

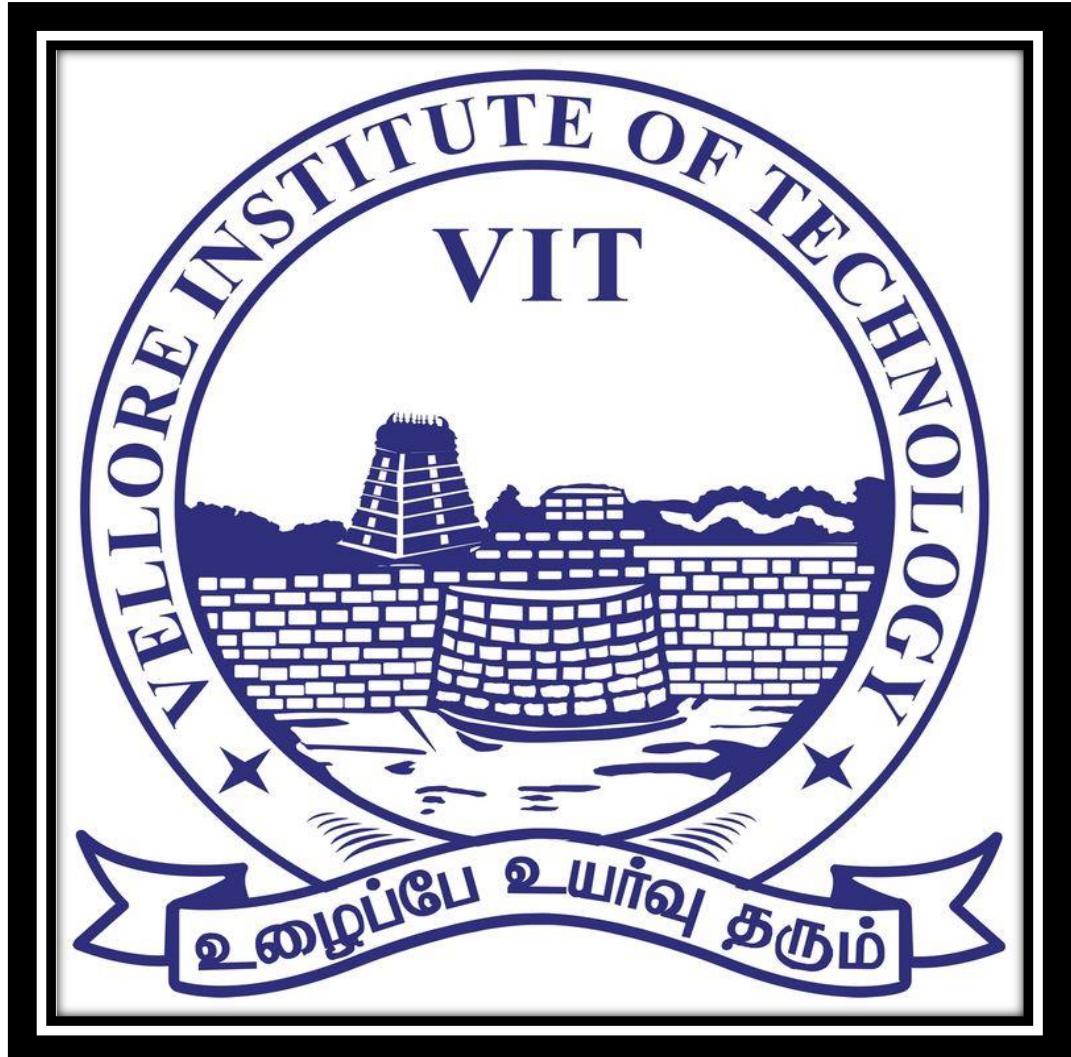


**VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



**VELLORE INSTITUTE OF TECHNOLOGY**

**VIT – CHENNAI CAMPUS**

**DEEP LEARNING LAB**

**LAB ASSIGNMENT – 3**

**FACULTY : DR. HARINI. S**



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

### BCSE332P - DL Lab

#### Lab 3 - DL Lab Assignment

1. AUTO-Implement 3 different neural networks as listed below

- 2 layers. 1<sup>st</sup> layer: ReLU (5 perceptrons),
- 2<sup>nd</sup> Layer: Sigmoid (1 perceptron)
- In the above layers, both sigmoid
- 4 layers with the first three layers: Sigmoid (3) and the last layer: sigmoid (1)

#### CODE:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

tf.random.set_seed(42)
np.random.seed(42)

X, Y = make_circles(n_samples=1000, factor = 0.5, noise = 0.1)
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.show()

# First Model
model = Sequential()
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
model.add(Dense(5, activation='relu', input_dim=X.shape[1]))
model.add(Dense(1, activation ='sigmoid'))\n\nmodel.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)
# Second Model\n\nmodel2 = Sequential()
model2.add(Dense(5, activation='sigmoid', input_dim=X.shape[1]))
model2.add(Dense(1, activation ='sigmoid'))\n\nmodel2.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model2.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)
model2.evaluate(X,Y)
model3 = Sequential()
model3.add(Dense(3, activation='sigmoid', input_dim=X.shape[1]))
model3.add(Dense(3, activation ='sigmoid'))
model3.add(Dense(3, activation ='sigmoid'))
model3.add(Dense(1, activation ='sigmoid'))\n\nmodel3.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model3.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)
model3.evaluate(X, Y)
dataset = "/content/creditcard.csv"\n\ndf = pd.read_csv(dataset)\n\nfeatures = df.drop(columns='Class')
target=df['Class']
features.head()
target.head()
from sklearn.model_selection import train_test_split\n\nX_train, X_test, Y_train, Y_test = train_test_split(features,
target,test_size=0.2, random_state=42)
X_train.shape\n\nX_test.shape
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()

scaled_features_train = scaler.fit_transform(X_train)
scaled_features_test = scaler.fit_transform(X_test)

scaled_feature_train_array = np.array(scaled_features_train)
scaled_feature_test_array = np.array(scaled_features_test)
target_train_array= np.array(Y_train)
target_test_array= np.array(Y_test)
scaled_feature_train_array.shape
# First Model
model_p = Sequential()
model_p.add(Dense(5, activation='relu',
input_dim=scaled_feature_train_array.shape[1]))
model_p.add(Dense(1, activation ='sigmoid'))

model_p.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model_p.fit(scaled_feature_train_array, target_train_array,
batch_size=32, epochs = 5, verbose = 1)
model_p.evaluate(scaled_feature_train_array,target_train_array)
predictions = model_p.predict(scaled_features_test_array)
binary_predictions = (predictions > 0.5).astype(int)

predicted_labels = np.squeeze(binary_predictions)
true_labels = target_test_array

result_comparison = pd.DataFrame({'True Labels': true_labels,
'Predicted Labels': predicted_labels})
print(result_comparison)
# Second Model

model2_p = Sequential()
model2_p.add(Dense(5, activation='sigmoid',
input_dim=scaled_feature_train_array.shape[1]))
model2_p.add(Dense(1, activation ='sigmoid'))

model2_p.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
model2_p.fit(scaled_feature_train_array, target_train_array,
batch_size=32, epochs = 5, verbose = 1)
model2_p.evaluate(scaled_feature_train_array,target_train_array)
predictions_2 = model2_p.predict(scaled_feature_test_array)
binary_predictions_2 = (predictions_2 > 0.5).astype(int)

predicted_labels_2 = np.squeeze(binary_predictions_2)
true_labels = target_test_array

result_comparison_2 = pd.DataFrame({'True Labels': true_labels,
'Predicted Labels': predicted_labels})
print(result_comparison_2)
model3_p = Sequential()
model3_p.add(Dense(3, activation='sigmoid',
input_dim=scaled_feature_train_array.shape[1]))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(1, activation ='sigmoid'))

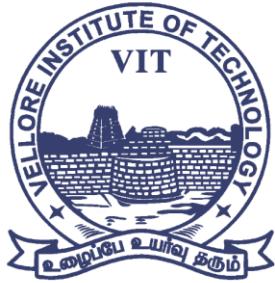
model3_p.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',metrics='accuracy')
model3_p.fit(scaled_feature_train_array, target_train_array,
batch_size=32, epochs = 5, verbose = 1)
model3_p.evaluate(scaled_feature_train_array, target_train_array)
predictions_3 = model3_p.predict(scaled_feature_test_array)
binary_predictions_3 = (predictions_3 > 0.5).astype(int)

predicted_labels_3 = np.squeeze(binary_predictions_3)
true_labels = target_test_array

