Perceptron Model

In [ ]:
```
import numpy as np
import pandas as pd
import sklearn
from sklearn.datasets import load_digits
```

In [ ]:
```
digits = load_digits()
digits
```

Out[2]:
```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
```

In [ ]:
```
digits['data']
```

Out[3]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [ ]:
```
digits['target']
```

Out[4]:
```
array([0, 1, 2, ..., 8, 9, 8])
```

In [ ]:
```
x = digits.data
y = digits.target
```

In [ ]:
```
#x = x/255.0
```

In [ ]:
```
# split the data into training and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, r
```

In [ ]:
```
# Building perceptron model
from sklearn.linear_model import Perceptron
```

In [ ]:
```
perceptron_model = Perceptron()
perceptron_model.fit(x_train, y_train)
```

Out[9]:
```
Perceptron()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```
print("What is the Traingin Accuracy of Perceptron Model :")
print(perceptron_model.score(x_train, y_train))

print("What is the Test Accuracy of Perceptron Model :")
print(perceptron_model.score(x_test, y_test))

predicted_train = perceptron_model.predict(x_train)
predicted_test = perceptron_model.predict(x_test)

from sklearn.metrics import accuracy_score, classification_report, confusio
print("Training Accuracy", accuracy_score(y_train, predicted_train))
print("Test Accuracy", accuracy_score(y_test, predicted_test))

print("classification_report Training")
print(classification_report(y_train, predicted_train))
print("classification_report Test")
print(classification_report(y_test, predicted_test))
```

```
What is the Traingin Accuracy of Perceptron Model :
0.9762435040831478
What is the Test Accuracy of Perceptron Model :
0.9688888888888889
Training Accuracy 0.9762435040831478
Test Accuracy 0.9688888888888889
classification_report Training
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       125
           1       0.95      0.99      0.97       140
           2       0.98      1.00      0.99       136
           3       0.98      0.98      0.98       131
           4       1.00      0.99      1.00       134
           5       0.93      1.00      0.97       143
           6       0.98      0.99      0.98       138
           7       0.98      1.00      0.99       131
           8       1.00      0.85      0.92       137
           9       0.99      0.97      0.98       132

    accuracy                           0.98      1347
   macro avg       0.98      0.98      0.98      1347
weighted avg       0.98      0.98      0.98      1347

classification_report Test
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       0.98      0.98      0.98        42
           2       0.91      0.98      0.94        41
           3       0.98      0.96      0.97        52
           4       0.98      1.00      0.99        47
           5       0.91      1.00      0.95        39
           6       1.00      1.00      1.00        43
           7       1.00      0.98      0.99        48
           8       1.00      0.81      0.90        37
           9       0.94      0.96      0.95        48

    accuracy                           0.97       450
   macro avg       0.97      0.97      0.97       450
weighted avg       0.97      0.97      0.97       450
```

In [ ]:

In [ ]:

Regression Problem

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
```

In [ ]:
```
data = pd.read_csv("/content/boston.csv")
data.head()
```

Out[23]:

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	I
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	

In [ ]:
```
data.isnull().sum()
```

Out[24]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NX         0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [ ]:
```
x = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

In [ ]:
```
x.head()
```

Out[26]:

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	I
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
pd.DataFrame(x).head()
```

Out[27]:

	0	1	2	3	4	5	6	7	
0	-0.419782	0.284830	-1.287909	-0.272599	-0.144217	0.413672	-0.120013	0.140214	-0.982
1	-0.417339	-0.487722	-0.593381	-0.272599	-0.740262	0.194274	0.367166	0.557160	-0.867
2	-0.417342	-0.487722	-0.593381	-0.272599	-0.740262	1.282714	-0.265812	0.557160	-0.867
3	-0.416750	-0.487722	-1.306878	-0.272599	-0.835284	1.016303	-0.809889	1.077737	-0.752
4	-0.412482	-0.487722	-1.306878	-0.272599	-0.835284	1.228577	-0.511180	1.077737	-0.752

In [ ]:
```
y.head()
```

Out[28]:
```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: MEDV, dtype: float64
```

In [ ]:
```
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, ran
```

In [ ]:
```
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape )
```

