# Experiment No.: 2
# Title: Feed Forward Neural Network

**Batch:** A4 **Roll No.:** 16010420117 **Experiment:** 2

**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 network along with Python code example. 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 good understanding on deep learning concepts, it is of utmost importance to learn the concepts behind feed forward neural network in a clear manner. Feed forward neural network learns the weights based on backpropagation algorithm which will be discussed in future posts.

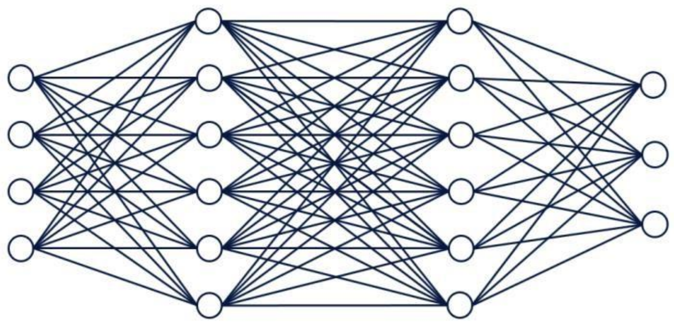


Fig 1. Feedforward neural network for classification task

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 the neural network through input layer (1st layer). No computations happen in the input layer.
2. Activations in first hidden layer: Sum of Input signals (variables) combined with weights and a bias element are fed into all the neurons of 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 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 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 next hidden layer. These data representation tends to learn the interaction between the features.

3. Activations in 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 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 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 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 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.

---

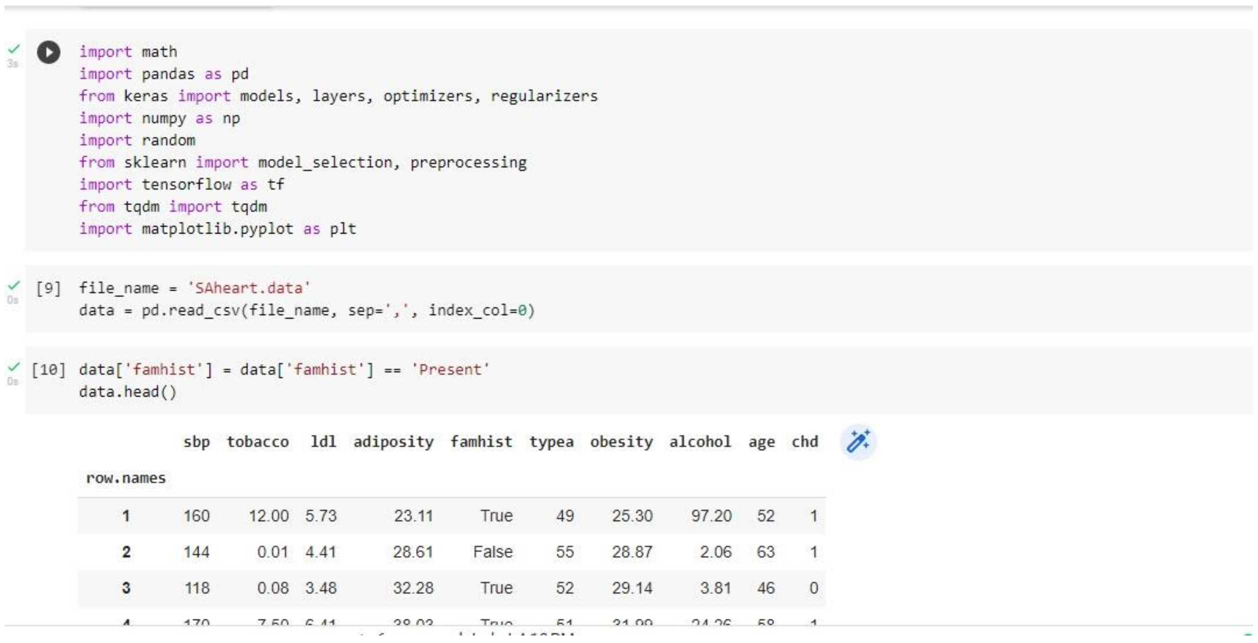
5. Forward propagate input signals to neurons in first hidden layer, use tanh function
6. Forward propagate activation signals from first hidden layer to neurons in second hidden layer, use tanh function

7. Forward propagate activation signals from second hidden layer to neurons in output layer
8. Calculate Probability as an output using softmax function.

