Exercise 3

Ilya Loshchilov
ILYA.LOSHCHILOV@gmail.com

Due: 09.12.2016 14:00

In this exercise we will study how the training of deep neural networks is affected by certain hyperparameters. More specifically, we investigate them in different scenarios independently and in combinations. The source code to run most experiments is already implemented for you. You will have to write a report describing your analysis of results. The code to make plots is available in MatLab, you may use it but you don't have to use it. Instead, you may write your own code for plotting. Please submit all your raw data (zipped .txt files) together with the report.

Scenario #1. Learning rate of SGD.

We train a CNN (its structure is given in build_cnn() function) on 5000 MNIST training examples under different settings of the learning rate $LR \in [0.001;\ 0.01;\ 0.1;\ 0.2;\ 0.3;\ 0.5;\ 1.0, 2.0]$.

Run the provided Lasagne code with "iscenario" set to 1. It will perform 3 independent runs for each setting of LR and store some statistic in text files. Follow the code and investigate what it does.

Plot the results using the provided MatLab code with "iscenario" set to 1. More specifically, add 3 plots to your report showing all performed runs where i) x-axis: epochs; y-axis: training loss, ii) x-axis: epochs: validations loss, ii) x-axis: epochs, y-axis: validation error.

Please try to explain in 1 or 2 sentences why certain hyperparameter values work better in terms of training loss vs epochs. More specifically, please try to characterize what happens when $LR = 0.001$ and $LR = 2.0$.
Please describe ***ALL*** sources of stochasticity of the training process (why curves of different runs are not exactly the same).

Scenario #2. Batch size.

We consider three settings for $LR \in [0.1;\ 0.2;\ 0.5]$ and three settings for batch size $BS \in [20;\ 100;\ 500]$. Run the provided Lasagne code with "iscenario" set to 2. It will perform one run for each combination of hyperparameter settings and store some statistics in text files. Follow the code and investigate what it does.

Plot the results using the provided MatLab code with "iscenario" set to 2.
Make a plot which includes 9 combinations/curves where x-axis: epochs, y-axis: training loss.

Investigate and describe which learning rates work best for each setting of batch size. Please make a guess how batch size and learning rate might be interdependent. Take into account that the greater the batch size the more steps SGD can perform within one epoch.
Make a plot where x-axis: training time, y-axis: training loss.
Please explain why curves ends at different positions on x-axis and why these positions are not different by a factor of 5 (e.g., some T, 5T and 25T for batch size 500, 100 and 20, respectively).

Scenario #3. SGD with momentum.

We consider three settings for $LR \in [0.001;\ 0.01;\ 0.1]$ and three settings for momentum factor $M \in [0;\ 0.5;\ 0.9]$. Run the provided Lasagne code with "iscenario" set to 3. It will perform one run for each combination of hyperparameter settings and store some statistics in text files. Follow the code and investigate what it does.

Plot the results using the provided MatLab code with "iscenario" set to 3.
Make a plot which includes 9 combinations/curves where x-axis: epochs, y-axis: training loss.
Investigate and describe which learning rates and momentum factors work best. Please don't forget to have a look at Lasagne's documentation of SGD with momentum. If momentum is useful, please describe why it might be the case.

Scenario #4. Number of filters.

We consider several settings for the number of convolutional filters $NF \in [1;\ 2;\ 4;\ \ldots; 256]$ per layer of our CNN defined in build_cnn() function. Run the provided Lasagne code with "iscenario" set to 4. It will perform one run for each combination of hyperparameter settings and store some statistics in text files. Follow the code and investigate what it does.

Plot the results using the provided MatLab code with "iscenario" set to 4.
Make a plot which includes all combinations/curves where x-axis: epochs, y-axis: training loss.
Investigate and describe how training curves change w.r.t. the number of filters used. Which number of filters you would select if in all cases you are allowed to train your network for 50 epochs.
Make a plot where x-axis: training time, y-axis: training loss. Please describe the difference between the two plots. Which number of filters you would use when your time budget is limited to 10 seconds.
Make a plot where x-axis: training time, y-axis: validation error. What this plot suggests?

Scenario #5. Random search.

We consider random search on a four-dimensional problem where the search space is defined as follows: $NF \in [10, 100]$, $LR \in [10^{-2}, 10^{-0.5}]$, $M \in [0, 1.0]$ and $BS \in [2^4, 2^8]$. Run the provided Lasagne code with "iscenario" set to 5. It will evaluate 100 random hyperparameters settings and store some statistics in text files. Follow the code and investigate what it does.

Plot the results using the provided MatLab code with "iscenario" set to 5.

After you run random search with 200 random hyperparameter settings, you may *simulate* additional (e.g., 1000) runs of random search by reshuffling values of your actual run. Make a plot which shows up to 20 simulated runs of random search and median performance of e.g. 1000 simulated runs where x-axis: number of evaluations, y-axis: best validation error seen so far (i.e., at t-th evaluation we plot what was the best error at evaluations $1, \dots, t$).

Please try to analyze which hyperparameter settings tend to lead to better validation errors. You can do it coordinate-wise (i.e., considering one hyper-parameter at a time) and/or considering multiple/all hyperparameters. This is up to you.

Scenario #6. Parallel evaluations.

Use Python to write two pieces of code (Master and Worker) which perform synchronous file-based communication of hyperparameter settings and their simulation results. More specifically, Master code will generate text files with hyperparameter settings. Worker code will read hyperparameter settings, evaluate them with CNNs and create simulation results in text files. Once the files are created, Master will take the results into account and generate new hyperparameter settings, write them to files, etc.
Each group should have 2 GPUs at their disposal, so you can use them to run two workers and validate your code with random search.