Phi Duong

3/24/2025

CSE 5693

**Machine Learning Written Assignment 3**

1. 4.1: What are the values of weights wo, w1, and w2 for the perceptron whose decision surface is illustrated in Figure 4.3? Assume the surface crosses the x1 axis at -1, and the x2 axis at 2.

To get the values of the weights we can express the equation of the line using the x1 and x2 variables

x2 = 2x1 + 2

Solve to get zero

x2 - 2x1 -2 = 0

We want the points on the left side of the line to be positive points. Therefore,

x2 - 2x1 -2 > 0

Rearrange

-2 - 2x1 +x2 > 0

This format matches the perceptron inequality

net = w0 + w1x1 + w2x2 > 0

Therefore.

w0 = -2, w1 = -2, w2 = 1

1. 4.2: Design a two-input perceptron that implements the boolean function A /\ -B. Design a two-layer network of perceptrons that implements A X OR B.

(by hand with only \*integers\* for the weights, not by a program to gain a better understanding, specify the weights and include a table for each hidden or output unit: rows have input combinations)

* + 1. units in the first/only layer: columns are input, output values (before and after threshold)
       1. (A /\ -B)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| boolean | inputs | weights | net | Threshold |
| A, B | x1=1, x2 =1 | w0 = -1,  w1= 1  w2= -1 | -1 + 1(1) - 1(1) = -1 | 0 |
| A, -B | x1=1, x2 =-1 | -1 + 1(1) - 1(-1) = 1 | 1 |
| -A, B | x1=-1, x2 =1 | -1 + 1(-1) - 1(1) = -3 | 0 |
| -A, -B | x1=-1, x2 =-1 | -1 + 1(-1) - 1(-1) = -1 | 0 |

* + - 1. (A X OR B)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| boolean | inputs | weights | | Hidden net inputs | Hidden Threshold |
| A, B | x1=1, x2 =1 | w1,0 =-1  w1,1 =-1  w1,2 =1 | w2,0 =-1  w2,1 =1  w2,2 =-1 | hnet1 = -1 - 1(1) + 1(1) = -1  hnet2 = -1 + 1(1) - 1(1) = -1 | h1 = 0  h2 = 0 |
| A, -B | x1=1, x2 =-1 | hnet1 = -1 - 1(1) + 1(-1) = -3  hnet2 = -1 + 1(1) - 1(-1) = 1 | h1 = 0  h2 = 1 |
| -A, B | x1=-1, x2 =1 | hnet1 = -1 - 1(-1) + 1(1) = 1  hnet2 = -1 + 1(-1) - 1(1) = -3 | h1 = 1  h2 = 0 |
| -A, -B | x1=-1, x2 =-1 | hnet1 = -1 - 1(-1) + 1(-1) = -1  hnet2 = -1 + 1(-1) - 1(-1) = -1 | h1 = 0  h2 = 0 |

* + 1. Units in the second layer: columns are input, hidden, output values (before and after threshold)
       1. (A X OR B)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| inputs | Hidden Inputs | weights | Net output | Output Threshold |
| x1=1  x2 =1 | h1 = 0  h2 = 0 | wO,0 =-1  wO,1 =2  wO,2 =2 | Outnet = -1 + 2(0) + 2(0) = -1 | Out = 0 |
| x1=1  x2 =-1 | h1 = 0  h2 = 1 | Outnet = -1 + 2(0) + 2(1) = 1 | Out = 1 |
| x1=-1  x2 =1 | h1 = 1  h2 = 0 | Outnet = -1 +2(1) + 2(0) = 1 | Out = 1 |
| x1=-1  x2 =-1 | h1 = 0  h2 = 0 | Outnet = -1 + 2(0) + 2(0) = -1 | Out = 0 |

1. 4.9: Recall the 8 x 3 x 8 network described in Figure 4.7. Consider trying to train a 8 x 1 x 8 network for the same task; that is, a network with just one hidden unit. Notice the eight training examples in Figure 4.7 could be represented by eight distinct values for the single hidden unit (e.g., 0.1,0.2, . . . ,0.8). Could a network with just one hidden unit therefore learn the identity function defined over these training examples? Hint: Consider questions such as "do there exist values for the hidden unit weights that can create the hidden unit encoding suggested above?” "do there exist values for the output unit weights that could correctly decode this encoding of the input?” and "is gradient descent likely to find such weights?'

No, it is not possible to do a single hidden node. It is possible to represent distinct values in a single unit for each input by setting distinct input weight values. The problem lies in the weights from the hidden units to the outputs. There are only 2 weight variables, w0 and w1. This creates a linear equation that intersects the horizon at only once. Multiple intersections are required to represent boundary outputs for each distinct values. For example, if we represent each output by a distinct value in the hidden unit (0.1, 0.2, …, 0.8) we would want to represent the second and third input by 2 boundary points: 0.15 < output2 <= 0.25; 0.25 < output3 <= 0.35. Since we only can use a single linear equation, we can only create 1 boundary point, therefore we can use, 0.15 < output2 and 0.25 < output3. The problem is that whenever the 3rd boundary is satisfied the 2nd will also be satisfied as well.

(d) With the programming assignment:

* + 1. discuss the hidden values in testIdentity using 3 and 4 hidden units (Why do 4 hidden units also work? What do the hidden values represent? Any significant difference in the number of iterations to convergence and why?)

The hidden values in the neural network represent abstract learned features the network gained from the training set. In this case, the hidden layers combined represent the features that make up the different outputs for the Identity test.

For these examples, I used 3 and 4 hidden units for this network. Both networks work to give 100% accuracy. We only need 3 layers because the dataset only allows 8 different outputs. Since this network uses a sigmoid function, each unit can represent roughly 2 possible outputs: 1 or 0. That means we need only 2^3 combinations to represent the unique outputs for the network. Additionally, 4 hidden units is possible because it can represent 2^4 = 16 different outputs which is greater than the 8 used.

Comparing the number of iterations, it took for each network to converge. It took around 3500 iterations for the 3 hidden units and 1500 iterations for the 4 hidden units. For this network, I used a learning rate of 0.15 and momentum constant of 0.05. The 4 hidden unit network performed faster, because 4 hidden units add another dimension to the hypothesis space which reduces the chances of getting stuck in local minimums. In addition, the higher complexity with 16 possible outputs increases the chances that the network can find more relationships and patterns within the training set.

* + 1. compare performance of using validation set to not using it in testIrisNoisy. Include a plot for the comparisons.

Firgures 1 and 2 shows the accuracy plot of the ANN with increasing noise in the training set. One of the ANN uses a validation set (Figure 2) while the other does not use one (Figure 1). Looking at the data, the the plot for both sets drops in accuracy quickly as the training set get populated with noise. For both Figures, there is a quick downward slope in accracy once noise was introduced to the training set. Firgure 2 takes a bit longer for it to loose accuracy compared to Figure1. Firgure 2 also has a dip up form the curve from 0% to 36% and back down to 0%. This is probably due to random chance that the ANN was able to ignore some of the noise of the data. The trainingset and testset follow roughly the same trajectory

A graph with a line and a line

AI-generated content may be incorrect.

**Figure 1: accuracy of the ANN with noise**

A graph with red and blue lines

AI-generated content may be incorrect.

**Figure 2: accuracy of the ANN with noise and a validation set**