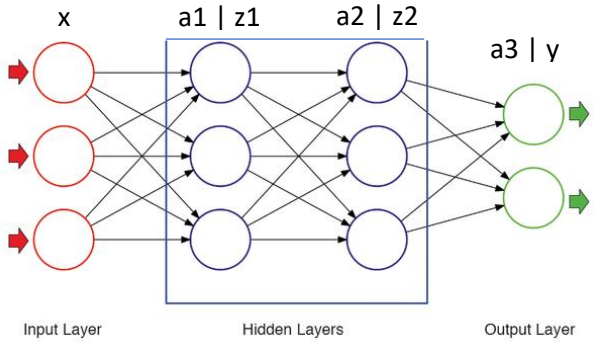
Implementing BackPropagation

Neural Network Model:

1. 2 Hidden layers with same number of nodes.
2. The input layer has 12 nodes as we did the PCA and reduced the dimension of the feature vector to 12.
3. The output layer has 2 nodes as we have to classify the images into 2 classes.
4. The activation function of hidden layers is g(a) = ReLU(a), and output has a SOFTMAX activation function. It gives the probability of input belonging to that particular class.



The weights and bias will have the following dimensions if the number of nodes in hidden layers is n

W1 = n x 12	b1 = n x 1
W2 = n x n	b2 = n x 1
W3 = 2 x n	b3 = 2 x 1

For updating, the weights and bias formulas used were according to the cross-entropy loss function, SOFTMAX function for output activation, and ReLU activation for hidden layers.

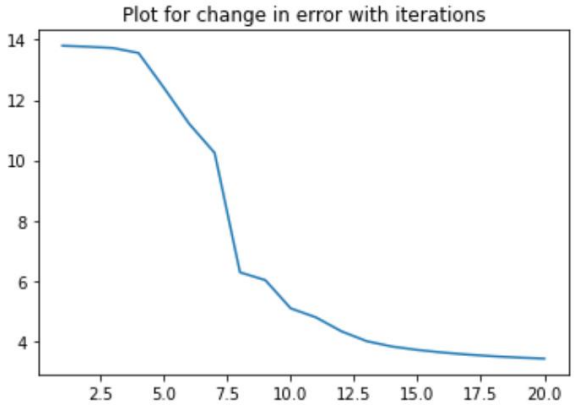
If L is the loss function we have,

1. $\delta L/ \delta a_3 = -(e-y)$ here, e is hot vector for known label and y is output of NN
2. $\delta L/ \delta W_3 = \delta L/ \delta a_3 \times \delta a_3/ \delta W_3 = \delta L/ \delta a_3 . z_2^T$
3. $\delta L/ \delta b_3 = \delta L/ \delta a_3$
4. $\delta L/ \delta z_2 = \delta L/ \delta a_3 \times \delta a_3/ \delta z_2 = W_3^T . \delta L/ \delta a_3$
5. $\delta L/ \delta a_2 = \delta L/ \delta z_2 \times \delta z_2/ \delta a_2$ = elementwise multiplication of $\delta L/ \delta z_2$ and $g'(a_2)$
6. $\delta L/ \delta W_2 = \delta L/ \delta a_2 \times \delta a_2/ \delta W_2 = \delta L/ \delta a_2 . z_1^T$
7. $\delta L/ \delta b_2 = \delta L/ \delta a_2$
8. $\delta L/ \delta z_1 = \delta L/ \delta a_2 \times \delta a_2/ \delta z_1 = W_2^T . \delta L/ \delta a_2$
9. $\delta L/ \delta a_1 = \delta L/ \delta z_1 \times \delta z_1/ \delta a_1$ = elementwise multiplication of $\delta L/ \delta z_1$ and $g'(a_1)$
10. $\delta L/ \delta W_1 = \delta L/ \delta a_1 \times \delta a_1/ \delta W_1 = \delta L/ \delta a_1 . x^T$
11. $\delta L/ \delta b_1 = \delta L/ \delta a_1$

If the number of nodes in the hidden layer is 10, if we update the weights and biases using the above formulas, we reduce cross entropy loss as shown in the plot below.

Observations:

a. Test accuracy of this case when the number of nodes was 10 fluctuated between 70, 80, and 90%. The fluctuation was due to random initial guesses.
b. When the number of nodes was increased from 10 to 15, the accuracy value fluctuated more between 80 and 90%, and the final loss was relatively less.



Error vs Iterations plot for nodes = 15 w/o momentum

c. When momentum term was used, which helps the network not converge at local minima, both the networks with 10 and 15 nodes reached 100% efficiency. The 15-node architecture had more frequency of getting 100% efficiency.
d. Adding more nodes and using momentum reduces the final converged error, which can be observed in the preview pdf and .ipynb files.
e. The learning rate value very much influenced the results as the number of iterations were fixed.

When number of nodes = 15 and momentum term is used, error reached close to 0

No of nodes used	Momentum	Final error
10	0	3.47705198
15	0	3.41911228
10	0.9	2.30842399
15	0.9	0.81090597

Final error for different conditions

Conclusion:

a. With the increase in the network's number of nodes, the accuracy of the network on test data increases, and final error further reduces.
b. With the momentum term, the network's accuracy on test data increases, and final error further reduces.