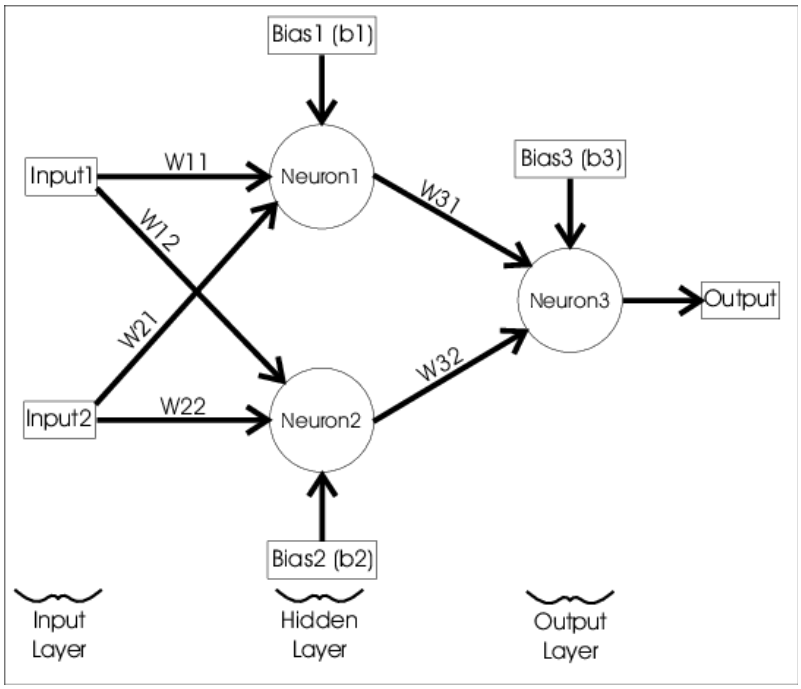
Roll No. MIT2019031
Name: Shrikant Temburwar

X-NOR using Back Propagation Algorithm

For implementing X-NOR, the neural network needs to produce two different decision planes to linearly separate the input data based on the output patterns. This is achieved by using the concept of hidden layers. The neural network will consist of one input layer with two nodes (X1, X2); one hidden layer with two nodes (since two decision planes are needed); and one output layer with one node (Y). Hence, the neural network looks like this:



It takes two input, processes it through multiple neurons from multiple hidden layers and returns the result using an output layer. This result estimation process is technically known as “Forward Propagation“. Next, we compare the result with actual output. The task is to make the output to the neural network as close to the actual (desired) output. Each of these neurons are contributing some error to final output.

Then we try to minimize the value/ weight of neurons those are contributing more to the error and this happens while travelling back to the neurons of the neural network and finding where the error lies. This process is known as “Backward Propagation“.

In order to reduce this number of iterations to minimize the error, the neural networks use a common algorithm known as “Gradient Descent”, which helps to optimize the task quickly and efficiently.
Steps involved in Neural Network methodology:

0.) We take input and output

X as an input matrix
y as an output matrix

1.) We initialize weights and biases with random values (This is one-time initiation. In the next iteration, we will use updated weights, and biases).
Let us define:

wh as weight matrix to the hidden layer
bh as bias matrix to the hidden layer
wout as weight matrix to the output layer
bout as bias matrix to the output layer

2.) We take matrix dot product of input and weights assigned to edges between the input and hidden layer then add biases of the hidden layer neurons to respective inputs, this is known as linear transformation:

hidden_layer_input= matrix_dot_product(X,wh) + bh

3) Perform non-linear transformation using an activation function (Sigmoid). Sigmoid will return the output as 1/(1 + exp(-x)).

hiddenlayer_activations = sigmoid(hidden_layer_input)

4.) Perform a linear transformation on hidden layer activation (take matrix dot product with weights and add a bias of the output layer neuron) then apply an activation function (again used sigmoid, but you can use any other activation function depending upon your task) to predict the output

output_layer_input = matrix_dot_product (hiddenlayer_activations * wout) + bout
output = sigmoid(output_layer_input)

All above steps are known as “Forward Propagation“

5.) Compare prediction with actual output and calculate the gradient of the error (Actual – Predicted). Error is the mean square loss = ((Y-t)^2)/2

E = y – output

6.) Compute the slope/ gradient of hidden and output layer neurons (To compute the slope, we calculate the derivatives of non-linear activations x at each layer for each neuron). Gradient of sigmoid can be returned as x * (1 – x).

slope_output_layer = derivatives_sigmoid(output)
slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)

7.) Compute change factor(delta) at the output layer, dependent on the gradient of error multiplied by the slope of output layer activation

d_output = E * slope_output_layer

8.) At this step, the error will propagate back into the network which means error at hidden layer. For this, we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (wout.T).

Error_at_hidden_layer = matrix_dot_product(d_output, wout.Transpose)

9.) Compute change factor(delta) at hidden layer, multiply the error at hidden layer with slope of hidden layer activation

d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer

10.) Update weights at the output and hidden layer: The weights in the network can be updated from the errors calculated for training example(s).

wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learning_rate

wh = wh + matrix_dot_product(X.Transpose,d_hiddenlayer)*learning_rate

learning_rate: The amount that weights are updated is controlled by a configuration parameter called the learning rate

11.) Update biases at the output and hidden layer: The biases in the network can be updated from the aggregated errors at that neuron.

bias at output_layer =bias at output_layer + sum of delta of output_layer at row-wise * learning_rate

bias at hidden_layer =bias at hidden_layer + sum of delta of output_layer at row-wise * learning_rate

```
bh = bh + sum(d_hiddenlayer, axis=0) * learning_rate
bout = bout + sum(d_output, axis=0)*learning_rate
```

Steps from 5 to 11 are known as “Backward Propagation“

One forward and backward propagation iteration is considered as one training cycle.
When do we train second time then update weights and biases are used for forward propagation.
Above, we have updated the weight and biases for hidden and output layer and we have used full batch gradient descent algorithm.

Result:

The output with epochs = 100000 and learning rate = 0.1 is:

Input: [0 0] [0 1] [1 0] [1 1]

Expected Output: [1] [0] [0] [1]

Predicted Output: [0.98679849] [0.01134609] [0.01125494] [0.98833762]

The predicted output is close to the expected output. Hence, the neural network has converged to the expected output.