CS/CNS/EE 156a

Homework 6

Sung Hoon Choi

(Last name: Choi)

1. Answer: [b]

If you use a simpler(less sophisticated) hypothesis, you will be able to capture less of the target function f, and the deterministic noise would increase.

2. Answer: [a]

The values I got are Ein = 0.0285714, Eout=0.084

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW6 Problem 2
import numpy as np

IN_DTA_NUM = 35        #number of insample data points
OUT_DTA_NUM = 250      #number of outsample data points

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                            # Format: (x1,x2,label)
                            #           ...
                            #        (x1,x2,label)

def nonlinear_trans(x,lineNum):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row)
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                           #               ...
                                           #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)

# Main Code
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
```

```
weights = calculate_weight(transformed_data,y)

calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
```

3. Answer: [d]
The values I got were Ein=0.0285714, Eout=0.08
Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW6 Problem 3
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points
NONLINEAR_TERMS_NUM = 8 # number of terms in the nonlinear transformed
                      # polynomial: (1,x1,x2,x1^2,x2^2, ... ,abs(x1+x2))

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                             # Format: (x1,x2,label)
                             #           ...
                             #        (x1,x2,label)

def nonlinear_trans(x,lineNum):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row)
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                            #               ...
                                            #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_regularized_weight(nonlinear_x, y, Lambda):
    reg_weight_inter1 = np.dot(np.transpose(nonlinear_x),nonlinear_x)+Lambda*np.identity(NONLINEAR_TERMS_NUM)
    reg_weight_inter2 = np.dot(np.linalg.inv(reg_weight_inter1),np.transpose((nonlinear_x)))
    reg_weight = np.dot(reg_weight_inter2,y)
    return reg_weight

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)

# Main Code
k = -3
Lambda = 10**k
```

```
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)

calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
```

4. Answer: [e]

The values I got are: Ein=0.371, Eout=0.436

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW6 Problem 4
import numpy as np

IN_DTA_NUM = 35         # number of insample data points
OUT_DTA_NUM = 250       # number of outsample data points
NONLINEAR_TERMS_NUM = 8 # number of terms in the nonlinear transformed
                      # polynomial: (1,x1,x2,x1^2,x2^2, ... ,abs(x1+x2))

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                             # Format: (x1,x2,label)
                             #          ...
                             #        (x1,x2,label)

def nonlinear_trans(x,lineNum):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row)
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                            #                ...
                                            #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_regularized_weight(nonlinear_x, y, Lambda):
    reg_weight_inter1 = np.dot(np.transpose(nonlinear_x),nonlinear_x)+Lambda*np.identity(NONLINEAR_TERMS_NUM)
    reg_weight_inter2 = np.dot(np.linalg.inv(reg_weight_inter1),np.transpose((nonlinear_x)))
    reg_weight = np.dot(reg_weight_inter2,y)
    return reg_weight

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)
```

```
# Main Code
k = 3
Lambda = 10**k

insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)
calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
```

5. Answer: [d]

The out-of-sample errors(Eout) I got are:

k=2: Eout=0.228

k=1: Eout=0.124

k=0: Eout=0.092

k=-1: Eout=0.056

k=-2: Eout=0.084

Thus, Eout is smallest when k=-1.

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW6 Problem 5
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points
NONLINEAR_TERMS_NUM = 8 # number of terms in the nonlinear transformed
                      # polynomial: (1,x1,x2,x1^2,x2^2, ... ,abs(x1+x2))

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                             # Format: (x1,x2,label)
                             #           ...
                             #        (x1,x2,label)

def nonlinear_trans(x,lineNum):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row)
    return nonlin_transformed_data            # Format: (1,x1,x2,x1^2,x2^2,...)
                                            #                ...
                                            #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_regularized_weight(nonlinear_x, y, Lambda):
    reg_weight_inter1 = np.dot(np.transpose(nonlinear_x),nonlinear_x)+Lambda*np.identity(NONLINEAR_TERMS_NUM)
    reg_weight_inter2 = np.dot(np.linalg.inv(reg_weight_inter1),np.transpose((nonlinear_x)))
    reg_weight = np.dot(reg_weight_inter2,y)
    return reg_weight

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)

# Main Code
k = 2
Lambda = 10**k
print("k=2----------------------------")
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                 #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)
calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
print("----------------------------------\n")

k = 1
Lambda = 10**k
print("k=1----------------------------")
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                 #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)
calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
print("----------------------------------\n")

k = 0
Lambda = 10**k
print("k=0----------------------------")
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                 #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)
calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
print("----------------------------------\n")

k = -1
Lambda = 10**k
print("k=-1----------------------------")
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                 #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)
calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error
```

```
outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
print("---------------------------------\n")

k = -2
Lambda = 10**k
print("k=-2----------------------------")
insample_data = extract_data("in.dta")
insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
y = insample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
weights = calculate_regularized_weight(transformed_data, y, Lambda)
calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error


outsample_data = extract_data("out.dta")
outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
y = outsample_data_np[:,2]                  #extract y-labels from the input data
transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
print("---------------------------------\n")
```