---

**Program:**

**Results:**

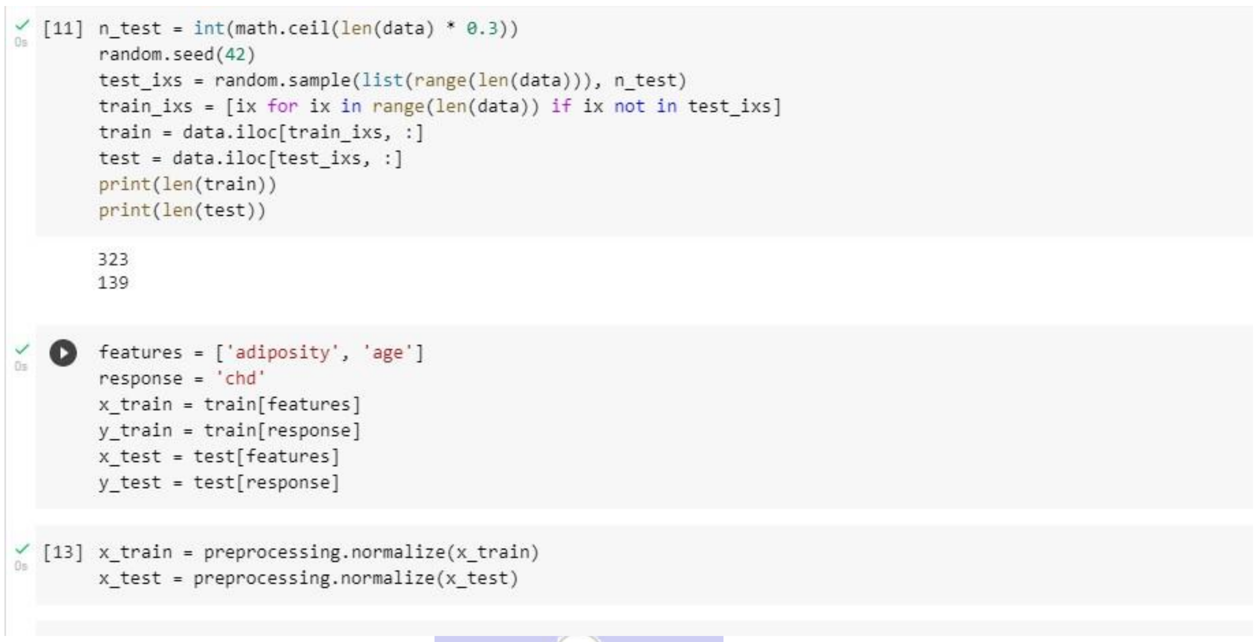


```
import math
import pandas as pd
from keras import models, layers, optimizers, regularizers
import numpy as np
import random
from sklearn import model_selection, preprocessing
import tensorflow as tf
from tqdm import tqdm
import matplotlib.pyplot as plt
```

```
[9] file_name = 'SAheart.data'
    data = pd.read_csv(file_name, sep=',', index_col=0)
```

```
[10] data['famhist'] = data['famhist'] == 'Present'
     data.head()
```

| | sbp | tobacco | ldl | adiposity | famhist | typea | obesity | alcohol | age | chd |
|---|---|---|---|---|---|---|---|---|---|---|
| row.names | | | | | | | | | | |
| 1 | 160 | 12.00 | 5.73 | 23.11 | True | 49 | 25.30 | 97.20 | 52 | 1 |
| 2 | 144 | 0.01 | 4.41 | 28.61 | False | 55 | 28.87 | 2.06 | 63 | 1 |
| 3 | 118 | 0.08 | 3.48 | 32.28 | True | 52 | 29.14 | 3.81 | 46 | 0 |

