##################### Binary Classification: ##############################

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.datasets import make\_blobs

X,y=make\_blobs(n\_samples=400,n\_features=2,

               centers=2,random\_state=100)

X

plt.scatter(X[:,0],X[:,1],c=y);

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,

                                  test\_size=0.2,random\_state=100)

X\_train.shape,X\_test.shape,y\_train.shape,y\_test.shape

### Building The Model ####

model\_1=tf.keras.Sequential()

model\_1.add(tf.keras.layers.Dense(8,activation='relu'))

model\_1.add(tf.keras.layers.Dense(1,activation='sigmoid'))

#### Compiling the model ####

model\_1.compile(optimizer=tf.keras.optimizers.Adam(),

                loss=tf.keras.losses.BinaryCrossentropy(),

                metrics=['accuracy'])

#### Training the Model ####

tf.random.set\_seed(100)

history=model\_1.fit(X\_train,y\_train,epochs=50)

hist=pd.DataFrame(history.history)

hist.plot();

### Model Summary ###

model\_1.summary()

### Plotting the Model ###

from tensorflow.keras.utils import plot\_model

plot\_model(model\_1,show\_shapes=True)

### Prediction ###

model\_1.predict(X\_test)

############# Regression using Feed Forward Neural Network ############

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from sklearn.datasets import fetch\_california\_housing

house=fetch\_california\_housing()

house

house['data']

house['data'].shape

house['feature\_names']

# Feature set

X=pd.DataFrame(house['data'],columns=house['feature\_names'])

X

# Target

house['target']

house['target\_names']

y=pd.DataFrame(house['target'],columns=house['target\_names'])

y

### Standardization ###

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

X\_scaled=sc.fit\_transform(X)

X\_scaled

### Train test split ###

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X\_scaled,

                      y, test\_size=0.2, random\_state=100)

X\_train.shape,X\_test.shape,y\_train.shape,y\_test.shape

### Building the model ###

model\_1=tf.keras.Sequential()

model\_1.add(tf.keras.layers.Dense(20,activation='relu'))

model\_1.add(tf.keras.layers.Dense(1))

### Compiling the model ###

model\_1.compile(optimizer=tf.keras.optimizers.Adam(),

                loss=tf.keras.losses.MeanSquaredError(),

                metrics=['mae'])

### Training the model ###

tf.random.set\_seed(100)

history=model\_1.fit(X\_train,y\_train,epochs=100)

hist=pd.DataFrame(history.history)

hist.plot();

### Model Summary ###

model\_1.summary()

### Plotting the model ###

from tensorflow.keras.utils import plot\_model

plot\_model(model\_1, show\_shapes=True)

### Evaluation ###

test\_mse,test\_mae=model\_1.evaluate(X\_test,y\_test)

print('Test MSE:',test\_mse)

print('Train MSE:0.3076' )

print('Test MAE:', test\_mae)

print('Train MAE:0.3835 ')

### prediction using the model ###

y\_pred=model\_1.predict(X\_test)

print('The predicted house prices:\n',y\_pred)

y\_test

##################### Multiclass Classification #######################

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

### Dataset ###

from tensorflow.keras.datasets import fashion\_mnist

(X\_train,y\_train),(X\_test,y\_test)=fashion\_mnist.load\_data()

X\_train.shape,y\_train.shape,X\_test.shape,y\_test.shape

### Visualization of the dataset ###

plt.imshow(X\_train[0],'Greys')

plt.title(y\_train[0]);

# Creating a dictionary of nos and respective fashion items

fashion\_dict={0:'T-shirt/top',

1:'Trouser',

2:'Pullover',

3:'Dress',

4:'Coat',

5:'Sandal',

6:'Shirt',

7:'Sneaker',

8:'Bag',

9:'Ankle boot'}

fashion\_dict

plt.imshow(X\_train[0],'Greys')

plt.title(fashion\_dict[y\_train[0]]);

plt.imshow(X\_train[10],'Greys')

plt.title(fashion\_dict[y\_train[10]]);

plt.imshow(X\_train[100],'Greys')

plt.title(fashion\_dict[y\_train[100]]);

plt.imshow(X\_train[500],'Greys')

plt.title(fashion\_dict[y\_train[500]]);

plt.imshow(X\_train[1000],'Greys')

plt.title(fashion\_dict[y\_train[1000]]);

plt.imshow(X\_train[3000],'Greys')

plt.title(fashion\_dict[y\_train[3000]]);

plt.imshow(X\_train[5000],'Greys')

plt.title(fashion\_dict[y\_train[5000]]);

plt.imshow(X\_train[15000],'Greys')

plt.title(fashion\_dict[y\_train[15000]]);

plt.imshow(X\_train[45000],'Greys')

plt.title(fashion\_dict[y\_train[45000]]);