result_comparison_3 = pd.DataFrame({'True Labels': true_labels,
'Predicted Labels': predicted_labels})
print(result_comparison_3)
```

### **LINK TO THE GOOGLE COLAB NOTEBOOK:**

<https://colab.research.google.com/drive/1t3phbps0ijnkK5PVE3Kv9NzYHS9Ie9U>



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

### OUTPUT OF THE CODE SNIPPET:

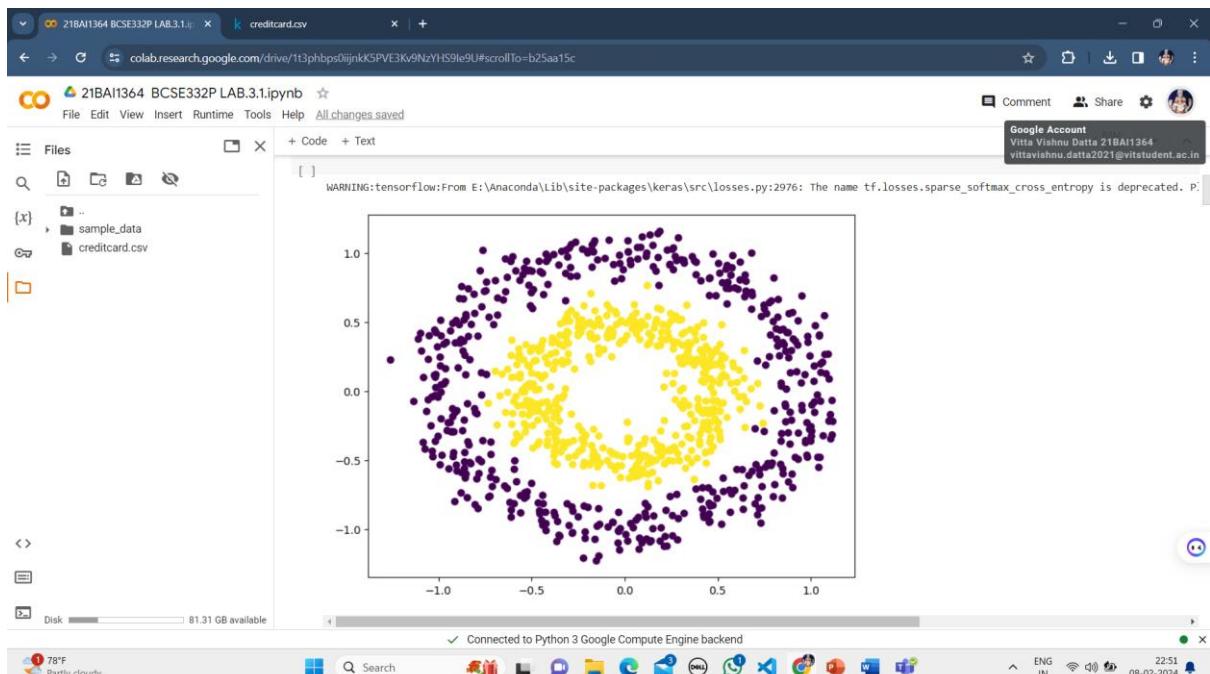
The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.1.ipynb". The code imports numpy, pandas, tensorflow, and other required modules. It generates two concentric circles of data points using plt.scatter and plots them. A warning message about a deprecated TensorFlow name is visible. The bottom of the screen shows a Windows taskbar with various icons and system status.

```
[ ] import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.initializers import RandomNormal
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

tf.random.set_seed(42)
np.random.seed(42)

X, Y = make_circles(n_samples=1000, factor = 0.5, noise = 0.1)
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.show()

WARNING:tensorflow:From E:\Anaconda\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. P:
```





# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.1.ipynb". The code cell contains Python code for a neural network:[ ] # First Model  
model = Sequential()  
model.add(Dense(5, activation='relu', input\_dim=X.shape[1]))  
model.add(Dense(1, activation ='sigmoid'))  
  
model.compile(optimizer=Adam(learning\_rate=0.01), loss='binary\_crossentropy',metrics='accuracy')  
model.fit(X, Y, batch\_size=32, epochs = 100, verbose = 1)

Output:

```
Epoch 18/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2261 - accuracy: 0.9450  
Epoch 19/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2195 - accuracy: 0.9400  
Epoch 20/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2127 - accuracy: 0.9440  
Epoch 21/100  
32/32 [=====] - 0s 2ms/step - loss: 0.2052 - accuracy: 0.9420  
Epoch 22/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1985 - accuracy: 0.9460  
Epoch 23/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1934 - accuracy: 0.9430  
Epoch 24/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1886 - accuracy: 0.9450  
Epoch 25/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1853 - accuracy: 0.9440  
Epoch 26/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1800 - accuracy: 0.9510  
Epoch 27/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1753 - accuracy: 0.9460  
Epoch 28/100  
32/32 [=====] - 0s 1ms/step - loss: 0.1712 - accuracy: 0.9500  
Epoch 29/100  
32/32 [=====] - 0s 2ms/step - loss: 0.1686 - accuracy: 0.9470  
Epoch 30/100
```

At the bottom, it says "Connected to Python 3 Google Compute Engine backend". The system tray shows "78°F Partly cloudy" and the date/time "08-02-2024 22:52".

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.1.ipynb". The code cell contains Python code for a neural network:[ ] model.evaluate(X,Y)  
32/32 [=====] - 0s 1ms/step - loss: 0.0672 - accuracy: 0.9840  
[0.06716607511043549, 0.984000027179718]  
  
[ ] # Second Model  
  
model2 = Sequential()  
model2.add(Dense(5, activation='sigmoid', input\_dim=X.shape[1]))  
model2.add(Dense(1, activation ='sigmoid'))  
  
model2.compile(optimizer=Adam(learning\_rate=0.01), loss='binary\_crossentropy',metrics='accuracy')  
model2.fit(X, Y, batch\_size=32, epochs = 100, verbose = 1)

Output:

```
32/32 [=====] - 0s 2ms/step - loss: 0.3070 - accuracy: 0.9600  
Epoch 53/100  
32/32 [=====] - 0s 2ms/step - loss: 0.3008 - accuracy: 0.9640  
Epoch 54/100  
32/32 [=====] - 0s 2ms/step - loss: 0.2936 - accuracy: 0.9660  
Epoch 55/100  
32/32 [=====] - 0s 2ms/step - loss: 0.2875 - accuracy: 0.9660  
Epoch 56/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2816 - accuracy: 0.9610  
Epoch 57/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2764 - accuracy: 0.9650  
Epoch 58/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2706 - accuracy: 0.9650  
Epoch 59/100  
32/32 [=====] - 0s 1ms/step - loss: 0.2659 - accuracy: 0.9620  
Epoch 60/100  
32/32 [=====] - 0s 2ms/step - loss: 0.2614 - accuracy: 0.9600  
Epoch 61/100  
32/32 [=====] - 0s 2ms/step - loss: 0.2570 - accuracy: 0.9600
```

At the bottom, it says "Connected to Python 3 Google Compute Engine backend". The system tray shows "78°F Partly cloudy" and the date/time "08-02-2024 22:52".



# VIT<sup>®</sup>

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)  
**CHENNAI**

21BAI1364 BCSE332P LAB.3.1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] model2.evaluate(X,Y)
32/32 [=====] - 0s 1ms/step - loss: 0.1654 - accuracy: 0.9560
[0.16543905436992645, 0.9559999704368962]

[ ] model3 = Sequential()
model3.add(Dense(3, activation='sigmoid', input_dim=X.shape[1]))
model3.add(Dense(3, activation='sigmoid'))
model3.add(Dense(3, activation='sigmoid'))
model3.add(Dense(1, activation ='sigmoid'))

model3.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy',metrics='accuracy')
model3.fit(X, Y, batch_size=32, epochs = 100, verbose = 1)

Epoch 1/100
32/32 [=====] - 1s 2ms/step - loss: 0.6951 - accuracy: 0.4880
Epoch 2/100
32/32 [=====] - 0s 2ms/step - loss: 0.6948 - accuracy: 0.4840
Epoch 3/100
32/32 [=====] - 0s 2ms/step - loss: 0.6939 - accuracy: 0.4980
Epoch 4/100
32/32 [=====] - 0s 2ms/step - loss: 0.6934 - accuracy: 0.4990
Epoch 5/100
32/32 [=====] - 0s 2ms/step - loss: 0.6940 - accuracy: 0.4870
Epoch 6/100
32/32 [=====] - 0s 2ms/step - loss: 0.6935 - accuracy: 0.5170
Epoch 7/100
32/32 [=====] - 0s 2ms/step - loss: 0.6944 - accuracy: 0.5000
Epoch 8/100
32/32 [=====] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5040
Epoch 9/100
32/32 [=====] - 0s 2ms/step - loss: 0.6928 - accuracy: 0.5010
```

Connected to Python 3 Google Compute Engine backend

78°F Partly cloudy

Search

21BAI1364 BCSE332P LAB.3.1.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] model3.evaluate(X, Y)
32/32 [=====] - 0s 2ms/step - loss: 0.0875 - accuracy: 0.9860
Epoch 28/100
32/32 [=====] - 0s 2ms/step - loss: 0.0780 - accuracy: 0.9840
Epoch 29/100
32/32 [=====] - 0s 2ms/step - loss: 0.0732 - accuracy: 0.9820

[ ] model3.evaluate(X, Y)
32/32 [=====] - 0s 2ms/step - loss: 0.0129 - accuracy: 0.9940
[0.012934341095387936, 0.9940000176429749]

[ ] dataset = "/content/creditcard.csv"
df = pd.read_csv(dataset)

features = df.drop(columns='Class')
target=df['Class']

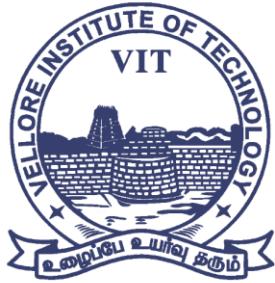
[ ] features.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.251412	-0.018307	0.277838	-0.110474									
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.069083	-0.225775	-0.638672	0.101288									
2	1.0	-1.358354	-1.340163	1.773209	0.397980	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.524980	0.247998	0.771679	0.909412									
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.208038	-0.108300	0.005274	-0.190321									
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.408542	-0.009431	0.798278	-0.137458									

Connected to Python 3 Google Compute Engine backend

78°F Partly cloudy

Search



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

21BAI1364 BCSE332P LAB.3.1.ipynb

```
[ ] target.head()
0    0
1    0
2    0
3    0
4    0
Name: class, dtype: int64

[ ] from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.2, random_state=42)

[ ] X_train.shape
(227845, 30)

[ ] X_test.shape
(56962, 30)

[ ] from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

scaled_features_train = scaler.fit_transform(X_train)
scaled_features_test = scaler.fit_transform(X_test)
```

