Batch: HO-DL-1 **Experiment Number: 2**

Roll Number: 16010421073 **Name: Keyur Patel**

Aim of the Experiment: To implement Feed forward neural network.

Program/ Steps:

1. Import all the required libraries.

2. Download any simple dataset for classification/prediction task.

3. Create sample weights to be applied in the input layer, first hidden layer and the second hidden layer. Weights for each layer is created as matrix of size M x N where M represents the number of neurons in the layer and N represents number of nodes / neurons in the next layer.

4. Propagate input signal (variables value) through different layer to the output layer.

   a. Weighted sum is calculated for neurons at every layer. Note that weighted sum is sum of weights and input signal combined with the bias element.

   b. Softmax function is applied to the output in the last layer.

5. Forward propagate input signals to neurons in first hidden layer, use tanh function

6. Forward propagate activation signals from first hidden layer to neurons in second hidden layer, use tanh function

7. Forward propagate activation signals from second hidden layer to neurons in output layer

8. Calculate Probability as an output using softmax function.

Output/Result:

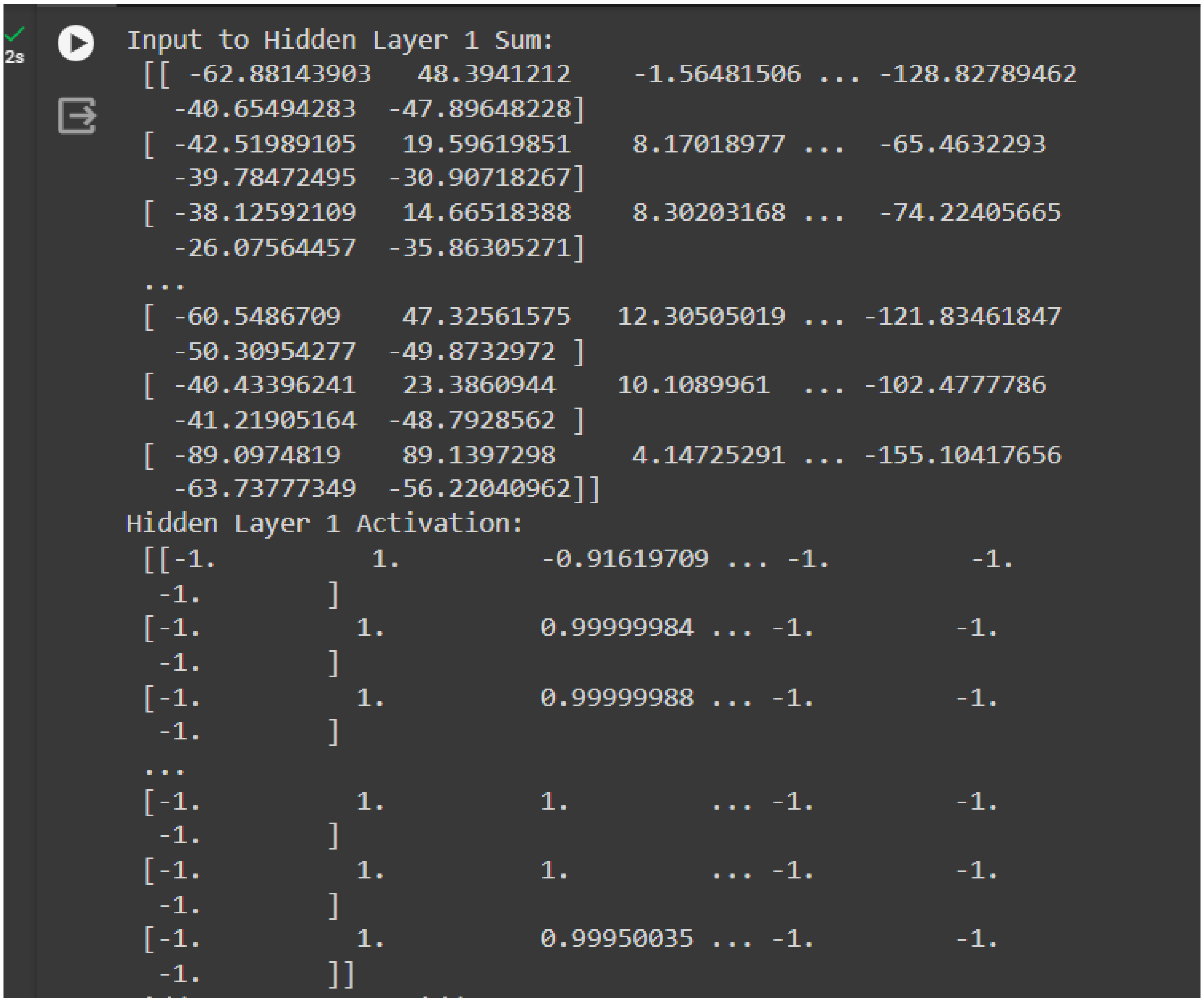
Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from scipy.special import softmax

