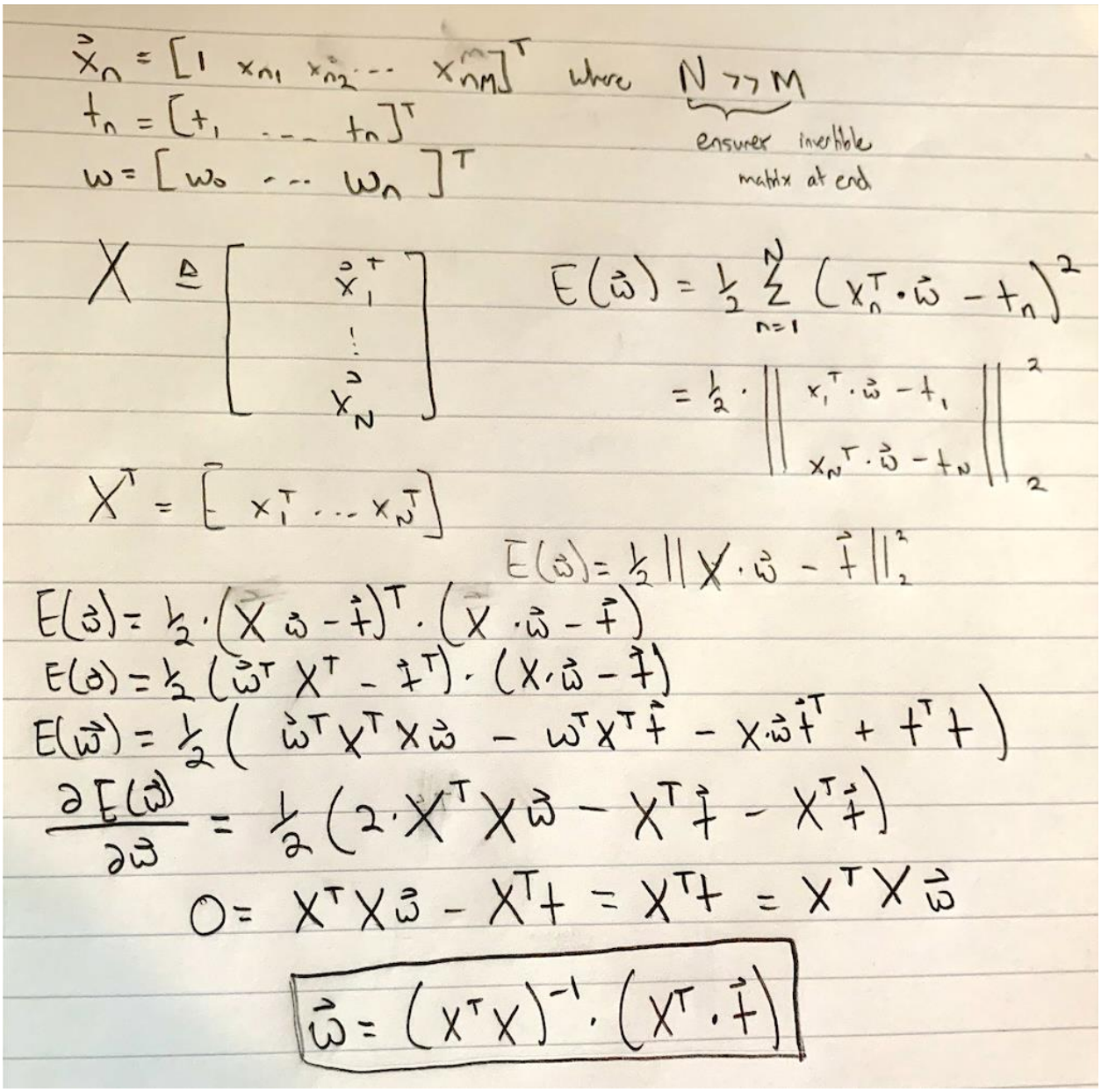
COEN 240 Machine Learning

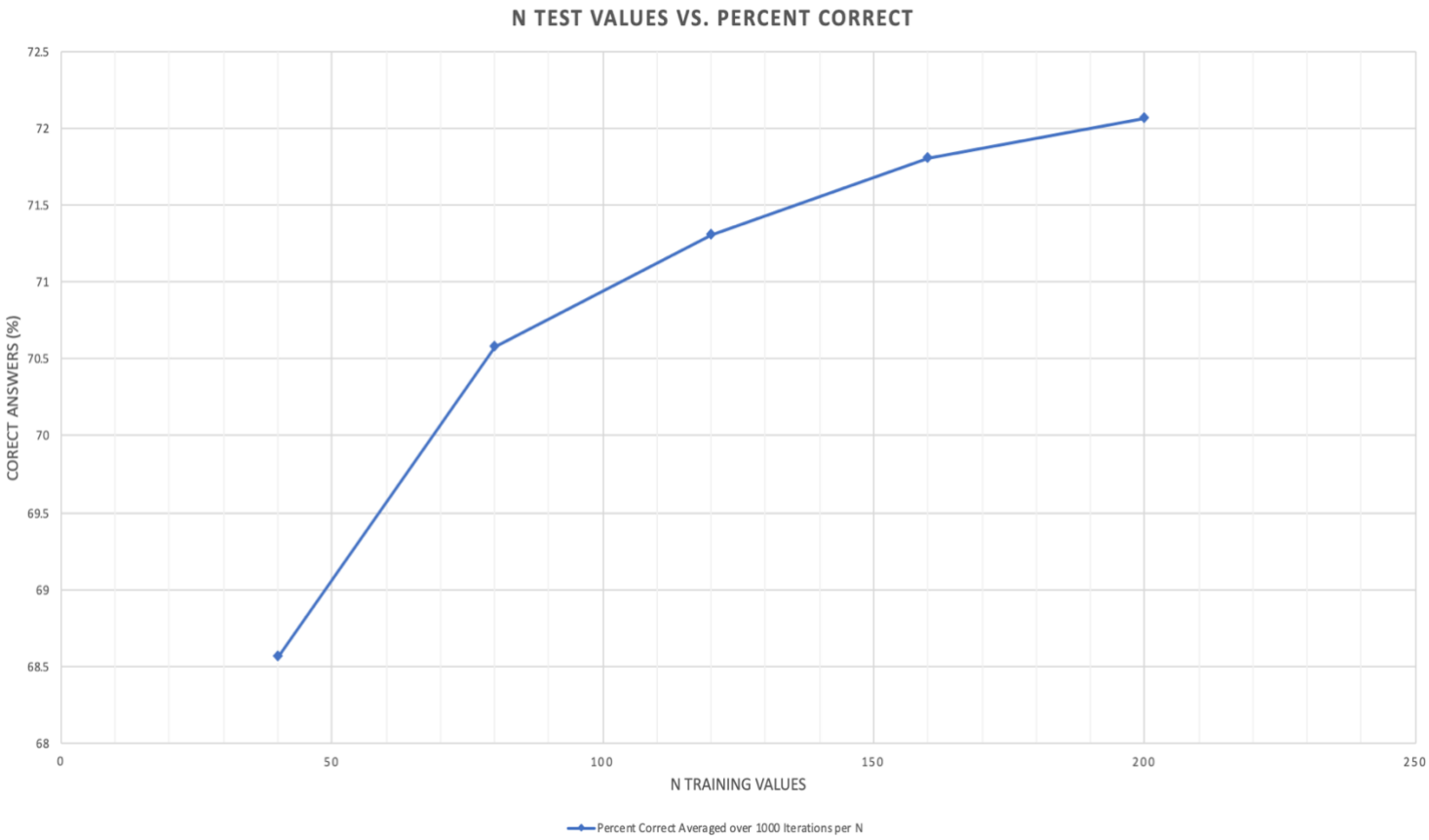
Homework #1

Name: Alex Cherekdjian **Student ID:** 00001083236

Problem 1



Problem 2



Comment: Through plotting all these points and averaging all 1000 iterations of each n, we can see that as the number of training values increases, the accuracy rating of the algorithm goes up slightly with a maximum increase from n=40 to n=200 of around four percent.

Results:
n=40, 68.566
n=80, 70.583
n=120, 71.311
n=160, 71.807
n=200, 72.069

Attachment

Problem 2 Code (in zip file):

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

import pandas as pd
```

```
# Print iterations progress
def printProgressBar (iteration, total, prefix = '', suffix = '', decimals =
1, length = 100, fill = '', printEnd = "\r"):
    """
    Call in a loop to create terminal progress bar
    @params:
        iteration   - Required  : current iteration (Int)
        total       - Required  : total iterations (Int)
        prefix      - Optional  : prefix string (Str)
        suffix      - Optional  : suffix string (Str)
        decimals    - Optional  : positive number of decimals in percent
complete (Int)
        length      - Optional  : character length of bar (Int)
        fill        - Optional  : bar fill character (Str)
        printEnd    - Optional  : end character (e.g. "\r", "\r\n") (Str)
    """
    percent = ("{0:." + str(decimals) + "f}").format(100 * (iteration /
float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print('\r%s |%s| %s%% %s' % (prefix, bar, percent, suffix), end =
printEnd)
    # Print New Line on Complete
    if iteration == total:
        print()

# importing data into pandas
diabetes_data = pd.read_csv('pima-indians-diabetes.csv', \
                            names=['Pregnancies', 'Glucose', 'Blood
Pressure', 'Skin Thickness', \
                                   'Insulin', 'BMI', 'Diabetes Pedigree
Function', 'Age', 'Outcome'])

# getting all positive and negative rows for test selection
outcome_positive = diabetes_data[diabetes_data['Outcome'] == 1]
outcome_negative = diabetes_data[diabetes_data['Outcome'] == 0]

N = 768 # number of samples
m = 8 # number of attributes

n_train_test_values = [40, 80, 120, 160, 200]
percent_total_list = [] # list of averaged totals from 1000 iterations of
each 40, 80, 120, 160, 200
n_test_total = 0 # total number of test questions for the 1000 iterations
correct_total = 0 # total number of correct answers for 1000 iterations
```

```
printProgressBar(0, 1000, prefix = 'Progress:', suffix = 'Complete', length =
50)


for value in n_train_test_values:

    n_train_test = value
    print("n value: " + str(n_train_test) + "\n")
    n_test_total = 0 # init to zero
    correct_total = 0 # init to zero

    for i in range (0,1000):
        # getting training data
        X_train = outcome_negative.sample(n=n_train_test) # get n_train_test
negative rows
        positive = outcome_positive.sample(n=n_train_test) # get n_train_test
positive rows
        X_train = X_train.append(positive, ignore_index = True) # append
positive rows to negative rows
        X_train = X_train.sample(frac=1) # shuffle training rows

        # getting testing data
        ans = pd.merge(diabetes_data,X_train, how='outer', indicator=True) #
get the rest of the data as training data
        X_test = ans[ans['_merge'] == 'left_only'] # get all values not in
both sets
        del X_test['_merge'] # remove last column

        # assign t values to outcomes of X
        t_train = pd.DataFrame(X_train['Outcome'], columns=['Outcome'])
        t_test = pd.DataFrame(X_test['Outcome'], columns=['Outcome'])

        # creating list of correct answers to allow for interation when
checking for correct answers
        t_test_list = X_test['Outcome']

        del X_test['Outcome'] # remove last column
        del X_train['Outcome'] # remove last column

        n_train,m = X_train.shape
        n_test,m = X_test.shape

        # define the tensors
```

```
        X = tf.placeholder(tf.float64, shape=(None, m), name='X') # input
features vector
        t = tf.placeholder(tf.float64, shape=(None, 1), name='t') # target
values
        n = tf.placeholder(tf.float64, name='n') # number of samples
        XT = tf.transpose(X)
        w = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT,X)), XT), t) #
w = inv(X'*X)*X'*t

        # predicted value
        y = tf.matmul(X,w)

        # mean squared error of the prediction training set
        MSE = tf.div(tf.matmul(tf.transpose(y-t), y-t), n)

        w_star = tf.placeholder(tf.float64, shape=(m, 1), name='w_star')
        y_test = tf.matmul(X, w_star)

        with tf.Session() as sess:
            # running tensorflow sessions
            MSE_train_val, w_val = \
            sess.run([MSE, w], feed_dict={X : X_train, t : t_train, n :
n_train})

            y_test_val = \
            sess.run([y_test], feed_dict={X : X_test, t : t_test, n : n_test,
w_star : w_val})

            correct = 0

            # testing all the values
            for prediction, actual in zip(y_test_val[0], t_test_list):

                if (prediction[0] >= 0.5 and actual == 1):
                    correct+=1

                elif (prediction[0] < 0.5 and actual == 0):
                    correct+=1

        # update progress bar on terminal screen
        printProgressBar(i+1, 1000, prefix = 'Progress:', suffix =
'Complete', length = 50)

        # increment total values tested and correct for n iteration
        n_test_total += n_test
```

```
        correct_total += correct

    # calculate total for n percent correct
    percent_correct = (correct_total/n_test_total)*100
    percent_total_list.append(percent_correct)

# print final list
print(percent_total_list)
```