Connected to Python 3 Google Compute Engine backend

78°F Partly cloudy

21BAI1364 BCSE332P LAB.3.1.ipynb

```
[ ] model_p.evaluate(scaled_feature_train_array, target_train_array)
7121/7121 [=====] - 9s 1ms/step - loss: 0.0032 - accuracy: 0.9994
[0.00319129740819335, 0.9994206428527832]

[ ] predictions = model_p.predict(scaled_features_test_array)
binary_predictions = (predictions > 0.5).astype(int)

predicted_labels = np.squeeze(binary_predictions)
true_labels = target_test_array

result_comparison = pd.DataFrame({'True Labels': true_labels, 'Predicted Labels': predicted_labels})
print(result_comparison)
```

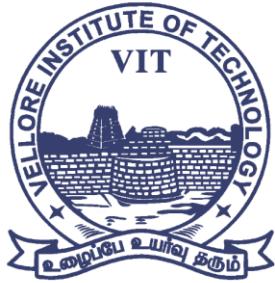
	True Labels	Predicted Labels
0	1	1
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
56957	0	0
56958	0	0
56959	0	0
56960	0	0
56961	0	0

[56962 rows x 2 columns]

```
[ ] # Second Model
model2 = Sequential()
```

Connected to Python 3 Google Compute Engine backend

78°F Partly cloudy



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

21BAI1364 BCSE332P LAB.3.1.ipynb creditcard.csv

```
[ ] model_p.evaluate(scaled_feature_train_array,target_train_array)
7121/7121 [=====] - 9s 1ms/step - loss: 0.0032 - accuracy: 0.9994
[0.00319129740819335, 0.9994206428527832]

[ ] predictions = model_p.predict(scaled_features_test_array)
binary_predictions = (predictions > 0.5).astype(int)

predicted_labels = np.squeeze(binary_predictions)
true_labels = target_test_array

result_comparison = pd.DataFrame({'True Labels': true_labels, 'Predicted Labels': predicted_labels})
print(result_comparison)
```

True Labels	Predicted Labels
0	1
1	0
2	0
3	0
4	0
...	...
56957	0
56958	0
56959	0
56960	0
56961	0

[56962 rows x 2 columns]

```
[ ] # Second Model
model2_p = Sequential()
```

Disk 81.31 GB available

78°F Partly cloudy

Connected to Python 3 Google Compute Engine backend

ENG IN 22:54 08-02-2024

21BAI1364 BCSE332P LAB.3.1.ipynb creditcard.csv

```
[ ] model2_p = Sequential()
model2_p.add(Dense(5, activation='sigmoid', input_dim=scaled_feature_train_array.shape[1]))
model2_p.add(Dense(1, activation = 'sigmoid'))

model2_p.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metrics='accuracy')
model2_p.fit(scaled_feature_train_array, target_train_array, batch_size=32, epochs = 5, verbose = 1)
```

Epoch 1/5  
7121/7121 [=====] - 10s 1ms/step - loss: 0.0108 - accuracy: 0.9985  
Epoch 2/5  
7121/7121 [=====] - 10s 1ms/step - loss: 0.0039 - accuracy: 0.9992  
Epoch 3/5  
7121/7121 [=====] - 10s 1ms/step - loss: 0.0037 - accuracy: 0.9993  
Epoch 4/5  
7121/7121 [=====] - 10s 1ms/step - loss: 0.0036 - accuracy: 0.9993  
Epoch 5/5  
7121/7121 [=====] - 10s 1ms/step - loss: 0.0036 - accuracy: 0.9993  
<keras.src.callbacks.History at 0x14ee297b00>

```
[ ] model2_p.evaluate(scaled_feature_train_array,target_train_array)
7121/7121 [=====] - 10s 1ms/step - loss: 0.0033 - accuracy: 0.9994
[0.0032731930259615183, 0.9994074702262878]
```

```
[ ] predictions_2 = model2_p.predict(scaled_feature_test_array)
binary_predictions_2 = (predictions_2 > 0.5).astype(int)
```

Disk 81.31 GB available

Connected to Python 3 Google Compute Engine backend

78°F Partly cloudy

ENG IN 22:54 08-02-2024



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
[ ] predictions_2 = model2_p.predict(scaled_feature_test_array)
binary_predictions_2 = (predictions_2 > 0.5).astype(int)

predicted_labels_2 = np.squeeze(binary_predictions_2)
true_labels = target_test_array

result_comparison_2 = pd.DataFrame({'True Labels': true_labels, 'Predicted Labels': predicted_labels_2})
print(result_comparison_2)

1781/1781 [=====] - 2s 1ms/step
    True Labels Predicted Labels
0           1           1
1           0           0
2           0           0
3           0           0
4           0           0
...
56957       0           0
56958       0           0
56959       0           0
56960       0           0
56961       0           0

[56962 rows x 2 columns]

[ ] model3_p = Sequential()
model3_p.add(Dense(3, activation='sigmoid', input_dim=scaled_feature_train_array.shape[1]))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(1, activation ='sigmoid'))
```

```
[ ] model3_p = Sequential()
model3_p.add(Dense(3, activation='sigmoid', input_dim=scaled_feature_train_array.shape[1]))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(3, activation ='sigmoid'))
model3_p.add(Dense(1, activation ='sigmoid'))

model3_p.compile(optimizer='Adam(learning_rate=0.01), loss='binary_crossentropy',metrics='accuracy')
model3_p.fit(scaled_feature_train_array, target_train_array, batch_size=32, epochs = 5, verbose = 1)

Epoch 1/5
7121/7121 [=====] - 12s 2ms/step - loss: 0.0152 - accuracy: 0.9982
Epoch 2/5
7121/7121 [=====] - 11s 2ms/step - loss: 0.0042 - accuracy: 0.9993
Epoch 3/5
7121/7121 [=====] - 11s 2ms/step - loss: 0.0039 - accuracy: 0.9993
Epoch 4/5
7121/7121 [=====] - 11s 2ms/step - loss: 0.0039 - accuracy: 0.9994
Epoch 5/5
7121/7121 [=====] - 12s 2ms/step - loss: 0.0038 - accuracy: 0.9994
<keras.src.callbacks.History at 0x14e8ccdae0>

[ ] model3_p.evaluate(scaled_feature_train_array, target_train_array)

7121/7121 [=====] - 9s 1ms/step - loss: 0.0041 - accuracy: 0.9993
[0.004130946937948465, 0.999275803565979]
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

The screenshot shows a Jupyter Notebook interface in Google Colab. The code in the notebook is as follows:

```
model3_p.evaluate(scaled_feature_train_array, target_train_array)
7121/7121 [=====] - 9s 1ms/step - loss: 0.0041 - accuracy: 0.9993
[0.004130946937948465, 0.999275803565979]

predictions_3 = model3_p.predict(scaled_feature_test_array)
binary_predictions_3 = (predictions_3 > 0.5).astype(int)

predicted_labels_3 = np.squeeze(binary_predictions_3)
true_labels = target_test_array

result_comparison_3 = pd.DataFrame({'True Labels': true_labels, 'Predicted Labels': predicted_labels})
print(result_comparison_3)

1781/1781 [=====] - 2s 1ms/step
True Labels Predicted Labels
0 1 1
1 0 0
2 0 0
3 0 0
4 0 0
...
56957 0 0
56958 0 0
56959 0 0
56960 0 0
56961 0 0

[56962 rows x 2 columns]
```

The notebook also displays a message: "Connected to Python 3 Google Compute Engine backend". The system tray at the bottom shows the date (08-02-2024), time (22:55), and battery level (78%).

## 2. Change the activation functions and implement the code

CODE:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.initializers import RandomNormal
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

tf.random.set_seed(42)
np.random.seed(42)

# Make data: Two circles on x-y plane as a classification problem
X, y = make_circles(n_samples=1000, factor=0.5, noise=0.1)
plt.figure(figsize=(8,6))
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

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

# Test performance with 3-layer binary classification network
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "relu"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

# Test performance with 3-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "tanh"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

# Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(1, "tanh")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

# Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "relu"),
    Dense(5, "relu"),
    Dense(5, "relu"),
    Dense(1, "relu")
])
```



## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
Dense(5, "relu"),
Dense(5, "relu"),
Dense(1, "relu")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))
# Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "sigmoid"),
    Dense(5, "sigmoid"),
    Dense(5, "sigmoid"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))
# Illustrate weights across epochs
class WeightCapture(Callback):
    "Capture the weights of each layer of the model"
    def __init__(self, model):
        super().__init__()
        self.model = model
        self.weights = []
        self.epoch = []

    def on_epoch_end(self, epoch, logs=None):
        self.epoch.append(epoch) # remember the epoch axis
        weight = {}
        for layer in model.layers:
            if not layer.weights:
                continue
            name = layer.weights[0].name.split("/") [0]
            weight[name] = layer.weights[0].numpy()
        self.weights.append(weight)
def make_mlp(activation, initializer, name):
    "Create a model with specified activation and initializer"
    model = Sequential([
        Input(shape=(2,), name=name+"0"),
        Dense(5, "relu"),
        Dense(5, "relu"),
        Dense(1, "relu")
    ])
    model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["acc"])
    model.fit(X, y, batch_size=32, epochs=100, verbose=0)
    print(model.evaluate(X,y))
    return model
```



```
Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"1"),
    Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"2"),
        Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"3"),
            Dense(5, activation=activation, kernel_initializer=initializer,
name=name+"4"),
                Dense(1, activation=activation, kernel_initializer=initializer,
name=name+"5")
)
return model
def plotweight(capture_cb):
    "Plot the weights' mean and s.d. across epochs"
    fig, ax = plt.subplots(2, 1, sharex=True, constrained_layout=True,
figsize=(8, 10))
    ax[0].set_title("Mean weight")
    for key in capture_cb.weights[0]:
        ax[0].plot(capture_cb.epochs, [w[key].mean() for w in
capture_cb.weights], label=key)
    ax[0].legend()
    ax[1].set_title("S.D.")
    for key in capture_cb.weights[0]:
        ax[1].plot(capture_cb.epochs, [w[key].std() for w in
capture_cb.weights], label=key)
    ax[1].legend()
    plt.show()
initializer = RandomNormal(mean=0, stddev=1)
batch_size = 32
n_epochs = 100
# sigmoid activation
model = make_mlp("sigmoid", initializer, "sigmoid")
capture_cb = WeightCapture(model)
capture_cb.on_epoch_end(-1)
model.compile(optimizer="rmsprop", loss="binary_crossentropy",
metrics=["acc"])
print("Before training: Accuracy", accuracy_score(y, (model(X).numpy() > 0.5).astype(int)))
model.fit(X, y, batch_size=batch_size, epochs=n_epochs,
callbacks=[capture_cb], verbose=0)
print("After training: Accuracy", accuracy_score(y, (model(X).numpy() > 0.5).astype(int)))
```



