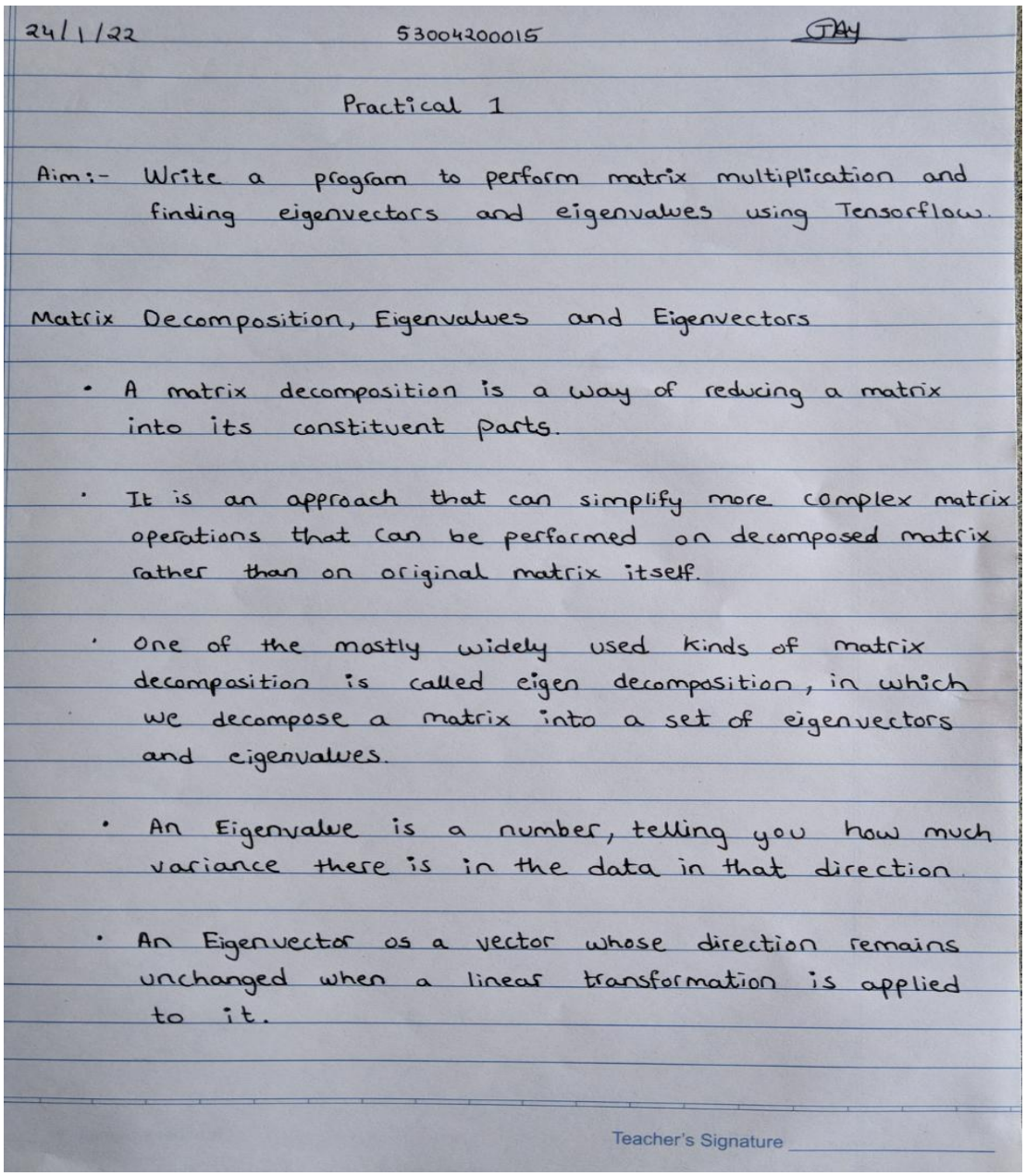
Practical 1

Write a program to perform matrix multiplication and finding eigenvectors and eigenvalues using TensorFlow.