### Building the model ###

fashion\_1=tf.keras.Sequential()

fashion\_1.add(tf.keras.layers.Flatten())

fashion\_1.add(tf.keras.layers.Dense(300,activation='relu'))

fashion\_1.add(tf.keras.layers.Dense(10,activation='softmax'))

### Compiling the model ###

fashion\_1.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                  optimizer=tf.keras.optimizers.Adam(),

                  metrics=['accuracy'])

### Training the model ###

tf.random.set\_seed(100)

history\_1=fashion\_1.fit(X\_train,y\_train,epochs=25)

pd.DataFrame(history\_1.history).plot();

### Evaluation of the model ###

test\_loss,test\_accuracy=fashion\_1.evaluate(X\_test,y\_test)

print('The Test Loss:',test\_loss)

print('The Test Accuracy:',test\_accuracy)

### Improving the model by adding one more hidden layer ###

fashion\_2=tf.keras.Sequential()

fashion\_2.add(tf.keras.layers.Flatten())

fashion\_2.add(tf.keras.layers.Dense(300,activation='relu'))

## Adding a layer of 100 units

fashion\_2.add(tf.keras.layers.Dense(100,activation='relu'))

fashion\_2.add(tf.keras.layers.Dense(10,activation='softmax'))

fashion\_2.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                  optimizer=tf.keras.optimizers.Adam(),

                  metrics=['accuracy'])

tf.random.set\_seed(100)

history\_2=fashion\_2.fit(X\_train,y\_train,epochs=25)

pd.DataFrame(history\_2.history).plot();

test\_loss,test\_accuracy=fashion\_2.evaluate(X\_test,y\_test)

print('Test Loss:',test\_loss)

print('Test accuracy:',test\_accuracy)

### Adding one more hidden layer ###

fashion\_3=tf.keras.Sequential()

fashion\_3.add(tf.keras.layers.Flatten())

fashion\_3.add(tf.keras.layers.Dense(300,activation='relu'))

fashion\_3.add(tf.keras.layers.Dense(100,activation='relu'))## Adding a layer of 100 units

# Adding a layer of 25 units

fashion\_3.add(tf.keras.layers.Dense(25,activation='relu'))

## Adding a layer of 10 units

fashion\_3.add(tf.keras.layers.Dense(10,activation='softmax'))

fashion\_3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                  optimizer=tf.keras.optimizers.Adam(),

                  metrics=['accuracy'])

tf.random.set\_seed(100)

history\_3=fashion\_3.fit(X\_train,y\_train,epochs=25)

pd.DataFrame(history\_3.history).plot();

### Improving the model by changing the number of units ###

fashion\_4=tf.keras.Sequential()

fashion\_4.add(tf.keras.layers.Flatten())

# Changing the no of units

fashion\_4.add(tf.keras.layers.Dense(200,activation='relu'))

fashion\_4.add(tf.keras.layers.Dense(100,activation='relu'))## Adding a layer of 100 units

fashion\_4.add(tf.keras.layers.Dense(10,activation='softmax'))

fashion\_4.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                  optimizer=tf.keras.optimizers.Adam(),

                  metrics=['accuracy'])

tf.random.set\_seed(100)

history\_4=fashion\_4.fit(X\_train,y\_train,epochs=25)

pd.DataFrame(history\_4.history).plot();

test\_loss,test\_accuracy=fashion\_4.evaluate(X\_test,y\_test)

print('Test Loss:',test\_loss)

print('Test Accuracy:',test\_accuracy)

### Model Summary ###

fashion\_2.summary()

