Due date: May 8, 2020, 11:59 PM (Arlington time). You have **two** late days throughout the semester — use it at as you wish. Once you run out of this quota, the penalty for late submission will be applied. You can either use your late days quota (or let the penalty be applied). **Clearly indicate** in your submission if you seek to use the quota.

What to turn in:

1. Your submission should include your complete code base in an archive file (`zip, tar.gz`) and `q1/, q2/,` and so on), and a very very clear README describing how to run it.
2. A brief report (typed up, submit as a PDF file, NO handwritten scanned copies) describing what you solved and implemented and known failure cases. The report is **important** since we will be evaluating the grades mostly based on the report.
3. Submit your entire code and report to Canvas.

Notes from instructor:

- You may ask the TA or instructor for suggestions, and discuss the problem with others (minimally). But **all parts of the submitted code must be your own**.
- Use Python for your implementation.
- Make sure that the TA can easily run the code by plugging in our test data.

Problem 1

(Logistic Regression, **30pts**) Implement logistic regression. For your training data, generate 1000 training instances in two sets of random data points (500 in each) from multi-variate normal distribution with

$$\mu_1 = [1, 0],\ \mu_2 = [0, 1.5],\ \Sigma_1 = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix},\ \Sigma_2 = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix} \tag{1}$$

and label them 0 and 1. Generate testing data in the same manner but include 250 instances for each class, i.e., 500 in total. You will implement a logistic regression from scratch. Use sigmoid function for your activation function and cross entropy for your objective function. Stop your training when the l_1-norm of your gradient is less than 0.001 (or close to 0 based on your own threshold) or the number of iteration reaches 100000. Don't forget to include bias term!

1. (**15pts**) Perform batch training using gradient descent. Divide the derivative with the total number of training dataset as you go through iteration (it is very likely that you will get NaN if you don't do this.). Change your learning rate as $\eta = \{1, 0.1, 0.01, 0.001\}$. Your report should include: 1) scatter plot of the testing data and the trained decision boundary, 2) figure of changes of training loss (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number of iterations it took for training and the accuracy that you have.
2. (**15pts**) Perform online training using gradient descent. Here, you do not need to normalize the gradient with the training dataset size since each iteration takes one training sample at a time. Try various learning rate as $\eta = \{1, 0.1, 0.01, 0.001\}$. Your report should include: 1) scatter plot of the testing data and the trained decision boundary, 2) figure of changes of training loss (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number of iterations it took for training and the accuracy that you have. Write your brief observation comparing this result from the result from batch training.

Problem 2

(Artificial Neural Network, **70pts**) Use **PyTorch** to design your own Neural Network for Image Classification on CIFAR-10 dataset. Convert color images to gray scale and make sure that the pixel intensities are normalized to stay in [0,1]. We are providing a sample code (template.zip) for you to complete. Please read `readme.md` carefully. First, start with only one hidden layer between input and output layers in your design. Use ReLu function as your activation function and cross entropy for your objective function to minimize. Set your (mini) batch size to 32, and number of neurons in the hidden layer to 64. Use soft-max function at the output layer for pseudo probability for multiple classes to predict. Train your model until convergence (i.e., when the gradient is sufficiently small or decrease in objective function is small) or when the number of epoch reaches 100 (max_epoch is there so that your ANN doesn't run forever). The criteria for convergences is up to you.

For each of the following questions, you should include:

- figure of changes of training loss, training accuracy and testing accuracy w.r.t. epoch.

and your observation/comparisons/analysis.

1. (**20pt**) Train your model on training dataset and test the trained model on testing dataset.
2. (**5pt**) Add one more hidden layer (with 64 nodes) to your network and try out your model. What has changed from the previous result by adding one more layer?
3. (**10pt**) Try dropout at the rate of 0.2 for the hidden layers and apply your model on the dataset. What does the dropout do? What has changed compared to the previous training/results?
4. (**10pt**) Now, you may freely change the architecture of the ANN (e.g., number of hidden layers, number of hidden nodes, activation function, dropout rate and other regularizers) to obtain better testing accuracy than the previous results. Try your best.
5. (**5pt**) Let's move on to the fashion MNIST dataset. The data loader should be given in the template. Apply your ANN from P2.4 to see if it can correctly classify different classes.
6. (**10pt**) Now, remove all the hidden layers and train the model. The trained model contains the weights directly from input to output nodes that it has learned from training. Plot the "new representation" that it has learned for each output node. That is, reshape the learned weights (i.e., vector) to the image dimension (in 2D, i.e., 28x28) and show them as an image. Can you see any interesting shapes?
7. (**10pt**) Select any two classes from the dataset and apply your ANN and logistic regression model from P1. Report ROC curve (you may use built-in functions) along with other results. Compare the training process / results between your ANN and Logistic regression.