EECS 4404E/5327
Assignment 2: Neural Networks (15 pts)

Due date: **Sunday, November 17 by 18:00**

Submission: Submit a .zip package of your work including a single pdf file of your assignment with your solutions, each question at a new page, plus a folder containing your TensorFlow code, each question as a separate .py file, on Moodle's respective assignment tab. Make sure you write your name, student ID, and assignment# on each of the file.

Objectives:

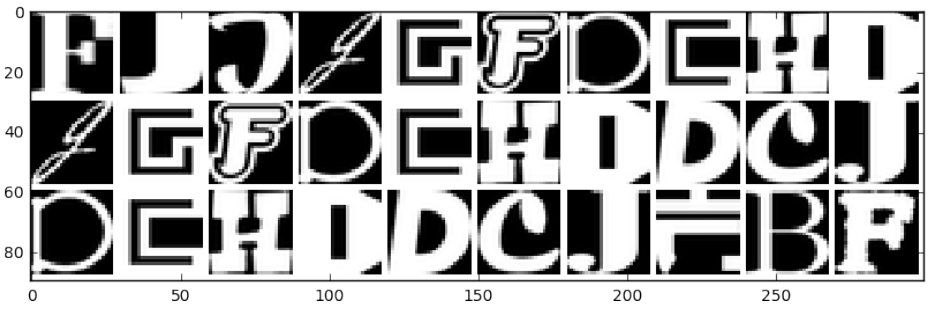
The purpose of this assignment is to investigate the classification performance of neural networks. In this assignment, you will gain some experience in training a neural network and will use an effective way to avoid overfitting. All the implementations need to be done using Python and TensorFlow. For consistency, use **TensorFlow 1.15 version** (either the CPU or GPU version). More info can be found https://www.tensorflow.org/install/gpu. You are encouraged to look up TensorFlow APIs for useful utility functions, at: https://www.tensorflow.org/versions/r1.15/api_docs/python/. Also, look for a quick installation and guide at Moodle and under Practical Materials > TensorFlow Materials.

Note. You must write **vectorized** TensorFlow function using the provided API by TensorFlow, i.e., define operations, matrices, etc. in tf format so it uses the optimized backend for both CPU and GPU. For instance, `tf.matmul(a, b)` for multiplying tensors of matrices a and b.

notMNIST Dataset

The dataset that we will use in this assignment is a permuted version of notMNIST[1], which contains 28-by-28 images of 10 letters (A to J) in different fonts. This dataset has 18720 instances, which can be divided into different sets for training, validation and testing. The provided file is in **.npz** format which is for Python. You can load this file as follows.

[1] http://yaroslavvb.blogspot.ca/2011/09/notmnist-dataset.html




```
with np.load("notMNIST.npz") as data:
    Data, Target = data ["images"], data["labels"]
    np.random.seed(521)
    randIndx = np.arange(len(Data))
    np.random.shuffle(randIndx)
    Data = Data[randIndx]/255.
    Target = Target[randIndx]
    trainData, trainTarget = Data[:15000], Target[:15000]
    validData, validTarget = Data[15000:16000], Target[15000:16000]
    testData, testTarget = Data[16000:], Target[16000:]
```

1 Neural Networks [15 pt.]

In this part, we use a neural network to classify the letters in the dataset. In all of the tasks, use ReLU activation functions, a cross-entropy cost function, and a softmax output layer. As an estimate of the running time, training the following neural networks will not take more than an hour on an Intel core i7 1.73-GHz CPU with 4 GB of RAM. Because you will be running a few experiments, it is a good idea to save ("checkpoint") your model during training at 25%, 50%, 75% and 100% of the training process, to four separate models. The function *tf.saver* can come in handy for checkpointing and you may reload your model later for inspection or use.