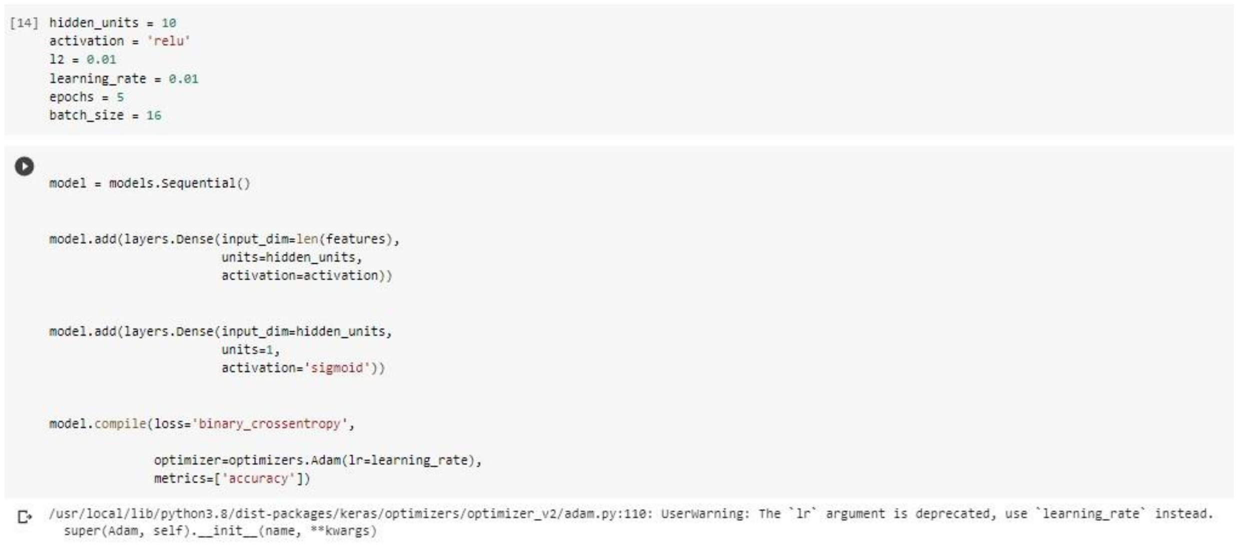


```
[11] n_test = int(math.ceil(len(data) * 0.3))
     random.seed(42)
     test_ixs = random.sample(list(range(len(data))), n_test)
     train_ixs = [ix for ix in range(len(data)) if ix not in test_ixs]
     train = data.iloc[train_ixs, :]
     test = data.iloc[test_ixs, :]
     print(len(train))
     print(len(test))
```

```
323
139
```

```
features = ['adiposity', 'age']
response = 'chd'
x_train = train[features]
y_train = train[response]
x_test = test[features]
y_test = test[response]
```

```
[13] x_train = preprocessing.normalize(x_train)
     x_test = preprocessing.normalize(x_test)
```



```
[14] hidden_units = 10
     activation = 'relu'
     l2 = 0.01
     learning_rate = 0.01
     epochs = 5
     batch_size = 16
```

```
model = models.Sequential()


model.add(layers.Dense(input_dim=len(features),
                       units=hidden_units,
                       activation=activation))


model.add(layers.Dense(input_dim=hidden_units,
                       units=1,
                       activation='sigmoid'))


model.compile(loss='binary_crossentropy',

              optimizer=optimizers.Adam(lr=learning_rate),
              metrics=['accuracy'])
```

