Ayush Chanchal

Sap:500097569

Batch:B6

EXPERIMENT:01

Simple Neural Network:

A simple neural network, also known as a feedforward neural network, consists of three main types of layers: an input layer, one or more hidden layers, and an output layer.

Steps:

Input Layer:

The input layer consists of nodes (neurons) that represent the features of the input data. Each node corresponds to one feature.

Hidden Layer:

The hidden layer(s) are between the input and output layers. They perform computations on the input data and pass the results to the output layer. Each node in the hidden layer is connected to every node in the previous layer.

Output Layer:

The output layer produces the final result of the neural network's computation. The number of nodes in the output layer depends on the nature of the problem (e.g., binary classification, multi-class classification, regression, etc.).

Connections (Weights):

Each connection between nodes has an associated weight. These weights are learned during the training process and represent the strength of the connection.

Activation Functions:

Each node (except the input nodes) has an activation function that applies a non-linear transformation to the weighted sum of its inputs. Common activation functions include sigmoid, tanh, ReLU, etc.

Bias:

Each node (except the input nodes) also has a bias term. The bias helps in shifting the activation function and allows the network to learn more complex relationships.

Forward Pass:

During the forward pass, the input data is fed through the network, and computations are performed layer by layer until the output is obtained.

Backpropagation:

After the forward pass, the network's output is compared to the true target values. The error (the difference between the predicted and actual values) is then propagated backward through the network using an optimization algorithm (e.g., gradient descent) to update the weights and biases.

Training:

The process of adjusting the weights and biases based on the error is called training. This is done over multiple iterations (epochs) until the model's performance improves.

Prediction:

Once the network is trained, it can be used to make predictions on new, unseen data by performing a forward pass with the new input.

Q). Implementation of Simple neural network using Activation Functions.

Sourcecode:

import numpy as np

class NeuralNetwork():

def \_\_init\_\_(self):

np.random.seed(1)

self.synaptic\_weights = 2\*np.random.random((3, 1))-1

def sigmoid(self, x):

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

def sigmoid\_derivative(self, x):

return x\*(1-x)

def train(self, training\_inputs, training\_outputs, training\_iterations):

for iteration in range(training\_iterations):

output = self.think(training\_inputs)

error = training\_outputs-output

adjustments = np.dot(training\_inputs.T, error \*

self.sigmoid\_derivative(output))

self.synaptic\_weights += adjustments

def think(self, inputs):

inputs = inputs.astype(float)

output = self.sigmoid(np.dot(inputs, self.synaptic\_weights))

return output

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

neural\_network = NeuralNetwork()

print("Beginning Randomly generated weights:")

print(neural\_network.synaptic\_weights)

training\_inputs = np.array(

[[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])

training\_outputs = np.array([[0, 1, 1,0]]).T

neural\_network.train(training\_inputs, training\_outputs, 15000)

print("Ending weights after training")

print(neural\_network.synaptic\_weights)

user\_input\_one = str(input("user input one:"))

user\_input\_two = str(input("user input two:"))

user\_input\_three = str(input("user input three:"))

print("Considering new situation:", user\_input\_one,

user\_input\_two, user\_input\_three)

print("New output data:")

print(neural\_network.think(

np.array([user\_input\_one, user\_input\_two, user\_input\_three])))

print('Yes,we did it!!!!')

Screenshot:

