Assignment No: 02

Aim:- Implementing Feed Forward neural network with Keras & Tensorflow

a] Import the necessary packages
b] Load training & testing data
c] Define network architecture using Keras
d] Train model using SGD
e] Evaluate the network
f] Plot the training loss and accuracy

Objective:- To learn how to develop a feed forward network & how to optimize it for better performance

Infrastructure:- Computer / Laptop

Software used:- Jupyter Notebook / Google Colab.

Theory:-

Feed Forward Neural Network:-

Deep feedforward network also often called feedforward neural networks, or multilayer perceptron's. The goal of a feedforward network is to approximate some function f^*. A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameter θ that result in the best function approximation.

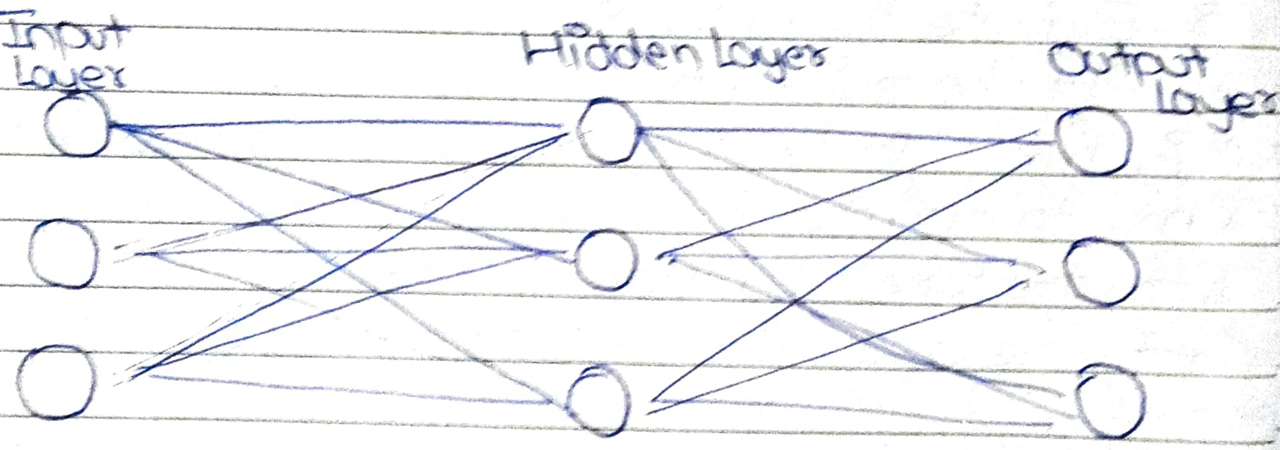
- These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f & finally to the output y. There are no feedback connections in which outputs of model are fed back into itself

Structure of feed forward Neural Networks

1°] Input Layer :- It is where the user accepts the layer for the neural network.

2°] Hidden Layer :- This is the layer where all th computation required for the predictions are do

3°] Output Layer :- The output from the hidden la is provided at output layer.



The nodes are connected with the help of edg The edges are represented as $W_{i,j}$ where i represent the node where the edge starts fro & j represent the node where the edge ends.

- The nodes compute the output for next layer by summation of the product of node input & the we associated with node edge, which is then ap to an activation function to decide whether the n should fire or not for the ~~output~~ input

SGD

· Stochastic Gradient Descent & its variants are probably the most used optimization algorithms for machine learning in general & for deep learning in particular. A crucial parameter for the SGD algorithm is learning rate

MNIST :-

The MNIST data set of handwritten digits has a training set of 70,000 examples and each row of the matrix corresponds to a 28×28 image. The unique values of the response variable y range from 0 to 9

CIFAR10 :-

CIFAR-10 is an established computer vision dataset used for object recognition. The data we will use in this example is a subset of an 80 million tiny images datasets & contains of 60,000 32×32 color images containing one of 10 object classes. Furthermore the data were converted from RGB to gray, normalised & rounded to 2 decimal places

Implementation

1] Import the necessary libraries
2] Load the dataset from libraries or from outside
3] Build the feed forward neural networks using Keras
4] Evaluate the model for accuracy & other evaluation metrics

4] Train the model with the dataset & use SGD as optimizer.
6] Plot the loss and accuracy function

Conclusion :-
We developed a feed forward neural network for hand written digit recognisation