Matrix Decomposition, Eigenvalues and Eigenvectors



24/1/22 53004200015 JAY

Practical 1

Aim:- Write a program to perform matrix multiplication and finding eigenvectors and eigenvalues using Tensorflow.

Matrix Decomposition, Eigenvalues and Eigenvectors

- A matrix decomposition is a way of reducing a matrix into its constituent parts.
- It is an approach that can simplify more complex matrix operations that can be performed on decomposed matrix rather than on original matrix itself.
- One of the mostly widely used kinds of matrix decomposition is called eigen decomposition, in which we decompose a matrix into a set of eigenvectors and eigenvalues.
- An Eigenvalue is a number, telling you how much variance there is in the data in that direction.
- An Eigenvector as a vector whose direction remains unchanged when a linear transformation is applied to it.

Teacher's Signature _____

53004200015

- An Eigenvector of a square matrix A is a non zero vector v such that multiplication by A alters only the scale of v, in short, this is a special vector that doesn't change the direction of the matrix when applied to it

$$Av = \lambda v$$

The scale λ is known as the Eignvalue corresponding to this eigenvector.

- Properties of Eigenvectors and Eigenvalues
  - There could be between 0 and n eigenvalues and eigenvectors for an nxn matrix.
  - Eigenvectors and Eigenvalues are not defined for rectangular matrices.
  - All eigenvalues of a positive definite matrix are positive.
  - All eigenvalues of a positive semi-definite matrix are non-negative.

- Application of Eigenvalues and Eigenvectors
  - Data compression
  - Dimensionality reduction
  - sparsity learning

Teacher's Signature

Problem Based Example

53004200015 JAY

Example

Eigenvector of A is

$$v = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$$

because

$$\begin{bmatrix} 5 & 1 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ -3 \end{bmatrix} = \begin{bmatrix} 2 \\ -6 \end{bmatrix}$$

and

$$2 \times \begin{bmatrix} 1 \\ -3 \end{bmatrix} = \begin{bmatrix} 2 \\ -6 \end{bmatrix}$$

