Experiment No.: 2

Title: Feed Forward Neural Network

Batch: HO-DL 1 **Roll No.: 16010422234** **Experiment No. 02**

Aim: To implement Feed forward neural network.

Resources needed: Python

Theory:

A feedforward neural network, also known as a multi-layer perceptron, is composed of layers of neurons that propagate information forward. In this post, you will learn about the concepts of feedforward neural networks along with Python code examples. We will start by discussing what a feedforward neural network is and why they are used. We will then walk through how to code a feedforward neural network in Python. To get a good understanding of deep learning concepts, it is of utmost importance to learn the concepts behind feed forward neural networks in a clear manner. Feed forward neural network learns the weights based on backpropagation algorithm which will be discussed in future posts.

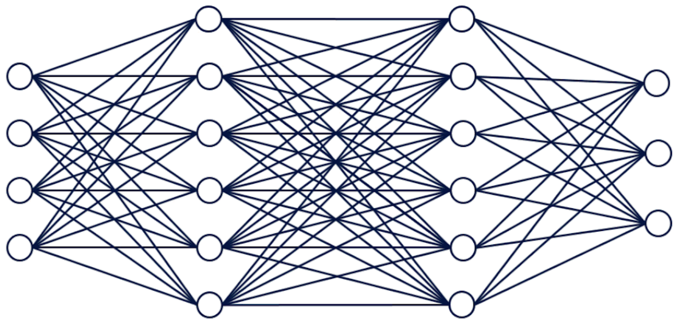


Fig 1. Feedforward neural network for classification task

The neural network shown in Fig.1 consists of 4 different layers – one input layer (layer 1), two hidden layers (layer 2 and 3) and one output layer (layer 4).

1. Input fed into input layer: There are four input variables which are fed into different nodes in the neural network through input layer (1st layer). No computations happen in the input layer.

2. Activations in the first hidden layer: Sum of Input signals (variables) combined with weights and a bias element are fed into all the neurons of the first hidden layer (layer 2). At each neuron, all incoming values are added together (weighted sum of input signals) and then fed into an activation function (such as relu, tanh, sigmoid function etc). Read about different activation functions in this post – Activation functions in neural networks. Based on whether the output of activation function is more than a threshold value decides

whether the neuron gets activated / fired or not. In the first hidden layer, note that only four neurons get activated / fired. The hidden layer can be seen to learn the data representation to be fed into the next hidden layer. This data representation tends to learn the interaction between the features.

3. Activations in the second hidden layer: The activation signals from layer 2 (first hidden layer) are then combined with weights, added with a bias element, and fed into layer 3 (second hidden layer). At each neuron in layer three, all incoming values (weighted sum of activation signals) are added together and then processed with an activation function same as that used in layer 2. This is why this neural network can be termed a fully connected neural network. Based on whether the output of activation function is more than a threshold value decides whether the neuron gets activated / fired or not. In the second hidden layer, note that only three neurons get activated / fired.

4. Softmax output in the final layer: Finally, the activation signals from the second hidden layer are combined with weights and fed into the output layer. At each node in the final / output layer, all incoming values (weighted sum of activation signals) are added together in different nodes and then processed with a function such as softmax function to output the probabilities (in case of classification).

In a feedforward neural network, the value that reaches to the new neuron is the sum of all input signals and related weights if it is first hidden layer, or, sum of activations and related weights in the neurons in the next layers.