```
(404, 13) (404,) (102, 13) (102,)
```

MultiLayer Perceptron Model - DNN

In [ ]:
```
dnn = Sequential()
# add hidden layer 1
dnn.add(Dense(16, activation='relu', input_dim = 13))
# batch normalization
#dnn.add(BatchNormalization())
# dropout
#dnn.add(Dropout(0.25))
# add one more hidden layer
#dnn.add(Dense(8, activation='relu'))
# batch normalization
#dnn.add(BatchNormalization())
# dropout
#dnn.add(Dropout(0.5))
# output
dnn.add(Dense(1, activation='linear'))
# summary
dnn.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 8)                 112

 batch_normalization_8 (Bat  (None, 8)                 32
 chNormalization)

 dropout_6 (Dropout)         (None, 8)                 0

 dense_16 (Dense)            (None, 8)                 72

 batch_normalization_9 (Bat  (None, 8)                 32
 chNormalization)

 dropout_7 (Dropout)         (None, 8)                 0

 dense_17 (Dense)            (None, 1)                 9

=================================================================
Total params: 257 (1.00 KB)
Trainable params: 225 (900.00 Byte)
Non-trainable params: 32 (128.00 Byte)
_________________________________________________________________
```

In [ ]:
```
# compile the model while using optimization and loss function
dnn.compile(optimizer='sgd', loss='mean_squared_error')

# Fit the model
history = dnn.fit(x_train, y_train, epochs=500)
```

```
Epoch 1/500
13/13 [==============================] - 6s 6ms/step - loss: 263.7253
Epoch 2/500
13/13 [==============================] - 0s 4ms/step - loss: 109.8043
Epoch 3/500
13/13 [==============================] - 0s 4ms/step - loss: 83.4480
Epoch 4/500
13/13 [==============================] - 0s 4ms/step - loss: 77.0359
Epoch 5/500
13/13 [==============================] - 0s 4ms/step - loss: 67.9864
Epoch 6/500
13/13 [==============================] - 0s 4ms/step - loss: 62.2285
Epoch 7/500
13/13 [==============================] - 0s 4ms/step - loss: 67.8151
Epoch 8/500
13/13 [==============================] - 0s 5ms/step - loss: 69.2439
Epoch 9/500
13/13 [==============================] - 0s 4ms/step - loss: 70.1299
Epoch 10/500
```

In [ ]:
```
# Predict the test dataset
y_pred = dnn.predict(x_test)
y_pred
```

```
4/4 [==============================] - 0s 3ms/step
```

Out[38]:
```
array([[42.066048 ],
       [28.673164 ],
       [16.047989 ],
       [16.490643 ],
       [28.134983 ],
       [36.2645   ],
       [42.893303 ],
       [11.216604 ],
       [31.150734 ],
       [ 9.252087 ],
       [24.992077 ],
       [15.366667 ],
       [20.12103  ],
       [20.722178 ],
       [22.332483 ],
       [21.97967  ],
       [11.298435 ],
       [33.14695  ],
       [25.301653 ],
       [23.739754 ],
       [13.843197 ],
       [20.791662 ],
       [21.458355 ],
       [29.901642 ],
       [36.99926  ],
       [17.445225 ],
       [27.09676  ],
       [17.777788 ],
       [25.060944 ],
       [34.022118 ],
       [21.52371  ],
       [19.667759 ],
       [39.86638  ],
       [42.36988  ],
       [30.88499  ],
       [22.008015 ],
       [15.616453 ],
       [17.78113  ],
       [ 7.3795614],
       [26.07758  ],
       [22.365658 ],
       [20.86047  ],
       [41.318607 ],
       [15.815165 ],
       [21.978218 ],
       [22.592115 ],
       [30.622635 ],
       [16.55214  ],
       [24.264685 ],
       [23.215982 ],
       [37.48295  ],
       [43.706207 ],
       [21.787628 ],
       [17.050863 ],
       [32.200417 ],
       [ 8.377619 ],
       [20.678692 ],
       [17.705532 ],
       [21.615456 ],
       [20.649569 ],
       [33.85414  ],
```

```
[ 9.616957 ],
[48.48142  ],
[20.65918  ],
[12.436285 ],
[22.0219   ],
[23.070957 ],
[20.644348 ],
[14.936278 ],
[20.5262   ],
[21.847637 ],
[21.954939 ],
[20.8123   ],
[19.319645 ],
[24.192919 ],
[20.01855  ],
[40.93     ],
[12.7487755],
[25.183353 ],
[15.0503845],
[16.093025 ],
[18.678638 ],
[28.624542 ],
[14.727659 ],
[13.854645 ],
[20.466852 ],
[21.5724   ],
[29.666527 ],
[22.34641  ],
[21.232998 ],
[14.41136  ],
[13.176874 ],
[25.100262 ],
[34.81701  ],
[ 7.779066 ],
[41.445396 ],
[12.45356  ],
[33.16234  ],
[ 7.682678 ],
[21.435665 ],
[37.624245 ],
[20.483479 ]], dtype=float32)
```

In [ ]:
```
print(y_pred[0:10])
print("************"*5)
print(y_test[0:10])
```

