2024

Rayyan Masood.
Enrollment no: 01-134212-151
Class: BSCS 6-C
Date: May 17 ,2024

[“ARTIFICIAL INTELLIGENCE LAB]

[“ASSIGNMENT NO:10”]

Lab 10

Artificial Neural Network (ANN)

Neural Networks is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form. The concept of the artificial neural network was inspired by human biology and the way neurons of the human brain function together to understand inputs from human senses.

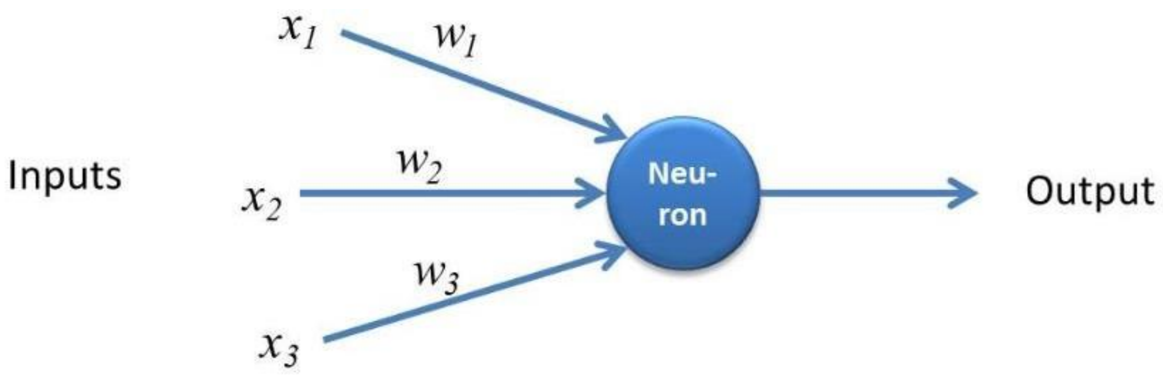
In simple words, Neural Networks are a set of algorithms that tries to recognize the patterns, relationships, and information from the data through the process which is inspired by and works like the human brain/biology.

Components of Neural Network

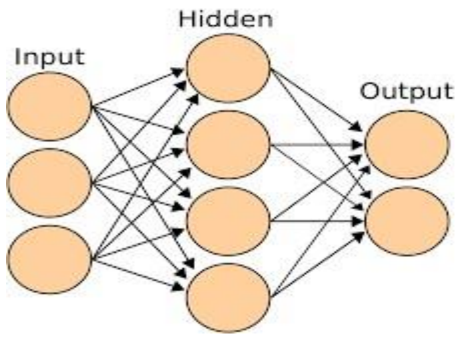
- **Input Layer:** Also known as Input nodes are the inputs/information from the outside world is provided to the model to learn and derive conclusions from. Input nodes pass the information to the next layer i.e Hidden layer.
- **Hidden Layer:** Hidden layer is the set of neurons where all the computations are performed on the input data. There can be any number of hidden layers in a neural network. The simplest network consists of a single hidden layer.
- **Output layer:** The output layer is the output/conclusions of the model derived from all the computations performed. There can be single or multiple nodes in the output layer. If we have a binary classification problem the output node is 1 but in the case of multi-class classification, the output nodes can be more than 1.

Perceptron & Multi-Layer Perceptron

Perceptron is a simple form of Neural Network and consists of a single layer where all the mathematical computations are performed.

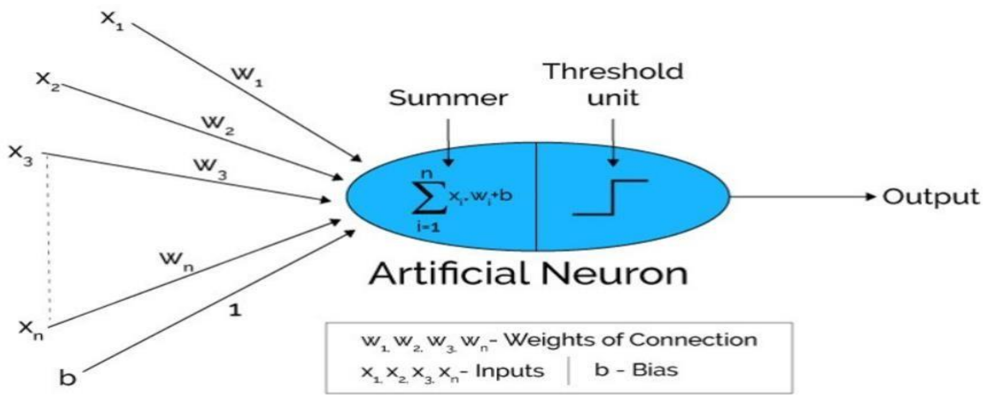


Whereas **Multilayer Perceptron** also known as **Artificial Neural Networks** consists of more than one perception which is grouped together to form a multiple layer neural network.