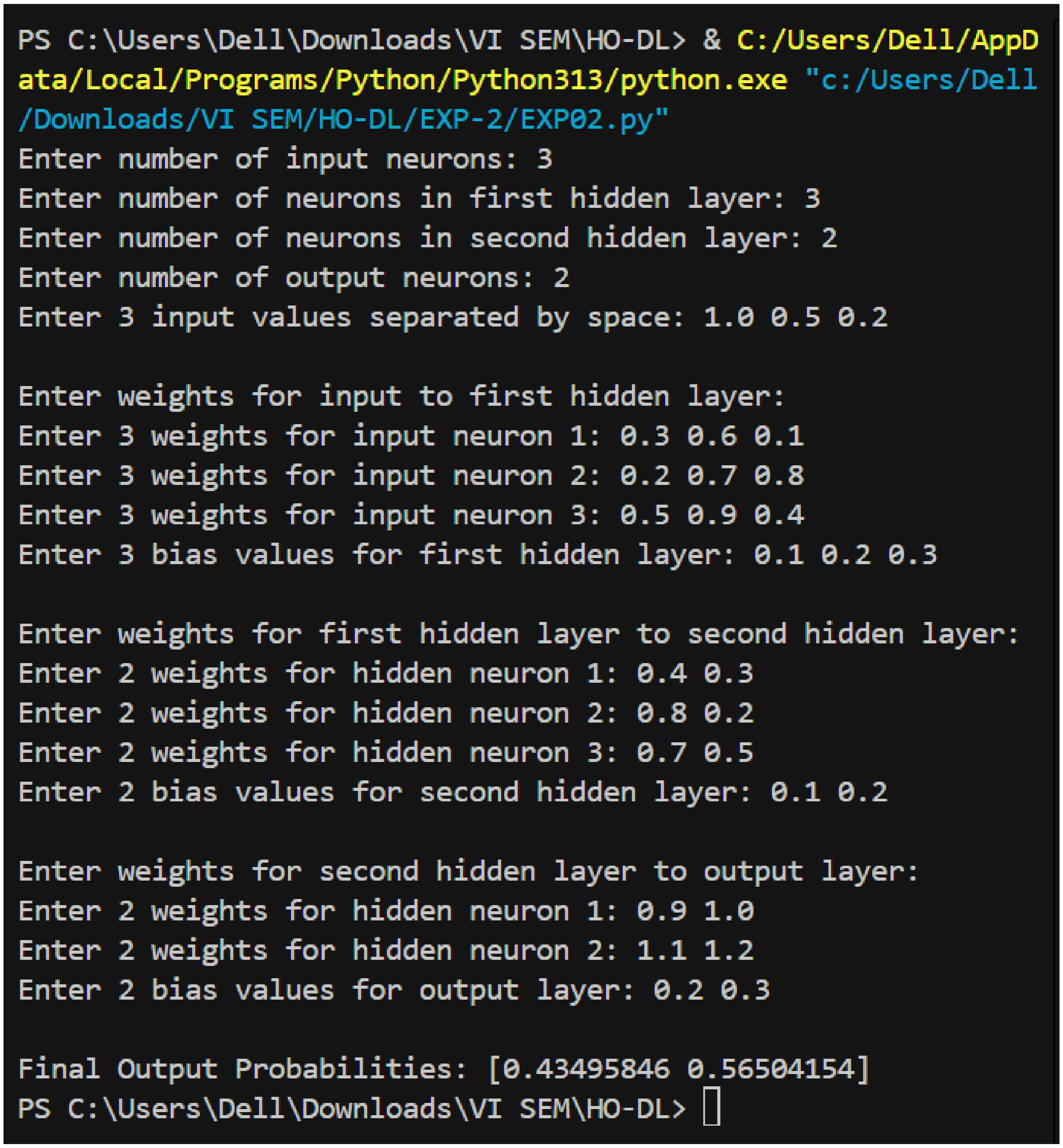
Activity:

1. Import all the required libraries.
2. Download any simple dataset for classification/prediction tasks.
3. Create sample weights to be applied in the input layer, first hidden layer and the second hidden layer. Weights for each layer are created as a 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 layers 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.