```
[[42.066048]
 [28.673164]
 [16.047989]
 [16.490643]
 [28.134983]
 [36.2645  ]
 [42.893303]
 [11.216604]
 [31.150734]
 [ 9.252087]]
************************************************************
195    50.0
4      36.2
434    11.7
458    14.9
39     30.8
304    36.1
225    50.0
32     13.2
157    41.3
404     8.5
Name: MEDV, dtype: float64
```

In [ ]:
```
from sklearn.metrics import r2_score
```

In [ ]:
```
print(r2_score(y_test, y_pred))
```

```
0.742262016726801
```

In [ ]:

In [ ]:

Classification Problem

In [ ]:
```
import pandas as pd
dataset = pd.read_csv("/content/Churn_Modelling.csv")
dataset.head()
```

Out[2]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8380
2	3	15619304	Onio	502	France	Female	42	8	15966
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12551

In [ ]:
```
dataset['Exited'].value_counts()
```

Out[3]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [ ]:
```
dataset = dataset.iloc[:,3:]
dataset.head()
```

Out[4]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

In [ ]:
```
dataset.isnull().sum()
```

Out[61]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [ ]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        10000 non-null  object 
 2   Gender           10000 non-null  object 
 3   Age              10000 non-null  int64  
 4   Tenure           10000 non-null  int64  
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

In [ ]:
```
dataset = pd.get_dummies(dataset, columns=['Geography', 'Gender'], drop_fir
```

In [ ]:
```
dataset.head()
```

Out[64]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
0	619	42	2	0.00	1	1	1	
1	608	41	1	83807.86	1	0	1	
2	502	42	8	159660.80	3	1	0	
3	699	39	1	0.00	2	0	0	
4	850	43	2	125510.82	1	1	1	

In [ ]:
```
# split the data into ind and dep variable
x = dataset.drop(['Exited'], axis=1)
y = dataset[['Exited']]
```

In [ ]:
```
x.shape
```

Out[71]:
```
(10000, 11)
```

In [ ]:
```
x.head()
```

Out[66]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
0	619	42	2	0.00	1	1	1	
1	608	41	1	83807.86	1	0	1	
2	502	42	8	159660.80	3	1	0	
3	699	39	1	0.00	2	0	0	
4	850	43	2	125510.82	1	1	1	

In [ ]:
```
y.head()
```

Out[67]:

	Exited
0	1
1	0
2	1
3	0
4	0

In [ ]:
```
# Balance the data
from imblearn.over_sampling import SMOTE
smote = SMOTE()
x_smote, y_smote = smote.fit_resample(x, y)
print(y.value_counts())
print(y_smote.value_counts())
```

```
Exited
0         7963
1         2037
dtype: int64
Exited
0         7963
1         7963
dtype: int64
```

In [ ]:
```
# split the data into train and test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_smote, y_smote, test_
```

In [ ]:
```
# feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

MLP

In [ ]:
```
model = Sequential()
model.add(Dense(32, activation='relu', kernel_regularizer = keras.regulariz
                kernel_initializer='he_normal',  input_dim=11))
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(32, activation='relu',  kernel_regularizer = keras.regulari
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 32)                384

 dropout_8 (Dropout)         (None, 32)                0

 batch_normalization_10 (Ba  (None, 32)                128
 tchNormalization)

 dense_19 (Dense)            (None, 32)                1056

 dropout_9 (Dropout)         (None, 32)                0

 batch_normalization_11 (Ba  (None, 32)                128
 tchNormalization)

 dense_20 (Dense)            (None, 1)                 33

=================================================================
Total params: 1729 (6.75 KB)
Trainable params: 1601 (6.25 KB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
```

In [ ]:

