Experiment No: 07

Aim: Use FCNN with only one neuron and plotting FCNN with one hidden layer and plotting.

Theory:

FCNN stands for **Fully Connected Neural Network**, and it is a type of artificial neural network that is characterized by its dense or fully connected layers. FCNNs are also commonly referred to as feedforward neural networks or multilayer perceptrons (MLPs).

Key components FCNNs:

Neurons/Nodes: FCNNs consist of layers of interconnected nodes, where each node is often called a "neuron." Neurons in a layer are fully connected to neurons in the adjacent layers, which means that each neuron in one layer is connected to every neuron in the next layer.

Layers: An FCNN typically consists of an input layer, one or more hidden layers, and an output layer. The input layer takes the raw input data, and the output layer produces the network's predictions. The hidden layers are intermediate layers that transform the input data as it passes through the network.

Weights and Biases: Each connection between neurons is associated with a weight, which represents the strength of the connection. The neurons also have biases that can be adjusted. During training, the network learns the optimal values for these weights and biases to make accurate predictions.

Activation Functions: Each neuron applies an activation function to the weighted sum of its inputs. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), tanh, and more. These functions introduce non-linearity to the model, allowing it to capture complex relationships in the data.

Forward Propagation: During forward propagation, data is passed through the network layer by layer. Each neuron in a layer receives input from all neurons in the previous layer, applies its activation function, and passes the result to the next layer. This process continues until the output layer produces the final prediction.

Loss/Cost Function: To measure how well the network's predictions match the target values, a loss or cost function is used. This function quantifies the error between predictions and actual values. The goal during training is to minimize this loss.

Backpropagation: Backpropagation is the process of updating the weights and biases in the network to reduce the loss. It works by calculating the gradients of the loss with respect to each weight and bias, and then using gradient descent or a similar optimization algorithm to update these parameters.

Training: Training an FCNN involves presenting the network with a dataset, forwarding the data through the network, computing the loss, and then using backpropagation to update the weights and biases. This process is typically repeated for multiple epochs until the network converges to a satisfactory solution.

Architecture:

Input Layer: The input layer of an FCNN receives raw data. Each neuron in the input layer corresponds to a feature or input dimension.

Hidden Layers: These are one or more layers of neurons situated between the input and output layers. The term "hidden" implies that these layers do not interact directly with the external world. The number of hidden layers and the number of neurons in each layer are configurable hyperparameters.

Output Layer: The output layer produces the final predictions or results. The number of neurons in this layer depends on the specific problem. For binary classification, it may have one neuron with a sigmoid activation function, while for multiclass classification, it may have multiple neurons with softmax activation.

Weights and Biases:

Weights: Each connection between neurons has an associated weight, which determines the strength of the connection. During training, these weights are adjusted to optimize the network's performance.

Biases: Neurons also have biases, which allow the network to shift the activation function. Biases are learned during training.

Activation Functions:

Activation functions introduce non-linearity into the network. This non-linearity is crucial for enabling the network to model complex relationships in the data. Common activation functions include:

Sigmoid: Outputs values in the range (0, 1).

Hyperbolic Tangent (tanh): Outputs values in the range (-1, 1).

Rectified Linear Unit (ReLU): Outputs the input for positive values and zero for negative values. ReLU is widely used in hidden layers due to its computational efficiency.

Forward Propagation:

During forward propagation, the input data is passed through the network layer by layer. Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to the next layer.

The final output of the network is produced by the output layer and represents the model's prediction.

Loss Function:

The loss or cost function quantifies the error between the model's predictions and the actual target values in the training data. The choice of loss function depends on the type of task (e.g., mean squared error for regression, cross-entropy for classification).

The goal during training is to minimize this loss function.

Backpropagation:

Backpropagation is the process of updating the network's weights and biases to minimize the loss. It involves computing the gradients of the loss with respect to each weight and bias in the network.

Optimization algorithms, such as gradient descent, are used to adjust the parameters based on these gradients.

Training:

Training an FCNN involves presenting a labeled dataset to the network, performing forward and backward passes (forward propagation and backpropagation), and updating the parameters iteratively.

Training is typically performed for multiple epochs (passes through the entire dataset) until the model converges to a satisfactory solution.

Regularization and Optimization:

To prevent overfitting, techniques like dropout, weight decay (L1 and L2 regularization), and early stopping can be applied.

