ECE421
Assignment 2:
Neural Networks

Due date: March 6, 2019

Submission: A hard copy of the report should be handed in to Sowndarya Sundar, BA3014

The code should be submitted electronically to: ece421ta2019@gmail.com

Objectives:

The purpose of this assignment is to investigate the classification performance of neural networks. You will be implementing a neural network model using Numpy, followed by a state-of-the-art implementation in Tensorflow. You are encouraged to look up TensorFlow APIs for useful utility functions, at: https://www.tensorflow.org/api_docs/python/.

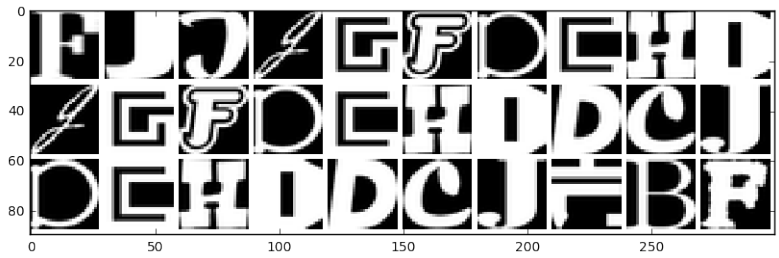
Guidelines:

- Full points are given for complete solutions, including justifying the choices or assumptions you made to solve each question. Both a written report and your complete source code (as an appendix and separate files) should be included in the final submission.
- Homework assignments are to be solved in groups of two. You are encouraged to discuss the assignment with other students, but you must solve it within your own group. Make sure to be closely involved in all aspects of the assignment. Please indicate the contribution percentage from each group member at the beginning of your report.

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:]
return trainData, validData, testData, trainTarget, validTarget, testTarget
```

Since you will be investigating multi-class classification, you will need to convert the data into a one-hot encoding format. The code snippet below will help you with that.

```
def convertOneHot(trainTarget, validTarget, testTarget):
    newtrain = np.zeros((trainTarget.shape[0], 10))
    newvalid = np.zeros((validTarget.shape[0], 10))
    newtest = np.zeros((testTarget.shape[0], 10))

