Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

ECE 521 Assignment 2

Work Breakdown

Group Member Name	Contribution Percentage
Jixong Deng	33%
Jeffrey Kirman	33%
Connor Smith	33%

Part 1: Linear regression

Linear regression is performed using the following minibatch stochastic gradient descent (SGD) algorithm:

```
Init: w = 0                          # Weights vector
      x_j = [1 x_j1 x_j2 ... x_jd]^T # Single in feature
      X = [x_1 ... x_n]              # Matrix of all in-features
      y = [y_1 ... y_n]              # Vector of all targets
Step: for each epoch
         Shuffle X -> X'             # Randomly shuffle the vectors in X into X'
         for each mini-batch: X' = [x'(mini-batch_size)*i ... x'(mini-batch_size)*(i+1)]
            g_t = ∇E_in(w^(t))       # Calculate the gradient of the loss
            w^(t+1) = w^(t) - ηg_t   # Update the weights, where η is the learning factor
Result: return w
```

The **SGD** function (**linear_regression2.py** in *Appendix A*) implements this. This function creates a graph in which with placeholders for input matrix **X** and target vector **y**. It then calculates the MSE loss according to the following equation:

$$\mathcal{L} = \sum_{n=1}^{N} \frac{1}{2N} \left\| \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{n}} - \boldsymbol{y}_{\boldsymbol{n}} \right\|_{\mathbf{2}}^{\mathbf{2}} + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2$$

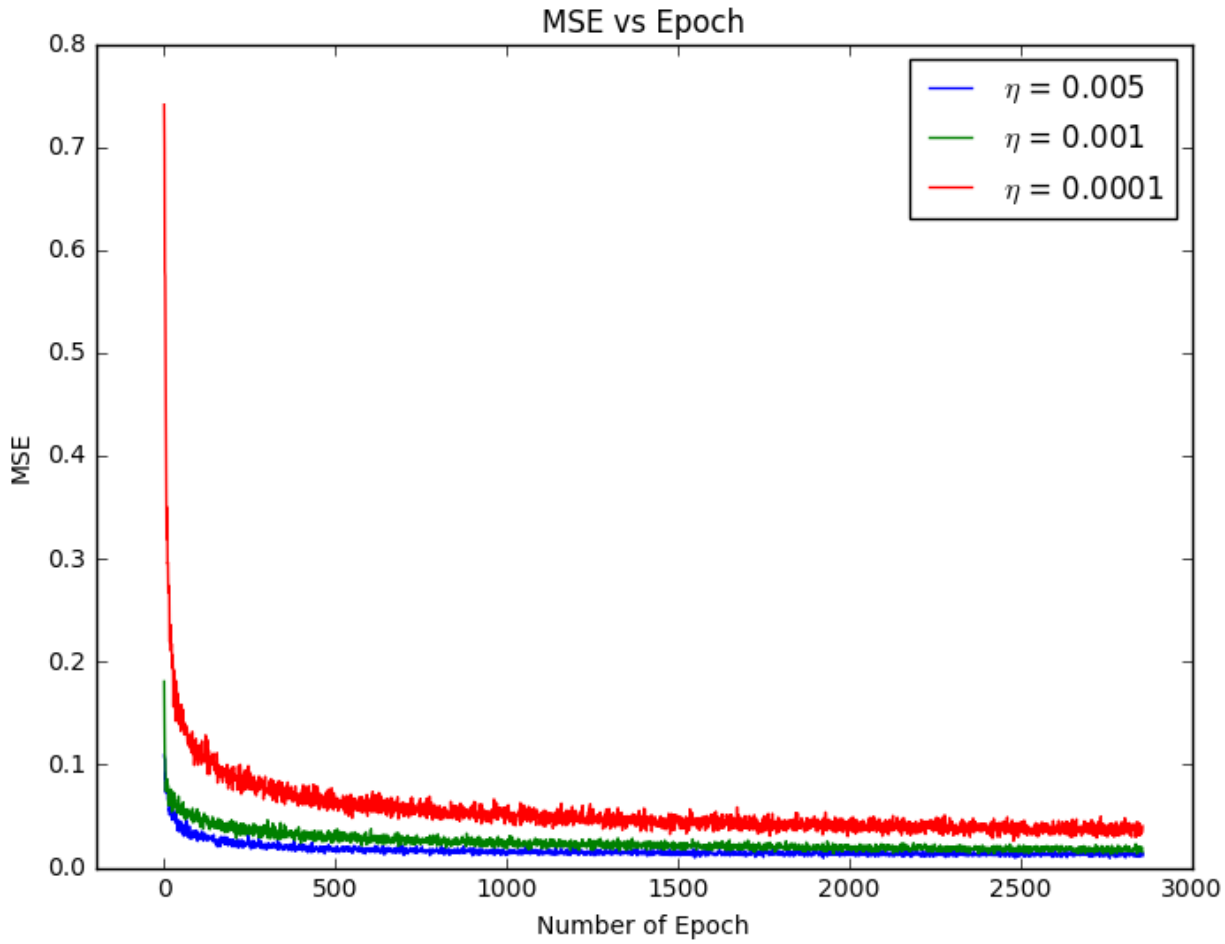
Where λ is the decay constant. The gradients are then calculated using a function in the tensorflow API, tf.gradients. Finally, the weights are updated.

After the graph is created the function calculates the number of epochs based on the iterations and mini-batch size. It then shuffles an array of indices and then uses them to index the matrix of in-features into a mini-batch. feed_dict is then used to feed the inputs into the placeholders and run the graph for one iteration.

Part 1 – 1

The results of changing the learning rate (η) can be seen in the figure below. Here we can see that as the learning rate is increased the quicker it converges and the lower final value it converges to. Therefore, **the best η in this case is 0.005.**

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)



Part 1 – 2: Effect of the mini-batch size

The final values of the MSE loss as well as computation time are summarized in the table below:

Mini-batch Size	MSE loss	Computation time (seconds)
500	0.0149	206
1500	0.0176	433
3500	0.0171	1243

The best mini-batch size in terms of training time is 500. This makes sense since each iteration contains less operations (i.e. less MSE loss and less gradients to compute), hence each update will happen faster.

Note: for mini-batches that do not equally divide the number of training examples, the remainder of unused examples are used in a shortened mini-batch at the end of each epoch.

Part 1 – 3: Generalization

The final values of the accuracy on the validation and test set, as well as computation time are summarized in the table below. When calculating accuracy, if a prediction was above 0.5 it was said to be classified as 1, and below 0.5, classified as 0.

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

λ	Validation Accuracy	Test Accuracy	Computation time (seconds)
0	0.959	0.99	209
0.001	**0.966**	**0.98**	**210**
0.1	0.952	0.95	211
1	0.938	0.94	211

The best decay coefficient in terms of validation accuracy is λ = 0.001. Here the validation accuracy is 0.966 and the test accuracy is 0.98. Using weight decay can help the model by regularizing the data as to not overfit the test set. It is useful in this sense because it gives a tunable parameter to change how the model predicts values outside of the training set.

Part 1 – 4: Comparing SGD with the normal equation

The normal equation (excluding the decay coefficient) is

$$(X^T X)\boldsymbol{w} = X\boldsymbol{y}$$

Using this we can calculate a prediction for the weights. The results of this can be see in the following table.

	MSE loss	Validation Accuracy	Test Accuracy	Computation Time (seconds)
SGD	0.0121	0.966	1.0	234
Normal Equation	0.0094	0.958	0.97	0.0213

Here we see that the normal equation performs better in terms of lower MSE but SGD performs in accuracy. In terms of computation time the SGD takes much longer than using the normal equation. That being said, SGD would be the better method to use with exceptionally large datasets since you could lower the batch size and retain high accuracy, whereas calculating matrix multiplications and inverses for the normal equation would become more and more computationally expensive.

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Part 2: Logistic Regression

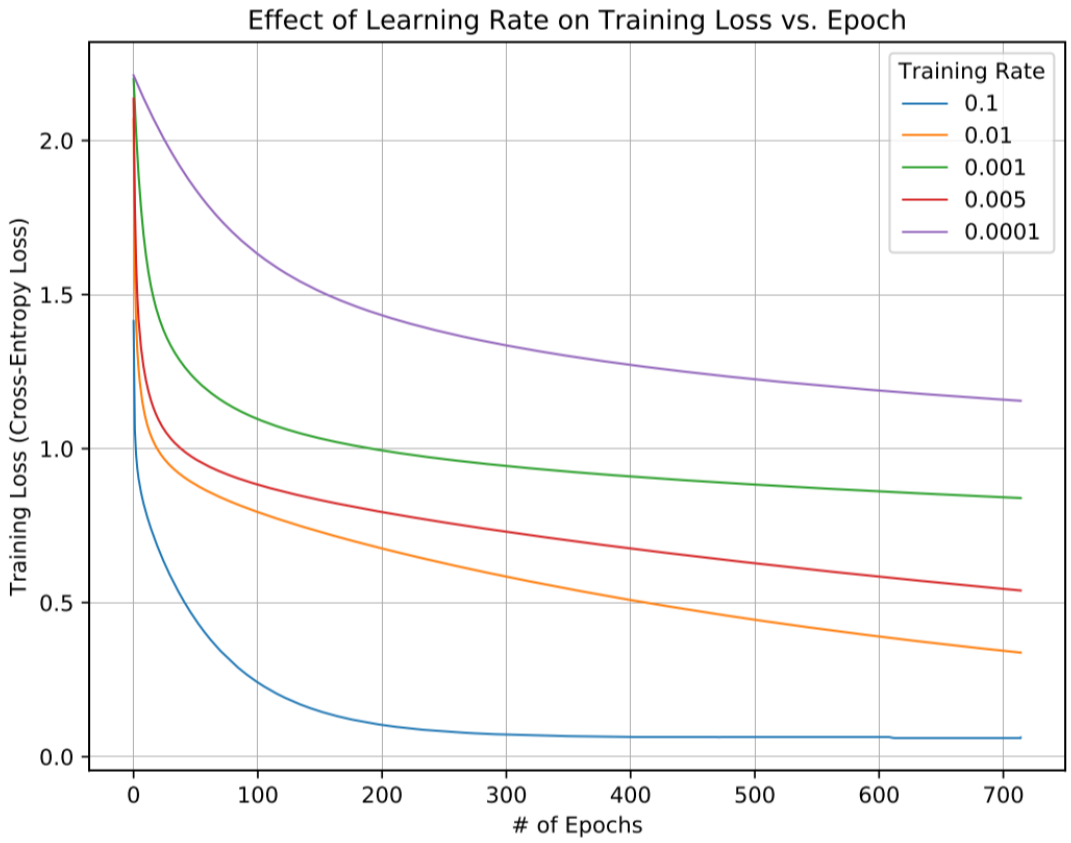
The Mean-Squared-Error loss function above is suitable for classic regression tasks, however it can be overly sensitive to mislabelled examples and outliers in the training data for a classification task. As a result, we find that the cross-entropy loss function described below can result in better model performance.