### Plotting the model ###

from tensorflow.keras.utils import plot\_model

plot\_model(fashion\_2,show\_shapes=True)

### Prediction ###

y\_pred=fashion\_2.predict(X\_test)

y\_pred

y\_pred[0]

np.argmax(y\_pred[0])

plt.imshow(X\_test[0],'Greys')

plt.title(fashion\_dict[np.argmax(y\_pred[0])]);

plt.imshow(X\_test[20],'Greys')

plt.title(fashion\_dict[np.argmax(y\_pred[20])]);

################ Hyper parameter tuning and Regularisation ############

Hyper parameters

* No of layers
* No of units
* Activation function
* Loss function
* Optimizer
* Dropout rate
* Learning rate
* Epoch

No of units:

* Output layer:
  + Regression: One
  + Binary classiifcation: One/ Two depends upon activation function
  + Multi-class classification: No of classes.
* Input layer: Depends upon the no of features/values in the data point
* Hidden layer: Depends, generally between 10 and 100.

Activation function

* Output layer
  + Regression: Linear
  + Binary classification: sigmoid (if the class labels are 0 and 1)

                   tanh (if the class labels are -1 and 1)

* + Multi-class classiifcation: softmax.
* Hidden layer: 'relu', in general.

Loss function

* Regression: MAE, MSE(more preferred)
* Binary classification: BinaryCrossentropy
* Multi-class Classification: Categorical Crossentropy /

                        Sparse Categorical Crossentropy

Optimizer

* Adam, in general

### Regularization ###

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

### Accessing the dataset ###

from tensorflow.keras.datasets import mnist

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

X\_train.shape,y\_train.shape,X\_test.shape,y\_test.shape

X\_train[0]

### Visualization ###

plt.imshow(X\_train[0],'gray')

plt.title(y\_train[0]);

plt.imshow(X\_train[10],'gray')

plt.title(y\_train[10]);

plt.imshow(X\_train[100],'gray')

plt.title(y\_train[100]);

plt.imshow(X\_train[1000],'gray')

plt.title(y\_train[1000]);

plt.imshow(X\_train[10000],'gray')

plt.title(y\_train[10000]);

plt.imshow(X\_train[40000],'gray')

plt.title(y\_train[40000]);

pd.DataFrame(y\_train).value\_counts()

### Building basic Model ###

model\_1=tf.keras.Sequential()

model\_1.add(tf.keras.layers.Flatten())

model\_1.add(tf.keras.layers.Dense(300, activation='relu'))

model\_1.add(tf.keras.layers.Dense(10,activation='softmax'))

# compiling the model

model\_1.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training the model

tf.random.set\_seed(100)

hist\_1=model\_1.fit(X\_train,y\_train,epochs=10)

pd.DataFrame(hist\_1.history).plot();

# Evaluation of the model

model\_1.evaluate(X\_test,y\_test)

### Model with normalization ###

# Normalization

normalizer=tf.keras.layers.Normalization()

normalizer.adapt(X\_train)

normalizer.adapt(X\_test)

X\_train=normalizer(X\_train)

X\_test=normalizer(X\_test)

X\_train[0]

model\_2=tf.keras.Sequential()

model\_2.add(tf.keras.layers.Flatten())

model\_2.add(tf.keras.layers.Dense(300, activation='relu'))

model\_2.add(tf.keras.layers.Dense(10,activation='softmax'))

model\_2.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training the model

tf.random.set\_seed(100)

# X\_train is the normalosed training set

hist\_2=model\_2.fit(X\_train,y\_train,epochs=10)

pd.DataFrame(hist\_2.history).plot();

model\_2.evaluate(X\_test,y\_test)

### Model with Cross Validation ###

model\_3=tf.keras.Sequential()

model\_3.add(tf.keras.layers.Flatten())

model\_3.add(tf.keras.layers.Dense(300, activation='relu'))

model\_3.add(tf.keras.layers.Dense(10,activation='softmax'))

model\_3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training the model

tf.random.set\_seed(100)

# validation\_split

hist\_3=model\_3.fit(X\_train,y\_train,epochs=10,validation\_split=0.2)

pd.DataFrame(hist\_3.history).plot();

model\_3.evaluate(X\_test,y\_test)