In [ ]:
```
initial_weights = model.get_weights()
```

In [ ]:
```
initial_weights[0]
```

Out[79]:
```
array([[-0.54668975, -0.65129626, -0.34773925, -0.43765387, -0.13219708,
         0.40628335,  0.5963399 , -0.13944611, -0.67357314,  0.46777704,
         0.08202546,  0.09328174, -0.4894623 , -0.46160665, -0.2967626 ,
         0.43073124, -0.0424548 ,  0.3775512 , -0.16614582, -0.88848394,
         0.25903097, -0.3694219 ,  0.11888529, -0.3252677 , -0.90093344,
         0.258748  , -0.19535325,  0.03168226, -0.43863618, -0.22901714,
        -0.37371907, -0.00289603],
       [-0.33343717,  0.34203184, -0.1816835 , -0.5410051 ,  0.42010593,
         0.03782547, -0.9005688 ,  0.42557123,  0.33136138, -0.14479339,
         0.17061973,  0.572867  ,  0.53384614, -0.16065665,  0.09030482,
        -0.43708202, -0.05674174, -0.9229993 ,  0.01270335, -0.13720037,
        -0.21643294, -0.0176984 , -0.3769747 ,  0.0724934 ,  0.52268326,
         0.11909305,  0.3787435 , -0.11065734,  0.11946512,  0.35163096,
        -0.3049398 , -0.17288749],
       [-0.6707052 ,  0.19441552,  0.85444504,  0.09757277, -0.24736226,
         0.30902478, -0.9635952 ,  0.5798555 ,  0.07710464,  0.3685232 ,
         0.30412   ,  0.30889067, -0.37271994,  0.0338873 , -0.03606715,
        -0.5522396 ,  0.20524049, -0.12220731,  0.15795179, -0.11916168,
         0.81712294, -0.15985255,  0.43719903, -0.0882709 ,  0.16368422,
        -0.4094959 ,  0.46604523,  0.87715536, -0.9229087 ,  0.05420799,
         0.20145623,  0.06400491],
       [-0.59011155, -0.1706088 , -0.15418188, -0.46109855,  0.376051  ,
        -0.53830504, -0.4536028 ,  0.49561328,  0.87093824,  0.15209937,
         0.8967263 , -0.10867216,  0.236418  ,  0.20448625, -0.3178406 ,
         0.62380826,  0.40930378, -0.3401883 , -0.440784  , -0.2554839 ,
        -0.24463907, -0.38798544, -0.25914854, -0.03174952,  0.56282204,
        -0.7897501 , -0.10709798, -0.07273412, -0.0012857 ,  0.28835443,
         0.76596403, -0.35351098],
       [ 0.16178158,  0.5164875 ,  0.5986191 ,  0.7643257 , -0.02356788,
        -0.09990684, -0.30404916,  0.3249353 , -0.15180139, -0.7914257 ,
         0.00950969,  0.1673037 , -0.44285193, -0.07842295,  0.5682622 ,
        -0.3971178 , -0.3681825 , -0.10635875, -0.15688257, -0.22003119,
        -0.15501045,  0.31533068,  0.04533852,  0.21004598,  0.14585237,
        -0.20075382, -0.9581097 , -0.27669483, -0.06659777,  0.28053594,
         0.2175407 ,  0.48155653],
       [ 0.10105354, -0.47790116,  0.30630532,  0.35401073, -0.58614093,
        -0.534392  , -0.16674317,  0.1270384 , -0.06432732,  0.25503188,
        -0.33423862,  0.20071338,  0.77736145,  0.66058093,  0.12558894,
        -0.62160236,  0.4061358 , -0.17535833, -0.4917394 ,  0.5973748 ,
         0.154569  , -0.39054844, -0.57388055, -0.25035602,  0.42744693,
        -0.22552805, -0.01345878,  0.04050945, -0.40922242, -0.25677332,
        -0.6854103 ,  0.08331082],
       [-0.7794143 , -0.441387  ,  0.26092175, -0.27987134, -0.02510745,
         0.49629712,  0.05601017,  0.9046479 ,  0.8586943 ,  0.03162046,
        -0.17282106, -0.4783718 ,  0.581717  , -0.11527868, -0.44586357,
         0.05758109,  0.5870187 , -0.6673186 , -0.12697433,  0.86570585,
        -0.31246552, -0.26802096, -0.04992589, -0.14419037, -0.48416272,
         0.5039342 , -0.18667896, -0.45094374,  0.15244646, -0.2246541 ,
        -0.45912313, -0.69949746],
       [-0.5235064 , -0.24813902, -0.39069626,  0.17995022,  0.4684486 ,
         0.502298  , -0.0228699 ,  0.28439292, -0.3089446 , -0.7090401 ,
        -0.31902638,  0.04118027,  0.03708094,  0.9333089 ,  0.23631923,
         0.6339769 ,  0.320012  ,  0.21103093,  0.5059184 ,  0.9060951 ,
         0.02999732, -0.5790142 ,  0.19311576,  0.16083027, -0.1093682 ,
         0.3905298 , -0.59100646,  0.85269266, -0.601895  ,  0.55693996,
         0.29225537, -0.04157944],
       [-0.8836916 , -0.03398053, -0.13592869,  0.80162734,  0.3130151 ,
         0.7343773 , -0.4224068 ,  0.4649117 , -0.6258109 ,  0.28781104,
         0.3736669 , -0.12653992,  0.30015615,  0.01169346,  0.6539728 ,
         0.01166124, -0.37860668,  0.09666455,  0.20125961,  0.6261301 ,
         0.6627176 ,  0.18909037,  0.15919103, -0.17986353, -0.33411294,
```

```
        0.15267096, -0.02720809,  0.26680857,  0.02224791,  0.10816642,
        0.33810794,  0.75967324],
       [-0.49329647,  0.46965238, -0.27189243, -0.29592723, -0.90025145,
        0.00624604, -0.30920708, -0.42753664,  0.2678729 , -0.06547512,
       -0.27227128, -0.7506083 , -0.26952654, -0.27699855,  0.0356667 ,
        0.6109149 , -0.92455775,  0.103516  , -0.31109095,  0.2357096 ,
        0.11149079,  0.7283719 , -0.03612939,  0.7158851 ,  0.4547818 ,
        0.44277015,  0.13056242, -0.2789623 ,  0.139415  , -0.26012784,
        0.39857274, -0.15462165],
       [ 0.20001316,  0.23183326, -0.12737186,  0.05888744,  0.15327434,
       -0.29089862,  0.31866238, -0.05950885,  0.8195137 , -0.38094664,
       -0.29375654, -0.48798296,  0.9661329 , -0.04904966,  0.22693932,
        0.12908521,  0.24713908, -0.08542947, -0.33748066, -0.03458998,
       -0.63114685, -0.16783531,  0.35632947,  0.5646529 , -0.3533206 ,
        0.62934977,  0.3607169 , -0.39739594,  0.08194552, -0.7697417 ,
       -0.36096147,  0.11240108]], dtype=float32)
```