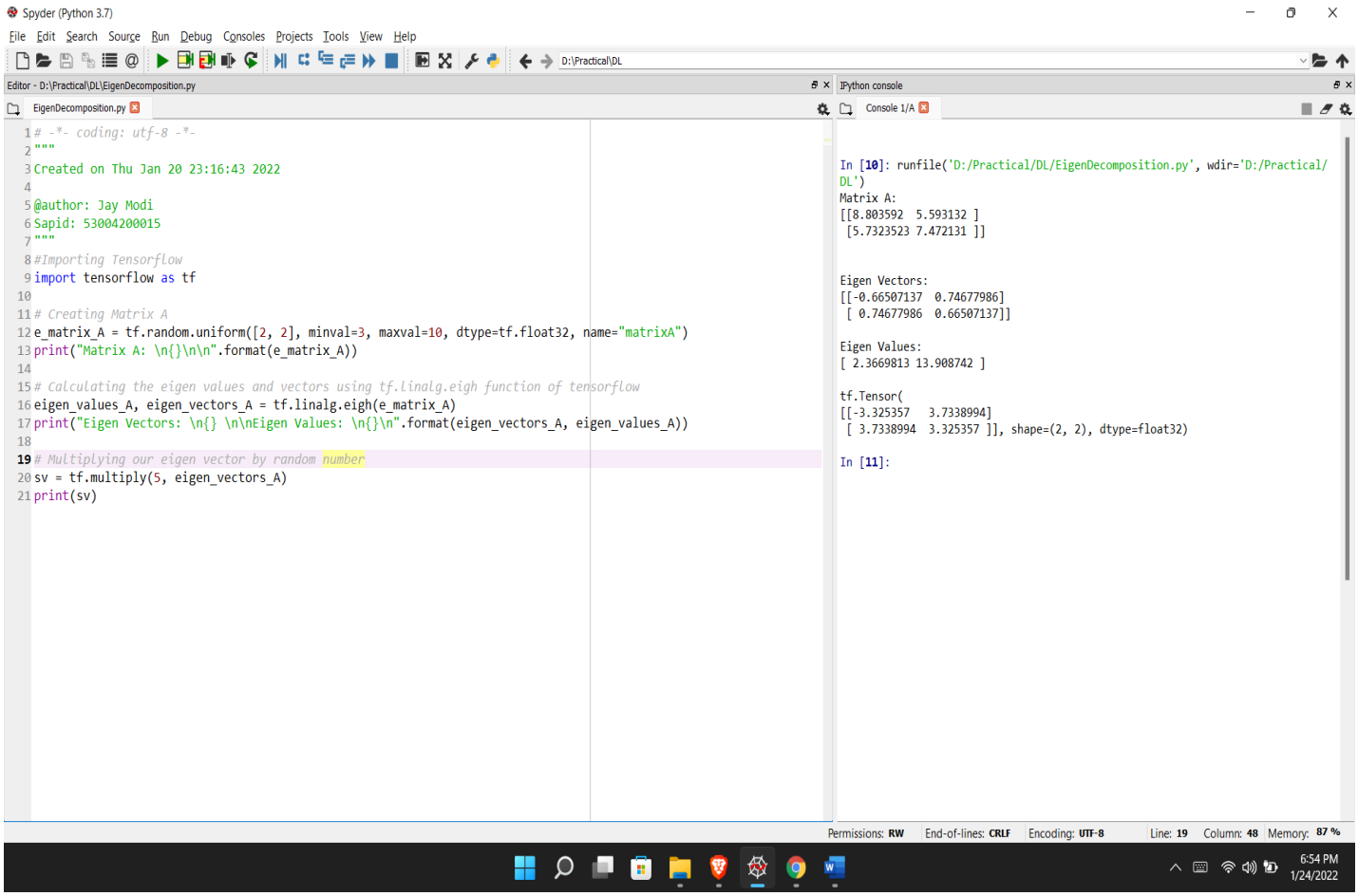
so the corresponding eigenvalue is $\lambda = 2$

