Deep Learning Lab

Exercise 2:Implementing a CNN in Tensorflow

Manav Madan (13.11.2018)

Introduction:

In this exercise, a Convolutional neural network was build, which involved working with The MNIST database of handwritten digits. The basic API of Tensorflow library is used. Which is a Deep learning framework from Google, and a scaled down version of ”LeNet” is built. The task was to built the model,train the model and test it by further optimizing the different parameters such as learning rate,number of filters, filter size and batch size.

Structure:

CNN consists of two convolutional layers (16 3 × 3 filters and a stride of 1), each followed by ReLU activation function and a max pooling layer (pool size 2). After the convolution layers we add a fully connected layer with 128 units and a softmax layer for the classification.
With above specification the different size of different layers are:

1) Input/conv1-input: (batch_size,28,28,1) (extracting randomly from 50,000 images).
2) Conv1-output/Pooling1-input: (batch_size,28,28,16)
3) Pooling1-output/Conv2-input: (batch_size,27,27,16)
4) Conv2-output/Pooling2-input: (batch_size,27,27,16)
5) Pooling2-output/Denselayer-input: (batch_size,26,26,16)
6) Flatten it.

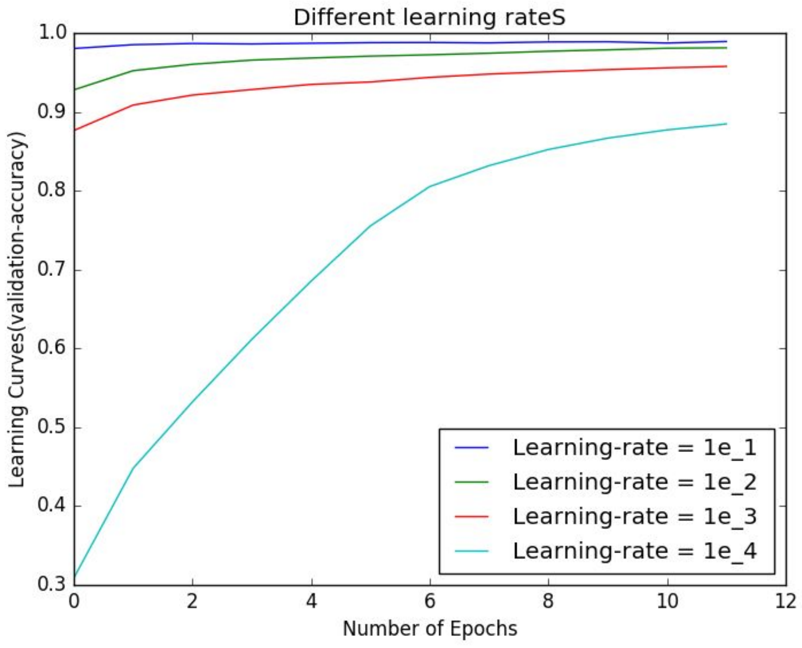
Problems Faced:

There were many errors which were faced during the training phase, here are some defined which must be discussed in detail.

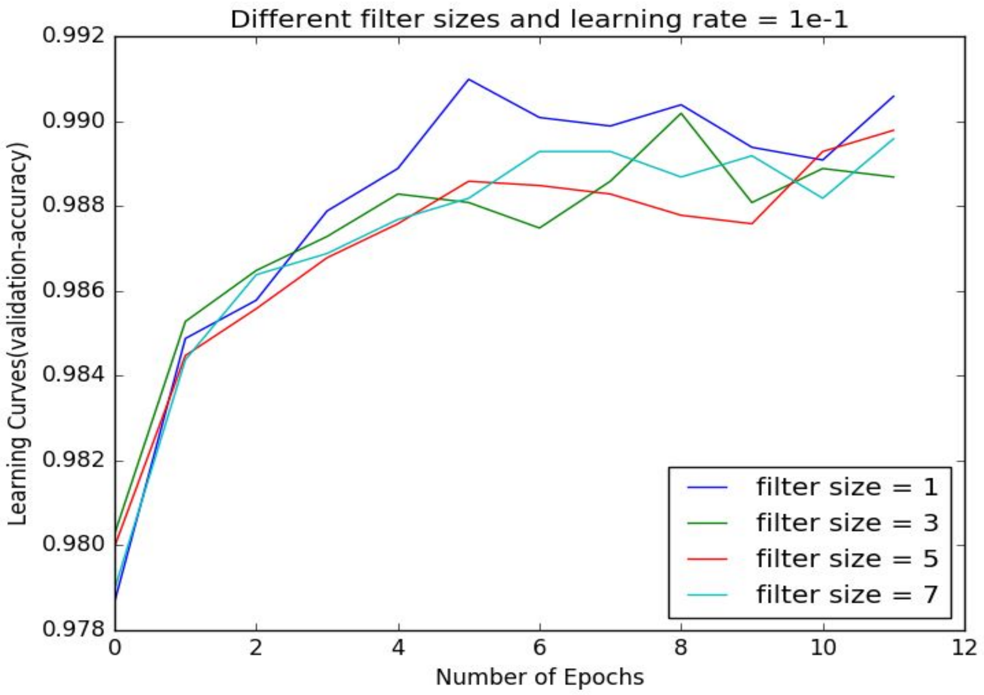
1) **Resource Exhausted**/Graph cannot exceed more than 2 gb
   Cant calculate the validation error for all the different rates in a for loop, won't run for more than 3 epochs.
2) **ValueError:** Input graph and Layer graph are not the same: Tensor("x_image:0", shape=(?, 28, 28, 1), dtype=float32) is not from the passed-in graph.
   In the formation of placeholders which were initially done globally, but when using the train_validate method in randomsearch.py. The placeholders were not found so with the tutors help, it was resolved. Instead of defining them globally now they are defined inside the function (train_vaidate).