In [ ]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur
```

In [ ]:
```
history = model.fit(x_train, y_train, batch_size=32, epochs=100, validation
```

```
Epoch 1/100
374/374 [==============================] - 5s 6ms/step - loss: 2.8482 -
accuracy: 0.5841 - val_loss: 1.5506 - val_accuracy: 0.7572
Epoch 2/100
374/374 [==============================] - 2s 5ms/step - loss: 1.1256 -
accuracy: 0.7222 - val_loss: 0.7566 - val_accuracy: 0.7953
Epoch 3/100
374/374 [==============================] - 3s 8ms/step - loss: 0.6724 -
accuracy: 0.7687 - val_loss: 0.5505 - val_accuracy: 0.7961
Epoch 4/100
374/374 [==============================] - 2s 5ms/step - loss: 0.5483 -
accuracy: 0.7863 - val_loss: 0.4821 - val_accuracy: 0.8046
Epoch 5/100
374/374 [==============================] - 2s 5ms/step - loss: 0.5106 -
accuracy: 0.7854 - val_loss: 0.4532 - val_accuracy: 0.8134
Epoch 6/100
374/374 [==============================] - 2s 6ms/step - loss: 0.4943 -
accuracy: 0.7935 - val_loss: 0.4462 - val_accuracy: 0.8144
Epoch 7/100
```

In [ ]:

In [ ]:

Optimization Method

In [ ]:
```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=11))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                384

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 1473 (5.75 KB)
Trainable params: 1473 (5.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

BGD -

In [ ]:
```
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
import time
```

In [ ]:
```
print(x_train.shape, x_test.shape)
```

```
(11944, 11) (3982, 11)
```