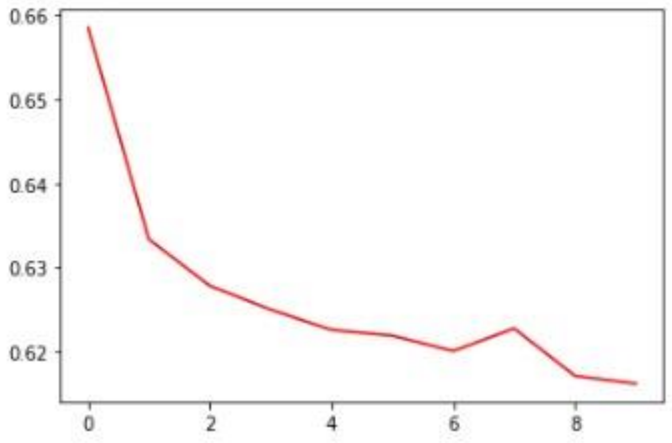
```
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

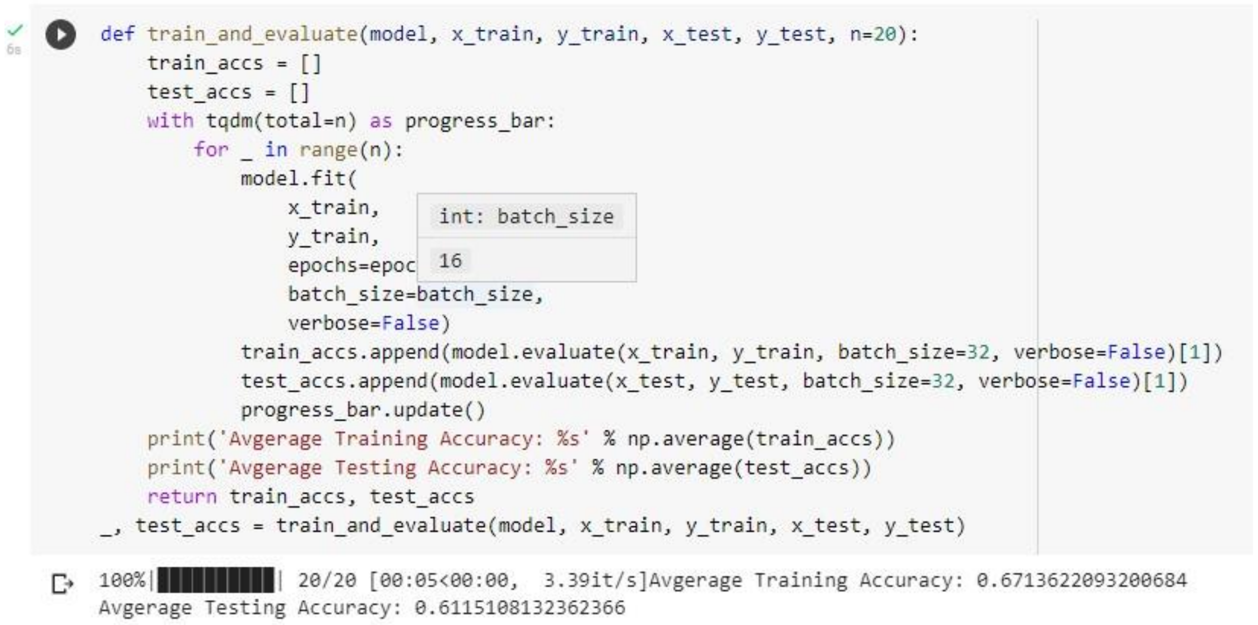
```
history = model.fit(x_train, y_train, epochs=10, batch_size=batch_size)


train_acc = model.evaluate(x_train, y_train, batch_size=32)[1]
test_acc = model.evaluate(x_test, y_test, batch_size=32)[1]
print('Training accuracy: %s' % train_acc)
print('Testing accuracy: %s' % test_acc)

losses = history.history['loss']
plt.plot(range(len(losses)), losses, 'r')
plt.show()
```