Results:

The graph depicting the learning rate, shows that the best performance for the 0.1 value of learning rate and the worst was with 0.0001. The graph was plotted with the number of epochs(12) and the different learning rates(0.1, 0.01, 0.001, 0.0001), further values are saved in four different .json files keeping the filter size constant to 3.



The graph for different filter size was plotted with learning rate 0.1, with different filter-sizes (1,3,5,7).

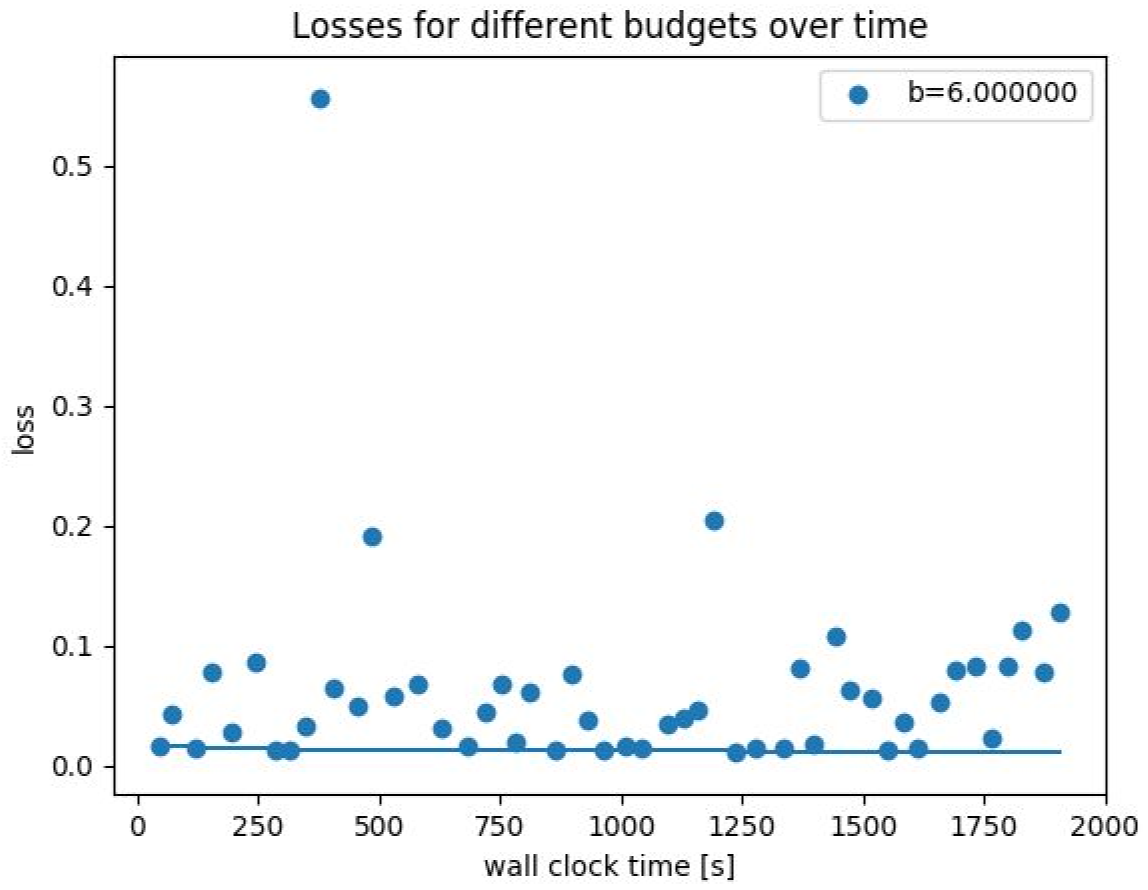


In the final task we optimized the hyperparameters

- ➔ learning rate ∈ [10 −4 , 10 −1] on log scale.
- ➔ batch size ∈ [16, 128] on log scale.
- ➔ number of filters ∈ [8 , 64] on log scale.
- ➔ Filter size ∈ {3, 5}

Using random search from the hpbandster package for 50 iterations and 6 epochs.

Best found configuration: {'learning_rate': 0.06848022352240996, 'num_filters': 17, 'filter_size': 3, 'batch_size': 34} and with test_error of the best model is: 0.0386


Losses for different budgets over time
b=6.000000
loss
wall clock time [s]