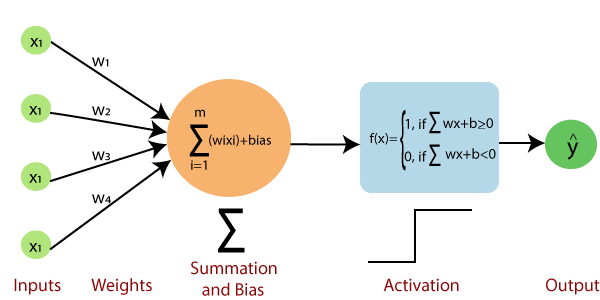
| **Ex no : 1** | **Implementation of Multi-layer network for MNIST Dataset.** |
| --- | --- |
| **Date :** |

**AIM**

To study the difference between single layer and multi-layer network and implement multi-layer network for MNIST dataset

**BASIC THEORY of ANN.**

* Input value or One input layer: The input layer of the perceptron is made of artificial input neurons and takes the initial data into the system for further processing.
* Weights and Bias:
* Weight: It represents the dimension or strength of the connection between units. If the weight to node 1 to node 2 has a higher quantity, then neuron 1 has a more considerable influence on the neuron.
* Bias: It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
* Net sum: It calculates the total sum.
* Activation Function: A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.

****

**PROCEDURE AND CODE**

1. Import the following libraries
   1. Tensorflow
   2. Form tensorflow import keras
   3. From keras import sequential model
   4. From keras import dense and flattern layers

**Code**

import tensorflow

from tensorflow import keras

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense,Flatten

1. Download MNIST dataset directly from keras example datasets and split train and test data.

**Code**

(X\_train,y\_train),(X\_test,y\_test) = keras.datasets.mnist.load\_data()

1. View the shape.

Code

X\_test.shape

X\_train.shape

y\_train.shape

y\_test.shape

1. View the class labels in y\_train

**Code**

y\_train

1. Import matplotlib and view any sample in x\_train and verify the value in y\_train matches the figure by manually.

**Code**

import matplotlib.pyplot as plt

plt.imshow(X\_train[2])

1. Divide x\_train and y\_train by 255

Code

X\_train = X\_train/255

X\_test = X\_test/255

1. View the data of first image in x\_train

**Code**

X\_train[0]

1. Create sequential model and add flatten and dense layers

**Code**

model = Sequential()

model.add(Flatten(input\_shape=(28,28)))

model.add(Dense(128,activation='relu'))

model.add(Dense(32,activation='relu'))

model.add(Dense(10,activation='softmax'))

1. **View model summary**

**Code**

model.summary()

1. **Compile and fit the model**

**Code**

model.compile(loss='sparse\_categorical\_crossentropy',optimizer='Adam',metrics=['accuracy'])

history = model.fit(X\_train,y\_train,epochs=25,validation\_split=0.2)

1. **Create y\_pred variable**

**Code**

y\_prob = model.predict(X\_test)

y\_pred = y\_prob.argmax(axis=1)

1. **Import metrics from sklearn and view the accuracy score**

**Code**

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,y\_pred)

1. **Plot the actual and validation losses, accuracies**

**Code**

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

1. **Test with any random input**

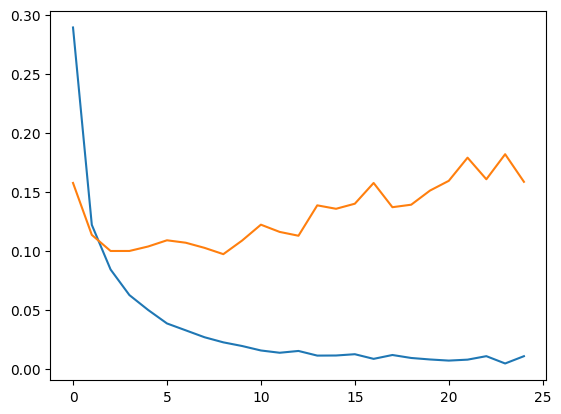
**Code**

plt.imshow(X\_test[1])

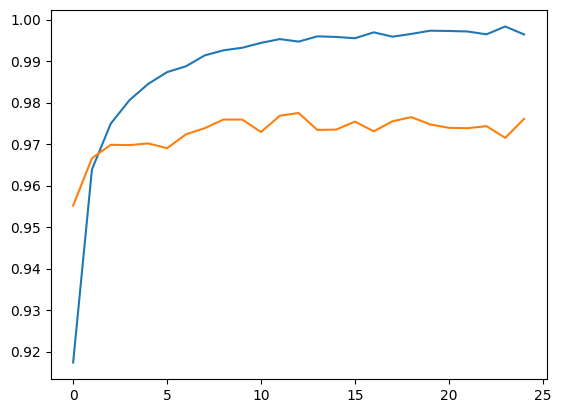
model.predict(X\_test[1].reshape(1,28,28)).argmax(axis=1)

**OUTPUT**

Loss during training and validation



Accuracy during training and validation



**RESULT**

Thus multi-layer neural network with a input layer (flatten layer), three hidden layers(dense) ,out of which last one is output layer has designed and implemented for MNIST dataset successfully.