Experiment No. 9

Expt No. 9	**Implementation of a Neural Network**
Date :	

Aim:

To implement and understand the working of a simple neural network for binary classification focusing on its structure, training process and evaluation.

System Requirements:

- Operating System: Windows 8 or above / Linux / macOS
- Memory (RAM): Minimum 4 GB
- Processor: Minimum 2.33 GHz (Dual Core or higher)

Software/Tools Required:

Jupyter Notebook/ Anaconda Navigator/ Google Colaboratory/Spyder, Python 3.x [With libraries such as Numpy, Pandas, Matplotlib, Scikit-Learn and Keras]

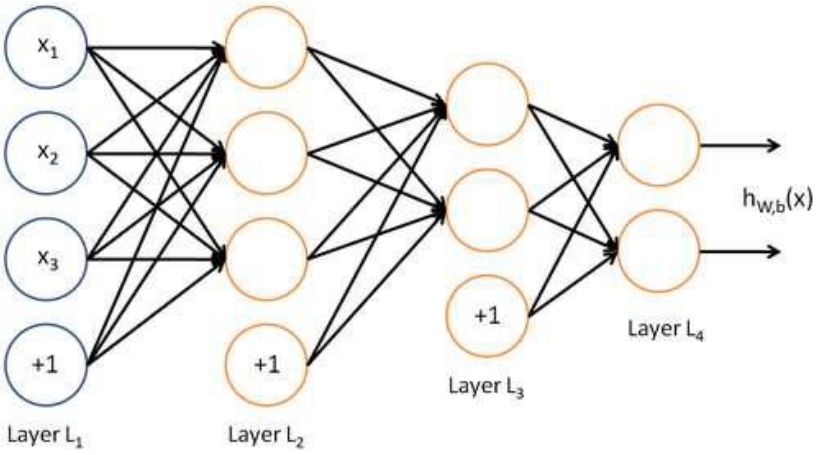
Expected Outcomes:

- Understand the structure of Neural Networks
- To choose appropriate activation functions
- To train and evaluate a Neural Network
- To apply Neural Networks to real-world applications.

Theory:

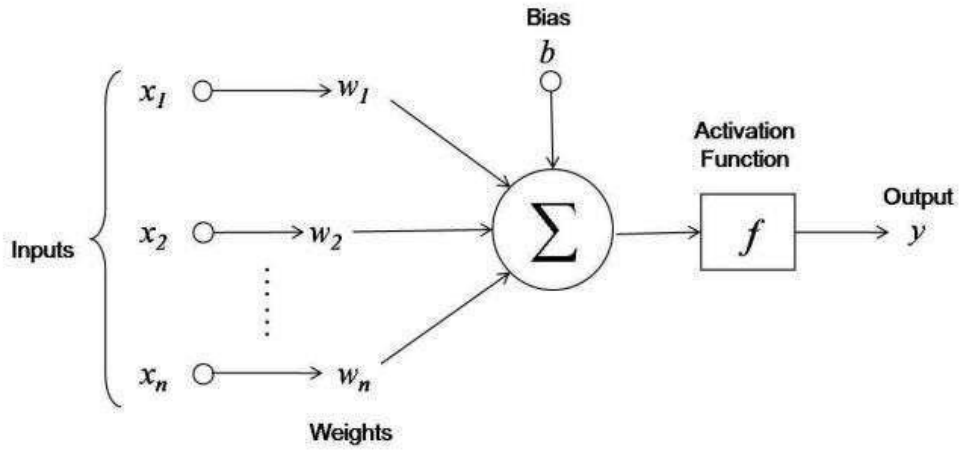
A Neural Network is a computational model inspired by the human brain, made up of layers of interconnected "neurons." Each neuron processes input data, applies weights and biases, and passes the result through an activation function. Neural networks learn by adjusting weights to minimize the error in predictions.

Based on nature, neural networks are the usual representation we make of the brain: neurons interconnected to other neurons which forms a network. A simple information transits in a lot of them before becoming an actual thing, like "move the hand to pick up this pencil"



Neural networks can usually be read from left to right. Here, the first layer is the layer in which inputs are entered. There are 2 internals layers (called hidden layers) that do some math, and one last layer that contains all the possible outputs.

What does a Neuron do?