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\mathcal{D}} + \mathcal{L}_{\mathcal{W}} \\ &= \sum_{n=1}^{N} \frac{1}{N}\left[-y^{(n)} \log \hat{y}\left(x^{(n)}\right) - \left(1 - y^{(n)}\right) \log\left(1 - y\left(x^{(n)}\right)\right)\right] + \frac{\lambda}{2}\|W\|_2^2\end{aligned}$$

Where $\hat{y}(x) = \sigma(W^T x + b)$ and $\sigma(z) = \frac{1}{1+e^{-z}}$, with all other constants defined as they were in Part 1 above. Code used to plot the following graphs can be found in *Appendix A: logistic_regression.py*

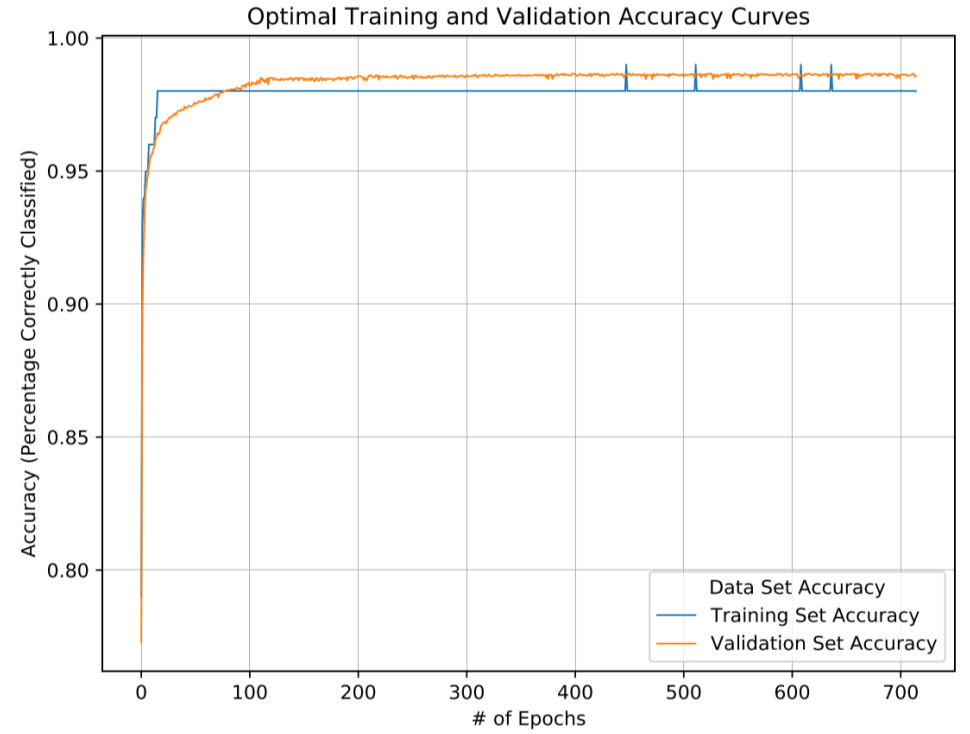
Part 2.1 – 1: Learning

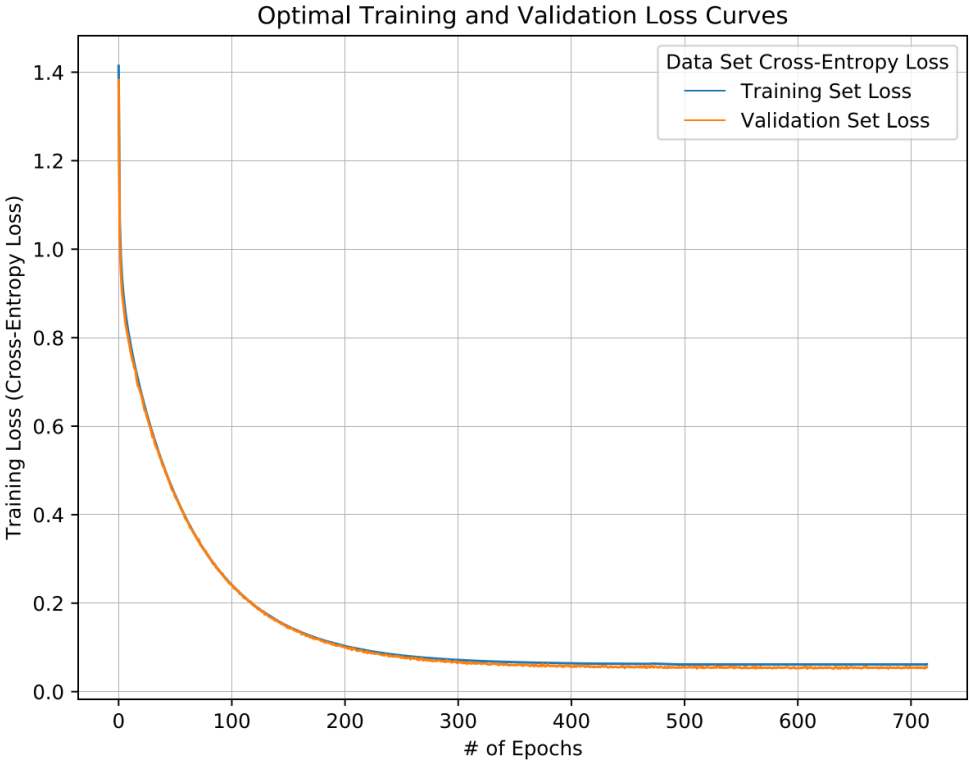
Utilizing a decay coefficient $\lambda = 0.01$ and a mini-batch size $B = 500$ with 5000 iterations, the following training curves were observed for different values of learning rates η:



Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Selecting the optimal learning rate value of $\eta = 0.1$, the following training and validation curves for cross-entropy loss and classification accuracy were plotted:



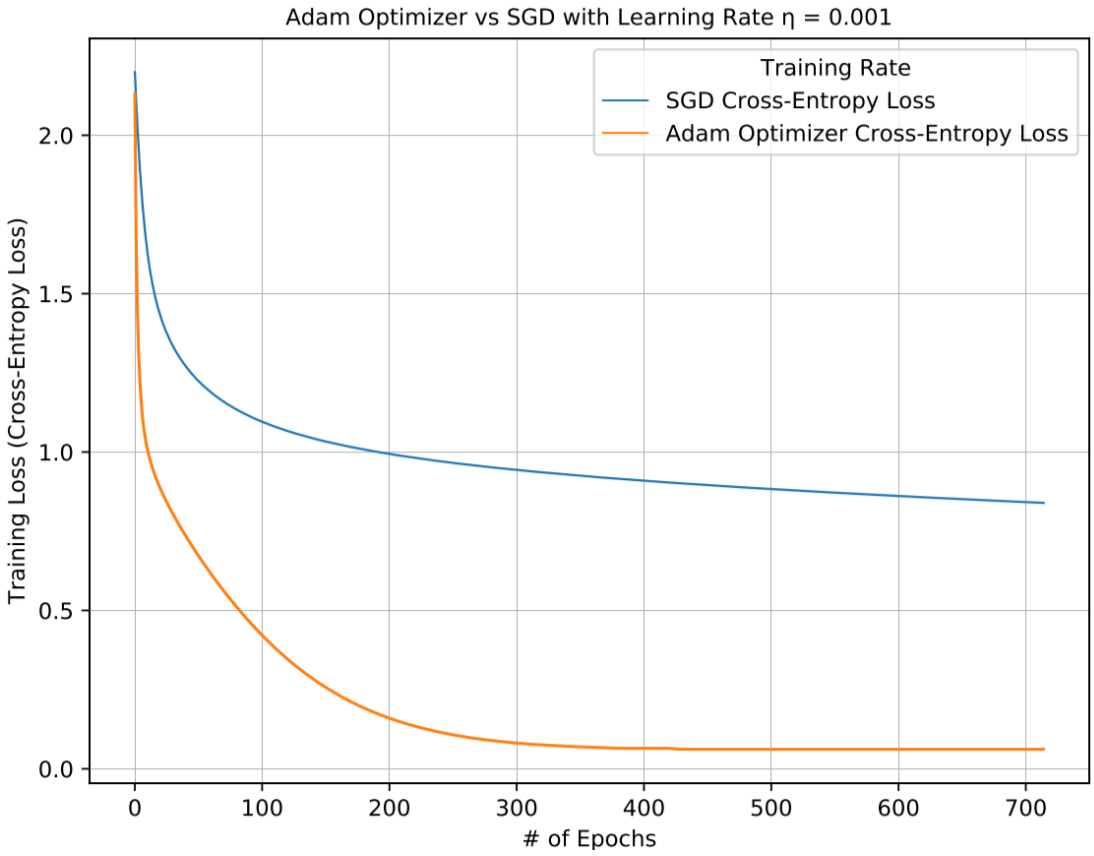


From these curves, we observed a maximum test classification accuracy of 97.93%.

