**Due date: May 1, 2018, 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 Blackboard.

**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, **60pts**) 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 500 instances for each class, i.e., 1000 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 (i.e., close to 0) or the number of iteration reaches 100000. Don't forget to include bias term!

1. (**30pt**) 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 error (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number iterations it took for training and the accuracy that you have.
2. (**30pt**) 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 $\eta = \{1, 0.1, 0.01, 0.001\}$. Set your maximum number of iterations to 100000. Your report should include: 1) scatter plot of the testing data and the trained decision boundary, 2) figure of changes of training error (cross entropy) w.r.t. iteration, 3) figure of changes of norm of gradient w.r.t. iteration. Also, report the number 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

(Tensorflow and Keras, **40pts**) Try out the tutorial for Deep Learning using Tensorflow at `https://www.tensorflow.org/tutorials`. It has the following lines of code.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation=tf.nn.relu),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

1. (**10pt**) In the report, write comments for each line of code given above and explain what this framework is doing.
2. (**10pt**) Change the number of hidden nodes to 5, 10, 128 and 512. Report how the testing accuracy changes for the testing data. Report the result and your observation in the report.
3. (**20pt**) Now, remove the hidden layer in the code and train the model. The trained model contains the weights that it has learned from training. Plot the "new representation" that it has learned for each number from training and include them in the report. That is, reshape the learned weights (i.e., vector) to the image dimension (in 2D, i.e., 28x28) and show them. You will see some number-like features.