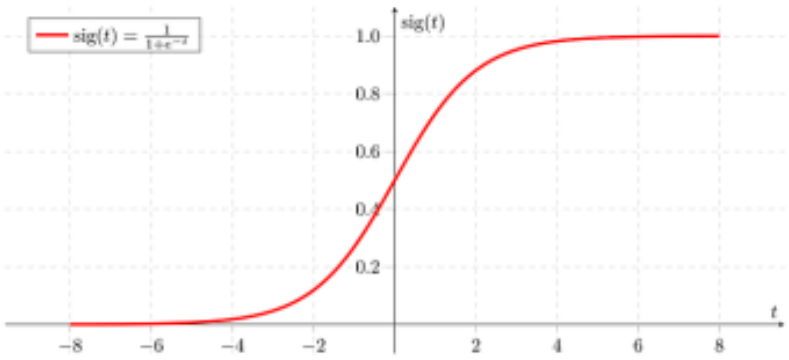


First, it adds up the value of every neuron from the previous column it is connected to. In the above figure, there are 3 inputs (x1, x2, x3) coming to the neuron, so 3 neurons of the previous column are connected to our neuron.

This value is multiplied, before being added, by another variable called "weight" (w1, w2, w3) which determines the connection between the two neurons. Each connection of neurons has its own weight, and those are the only values that will be modified during the learning process.

Moreover, a bias value may be added to the total value calculated. It is not a value coming from a specific neuron and is chosen before the learning phase, but can be useful for the network.

After all those summations, the neuron finally applies a function called "activation function" to the obtained value.



The so-called activation function usually serves to turn the total value calculated before to a number between 0 and 1 (done for example by a sigmoid function shown in the figure). Other function exists and may change the limits of our function, but keeps the same aim of limiting the value.

That's all a neuron does! Take all values from connected neurons multiplied by their respective weight, add them, and apply an activation function. Then, the neuron is ready to send its new value to other neurons After every neurons of a column did it, the neural network passes to the next column. In the end, the last values obtained should be one usable to determine the desired output.

Key Concepts –

- Input Layer: Takes the feature data as input
- Hidden Layers: Process inputs through interconnected neurons and apply non-linear transformations
- Output Layer: Provides the final prediction. For binary classification, it outputs values between 0 and 1
- Activation Function: Introduces non-linearity, allowing the network to learn complex patterns. Common functions include ReLU, Sigmoid, and Softmax
- Loss Function: Measures the difference between predictions and actual values (e.g., Binary Cross-Entropy for binary classification)
- Optimizer: Adjusts weights and biases to minimize the loss (e.g., Stochastic Gradient Descent or Adam)

Procedure to Implement a Neural Network:

1. **Data Collection and Preparation**:

    - Load the dataset (e.g., diabetes.csv) using pd.read_csv().
    - Split the dataset into predictors (X) and response variable (Y) using data.iloc[:, 0:8] for predictors and data.iloc[:, 8] for the target variable.
    - Split the data into training and testing sets using train_test_split(predictors, response, test_size=0.2, random_state=0).

2. **Defining the Neural Network Model**:

    - Initialize the model using Keras' Sequential() API.
    - Add the input layer with Dense(units=12, activation='relu', input_dim=8) for 8 features from the dataset.
    - Add a hidden layer with Dense(units=8, activation='relu') to introduce more learning capacity.
    - Add an output layer with Dense(units=1, activation='sigmoid') for binary classification.

3. **Compiling the Model**:

    - Compile the model using kerasmodel.compile() with:
    - Loss function: binary_crossentropy (appropriate for binary classification).
    - Optimizer: adam (a popular optimizer for neural networks).
    - Metrics: accuracy to track model performance during training.

4. **Training the Model**:

   - Fit the model on the training data using kerasmodel.fit(X_train, Y_train, epochs=150, batch_size=10).
   - Monitor the model's learning progress over 150 epochs with a batch size of 10.

5. **Evaluating the Model**:

   - Evaluate the model on the test data using kerasmodel.evaluate(X_test, Y_test).
   - Print the test accuracy to assess how well the model generalizes to unseen data.

6. **Making Predictions and Calculating Test Accuracy**:

   - Predict the class probabilities for the test set using kerasmodel.predict(X_test).
   - Convert the predicted probabilities into class labels (0 or 1) by using a threshold of 0.5.
   - Calculate the accuracy using accuracy_score(Y_test, y_pred) from sklearn.metrics.

