8/9/25

Week-6

Aim: Implementing gradient descent and back propagation in deep neural Network.

Algorithm:

1) Initialize neural network parameters: weights and biases (random small values)

2) perform the forward pass:
   - → compute weighted sum of inputs at each layer.
   - → Apply Activation functions to compute outputs.

3) Compute the loss using a suitable loss function (mean squared error (or) cross-Entropy).

4) perform the backward pass:
   - → compute gradients of loss w.r.to weights and biases using the chain rule.
   - → propagate errors from output layer to hidden layers.

5) updates weights using gradient descent rule:

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial w}$$

6) Repeat steps 3-6 for multiple epochs until the convergence.

Pseudo code:

```
Begin
  Initialize weights and biases randomly
  Choose learning rate η
  For epoch in range (1, max_epochs):
     For each input-output pair (x,y):
        z1 = w1 * x + b1
        a1 = activation (z1)
        z2 = w2 * a1 + b2
```

a_2 = activation (z_2)

y-pred = final-output

L = loss-function (y, y-pred)

compute dL/dy-pred

compute gradients of each layer using chain rule.

update weights :

$w = w - \eta * \text{gradient}$

update biases :

$b = b - \eta * \text{gradient}$

End.

observations :-

*) Gradient descent gradually reduces the error function by adjusting weights.

*) Small learning rate causes slow convergence.

*) Backpropagation efficiently computes the weight updates using the chain rule.

*) Training error decreases with epochs when gradient descent is applied correctly.

output :-

Iteration 0, Loss : 0.2558

Iteration 1000, Loss : 0.2494

Iteration 2000, Loss : 0.24574

Iteration 3000, Loss : 0.20470

Iteration 4000, Loss : 0.15320

⋮

Iteration 9000, Loss : 0.12884

Training complete!

Final output :

[[0.5300868]
 [0.49554213]
 [0.95091319]
 [0.50 319888]]

```python
import numpy as np

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

def sigmoid_derivative(x):
    return x * (1 - x)

inputs = np.array([[0,0], [0,1], [1,0], [1,1]])
labels = np.array([[0], [1], [1], [0]])

np.random.seed(42)
weights_input_hidden = np.random.randn(2, 2)
bias_hidden = np.zeros((1, 2))

weights_hidden_output = np.random.randn(2, 1)
bias_output = np.zeros((1, 1))

learning_rate = 0.1

for iteration in range(10000):
    hidden_input = np.dot(inputs, weights_input_hidden) + bias_hidden
    hidden_output = sigmoid(hidden_input)

    final_input = np.dot(hidden_output, weights_hidden_output) + bias_output
    final_output = sigmoid(final_input)

    loss = np.mean((final_output - labels) ** 2)

    error_output = final_output - labels
    delta_output = error_output * sigmoid_derivative(final_output)

    error_hidden = np.dot(delta_output, weights_hidden_output.T)
    delta_hidden = error_hidden * sigmoid_derivative(hidden_output)

    gradient_weights_hidden_output = np.dot(hidden_output.T, delta_output)
    gradient_bias_output = np.sum(delta_output, axis=0, keepdims=True)

    gradient_weights_input_hidden = np.dot(inputs.T, delta_hidden)
    gradient_bias_hidden = np.sum(delta_hidden, axis=0, keepdims=True)

    weights_hidden_output -= learning_rate * gradient_weights_hidden_output
    bias_output -= learning_rate * gradient_bias_output

    weights_input_hidden -= learning_rate * gradient_weights_input_hidden
    bias_hidden -= learning_rate * gradient_bias_hidden

    if iteration % 1000 == 0:
        print(f"Iteration {iteration}, Loss: {loss}")

print("Training complete!")
print("Final output:")
print(final_output)
```

```
Iteration 0, Loss: 0.2558299419444368
Iteration 1000, Loss: 0.24940565956551236
Iteration 2000, Loss: 0.24544465159719808
Iteration 3000, Loss: 0.2047073304071442
Iteration 4000, Loss: 0.15320405369970766
Iteration 5000, Loss: 0.13869146014771938
Iteration 6000, Loss: 0.13359363321851758
Iteration 7000, Loss: 0.13115112181268004
Iteration 8000, Loss: 0.12974916048385188
Iteration 9000, Loss: 0.12884908965171127
Training complete!
Final output:
[[0.05300868]
 [0.49554213]
 [0.95091319]
 [0.50319888]]
```