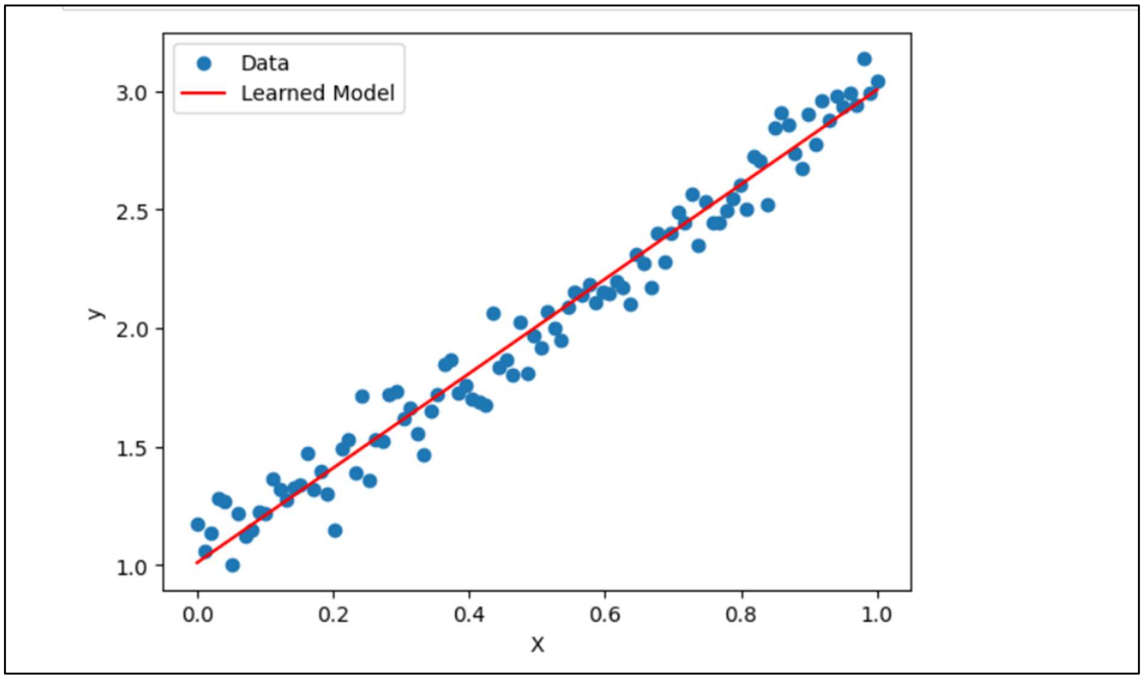
Optimization algorithms can include variations of gradient descent like stochastic gradient descent (SGD), Adam, RMSprop, etc.

FCNNs are powerful and flexible, but their performance depends on factors like architecture, hyperparameters, and the quality and quantity of training data. They have been instrumental in solving a wide range of machine learning and deep learning tasks, making them a fundamental tool in the field of artificial intelligence.

Implementation:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate some example data
np.random.seed(0)
X = np.linspace(0, 1, 100)
y = 2 * X + 1 + 0.1 * np.random.randn(100)  # Linear function with noise

# Define the FCNN model
class SimpleFCNN:
    def __init__(self):
        self.weights = np.random.randn(2)  # Weights for the input and bias
        self.learning_rate = 0.01

    def predict(self, x):
        return np.dot(x, self.weights)

    def train(self, x, y):
        y_pred = self.predict(x)
        error = y_pred - y
        gradient = np.dot(x, error)
        self.weights -= self.learning_rate * gradient

# Training the model
model = SimpleFCNN()
for epoch in range(1000):
    for xi, yi in zip(X, y):
        model.train(np.array([xi, 1]), yi)

# Making predictions
y_pred = [model.predict(np.array([xi, 1])) for xi in X]

# Plot the data and the learned model
plt.scatter(X, y, label='Data')
plt.plot(X, y_pred, color='red', label='Learned Model')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

Output:



Conclusion:

A Fully Connected Neural Network (FCNN) with only one neuron is essentially a linear regression model, while an FCNN with one hidden layer can capture more complex patterns. In a simple example, the one-neuron FCNN is limited to modeling linear relationships, whereas a one-hidden-layer FCNN can capture non-linear patterns, making it more versatile for various tasks.