Sample Code

```
# Import Necessary Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense

#keras documentation http://keras.io/

#Loading Diabetes Dataset

#Link to download diabetes dataset :
https://github.com/25-Madhuri/Dataset/blob/main/diabetes.csv
```

```
data = pd.read_csv('/content/diabetes.csv')

data.head()

# The objective is to predict based on diagnostic measurements whether a patient has diabetes

import seaborn as sns
data['Outcome'].value_counts().plot(kind='bar')

# Preparing data for modelling

# Split into input (X) and output (Y) variables
predictors = data.iloc[:, 0:8]  # Features (first 8 columns)
response = data.iloc[:, 8]       # Target (9th column)

X_train, X_test, Y_train, Y_test = train_test_split(predictors, response, test_size=0.2, random_state=0)

# Check the shapes of the splits to ensure everything is correct
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)

# Training the neural Network Model
#There are two ways to build Keras models: sequential and functional
 #The sequential API allows to create models layer by layer

from keras.models import Sequential
from keras.layers import Dense

# Initialize the Sequential model
kerasmodel = Sequential()

# Add layers to the model
kerasmodel.add(Dense(units=12, activation='relu', input_dim=8))  # Input layer with 8 features (X_train.shape[1])
kerasmodel.add(Dense(units=8, activation='relu'))              # Hidden layer with 8 units
kerasmodel.add(Dense(units=1, activation='sigmoid'))           # Output layer (binary classification)
```

```
# Compile the model
kerasmodel.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


#fitting the model
kerasmodel.fit(X_train, Y_train, epochs=150, batch_size=10)

#Train Accuracy
_, accuracy = kerasmodel.evaluate(X_train, Y_train)
print('Train Accuracy: %.2f' % (accuracy*100))

#Test Accuracy
_, accuracy = kerasmodel.evaluate(X_test, Y_test)
print('Test Accuracy: %.2f' % (accuracy*100))

from sklearn.metrics import accuracy_score

# Predict probabilities for the test set
y_pred_prob = kerasmodel.predict(X_test)

# Convert probabilities to class labels (0 or 1) based on a threshold of 0.5
y_pred = (y_pred_prob > 0.5).astype(int)

# Calculate accuracy
accuracy = accuracy_score(Y_test, y_pred)

print("Test Accuracy:", accuracy)
```

Observations -

- Track the model's training and validation accuracy over epochs to understand how well the model is learning.
- Note the final accuracy on the test set and compare it with the training performance to evaluate how well the model generalizes to unseen data.

Conclusion –

Hence, the model summarizes the performance of the neural network, highlighting areas for improvement such as adding more layers, adjusting hyperparameters, or using regularization techniques like dropout to prevent overfitting and enhance generalization.

References –

a. Textbook –

i. Machine Learning with Python – An approach to Applied ML – Abhishek Vijayvargiya, BPB Publications, 1st Edition 2018

ii. Machine Learning, Tom Mitchell, McGraw Hill Education, 1st Edition 1997

b. Online references –

i. https://mize.tech/blog/how-does-a-neural-network-work-implementation-and-5-examples/

ii. https://towardsdatascience.com/first-neural-network-for-beginners-explained-withcode-4cfd37e06eaf

Viva Questions

Sr.No	Question	CO
1.	What is a Neural Network and how does it work?	**1ICPC406_5**
2.	What is a Neural Network and how does it work?	**1ICPC406_5**
3.	What are the main components of a neural network? Describe their roles	**1ICPC406_5**
4.	What is an activation function? Why is it important in a neural network?	**1ICPC406_5**
5.	Define the terms input layer, hidden layer and output layer in a neural network?	**1ICPC406_5**
6.	How do neural networks learn from data? Briefly describe the learning process	**1ICPC406_5**
7.	What is the purpose of the loss function in training a neural network? Provide examples of common loss function	**1ICPC406_5**
8.	Describe the role of an optimizer in a neural network	**1ICPC406_5**
9.	What are hyper-parameters in a neural network? Provide examples and explain their impact on model performance	**1ICPC406_5**
10.	What are hyper-parameters in a neural network? Provide examples and explain their impact on model performance	**1ICPC406_5**