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Part 2.1 – 2: Beyond Plain SGD

Utilizing a decay coefficient $\lambda = 0.01$, learning rate $\eta = 0.001$ and a mini-batch size $B = 500$ with 5000 iterations and the Adam Optimizer implementation provided by TensorFlow, we found the model to converge much faster than using the regular *GradientDescentOptimizer* used in the above sections. A plot comparing this convergence can be found below.



The Adam Optimizer can help to train on the notMNIST dataset with much fewer epochs while retaining the same or better level of loss and thereby predicted accuracy. This results in a faster model training time.

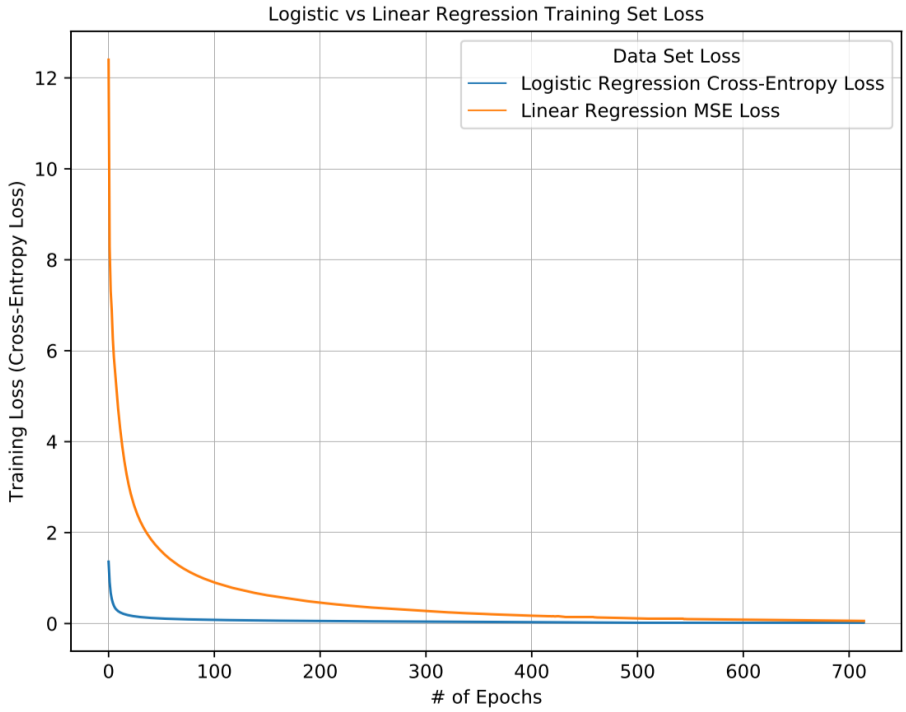
Part 2.1 – 3: Comparison with linear regression

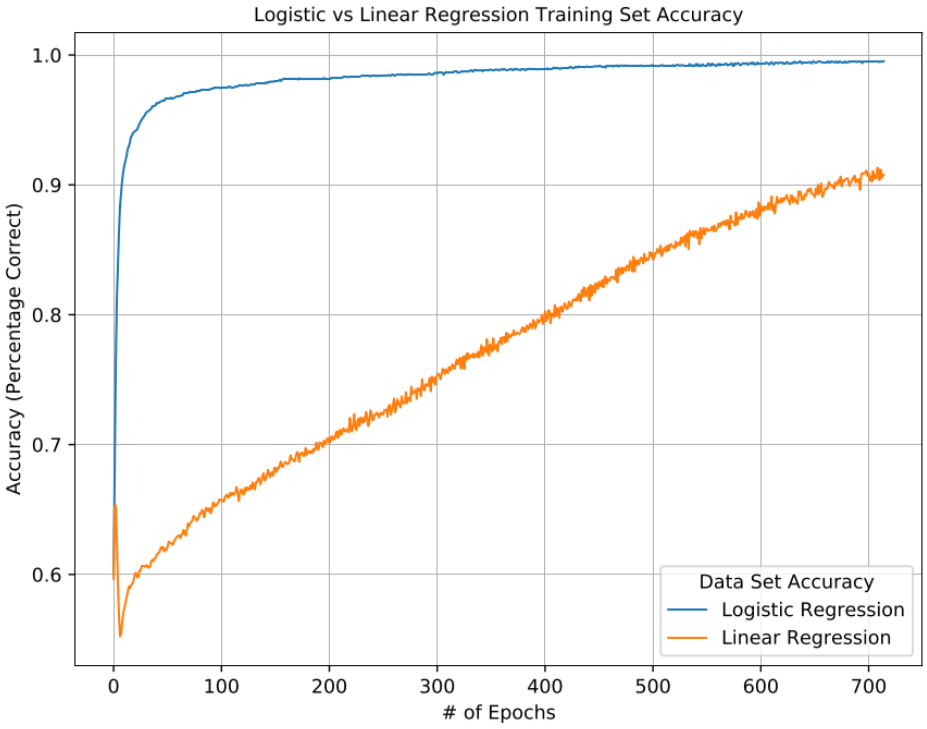
Setting $\lambda = 0$, the optimal logistic regression learning rate was $\eta = 0.1$. Comparing the train, validation and test data set classification accuracy of this regression with the optimal linear regression classifier found using the "normal equation", the following results were achieved:

	Train Accuracy	Validation Accuracy	Test Accuracy
Adam Optimizer	0.99514	0.98	0.9793
Normal Equation	0.993	0.959	0.97

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Furthermore, comparing the optimal logistic regression classifier with a learned linear regression classifier produced the following accuracy and loss curves for the training set:



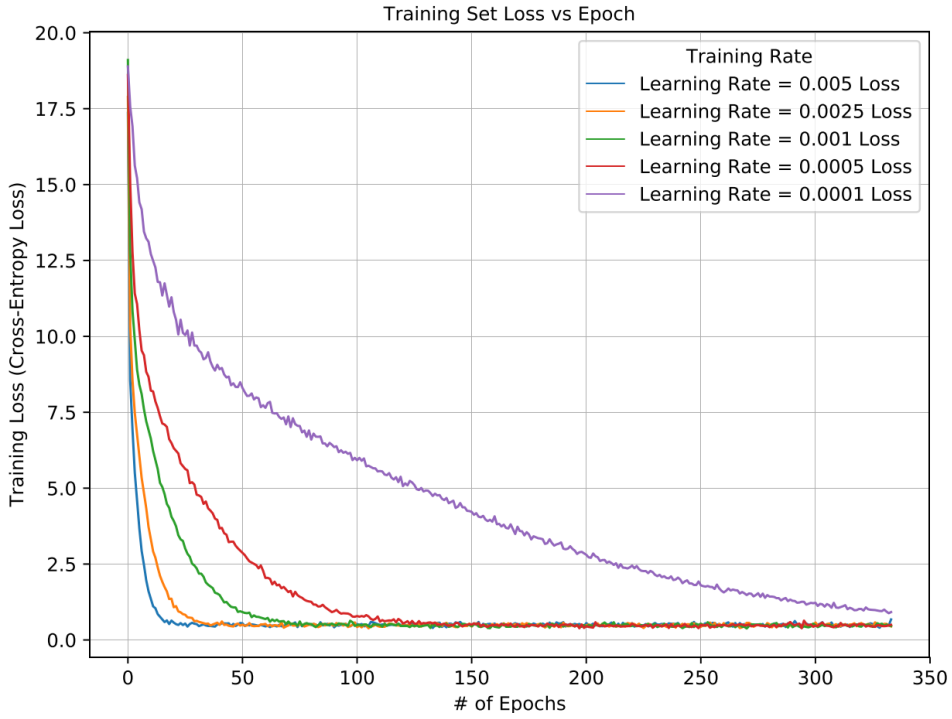


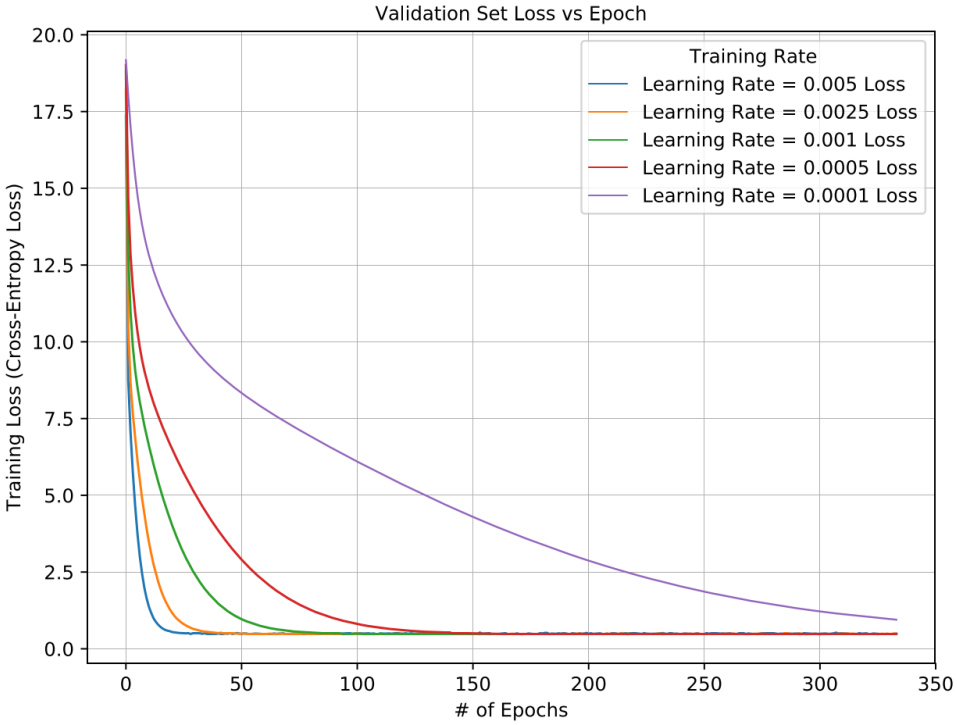
From these graphs, it is clear that Logistic regression converges much faster while also providing much greater accuracy over 730 epochs.

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

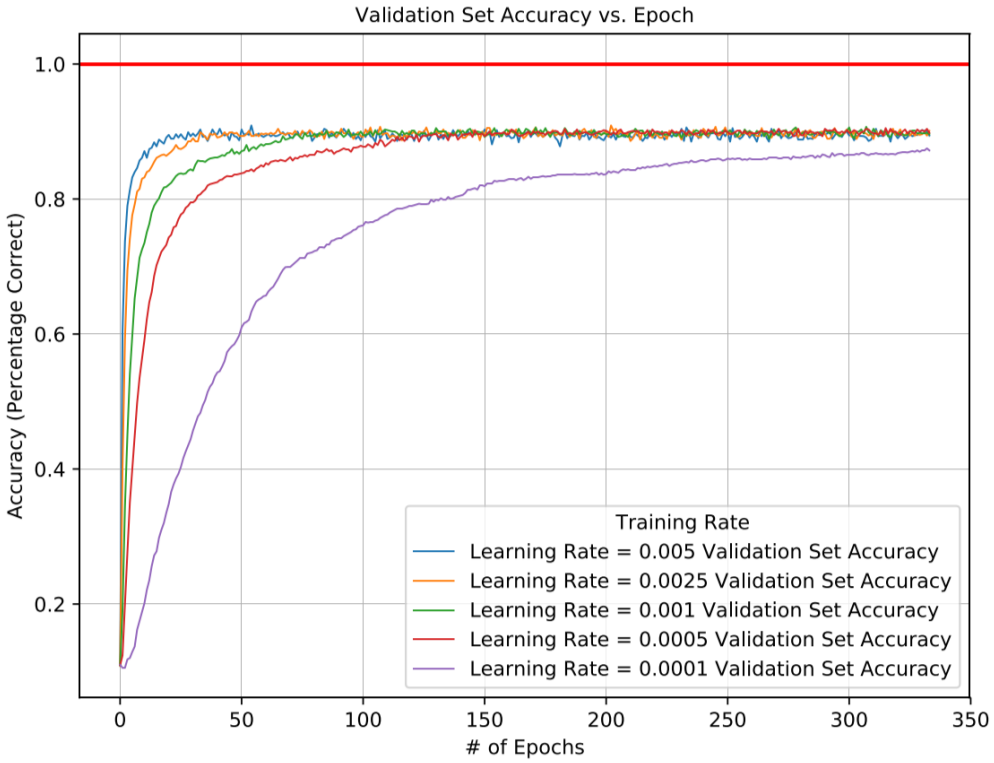
Part 2.2 – 1: Multi-class Classification using Softmax Cross-Entropy on notMNIST

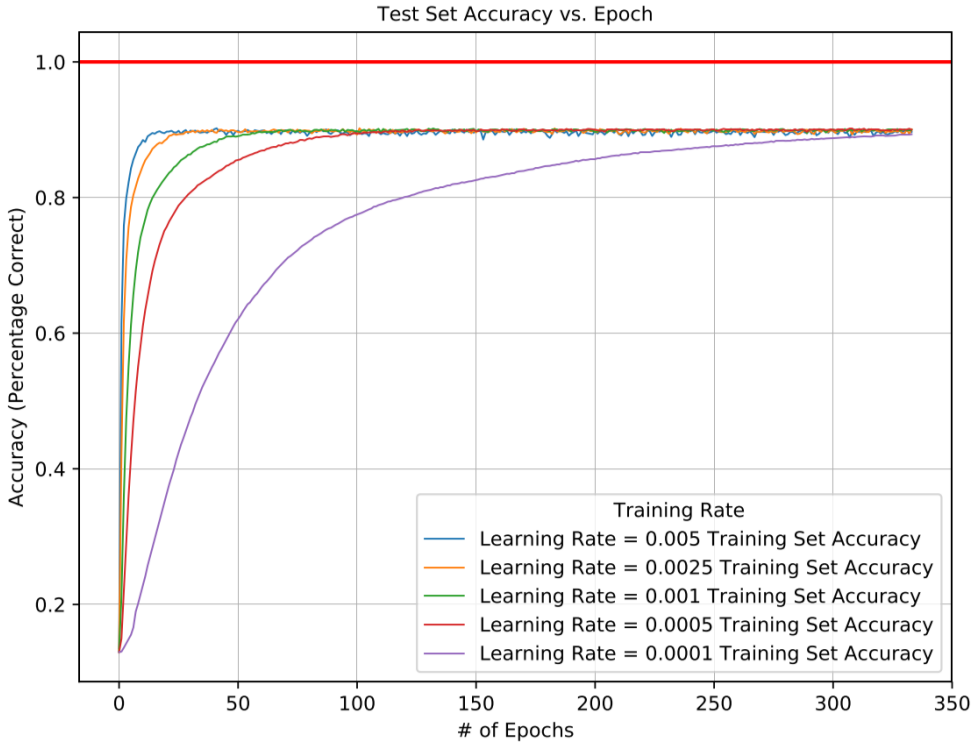
Utilizing similar code as in Part 2.1 with $\lambda = 0.01$ and $B = 500$, but substituting the loss function for the softmax cross-entropy function, the following training and validation curves for loss and accuracy were found:





Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)



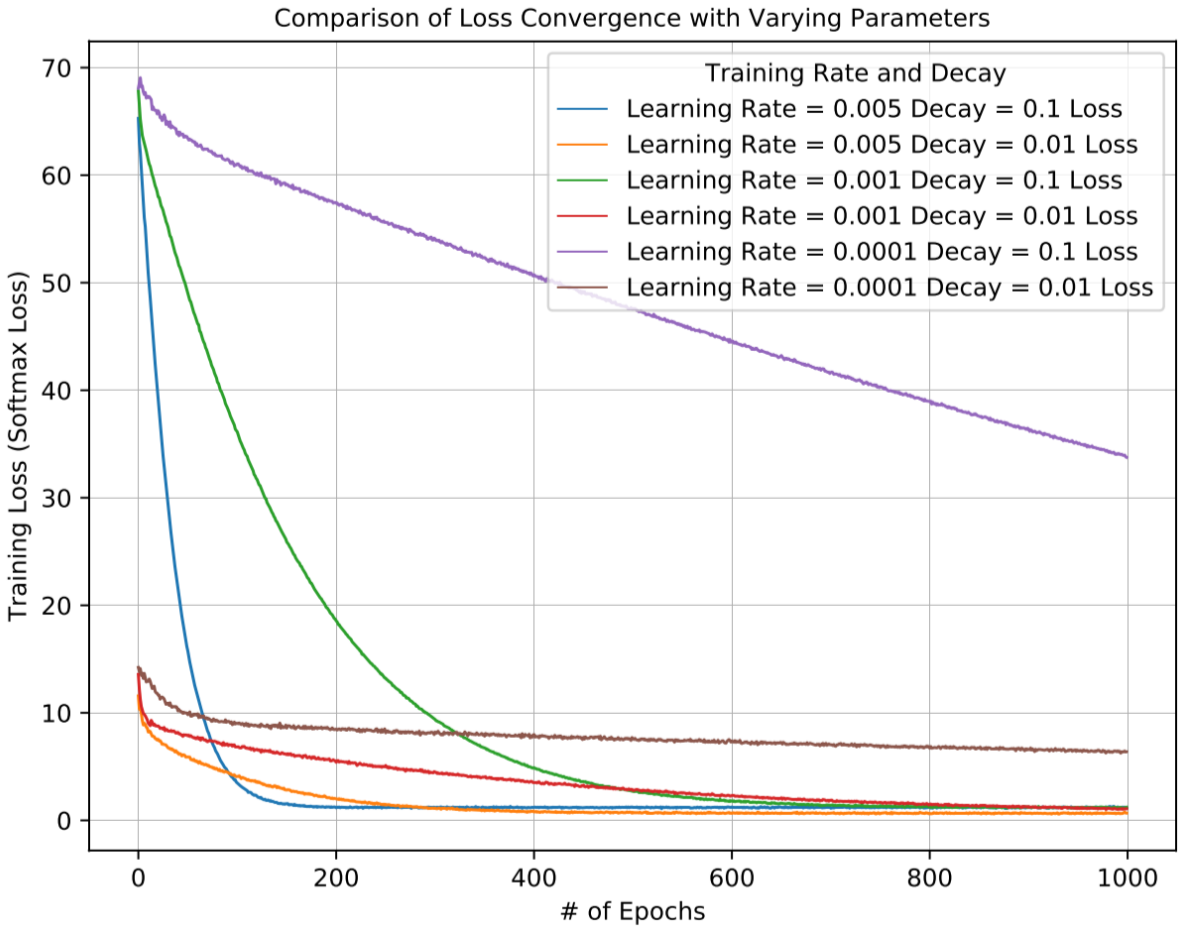


Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

This lead to us selection $\eta = 0.005$ as the optimal learning rate for this classifier. Using this learning rate, the best test classification accuracy was measured as 0.8961 or 89.61% correct. This is approximately 10% worse than in Part 2.1. Intuitively, this makes sense as there are far more classes than in the previous parts, making the model much more uncertain with its classifications.

