U18169024

Report:

My convergence condition was met after my program trained using 15000 training cases this was because my chosen epoch value is 15000. I chose this because I saw that I would in general get better test accuracy and the performance gap between 15000 and 10000 was less that a second(On my computer anyway)

Bias node values are always 1

I found the following to be the best parameters:

Learning rate = 0.1

Default weights: Random number between 0 and 0.5

The number of hidden layers: 1

I read many different reports of people doing neural networks with backpropagation on the iris dataset and most of them came to the agreement that one hidden would be enough. After some testing, I discovered that one hidden layer is enough. This is mainly because even if I added more layers, the output would still be a linear combination of the input data.

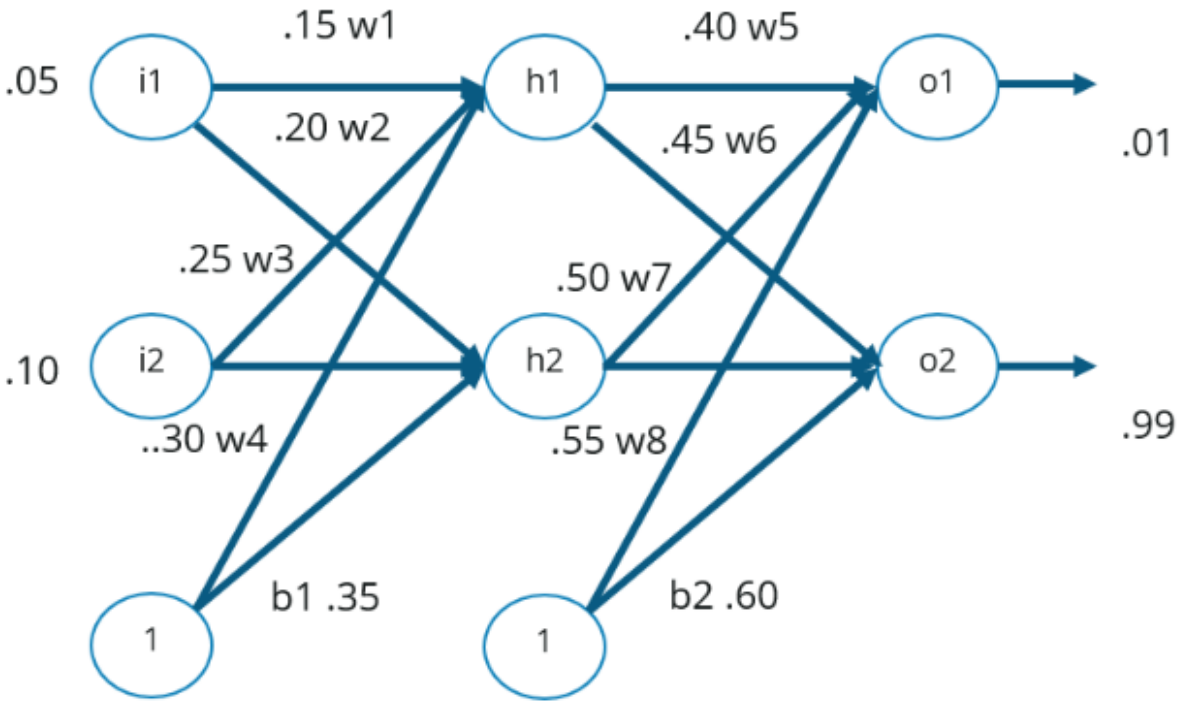
Number of nodes in the hidden layer: 4

I used the same method in determining the number of hidden layers as I did in determining the number of nodes in the hidden layer. Most reports seem to indicate that their where less unsolved inputs when 4 hidden layers was used. I also tested this, and it seemed that they were correct.

Activation function: Sigmoid

I used Sigmoid from the start because it seemed like most of the research papers used it. It seemed like the logical 'squash' function to use because it steadily increases or decreases. It doesn't have a sudden jerk in one direction. The sigmoid function works as follows: $1/(1+e^{-x})$

My learning rules differed a bit from what was said in the lecture notes because I struggled to figure out how it works. So, I tried the following:



Let's start with this image. I1 and i2 is the input nodes. W1-w4 is the input to hidden layer weights. H1 and h2 is hidden layer 1 and 2. W5-W8 is the weights between the hidden layer and the output layer. And for the output, o1 and o2 is the output layers with 0.01 and 0.99 what the output should be. Lastly b1 and b2 is the bias with 1 base number for both and with weights on all layers 0.35 and 0.60 respectively.