data = pd.read_csv("Surgical-deepnet.csv")

# Assuming the target variable is named "label" and other columns are
features
```

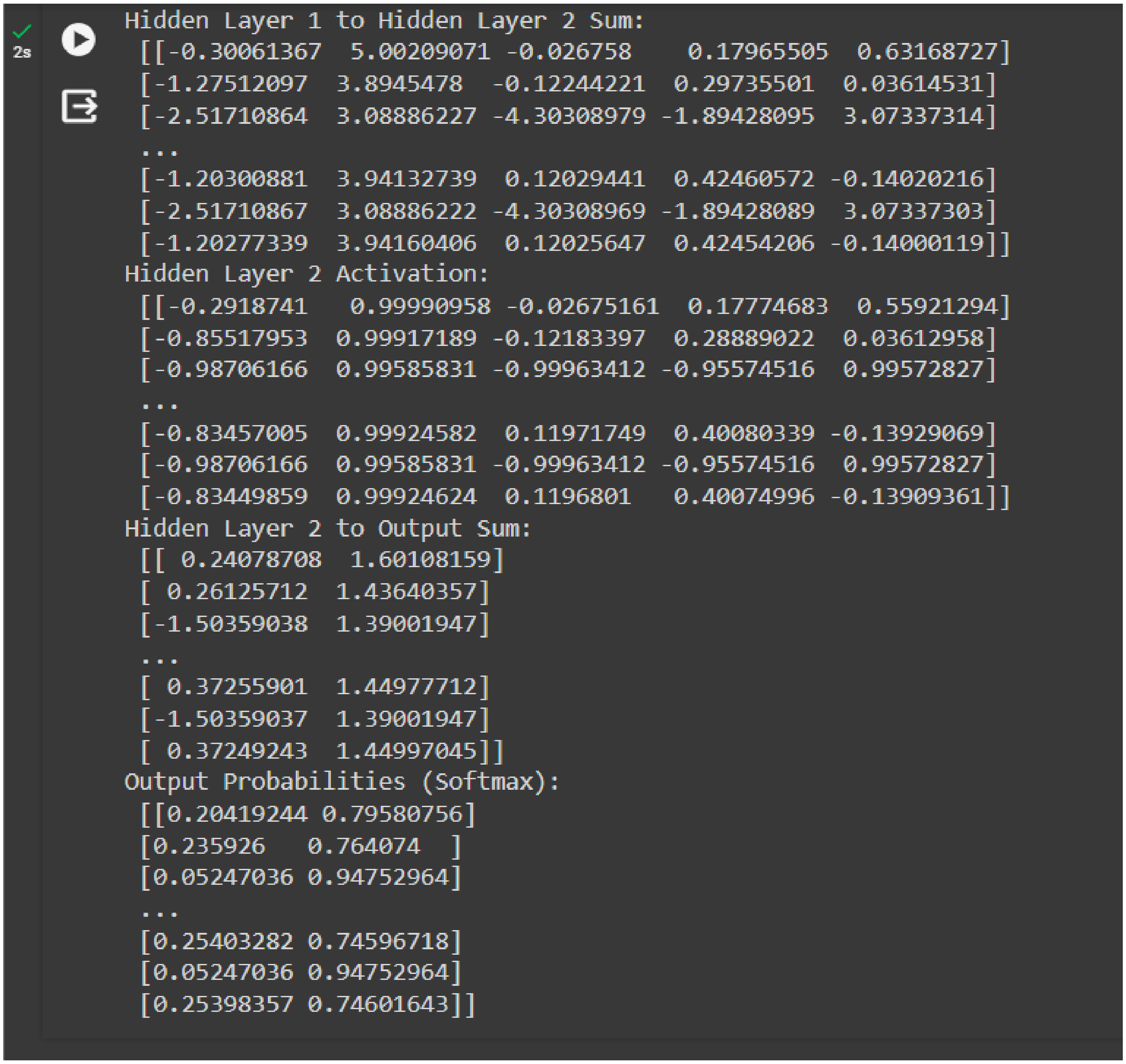
```
X = data.drop("baseline_diabetes", axis=1)
y = data["bmi"]
put_hidden1_weights = np.random.randn(X.shape[1], 10)
hidden1_hidden2_weights = np.random.randn(10, 5)
hidden2_output_weights = np.random.randn(5, 2)