# VIT®

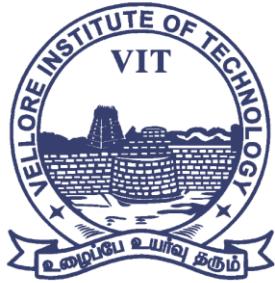
## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
print(model.evaluate(X,y))  
plotweight(capture_cb)  
# tanh activation  
model = make_mlp("tanh", initializer, "tanh")  
capture_cb = WeightCapture(model)  
capture_cb.on_epoch_end(-1)  
model.compile(optimizer="rmsprop", loss="binary_crossentropy",  
metrics=["acc"])  
print("Before training: Accuracy", accuracy_score(y, (model(X).numpy()>  
0.5).astype(int)))  
model.fit(X, y, batch_size=batch_size, epochs=n_epochs,  
callbacks=[capture_cb], verbose=0)  
print("After training: Accuracy", accuracy_score(y, (model(X).numpy() >  
0.5).astype(int)))  
print(model.evaluate(X,y))  
plotweight(capture_cb)  
# ReLu activation  
model = make_mlp("relu", initializer, "relu")  
capture_cb = WeightCapture(model)  
capture_cb.on_epoch_end(-1)  
model.compile(optimizer="rmsprop", loss="binary_crossentropy",  
metrics=["acc"])  
print("Before training: Accuracy", accuracy_score(y, (model(X).numpy()>  
0.5).astype(int)))  
model.fit(X, y, batch_size=batch_size, epochs=n_epochs,  
callbacks=[capture_cb], verbose=0)  
print("After training: Accuracy", accuracy_score(y, (model(X).numpy() >  
0.5).astype(int)))  
print(model.evaluate(X,y))  
plotweight(capture_cb)
```

### LINK TO THE GOOGLE COLAB NOTEBOOK:

<https://colab.research.google.com/drive/1jY7rhJ1Zz9wDsICAzJCxPOGDTsWMkIqq>



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

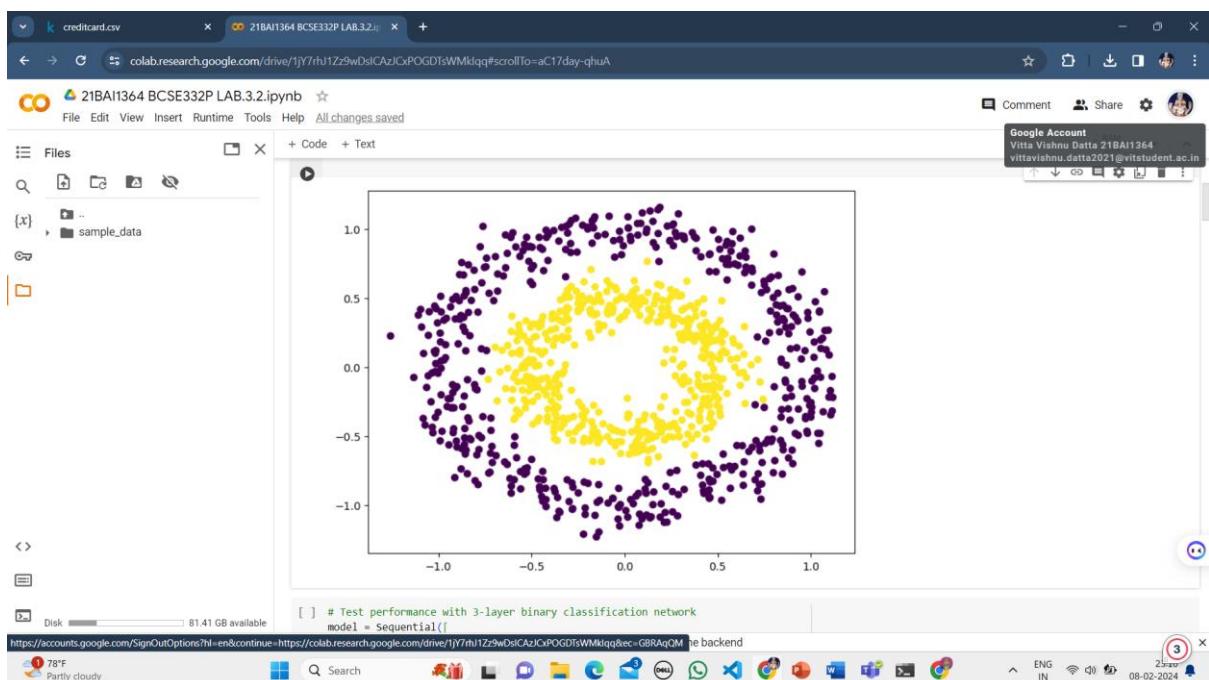
### OUTPUT OF THE CODE SNIPPET:

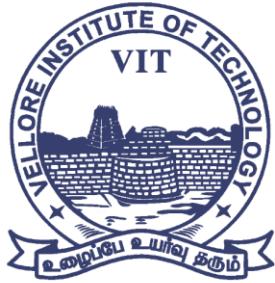
A screenshot of a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.2.ipynb". The code imports TensorFlow and its layers, initializes a Sequential model with Dense layers, and generates a scatter plot of two concentric circles. The plot shows two distinct clusters of points: one purple cluster on the outer ring and one yellow cluster on the inner ring, both centered at (0,0).

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import Callback
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras import Sequential
from tensorflow.keras.initializers import RandomNormal
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.metrics import accuracy_score

tf.random.set_seed(42)
np.random.seed(42)

# Make data: Two circles on x-y plane as a classification problem
X, y = make_circles(n_samples=1000, factor=0.5, noise=0.1)
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```





# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.2.ipynb". The code in the cell is as follows:

```
[ ] # Test performance with 3-layer binary classification network
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "relu"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

32/32 [=====] - 0s 2ms/step - loss: 0.3585 - acc: 0.8370
[0.35851335525512695, 0.8370000123977661]

[ ] # Test performance with 3-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "tanh"),
    Dense(1, "sigmoid")
])
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

32/32 [=====] - 0s 2ms/step - loss: 0.4341 - acc: 0.9280
[0.4341290593147278, 0.927999973297191]
```

The notebook also displays a message from the account: "Google Account Vitta Vishnu Datta 21BAI1364 vittavishnu.datta2021@vitstudent.ac.in". The status bar at the bottom shows "78°F Partly cloudy" and the date "08-02-2024".

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.2.ipynb". The code in the cell is as follows:

```
[ ] # Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(5, "tanh"),
    Dense(1, "tanh")
])
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))

32/32 [=====] - 0s 2ms/step - loss: 0.1110 - acc: 0.9920
[0.11098651587963104, 0.9919999837875366]

[ ] # Test performance with 5-layer network with sigmoid activation
model = Sequential([
    Input(shape=(2,)),
    Dense(5, "relu"),
    Dense(5, "relu"),
    Dense(5, "relu"),
    Dense(5, "relu"),
    Dense(1, "relu")
])
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
print(model.evaluate(X,y))
```

The notebook also displays a message from the account: "Google Account Vitta Vishnu Datta 21BAI1364 vittavishnu.datta2021@vitstudent.ac.in". The status bar at the bottom shows "78°F Partly cloudy" and the date "08-02-2024".



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

A screenshot of a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.2.ipynb". The code implements a neural network with four layers and plots the mean and standard deviation of weights across 100 epochs. The output shows training progress and the generated plots.

```
32/32 [=====] - 0s 2ms/step - loss: 0.6913 - acc: 0.6100
[0.691297173500061, 0.6100000143051147]

[ ] # Illustrate weights across epochs
class weightCapture(callback):
    "Capture the weights of each layer of the model"
    def __init__(self, model):
        super().__init__()
        self.model = model
        self.weights = []
        self.epochs = []

    def on_epoch_end(self, epoch, logs=None):
        self.epochs.append(epoch) # remember the epoch axis
        weight = {}
        for layer in model.layers:
            if not layer.weights:
                continue
            name = layer.weights[0].name.split("/")[0]
            weight[name] = layer.weights[0].numpy()
        self.weights.append(weight)

    def make_mlp(activation, initializer, name):
        "Create a model with specified activation and initializer"
        model = Sequential([
            Input(shape=(2,), name=name+"0"),
            Dense(5, activation=activation, kernel_initializer=initializer, name=name+"1"),
            Dense(5, activation=activation, kernel_initializer=initializer, name=name+"2"),
            Dense(5, activation=activation, kernel_initializer=initializer, name=name+"3"),
            Dense(5, activation=activation, kernel_initializer=initializer, name=name+"4"),
            Dense(1, activation=activation, kernel_initializer=initializer, name=name+"5")
        ])
        return model

[ ] def plotweight(capture_cb):
    "Plot the weights' mean and s.d. across epochs"
    fig, ax = plt.subplots(2, 1, sharex=True, constrained_layout=True, figsize=(8, 10))
    ax[0].set_title("Mean weight")
    for key in capture_cb.weights[0]:
        ax[0].plot(capture_cb.epochs, [w[key].mean() for w in capture_cb.weights], label=key)
    ax[0].legend()
    ax[0].set_title("S.D.")
    for key in capture_cb.weights[0]:
        ax[1].plot(capture_cb.epochs, [w[key].std() for w in capture_cb.weights], label=key)
    ax[1].legend()
    plt.show()