Teacher's Signature _____

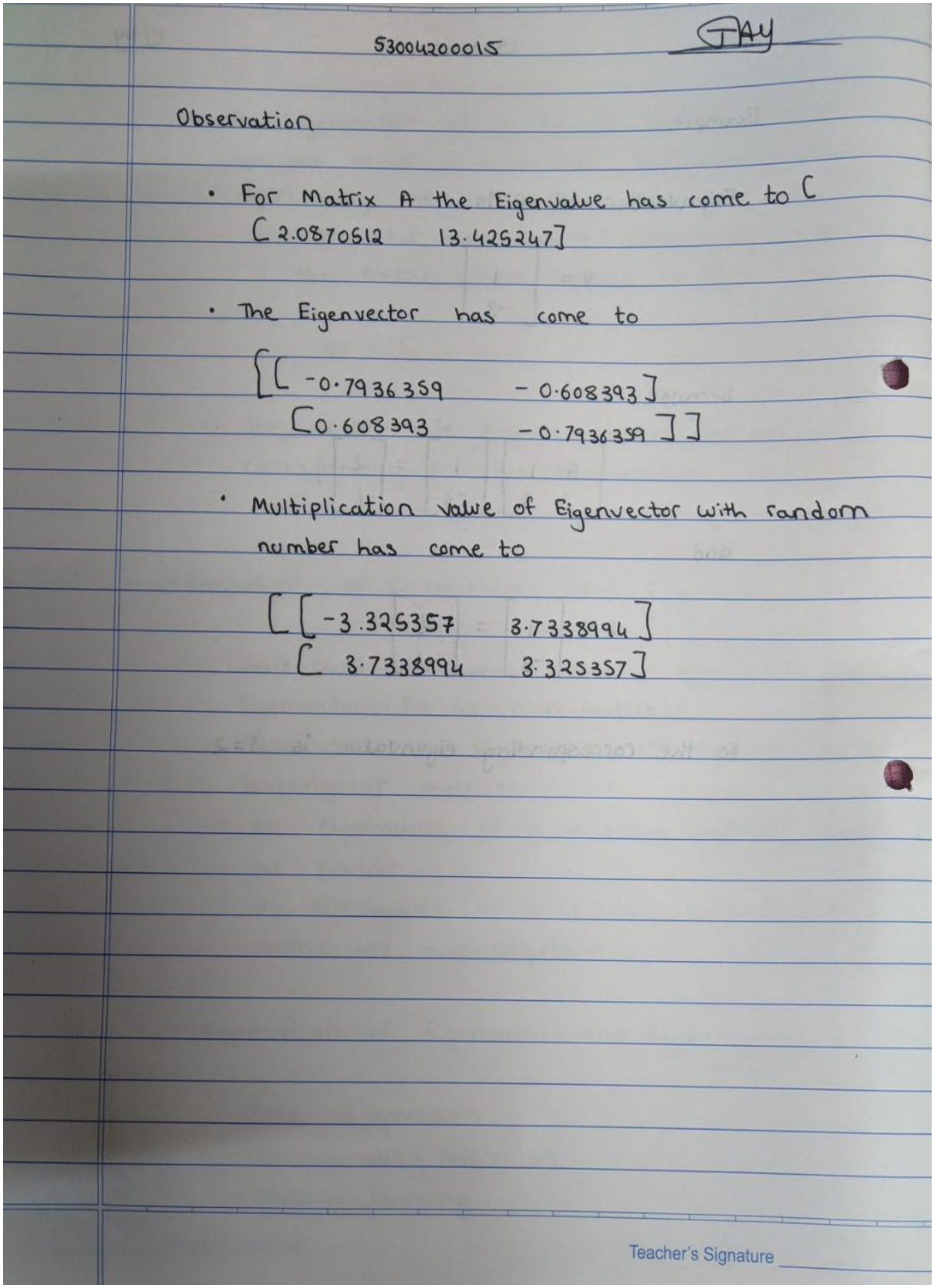
Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 20 23:16:43 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Importing Tensorflow
import tensorflow as tf
# Creating Matrix A
e_matrix_A = tf.random.uniform([2, 2], minval=3, maxval=10,
dtype=tf.float32, name="matrixA")
print("Matrix A: \n{}\n\n".format(e_matrix_A))
# Calculating the eigen values and vectors using tf.linalg.eigh function of
tensorflow
eigen_values_A, eigen_vectors_A = tf.linalg.eigh(e_matrix_A)
print("Eigen Vectors: \n{} \n\nEigen Values: \n{}\n".format(eigen_vectors_A,
eigen_values_A))
# Multiplying our eigen vector by random number
sv = tf.multiply(5, eigen_vectors_A)
print(sv)
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Practical\DL
Editor - D:\Practical\DL\EigenDecomposition.py
EigenDecomposition.py
-*- coding: utf-8 -*-
"""
Created on Thu Jan 20 23:16:43 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Importing Tensorflow
import tensorflow as tf
# Creating Matrix A
e_matrix_A = tf.random.uniform([2, 2], minval=3, maxval=10, dtype=tf.float32, name="matrixA")
print("Matrix A: \n{}\n\n".format(e_matrix_A))
# Calculating the eigen values and vectors using tf.linalg.eigh function of tensorflow
eigen_values_A, eigen_vectors_A = tf.linalg.eigh(e_matrix_A)
print("Eigen Vectors: \n{} \n\nEigen Values: \n{}\n".format(eigen_vectors_A, eigen_values_A))
# Multiplying our eigen vector by random number
sv = tf.multiply(5, eigen_vectors_A)
print(sv)
IPython console
Console 1/A
In [10]: runfile('D:/Practical/DL/EigenDecomposition.py', wdir='D:/Practical/DL')
Matrix A:
[[8.803592 5.593132]
[5.7323523 7.472131]]
Eigen Vectors:
[[-0.66507137 0.74677986]
[0.74677986 0.66507137]]
Eigen Values:
[2.3669813 13.908742]
tf.Tensor(
[[-3.325357 3.7338994]
[3.7338994 3.325357]], shape=(2, 2), dtype=float32)
In [11]:
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 19
Column: 48
Memory: 87 %
6:54 PM
1/24/2022