Forward propagate:

So, for net input h1you would do the following:

Net h1 = weight 1 * input 1 + weight 2 * input 2 + bias 1 weight * bias 1 base number

And then for output of h1:

And then you would apply the activation function like follows: $1/(1+e^{-(\text{Net h1})})$

The same would be done to determine h2 except other weights would be used like:

Net h2 = weight 3 * input 1 + weight 4 * input 2 + bias 1 weight * bias 1 base number

Sigmoid would be applied to h2 again

And then for output o1:

Out o1 = weight 5 * hidden 1 + weight 6 * hidden 2 + bias 2 weight * bias 2 base number

And you would again apply the sigmoid function for the output as follows:

$1/(1+e^{-(\text{Out 1})})$

The same would again be done to determine o2:

Out o2 = weight 7 * hidden 1 + weight 8 * hidden 2 + bias 2 weight * bias 2 base number and this value would be squashed again

Backpropagate:

Because the backpropagation on the output to hidden layer weights isn't applied when calculating the hidden to input weights meant that I start my backpropagation with the hidden to start weights and biases.

Update weights and biases in the hidden to start layer weights. To do this we will need all the weights and output nodes that the hidden layer node from the targeted weight is connected to. This means that for weight w1 we will need h1 , w5 and o1 as well w6 and o2. We will also need the target output for both o1 and o2. We will then use that data and compute the following:

(A) = (o1-target o1) * (o1(1-o1)) * weight w5
(B) = (o2-target o2) * (o2(1-o2)) * weight w6
(C) = (A)+(B)
(D) = (h1(1-h1)) *(C)

(Weight 1 update) = weight 1 – learning rate*(D)

(Bias 1weight 1) = Bias1 Weight 1 - learning rate*(D)

(Weight 3 update) = weight 3 – learning rate*(D)

The same process will be followed more or less when updating w2 , w4 and bias 1 weight 2:

(A) = (o2-target o2) * (o2(1-o2)) * weight w8
(B) = (o1-target o1) * (o1(1-o1)) * weight w7
(C) = (A)+(B)

(D) = (h2(1-h2)) *(C)

(Weight 3 update) = weight 3 – learning rate*(D)

(Bias 1weight 2) = Bias1 Weight 2 - learning rate*(D)

(Weight 4 update) = weight 4 – learning rate*(D)

From here we will calculate the updated values for w5 – w8:

The calculations are a bit different from those used to find the updated values of weights w1-w4 and the bias 1 because we don't have to calculate the impact of the weights that are connected to the output layer before we can calculate the update. The way the weights are updated are:

(A) = (o1 – target o1)
(B) =-o1(1-o1)
(C) = 1*h1
(D) = (A)*(B)*(C)

(Wight w5 update) = w5 – learning rate*(D)

(Bias 2 weight 1) = Bias 2 weight 1– learning rate*(D)

The rest of the weight can be generated similarly:

(A) = (o2 – target o2)
(B) =-o2(1-o2)
(C) = 1*h1
(D) = (A)*(B)*(C)

(Wight w6 update) = w6 – learning rate*(D)

(Bias 2 weight 2) = Bias 2 weight 2– learning rate*(D)

And for w7 and w8 as follows:

(A) = (o1 – target o1)
(B) =-o1(1-o1)
(C) = 1*h2
(D) = (A)*(B)*(C)

(Wight w7 update) = w7 – learning rate*(D)

(E) = (o2 – target o2)
(F) =-o2(1-o2)
(G) = 1*h2
(H) = (E)*(F)*(G)

(Wight w8 update) = w8 – learning rate*(H)

From here all the weights should be updated, and the algorithm should be ready to run again.

I know I'm not the best at explaining so I'm leaving two links here that might help clear things up with my explanation:

- https://www.edureka.co/blog/backpropagation/

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

I used both sources when I tried to figure out backpropagation works. I printed the second link out and followed on both 1 and 2. I really hope what I did seems clear to you.