[ ] initializer = RandomNormal(mean=0, stddev=1)
batch_size = 32
n_epochs = 100

[ ] # sigmoid activation
model = make_mlp("sigmoid", initializer, "sigmoid")
capture_cb = WeightCapture(model)
```

A screenshot of a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.2.ipynb". The code defines a neural network with five layers and plots the mean and standard deviation of weights across 100 epochs. The output shows training progress and the generated plots.

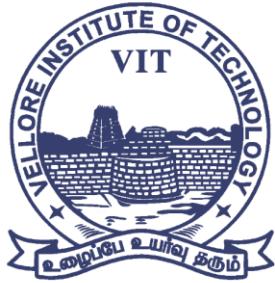
```
Input(shape=(2,), name=name+"0"),
Dense(5, activation=activation, kernel_initializer=initializer, name=name+"1"),
Dense(5, activation=activation, kernel_initializer=initializer, name=name+"2"),
Dense(5, activation=activation, kernel_initializer=initializer, name=name+"3"),
Dense(5, activation=activation, kernel_initializer=initializer, name=name+"4"),
Dense(1, activation=activation, kernel_initializer=initializer, name=name+"5")

[ ]) return model

[ ] def plotweight(capture_cb):
    "Plot the weights' mean and s.d. across epochs"
    fig, ax = plt.subplots(2, 1, sharex=True, constrained_layout=True, figsize=(8, 10))
    ax[0].set_title("Mean weight")
    for key in capture_cb.weights[0]:
        ax[0].plot(capture_cb.epochs, [w[key].mean() for w in capture_cb.weights], label=key)
    ax[0].legend()
    ax[0].set_title("S.D.")
    for key in capture_cb.weights[0]:
        ax[1].plot(capture_cb.epochs, [w[key].std() for w in capture_cb.weights], label=key)
    ax[1].legend()
    plt.show()

[ ] initializer = RandomNormal(mean=0, stddev=1)
batch_size = 32
n_epochs = 100

[ ] # sigmoid activation
model = make_mlp("sigmoid", initializer, "sigmoid")
capture_cb = WeightCapture(model)
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

creditcard.csv    21BAI1364 BCSE332P LAB.3.2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files    + Code + Text

[x] .. sample\_data

[x]

```
[ ] # sigmoid activation
model = make_mlp("sigmoid", initializer, "sigmoid")
capture_cb = WeightCapture(model)
capture_cb.on_epoch_end -= 1
model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["acc"])
print("Before training: Accuracy 0.457")
E:\Anaconda\lib\site-packages\keras\src\initializers\initializers.py:120: UserWarning: The initializer RandomNormal is unseeded and being called.
    warnings.warn(
After training: Accuracy 0.623
32/32 [=====] - 0s 2ms/step - loss: 0.6078 - acc: 0.6230
[0.6078450679779053, 0.6230000257492065]
```

Mean weight

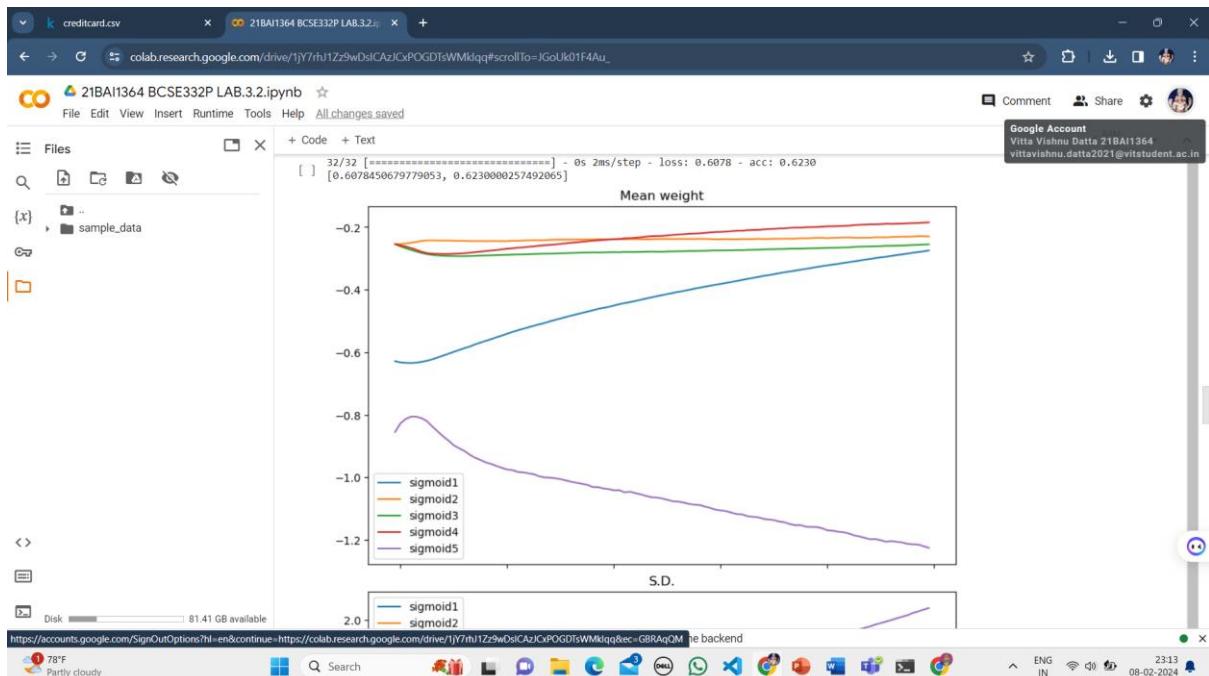
Disk 81.41 GB available

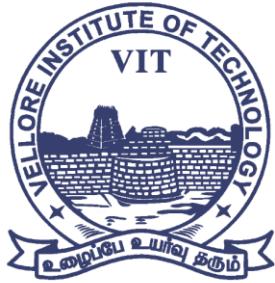
78°F Partly cloudy

https://accounts.google.com/SignOutOptions?hl=en&continue=https://colab.research.google.com/drive/1jY7rhJ1Zz9wDsICazJCxPOGDtsWMklqq&ec=GBRAqQM

Backend

ENG IN 25% 08-02-2024

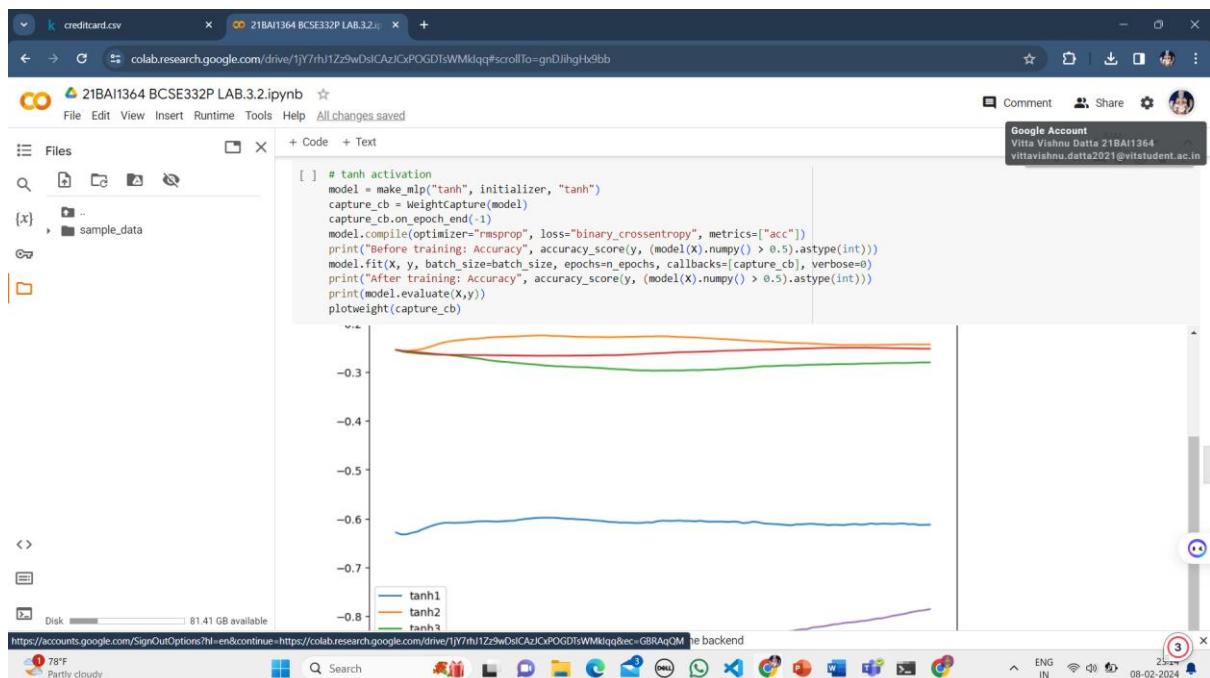
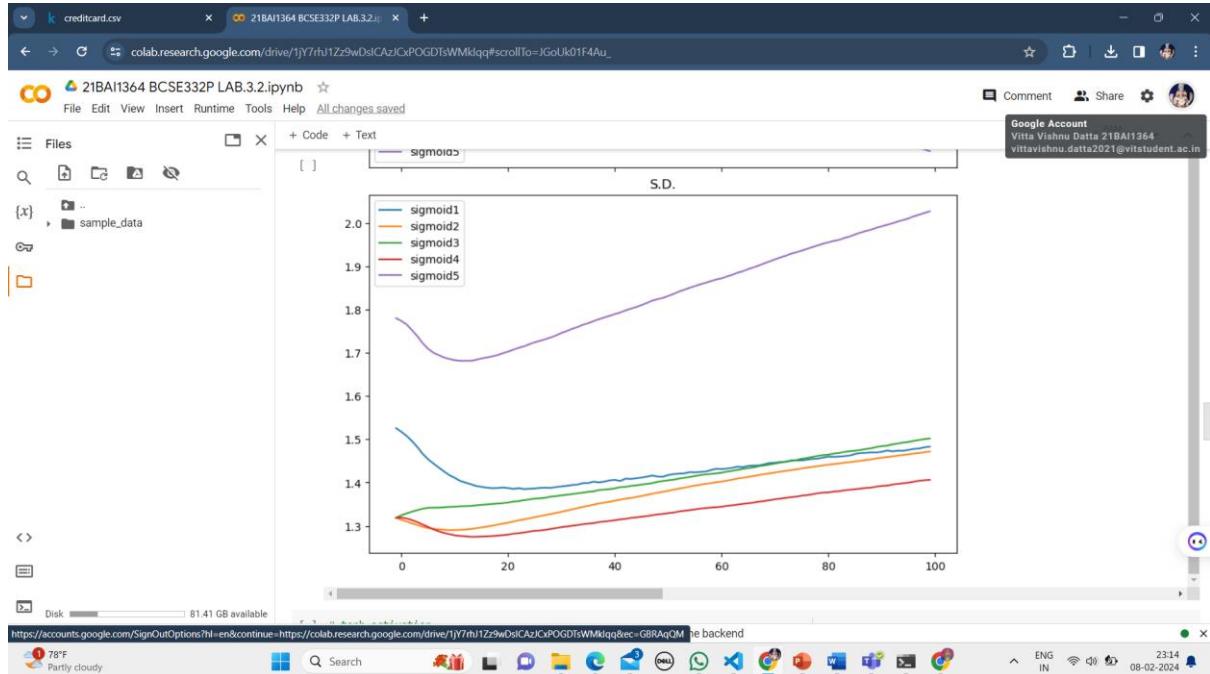


