Deep learning Lab
Uni Freiburg

Assignment 2
Maryna Kapitonova

Classifying MNIST with CNN in TensorFlow

Introduction

The task was to implement convolutional neural network (CNN) for MNIST digits classification using TensorFlow and evaluate its performance.

1. Implementation

CNN includes two convolutional layers (16 3x3 filters used with stride 1) with following ReLu activation interspersed with max pooling layer. After the convolutional layers a fully connected layer with 128 units and softmax layer with 10 units (for digits classes) are added. Network is trained by optimizing the cross-entropy loss with stochastic gradient descent. Validation performance is saved after each epoch (learning curves).

Convolutional layer:

- ➔ Accepts a volume of size W_1 x H_1 x D_1
- ➔ Requires four hyperparameters:
  - ◆ Number of filters K = 16 (default for 1st two tasks)
  - ◆ their spatial extent F
  - ◆ stride S
  - ◆ amount of zero padding P
- ➔ Produces a volume of size W2 x H2 x D2
  - ◆ W2 = (W1 - F + 2P)/S + 1
  - ◆ H2 = (H1 - F + 2P)/S + 1
  - ◆ D2 = K

With parameters sharing it introduces $F \cdot F \cdot D1$ weights per filter, for a total of $(F \cdot F \cdot D1) \cdot K$ weights and K biases

In the output volume, the d-th depth slice (of size W2xH2) is the result of performing valid convolution of the d-th filter over the input volume with a stride S, and then offset by d-bias

A common setting of the hyperparameters is $F=3,S=1,P=1$

Pooling layer

Makes the representations smaller and more manageable, operates over each activation map independently.

Results

All the computations were performed on following architecture:

CPU: **Intel(R) Core(™) i7-6700HQCPU 2.8 GHz**
GPU: **NVIDIA Quadro M1000M**

2. Changing learning rates

Default parameters here was number of filters 16, batch size = 100, epochs 100
Learning rates values [0.1, 0.01, 0.001, 0.0001]
The resulting curves are depicted in Fig.2.1

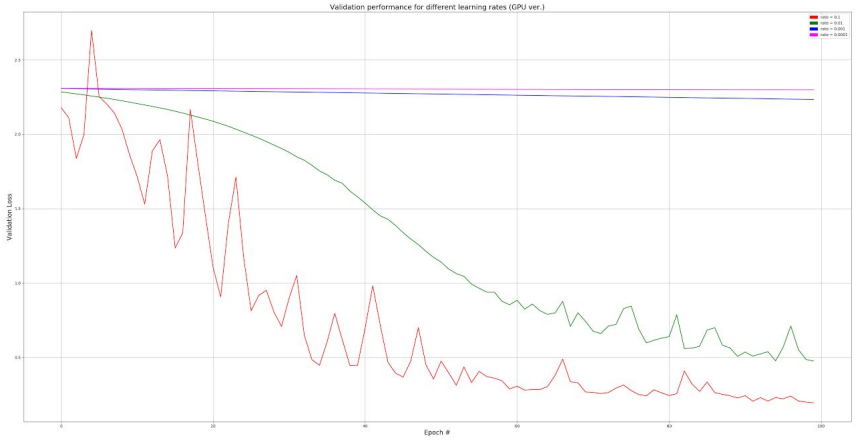

Fig. 2. 1. Validation performance after each epoch for different learning rates (GPU)

From obtained results follows that the most appropriate learning rate is 0.1, because of the fact that we've got the smallest validation error in reasonable time. Learning rate 0.01 converges a bit slowly, and other two values are almost constant at least for chosen numbers of the epochs, gradient has small steps in that cases.
Therefore, for future computation learning rate of 0.1 was picked.

