Date:-

Exp No

Aim:- Working with tensorflow, Numpy, Keras, Pandas, Matplotlib

Description:-

Tensorflow:- It is an open-source machine learning library developed by the Google Brain team. It is widely used for building and training machine learning models

Numpy:- It is a fundamental package for scientific computing in python. It provides support for large, multi-dimencional array and matrices.

Keras:- It is an open source high-level neural networks API written in python. It acts as an interface for building and training deep learning models.

Pandas:- It is a powerful data manipulation and analysis library for python.

Matplotlib:- It is a 2D plotting library for creating static, animated and interactive visualizations in python.

Program :-

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import sequential
from sklearn.metrics import mean_squared_error
np.random.seed(42)
data={'x':np.random.rand(100,1).flatten()}
data['y']=3 * data['x']+ 2+0.1 * np.random.randn(100)
df=pd.DataFrame(data)
print(df.info())
X_train, X_test, y_train, y_test = train_test_split
(df['x'], df['y'], test_size=0.2, random_state=42)
model=Sequential()
model.add(Dense(units=6, activation='relu'))
model.add(Dense(units=1, activation='linear'))
model.build(input_shape=(None,1))
model.compile(optimizer='sgd', loss='mean_square_error
model.summary()
# Train the model
history = model.fit(X_train, y_train, epochs=100,
                                    verbose=0)
y_pred=model.predict(X_test)
mse = mean_squared_error(y_test, y_pred
```

```
print(f'Mean Squared Error on test set : {mse:.4f}')
plt.scatter(x_test, y_test, color='black', label='Actual')
plt.plot(x_test, y_pred, color='blue', linewidth=3,
         label='predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Scatter plot of Actual Vs Predicted Value
          (simple Linear Regression)')
plt.legend()
plt.show()
```

Output:-

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex : 100 entries, 0 to 99
Data columns (total 2 columns):
```

#	column	Non-Null count	Dtype
0	x	100 non-null	float64
1	y	100 non-null	float64

```
dtypes : float64(2)
memory usage : 1.7 KB
None
Model : "Sequential"
```

Layer(type)	Output shape	Param#

dense-2 (Dense) (None, 6) 12

dense-3 (Dense) (None, 1) 7

Total params : 19 (0.00 Byte)

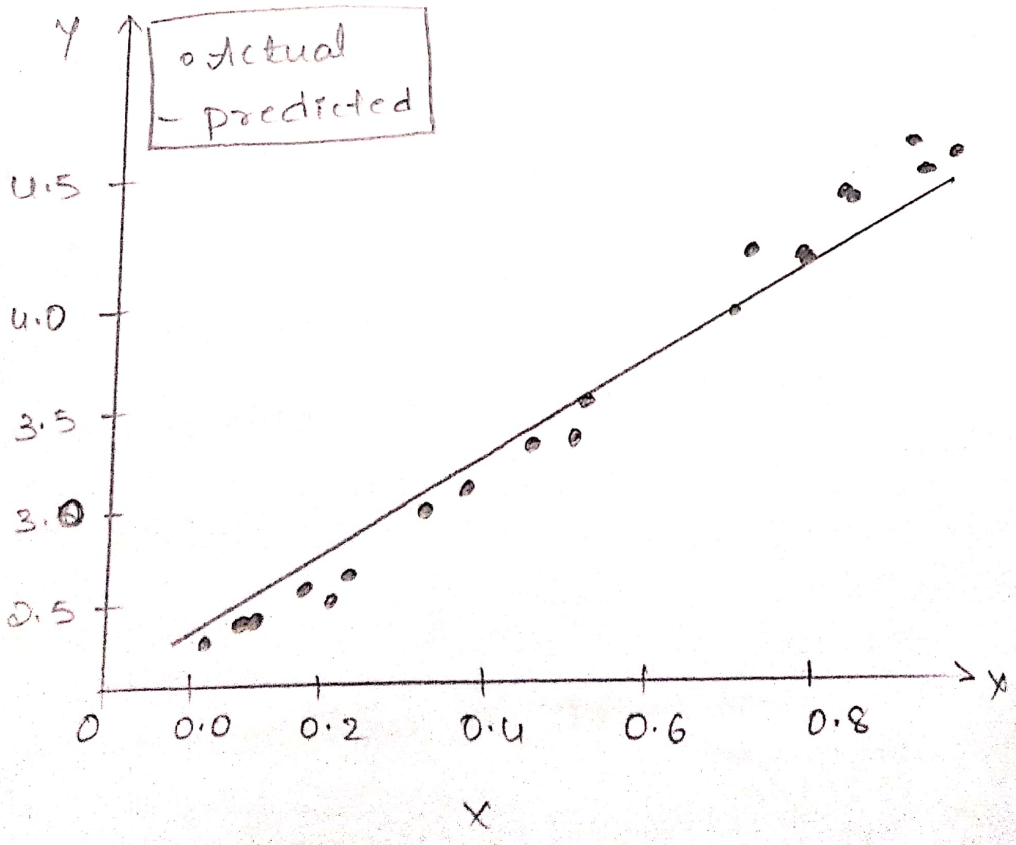
Trainable params : 19 (0.00 Byte)