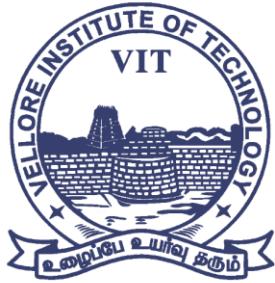


# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

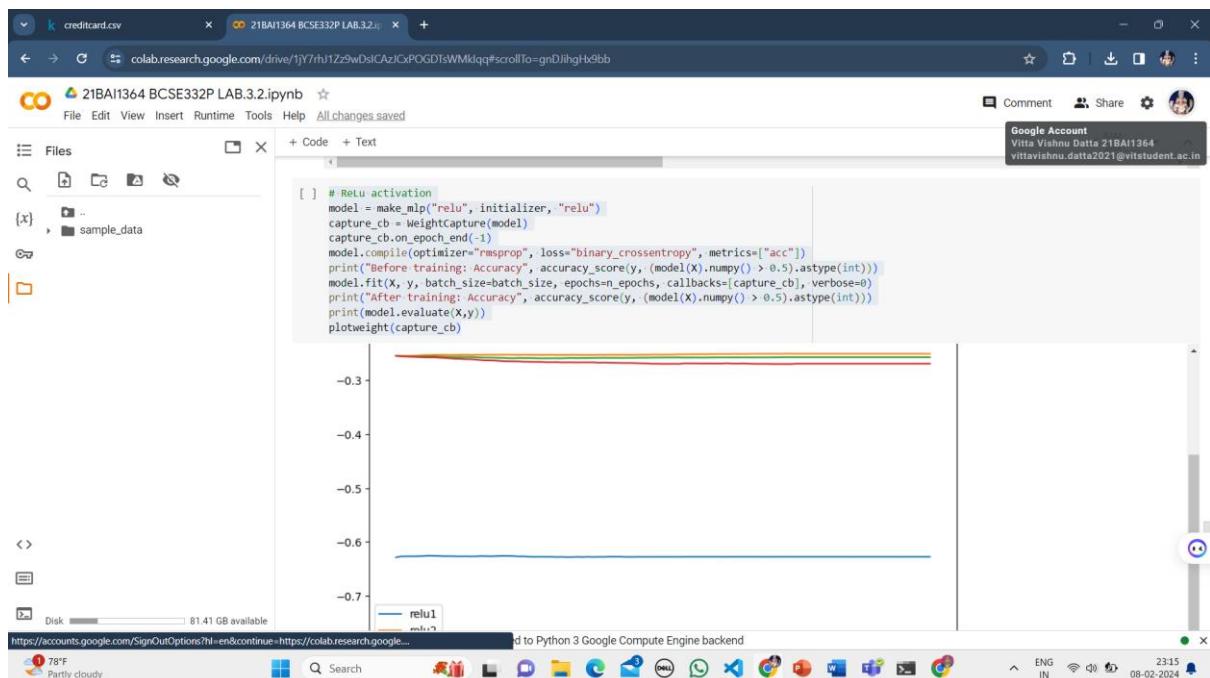
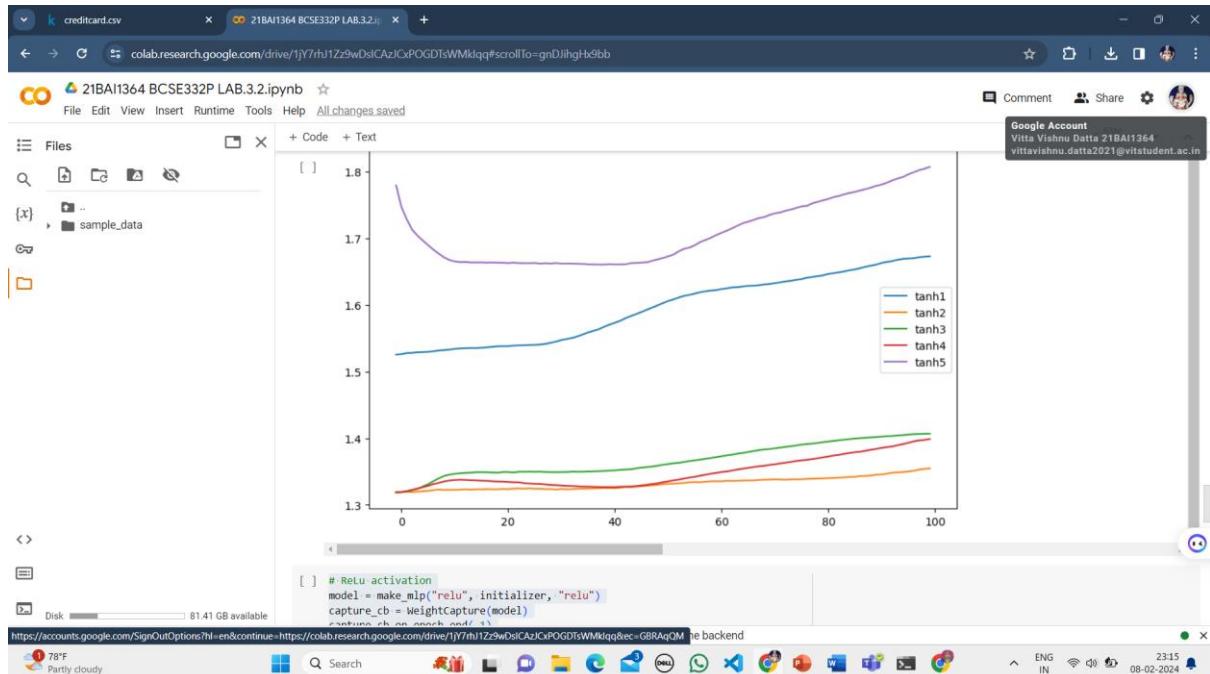


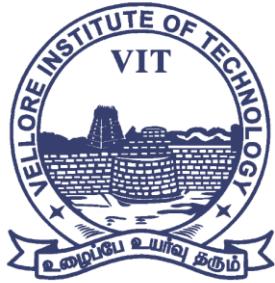


# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

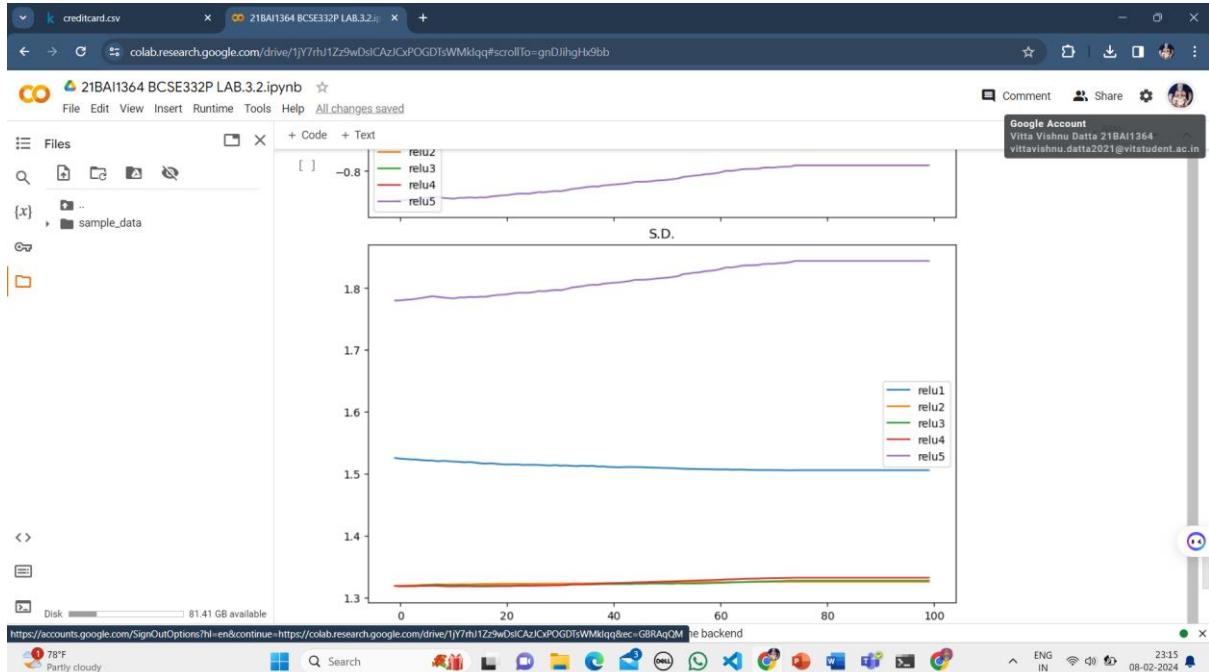




# VIT®

## Vellore Institute of Technology

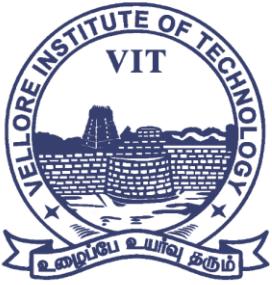
(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI



3. Implement a code to compare GD, SGD, Minibatch SGD for any input/neural model of your choice.

### CODE:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
X_test = scaler.transform(X_test)
# Build a simple neural network model
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model
# Train the model using different optimizers
def train_model(optimizer, epochs=50, batch_size=None):
    model = build_model()
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

    history = None
    if batch_size:
        history = model.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size, validation_data=(X_test, y_test), verbose=0)
    else:
        history = model.fit(X_train, y_train, epochs=epochs,
validation_data=(X_test, y_test), verbose=0)

    # Evaluate the model on test set
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f'\nOptimizer: {optimizer.get_config()["name"]}')
    print(f'Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}')

    # Make predictions
    predictions = model.predict(X_test)
    print(predictions)

    # Plot training history
    plot_training_history(history, optimizer.get_config()["name"])

    # Plot ROC curve
    plot_roc_curve(y_test, predictions, optimizer.get_config()["name"])

# Plot training history
def plot_training_history(history, optimizer_name):
    plt.figure(figsize=(12, 4))

    # Plot training & validation accuracy values
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title(f'Model Accuracy - {optimizer_name}')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title(f'Model Loss - {optimizer_name}')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()
```

```
# Plot ROC curve
def plot_roc_curve(y_true, y_pred, optimizer_name):
    from sklearn.metrics import roc_curve, auc

    fpr, tpr, thresholds = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve
(area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {optimizer_name}')
    plt.legend(loc='lower right')
    plt.show()
```

```
# Compare GD, SGD, and Mini-batch SGD
epochs = 50
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

```
# Gradient Descent
gd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(gd_optimizer, epochs=epochs)

# Stochastic Gradient Descent
sgd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(sgd_optimizer, epochs=epochs)

# Mini-batch Stochastic Gradient Descent
batch_size = 32
minibatch_sgd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(minibatch_sgd_optimizer, epochs=epochs,
batch_size=batch_size)
```

### LINK TO THE GOOGLE COLAB NOTEBOOK:

<https://colab.research.google.com/drive/1XSPfKF55OUW0YKZLdHdBLNfpGuZWCCrt#scrollTo=39207b41-502d-49e6-8de7-0ca37f3e74e7>

### OUTPUT OF THE CODE SNIPPET:

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.3.ipynb". The code cell contains the following Python code:

```
[1] import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

[2] # Generate synthetic dataset
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

[3] # Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

[4] # Build a simple neural network model
def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
    model.add(layers.Dense(1, activation='sigmoid'))
    return model

[5] # Train the model using different optimizers
def train_model(optimizer, epochs=50, batch_size=None):
    model = build_model()
```

The notebook interface shows the code being run, with the status bar indicating "Running" and the progress bar showing completion. The user's Google Account information is visible in the top right corner.



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.3.ipynb". The code cell [5] contains Python code for training a model using different optimizers (SGD, Adam, RMSprop) and evaluating it on a test set. It also plots training history and ROC curves. The code cell [6] is partially visible below it. The notebook interface includes a file browser on the left, a code editor with syntax highlighting, and a toolbar with various tools. The status bar at the bottom shows the user's account information and the date/time.

```
# Train the model using different optimizers
def train_model(optimizer, epochs=50, batch_size=None):
    model = build_model()
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    history = None
    if batch_size:
        history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test), verbose=0)
    else:
        history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=0)

    # Evaluate the model on test set
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f'\nOptimizer: {optimizer.get_config()["name"]}')
    print(f'Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}')

    # Make predictions
    predictions = model.predict(X_test)
    print(predictions)

    # Plot training history
    plot_training_history(history, optimizer.get_config()["name"])

    # Plot ROC curve
    plot_roc_curve(y_test, predictions, optimizer.get_config()["name"])

# Plot training history
def plot_training_history(history, optimizer_name):
    plt.figure(figsize=(12, 4))

[6] # Plot training history
def plot_training_history(history, optimizer_name):
    plt.figure(figsize=(12, 4))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'Model Accuracy - ({optimizer_name})')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

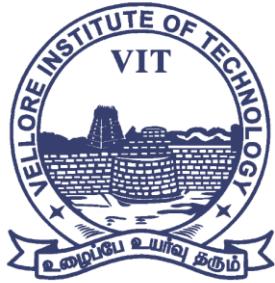
    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'Model Loss - ({optimizer_name})')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()
```

This screenshot shows the same Google Colab notebook and code as the previous one, but the code cell [6] is fully visible. It contains code for plotting training history and ROC curves. The notebook interface and status bar are identical.

```
# Plot ROC curve
def plot_roc_curve(y_true, y_pred, optimizer_name):
    from sklearn.metrics import roc_curve, auc

[7] # Plot ROC curve
def plot_roc_curve(y_true, y_pred, optimizer_name):
    from sklearn.metrics import roc_curve, auc
```



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

The screenshot shows a Google Colab notebook titled "21BAI1364 BCSE332P LAB.3.3.ipynb". The code cell [7] contains a function to plot an ROC curve, which is then executed. The output shows the ROC curve with an AUC of approximately 0.86. Cell [8] compares the performance of Gradient Descent (GD), Stochastic Gradient Descent (SGD), and Mini-batch SGD. The SGD and Mini-batch SGD results are identical, showing a loss of 0.3612 and an accuracy of 0.8600. The output for GD shows a loss of 0.8737522 and an accuracy of 0.4455164. The output for SGD and Mini-batch SGD is as follows:

```
[9] # Gradient Descent
gd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(gd_optimizer, epochs=epochs)
[0.8737522]
[0.9870218]
[0.9776322]
[0.4455164]
[0.86709195]
```

The notebook also includes sections for "Model Accuracy - SGD" and "Model Loss - SGD". The status bar at the bottom indicates it's 78°F Partly cloudy, ENG IN, 23:31, 08-02-2024.

This screenshot shows the continuation of the Google Colab notebook. The code cell [9] has been run, and its output is displayed. The output shows the training progress of the SGD optimizer, indicating a loss of 0.3612 and an accuracy of 0.8600. The output for SGD is as follows:

```
[9] # Gradient Descent
gd_optimizer = optimizers.SGD(learning_rate=0.01)
train_model(gd_optimizer, epochs=epochs)
7/7 [=====] - 0s 3ms/step - loss: 0.3612 - accuracy: 0.8600
Optimizer: SGD
Test loss: 0.3612, Test Accuracy: 0.8600
7/7 [=====] - 0s 2ms/step
[[0.4334706]
[0.70308864]
[0.4266868]
[0.7969663]
[0.8780033]
[0.19595109]
[0.24782598]
[0.76457614]
[0.7274654]
[0.55592245]
[0.88541335]
[0.45303386]
[0.01337167]
[0.57266784]
[0.73997533]
[0.3011468]
[0.07394756]
[0.91655153]
[0.77218306]
[0.62539065]
[0.9649965]
[0.21673346]
[0.17623289]
[0.06532242]]
```

