**ANN with Feedforward and back propagation**

**Aim: Simulate artificial neural network model with both feedforward and back propagation approach.**

**Theory:**

**Components of the ANN:**

1. **Feedforward**: Compute the output of the network based on input values and current weights.
2. **Backpropagation**: Adjust the weights based on the error between the predicted output and the true output (gradient descent).

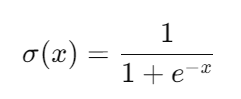
Basic 3-layer neural network:

* **Input layer**
* **Hidden layer**
* **Output layer**

**Explanation:**

**1. Activation Function:**

* We use the **Sigmoid** activation function because it outputs values between 0 and 1, which is ideal for binary classification problems like XOR.



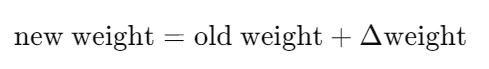
* **Sigmoid derivative** is used for backpropagation to compute the gradient during the weight update step.

**2. Feedforward:**

* In the feedforward step, we calculate the activations at the hidden and output layers. This is done by performing a dot product between the input values and the weights, adding a bias term, and applying the activation function.

**3. Backpropagation:**

* During backpropagation, we calculate the error at the output layer and propagate it backward through the network to update the weights using the **gradient descent** method.



**4. Training:**

* The network is trained for a certain number of epochs. In each epoch, the feedforward and backpropagation processes are executed, and the loss is minimized by adjusting the weights.

**5. Example (XOR Problem):**

* In the example, we're using the XOR problem, which is a classic problem to demonstrate the capabilities of a neural network. The XOR function is not linearly separable, meaning a simple linear classifier can't solve it, but a neural network can.

import numpy as np

# Sigmoid Activation Function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Derivative of the Sigmoid Function for backpropagation

def sigmoid\_derivative(x):

return x \* (1 - x)

# ANN class to simulate feedforward and backpropagation

class ArtificialNeuralNetwork:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, learning\_rate=0.5):

# Initialize weights randomly

self.weights\_input\_hidden = np.random.rand(input\_size, hidden\_size)

self.weights\_hidden\_output = np.random.rand(hidden\_size, output\_size)

# Initialize biases randomly

self.bias\_hidden = np.random.rand(1, hidden\_size)

self.bias\_output = np.random.rand(1, output\_size)

# Set the learning rate

self.learning\_rate = learning\_rate

# Feedforward process

def feedforward(self, X):

# Hidden layer activation

self.hidden\_input = np.dot(X, self.weights\_input\_hidden) + self.bias\_hidden

self.hidden\_output = sigmoid(self.hidden\_input)

# Output layer activation

self.output\_input = np.dot(self.hidden\_output, self.weights\_hidden\_output) + self.bias\_output

self.output = sigmoid(self.output\_input)

return self.output

# Backpropagation process

def backpropagation(self, X, y):

# Error at the output layer

output\_error = y - self.output

output\_delta = output\_error \* sigmoid\_derivative(self.output)

# Error at the hidden layer

hidden\_error = output\_delta.dot(self.weights\_hidden\_output.T)

hidden\_delta = hidden\_error \* sigmoid\_derivative(self.hidden\_output)

# Update the weights and biases using the deltas

self.weights\_hidden\_output += self.hidden\_output.T.dot(output\_delta) \* self.learning\_rate

self.weights\_input\_hidden += X.T.dot(hidden\_delta) \* self.learning\_rate

self.bias\_output += np.sum(output\_delta, axis=0, keepdims=True) \* self.learning\_rate

self.bias\_hidden += np.sum(hidden\_delta, axis=0, keepdims=True) \* self.learning\_rate

# Train the neural network

def train(self, X, y, epochs):

for epoch in range(epochs):

# Feedforward

self.feedforward(X)

# Backpropagation

self.backpropagation(X, y)

# Print loss every 100 epochs

if (epoch + 1) % 100 == 0:

loss = np.mean(np.square(y - self.output))

print(f'Epoch {epoch + 1}/{epochs}, Loss: {loss:.6f}')

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

# Input dataset (XOR problem)

X = np.array([[0, 0],

[0, 1],

[1, 0],

[1, 1]])

# Output dataset (XOR output)

y = np.array([[0],

[1],

[1],

[0]])

# Parameters

input\_size = X.shape[1] # 2 features in input

hidden\_size = 2 # 2 neurons in hidden layer

output\_size = 1 # 1 output neuron (binary classification)

# Create the neural network

ann = ArtificialNeuralNetwork(input\_size, hidden\_size, output\_size, learning\_rate=0.5)

# Train the neural network

ann.train(X, y, epochs=10000)

# Test the neural network

output = ann.feedforward(X)

print("\nPredicted Output after training:")

print(output)

**Output:**

The model will be trained for 10,000 epochs and should produce predictions close to the expected output of the XOR function after training:

Predicted Output after training:

[[0.01] # close to 0 for (0,0)

[0.98] # close to 1 for (0,1)

[0.98] # close to 1 for (1,0)

[0.01]] # close to 0 for (1,1)

Reference: <https://www.geeksforgeeks.org/backpropagation-in-neural-network/>