LIGHTMATTER

Programming Assignment
Software Engineering Candidate

Congratulations on reaching the next stage of your Lightmatter interview! In this stage, we would like to evaluate your ability to tackle the types of programming problems you will be working on at Lightmatter. First, some general guidelines:

- We would appreciate comments so we know what you were thinking when you wrote your code!
- At Lightmatter, we'd like you to use git for version control. For that reason, please create a **private** repository on bitbucket.org, and invite Darius (dbunandar) and Tomo (lazovich) to be collaborators to your repository. You will submit your assignment there.
- You can use either Python 2 or 3, and/or Julia.
- We don't expect you to spend days and days building production quality code or anything like that. We're just looking for you to demonstrate your understanding of the machine learning concepts and your ability to write code to implement them.
- We would like to see a short (maximum 3 pages with 10 pt. font) report explaining what you have implemented. If you're familiar with LaTeX, you might want to use it to write this report.
- You are allowed to use *any* resource (books, internet, etc.) you wish to use, but you're not allowed to ask anyone.

Learning handwritten digits with convolutional neural network

In this part, you will be writing code that implements the math behind training a deep neural network.

Part 1 - Example code

In this first part, you will be using Tensorflow which includes convenient tool sets for designing deep neural networks. We will start from Tensorflow for beginners:
https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners

Download the example code:
https://github.com/tensorflow/tensorflow/blob/r1.1/tensorflow/examples/tutorials/mnist/mnist_softmax.py
Every single line in this code is explained in Tensorflow's MNIST for beginners page.

Make sure you can run the code. You should achieve an accuracy of about 92% with this fully connected network. Include a plot of your training loss and accuracy (test loss) in your report. Congratulations! You just implemented a feed forward neural network.

Part 2 - Improving your Tensorflow code

Let's improve the Tensorflow code slightly by changing the neural network from a simple fully connected layer to a more sophisticated network.

Implement the following deep neural network:

1. 2D convolution layer:
    a. 2D convolution with a 10x10 filter, a stride of 1, and 32 output channels.
    b. Add a bias to each output channel. There should only be 32 bias values, one for each output channel.
    c. ReLU activation function.
    d. 2x2 max pooling layer.
2. 2D convolution layer:
    a. 2D convolution with a 5x5 filter, a stride of 1, and 16 output channels.
    b. Add a bias to each output channel. There should only be 16 bias values.
    c. ReLU activation function.
    d. 2x2 max pooling layer.
    e. Flatten the output of layer 2 into a vector.
3. Fully connected layer:
    a. Output neurons of 1024, and with a bias vector.
    b. ReLU activation function.
4. Fully connected layer:

   a. Output neurons of 10, and with a bias vector.
   b. Softmax activation function.

5. Cross entropy loss.

Layers 5 and 6 are similar to what you implemented in Part 1. Please also follow the following restrictions:

1. For layers 1 and 2, you are only allowed to use tf.nn.conv2d, tf.nn.relu, and tf.nn.max_pool. You are prohibited from using anything within tf.layers.
2. Initialize all filters and weight matrices using tf.truncated_normal with standard deviation of 0.1.
3. Initialize all bias vectors with a constant of value 0.1.
4. Train using the AdaGrad algorithm. You might need to change the initial learning rate.
5. Train with a batch size of 100 and upto 200 epochs.

Now that you have all the pieces in place, train your neural network on the dataset and visualize the results. You should get an accuracy of about 94% which is slightly better than the vanilla fully connected network.

Part 3 - Your own deep learning software

Implement the network in Part 2 using only numpy. You are not allowed to use Tensorflow, scikit-learn, or any other machine learning package to help you. This part requires you to understand backpropagation, stochastic gradient descent, and some basic calculus. The code can easily bloat up, so using objects/classes can help you manage the development. Train your network on the dataset and retrieve the accuracy you had in Part 2.