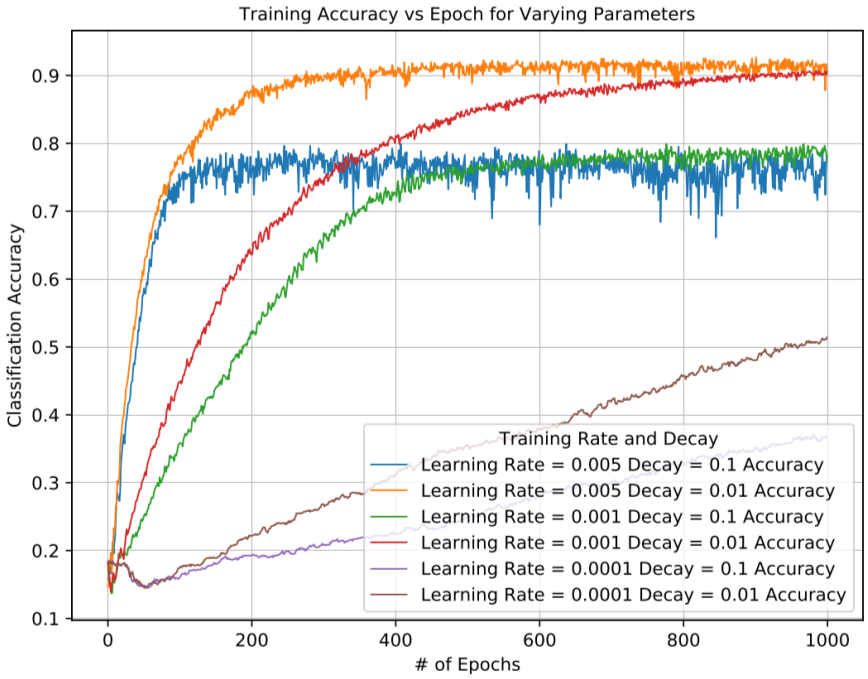
Part 2.2 – 2: Multi-class Classification using Softmax Cross-Entropy on FaceScrub

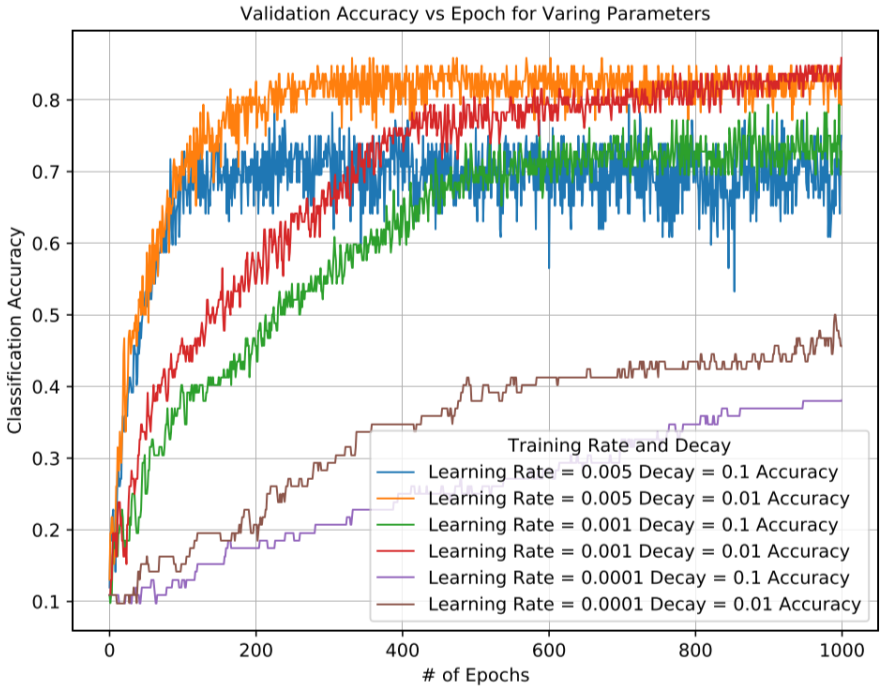
Repeating the above part for the FaceScrub dataset with a batch size $B = 300$ the following graphs were used to tune both the decay coefficient λ and learning rate η:



Note that the above graph only covers a small sample of the parameters tested. For the full list, please refer to *Appendix A – logistic_regression_multi.py.* The training and validation accuracy and loss curves for these parameters is:

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)





This resulted in optimal parameters $\eta = 0.005$ and $\lambda = 0.01$, with the best test classification accuracy being 89.25%. The best results achieved with the k-NN classifier in Assignment 1 for this dataset was 71.0%, meaning this logistic classifier was over 18 percentage points more accurate.