In the above image, The Artificial Neural Network consists of three layers interconnected with each other:

- An input layer, with 3 input nodes
- Hidden Layer 1, with 4 hidden nodes/4 perceptron's
- Output layer with 2 output nodes



x_1
x_2
x_3
x_n
b
w_1
w_2
w_3
w_n
1
Summer
Threshold unit
$\sum_{i=1}^{n} x_i . w_i + b$
Output
Artificial Neuron
w_1, w_2, w_3, w_n - Weights of Connection
x_1, x_2, x_3, x_n - Inputs
b - Bias

Step by Step Working of Artificial Neural Network

1. In the first step, **Input units are passed i.e data is passed with some weights attached to it to the hidden layer**. We can have any number of hidden layers. In the above image inputs $x_1,x_2,x_3,\ldots x_n$ is passed.
2. Each hidden layer consists of neurons. All the inputs are connected to each neuron.
3. After passing on the inputs, **all the computation is performed in the hidden layer** (Blue oval in the picture)

   Computation performed in hidden layers are done in two steps which are as follows :

   - First of all, **all the inputs are multiplied by their weights**. Weight is the gradient or coefficient of each variable. It shows the strength of the particular input. After assigning the weights, a bias variable is added. **Bias** is a constant that helps the model to fit in the best way possible.

     $\mathbf{Z_1 = W_1*In_1 + W_2*In_2 + W_3*In_3 + W_4*In_4 + W_5*In_5 + b}$

     W_1, W_2, W_3, W_4, W5 are the weights assigned to the inputs In_1, In_2, In_3, In_4, In_5, and b is the bias.
   - Then in the second step, the **activation function is applied to the linear equation Z1.** The activation function is a nonlinear transformation that is applied to the input before sending it to the next layer of neurons. The importance of the activation function is to inculcate nonlinearity in the model.

     There are several activation functions that will be listed in the next section.
4. The whole process described in point 3 is performed in each hidden layer. After passing through every hidden layer, **we move to the last layer i.e our output layer which gives us the final output.**

   The process explained above is known as forwarding Propagation.
5. After getting the predictions from the output layer, the **error is calculated i.e the difference between the actual and the predicted output.**

   If the error is large, then the steps are taken to minimize the error and for the same purpose, **Back Propagation is performed.**

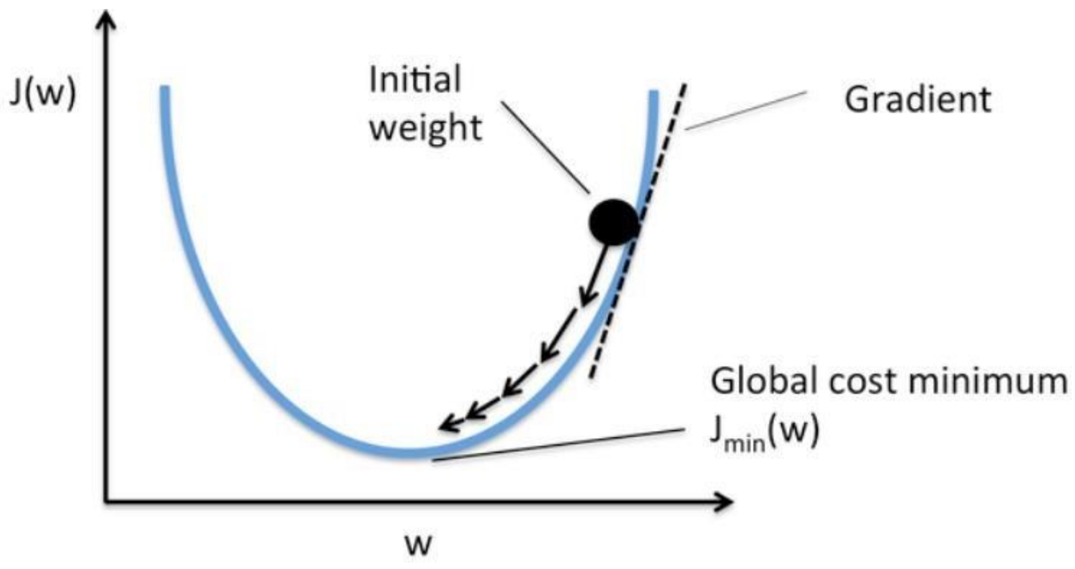
What is Back Propagation & How it works?

