CSE4088 Introduction to Machine Learning

Homework 3

Huzeyfe Ayaz

150119700

Part 1 - Gradient Descent

Q4

$$(ue^v - 2ve^{-u})^2 \Rightarrow \frac{\partial E}{\partial u}(ue^v - 2ve^{-u})^2$$

$$\Rightarrow 2\frac{\partial E}{\partial u}(ue^v - 2ve^{-u}) \Rightarrow 2(ue^v - 2ve^{-u})\left[e^v - 2v(-1e^{-u}\cdot \ln e)\right]$$

$$\Rightarrow \boxed{2(ue^v - 2ve^{-u})(e^v + 2ve^{-u})}$$

Q5

For the question 5 I have implemented 3 functions which take the derivative of u, v and finds the error by given function in the question.

```
def partial_u(u, v): # taking the derivative of u
    return 2*(np.e**v + 2*v*np.e**-u)*(u*np.e**v - 2*v*np.e**-u)

def partial_v(u, v): # taking the derivative of v
    return 2*(u*np.e**v - 2*np.e**-u) * ( u*np.e**v - 2*v*np.e**-u)

def error(u, v): # taking the error by given function
    return (u*np.e ** v - 2*v* np.e ** -u)**2
```

Then, I wrote a script in order to find the number of steps required to get smaller error rate than 1e-14.

```
u, v, lr = 1, 1, 0.1 # setting the values of u, v and learning rate
step = 0 # keeps the number steps
e = 1 # initializing error with a bigger number than 1e-14
while e >= 1e-14:
    u, v = u - lr * partial_u(u, v), v - lr * partial_v(u, v)
    e = error(u, v)
    step += 1
```

Output: 10

Q6

For the question 6, I simply printed the values of u and v that comes from previous question as an output to see the result.

Output: (0.0447..., 0.0239...)

Q7

For the question 7, I have re-calculated an error after each derivation and iterate it in 15 times that makes 30 steps in total. At the end of iterations, result of the error is printed out.

```
u, v, lr = 1, 1, 0.1 # setting the values of u, v and learning rate
e = 1 # # initializing error with a bigger number than 1e-14
for i in range(15): # since we making 2 updates in for loop that makes
15*2=30 steps
    u = u - lr * partial_u(u, v) # taking the derivative of u
    e = error(u, v)              # finding the error
    v = v - lr * partial_v(u, v) # taking the derivative of v
    e = error(u, v)              # fing the error

print(e)
```

Output: 0.139813...

Part 2 – Logistic Regression

Q8 & Q9

Since the question 8 and 9 are related each other, I handled them in a single run. ‘determinant’ and ‘create_data_and_pick_target_function’ functions are implemented in previous homework helped me for this homework as well. I only made slight modifications on them. An additional function which is called as shuffle data is implemented to shuffling data as the name implies. On the other hand, LogisticRegression is implemented as a class like I did in previous

homework for PLA and LinearRegression. Stochastic Gradient Descent is implemented in update_weights function that applied for each sample in an epoch. When the norm of differences between previous weight and next weight is below 0.01 threshold. Training iteration stops and returns the number of epoch to find the answer of the question. Cross Entropy Loss is implemented to find the out-sample error. This process repeated 100 times and average of epoch and cross entropy loss is calculated.

```
avr_epoch = 0 # average of epoch
score = 0 # average of score
for i in range(100):
    X, f, y = create_data_and_pick_target_function(100) # creating trai
ning dataset
    X_test, _, y_test = create_data_and_pick_target_function(100, targe
t_function=f) # creating test dataset

    lr = LogisticRegression() # creating model
    epoch = lr.fit(X, y)
    y_prob, y_pred = lr.predict(X_test)

    avr_epoch += epoch
    score += np.mean(np.log(1 + np.exp(-
y_test * y_prob))) # cross entropy loss

print("Avrage Epoch: ", avr_epoch / 100)
print("Avrage Score: ", score / 100)
```

Output: Average Epoch: 348.68

Average Score: 0.1013…

Part 3 - Regularization with weight decay

Loading Data

Shared 2 raw data saved into txt files and read on the notebook. Since the data is not organized, a small data pre-processing step is applied before starting the questions.

```
with open('data/data_in.txt', 'r') as f: # reading the training data
    data_in = f.read().splitlines() # reads the txt file and split into
 lines

with open('data/data_out.txt', 'r') as f: # reading the test data
    data_out = f.read().splitlines()
```

```
data_in = np.array([i.split() for i in data_in]).astype(np.float32)
# data splitted from white spaces and converted to float32 from string
data_out = np.array([i.split() for i in data_out]).astype(np.float32)

X_train, y_train, X_test, y_test = data_in[:, :2], data_in[:, 2].astype
(int), data_out[:, :2], data_out[:, 2].astype(int)
```

Q2

For question 2, implementation of LinearRegression class is taken from previous homework but additionally, I have added else part in update_weights method that apply regularization if k is specified.

```
...
else:
      X_inv = np.dot(np.linalg.inv(np.dot(X.T, X) + 10**k *
                    np.identity(X.shape[1])), X.T)
      self.w = np.dot(X_inv, y)
```

After data pre-processing and class implementation, I have applied nonlinear transformation to both train and test data. Then, a linear regression model is created and trained with transformed data. In order to find the in and out sample errors I have utilized the accuracy_score function from scikit-learn and subtract the score from 1 to find the error.

```
trans_X_train = non_linear_transformation(X_train)
trans_X_test = non_linear_transformation(X_test)

lr = LinearRegression()
lr.fit(trans_X_train, y_train)

y_train_pred = lr.predict(trans_X_train)
y_test_pred = lr.predict(trans_X_test)


in_sample_error = 1-accuracy_score(y_train, y_train_pred)
out_sample_error = 1-accuracy_score(y_test, y_test_pred)


in_sample_error, out_sample_error
```

Output: (0.0285..., 0.08399...)

Q3

For question 3, I followed the same steps but in this case k parameter of fit method is specified as -3 for the sake of question.

```
lr = LinearRegression()
lr.fit(trans_X_train, y_train, -3)

y_train_pred = lr.predict(trans_X_train)
y_test_pred = lr.predict(trans_X_test)


in_sample_error = 1-accuracy_score(y_train, y_train_pred)
out_sample_error = 1-accuracy_score(y_test, y_test_pred)


in_sample_error, out_sample_error
```

Output: (0.0285…, 0.0799…)

Q4

For question 4, I followed the same steps but in this case k parameter of fit method is specified as 3 for the sake of question.

```
lr = LinearRegression()
lr.fit(trans_X_train, y_train, 3)

y_train_pred = lr.predict(trans_X_train)
y_test_pred = lr.predict(trans_X_test)


in_sample_error = 1-accuracy_score(y_train, y_train_pred)
out_sample_error = 1-accuracy_score(y_test, y_test_pred)


in_sample_error, out_sample_error
```

Output: (0.3714…, 0.4360…)

Q5 & Q6

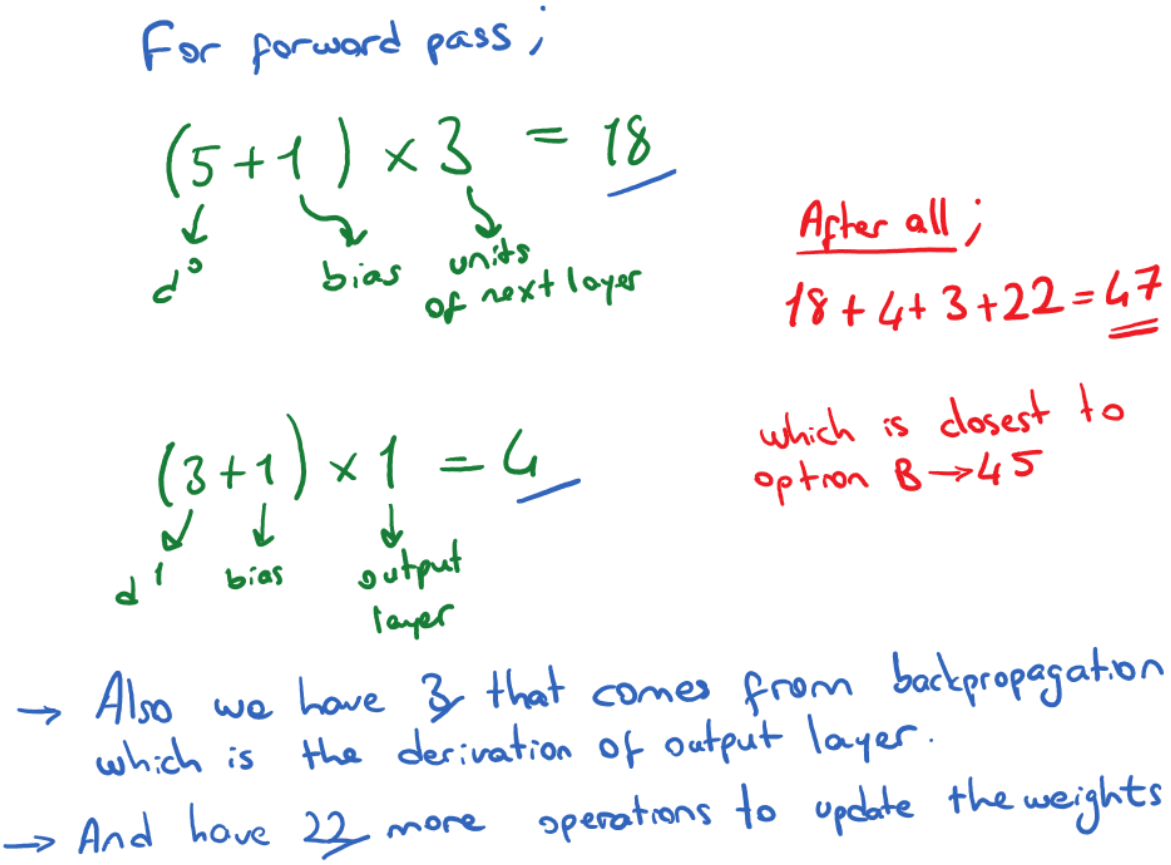
For question 5 and 6, I followed the same steps but in this case k parameter of fit method is specified in for loop that values comes from the choices of the question. Then, out sample error for each k value is hold in a dictionary.

```
min_out = {}
for k in [2, 1, 0, -1, -2]: # looping over the choices :)
    lr = LinearRegression()
    lr.fit(trans_X_train, y_train, k)

    y_train_pred = lr.predict(trans_X_train)
    y_test_pred = lr.predict(trans_X_test)

    in_sample_error = 1-accuracy_score(y_train, y_train_pred)
    out_sample_error = 1-accuracy_score(y_test, y_test_pred)

    min_out[k] = out_sample_error


min_out # -1 gives the min out error which
          closest to 0.06 in question 6
```

Output: {-2: 0.08399999999999996,
-1: 0.05600000000000005,
0: 0.09199999999999997,
1: 0.124,
2: 0.22799999999999998}

Part 4 – Neural Networks

Q8



Q9

The way of having the minimum number of weights is creating a fully connected layers with single unit. However, since we have bias term each must have 2 units. In that case, we have 18*2 number of weights for hidden layers which makes 36 weights. Additionally, we have 10 input layer and 10 weights come from there. As a result, we have 36+10=46 min number of weights in total.

Q10

For question 10, it would be better to keep the layer size as small as possible since it will generate more weights since neurons are fully connected. However, 10 input unit and 36 hidden unit in a single layer will yield 396 number of weights and incrementing hidden layer size to 2 may cause more weights since we will increase the connections. For that reason, I have written a simple script to find the max number of weights for 2 hidden layers with different unit sizes. In the code that I wrote placed in next page, updates max_num_weights and pattern when we have more weights. At the end, output is printed out.

```
max_num_weights = 0
pattern = 0

total_hidden_units = 36
for l1_u in range(2, 34): # since a layer must have at least 2 unit wit
h bias term
    l2_u = total_hidden_units - l1_u

    num_weights = (l1_u-1)*10 + (l2_u-1)*l1_u + (1 * l2_u)
    # I subtract -1 from each layer units to eliminate bias
    # and 1 * l2_u comes from the last hidden layer and output layer

    if num_weights > max_num_weights:
        max_num_weights = num_weights
        pattern = [10, l1_u, l2_u, 1]

pattern, max_num_weights
```

Output: ([10, 22, 14, 1], 510)