ML

(21)

Name: Bakshi N. Patel
Sapid: 60004210126
Branch: Computer Engineering
T.Y.B-Tech, Sem VI - C2-1

Experiment no. 5

Aim: To implement Backpropagation algorithm.

Theory:

A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduit image understanding, human learning, computer speech, etc.

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

Backpropagation in a neural network is a short form for "Backward propagation of error". It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network. The back propagation algorithm in neural network computes the gradient of loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient but it does not define how the gradient is used. It generalizes the computation in the delta rule. In an artificial neural network, we have three different types of layers:-

P.T.O.

1. Input layer
2. Hidden layer (can be one or many)
3. output layer

Each layer has its own way of working & its own way to act such that we are able to get the desired results and correlate these scenarios to our conditions.

Steps in Backpropagation Algorithm

1. Create a feed-forward network with n_i inputs, n hidden units, & n_o output units.
2. Initialize all network weights to small random numbers
3. Until the termination condition is met, Do.
4. For each (x, t) in training examples, DO

propagate the input forward through the network:

1. Input the instance x, to the network & compute the output o_u of every unit u in the network.

propagate the error backward through the network.

2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k (1 - o_k)(t_k - o_k)$$

3. For each network unit h, calculate its error term δ_h

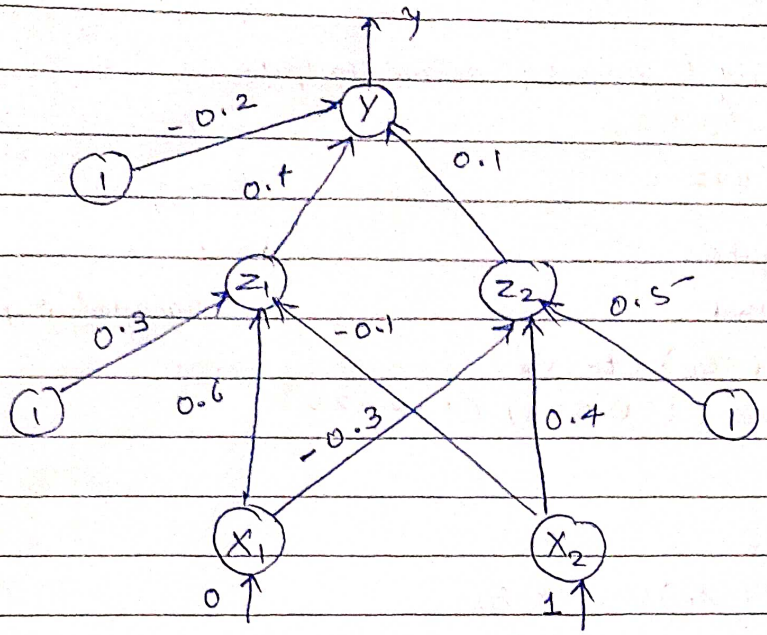
$$\delta_h \leftarrow o_h (1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$

4. update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where $$\Delta w_{ji} = \eta \delta_j . x_{ji}$$

Problem :



Solution :

Step 1] Forward propagation :

$Z_1 = F(x_1 \times w_{13} + x_2 w_{23} + 0.3)$

$= F(0 \times 0.6 + 1 \times (-0.1) + 0.3)$

$= F(0.2)$

$= 0.5498$

≈ 0.55

Assuming activation function,

$F(x) = \frac{1}{1 + e^{-x}}$

$Z_2 = F(x_1 w_{14} + x_2 w_{24} + 0.5)$

$= F(0 \times 0.3 + 0.4 + 0.5)$

$= F(0.9)$

$= 0.71$

$y = F(Z_1 w_{35} + Z_2 w_{45} + (-0.2))$

$= F(0.4 \times 0.55 + 0.1 \times 0.71 + (-0.2))$

$= F(0.091)$

$= 0.523$

$= O_k$

t_R is expected to be 1

Error = target output − actual output

= 1 − 0.523

= 0.477

Step 3] Back propagation :

We know that, targeted output is expected to 1

$\delta_k = O_k (1 - O_k)(t_k - O_k)$

$= 0.523 (1 - 0.523)(1 - 0.523)$

$\delta_k = 0.38$

Hidden layer,

$\delta_3 = z_1 (1 - z_1) w_{35} \times \delta_k$

$= 0.55 (1 - 0.55) \times 0.4 \times 0.38$

$= 0.0376$

$\delta_3 \approx 0.038$

$\delta_4 = z_2 (1 - z_2) w_{45} \times \delta_5$

$= 0.71 \times 0.29 \times 0.1 \times 0.38$

$\delta_4 = 0.0078$

i	j	w_{ij}	δ_j	x_i	η	updated weights
1	3	0.6	0.038	0	0.1	0.6
2	3	-0.1	0.038	1	0.1	-0.0962
1	4	-0.3	0.0078	0	0.1	-0.3
2	4	0.4	0.0078	1	0.1	0.4008
3	5	0.4	0.38	0.55	0.1	0.421
4	5	0.1	0.38	0.71	0.1	0.1269