Non-trainable params : 0 (0.00 Byte)

1/1 [==========] - 0s 355ms/step

Mean squared Error on Test set : 0.0181

Scatter plot by Actual Vs predicted values (Simple Linear Regression)



ExpNO:- 03 Date:-

Aim:- To implement a MLP using Keras with Tensor flow for classification problem (heart disease predication)

Description:-

This keras implementation with Tensorflow backend defines a Multilayer perceptron (MLP) for a classification task. The model. comprised input, hidden, and output layers, utilizing densely connected neurons. Activation function like RELU in hidden layer and softmax in the output layer enable non-linearity and probability distribution for multiple classes

program:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Flatten,
                                    Dropout
from sklearn.model_selection import train_test_
                                    split
from sklearn.preprocessing import MinMaxScaler
sns.set()
tf._version_
data = pd.read_csv("c://Desktop//DL//heart.csv")
x = data.drop('Heart Disease', axis = 1)
y = data['Heart Disease']
x = pd.get_dummies(x, columns=['sex', 'chestPainType'
'fasting Bs', 'Resting ECG', 'Exercise Angina', 'ST_slope'])
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size = 0.25), random_state
                                    = 42)
scaler = MinMaxScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
model = Sequential([
Dense(300, activation = 'relu', input_shape = (x_train.
                                    shape[1],)),
Dense(150, activation = 'relu'),
Dense(75, activation = 'relu'),
Dense(1, activation = 'sigmoid')])
```

```
model.compile(optimizer='adam', loss='binary-cross
                entropy', metrics=['accuracy'])
y_train_binary = y_train.astype('float32')
history = model.fit(x_train_scaled, y_train_
                binary epochs=10, verbose=2)

model.summary()
print("List of layers:", model.layers)
print("In Name of the second layer:", model.layers
                [1].name)

hidden2 = model.layers[2]
weights, bias = hidden2.get_weights()
print("weights:", weights)
print("Bias:", bias)
pd.DataFrame(history.history).plot()
plt.xlabel("Epoch")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2)
plt.show()
loss, acc = model.evaluate(x_test_scaled, y_test)
print('Test loss:', loss)
print("Test Accuracy:", acc)
```

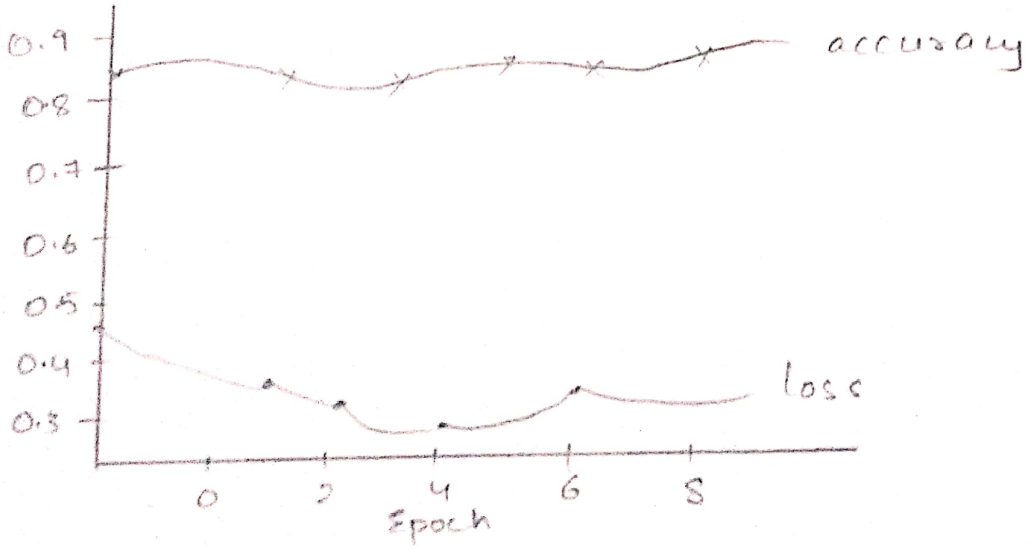
Output:-

list of layers: [<keras.src.layers.core.dense.Dense object at 0x000002AFF903C4FF90>, <keras.src.layers.core.dense.Dense object at 0x000002AFF903CA6A90>, <keras.src.layers.core.dense.Dense object at 0x000002AFFC3A2B310>]

Name of the second layer: Dense_1

weights: [[0.06995429 -0.11436618 0.14342096 ... -0.12740202
0.13114259 -0.05110514]
:
[0.03726638 0.15389721 0.13552894 ... 0.05107746
-0.04302126 -0.07474063]]

Bias: [-1.13932737e-02 6.22983277e-03 ... 0.0000000e+00]



8/8 [==========] - 0s 3ms/step - loss: 0.3098 - acc: 0.8696

Test loss: 0.30976164340972900

Test Accuracy: 0.8695651888847351