3. Runtime check

The aim of the task was to compare runtime for CPU and GPU with filter sets [8, 16, 32, 64] and [8, 16, 32, 64, 128, 256] correspondingly.
Both computations were performed for learning rate 0.1, batch size 100 and 20 epochs.
I've obtained following results:

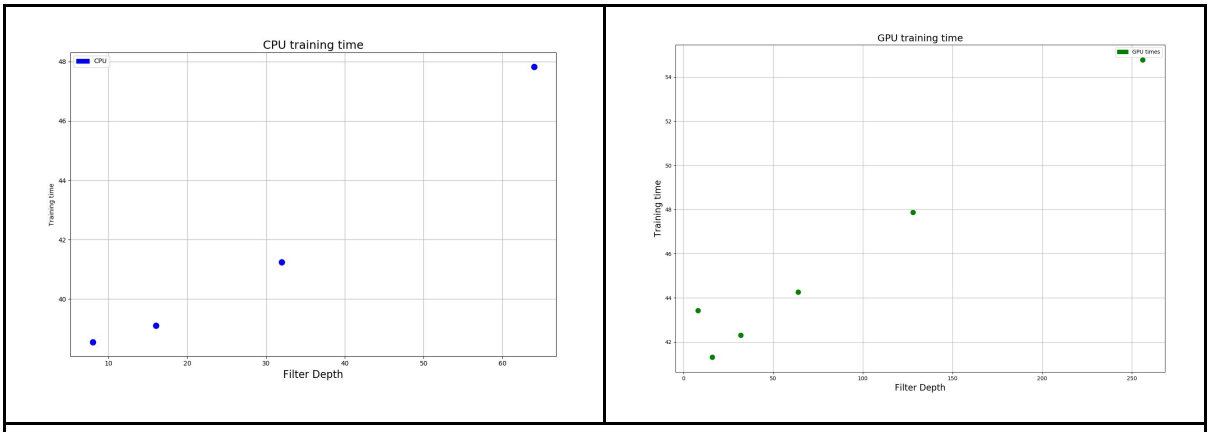

Fig. 3.1. Filter Depth vs. Training time [ms]

For my configuration GPU was only a little faster than CPU, I've checked this several times. Because I had issue changing a device while running GPU version of tensorflow (all the tricks from stackoverflow and similar resources resulted in running computations on GPU instead of CPU), I've installed CPU version only as well and made same experiments using it. Afterwards I saved data and made a scatter plot for both test cases:

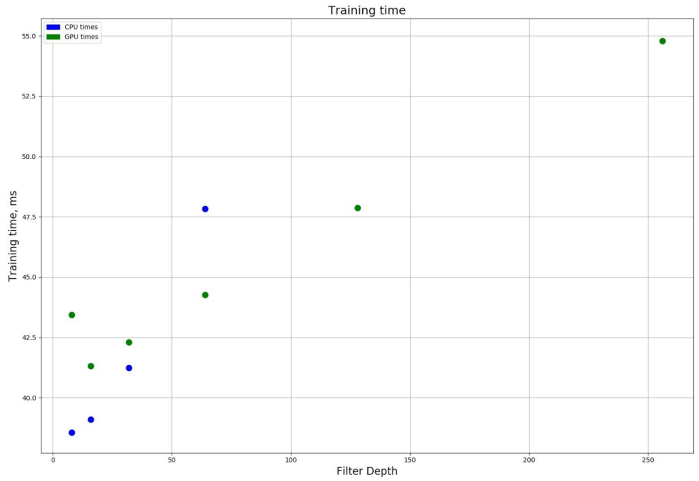

Fig.3.2. CPU and GPU runtime vs filter depth

However, GPU computations supposed to be much faster than CPU. You can decrease runtime of the last one using all the CPU cores and adapting parallel computing techniques. Deep learning as well involves lots of matrix multiplications (for ex. backpropagation), which are speed up on GPU's.

We can see as well that runtime increases while increasing total number of parameters (number of parameters can be calculated using information in implementation section and summing up across the layers.

filters	8	16	32	64	128	256
parameters	52330	104250	211690	440394	953098	2199690

Conclusions:
In this assignment Tensorflow framework and implementation of simple CNN was introduced; GPU vs. CPU performance characteristics checked.

Used resources:
https://www.tensorflow.org/
https://www.tensorflow.org/tutorials/layers (network skeleton)
http://cs231n.github.io/convolutional-networks/