```
Epoch 1/10
21/21 [==============================] - 1s 4ms/step - loss: 0.6585 - accuracy: 0.6192
Epoch 2/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6333 - accuracy: 0.6718
Epoch 3/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6278 - accuracy: 0.6718
Epoch 4/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6250 - accuracy: 0.6718
Epoch 5/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6225 - accuracy: 0.6718
Epoch 6/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6219 - accuracy: 0.6718
Epoch 7/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6200 - accuracy: 0.6718
Epoch 8/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6227 - accuracy: 0.6718
Epoch 9/10
21/21 [==============================] - 0s 2ms/step - loss: 0.6171 - accuracy: 0.6718
Epoch 10/10
21/21 [==============================] - 0s 3ms/step - loss: 0.6162 - accuracy: 0.6718
11/11 [==============================] - 0s 2ms/step - loss: 0.6142 - accuracy: 0.6718
5/5 [==============================] - 0s 4ms/step - loss: 0.6604 - accuracy: 0.6115
Training accuracy: 0.6718266010284424
Testing accuracy: 0.6115108132362366
```

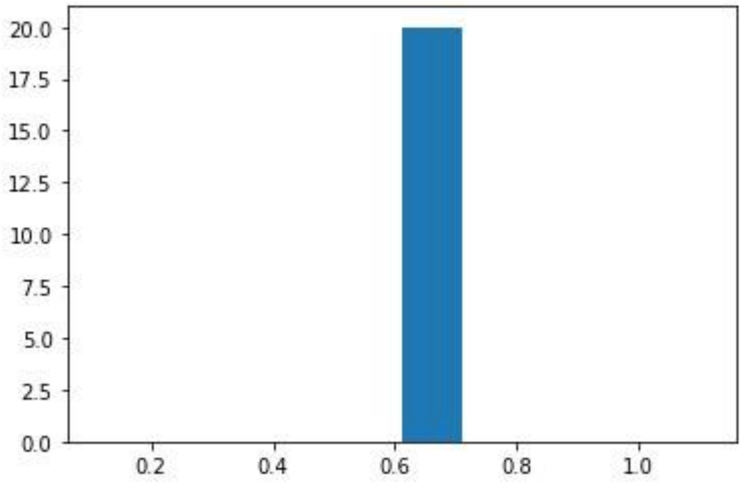




```
def train_and_evaluate(model, x_train, y_train, x_test, y_test, n=20):
    train_accs = []
    test_accs = []
    with tqdm(total=n) as progress_bar:
        for _ in range(n):
            model.fit(
                x_train,
                y_train,
                epochs=epoc
                batch_size=batch_size,
                verbose=False)
            train_accs.append(model.evaluate(x_train, y_train, batch_size=32, verbose=False)[1])
            test_accs.append(model.evaluate(x_test, y_test, batch_size=32, verbose=False)[1])
            progress_bar.update()
    print('Avgerage Training Accuracy: %s' % np.average(train_accs))
    print('Avgerage Testing Accuracy: %s' % np.average(test_accs))
    return train_accs, test_accs
_, test_accs = train_and_evaluate(model, x_train, y_train, x_test, y_test)
```

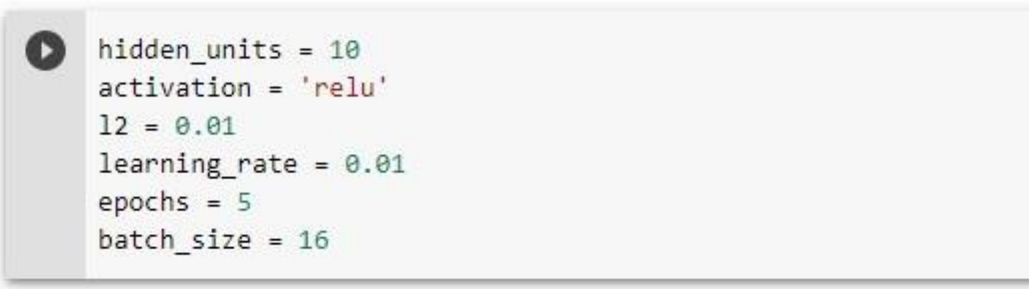
```
100%|██████████| 20/20 [00:05<00:00,  3.39it/s]Avgerage Training Accuracy: 0.6713622093200684
Avgerage Testing Accuracy: 0.6115108132362366
```