# Assuming binary classification
# Assuming you have already imported the required libraries and loaded the
dataset
# Step 3: Create sample weights for each layer

input_hidden1_weights = np.random.randn(X.shape[1], 10)
hidden1_hidden2_weights = np.random.randn(10, 5)
hidden2_output_weights = np.random.randn(5, 2)

# Assuming binaryclassification

input_hidden1_bias = np.random.randn(1, 10)
hidden1_hidden2_bias = np.random.randn(1, 5)
hidden2_output_bias = np.random.randn(1, 2)

# Step 4: Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Step 5: Propagate input signal through different layers to the output
layer.
# 5a. Weighted sum is calculated for neurons at every layer

input_hidden1_sum = np.dot(X_train, input_hidden1_weights) +
input_hidden1_bias
hidden1_activation = np.tanh(input_hidden1_sum)
hidden1_hidden2_sum = np.dot(hidden1_activation, hidden1_hidden2_weights)+
hidden1_hidden2_bias
hidden2_activation = np.tanh(hidden1_hidden2_sum)
hidden2_output_sum = np.dot(hidden2_activation, hidden2_output_weights)
+hidden2_output_bias

# 5b. Softmax function is applied to the output in the last layer

output_probabilities = softmax(hidden2_output_sum, axis=1)

# Step 6: Forward propagate input signals to neurons in the first hidden
layer, use tanh function
```

```
hidden1_activation = np.tanh(input_hidden1_sum)

# Step 7: Forward propagate activation signals from the first hidden layer
to neurons in the second hidden layer, use tanh function

hidden2_activation = np.tanh(hidden1_hidden2_sum)

# Step 8: Forward propagate activation signals from the second hidden
layer to neurons in the output layer

output_layer_activation = hidden2_output_sum

# Assuming no activation function for the output layer

# Print intermediate results to check the values
print("Input to Hidden Layer 1 Sum:\n", input_hidden1_sum)
print("Hidden Layer 1 Activation:\n", hidden1_activation)
print("Hidden Layer 1 to Hidden Layer 2 Sum:\n", hidden1_hidden2_sum)
print("Hidden Layer 2 Activation:\n", hidden2_activation)
print("Hidden Layer 2 to Output Sum:\n", hidden2_output_sum)
print("Output Probabilities (Softmax):\n", output_probabilities)
```

Output Screenshots:



```
Input to Hidden Layer 1 Sum:
 [[ -62.88143903   48.3941212    -1.56481506 ... -128.82789462
   -40.65494283  -47.89648228]
 [ -42.51989105   19.59619851    8.17018977 ...  -65.4632293
   -39.78472495  -30.90718267]
 [ -38.12592109   14.66518388    8.30203168 ...  -74.22405665
   -26.07564457  -35.86305271]
 ...
 [ -60.5486709    47.32561575   12.30505019 ... -121.83461847
   -50.30954277  -49.8732972 ]
 [ -40.43396241   23.3860944    10.1089961  ... -102.4777786
   -41.21905164  -48.7928562 ]
 [ -89.0974819    89.1397298     4.14725291 ... -155.10417656
   -63.73777349  -56.22040962]]
Hidden Layer 1 Activation:
 [[-1.          1.         -0.91619709 ... -1.         -1.
  -1.        ]
 [-1.          1.          0.99999984 ... -1.         -1.
  -1.        ]
 [-1.          1.          0.99999988 ... -1.         -1.
  -1.        ]
 ...
 [-1.          1.          1.         ... -1.         -1.
  -1.        ]
 [-1.          1.          1.         ... -1.         -1.
  -1.        ]
 [-1.          1.          0.99950035 ... -1.         -1.
  -1.        ]]
```



```
Hidden Layer 1 to Hidden Layer 2 Sum:
 [[-0.30061367  5.00209071 -0.026758    0.17965505  0.63168727]
 [-1.27512097  3.8945478  -0.12244221  0.29735501  0.03614531]
 [-2.51710864  3.08886227 -4.30308979 -1.89428095  3.07337314]
 ...
 [-1.20300881  3.94132739  0.12029441  0.42460572 -0.14020216]
 [-2.51710867  3.08886222 -4.30308969 -1.89428089  3.07337303]
 [-1.20277339  3.94160406  0.12025647  0.42454206 -0.14000119]]
Hidden Layer 2 Activation:
 [[-0.2918741   0.99990958 -0.02675161  0.17774683  0.55921294]
 [-0.85517953  0.99917189 -0.12183397  0.28889022  0.03612958]
 [-0.98706166  0.99585831 -0.99963412 -0.95574516  0.99572827]
 ...
 [-0.83457005  0.99924582  0.11971749  0.40080339 -0.13929069]
 [-0.98706166  0.99585831 -0.99963412 -0.95574516  0.99572827]
 [-0.83449859  0.99924624  0.1196801   0.40074996 -0.13909361]]
Hidden Layer 2 to Output Sum:
 [[ 0.24078708  1.60108159]
 [ 0.26125712  1.43640357]
 [-1.50359038  1.39001947]
 ...
 [ 0.37255901  1.44977712]
 [-1.50359037  1.39001947]
 [ 0.37249243  1.44997045]]
Output Probabilities (Softmax):
 [[0.20419244 0.79580756]
 [0.235926   0.764074  ]
 [0.05247036 0.94752964]
 ...
 [0.25403282 0.74596718]
 [0.05247036 0.94752964]
 [0.25398357 0.74601643]]
```

Post Lab Question-Answers:

1. What is perceptron?

a. a single layer feed-forward neural network with pre-processing

b. an auto-associative neural network

c. a double layer auto-associative neural network

d. a neural network that contains feedback

2. A 4-input neuron has weights 1, 2, 3 and 4. The transfer function is linear with the constant of proportionality being equal to 2. The inputs are 4, 10, 5 and 20 respectively. What will be the output?

a. 76
b. 119
c. 123
d. 238

3. A perceptron adds up all the weighted inputs it receives, and if it exceeds a certain value, it outputs a 1, otherwise it just outputs a 0.

a. True
b. False
c. Sometimes – it can also output intermediate values as well
d. Can’t say

Outcomes: CO1 Understand the evolution of deep learning.

Conclusion (based on the Results and outcomes achieved):
Through this experiment, I learnt about Feed Forward Neural Network

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Jacek M. Zurada, “Introduction to artificial neural systems”, West Publishing

Company

2. Josh Patterson and Adam Gibson, “Deep Learning A Practitioner’s Approach”,

O’Reilly Media 2017