CS430 - DATA MINING- HOMEWORK 7

1.) EXERCISES

1.1) [2 points] Using this information, answer the following questions:

(Show all work, and all answers should be rounded to 3 decimal places OR POINTS WILL BE TAKEN OFF!)

(a) Use forward propagation to compute the predicted output.

The weight matrix, W, is: [1, 1, -1, 0.5, 1, 2]T .
Hidden layer uses RelU
Output layer uses Sigmoid
Assume squared error
x = 4
y = 0
A

z : [4 * 1 , 4 * 1 , 4 * (-1)] = [4, 4, -4]

RelU(z) → Z: [4, 4, 0]
Y = Z * W [0.5, 1, 2]T = 4 * 0.5 + 4 * 1 + (0) * 2 = 2 + 4 + 0 = 6

y = sigmoid(Y) = 1 /(1 + e^(-Y)) = 0.998

(b) What is the loss or error value?

Squared Error = (correct value - prediction)^2 = (0 - 0.998)^2 = (-0.998)^2 = 0.996

(c) Using backpropagation, compute the gradient of the weight vector, that is, compute the partial derivative of the error with respect to all of the weights.

aa
Gradient Vector = gradient [W1, W2, W3, W4, W5, W6] = derivative [dE/dW1, dE/dW2, dE/dW3, dE/dW4, dE/dW5, dE/dW6]

In order to calculate the derivatives of the error with respect to the error, that is the gradient of the error in terms of the weights, we will need the equation of the error in terms of the weights.

Squared Error = (correct value - prediction)^2 = (correct value - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6))^2

dE/dW1 = (-2) * error * (sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) * (1 - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6)) * x * W4)

as long as x*W1 > 0

dE/dW2 = (-2) * error * (sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) * (1 - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6)) * x * W5)
as long as x*W2 > 0

dE/dW3 = (-2) * error * (sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) * (1 - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6)) * x * W6)

as long as x*W3 > 0, if not… = 0

dE/dW4 = (-2) * error * (sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) * (1 - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6)) * RelU(x*W1)

dE/dW5 = (-2) * error * (sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) * (1 - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6)) * RelU(x*W2)

dE/dW6 = (-2) * error * (sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) * (1 - sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6)) * RelU(x*W3)

Taking ‘constant1’ = sigmoid(RelU(x*W1) * W4 + RelU(x*W2) * W5 + RelU(x*W3) * W6) = sigmoid(6) = 0.9975, which is an often repeated structure.

Error = - 0.998
Taking ‘constant2’ = (-2) * error * (‘constant1’) * (1 - ‘constant1’) =
W: [1, 1, -1, 0.5, 1, 2]

‘constant’ = (-2) * (-0.9975) * (0.9975 *(1 - 0.9975)) = + 0.004975

dE/dW1 = ‘constant2’ * x * W4 = (0.004975) * 4 * 0.5 = 0.00995

dE/dW2 = ‘constant2’ * x * W5 = (0.004975) * 4 * 1 = 0.0199

dE/dW3 = ‘constant2’ * x * W6 = (0.004975) * 4 * 2 = 0.0398 !!!!!!!

as long as x*W3 > 0, if not… = 0
dE/dW3 = 0
4 * (-1) = -4 !!!!!!!!

dE/dW4 = ‘constant2’ * RelU(x*W1) = (0.004975) * 4 = 0.0199

dE/dW5 = ‘constant2’ * RelU(x*W2) = (0.004975) * 4 = 0.0199

dE/dW6 = ‘constant2’ * RelU(x*W3) = (0.004975) * RelU(-4) = 0

Gradient Vector = [0.00995, 0.0199, 0.00, 0.0199, 0.0199, 0.00]

(d) Using a learning rate of 1.0, compute new weights from the gradient. With the new weights, use forward propagation to compute the new predicted output, and the loss (error).

W, is: [1, 1, -1, 0.5, 1, 2]T .

Gradient Vector = [0.00995, 0.0199, 0.00, 0.0199, 0.0199, 0.00]

Weights = Weights - 1.0 * Gradient Vector

W1 = 1.0 -1.0 * dE/dW1 = 1.0 -1.0 * (0.00995) = ***0.99***

W2 = 1.0 - 1.0 * dE/dW2 = 1.0 -1.0 * (0.0199) = ***0.98***

W3 = -1.0 -1.0 * dE/dW3 = -1.0 -1.0 * (0) = ***-1.00***

W4 = 0.5 -1.0 * dE/dW4 = 0.5 -1.0 * (0.0199) = ***0.48***

W5 = 1.0 -1.0 * dE/dW5 = 1.0 -1.0 * (0.0199) = ***0.98***

W6 = 2.0 -1.0 * dE/dW6 = 2.0 -1.0 * 0 = ***2.0***

W = [0.99, 0.98, -1.00, 0.48, 0.98, 2.0]

z : [4 * 0.99 , 4 * 0.98 , 4 * (-1.00)] = [3.96, 3.92, -4.00]

RelU(z) → Z: [3.96, 3.92, 0.00]

Y = Z * [0.48, 0.98, 2.0]T = 3.96 * 0.48 + 3.92 * 0.98 + (0) * 2.0 = 1.9 + 3.84 + 0.0 = 5.74

y = sigmoid(Y) = 1 /(1 + e^(-5.74)) = 0.997

(e) Comment on the difference between the loss values you observe in (b) and (d).

The value gotten in (d) should be closer to the correct value than the one predicted in (b). That is due to the fact that for the (d) case, one iteration of the Gradient Descent has been performed, therefore the weights have been adjusted closer to the correct ones in order to get correct predictions. That is to say, the model has learnt from previous data, hence it should get more accurate the more the model is trained.