In [ ]:
```
start = time.time()
history = model.fit(x_train, y_train, batch_size=11944, epochs=5, validatio
print(time.time()-start)
```

```
Epoch 1/5
1/1 [==============================] - 5s 5s/step - loss: 0.7188 - accurac
y: 0.5041 - val_loss: 0.6945 - val_accuracy: 0.5402
Epoch 2/5
1/1 [==============================] - 0s 34ms/step - loss: 0.6991 - accur
acy: 0.5321 - val_loss: 0.6818 - val_accuracy: 0.5600
Epoch 3/5
1/1 [==============================] - 0s 32ms/step - loss: 0.6862 - accur
acy: 0.5504 - val_loss: 0.6716 - val_accuracy: 0.5801
Epoch 4/5
1/1 [==============================] - 0s 33ms/step - loss: 0.6759 - accur
acy: 0.5661 - val_loss: 0.6629 - val_accuracy: 0.5974
Epoch 5/5
1/1 [==============================] - 0s 34ms/step - loss: 0.6672 - accur
acy: 0.5826 - val_loss: 0.6551 - val_accuracy: 0.6123
5.600061655044556
```

BGD :

Time : 5.600061655044556

loss: 0.6672 - accuracy: 0.5826 - val_loss: 0.6551 - val_accuracy: 0.6123

SGD

In [ ]:
```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=11))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
start = time.time()
history1 = model.fit(x_train, y_train, batch_size=1, epochs=5, validation_d
print(time.time()-start)
```

```
Epoch 1/5
11944/11944 [==============================] - 47s 4ms/step - loss: 0.4665
- accuracy: 0.7970 - val_loss: 0.4622 - val_accuracy: 0.8122
Epoch 2/5
11944/11944 [==============================] - 40s 3ms/step - loss: 0.4460
- accuracy: 0.8178 - val_loss: 0.4346 - val_accuracy: 0.8109
Epoch 3/5
11944/11944 [==============================] - 41s 3ms/step - loss: 0.4390
- accuracy: 0.8207 - val_loss: 0.4356 - val_accuracy: 0.8227
Epoch 4/5
11944/11944 [==============================] - 41s 3ms/step - loss: 0.4383
- accuracy: 0.8192 - val_loss: 0.4469 - val_accuracy: 0.8240
Epoch 5/5
11944/11944 [==============================] - 45s 4ms/step - loss: 0.4372
- accuracy: 0.8245 - val_loss: 0.4524 - val_accuracy: 0.8187
213.92868089675903
```

In [ ]:
```
11944/64
```

Out[22]:
```
186.625
```

In [ ]:
```
# BGD :
## Time : 5.600061655044556
## loss: 0.6672 - accuracy: 0.5826 - val_loss: 0.6551 - val_accuracy: 0.612


# SGD
## Time : 213.92868089675903
## loss: 0.4372 - accuracy: 0.8245 - val_loss: 0.4524 - val_accuracy: 0.818

# MBGD
```

In [ ]:
```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=11))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', metrics=['accuracy'])

start = time.time()
history2 = model.fit(x_train, y_train, batch_size=64, epochs=5, validation_
print(time.time()-start)
```

```
Epoch 1/5
187/187 [==============================] - 3s 7ms/step - loss: 0.5168 - ac
curacy: 0.7608 - val_loss: 0.4604 - val_accuracy: 0.7850
Epoch 2/5
187/187 [==============================] - 1s 4ms/step - loss: 0.4469 - ac
curacy: 0.7954 - val_loss: 0.4343 - val_accuracy: 0.7973
Epoch 3/5
187/187 [==============================] - 1s 4ms/step - loss: 0.4218 - ac
curacy: 0.8086 - val_loss: 0.4175 - val_accuracy: 0.8106
Epoch 4/5
187/187 [==============================] - 1s 4ms/step - loss: 0.4044 - ac
curacy: 0.8186 - val_loss: 0.4035 - val_accuracy: 0.8189
Epoch 5/5
187/187 [==============================] - 1s 4ms/step - loss: 0.3926 - ac
curacy: 0.8255 - val_loss: 0.3983 - val_accuracy: 0.8214
10.908029556274414
```

BGD :

