Experiment No. 1

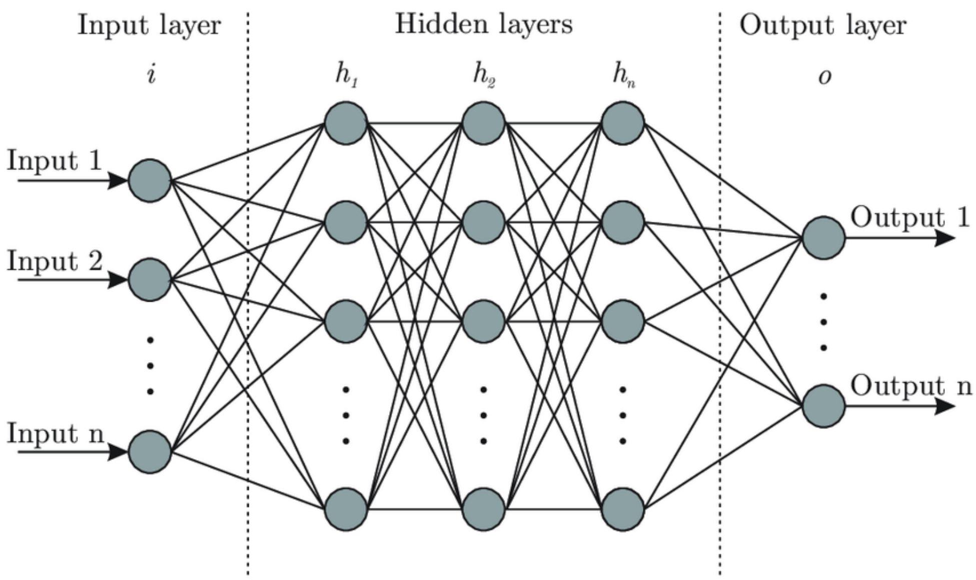
Title - Implement an Artificial Neural Network

Theory -

An Artificial Neural Network (ANN) is a computational model inspired by the structure and function of the human brain. It's designed to learn and recognize complex patterns in data, making it a fundamental component of machine learning and artificial intelligence.

Basic Structure

- ***Neurons (or Nodes)***: Mimic the function of biological neurons in the brain. They receive inputs, perform computations, and generate outputs.
- ***Layers***: Neurons are organised in layers:
  - Input Layer: Receives the initial data features.
  - Hidden Layers: Intermediate layers between the input and output; they process and transform information.
  - Output Layer: Produces the final output, such as predictions or classifications.



How It Works

1. Input Data: The neural network takes input data represented as numerical features.
2. Forward Propagation: The input data is passed through the network layer by layer.

- Each neuron in a layer receives inputs, performs a computation (weighted sum and activation), and passes the output to the next layer.
- Activation functions (like sigmoid, ReLU, etc.) introduce non-linearity, enabling the network to model complex relationships.

3. Output Prediction: The final layer produces the network's output, such as classification probabilities or numerical predictions.

Learning Process

- Training: The network learns by adjusting its weights and biases during the training phase to minimise the difference between predicted and actual outputs.
- Back-propagation: It's a key algorithm for training neural networks. It calculates the gradient of the error and adjusts the network's parameters (weights and biases) by propagating the error backward through the network.

Key Concepts and Components

- Weights and Biases: Parameters that the network learns and adjusts during training.
- Activation Function: Adds non-linearity to the network, allowing it to model complex relationships in data.
- Loss/Cost Function: Measures the difference between predicted and actual outputs.
- Optimisation Algorithms: Like gradient descent, used to update weights and biases to minimise the loss function.

Implementation

```
import numpy as np
import pandas as pd

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

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

def initialize_weights():
    weights = np.random.randn(1, 1)
    bias = np.random.randn(1)
    return weights, bias

def forward_propagation(X, weights, bias):
    weighted_sum = np.dot(X, weights) + bias
    output = sigmoid(weighted_sum)
    return output

def backward_propagation(X, y_true, output, weights, bias, learning_rate):
    error = y_true - output
    d_output = error * sigmoid_derivative(output)
    d_weights = np.dot(X.T, d_output)
    d_bias = np.sum(d_output)
    weights += learning_rate * d_weights
    bias += learning_rate * d_bias
    return weights, bias

def train(X, y_true, learning_rate, epochs):
    weights, bias = initialize_weights()
    for epoch in range(epochs):
        output = forward_propagation(X, weights, bias)
        weights, bias = backward_propagation(X, y_true, output, weights, bias, learning_rate)
        if epoch % 100 == 0:
            loss = np.mean(np.square(y_true - output))
            print(f'Epoch: {epoch}, Loss: {loss:.4f}')
    return weights, bias
```

```
def predict(X, weights, bias):
    return forward_propagation(X, weights, bias)

data = pd.read_csv('data.csv')
height = np.array(data.iloc[:, 0].values).reshape(-1, 1)
weight = np.array(data.iloc[:, 1].values).reshape(-1, 1)

# Normalize data
normalized_height = height * (1 / np.max(height))
normalized_weight = weight * (1 / np.max(weight))

# Train the neural network
trained_weights, trained_bias = train(normalized_height, normalized_weight, learning_rate=0.1, epochs=10000)

# Predict weight for a new height value
row = 7
n_height = np.array(data.iloc[row, 0])
n_weight = np.array(data.iloc[row, 1])
normalized_n_height = n_height * (1 / np.max(height))
predicted_weight = predict(normalized_n_height, trained_weights, trained_bias) * np.max(weight)
print(f'Predicted weight for height {n_height} cm: {predicted_weight[0][0]:.2f} kg')
print(f'True weight for height {n_height} cm: {n_weight:.2f} kg')
```

Output :

```
Epoch: 0, Loss: 0.0138
Epoch: 100, Loss: 0.0705
...
Epoch: 5500, Loss: 0.0705
...
Epoch: 9800, Loss: 0.0705
Epoch: 9900, Loss: 0.0705
```

```
Predicted weight for height 177.83739 cm: 77.53 kg
True weight for height 177.83739 cm: 61.90 kg
```

Applications

- Pattern Recognition: Image and speech recognition, object detection, and natural language processing.
- Prediction and Forecasting: Predictive analytics in various fields, including finance, healthcare, and weather forecasting.
- Decision-Making: Recommender systems, autonomous vehicles, and gaming AI.

Conclusion

Performing an ANN experiment involves a structured process of data preparation, model training, evaluation, prediction, and iterative refinement, ultimately aiming to develop an effective and accurate predictive model for the given problem domain.