1.2) [1 point] Tan Chapter 4, questions 14, 15.

14. For each of the Boolean functions given below, state whether the problem is linearly separable.

a. A AND B AND C

Yes. There is only one true case, apart from that one, all of them are false. So, it is possible to isolate one value linearly with a plane(even if the plane has two dimensions, it is a linear subspace of the three dimensions stated in the problem) because this value is in the extreme position or corner (A=B=C= 1).

b. NOT A AND B

The AND function is always separable linearly, so this one also. We just have to see it as if NOT A was another variable called C, then we would have a normal AND. It only would change that the only TRUE value of the four possible combinations will be placed in A = 0, B = 1, but it could be linearly isolated easily.

c. (A OR B) AND (A OR C)

Yes.

The conditions for the result being FALSE are:
A = FALSE, B = FALSE → Result = FALSE
A = FALSE, C = FALSE → Result = FALSE

The only FALSE cases are:

RESULT			
A	***B***	***C***	***Result***
0	0	1	FALSE
0	0	0	FALSE
0	1	0	FALSE

And by plotting this cases in a cube with 2^3 = 8 corners, which is the number of possible cases. We can see that the false corners are close to each other, and that they can be isolated linearly by a hyperplane..

d. (A XOR B) AND (A OR B)

No.

The conditions for the result being FALSE are:
A = FALSE, B = FALSE → Result = FALSE
A = TRUE, B = TRUE → Result = FALSE

RESULT		
A	***B***	***Result***
0	0	FALSE
1	1	FALSE

By looking at the truth table, we can see that this logic is the same as the simple XOR, therefore, as XOR function is not linearly separable, NO.

AND, OR and NAND logic function are linearly separable as it has been seen, but the XOR function NO.

15.
a. Demonstrate how the perceptron model can be used to represent the AND and OR functions between a pair of Boolean variables.

The AND function taking into account a pair of boolean variables would be like:

I1 = Input 1
I2 = Input 2
O = Output

AND		
I1	***I2***	***O***
0	0	0
1	0	0
0	1	0
1	1	1

GRAPHICAL REPRESENTATION OF AND		
I2 = 1	0	1
I2 = 0	0	0
I2 / I1	***I1 = 0***	***I1 = 1***

Graphically the two clusters can be divided by a straight line, therefore the Perceptron will be enough in order to separate both clusters. If two straight lines or a more complex non-linear line was needed, probably we would need more than a Neuron or a NN structure different from the one of the Perceptron.

y = sigmoid (x1 + x2 − 1.5) can be the perceptron function.
In order to overcome the -1.5, x1=x2=1 is needed.
x1 + x2 − 1.5 = 0 would be the hyperplane that divides both classes.

The OR function taking into account a pair of boolean variables would be like:

OR		
I1	***I2***	***O***
0	0	0
1	0	1
0	1	1
1	1	1

GRAPHICAL REPRESENTATION OF OR		
I2 = 1	1	1
I2 = 0	0	1
I2 / I1	***I1 = 0***	***I1 = 1***

Same happens with the OR logic function. Graphically the two clusters can be divided by a straight line leaving three ones at one side, and one zero at the other side. Therefore the Perceptron will be enough in order to separate both clusters. If two straight lines or a more complex non-linear line was needed, probably we would need more than a Neuron or a NN structure different from the one of the Perceptron.

y = sigmoid (x1 + x2 − 0.5) can be the perceptron function.
If x1 or x2 = 1, we overcome the -0.5.
x1 + x2 − 0.5 = 0 would be the hyperplane that divides both classes.

b. Comment on the disadvantage of using linear functions as activation functions for multi-layer neural networks.

When in a Neural Network composed of several layers, linear activation functions are used, some common issues may be encountered.

In the first place, as all the layers will be activated with linear functions, the whole set of multi-layer structures could be summed up into one layer. Therefore, the multilayer concept stops making sense. It would be like having a single layer Neural Network just with different weights that will be a function of the constants of the linear functions used as activation functions. That is to say, it would be like having a perceptron.

Secondly, backpropagation becomes harder to be performed due to the fact that the derivatives of the linear functions will be some constants that have nothing to do with the weight values. Therefore, the gradient descent of the backpropagation will mainly depend on the values of $x\rightarrow$ (x1, x2,..., xn) the estimators, and will depend to a lesser extent on the weights. However, with non linear activation functions such as sigmoid, there is more dependence of the gradient on the weights values because when multiplying the derivative of the error function, they will appear some terms depending also on the weight values, not only on the x estimators.

Finally, using linear activation functions does not stand up for the idea of computing non-linear data in the Deep Neural Networks. Therefore, it means making our Neural Network not able to handle non-linear data, which is the main purpose of the Deep Neural Networks, or at least one of the domains in which this kind of AI structure has excelled these last recent years.

1.3 [1 point] Consider a dataset that has 8 predictors. You train a neural network with 3 hidden layers and an output layer that predicts a continuous value (a regression problem). The first hidden layer has 16 neurons, the second has 8 neurons, and the third has 4 neurons. In this network, how many total parameters will you have?

The formula that returns the number of parameters of a Neural Network as a function of its nodes will be used as follows.

Parameters = INPUT * HIDDEN + HIDDEN * OUTPUT + HIDDEN + OUTPUT
parameters = 8 * 16 + 16 * 8 + 8 * 4 + 4 * 1 + 16 + 8 + 4 + 1 = 321

Check the formula in the lecture

Layers: 8 - 16 - 8 - 4 - 1

Therefore, the number of parameters due to the weights will be: 128 + 128 + 32 + 4 = 292 weights.

Biases? One bias per neuron in the hidden layers and output layers = 16 + 8 + 4 + 1 = 29

Parameters = 292 + 29 = 321