1.1 Feedforward fully connected neural networks [6 pt.]

Implement a simple neural network with one hidden layer and 1000 hidden units. Train your neural network on the entire notMNIST training set of ten classes. Because the neural network loss functions are non-convex, a proper weights initialization scheme is crucial to prevent vanishing gradient during back-propagation as a result of learning stuck at a plateau at the beginning of the training. You will use the Xavier initialization to initialize the weight matrices for all the neural networks in this assignment. That is, each weight matrix is initialized from zero-mean independent Gaussians whose variance is $3/(\#input\ units + \#output\ units)$. Unlike the weight matrices, the bias units will be initialized to zero.

1. **layer-wise building block:** Write a vectorized Tensorflow Python function that takes the

hidden activations from the previous layer then return *the weighted sum of the inputs*(i.e. the **z**) for the current hidden layer. You will also initialize the weight matrix and the biases in the same function. You should use Xavier initialization for the weight matrix. Your function should be able to compute the weighted sum for all the data points in your mini-batch at once using matrix multiplication. It should not contain loops over the training examples in the mini-batch. The function should accept two arguments, the input tensor and the number of the hidden units. Include the snippets of the Python code. [3 pt.]

2. **Learning:** Use your function from the previous question to build your neural network model with ReLU activation functions in TensorFlow and *tf.nn.relu* can be useful. For training your network, you are supposed to find a reasonable value for your learning rate. You should train your neural network for different values of learning rate and choose the one that gives you the fastest convergence in terms of the training loss function. (You might want to "babysit" your experiments and terminate a particular run prematurely as soon as you find out that the learning rate value is not very good.) Trying 3 different values should be enough. You may also find it useful to apply a small amount of weight decay to prevent overfitting. (e.g. $\lambda = 3e - 4$). On the training set, validation set and test set, record your classification errors and cross-entropy losses after each epoch. Plot the training, validation, and test classification error vs. the number of epochs. Make a second plot for the cross-entropy loss vs. the number of epochs. Comment on your observations. [2 pt.]

3. **Early stopping:** *Early stopping* is the simplest procedure to avoid overfitting. Determine and highlight the early stopping point on the classification error plot from question 1.1.2, and report the training, validation and test classification error at the early stopping point. Are the early stopping points the same on the two plots? Why or why not? Which plot should be used for early stopping, and why? [1 pt.]

1.2 Effect of hyperparameters [4 pt.]

1. **Number of hidden units:** Instead of using 1000 hidden units, train different neural networks with [100, 500, 1000] hidden units. Find the best validation error for each one. Choose the model which gives you the best result, and then use it for classifying the test set. Report the test classification error. In one sentence, summarize your observation about the effect of the number of hidden units on the final results. [2 pt.]

2. **Number of layers:** For this task, train a neural network with two hidden layers of 500 hidden units each (1000 total). Plot the training and validation errors (or training and validation classification errors) vs. the number of epochs. What is the final validation error (and the validation classification error) when training is complete? Using the test set, compare this architecture with the one-layer case. [2 pt.]

1.3 Regularization and visualization [5 pt.]

Dropout is a powerful technique to reduce overfitting and to enhance the overall performance of the neural network. Using the same architecture in Sec. 1.1.

1. **Dropout**: Introduce dropout on the hidden layer of the neural network (with dropout rate 0.5) and train you neural network. As you know, dropout should only be used in the training procedure, and the "mean network" should be used instead during the evaluation. Plot the number of training and validation errors (or training and validation classification errors) vs the number of epochs. Compare the results with the case that we do not have any dropout. Summarize the observed effect of dropout. (You may want to use the *tf.nn.dropout* utility function.) [2 pt.]

2. **Visualization**: It is generally hard to figure out what the hidden neurons are doing in a neural network. However, one can visualize the incoming weight vector to each hidden neuron in the first hidden layer as an image and inspect what that neuron has learnt from the dataset. For the neural networks we have trained in this assignment, we can visualize the values of the 28x28 incoming weights of each hidden units in the first layer as a grey-scale image. For the models trained with and without dropout, checkpoint the model during training at 25%, 50%, 75% and 100% of the early stopping point. For each of the checkpoints, make a figure visualizing the 1000 neurons in the first hidden layer. Comment on the learning progression of the hidden neurons, and comment on any difference between the dropout model and the one without dropout. [3 pt.]