Time : 5.600061655044556

loss: 0.6672 - accuracy: 0.5826 - val_loss: 0.6551 - val_accuracy: 0.6123

SGD

Time : 213.92868089675903

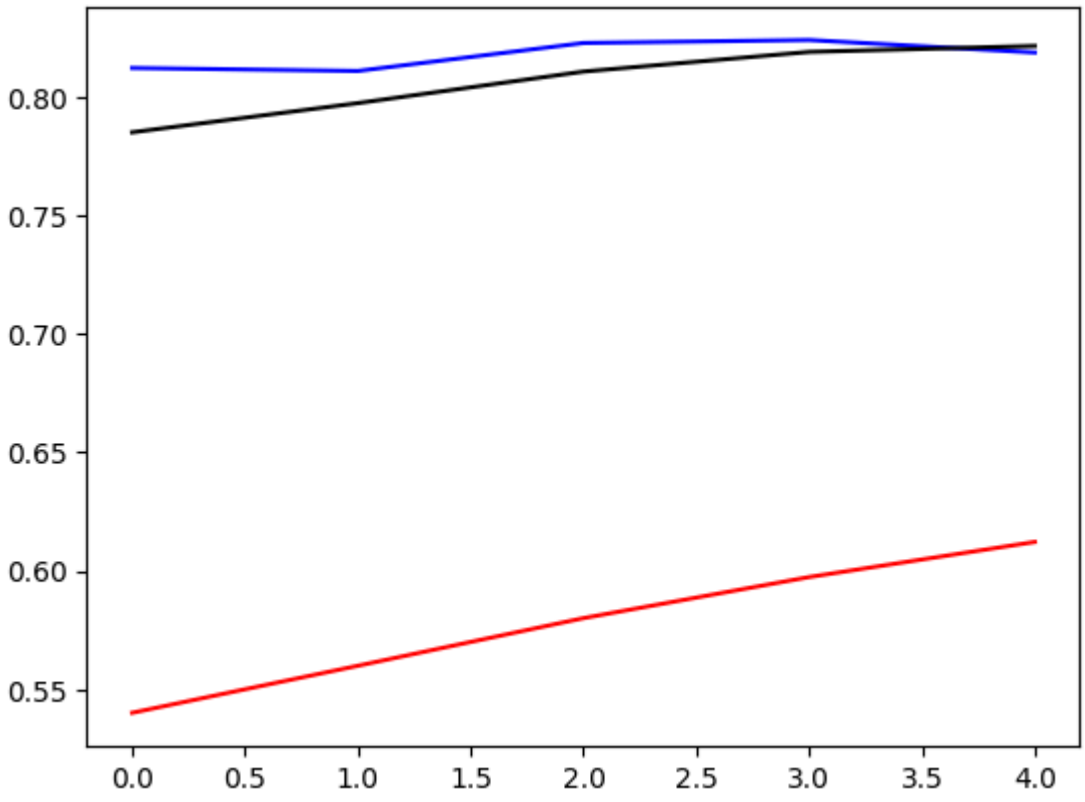
loss: 0.4372 - accuracy: 0.8245 - val_loss: 0.4524 - val_accuracy: 0.8187