Back Propagation is the process of updating and finding the optimal values of weights or coefficients which helps the model to minimize the error i.e difference between the actual and predicted values.

But here are the question is: How the weights are updated and new weights are calculated? The weights are updated with the help of optimizers. Optimizers are the methods/ mathematical formulations to change the attributes of neural networks i.e weights to minimizer the error.

Back Propagation with Gradient Descent

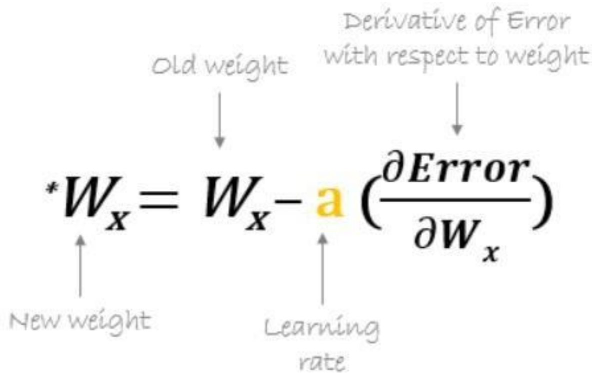
Gradient Descent is one of the optimizers which helps in calculating the new weights. Let’s understand step by step how Gradient Descent optimizes the cost function. In the image below, the curve is our cost function curve and our aim is the minimize the error such that J_{min} i.e global minima is achieved.



<u>Steps to achieve the global minima:</u>

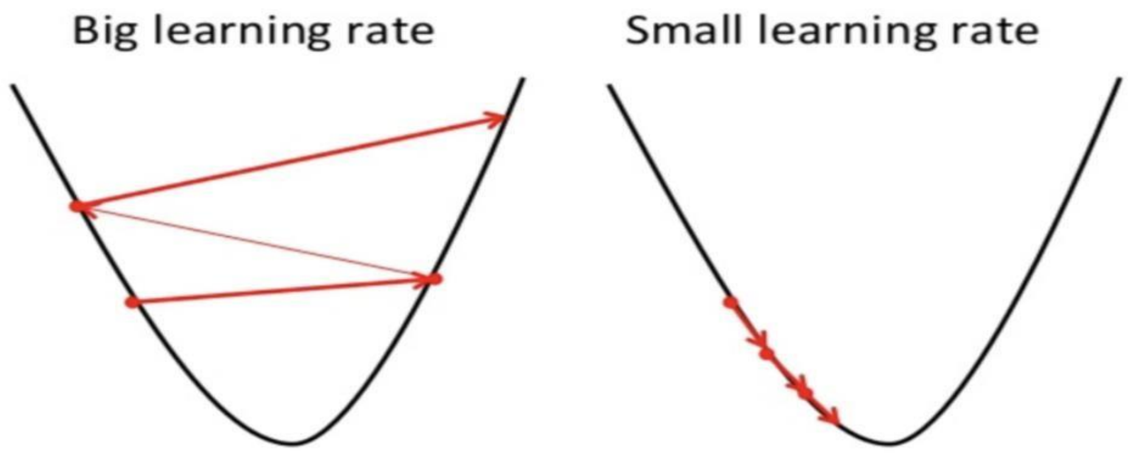
1. First, **the weights are initialized randomly** i.e random value of the weight, and intercepts are assigned to the model while forward propagation and **the errors are calculated** after all the computation. (As discussed above)
2. Then the **gradient is calculated i.e derivative of error w.r.t current weights**
3. Then new weights are calculated using the below formula, where **a is the learning rate** which is the parameter also known as step size to control the speed or steps of the backpropagation. It gives additional control on how fast we want to move on the curve to reach global minima.

4. This process of calculating the new weights, then errors from the new weights, and then updation of weights **continues till we reach global minima and loss is minimized**.