6. Answer: [b]

By going through different integer values of k (from -20 to 20), I found that the out-of-sample error (Eout) is at its minimum at k=-1, with Eout=0.056. Thus, the answer is [b].

Please refer to the code below for derivation.

```
#Sung Hoon Choi
#CS/CNS/EE156a HW6 Problem 6
import numpy as np

IN_DTA_NUM = 35        # number of insample data points
OUT_DTA_NUM = 250      # number of outsample data points
NONLINEAR_TERMS_NUM = 8 # number of terms in the nonlinear transformed
                      # polynomial: (1,x1,x2,x1^2,x2^2, ... ,abs(x1+x2))

def extract_data(filename):
    data_array = []
    for line in open(filename):
        data=[]
        data_entries = line.split('  ')
        data_row = [float(data_entries[1]),float(data_entries[2]),float(data_entries[3].rstrip("\n"))]
        data_array.append(data_row)
    return data_array          #return the data from given dta file.
                             # Format: (x1,x2,label)
                             #            ...
                             #        (x1,x2,label)

def nonlinear_trans(x,lineNum):
    nonlin_transformed_data = []
    for i in range(0,lineNum):
        nonlin_transformed_row = []
        nonlin_transformed_row.append(1)
        nonlin_transformed_row.append(x[i][0])
        nonlin_transformed_row.append(x[i][1])
        nonlin_transformed_row.append(x[i][0]**2)
        nonlin_transformed_row.append(x[i][1]**2)
        nonlin_transformed_row.append((x[i][0]*x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]-x[i][1]))
        nonlin_transformed_row.append(abs(x[i][0]+x[i][1]))
        nonlin_transformed_data.append(nonlin_transformed_row)
    return nonlin_transformed_data             # Format: (1,x1,x2,x1^2,x2^2,...)
                                            #                ...
                                            #        (1,x1,x2,x1^2,x2^2,...)

def calculate_weight(nonlinear_x, label):
    return np.dot(np.linalg.pinv(nonlinear_x), label)

def calculate_regularized_weight(nonlinear_x, y, Lambda):
    reg_weight_inter1 = np.dot(np.transpose(nonlinear_x),nonlinear_x)+Lambda*np.identity(NONLINEAR_TERMS_NUM)
    reg_weight_inter2 = np.dot(np.linalg.inv(reg_weight_inter1),np.transpose((nonlinear_x)))
    reg_weight = np.dot(reg_weight_inter2,y)
    return reg_weight

def calculate_error(weights, nonlin_input_x, y, data_num):
    error_count = 0
    for i in range(0,data_num):
        g = np.sign(np.dot(weights.T,nonlin_input_x[i]))
        if(g != y[i]):
            error_count = error_count + 1
    print("Error: ", error_count/data_num)

# Main Code
for k in range(-20,20):     #Go through different integer values of k.
    Lambda = 10**k
    print("k= %d ----------------------------",k)
    insample_data = extract_data("in.dta")
    insample_data_np = np.array(insample_data)  #turn it into a numpy array to use numpy library functions
    y = insample_data_np[:,2]                  #extract y-labels from the input data
    transformed_data = nonlinear_trans(insample_data,IN_DTA_NUM)
    weights = calculate_regularized_weight(transformed_data, y, Lambda)
    calculate_error(weights,transformed_data,y,IN_DTA_NUM)  #Ein - InSample Error

    outsample_data = extract_data("out.dta")
    outsample_data_np = np.array(outsample_data)  #turn it into a numpy array to use numpy library functions
    y = outsample_data_np[:,2]                  #extract y-labels from the input data
    transformed_data = nonlinear_trans(outsample_data,OUT_DTA_NUM)
    calculate_error(weights,transformed_data,y,OUT_DTA_NUM)  #Eout - OutSample Error
    print("----------------------------------\n")
```

7. Answer: [c]

[a]: For H(10,0,3), w_q=0 for all q≥3. For H(10,0,4), w_q=0 for all q≥4. Thus, both don't have any polynomials of degree greater than or equal to 4 and their union won't have polynomial greater than or equal to 4.
[b]: H(10,1,3) has non-zero weights for polynomials of degrees greater than or equal to 3. H(10,1,4) has non-zero weights for polynomials of degrees greater than or equal to 4. Thus, the union of the two would include polynomials of degree greater than or equal to 4.

[c]: For H(10,0,3), w_q=0 for all q≥3. For H(10,0,4), w_q=0 for all q≥4. So, the intersection of these two would not have any polynomials of degree greater than or equal to 3. Therefore, H(10,0,3)∩H(10,0,4)=H_2
[d]: H(10,1,3) has non-zero weights for polynomials of degrees greater than or equal to 3. H(10,1,4) has non-zero weights for polynomials of degrees greater than or equal to 4. Therefore, H(10,1,3)∩H(10,1,4) would have polynomials of degrees greater than or equal to 4.

8. Answer: [d]

For forward-propagation, we do $x_j^{(l)} = \theta(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)})$. Thus, the number of $w_{ij}^{(l)} x_i^{(l-1)}$ operation is equivalent to the number of weights. If we take the constant nodes (bias) into account, the number of weights are 22 because $(5+1)*3+(3+1)*1=22$. Thus, the number of $w_{ij}^{(l)} x_i^{(l-1)}$ operations for forward-propagation is 22. Also, when calculating the error $\delta_i{}^l$, we need 3 operations of $w_{ij}^{(l)}\delta_j^{(l)}$ because the errors of last layer does not have $w_{ij}^{(l)}\delta_j^{(l)}$ terms, and the first layer(inputs) do not have errors by themselves.(Because they are "inputs" and there is no preceding signals) So, since we have three nodes in the middle layer, calculating $\delta_i{}^l$ requires 3 operations. Lastly, the back-propagation, we have to update each weights by $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta x_i^{(l-1)}\delta_j^{(l)}$. Thus, we need 22 operations of $x_i^{(l-1)}\delta_j^{(l)}$. Therefore, to sum up, we need 22+3+22=47 operations.

9. Answer: [a]
To minimize the total number of weights, we must disperse the units into as many layers as possible. This is because we "multiply" between the numbers of nodes on the pair of layers when finding the number of

weights, and thus it is best to spread the nodes into as many layers as possible. The way to disperse the units as widely as possible throughout the layers is by only putting two units per each layer.(One variable unit and one constant $x_0^{(l)}$). If we put two units on each layer, the total number of hidden layers would be 18. Thus, the number of weights can be calculated as following:

- From the input to the first hidden layer: 10 * 1 = 10
- Between hidden layers: 2 * 17 = 34
- From the last hidden layer to output: 2 * 1 = 2

Total number of weights = 10 + 34 + 2 = 46

Therefore, the total number of minimum weights is 10+34+2=46.

10. Answer: [e]

By using the same intuition from problem 9, we can deduce that we should minimize the number of hidden layers to increase the number of weights. If we use 1 hidden layer, the number of weights is 10*(36-1)+36*1=386. Now, let's find the maximum possible number of weights for 2 hidden layer case. If we let x units for 1st hidden layer and thus 36-x units for 2nd hidden layer, then the number of weights is

$$10 * (\mathrm{x} - 1) + \mathrm{x} * (36 - \mathrm{x} - 1) + (36 - \mathrm{x}) * 1$$

Notice that we are subtracting 1 from each term because the constant bias term is already included when counting the number of units on each hidden layer.

If we find the maximum value of this equation, we get 510 when x=22 (I used Wolfram-Alpha to find the maximum value). Therefore, the maximum possible number of weights would be [e], 510.