############################ Regularization ##########################

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

### Accessing the Dataset ###

from tensorflow.keras.datasets.mnist import load\_data

(X\_train,y\_train),(X\_test,y\_test)=load\_data()

X\_train.shape,y\_train.shape,X\_test.shape,y\_test.shape

### Visualization ###

plt.imshow(X\_train[0],'gray')

plt.title(y\_train[0]);

plt.imshow(X\_train[50],'gray')

plt.title(y\_train[50]);

### Building Basic Model ###

model\_1=tf.keras.Sequential()

model\_1.add(tf.keras.layers.Flatten())

model\_1.add(tf.keras.layers.Dense(300,activation='relu'))

model\_1.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_1.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training

tf.random.set\_seed(100)

hist\_1=model\_1.fit(X\_train,y\_train,epochs=10)

pd.DataFrame(hist\_1.history).plot();

print(' Evaluation Result:\n')

model\_1.evaluate(X\_test,y\_test)

### Model with Normalization ###

normaliser=tf.keras.layers.Normalization()

normaliser.adapt(X\_train)

normaliser.adapt(X\_test)

X\_train=normaliser(X\_train)

X\_test=normaliser(X\_test)

X\_train[0]

model\_2=tf.keras.Sequential()

model\_2.add(tf.keras.layers.Flatten())

model\_2.add(tf.keras.layers.Dense(300,activation='relu'))

model\_2.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_2.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training

tf.random.set\_seed(100)

hist\_2=model\_2.fit(X\_train,y\_train,epochs=10)

pd.DataFrame(hist\_2.history).plot();

print(' Evaluation Result:\n')

model\_2.evaluate(X\_test,y\_test)

### Model Validation ###

model\_3=tf.keras.Sequential()

model\_3.add(tf.keras.layers.Flatten())

model\_3.add(tf.keras.layers.Dense(300,activation='relu'))

model\_3.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training

tf.random.set\_seed(100)

## Validation split

hist\_3=model\_3.fit(X\_train,y\_train,epochs=10, validation\_split=0.2)

pd.DataFrame(hist\_3.history).plot();

print(' Evaluation Result:\n')

model\_3.evaluate(X\_test,y\_test)

### Model Dropout ###

model\_4=tf.keras.Sequential()

model\_4.add(tf.keras.layers.Flatten())

model\_4.add(tf.keras.layers.Dense(300,activation='relu'))

# Dropout layer

model\_4.add(tf.keras.layers.Dropout(0.1)) # 10% of units will be dropped

model\_4.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_4.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Training

tf.random.set\_seed(100)

## Validation split

hist\_4=model\_4.fit(X\_train,y\_train,epochs=10, validation\_split=0.2)

pd.DataFrame(hist\_4.history).plot();

print(' Evaluation Result:\n')

model\_4.evaluate(X\_test,y\_test)

### Model with Early Stop ###

model\_5=tf.keras.Sequential()

model\_5.add(tf.keras.layers.Flatten())

model\_5.add(tf.keras.layers.Dense(300,activation='relu'))

model\_5.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_5.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Introducing early stop

early\_stop=tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=2)

# Training

tf.random.set\_seed(100)

## Validation split,callbacks

hist\_5=model\_5.fit(X\_train,y\_train,epochs=10, validation\_split=0.2,callbacks=[early\_stop])

pd.DataFrame(hist\_5.history).plot();

print(' Evaluation Result:\n')

model\_5.evaluate(X\_test,y\_test)

### Model with Batch Normalization ###

model\_6=tf.keras.Sequential()

model\_6.add(tf.keras.layers.Flatten())

model\_6.add(tf.keras.layers.Dense(300,activation='relu'))

# Batch Normalisation

model\_6.add(tf.keras.layers.BatchNormalization())

model\_6.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_6.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Introducing early stop

early\_stop=tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=2)

# Training

tf.random.set\_seed(100)

## Validation split,callbacks

hist\_6=model\_6.fit(X\_train,y\_train,epochs=10, validation\_split=0.2,callbacks=[early\_stop])

pd.DataFrame(hist\_6.history).plot();

print(' Evaluation Result:\n')

model\_6.evaluate(X\_test,y\_test)