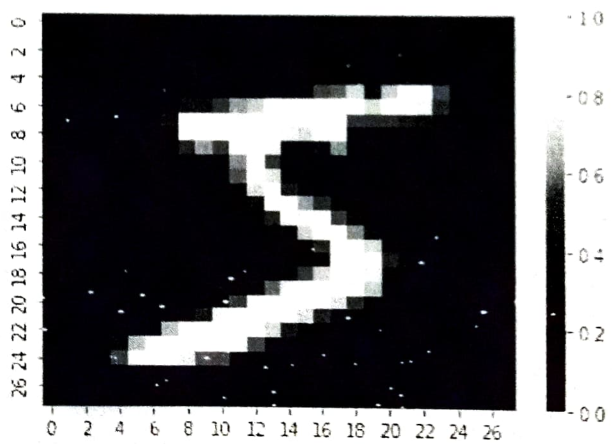
Name: Ojas Dhananjay Bhat
Roll No.: 012
PRN No:- 72176150L
Class:- BE[IT]
Subject: LP-IV(DL)

In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout, Flatten
import matplotlib.pyplot as plt
# import seaborn as sns
```

MNIST dataset

In [2]:
```
mnist = tf.keras.datasets.mnist
(x_train, y_train) , (x_test, y_test) = mnist.load_data() # Data Loading
x_train, x_test = x_train/255.0 , x_test/255.0 #Normalizing the data
```

In [27]:
```
sns.heatmap(x_train[0])
plt.show()
```



Prepearing the model

In [3]:
```
model = Sequential([
     Flatten(input_shape=(28,28)),
     Dense(128, activation="relu"),
     Dropout(0.2),
     Dense(10)
])
```

```
In [5]: predictions = model(x_train[:1]).numpy()
        predictions

Out[5]: array([[-0.24707137, -0.64293617,  0.32793105, -0.7325163 , -0.10029303,
                 0.42578584, -0.5628654 , -0.9137927 , -1.2458755 ,  0.75219357]],
              dtype=float32)

In [6]: tf.nn.softmax(predictions).numpy()

Out[6]: array([[0.08687506, 0.0584754 , 0.15438868, 0.05346493, 0.10060976,
                0.17026025, 0.06335013, 0.0446007 , 0.03199779, 0.23597735]],
              dtype=float32)

In [8]: loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

In [9]: model.compile(optimizer="adam", loss = loss_fn, metrics=["accuracy"])

In [10]: model.fit(x_train, y_train, epochs=5)

         Epoch 1/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.2930 - accur
         acy: 0.9143
         Epoch 2/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.1412 - accur
         acy: 0.9575
         Epoch 3/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.1048 - accur
         acy: 0.9683
         Epoch 4/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.0855 - accur
         acy: 0.9736
         Epoch 5/5
         1875/1875 [==============================] - 5s 3ms/step - loss: 0.0729 - accur
         acy: 0.9769

Out[10]: <keras.callbacks.History at 0x27c0bc71210>

In [11]: model.evaluate(x_test, y_test, verbose=2)

         313/313 - 1s - loss: 0.0750 - accuracy: 0.9764 - 849ms/epoch - 3ms/step

Out[11]: [0.07503402978181839, 0.9764000177383423]
```

Validation of Model

```
In [12]: val = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test), ba1

         Epoch 1/5
         300/300 [==============================] - 1s 4ms/step - loss: 0.0516 - accurac
         y: 0.9841 - val_loss: 0.0659 - val_accuracy: 0.9793
         Epoch 2/5
         300/300 [==============================] - 1s 4ms/step - loss: 0.0456 - accurac
         y: 0.9858 - val_loss: 0.0641 - val_accuracy: 0.9799
         Epoch 3/5
         300/300 [==============================] - 1s 4ms/step - loss: 0.0437 - accurac
         y: 0.9865 - val_loss: 0.0649 - val_accuracy: 0.9807
         Epoch 4/5
         300/300 [==============================] - 1s 4ms/step - loss: 0.0415 - accurac
         y: 0.9870 - val_loss: 0.0633 - val_accuracy: 0.9812
         Epoch 5/5
         300/300 [==============================] - 1s 4ms/step - loss: 0.0398 - accurac
         y: 0.9874 - val_loss: 0.0627 - val_accuracy: 0.9808
```

```
In [13]: plt.title("Model Accuracy")
         plt.ylabel("Accuracy")
         plt.xlabel("epoch")
         plt.plot(val.history["accuracy"])
         plt.plot(val.history["val_accuracy"])
         plt.legend(["train","val"])
         plt.show()
```