Observation



53004200015

Observation

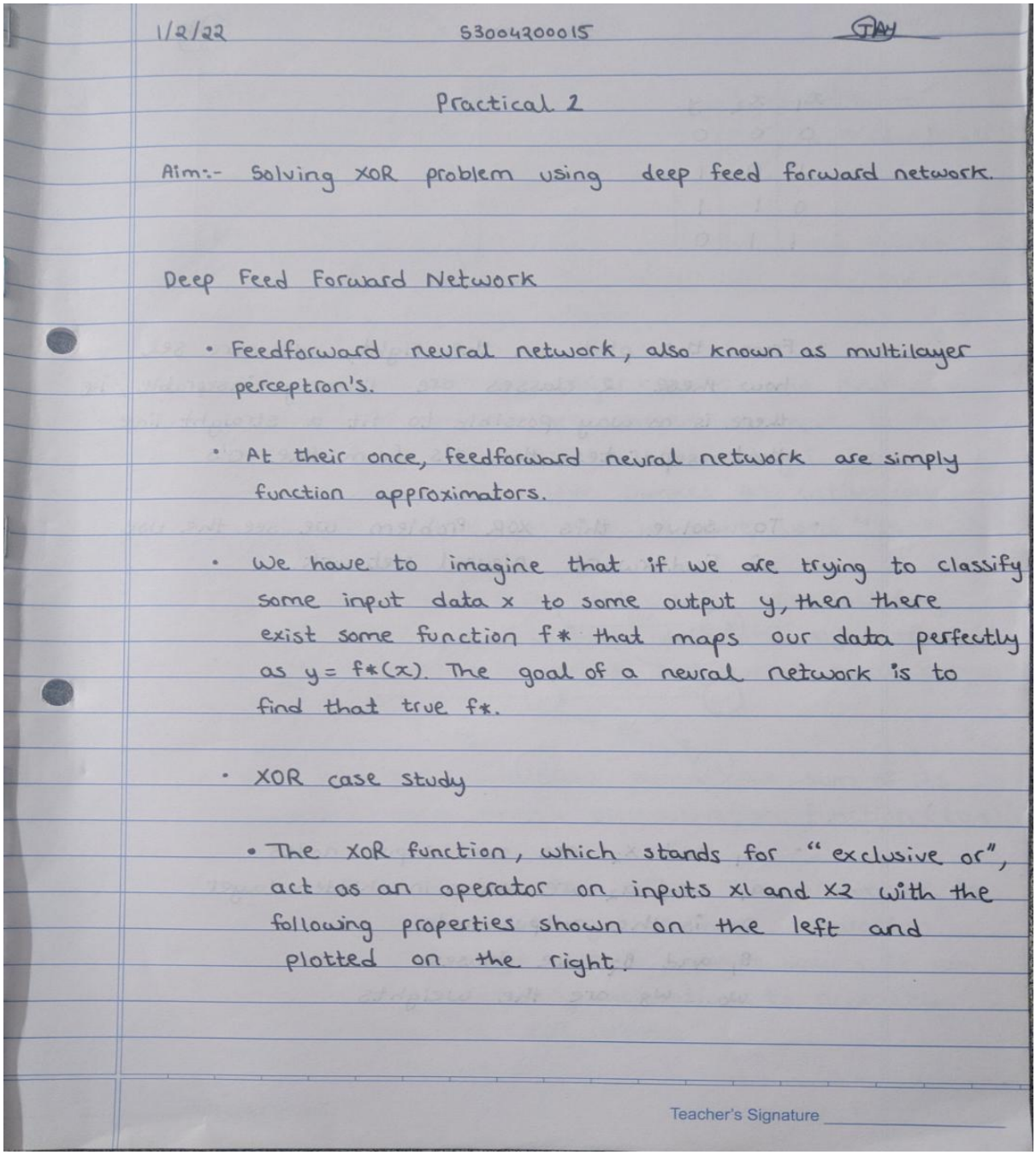
- For Matrix A the Eigenvalue has come to
  [2.0870512 13.425247]
- The Eigenvector has come to
  [[-0.7936359 -0.608393]
  [0.608393 -0.7936359]]
- Multiplication value of Eigenvector with random number has come to
  [[-3.325357 3.7338994]
  [3.7338994 3.325357]]