MBGD

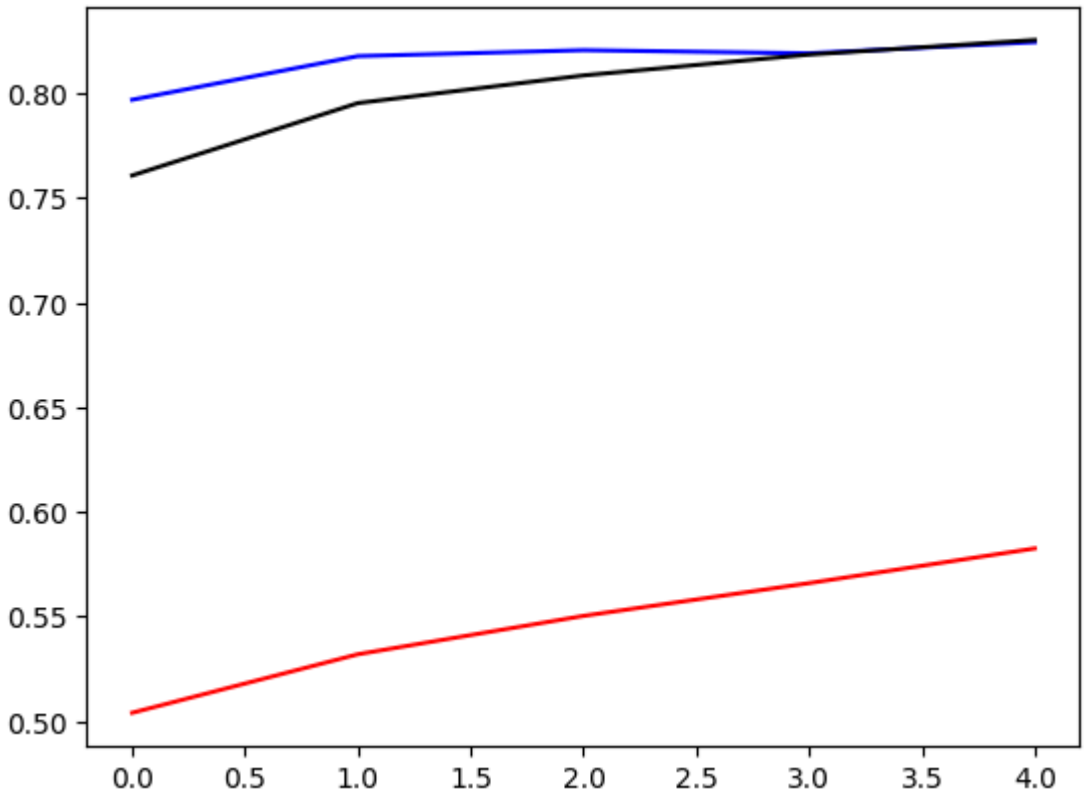
Time : 10.908029556274414

loss: 0.3926 - accuracy: 0.8255 - val_loss: 0.3983 - val_accuracy: 0.8214

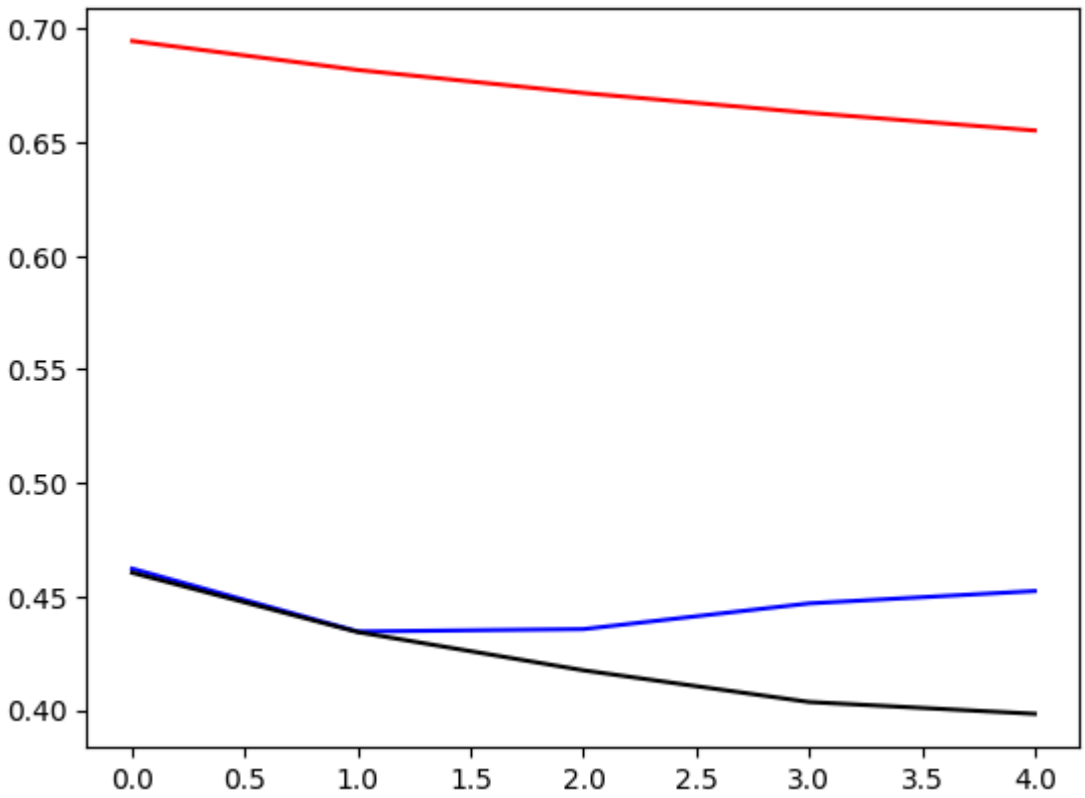
In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['val_accuracy'], color='red')
plt.plot(history1.history['val_accuracy'], color='blue')
plt.plot(history2.history['val_accuracy'], color='black')
plt.show()
```



In [ ]:
```
plt.plot(history.history['accuracy'], color='red')
plt.plot(history1.history['accuracy'], color='blue')
plt.plot(history2.history['accuracy'], color='black')
plt.show()
```



In [ ]:
```
plt.plot(history.history['val_loss'], color='red')
plt.plot(history1.history['val_loss'], color='blue')
plt.plot(history2.history['val_loss'], color='black')
plt.show()
```



In [ ]:

In [ ]: