CS 156a Problem Set 6

Amitesh Anand Pandey

October 2024

Linear Regression Error

Problem 1

Since $\mathcal{H}' \subset \mathcal{H}$, in general, it will fail to explain as much of f as $\mathcal{H}$ and the deterministic noise will increase, option **[b]**.

Regularization with Weight Decay

Problem 2

From running the code in the appendix, we get the closest error as $(0.03, 0.08)$, so **[a]** is the correct option.

Problem 3

From running the code in the appendix, we get the closest error for $k = -3$ as $(0.03, 0.08)$ again, so **[d]** is correct.

Problem 4

From running the code in the appendix, we get the error as $(0.37, 0.44)$ for $k = 3$, so the closest option is $(0.4, 0.4)$, so the correct option is **[e]**.

Problem 5

From running the code in the appendix, we find that $k = -1$ minimizes out of sample error rate of classification so the correct option is **[d]**.

Problem 6

From varying k between $[-50, 50) \cap \mathbb{Z}$, we find that the smallest out of sample error is 0.056, closest to 0.06, so **[b]**.

Regularization with Polynomials

Problem 7

Let's look at $\mathcal{H}(10, 0, 3)$ first, by definition we have

$$\mathcal{H}(10,0,3) = \left\{ h \mid h(x) = \sum_{q=0}^{10} w_q L_q(x); w_q = 0 \text{ for } q \geq 3 \right\}$$
$$= \left\{ h \mid h(x) = \sum_{q=0}^{2} w_q L_q(x) \right\} = \mathcal{H}_2$$
$$\mathcal{H}(10,0,4) = \mathcal{H}_3$$

Now we make the observation that necessarily for $\forall i, j \in \mathbb{N}$, if $i < j$, then $\mathcal{H}_i \subset \mathcal{H}_j$, this is a logical conclusion from the definitions in the question. Set theoretically, we also know $\mathcal{H}_i \subset \mathcal{H}_j \implies \mathcal{H}_i \cap \mathcal{H}_j = \mathcal{H}_i$. This brings us to the fact that $\mathcal{H}(10,0,3) \cap \mathcal{H}(10,0,4) = \mathcal{H}_2 \cap \mathcal{H}_3$ given $\mathcal{H}_2 \subset \mathcal{H}_3$ this implies directly that $\mathcal{H}_2 \cap \mathcal{H}_3 = \mathcal{H}_2$. So the correct option then is option **[c]**.

Neural Networks

Problem 8

First, for the forward pass, operations of the form $w_{ij}^{(l)} x_i^{(l-1)}$, we have 2 layers, for the first layer, there are 5 inputs causing 2 operations each for this results in 10 operations. In the second layer, we effectively have 3 inputs sending 1 signal each, this results in 3 operations, so the forward pass has 13 operations in total. In the backward pass, we have the final layer sending 1 signal each to three "inputs" of the second layer, resulting in 3 operations. Then two non trivial nodes of the second layer each send 5 signals (one to each node of the first layer). This results in 13 operations. Each of these bring about a corresponding update to the weight vector, which means there are 13 total updates to the weight vector systematically at each of these steps. Finally, once back propagation has wrapped up, there is an ultimate update to the weight vector once again causing 5 operations, so the total operations taken is 44, closest to 45, which is option **[d]**.

Problem 9

The total weights are reduced greatly when there is only one weight to pass over from one unit of the hidden layer to another. We are forced by the question to have 10 units sending a signal each to a 1-node first unit, causing 10 weights. Now we have 36 units, which require $\left\lceil \frac{36-1}{2} \right\rceil$ weights to reach the final layer. This gets us to 18 weights. For each of these units, there is also the trivial node causing 1 weight for each unit traversal, causing 18 more units. Thus we have $10 + 18 + 18 = 46$ weights, equal to option **[a]**.

Problem 10

This problem does appear tedious calculatively, and I have run out of time to code this. Because of the nature of the question, I am **guessing** the largest possible **[e]** is correct.

Linear Regression with Weight Decay

[46]:
```
import numpy as np

in_sample_data = []
out_sample_data = []
with open('in_dta.txt', 'r') as f:
    for line in f:
        in_sample_data.append(line)

with open('out_dta.txt', 'r') as f:
    for line in f:
        out_sample_data.append(line)
```

[47]:
```
for i in range(len(in_sample_data)):
    act = in_sample_data[i].split(' ')
    act2 = []
    for val in act:
        if val != '':
            act2.append(float(val))
    in_sample_data[i] = act2
in_sample_data = np.asarray(in_sample_data)
```

[48]:
```
for i in range(len(out_sample_data)):
    act = out_sample_data[i].split(' ')
    act2 = []
    for val in act:
        if val != '':
            act2.append(float(val))
    out_sample_data[i] = act2
out_sample_data = np.asarray(out_sample_data)
```

[49]:
```
def non_linear_transform(X):
    x_transformed = []
    for x in X:
        x1 = x[0]
        x2 = x[1]
        x_transformed.append([1, x1, x2, x1**2, x2**2, x1*x2, abs(x1 - x2),␣
 ↪abs(x1 + x2)])
    return np.asarray(x_transformed)
```

[50]:
```
def linear_regression(X, y):
    X_plus = np.linalg.inv(X.transpose().dot(X)).dot(X.transpose())
    w = X_plus.dot(y)
    return(w)
```

[51]:
```
def linear_regression_weight_decay(X, y, l):
    X_prime = np.linalg.inv(X.transpose().dot(X) + l*np.eye(X.shape[1])).dot(X.
 ↪transpose())
    w = X_prime.dot(y)
    return(w)
```

[54]:
```
def get_error(X, w, y):
    correct_pos = []
    ct = 0
    for x in X:
        if np.sign(w.dot(x)) == y[ct]:
            correct_pos.append(ct)
        ct += 1
    err = 1-len(correct_pos)/float(ct)
    return err
```

[56]:
```
X = in_sample_data[:,:2]
y = in_sample_data[:,2]
Z = non_linear_transform(X)
w = linear_regression(Z,y)
in_sample_err = get_error(Z,w,y)
test_X = out_sample_data[:,:2]
test_Z = non_linear_transform(test_X)
test_y = out_sample_data[:,2]
out_sample_err = get_error(test_Z,w,test_y)
print('Regular Linear Regression: ('+ str(round(in_sample_err, 2)) + ', ' +␣
 ↪str(round(out_sample_err,2)) + ')')

for k in [-3, 3, 2, 1, 0, -1, -2]:
    l = 10**(k)
    X = in_sample_data[:,:2]
    X = non_linear_transform(X)
    y = in_sample_data[:,2]
    w = linear_regression_weight_decay(X,y,l)
    in_sample_err = get_error(X,w,y)
    test_X = non_linear_transform(out_sample_data[:,:2])
    test_y = out_sample_data[:,2]
    out_sample_err = get_error(test_X,w,test_y)
    print('Linear Regression with weight decay' + ' k = ' + str(k) + ': ('+␣
 ↪str(round(in_sample_err, 2)) + ', ' + str(round(out_sample_err,2)) + ')')
```

```
Regular Linear Regression: (0.03, 0.08)
Linear Regression with weight decay k = -3: (0.03, 0.08)
Linear Regression with weight decay k = 3: (0.37, 0.44)
Linear Regression with weight decay k = 2: (0.2, 0.23)
Linear Regression with weight decay k = 1: (0.06, 0.12)
Linear Regression with weight decay k = 0: (0.0, 0.09)
Linear Regression with weight decay k = -1, -2: (0.03, 0.06), (0.03, 0.08)
```

[57]:
```
min_err = 100
for k in range(-50,50):
    l = 10**(k)
    w = linear_regression_weight_decay(X,y,l)
    test_X = non_linear_transform(out_sample_data[:,:2])
    test_y = out_sample_data[:,2]
    out_sample_err = get_error(test_X,w,test_y)
    min_err = min(min_err, get_error(test_X,w,test_y))
print(min_err)
```

```
0.05600000000000005
```