**S. B. JAIN INSTITUTE OF TECHNOLOGY,
MANAGEMENT & RESEARCH, NAGPUR.**

Practical No. 06

Aim: Develop and implement Back Propagation Neural Network to identify the error in the network. [XOR Problem]

Name of Student **: Shrutika Pradeep Bagdi**

Roll No **: CS22130**

Semester/Year **:** VIIth Sem / IVth Year

Academic Session **:** 2025-2026 [ODD]

Date of Performance : _____

Date of Submission : _____

AIM: Develop and implement Back Propagation Neural Network to identify the error in the network. [XOR Problem].

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To develop a learning algorithm for multilayer feedforward neural networks, empowering the networks to be trained to capture the mapping implicitly.
- To develop learning algorithm for multilayer feedforward neural network.
- Efficiently compute the gradient of the loss function with respect to the network weights.

HARDWARE AND SOFTWARE REQUIRMENTS:

Hardware Requirement: Computer System with high configurations

Software Requirement: Google Colab

THEORY:

Backpropagation is a process involved in training a neural network. It involves taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights.

Backpropagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration.) Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

When Do You Use Backpropagation in Neural Networks

We now have a model that does not give accurate predictions. It gave us the value four instead of one and that is attributed to the fact that its weights have not been tuned yet. They're all equal to one. We also have the loss, which is equal to -4. Backpropagation is all about feeding this loss backward in such a way that we can fine-tune the weights based on this. The optimization function, gradient descent in our example, will help us find the weights that will hopefully yield a smaller loss in the next iteration. So, let's get to it.

If feeding forward happened using the following functions: f(a) = a
Then feeding backward will happen through the partial derivatives of those functions. There is no need to go through the equation to arrive at these derivatives. All we need to know is that the above functions will follow:

f '(a) = a

J'(w) = Z . delta

Z is just the z value we obtained from the activation function calculations in the feed-forward step, while delta is the loss of the unit in the layer.

Updating the Weights in Backpropagation for a Neural Network

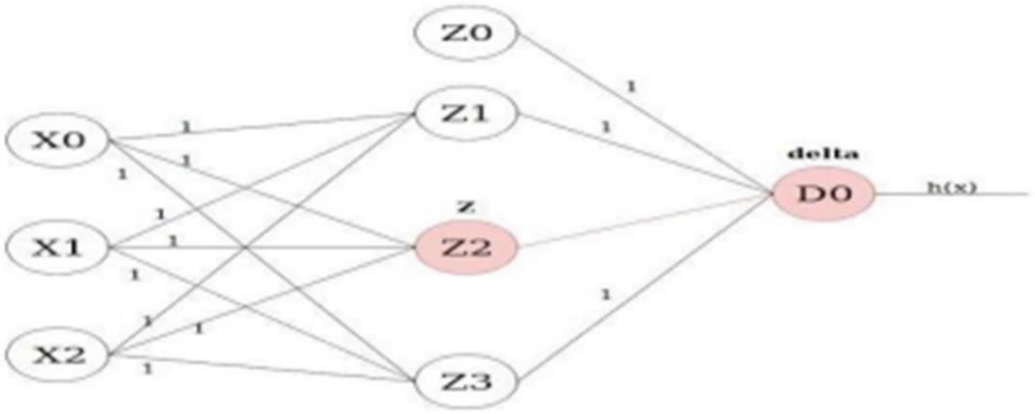
This follows the batch gradient descent formula:

W := W - alpha . J'(W)

Where W is the weight at hand, alpha is the learning rate (i.e. 0.1 in our example) and J'(W) is the partial derivative of the cost function J(W) with respect to W. Again, there's no need for us to get into the math. Therefore, let's use Mr. Andrew Ng's partial derivative of the function:

J'(W) = Z . delta

Where Z is the Z value obtained through forward propagation, and delta is the loss at the unit on the other end of the weighted link:



Now we use the batch gradient descent weight update on all the weights, utilizing our partial derivative values that we obtain at every step. It is worth emphasizing that the Z values of the input nodes (X0, X1, and X2) are equal to one, zero, zero, respectively.

The one is the value of the bias unit, while the zeroes are actually the feature input values coming from the data set. There is no particular order to updating the weights. You can update them in any order you want, as long as you don't make the mistake of updating any weight twice in the same iteration..