$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

5. A point to note here is that the learning rate i.e **a in our weight updation** equation should be chosen wisely. Learning rate is the amount of change or step size taken towards reaching global minima. **It should not be very small** as it will take time to converge as well as **it should not be very large** that it doesn't reach global minima at all. Therefore, the learning rate is the hyperparameter that we have to choose based on the model.



Activation Function

Activation functions are attached to each neuron and are mathematical equations that determine whether a neuron should be activated or not based on whether the neuron's input is relevant for the model's prediction or not. The purpose of the activation function is to introduce the nonlinearity in the data.

- Sigmoid Activation Function
- TanH / Hyperbolic Tangent Activation Function
- Rectified Linear Unit Function (ReLU)
- Leaky ReLU
- Softmax

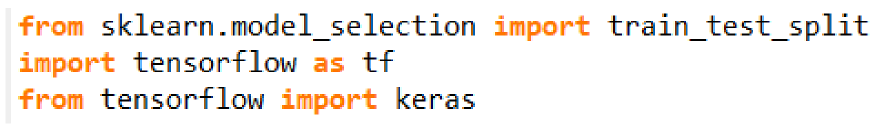
Installation of Libraries

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. We will use the Keras API with Tensorflow backends for creating our neural network.

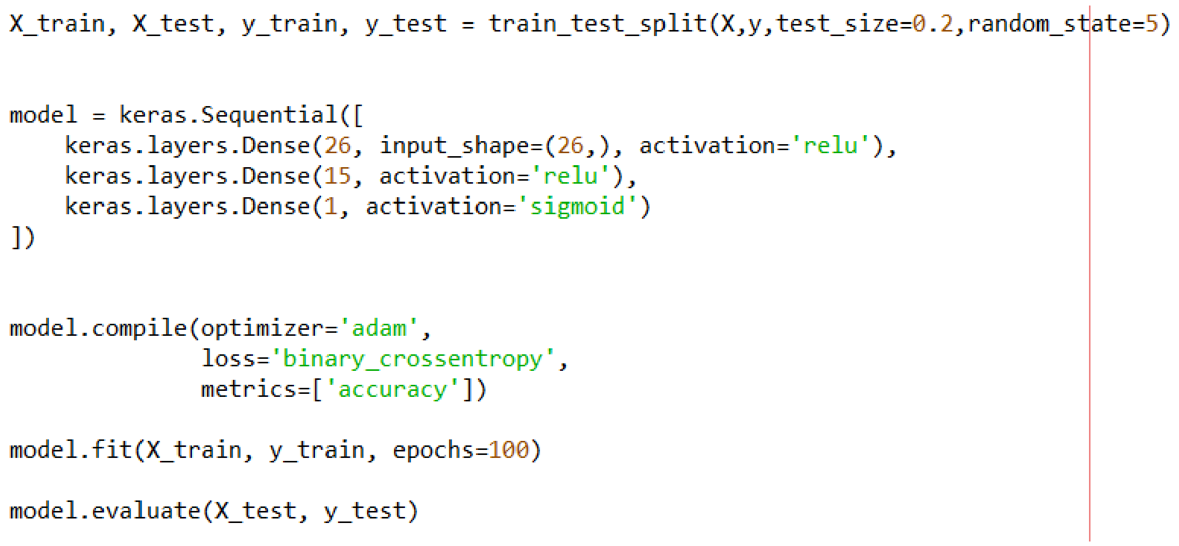
You can use following given command in anaconda prompt to install tenserflow.

```
pip install tensorflow
```

After installation you can use tensorflow library with keras API for designing neural network architecture.



```python
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
```



