In this assignment you need to implement a feedforward neural network in python (we strongly recommend using numpy for all matrix/vector operations). This network will be trained and tested using the MNIST handwritten digit recognition dataset [1]. Specifically, given an input image (28 x 28 = 784 pixels) from the MNIST dataset, the network will be trained to classify the image into 1 of 10 classes (10 digits). Your implementation should support the use of the following hyper-parameters/options :

- --lr (initial learning rate η for gradient descent based algorithms)
- --momentum (momentum to be used by momentum based algorithms)
- --num_hidden (number of hidden layers - this **does not include** the 784 dimensional input layer and the 10 dimensional output layer)
- --sizes (a comma separated list for the size of each hidden layer)
- --activation (the choice of activation function - valid values are tanh/sigmoid)
- --loss (possible choices are squared error[sq] or cross entropy loss[ce])
- --opt (the optimization algorithm to be used: gd, momentum, nag, adam - you will be implementing the mini-batch version of these algorithms)
- --batch_size (the batch size to be used - valid values are 1 and multiples of 5)
- --anneal (if true the algorithm should halve the learning rate if at any epoch the validation loss decreases and then restart that epoch)
- --save_dir (the directory in which the pickled model should be saved - by model we mean all the weights and biases of the network)
- --expt_dir (the directory in which the log files will be saved - see below for a detailed description of which log files should be generated)
- --mnist (path to the mnist data in pickeled format [2])

You should use the `argparse` module in python for parsing these parameters.

Instructions:

- You need to submit the source code for the assignment. Your code should include one file called train.py which should be runnable using the following command:

  python train.py --lr 0.01 --momentum --0.5 --num_hidden 3 --sizes 100,100,100 --activation sigmoid --loss sq --opt adam --batch_size 20 --anneal true --save_dir /home/mitesh/pa1 --expt_dir /home/mitesh/pa1/exp1
  --mnist /home/mitesh/pa1/data/mnist.pkl.gz

[1] `http://deeplearning.net/tutorial/gettingstarted.html`

[2] in the same format as available here `http://deeplearning.net/data/mnist/mnist.pkl.gz`

Of course, the actual values for the options can change (for example, we could try different learning rates, momentums, optimization algorithms, etc.). For the remainder of this document we will assume that the above command is saved as a shell script in run.sh (from now on if we refer to run.sh then it means we are referring to the above command).

- The mnist.pkl.gz file available at the above mentioned url contains 70000 MNIST images split into a train (50000), valid (10000) and test set (10000). Refer to this url[3] to see how to read this pickled file.

- Your code should create the following log files: log_loss_train.txt, log_loss_valid.txt, log_loss_test.txt, log_err_train.txt, log_err_valid.txt, log_err_test.txt in the expt_dir.

- The log_loss_*.txt files should contain the loss (squared error or cross entropy as specified) on the train/valid/test data respectively after every 100 steps (refer to the Lecture slides for the definition of a step). Each log_loss_*.txt should contain only the following lines:

  Epoch 0, Step 100, Loss: ⟨value⟩, lr: 0.01
  Epoch 0, Step 200, Loss: ⟨value⟩, lr: 0.01
  Epoch 0, Step 300, Loss: ⟨value⟩, lr: 0.01
  Epoch 0, Step 400, Loss: ⟨value⟩, lr: 0.01
  ...
  ...
  Epoch 1, Step 100, Loss: ⟨value⟩, lr: 0.01
  ...
  ...
  Epoch 2, Step 100, Loss: ⟨value⟩, lr: 0.01
  ...
  ...
  Epoch ⟨max_epoch⟩, Step 100, Loss: ⟨value⟩, lr: 0.01
  ...
  ...
  (where lr is the learning rate which may change if you use annealing)

  You need to strictly adhere to the above format. These log files will be parsed using a script which can parse only those strings which fit the above pattern. If your log file does not adhere to this pattern or contains any additional lines then you will get a 0 on the assignment.

- The log_err_*.txt files should contain the error rate (*i.e.*, percentage of examples which were classified incorrectly) on the train/valid/test data respectively after every 100 steps (refer to the Lecture slides for the definition of a step). Each log_err_*.txt should contain only the following lines:

[3] http://deeplearning.net/tutorial/gettingstarted.html

Epoch 0, Step 100, Error: ⟨value⟩, lr: 0.01
Epoch 0, Step 200, Error: ⟨value⟩, lr: 0.01
Epoch 0, Step 300, Error: ⟨value⟩, lr: 0.01
Epoch 0, Step 400, Error: ⟨value⟩, lr: 0.01
...
...
Epoch 1, Step 100, Error: ⟨value⟩, lr: 0.01
...
...
Epoch 1, Step 100, Error: ⟨value⟩, lr: 0.01
...
...
Epoch ⟨max_epoch⟩, Step 100, Error: ⟨value⟩, lr: 0.01
...
...
(Error can take on a real value between 0 to 100 but round it off to two decimal places)

You need to strictly adhere to the above format. These log files will be parsed using a script which can parse only those strings which fit the above pattern. If your log file does not adhere to this pattern or contains any additional lines then you will get a 0 on the assignment.

- In addition, your code should also generate a valid_predictions.txt and test_predictions.txt file in the expt_dir. Each of these files should contain 10K lines. Each line should contain the predicted label of the corresponding test/valid image. Here's what a sample valid_predictions.txt file would look like:
  9
  3
  0
  2
  7
  ...
  ...

- Notice that you will have to initialize the weights of the network randomly. To ensure replicability of your results make sure that you set numpy.random.seed(1234) before initializing the weights and biases. If you don't do this we may get very different results when we run your code after submission.

- We anticipate that some of you may not be able to support all values of hyperparameters mentioned above. To help us evaluate only those options which are supported by your code you need to submit a file (supported.txt) listing the supported options for the following hyperparameters: anneal, opt, loss, activation. For example, if you have supported all possibles option for these hyperparameters then supported.txt will

contain the following contents:
--anneal: true,false
--opt: gd,momentum,nag,adam
--loss: sq,ce
--activation: tanh,sigmoid

However, if your code does not support tanh activation and adam and squared error loss then supported.txt will contain the following contents:
--anneal: true,false
--opt: gd,momentum,nag
--loss: ce
--activation: sigmoid
Again, the contents of this file should be exactly in the format specified above.

- You need a single tar.gz file containing the following:
  - train.py
  - run.sh (containing the best hyperparameters)
  - any other python scripts that you have written
  - supported.txt (as described above)

  The tar.gz should be named as <RollNo1>_<RollNo2>_backprop.tgz if there are two team members or as <RollNo>_backprop.tgz if you are doing the assignment alone.

Evaluation:

- Your task is to achieve an error rate of $\leq 1\%$ on the test data. You will be evaluated based on how close you get to achieving this error rate.
- Along with the source code you need to submit a run.sh file containing the command (with hyperparameters) that gave you the lowest error rate on the test set.
- In addition, we will also run your code using different hyperparameter configurations (for example, number of hidden layers, size of hidden layers, etc.). You will then be evaluated based on how good/bad your performance is compared to the performance of other teams on different hyperparameter configurations.
- And of course, you will also be evaluated based on which of the specified hyperparameters are supported correctly by your code.