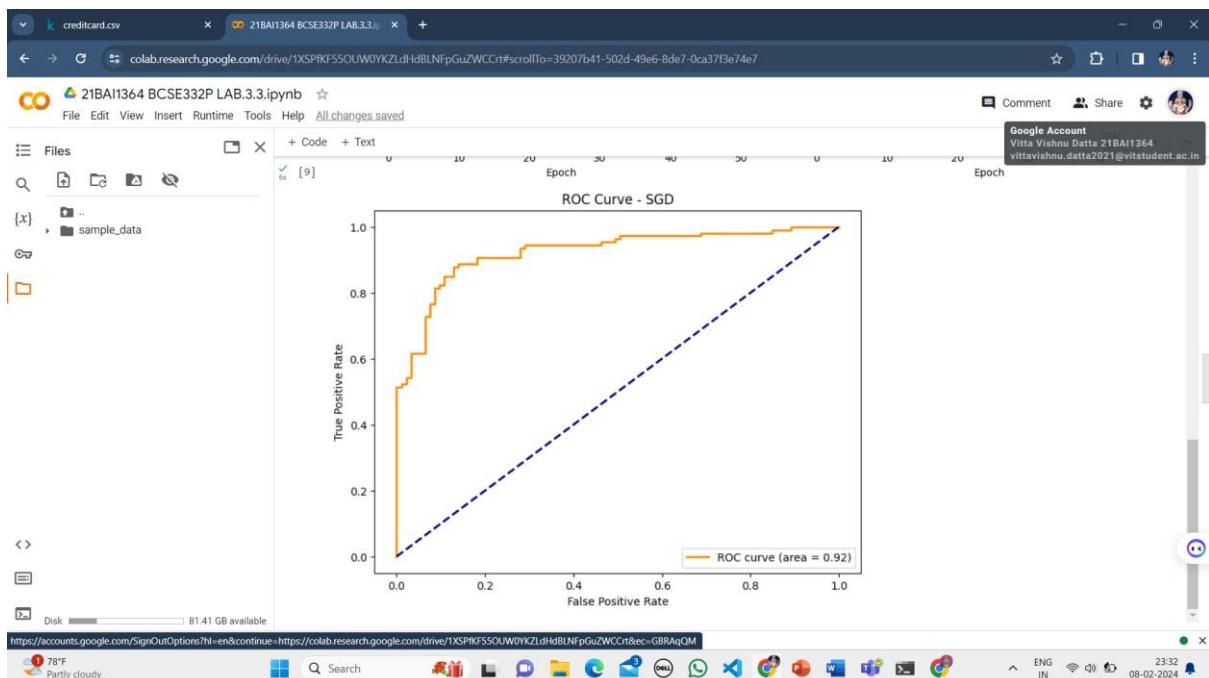
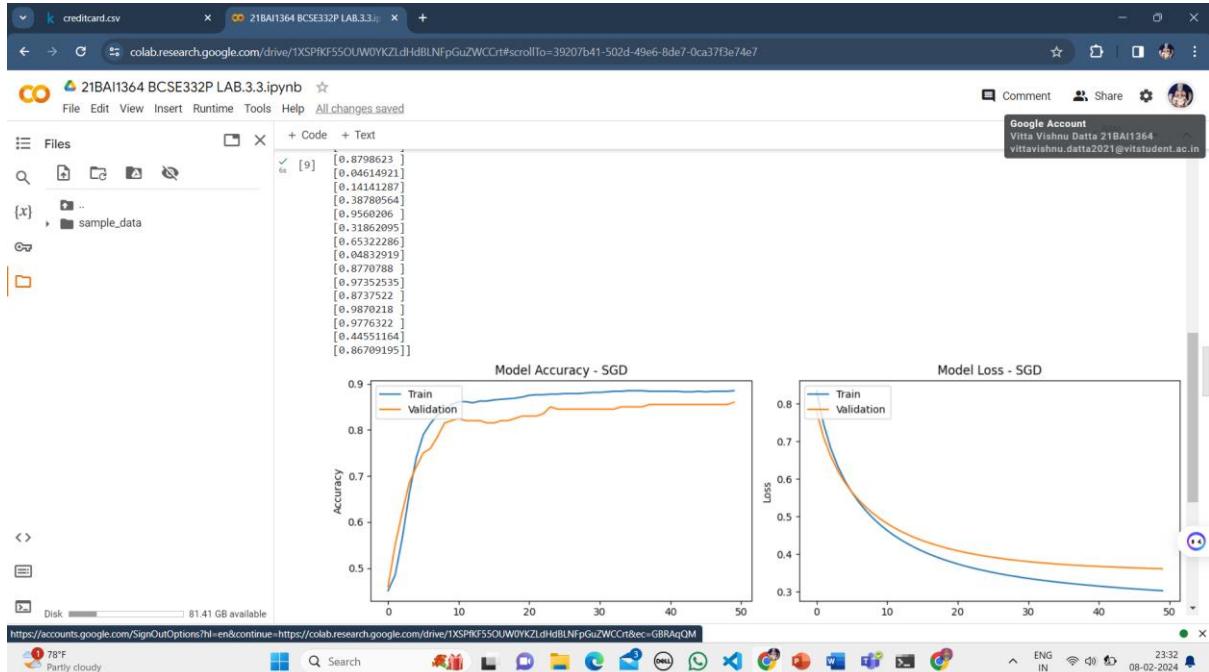
The status bar at the bottom indicates it's 78°F Partly cloudy, ENG IN, 23:31, 08-02-2024.

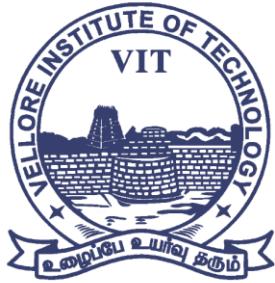


# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI





# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

creditcard.csv    21BAI1364 BCSE332P LAB.3.3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files    + Code + Text

[x] .. sample\_data

Model Accuracy - SGD

# Stochastic Gradient Descent  
sgd\_optimizer = optimizers.SGD(learning\_rate=0.01)  
train\_model(sgd\_optimizer, epochs=epochs)  
[0.9224882 ]  
[0.9924371 ]  
[0.94340944]  
[0.16461591]  
[0.7995752 ]]

Model Loss - SGD

Train Validation

ROC Curve - SGD

Epoch

Accuracy

Loss

Disk 81.41 GB available

78°F Partly cloudy

https://accounts.google.com/SignOutOptions?hl=en&continue=https://colab.research.google.com/drive/1XSPkF55OUW0YKZLdIdBLNfpGuZWCCrt&ec=GRB4qQM

Comment Share Google Account Vitta Vishnu Datta 21BAI1364 vittavishnu.datta2021@vitstudent.ac.in

ENG IN 23:32 08-02-2024

creditcard.csv    21BAI1364 BCSE332P LAB.3.3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files    + Code + Text

[x] .. sample\_data

ROC Curve - SGD

Epoch

True Positive Rate

False Positive Rate

ROC curve (area = 0.92)

Disk 81.41 GB available

78°F Partly cloudy

https://accounts.google.com/SignOutOptions?hl=en&continue=https://colab.research.google.com/drive/1XSPkF55OUW0YKZLdIdBLNfpGuZWCCrt&ec=GRB4qQM

completed at 11:29 PM

Comment Share Google Account Vitta Vishnu Datta 21BAI1364 vittavishnu.datta2021@vitstudent.ac.in

ENG IN 23:32 08-02-2024



# VIT®

## Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)  
CHENNAI

creditcard.csv    21BAI1364 BCSE332P LAB.3.3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files    + Code + Text

# Mini-batch Stochastic Gradient Descent  
batch\_size = 32  
minibatch\_sgd\_optimizer = optimizers.SGD(learning\_rate=0.01)  
train\_model(minibatch\_sgd\_optimizer, epochs=epochs, batch\_size=batch\_size)

[0.969891 ]  
[0.9931721 ]  
[0.9441694 ]  
[0.28883874]  
[0.988873 ]]

Model Accuracy - SGD

Model Loss - SGD

ROC Curve - SGD

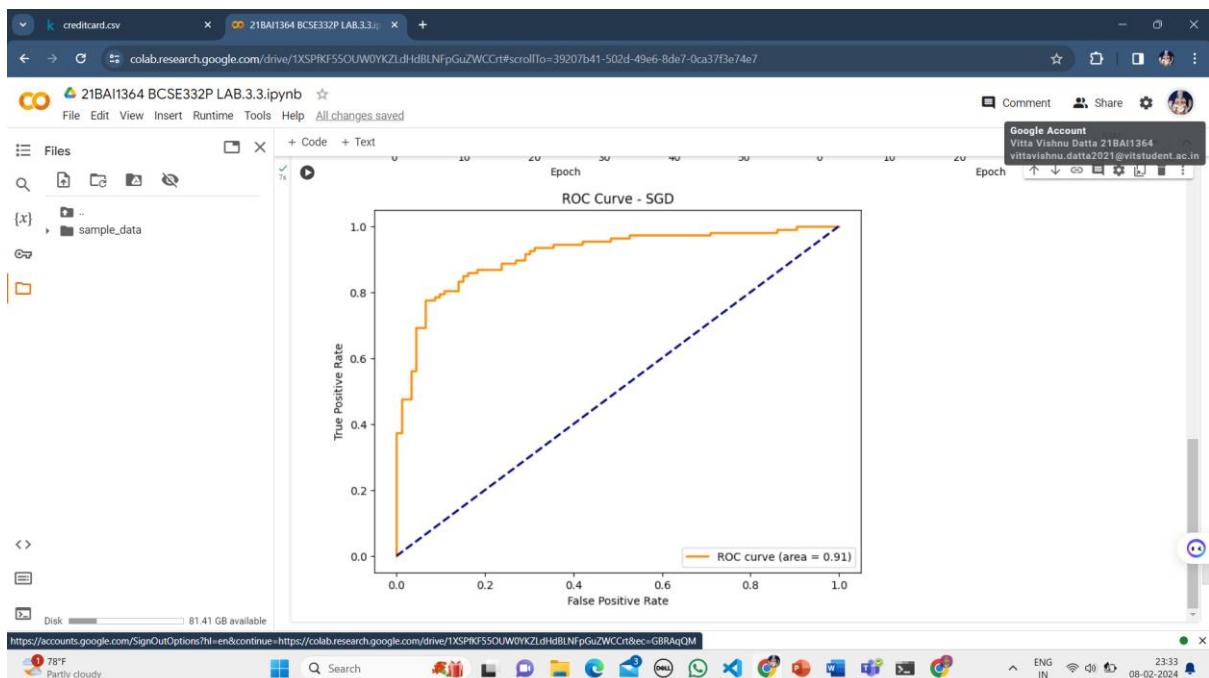
Disk 81.41 GB available

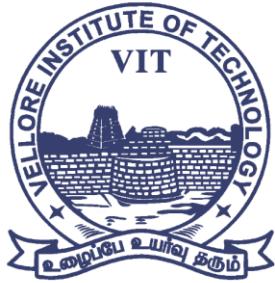
https://accounts.google.com/SignOutOptions?hl=en&continue=https://colab.research.google.com/

78°F Partly cloudy

Search    6s completed at 11:29 PM

ENG IN 23:33 08-02-2024





**VIT**<sup>®</sup>

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)  
**CHENNAI**



**PREPARED AND SUBMITTED BY:**

**VITTA VISHNU DATTA.**

**21BAI1364 – SCOPE.**

**VIT – CHENNAI CAMPUS.**