We know that,

$\Delta w_{ij} = \eta \delta_j w_i$

$\therefore w_{13}$ (new) $= \eta \delta_j x_1 + w_{13} = 0.6$

$w_{23} = 0.1 \times 0.038 \times 1 + w_{23} = -0.0962$

PTO

$w_{14} = w_{14} + \eta \delta_j x_i = -0.3$

$w_{24} = w_{24} + \eta \delta_j x_i = 0.4008$

$w_{35} = w_{35} + \eta \delta_j x_i = 0.42$

$w_{45} = w_{45} + \eta \delta_j x_i = 0.127$

∴ Final weights after 1 epoch =

$w_{13} = 0.6$

$w_{23} = -0.962$

$w_{14} = -0.3$

$w_{24} = 0.4008$

$w_{35} = 0.42$

$w_{45} = 0.127$

Conclusion :-

Thus we have implemented back propagation on an Artificial neural network with ~~to~~ one (1) hidden layer.

m2
12/3/24

Name:-Preksha Patel
Sapid:-60004210126
Branch:-Computer Engineering
Div:-C2,Batch:-1

MACHINE LEARNING
EXPERIMENT-5

```
import numpy as np

class NeuralNetwork:
  def __init__(self, input_size, hidden_size, output_size, learning_rate):
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.output_size = output_size
    self.learning_rate = learning_rate

   #initialize wieghts and outputs
    self.w1 = np.array([[0.6, -0.1], [-0.3, 0.4]])  # Weights for x1 and x2 to z1 and z2
    self.w2 = np.array([[0.4, 0.1]])  # Weights for z1 and z2 to Y
    self.b1 = np.array([[0.3], [0.5]] )  # Biases for z1 and z2
    self.b2 = np.array([[-0.2]])

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

  def sigmoid_derivative(self, x):
    return self.sigmoid(x) * (1 - self.sigmoid(x))

  def forward(self, x):
    # Forward pass
    hidden_layer_activation = self.sigmoid(np.dot(self.w1, x))
    output = self.sigmoid(np.dot(self.w2, hidden_layer_activation))
    return hidden_layer_activation, output

  def backward(self, x, y, hidden_layer_activation, output):
    # Backward pass
    output_error = y - output
    delta_output = output_error * self.sigmoid_derivative(output)

    hidden_layer_error = np.dot(self.w2.T, delta_output)
    delta_hidden = hidden_layer_error * self.sigmoid_derivative(hidden_layer_activation)

    # Update weights
    self.w2 += self.learning_rate * np.outer(delta_output, hidden_layer_activation)
    self.w1 += self.learning_rate * np.outer(delta_hidden, x)
```

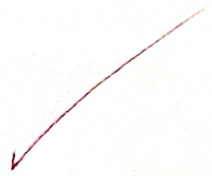
```
    def train(self, data, epochs):
        for epoch in range(epochs):
            for x, y in data:
                hidden_layer_activation, output = self.forward(x)
                self.backward(x, y, hidden_layer_activation, output)


                print(f"Epoch: {epoch + 1}")
                print(f"Hidden Layer Output: {hidden_layer_activation}")
                print(f"Output: {output}")
                print(f"Error: {y - output}")



nn = NeuralNetwork(2, 2, 1, 0.1)  # 2 inputs, 4 hidden nodes, 1 output, learning rate of 0.1
data = [(np.array([0, 1]), np.array([1]))]

nn.train(data, 10)
```



```
Epoch: 1
Hidden Layer Output: [0.47502081 0.59868766]
Output: [0.56214625]
Error: [0.43785375]
Epoch: 2
Hidden Layer Output: [0.47525959 0.59874334]
Output: [0.56362655]
Error: [0.43637345]
Epoch: 3
Hidden Layer Output: [0.47550032 0.59880216]
Output: [0.56510159]
Error: [0.43489841]
Epoch: 4
Hidden Layer Output: [0.47574298 0.59886409]
Output: [0.5665714]
Error: [0.4334286]
Epoch: 5
Hidden Layer Output: [0.47598754 0.5989291 ]
Output: [0.56803599]
Error: [0.43196401]
Epoch: 6
Hidden Layer Output: [0.47623397 0.59899714]
Output: [0.5694954]
Error: [0.4305046]
Epoch: 7
Hidden Layer Output: [0.47648225 0.59906818]
Output: [0.57094964]
Error: [0.42905036]
Epoch: 8
Hidden Layer Output: [0.47673235 0.59914219]
Output: [0.57239874]
Error: [0.42760126]
Epoch: 9
Hidden Layer Output: [0.47698424 0.59921912]
Output: [0.57384271]
Error: [0.42615729]
Epoch: 10
Hidden Layer Output: [0.4772379  0.59929895]
Output: [0.5752816]
Error: [0.4247184]
```

```
import matplotlib.pyplot as plt

epochs = range(1, 10)
errors = []

for epoch in epochs:
  error = 0
  for x, y in data:
    _, output = nn.forward(x)
    error += (y - output)**2
  error /= len(data)
  errors.append(error)

plt.plot(epochs, errors)
plt.xlabel("Epoch")
plt.ylabel("Error")
plt.show()
```