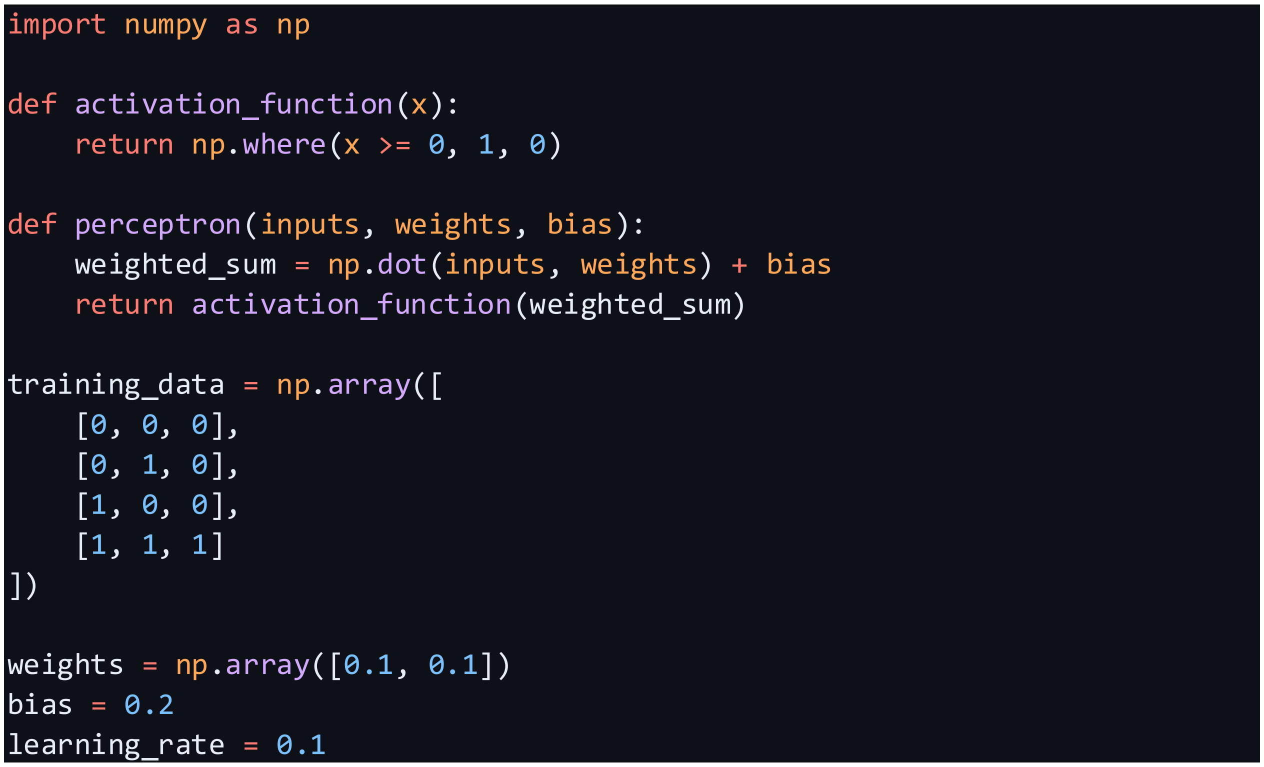
```python
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=5)


model = keras.Sequential([
    keras.layers.Dense(26, input_shape=(26,), activation='relu'),
    keras.layers.Dense(15, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])


model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100)

model.evaluate(X_test, y_test)
```

Lab Journal 10:

