Deep Learning
MiniProject #2
M. Hagan, A. Jafari
Mar-2-2017

Training Convolution networks with Caffe on MNIST dataset

The objective of this miniproject is to become familiar with Caffe for training deep convolution networks on large datasets. You will be provided with a sample program that loads in the MNIST data set and sets up an example convolution network to be trained on it. (See http://caffe.berkeleyvision.org/gathered/examples/mnist.html for a discussion of this example.) The instructions below provide a rough framework of what you should do for the miniproject. You should experiment with using Caffe on this large dataset. The project is open-ended. Learn as much as you can about using deep convolution networks, and relate what you have learned in your project report. For all parts below, and any other experiments you run, include the results into one PDF file, and upload it to the dropbox on D2L. Include all program listings, plots, command line printouts, discussion, etc.

1. Download `train_mnist.py`, `get_mnist.sh` and `create_mnist.sh` from D2L and put them in the directory where you want to run your programs.

2. Run the program `train_mnist.py` in PyCharm and investigate and verify its performance. You may need to change the line ”`my_root =`” to the appropriate path. Also, the following lines only need to be run once, to create the mnist data set. After the first run, you can comment these lines.

```
os.system("sh get_mnist.sh")
os.system("sh create_mnist.sh")
```

3. Investigate the kernels in the two convolution layers. Can you identify kernels that would be useful for particular numerals?

4. How does the performance of the convolution network compare with the multilayer networks that you used in MiniProject #1?

5. Change the size of the minibatches (`batch_size` parameter). If you make the batch size very large, does it affect the computation time significantly? Describe the advantages and disadvantages of increasing the batch size. Find a good choice.

6. Use a dropout layer at layer `fc1`. Make `fc1` the top and the bottom for the dropout layer. (See https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf for a description of dropout.) Does dropout improve the testing error?

7. Experiment with different numbers of layers and different numbers of kernels. Maintain the total number of weights and biases in the network, while increasing the number of layers in the network. Describe how the performance changes as the number of layers increases – both in terms of training time and performance.

8. Try one other training function from the list on this page: `http://caffe.berkeleyvision.org/tutorial/solver.html`. Compare the performance with gradient descent.