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

Appendix A – Python code

linear_regression2.py

```
import _pickle
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import time

def load_data():
    with np.load("./notMNIST.npz") as data:
        Data, Target = data["images"], data["labels"]
        posClass = 2
        negClass = 9
        dataIndx = (Target==posClass) + (Target==negClass)
        Data = Data[dataIndx]/255
        Target = Target[dataIndx].reshape(-1,1)
        Target[Target==posClass] = 1
        Target[Target==negClass] = 0
        np.random.seed(521)
        randIndx = np.arange(len(Data))
        np.random.shuffle(randIndx)
        Data, Target = Data[randIndx], Target[randIndx]

        trainData, trainTarget = Data[:3500], Target[:3500]
        validData, validTarget = Data[3500:3600], Target[3500:3600]
        testData, testTarget = Data[3600:], Target[3600:]
        return (trainData, trainTarget, validData, validTarget, testData, testTarget)


def SGD(xTrain, yTrain, batchSize, iters, learning_rate, decay_coefficient, use_normal_eqn = False):
    """
    Do stochastic gradient descent
    :param xTrain: numpy array of shape (number of samples, width, height)
    :param yTrain: np array of shape (number of samples, 1)
    :param batchSize: integer
    :param iters: integer
    :param learning_rate: float
    :param decay_coefficient: float
    :param use_normal_eqn: bool
    :return: losses for each epoch, list, each element is np.array of shape (1)
    """
    X = tf.placeholder(tf.float64, shape=(None, xTrain.shape[1]*xTrain.shape[2]), name='X')
    Y = tf.placeholder(tf.float64, shape=(None, 1), name='Y')
    with tf.variable_scope('weight', reuse=tf.AUTO_REUSE):
        weights = tf.get_variable('weights', shape=[X.shape[1] + 1, 1], dtype=tf.float64)   # (W*H +1)
x 1
    x_in = tf.pad(X, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)        # bn x (W*H +1)
    z = tf.tensordot(x_in, weights, axes=1)
    dc = tf.placeholder(tf.float64, name='dc')
    mse_loss = tf.reduce_mean(tf.square(Y-z))/(2)
    decay_loss = dc*tf.sqrt(tf.reduce_sum(tf.square(weights)))/2
    total_loss = mse_loss + decay_loss
    full_mse_loss = tf.reduce_mean(tf.square(Y-tf.tensordot(x_in, weights, axes=1)))/2
    full_total_loss = full_mse_loss + decay_loss
    if use_normal_eqn:
        grads = [tf.reshape(tf.reduce_mean(-tf.transpose((Y - z) * x_in) + tf.expand_dims(dc,axis = 0)
* weights, axis=-1), (-1, 1)),]
    else:
        grads = tf.gradients(total_loss, weights)
    updates = tf.assign_sub(weights, learning_rate * grads[0])
    num_sample = xTrain.shape[0]
    num_epochs = (batchSize*iters + 1)//num_sample + 1
    curr_iter = 0
    losses = []
    with tf.Session() as sess:
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
        sess.run(tf.global_variables_initializer())
        for i in range(num_epochs):
            inds = np.arange(num_sample)
            np.random.shuffle(inds)
            this_epoch_loss = 0.0
            curr_samples = 0
            for j in range((num_sample+1)//batchSize + 1):
                l = j*batchSize
                u = min(xTrain.shape[0], l+batchSize)
                if l == u:
                    continue
                if curr_iter >= iters:
                    break
                curr_iter += 1
                curr_samples += (u - l)
                this_inds = inds[l:u]
                this_batch_X = xTrain[this_inds]
                this_batch_Y = yTrain[this_inds]
                this_batch_X = np.reshape(this_batch_X, (this_batch_X.shape[0],
this_batch_X.shape[1]*this_batch_X.shape[2]))
                this_loss, gr, wts, full_loss = sess.run([total_loss, grads, weights, full_total_loss],
                                        feed_dict={X: this_batch_X, Y:this_batch_Y.astype(np.float64),
                                                   dc:np.array([decay_coefficient], dtype=np.float64)})
                this_epoch_loss += this_loss*(u-l)
                sess.run(updates,
                         feed_dict={X: this_batch_X, Y:this_batch_Y.astype(np.float64),
                                    dc:np.array([decay_coefficient], dtype=np.float64)})
            if curr_samples == 0:
                continue
            this_epoch_loss /= curr_samples
            losses.append(full_loss)
    return [losses, wts]


def tuning_the_learning_rate():

    iters = 20000
    batch_size = 500
    learning_rates = [0.005, 0.001, 0.0001]
    decay_coefficient = 0.

    (trainData, trainTarget,
     testData, testTarget,
     validData, validTarget) = load_data()

    plt.figure(figsize=(8, 6), dpi=80)
    plt.subplot(1, 1, 1)

    for i in range(0,len(learning_rates)):
        loss, _ = SGD(trainData, trainTarget, batch_size, iters, learning_rates[i], decay_coefficient)
        x = np.arange(len(loss))
        plt.plot(x, np.array(loss),label = r'$\eta$' + " = " + str(learning_rates[i]))

    plt.xlim(-200, 3000)
    plt.xlabel('Number of Epoch')
    plt.ylabel('MSE')
    plt.title("MSE vs Epoch")
    plt.legend()
    plt.show()

    plt.savefig("Tuning the Learning Rate.pdf", format="pdf")

    return plt


def effect_of_minibatch_size():
    iters = 20000
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
    batch_sizes = [500, 1500, 3500]
    learning_rate = 0.005
    decay_coefficient = 0
    losses = []

    (trainData, trainTarget,
     testData, testTarget,
     validData, validTarget) = load_data()

    for i in range(0,len(batch_sizes)):
        time_start = time.clock()
        loss = SGD(trainData, trainTarget, batch_sizes[i], iters, learning_rate, decay_coefficient)[0]
        x = np.arange(len(loss))
        plt.plot(x, np.array(loss), label='batch_size = %f' % batch_sizes[i])
        losses.append(loss)
        time_elapsed = (time.clock() - time_start)
        print("Time passed for B = " + str(batch_sizes[i]) + ": " + str(time_elapsed))

    plt.xlabel('Number of Epoch')
    plt.ylabel('MSE')
    plt.title("MSE vs Epoch")
    plt.legend()
    plt.show()
    return losses


def generalization():
    iters = 20000
    batch_size = 500
    learning_rate = 0.005
    decay_coefficients = [0.0, 0.001, 0.1, 1]

    (trainData, trainTarget,
     testData, testTarget,
     validData, validTarget) = load_data()
    final_weights = []
    validation_accuracy = []
    test_accuracy = [];

    for i in range(0,len(decay_coefficients)):
        time_start = time.clock()
        (loss, wt) = SGD(trainData, trainTarget, batch_size, iters, learning_rate,
decay_coefficients[i])
        time_elapsed = (time.clock() - time_start)
        print("Time passed for dc = " + str(decay_coefficients[i]) + ": " + str(time_elapsed))
        x = np.arange(len(loss))
        plt.plot(x, np.array(loss), label='decay_coefficient = %f' % decay_coefficients[i])
        final_weights.append(wt)

        valid_linear = np.reshape(validData, (validData.shape[0],
validData.shape[1]*validData.shape[2]))
        x_in_valid = tf.pad(valid_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
        valid_y_pred = tf.matmul(x_in_valid, wt)
        same_valid = tf.equal(tf.greater(valid_y_pred, tf.constant(0.5, tf.float64)),
tf.constant(validTarget, tf.bool))
        v_accuracy = tf.count_nonzero(same_valid) / tf.constant(validTarget).shape[0]
        with tf.Session() as sess:
            validation_accuracy.append(sess.run(v_accuracy))

        test_linear = np.reshape(testData, (testData.shape[0], testData.shape[1]*testData.shape[2]))
        x_in_test = tf.pad(test_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
        test_y_pred = tf.matmul(x_in_test, wt)
        same_test = tf.equal(tf.greater(test_y_pred, tf.constant(0.5, tf.float64)),
tf.constant(testTarget, tf.bool))
        t_accuracy = tf.count_nonzero(same_test) / tf.constant(testTarget).shape[0]
        with tf.Session() as sess:
            test_accuracy.append(sess.run(t_accuracy))
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
    plt.xlabel('Number of Epoch')
    plt.ylabel('MSE')
    plt.title("MSE vs Epoch")
    plt.legend()
    plt.show()

    return (validation_accuracy, test_accuracy)

def sgd_vs_normal_equation():
    iters = 20000
    batch_size = 500
    learning_rate = 0.005
    decay_coefficient = 0

    (trainData, trainTarget,
     testData, testTarget,
     validData, validTarget) = load_data()
    validation_accuracy = []
    test_accuracy = []

    time_start = time.clock()
    (loss, wt) = SGD(trainData, trainTarget, batch_size, iters, learning_rate, decay_coefficient)
    time_elapsed = (time.clock() - time_start)
    print("Time passed for SGD = : " + str(time_elapsed))

    valid_linear = np.reshape(validData, (validData.shape[0], validData.shape[1]*validData.shape[2]))
    x_in_valid = tf.pad(valid_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
    valid_y_pred = tf.matmul(x_in_valid, wt)
    same_valid = tf.equal(tf.greater(valid_y_pred, tf.constant(0.5, tf.float64)),
tf.constant(validTarget, tf.bool))
    v_accuracy = tf.count_nonzero(same_valid) / tf.constant(validTarget).shape[0]
    with tf.Session() as sess:
        validation_accuracy.append(sess.run(v_accuracy))

    test_linear = np.reshape(testData, (testData.shape[0], testData.shape[1]*testData.shape[2]))
    x_in_test = tf.pad(test_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
    test_y_pred = tf.matmul(x_in_test, wt)
    same_test = tf.equal(tf.greater(test_y_pred, tf.constant(0.5, tf.float64)), tf.constant(testTarget,
tf.bool))
    t_accuracy = tf.count_nonzero(same_test) / tf.constant(testTarget).shape[0]
    with tf.Session() as sess:
        test_accuracy.append(sess.run(t_accuracy))

    time_start = time.clock()
    train_linear = np.reshape(trainData, (trainData.shape[0], trainData.shape[1]*trainData.shape[2]))
    x_in_train = tf.pad(train_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
    y_in_train = tf.constant(trainTarget, tf.float64)
    inv = tf.matrix_inverse(tf.matmul(x_in_train,x_in_train,True))
    wt = tf.matmul(inv, tf.matmul(x_in_train,y_in_train, True))
    time_elapsed = (time.clock() - time_start)
    print("Time passed for normal = : " + str(time_elapsed))

    z = tf.matmul(wt, x_in_train, True, True)
    n_loss = tf.reduce_mean(tf.square(y_in_train-tf.transpose(z)))/(2)
    with tf.Session() as sess:
        normal_loss = sess.run(n_loss)

    train_linear = np.reshape(trainData, (trainData.shape[0], trainData.shape[1]*trainData.shape[2]))
    x_in_train = tf.pad(train_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
    train_y_pred = tf.matmul(x_in_train, wt)
    same_train = tf.equal(tf.greater(train_y_pred, tf.constant(0.5, tf.float64)),
tf.constant(trainTarget, tf.bool))
    tr_accuracy = tf.count_nonzero(same_train) / tf.constant(trainTarget).shape[0]
    with tf.Session() as sess:
        print(sess.run(tr_accuracy))
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
    valid_linear = np.reshape(validData, (validData.shape[0], validData.shape[1]*validData.shape[2]))
    x_in_valid = tf.pad(valid_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
    valid_y_pred = tf.matmul(x_in_valid, wt)
    same_valid = tf.equal(tf.greater(valid_y_pred, tf.constant(0.5, tf.float64)),
tf.constant(validTarget, tf.bool))
    v_accuracy = tf.count_nonzero(same_valid) / tf.constant(validTarget).shape[0]
    with tf.Session() as sess:
        validation_accuracy.append(sess.run(v_accuracy))

    test_linear = np.reshape(testData, (testData.shape[0], testData.shape[1]*testData.shape[2]))
    x_in_test = tf.pad(test_linear, [[0, 0], [0, 1]], "CONSTANT", constant_values=1)
    test_y_pred = tf.matmul(x_in_test, wt)
    same_test = tf.equal(tf.greater(test_y_pred, tf.constant(0.5, tf.float64)), tf.constant(testTarget,
tf.bool))
    t_accuracy = tf.count_nonzero(same_test) / tf.constant(testTarget).shape[0]
    with tf.Session() as sess:
        test_accuracy.append(sess.run(t_accuracy))

    x = np.arange(len(normal_loss))
    plt.plot(x, np.array(loss), label='default_gradient')
    plt.plot(x, np.array(normal_loss), label='normal_gradient')
    plt.xlabel('Number of Epoch')
    plt.ylabel('MSE')
    plt.title("MSE vs Epoch")
    plt.legend()
    plt.show()


    return (loss, normal_loss, validation_accuracy, test_accuracy)

if __name__ == '__main__':
    tuning_the_learning_rate()
    losses = effect_of_minibatch_size()
    (valid, test) = generalization()
    (l, nl, valid2, test2) = sgd_vs_normal_equation()
```

