Experiment No-02.

Aim :-

implementing ~~fo~~ feedforword neural Network with Keras and Tensorflow

Objective :-

a. import the necessary package's
b. Load the training and testing data (MNIST).
c. Define the network architecture using keras.
d. Train the model using SGD.
E Evaluate the Network.
F. Plot the training less and accuracy.

Theory :-

- To Demostrate how you can implement ~~feedforword multi-layer network and apploy them to the~~ MNIST ~~and CIFAR-10 Datasets~~. Feed simple neural Networks using the keras library.
- Obtain basline standard Nural Network which we will e later compare to convolution Nural Network

~~MNIST~~ :-

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
get_ipython().run_line_magic('matplotlib',
  'inline')
```

MNIST :-

- MNIST stand "Modified National Institute of Standard and Technology"
- It is a Dataset of 70,000 handwrite image each image is of 20 X 20 pixels.
  i.e about 784 features.
  eachfeatyrs. each represent only one pixel intensiy i.e from 0 (White) to 255 (black).
- these Database is further divided into 60,000 training and 10,000 testing image.

```
# import dataset and split into train and test & data
mnist = tf, keres, dataset.mnist
(X_train, y_tarin), (X_test, y_test) = mnist.load_
 data ().
```

```
# shape of training dataset 60,000, images
  having 28x28 size x_train.shape.
```

Network Architecture using keras :-

- Tenserflow is an opensource set of libraries for Creating and Working with Neural Network, such as those used in ML. and DL project.

Creating the model :-

- The ReLU function is one of the most popular activation function's.
- It Stand for "reflected linear unit" mathamatically this function is define as

- $y = max(0, x)$ The ReLU function returns "0" if the input is Nagative and is linear if.
- The input is positive.
- the softmax is another activation function.

```
model = keras.sequantial([
keras.layers.flatten(input_shape = (28,28)),
keras.layers.Dense(128, Activation 'relu'),
keras, layers, Dense(10, activation = 'softmax')
])
```

~~Graph~~ # Train the model

```
history= model.fit(x_train, y_train, validation.data
(x_test, y_test), epochs =10)
```

Evaluate the model / Network:-

```
test_loss, test_acc = model.evaluate (x_test, y_test)
print ("loss = %., 3f" %. test_loss)
print ("Accuracy = %. 3f"%. test_acc)
```

- prediction of the data
- plot graph for Accuracy and loss

```
get_ipython().run_line_magic('pinfo2', 'history.
history')
history.history.keys()
```

graph representing the model's accuracy

In [23]:

```
plt.plot (history.history ['loss'])
plt.plot(history.history ['val_loss'])
plt.title ('model loss')
plt.xlabel ('epoch')
plt.legend (['Train', 'validation'], loc = 'upper left')
plt.show()
```

graph represent's the model's loss

In [24]:

```
plt.plot (history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot (history.history['loss']).
plt.title ['Training loss and accuracy')
plt.ylabel ('accuracy loss')
plt xlabel ('epoch')
plt-legend ( ['accuracy', 'val_accuracy', 'loss', 'val_
                loss']).
plt.show()
```

Conclusion :-

With above code We can see. that throughput With epoochs, Our model accuracy increase and our model loss decreasses, that is good since our model gains confidence With its prediction.