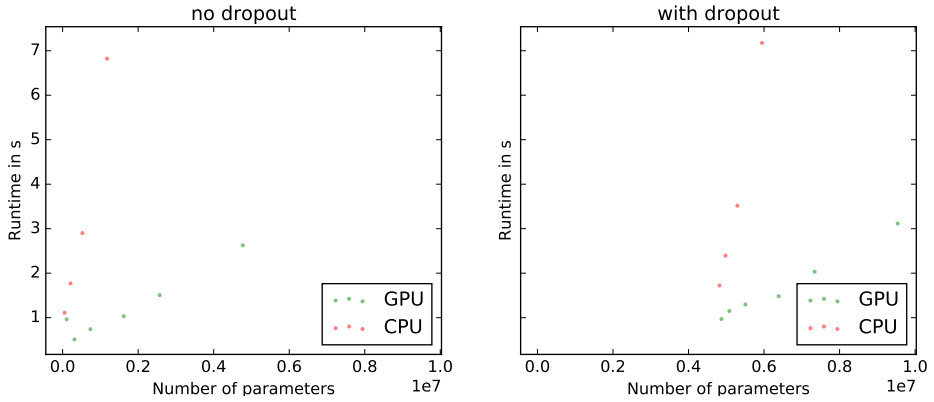

Figure 4: Runtime vs. Number of Parameter with and without dropout: 1 Epoch, 10000 training samples, batch size 64, 1 epoch, learning rate 0.1

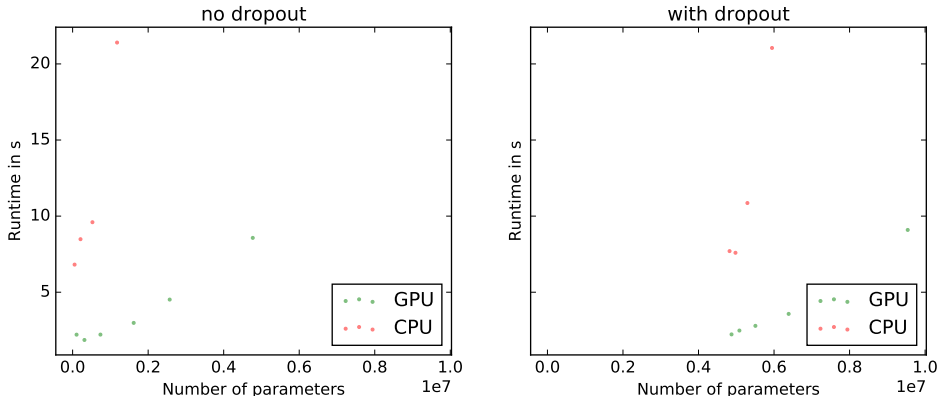

Figure 5: Runtime vs. Number of Parameter with and without dropout: 1 Epoch, 10000 training samples, batch size 16, 1 epoch, learning rate 0.1

3 Conclusion

In this Assignment we got familiar with the python Tensorflow framework. This was achieved by implementing a small convolutional neural network (CNN) and training it on the MNIST dataset. With the Architecture of the Network we can perform a classification task of digits. Furthermore the impact different hyperparameter on the performance and runtime was examined. Different learning rates, optimizer dropout and number of filter were evaluated.

With a CNN implementation in the python Tensorflow framework we could examine the effect of different hyperparameter and network architectures. Training the implemented network on the MNIST dataset the importance of the choose of the learning rate, architecture and optimizers could be observed and how they affect the performance of the network. Additional it could be shown how the use of different hardware and network architectures can dramatically effect the runtime.

List of Figures

List of Tables