Problem 1. Implementing a Neural Network Classifier

In this homework, you will implement a neural network and the backpropagation algorithm and stochastic gradient descent with mini-batches. The goal is to implement this yourself with only the basic linear algebra package *numpy*, rather than a modern deep learning library. We will use MNIST, a common classification data set consisting of 28x28 pixel grayscale images of handwritten digits (0-9), with 60,000 training examples and 10,000 test examples. The task is to probabilistically predict the digit from the image using multi-class classification.

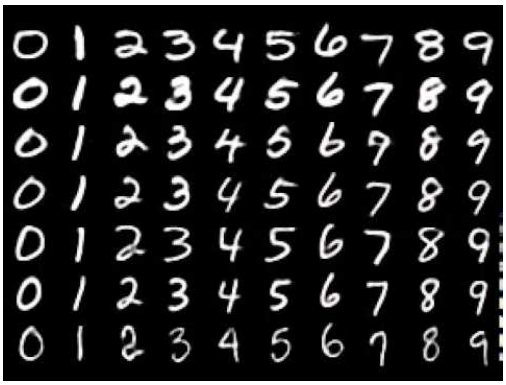

Figure 1: MNIST example images

Requirements:

(a) Your model should use a *softmax* output layer with categorical cross-entropy loss.

$$\text{Loss}_{ce} = \sum_{i}^{C} y_i \log(\hat{y}_i) \tag{1}$$

(b) Your model should have three hidden layers. The layers should be fully connected. The activation/transfer function of the hidden layers should have ReLU, SELU, and hyperbolic tangent functions, respectively.

$$\text{ReLU}(x) = \max(0, x) \tag{2}$$

$$\text{SELU}(x) = \begin{cases} 1.75809 * (e^x - 1) & \text{for} \quad x \leq 0 \\ 1.0507 * x & \text{for} \quad x > 0 \end{cases} \tag{3}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

(c) Use mini-batch gradient descent (e.g. mini-batches of size 100).

(d) Implement momentum, RMSProp, or ADAM optimization algorithm.

(e) Train the network to achieve at least 94% accuracy on the test set. Tune the hyper-parameters (e.g. weight initialization, learning rate) to get good performance on the test data.

(f) Plot both the training and test *loss* (the NLL) throughout training (at every epoch).

(g) Write a paragraph describing the network, training algorithm, and any hyperparameter choices. Make sure to report the *accuracy* (also called the *01-loss*. To get a class prediction from a softmax layer, we simply take the *argmax* of the probabilities.

(h) Plot another figure that demonstrates *overfitting* to the training data, so that the test loss increases while the training loss continues to decline. (Hint: this should be easier with a network of a single hidden layer, so you can change the architecture.)

(i) Submit your homework as a Jupyter notebook to the github classroom.

Tips:

(a) For a minimalist implementation of a neural network, see:
https://iamtrask.github.io/2015/07/12/basic-python-network/

(b) To quickly download the MNIST data set, see:
https://github.com/hsjeong5/MNIST-for-Numpy

(c) A single pass through the training data is called an *epoch*. You may have to do 50-100 epochs to see overfitting. If your computer is slow, go ahead and use a fraction of the data set, e.g. 6,000 examples.

(d) We typically compute the validation loss at the end of every training epoch to see if we are overfitting. For the training data set, we usually keep a running average of the loss while performing the mini-batch training updates to save on computation.