```
plt.hist(test_accs)
plt.show()
```

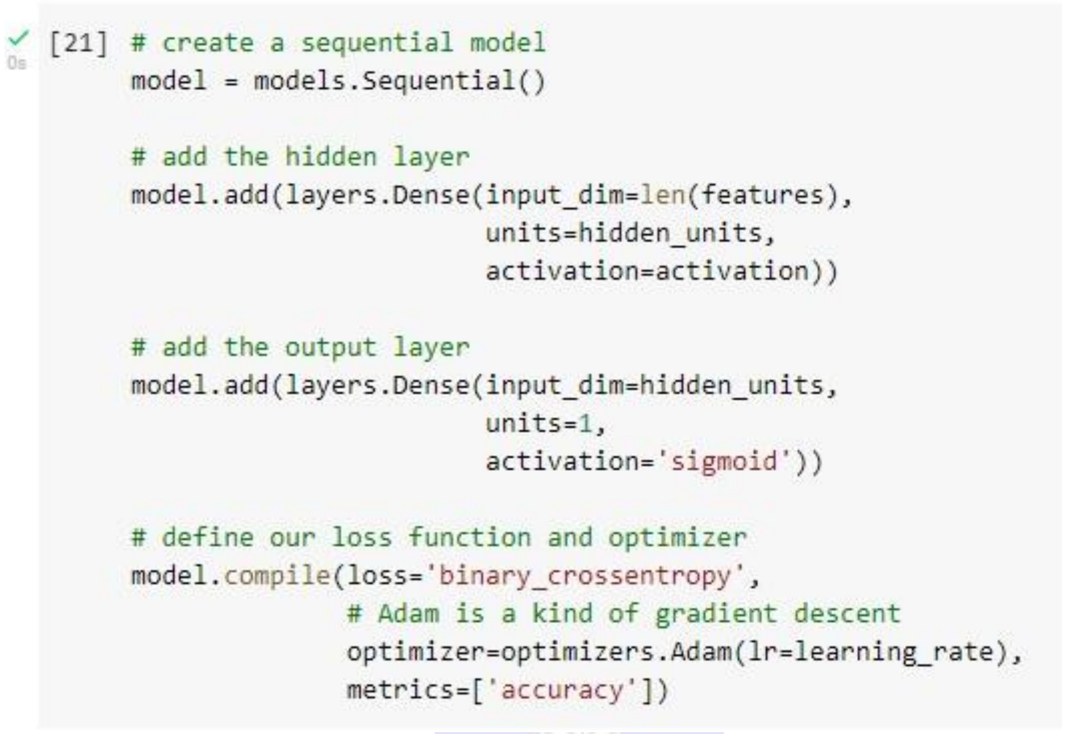


```
[19] print('Min: %s' % np.min(test_accs))
     print('Max: %s' % np.max(test_accs))
```

```
Min: 0.6115108132362366
Max: 0.6115108132362366
```



```
hidden_units = 10
activation = 'relu'
l2 = 0.01
learning_rate = 0.01
epochs = 5
batch_size = 16
```

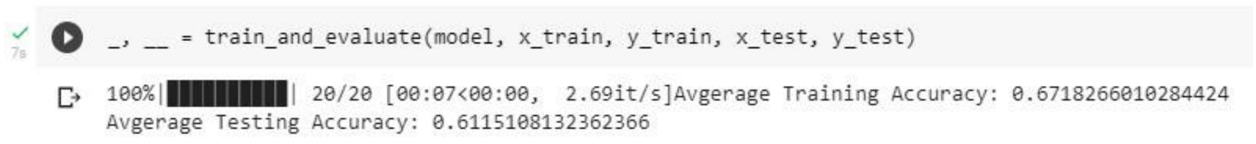


```
[21] # create a sequential model
     model = models.Sequential()

     # add the hidden layer
     model.add(layers.Dense(input_dim=len(features),
                            units=hidden_units,
                            activation=activation))

     # add the output layer
     model.add(layers.Dense(input_dim=hidden_units,
                            units=1,
                            activation='sigmoid'))

     # define our loss function and optimizer
     model.compile(loss='binary_crossentropy',
                   # Adam is a kind of gradient descent
                   optimizer=optimizers.Adam(lr=learning_rate),
                   metrics=['accuracy'])
```



```
_, __ = train_and_evaluate(model, x_train, y_train, x_test, y_test)

100%|██████████| 20/20 [00:07<00:00,  2.69it/s]Avgerage Training Accuracy: 0.6718266010284424
Avgerage Testing Accuracy: 0.6115108132362366
```

**Post-lab questions:**

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

**Ans. a**

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. d**

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. a**

**Outcome:**

**CO2:** Comprehend the Deep Network concepts.

**Conclusion:**

Successfully Implemeneted feed forward neural network and soft max function in python..

**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