logistic_regression.py

```
import tensorflow as tf
import numpy as np
import util

def binary_cross_entropy(Q=1):
    xTrain, yTrain, xValid, yValid, xTest, yTest = util.load_data()

    with tf.Graph().as_default():
        decay = 0
        B = 500
        learning_rates = [0.005, 0.001, 0.0001]
        iters = 5000
        num_iters_per_epoch = len(xTrain)//B # number of iterations we have to do for one epoch
        print("Num epochs = ",iters/num_iters_per_epoch)

        # optimized parameters
        w = tf.Variable(tf.truncated_normal(shape=[784,1], stddev=0.5, seed=521), dtype=tf.float32,
name="weight-vector")
        b = tf.Variable(tf.zeros([1]), dtype=tf.float32, name="bias-term")

        # hyperparameters
        learning_rate = tf.placeholder(dtype=tf.float32, name="learning-rate")

        # Get Data
        xTrainTensor = tf.constant(xTrain, dtype=tf.float32, name="X-Training")
        yTrainTensor = tf.constant(yTrain, dtype=tf.float32, name="Y-Training")
        xTestTensor = tf.constant(xTest, dtype=tf.float32, name="X-Test")
        yTestTensor = tf.constant(yTest, dtype=tf.float32, name="Y-Test")
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
        xValidTensor = tf.constant(xValid, dtype=tf.float32, name="X-Validation")
        yValidTensor = tf.constant(yValid, dtype=tf.float32, name="Y-Validation")

        Xslice, yslice = tf.train.slice_input_producer([xTrainTensor, yTrainTensor], num_epochs=None)

        Xbatch, ybatch = tf.train.batch([Xslice, yslice], batch_size = B)

        # setting up batch loss function
        y_pred = tf.matmul(Xbatch, w) + b
        logLoss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_pred, labels=ybatch))
+ decay * tf.nn.l2_loss(w)

        # setting up epoch loss function
        train_logLoss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.matmul(xTrainTensor, w)+b,
labels=yTrainTensor)) + decay * tf.nn.l2_loss(w)
        valid_logLoss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=tf.matmul(xValidTensor, w)+b,
labels=yValidTensor)) + decay * tf.nn.l2_loss(w)

        # accuracy function
        train_y_pred = tf.round(tf.sigmoid(tf.matmul(xTrainTensor, w) + b))
        valid_y_pred = tf.round(tf.sigmoid(tf.matmul(xValidTensor, w) + b))
        test_y_pred = tf.round(tf.sigmoid(tf.matmul(xTestTensor, w) + b))
        train_accuracy = tf.count_nonzero(tf.equal(train_y_pred, yTrainTensor)) / yTrainTensor.shape[0]
        test_accuracy = tf.count_nonzero(tf.equal(test_y_pred, yTestTensor)) / yTestTensor.shape[0]
        valid_accuracy = tf.count_nonzero(tf.equal(valid_y_pred, yValidTensor)) / yValidTensor.shape[0]

        # optimizer function
        gradientOptimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(logLoss)

        if Q==2: # Part 2.1 Q2
            adamOptimizer = tf.train.AdamOptimizer(learning_rate).minimize(logLoss)
            for optimizer, name in [(gradientOptimizer, "Gradient Descent"), (adamOptimizer, "Adam
Optimizer")]:
                loss_amounts = []
                valid_accuracies = []
                train_accuracies = []
                with tf.Session() as sess:
                    coord = tf.train.Coordinator()
                    threads = tf.train.start_queue_runners(sess=sess, coord=coord)
                    sess.run(tf.global_variables_initializer())
                    sess.run(tf.local_variables_initializer())
                    print("Running", name)
                    for i in range(iters):
                        sess.run([optimizer], feed_dict={learning_rate: 0.001})
                        if (i % num_iters_per_epoch == 0):
                            loss_amount, train_acc, valid_acc = sess.run([train_logLoss,
train_accuracy, valid_accuracy])
                            loss_amounts.append(loss_amount)
                            valid_accuracies.append(valid_acc)
                            train_accuracies.append(train_acc)
                            print("Epoch: {}, Loss: {}".format(i//num_iters_per_epoch, loss_amount))
                    coord.request_stop()
                    coord.join(threads)
                    np.save("{}_loss".format(name),loss_amounts)
                    np.save("{}_v_acc".format(name), valid_accuracies)
                    np.save("{}_t_acc".format(name), train_accuracies)
        elif Q==1: # Part 2.1 Q1
            for r in learning_rates:
                loss_amounts = []
                v_loss_amounts = []
                valid_accuracies = []
                train_accuracies = []
                test_accuracies = []
                print("Running learning rate", r)
                with tf.Session() as sess:
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
                    sess.run(tf.global_variables_initializer())
                    sess.run(tf.local_variables_initializer())

                    coord = tf.train.Coordinator()
                    threads = tf.train.start_queue_runners(sess=sess, coord=coord)
                    for i in range(iters):
                        # run one iteration of the optimizer
                        sess.run([gradientOptimizer], feed_dict={learning_rate: r})
                        # calculate our loss for this iteration
                        if (i % num_iters_per_epoch == 0):
                            loss_amount, v_loss_amount, train_acc, valid_acc, test_acc =
sess.run([train_logLoss, valid_logLoss, train_accuracy, valid_accuracy, test_accuracy])
                            print("Epoch {}, loss = {}".format(i//num_iters_per_epoch, loss_amount))
                            print("\t Train Acc = {}, Valid Acc = {}".format(train_acc, valid_acc))
                            loss_amounts.append(loss_amount)
                            v_loss_amounts.append(v_loss_amount)
                            valid_accuracies.append(valid_acc)
                            train_accuracies.append(train_acc)
                            test_accuracies.append(test_acc)

                    coord.request_stop()
                    coord.join(threads)
                    np.save("log_q1_"+str(r)+"_loss.npy", loss_amounts)
                    np.save("log_q1_"+str(r)+"_v_loss.npy", v_loss_amounts)
                    np.save("log_q1_"+str(r)+"_valid_acc.npy", valid_accuracies)
                    np.save("log_q1_"+str(r)+"_train_acc.npy", train_accuracies)
                    np.save("log_q1_"+str(r)+"_test_acc.npy", test_accuracies)
        elif Q==3: # Part 2.1 Q3
            loss_amounts = []
            valid_accuracies = []
            train_accuracies = []
            test_accuracies = []
            logOptimizer = tf.train.AdamOptimizer(learning_rate).minimize(logLoss)
            with tf.Session() as sess:
                sess.run(tf.global_variables_initializer())
                sess.run(tf.local_variables_initializer())

                coord = tf.train.Coordinator()
                threads = tf.train.start_queue_runners(sess=sess, coord=coord)
                for i in range(iters):
                    # run one iteration of the optimizer
                    sess.run([logOptimizer], feed_dict={learning_rate: 0.001})
                    # calculate our loss for this iteration
                    if (i % num_iters_per_epoch == 0):
                        loss_amount, train_acc, valid_acc, test_acc = sess.run([train_logLoss,
train_accuracy, valid_accuracy, test_accuracy])
                        print("Epoch {}, loss = {}".format(i//num_iters_per_epoch, loss_amount))
                        print("\t Train Acc = {}, Valid Acc = {}".format(train_acc, valid_acc))
                        loss_amounts.append(loss_amount)
                        valid_accuracies.append(valid_acc)
                        train_accuracies.append(train_acc)
                        test_accuracies.append(test_acc)

                coord.request_stop()
                coord.join(threads)
                np.save("2.1.3_Log_loss",loss_amounts)
                np.save("2.1.3_Log_valid_acc", valid_accuracies)
                np.save("2.1.3_Log_train_acc", train_accuracies)
                np.save("2.1.3_Log_test_acc", test_accuracies)

            loss_amounts = []
            valid_accuracies = []
            train_accuracies = []
            lin_t_y_pred = tf.minimum(tf.maximum(tf.round(tf.matmul(xTrainTensor, w) + b), 0), 1)
            lin_v_y_pred = tf.minimum(tf.maximum(tf.round(tf.matmul(xValidTensor, w) + b), 0), 1)
            linearLoss = tf.reduce_mean(tf.square(y_pred - ybatch))/2
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
            linearLoss_epoch = tf.reduce_mean(tf.square(tf.matmul(xTrainTensor, w) + b -
yTrainTensor))/2
            lin_accuracy = tf.count_nonzero(tf.equal(tf.minimum(tf.maximum(tf.round(y_pred), 0), 1),
ybatch)) / yTrainTensor.shape[0]
            lin_t_accuracy = tf.count_nonzero(tf.equal(lin_t_y_pred, yTrainTensor)) /
yTrainTensor.shape[0]
            lin_v_accuracy = tf.count_nonzero(tf.equal(lin_v_y_pred, yValidTensor)) /
yValidTensor.shape[0]
            linearOptimizer = tf.train.AdamOptimizer(learning_rate).minimize(linearLoss)
            with tf.Session() as sess:
                sess.run(tf.global_variables_initializer())
                sess.run(tf.local_variables_initializer())

                coord = tf.train.Coordinator()
                threads = tf.train.start_queue_runners(sess=sess, coord=coord)
                for i in range(iters):
                    # run one iteration of the optimizer
                    sess.run([linearOptimizer], feed_dict={learning_rate: 0.001})
                    # calculate our loss for this iteration
                    if (i % num_iters_per_epoch == 0):
                        loss_amount, train_acc, valid_acc = sess.run([linearLoss_epoch, lin_t_accuracy,
lin_v_accuracy])
                        print("Epoch {}, loss = {}".format(i//num_iters_per_epoch, loss_amount))
                        print("\t Train Acc = {}, Valid Acc = {}".format(train_acc, valid_acc))
                        loss_amounts.append(loss_amount)
                        valid_accuracies.append(valid_acc)
                        train_accuracies.append(train_acc)
                np.save("2.1.3_Lin_loss",loss_amounts)
                np.save("2.1.3_Lin_v_acc", valid_accuracies)
                np.save("2.1.3_Lin_t_acc", train_accuracies)

                coord.request_stop()
                coord.join(threads)

binary_cross_entropy(3)
```