Best Practices for Optimizing Backpropagation

1. Select A Training Method

2. Provide Plenty of Data
3. Clean All Data
4. Consider The Impacts of Learning Rate
5. Test The Model with Different Examples

Advantages of backpropagation algorithm are as follows

1. It does not have any parameters to tune except for the number of inputs.
2. It is highly adaptable and efficient and does not require any prior knowledge about the network. 3. It is a standard process that usually works well.

Disadvantages of backpropagation algorithm are as follows

1. It prefers a matrix-based approach over a mini-batch approach.
2. Data mining is sensitive to noise and irregularities.
3. Performance is highly dependent on input data.

Applications of Naive Bayes:

Backpropagation is an algorithm that backpropagates the errors from the output nodes to the input nodes. Therefore, it is simply referred to as the backward propagation of errors. It uses in the vast applications of neural networks in data mining like Character recognition, Signature verification, etc.

1. The neural network is trained to enunciate each letter of a word and a sentence. 2. It is used in the field of speech recognition.
3. It is used in the field of character and face recognition.

PROGRAM CODE:

```
import numpy as np
# XOR input and output
X = np.array([[0,0],
[0,1],
[1,0],
[1,1]])
y = np.array([[0],
[1],
[1],
[0]])

# Sigmoid activation function
```

```
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

# Derivative of sigmoid
def sigmoid_derivative(x):
  return x * (1 - x)

# Network architecture
input_neurons = 2
hidden_neurons = 2
output_neurons = 1

# Initialize weights and biases
np.random.seed(42)
W1 = np.random.uniform(size=(input_neurons, hidden_neurons))
b1 = np.zeros((1, hidden_neurons))
W2 = np.random.uniform(size=(hidden_neurons, output_neurons))
b2 = np.zeros((1, output_neurons))

# Hyperparameters
lr = 0.5
epochs = 10000

# Training loop
for epoch in range(epochs):
  # Forward pass
  hidden_input = np.dot(X, W1) + b1
  hidden_output = sigmoid(hidden_input)
  final_input = np.dot(hidden_output, W2) + b2
  final_output = sigmoid(final_input)

  # Error
  error = y - final_output

  # Backward pass
  d_output = error * sigmoid_derivative(final_output)
  d_hidden = d_output.dot(W2.T) * sigmoid_derivative(hidden_output)

  # Update weights
  W2 += hidden_output.T.dot(d_output) * lr
  b2 += np.sum(d_output, axis=0, keepdims=True) * lr
  W1 += X.T.dot(d_hidden) * lr
  b1 += np.sum(d_hidden, axis=0, keepdims=True) * lr

  # Print loss occasionally
  if (epoch+1) % 2000 == 0:
    loss = np.mean(np.square(error))
```

```
    print(f"Epoch {epoch+1}, Loss: {loss:.4f}")

# Predictions
print("\nFinal Predictions: ")
hidden_input = np.dot(X, W1) + b1
hidden_output = sigmoid(hidden_input)
final_input = np.dot(hidden_output, W2) + b2
final_output = sigmoid(final_input)
print(final_output.round(3))
```

OUTPUT (SCREENSHOT):

```
# Predictions
print("\nFinal Predictions: ")
hidden_input = np.dot(X, W1) + b1
hidden_output = sigmoid(hidden_input)
final_input = np.dot(hidden_output, W2) + b2
final_output = sigmoid(final_input)
print(final_output.round(3))
```

```
Epoch 2000, Loss: 0.0028
Epoch 4000, Loss: 0.0009
Epoch 6000, Loss: 0.0006
Epoch 8000, Loss: 0.0004
Epoch 10000, Loss: 0.0003

Final Predictions:
[[0.019]
 [0.984]
 [0.984]
 [0.017]]
```

CONCLUSION:

DISCUSSION AND VIVA VOCE:

1. What is backpropagation?

2. Why do we need backpropagation in a neural network?
3. What are the main differences between AI, Machine Learning, and Deep Learning?
4. What is the use of the Activation function?
5. How many types of activation function are available?

REFERENCE:

- https://builtin.com/machine-learning/backpropagation-neural-network
- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
- https://www.javatpoint.com/deep-learning-interview-questions