### Model with 2 hidden layers ###

model\_7=tf.keras.Sequential()

model\_7.add(tf.keras.layers.Flatten())

model\_7.add(tf.keras.layers.Dense(300,activation='relu'))

## Adding one more layer

model\_7.add(tf.keras.layers.Dense(100,activation='relu'))

model\_7.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_7.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Introducing early stop

early\_stop=tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=2)

# Training

tf.random.set\_seed(100)

## Validation split,callbacks

hist\_7=model\_7.fit(X\_train,y\_train,epochs=10, validation\_split=0.2,callbacks=[early\_stop])

pd.DataFrame(hist\_7.history).plot();

print(' Evaluation Result:\n')

model\_7.evaluate(X\_test,y\_test)

### model with changed number of units ###

model\_8=tf.keras.Sequential()

model\_8.add(tf.keras.layers.Flatten())

# Change the no of units to 50

model\_8.add(tf.keras.layers.Dense(50,activation='relu'))

model\_8.add(tf.keras.layers.Dense(10,activation='softmax'))

# Compiling the model

model\_8.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),

                optimizer=tf.keras.optimizers.Adam(),

                metrics=['accuracy'])

# Introducing early stop

early\_stop=tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=2)

# Training

tf.random.set\_seed(100)

## Validation split,callbacks

hist\_8=model\_8.fit(X\_train,y\_train,epochs=10, validation\_split=0.2,callbacks=[early\_stop])

pd.DataFrame(hist\_8.history).plot();

print(' Evaluation Result:\n')

model\_8.evaluate(X\_test,y\_test)

### Saving the model ###

model\_8.save('best\_mnist\_model.h5')

### Loading the model ###

my\_mnist=tf.keras.models.load\_model('best\_mnist\_model.h5')

my\_mnist.evaluate(X\_test,y\_test)

##################### Lecture 10 RNN #############################

### Stock market prediction using LSTM ###

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

### Dataset Access ###

adani=pd.read\_csv('https://query1.finance.yahoo.com/v7/finance/download/ADANIPOWER.NS?period1=1646742145&period2=1678278145&interval=1d&events=history&includeAdjustedClose=true')

adani

# Use 'Close'

data=adani['Close']

data

### Train test Split ###

train=data[:200]

test=data[200:]

train

test

### Visualization ###

plt.figure(figsize=(15,6))

plt.plot(train)

plt.plot(test)

plt.xlabel('Market day')

plt.ylabel('Stock Price')

plt.legend(['Train','Test']);

### Standardization ###

from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler()

train\_scaled=scaler.fit\_transform(np.array(train).reshape(-1,1))

test\_scaled=scaler.fit\_transform(np.array(test).reshape(-1,1))

train\_scaled

### Converting Test scaled to regression data ###

X\_train=[]

y\_train=[]

# window\_size=10

for i in range(10,200):

  X\_train.append(train\_scaled[i-10:i,0])

  y\_train.append(train\_scaled[i])

X\_train

y\_train

# Converting to np.array

X\_train=np.array(X\_train)

y\_train=np.array(y\_train)

X\_train.shape

y\_train.shape

# Converting text data

X\_test=[]

y\_test=[]

# window\_size=10

for i in range(10,50):

  X\_test.append(test\_scaled[i-10:i,0])

  y\_test.append(test\_scaled[i])

X\_test=np.array(X\_test)

y\_test=np.array(y\_test)

X\_test.shape

y\_test.shape

### Model Building ###

lstm=tf.keras.Sequential()

lstm.add(tf.keras.layers.LSTM(50,return\_sequences=True,

                              input\_shape=(X\_train.shape[1],1))) # (10,1)

lstm.add(tf.keras.layers.Dense(1))

# Compiling

lstm.compile(loss=tf.keras.losses.MeanSquaredError(),

             optimizer=tf.keras.optimizers.Adam(),

             metrics=['mae'])

# Training

tf.random.set\_seed(10)

hist=lstm.fit(X\_train,y\_train,epochs=100)

pd.DataFrame(hist.history).plot();

### Evaluation of the model ###

lstm.evaluate(X\_test,y\_test)

lstm.summary()