logistic_regression_multi.py

```
import tensorflow as tf
import numpy as np
import util

def load_notmnist_data():
    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]
        t = np.zeros((trainTarget.shape[0], 10))
        t[np.arange(trainTarget.shape[0]), trainTarget] = 1
        trainTarget = t
        validData, validTarget = Data[15000:16000], Target[15000:16000]
        t = np.zeros((validTarget.shape[0], 10))
        t[np.arange(validTarget.shape[0]), validTarget] = 1
        validTarget = t
        testData, testTarget = Data[16000:], Target[16000:]
        t = np.zeros((testTarget.shape[0], 10))
        t[np.arange(testTarget.shape[0]), testTarget] = 1
        testTarget = t
        return (trainData.reshape(trainData.shape[0], -1), trainTarget,
validData.reshape(validData.shape[0], -1), validTarget, testData.reshape(testData.shape[0], -1),
testTarget)
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
def load_facescrub_data(task=0):
    # task = 0 >> select the name ID targets for face recognition task
    # task = 1 >> select the gender ID targets for gender recognition task
    data = np.load("./facescrub_data.npy")/255.0
    data = np.reshape(data, [-1, 32*32])

    target = np.load("./facescrub_target.npy")

    np.random.seed(45689)
    rnd_idx = np.arange(np.shape(data)[0])
    np.random.shuffle(rnd_idx)

    trBatch = int(0.8*len(rnd_idx))
    validBatch = int(0.1*len(rnd_idx))

    trainData, validData, testData = data[rnd_idx[1:trBatch],:], \
    data[rnd_idx[trBatch+1:trBatch + validBatch],:],\
    data[rnd_idx[trBatch + validBatch+1:-1],:]

    trainTarget, validTarget, testTarget = target[rnd_idx[1:trBatch], task], \
    target[rnd_idx[trBatch+1:trBatch + validBatch], task],\
    target[rnd_idx[trBatch + validBatch + 1:-1], task]

    t = np.zeros((testTarget.shape[0], 6))
    t[np.arange(testTarget.shape[0]), testTarget] = 1
    testTarget = t

    t = np.zeros((validTarget.shape[0], 6))
    t[np.arange(validTarget.shape[0]), validTarget] = 1
    validTarget = t

    t = np.zeros((trainTarget.shape[0], 6))
    t[np.arange(trainTarget.shape[0]), trainTarget] = 1
    trainTarget = t


    return trainData.reshape(trainData.shape[0], -1), trainTarget,
validData.reshape(validData.shape[0], -1), validTarget, testData.reshape(testData.shape[0], -1),
testTarget



def multiclass_not_mnist():
    xTrain, yTrain, xValid, yValid, xTest, yTest = load_notmnist_data()
    with tf.Graph().as_default():
        decay = 0.01
        B = 500
        iters = 10000
        learning_rates = [0.001, 0.005, 0.0025, 0.0005, 0.0001]
        learning_rate = tf.placeholder(dtype=tf.float32, name="learning-rate")


        num_iters_per_epoch = len(xTrain)//B # number of iterations we have to do for one epoch
        print("Num epochs = ",iters/num_iters_per_epoch)

        # optimized parameters
        w = tf.Variable(tf.truncated_normal(shape=[784,10], stddev=0.5, seed=521), dtype=tf.float32,
name="weight-vector")
        b = tf.Variable(tf.zeros([1]), dtype=tf.float32, name="bias-term")

        # input tensors
        xTrainTensor = tf.constant(xTrain, dtype=tf.float32, name="X-Training")
        yTrainTensor = tf.constant(yTrain, dtype=tf.float32, name="Y-Training")
        xValidTensor = tf.constant(xValid, dtype=tf.float32, name="X-Validation")
        yValidTensor = tf.constant(yValid, dtype=tf.float32, name="Y-Validation")
        xTestTensor = tf.constant(xTest, dtype=tf.float32, name="X-Test")
        yTestTensor = tf.constant(yTest, dtype=tf.float32, name="Y-Test")
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
        # Create randomly shuffled batches
        Xslice, yslice = tf.train.slice_input_producer([xTrainTensor, yTrainTensor], num_epochs=None)

        Xbatch, ybatch = tf.train.batch([Xslice, yslice], batch_size = B)

        # setting up batch loss function
        y_pred = tf.matmul(Xbatch, w) + b
        y_pred_tr = tf.matmul(xTrainTensor, w) + b
        y_pred_v = tf.matmul(xValidTensor, w) + b
        y_pred_te = tf.matmul(xTestTensor, w) + b
        softmaxLoss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred,
labels=ybatch)) + decay * tf.nn.l2_loss(w)
        softmaxLoss_train = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred_tr,
labels=yTrainTensor)) + decay * tf.nn.l2_loss(w)
        softmaxLoss_valid = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred_v,
labels=yValidTensor)) + decay * tf.nn.l2_loss(w)
        softmaxLoss_test = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred_te,
labels=yTestTensor)) + decay * tf.nn.l2_loss(w)

        # accuracy function TODO these are also probably wrong
        train_y_pred = tf.sigmoid(tf.matmul(xTrainTensor, w) + b)
        valid_y_pred = tf.sigmoid(tf.matmul(xValidTensor, w) + b)
        test_y_pred = tf.sigmoid(tf.matmul(xTestTensor, w) + b)
        train_accuracy = tf.count_nonzero(tf.equal(tf.argmax(train_y_pred, 1), tf.argmax(yTrainTensor,
1))) / yTrainTensor.shape[0]
        valid_accuracy = tf.count_nonzero(tf.equal(tf.argmax(valid_y_pred, 1), tf.argmax(yValidTensor,
1))) / yValidTensor.shape[0]
        test_accuracy = tf.count_nonzero(tf.equal(tf.argmax(test_y_pred, 1), tf.argmax(yTestTensor,
1))) / yTestTensor.shape[0]

        # optimizer function
        optimizer = tf.train.AdamOptimizer(learning_rate).minimize(softmaxLoss)

        for r in learning_rates:
            print("Running learning rate", r)
            loss_amounts = []
            valid_accuracies = []
            train_accuracies = []
            test_accuracies = []
            train_losses = []
            valid_losses = []
            with tf.Session() as sess:
                coord = tf.train.Coordinator()
                threads = tf.train.start_queue_runners(sess=sess, coord=coord)
                sess.run(tf.global_variables_initializer())
                sess.run(tf.local_variables_initializer())
                for i in range(iters):
                    sess.run([optimizer], feed_dict={learning_rate: r})
                    if (i % num_iters_per_epoch == 0):
                        loss_amount, loss_t, loss_v, train_acc, valid_acc, test_acc =
sess.run([softmaxLoss, softmaxLoss_train, softmaxLoss_valid, train_accuracy, valid_accuracy,
test_accuracy])
                        loss_amounts.append(loss_amount)
                        valid_accuracies.append(valid_acc)
                        test_accuracies.append(test_acc)
                        train_accuracies.append(train_acc)
                        train_losses.append(loss_t)
                        valid_losses.append(loss_v)
                        print("Epoch: {}, Loss: {}, trainAcc: {}".format(i//num_iters_per_epoch,
loss_amount, train_acc))
                coord.request_stop()
                coord.join(threads)
                np.save("2.2.1_{}_notmnist_loss".format(r), loss_amounts)
                np.save("2.2.1_{}_notmnist_v_acc".format(r), valid_accuracies)
                np.save("2.2.1_{}_notmnist_train_acc".format(r), train_accuracies)
                np.save("2.2.1_{}_notmnist_test_acc".format(r), test_accuracies)
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
                np.save("2.2.1_{}_notmnist_t_loss".format(r), train_losses)
                np.save("2.2.1_{}_notmnist_v_loss".format(r), valid_losses)


def multiclass_facescrub():
    xTrain, yTrain, xValid, yValid, xTest, yTest = load_facescrub_data()
    with tf.Graph().as_default():
        B = 300
        iters = 2000
        learning_rates = [0.01, 0.005, 0.001, 0.0001]
        decay_rates = [0.1, 0.01, 0.001, 0]
        learning_rate = tf.placeholder(dtype=tf.float32, name="learning-rate")
        decay = tf.placeholder(dtype=tf.float32, name="decay")


        num_iters_per_epoch = len(xTrain)//B # number of iterations we have to do for one epoch
        print("Num epochs = ",iters/num_iters_per_epoch)

        # optimized parameters
        w = tf.Variable(tf.truncated_normal(shape=[1024,6], stddev=0.5, seed=521), dtype=tf.float32,
name="weight-vector")
        b = tf.Variable(tf.zeros([1]), dtype=tf.float32, name="bias-term")

        # input tensors
        xTrainTensor = tf.constant(xTrain, dtype=tf.float32, name="X-Training")
        yTrainTensor = tf.constant(yTrain, dtype=tf.float32, name="Y-Training")
        xValidTensor = tf.constant(xValid, dtype=tf.float32, name="X-Validation")
        yValidTensor = tf.constant(yValid, dtype=tf.float32, name="Y-Validation")
        xTestTensor = tf.constant(xTest, dtype=tf.float32, name="X-Test")
        yTestTensor = tf.constant(yTest, dtype=tf.float32, name="Y-Test")

        # Create randomly shuffled batches
        Xslice, yslice = tf.train.slice_input_producer([xTrainTensor, yTrainTensor], num_epochs=None)

        Xbatch, ybatch = tf.train.batch([Xslice, yslice], batch_size = B)

        # setting up batch loss function
        y_pred = tf.matmul(Xbatch, w) + b
        y_pred_tr = tf.matmul(xTrainTensor, w) + b
        y_pred_v = tf.matmul(xValidTensor, w) + b
        y_pred_te = tf.matmul(xTestTensor, w) + b
        softmaxLoss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred,
labels=ybatch)) + decay * tf.nn.l2_loss(w)
        softmaxLoss_train = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred_tr,
labels=yTrainTensor)) + decay * tf.nn.l2_loss(w)
        softmaxLoss_valid = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred_v,
labels=yValidTensor)) + decay * tf.nn.l2_loss(w)
        softmaxLoss_test = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_pred_te,
labels=yTestTensor)) + decay * tf.nn.l2_loss(w)

        # accuracy function TODO these are also probably wrong
        train_y_pred = tf.sigmoid(tf.matmul(xTrainTensor, w) + b)
        valid_y_pred = tf.sigmoid(tf.matmul(xValidTensor, w) + b)
        test_y_pred = tf.sigmoid(tf.matmul(xTestTensor, w) + b)
        train_accuracy = tf.count_nonzero(tf.equal(tf.argmax(train_y_pred, 1), tf.argmax(yTrainTensor,
1))) / yTrainTensor.shape[0]
        valid_accuracy = tf.count_nonzero(tf.equal(tf.argmax(valid_y_pred, 1), tf.argmax(yValidTensor,
1))) / yValidTensor.shape[0]
        test_accuracy = tf.count_nonzero(tf.equal(tf.argmax(test_y_pred, 1), tf.argmax(yTestTensor,
1))) / yTestTensor.shape[0]

        # optimizer function
        optimizer = tf.train.AdamOptimizer(learning_rate).minimize(softmaxLoss)

        for r in learning_rates:
            for d in decay_rates:
                print("Running learning rate {} with decay {}".format(r, d))
```

Jixong Deng (1001304741)
Jeffrey Kirman (1004387424)
Connor Smith (1000421411)

```
                loss_amounts = []
                valid_accuracies = []
                train_accuracies = []
                test_accuracies = []
                train_losses = []
                valid_losses = []
                with tf.Session() as sess:
                    coord = tf.train.Coordinator()
                    threads = tf.train.start_queue_runners(sess=sess, coord=coord)
                    sess.run(tf.global_variables_initializer())
                    sess.run(tf.local_variables_initializer())
                    for i in range(iters):
                        sess.run([optimizer], feed_dict={learning_rate: r, decay: d})
                        if (i % num_iters_per_epoch == 0):
                            loss_amount, loss_t, loss_v, train_acc, valid_acc, test_acc =
sess.run([softmaxLoss, softmaxLoss_train, softmaxLoss_valid, train_accuracy, valid_accuracy,
test_accuracy], feed_dict={decay: d})
                            loss_amounts.append(loss_amount)
                            valid_accuracies.append(valid_acc)
                            test_accuracies.append(test_acc)
                            train_accuracies.append(train_acc)
                            train_losses.append(loss_t)
                            valid_losses.append(loss_v)
                            print("Epoch: {}, Loss: {}, trainAcc: {}".format(i//num_iters_per_epoch,
loss_amount, train_acc))
                    coord.request_stop()
                    coord.join(threads)
                    np.save("2.2.2_r{}_d{}_facescrub_loss".format(r, d), loss_amounts)
                    np.save("2.2.2_r{}_d{}_facescrub_v_acc".format(r, d), valid_accuracies)
                    np.save("2.2.2_r{}_d{}_facescrub_train_acc".format(r, d), train_accuracies)
                    np.save("2.2.2_r{}_d{}_facescrub_test_acc".format(r, d), test_accuracies)
                    np.save("2.2.2_r{}_d{}_facescrub_t_loss".format(r, d), train_losses)
                    np.save("2.2.2_r{}_d{}_facescrub_v_loss".format(r, d), valid_losses)


multiclass_facescrub()
```