Program:

```
import numpy as np

def tanh(x):
    return np.tanh(x)

def softmax(x):
    exp_x = np.exp(x - np.max(x))  # for numerical stability
    return exp_x / exp_x.sum()

# User input for the number of neurons in each layer
num_input_neurons = int(input("Enter number of input neurons: "))
num_hidden1_neurons = int(input("Enter number of neurons in first hidden layer: "))
num_hidden2_neurons = int(input("Enter number of neurons in second hidden layer: "))
num_output_neurons = int(input("Enter number of output neurons: "))

# Input values
inputs = np.array([float(x) for x in input(f"Enter {num_input_neurons} input values separated by space: ").split()])

# Initializing weights and biases for each layer
print("\nEnter weights for input to first hidden layer:")
weights_input_hidden1 = np.array([
    [float(x) for x in input(f"Enter {num_hidden1_neurons} weights for input neuron {i+1}: ").split()]
    for i in range(num_input_neurons)
])
bias_hidden1 = np.array([float(x) for x in input(f"Enter {num_hidden1_neurons} bias values for first hidden layer: ").split()])

print("\nEnter weights for first hidden layer to second hidden layer:")
weights_hidden1_hidden2 = np.array([
    [float(x) for x in input(f"Enter {num_hidden2_neurons} weights for hidden neuron {i+1}: ").split()]
    for i in range(num_hidden1_neurons)
])
bias_hidden2 = np.array([float(x) for x in input(f"Enter
{num_hidden2_neurons} bias values for second hidden layer: ").split()])

print("\nEnter weights for second hidden layer to output layer:")
weights_hidden2_output = np.array([
    [float(x) for x in input(f"Enter {num_output_neurons} weights for
hidden neuron {i+1}: ").split()]
    for i in range(num_hidden2_neurons)
])
bias_output = np.array([float(x) for x in input(f"Enter
{num_output_neurons} bias values for output layer: ").split()])

# Forward propagation
hidden1_input = np.dot(inputs, weights_input_hidden1) + bias_hidden1
hidden1_output = tanh(hidden1_input)

hidden2_input = np.dot(hidden1_output, weights_hidden1_hidden2) +
bias_hidden2
hidden2_output = tanh(hidden2_input)

output_input = np.dot(hidden2_output, weights_hidden2_output) +
bias_output
output = softmax(output_input)

# Display output probabilities
print("\nFinal Output Probabilities:", output)
```

Output:



```
PS C:\Users\Dell\Downloads\VI SEM\HO-DL> & C:/Users/Dell/AppD
ata/Local/Programs/Python/Python313/python.exe "c:/Users/Dell
/Downloads/VI SEM/HO-DL/EXP-2/EXP02.py"
Enter number of input neurons: 3
Enter number of neurons in first hidden layer: 3
Enter number of neurons in second hidden layer: 2
Enter number of output neurons: 2
Enter 3 input values separated by space: 1.0 0.5 0.2

Enter weights for input to first hidden layer:
Enter 3 weights for input neuron 1: 0.3 0.6 0.1
Enter 3 weights for input neuron 2: 0.2 0.7 0.8
Enter 3 weights for input neuron 3: 0.5 0.9 0.4
Enter 3 bias values for first hidden layer: 0.1 0.2 0.3

Enter weights for first hidden layer to second hidden layer:
Enter 2 weights for hidden neuron 1: 0.4 0.3
Enter 2 weights for hidden neuron 2: 0.8 0.2
Enter 2 weights for hidden neuron 3: 0.7 0.5
Enter 2 bias values for second hidden layer: 0.1 0.2

Enter weights for second hidden layer to output layer:
Enter 2 weights for hidden neuron 1: 0.9 1.0
Enter 2 weights for hidden neuron 2: 1.1 1.2
Enter 2 bias values for output layer: 0.2 0.3

Final Output Probabilities: [0.43495846 0.56504154]
PS C:\Users\Dell\Downloads\VI SEM\HO-DL> 
```

Post-lab questions:

1. **What is a 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

    Ans: a single layer feed-forward neural network with pre-processing

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**

    Ans: 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

    Ans: Sometimes – it can also output intermediate values as well

---

CO–2: Comprehend the Deep Network concepts.

Conclusion:

In this experiment, we successfully implemented a feedforward neural network and observed its functioning in propagating inputs through multiple layers to reach the final output. The neural network was designed with an input layer, two hidden layers, and an output layer, utilizing activation functions such as $tan\,h$ in hidden layers and softmax in the output layer. The process involved computing weighted sums at each layer, applying activation functions, and obtaining probabilities for classification tasks. Through this, we gained insights into how deep networks process information, recognize patterns, and make predictions. This experiment provided a foundational understanding of deep learning and how weights and activations play a crucial role in decision-making within a 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