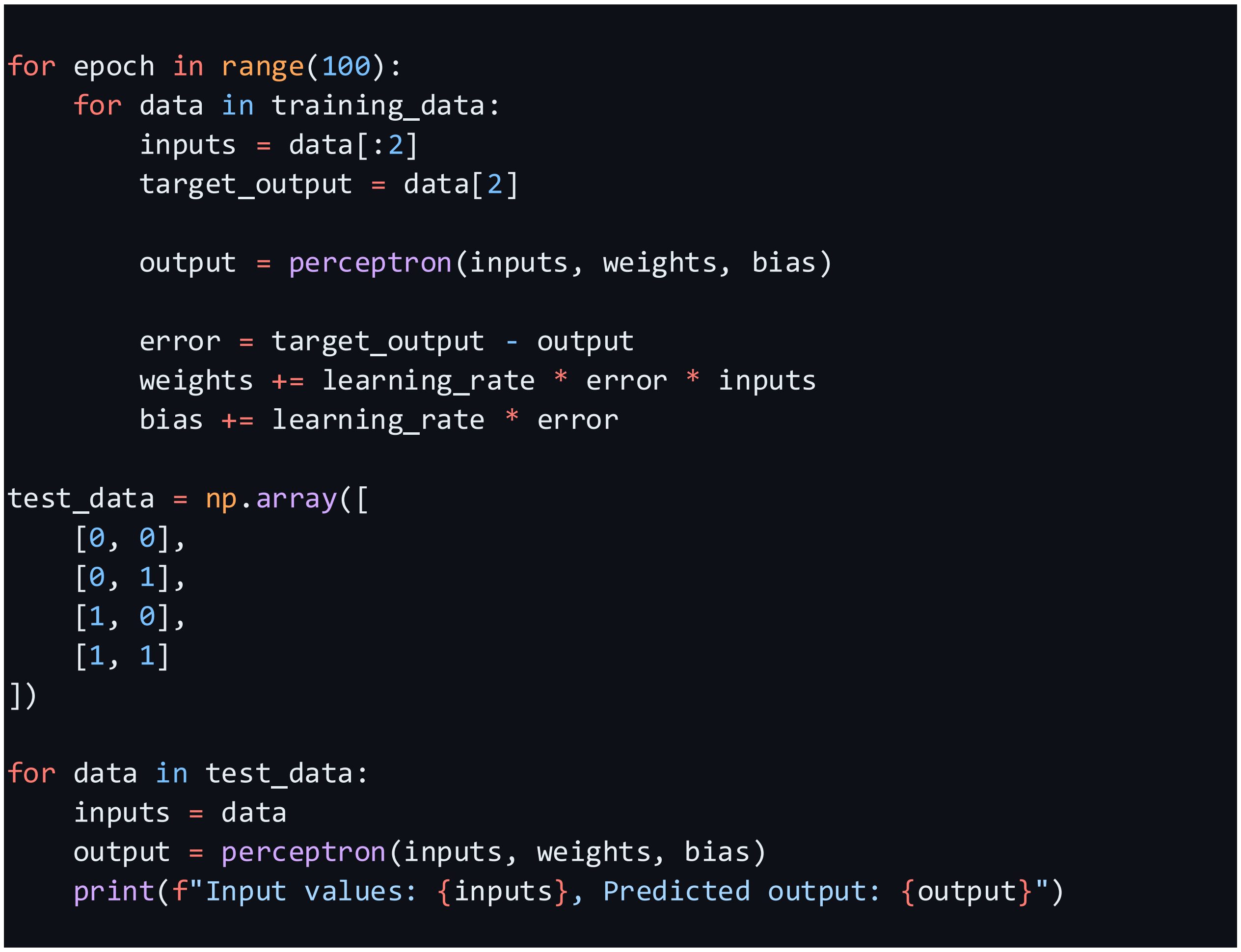
1. Develop a perceptron to learn the weights of following given **OR gate** using **step as activation function, bias as 0.2 and learning rate as 0.1.**

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Code:

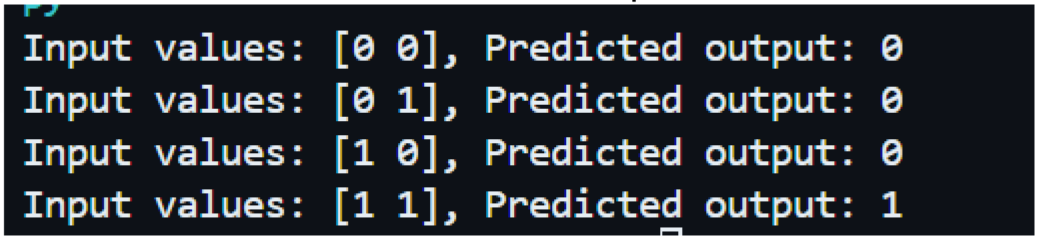


```python
import numpy as np

def activation_function(x):
    return np.where(x >= 0, 1, 0)

def perceptron(inputs, weights, bias):
    weighted_sum = np.dot(inputs, weights) + bias
    return activation_function(weighted_sum)

training_data = np.array([
    [0, 0, 0],
    [0, 1, 0],
    [1, 0, 0],
    [1, 1, 1]
])

weights = np.array([0.1, 0.1])
bias = 0.2
learning_rate = 0.1
```



```python
for epoch in range(100):
    for data in training_data:
        inputs = data[:2]
        target_output = data[2]

        output = perceptron(inputs, weights, bias)

        error = target_output - output
        weights += learning_rate * error * inputs
        bias += learning_rate * error

test_data = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

for data in test_data:
    inputs = data
    output = perceptron(inputs, weights, bias)
    print(f"Input values: {inputs}, Predicted output: {output}")
```

Output:



```
Input values: [0 0], Predicted output: 0
Input values: [0 1], Predicted output: 0
Input values: [1 0], Predicted output: 0
Input values: [1 1], Predicted output: 1
```