    for item in range(0, trainTarget.shape[0]):
        newtrain[item][trainTarget[item]] = 1
    for item in range(0, validTarget.shape[0]):
        newvalid[item][validTarget[item]] = 1
    for item in range(0, testTarget.shape[0]):
        newtest[item][testTarget[item]] = 1
    return newtrain, newvalid, newtest
```

1 Neural Networks using Numpy [14 pts.]

In this part, you will be tasked with implementing and training a neural network to classify the letters using Numpy and Gradient Descent with Momentum. The network you will be implementing has the following structure:

- 3 layers - 1 input, 1 hidden with ReLU activation and 1 output with Softmax
- Cross Entropy Loss

During the training process, it may be beneficial to save weights to a file during the training process - the function *numpy.savetext* may be useful. As an estimate of the running time, training the Numpy implementation should not take longer than an hour (tested on an Intel i7 3770K at 3.40 GHz and 16 GB of RAM). For this part only, **Tensorflow implementations will not be accepted**.

1.1 Helper Functions [4 pt.]

To implement the neural network described earlier, you will need to implement the following *vectorized* (i.e. no for loops, they must rely on matrix/vector operations) helper functions. Include the snippets of your Python code in the report.

1. *ReLU()*: This function will accept one argument and return Numpy array with the ReLU activation and the equation is given below. [0.5 pt]

$$ReLU(x) = max(x, 0)$$

2. *softmax()*: This function will accept one argument and return a Numpy array with the softmax activations of each of the inputs and the equation is shown below. [0.5 pt]

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1...K, \text{ for K classes.}$$

3. *compute()*: This function will accept 3 arguments: a weight, an input, and a bias matrix and return the product between the weights and input, plus the biases (i.e. a prediction for a given layer). [0.5 pt]

4. *averageCE()*: This function will accept two arguments, the targets (e.g. labels) and predictions - both are matrices of the same size. It will return a number, average the cross entropy loss for the dataset (i.e. training, validation, or test). For K classes, the formula is shown below. [0.5 pt]

$$averageCE = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} t_k^n log(s_k^n)$$

Here, t_k is the groundtruth and s_k is the score (i.e. softmax of the k^{th} ~~example~~ class) and N is the number of examples.

5. *gradCE()*: This function will accept two arguments, the targets (e.g. labels) and the outer layer predictions. It will return the gradient of the cross entropy loss with respect to the softmax of the outer layer predictions (i.e. the probability labels). Code/derivations for the both the *average* and *total* cross entropy loss will be accepted. **Show the analytical expression in your report.** [2 pt.]

1.2 Backpropagation Derivation [4 pts.]

To train the neural network, you will need to implement the backpropagation algorithm. For the neural network architecture outlined in the assignment description, derive the following analytical expressions and include them in your report:

1. $\frac{\partial L}{\partial W_o}$, the gradient of the loss with respect to the outer layer weights. [1 pt.]
   - Shape: $(K \times 10)$, with K units
2. $\frac{\partial L}{\partial b_o}$, the gradient of the loss with respect to the outer layer biases. [1 pt.]
   - Shape: (1×10)
3. $\frac{\partial L}{\partial W_h}$, the gradient of the loss with respect to the hidden layer weights. [1 pt.]
   - Shape: $(F \times K)$, with F features, K units
4. $\frac{\partial L}{\partial b_h}$, the gradient of the loss with respect to the hidden layer biases. [1 pt.]
   - Shape: $(1 \times K)$, with K units.

Hints: The labels (y_{true}) have been one hot encoded. You will also need the derivative of the ReLU() function in order to backpropagate the gradient through the activation.

You may also wish to carry out your computations with the matrices transposed - this is also acceptable (although be careful when using the *np.argmax* function).

1.3 Learning [6 pts.]

Construct the neural network and train it for 200 epochs with a hidden unit size of 1000. First, initialize your weight matrices following the Xaiver initialization scheme (zero-mean Gaussians with variance $\frac{2}{unitsin+unitsout}$) and your bias matrices/vectors, each with the shapes as outlined in section 1.2. Using these matrices, compute a forward pass of the training data and then, using the gradients derived in section 1.2, implement the backpropagation algorithm to update all of the network's weights and biases. The optimization technique to be used for backpropagation will be Gradient Descent with momentum and the equation is shown below.

$$\boldsymbol{\nu_{new}} \leftarrow \gamma \boldsymbol{\nu_{old}} + \alpha \frac{\partial L}{\partial \boldsymbol{W}}$$
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boldsymbol{\nu_{new}}$$

For the $\boldsymbol{\nu}$ matrices, initialize them to the same size as the hidden and output layer weight matrix sizes, with a very small value (e.g. $1e^{-5}$). Additionally, initialize your γ values to values slightly less than 1 (e.g. 0.9 or 0.99).

Plot the training, validation and testing loss and accuracy curves and include them in your report. For the accuracy metric, the *np.argmax()* function will be helpful.

1.4 Hyperparameter Investigation [4 pts.]

1. **Number of hidden units:** Instead of using 1000 units, investigate the classification accuracy of neural networks with [100, 500, 2000] units. For each scenario report the test accuracy and summarize your observation about the effect on the number of hidden units. [2 pts.]
2. **Early stopping:** This technique is one of the simplest methods to control overfitting in neural networks. From the plots from Section 1.3, identify the early stopping point and report on the training, validation and test classification accuracies. [2 pts.]

2 Neural Networks in Tensorflow [14 pts.]

In this part, you will be implementing a Convolutional Neural Network, one of the state-of-the-art techniques for image recognition, using Tensorflow. It is recommended that you train the neural network using a GPU (although this is not required.) The neural network architecture that you will be implementing is as follows:

1. Input Layer
2. A 3×3 convolutional layer, with 32 filters, using vertical and horizontal strides of 1.
3. ReLU activation
4. A batch normalization layer
5. A 2×2 max pooling layer
6. Flatten layer
7. Fully connected layer (with 784 output units, i.e. corresponding to each pixel)
8. ReLU activation
9. Fully connected layer (with 10 output units, i.e. corresponding to each class)
10. Softmax output
11. Cross Entropy loss

2.1 Model implementation [4 pts.]

Implement the described neural network architecture described at the beginning of this section. You will find the *tf.nn.conv2d*, *tf.nn.relu*, *tf.nn.batch_normalization* and *tf.nn.max_pool* utility functions useful. For the convolutional layer, initialize each filter with the Xaiver scheme. Initialize your weight and biases for the other layers like you did in Part 1 (but with Tensorflow tensors).

For the *padding* parameter, set it to the 'SAME' method. For the batch normalization layer, the *tf.nn.moments* function will be useful for obtaining the mean and variance. **You are allowed to use the built-in cross-entropy loss function**. In total, you should have 6 *tf.get_variable* statements - one for the filter, one for the first bias to be added after applying the filter in step 2, weight and bias matrices between steps 5 and 7, and weight and bias matrices for the final two steps. Include your Python code snippets in your report.

2.2 Model Training [4 pts.]

Train your implemented model using SGD for a batch size of 32, for 50 epochs and the Adam optimizer for learning rate of $\alpha = 1 \times 10^{-4}$, making sure to shuffle your training data after each epoch. Your objective function will be to minimize the cross entropy loss. Plot the training, validation and test loss and accuracy curves and include them in your report.

2.3 Hyperparameter Investigation [6 pts.]

1. **L2 Normalization [3 pts.]**: As in the first assignment, implement L2 regularization for the weights in your model and test for weight decay coefficients $\lambda = [0.01, 0.1, 0.5]$ while holding all other parameters constant as in section 2.2. Present in tabular format the final training, validation and test accuracies for each scenario. What is impact of L2 regularization on the final test and validation accuracies, if any?

2. **Dropout [3 pts.]**: Another method to control overfitting in very deep neural networks is to apply dropout to certain layers in the model. Add a dropout layer after step 7, described in section 2 and test it with probabilities $p = [0.9, 0.75, 0.5]$ while holding all other parameters constant as in section 2.2 with no regularization and plot the training, validation and testing accuracies after 50 epochs and include them in your report.