Teacher's Signature

Practical 2

Solving XOR problem using deep feed forward network.

Deep Feed Forward Network



1/2/22 53004200015

Practical 2

Aim:- Solving XOR problem using deep feed forward network.

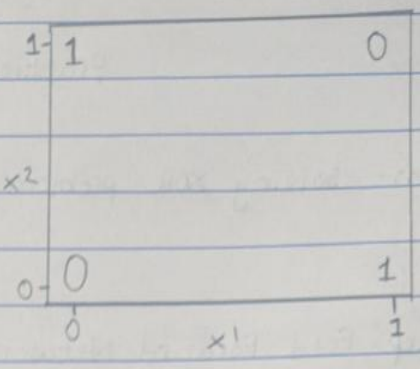
Deep Feed Forward Network

- Feedforward neural network, also known as multilayer perceptron's.
- At their once, feedforward neural network are simply function approximators.
- We have to imagine that if we are trying to classify some input data x to some output y, then there exist some function f* that maps our data perfectly as y = f*(x). The goal of a neural network is to find that true f*.
- XOR case study
- The XOR function, which stands for "exclusive or", act as an operator on inputs x1 and x2 with the following properties shown on the left and plotted on the right.

Teacher's Signature _____

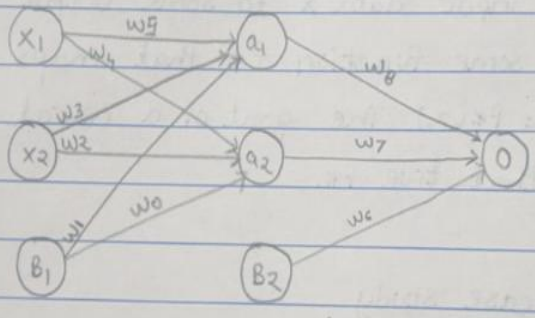
53004200015

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0



- From the plot on the right, we can see how these 2 classes are linearly inseprable, i.e there is no way possible to fit a straight line that separates the 1's from the 0's.

- To solve this XOR Problem we see the use of FeedForward Neural Network.



- X_1 and X_2 are our input nodes.
  a_1 and a_2 are nodes in hidden layer
  0 is the output node.
  B_1 and B_2 are biases.
  $w_0 \ldots w_8$ are the weights.

Teacher's Signature

53004200015

- There are two inputs types and one target variable. Model architecture is Input:2, Hidden layer:2 output:1

- Forward propogation begins with the input values and bias unit from the input layer being multiplied by their respective weights.

- The product of input layer values and their respective weights are parsed as input to the non-bias unit in the hidden layer. Each non-bias hidden unit invokes an activation function (in our case we have used "tanh")

- The output of each hidden layer unit, including the bias unit, are then multiplied by another set of respective weights and parsed to an output unit.

- The output unit also parses the sum of its input value through an activation function (tanh)

- This is the predicted output. This archieture is capable of achieving non-linear separation. Thus, with right set of weight values, it can provide neccessary separation to accurately classify the XOR inputs.

Teacher's Signature _____

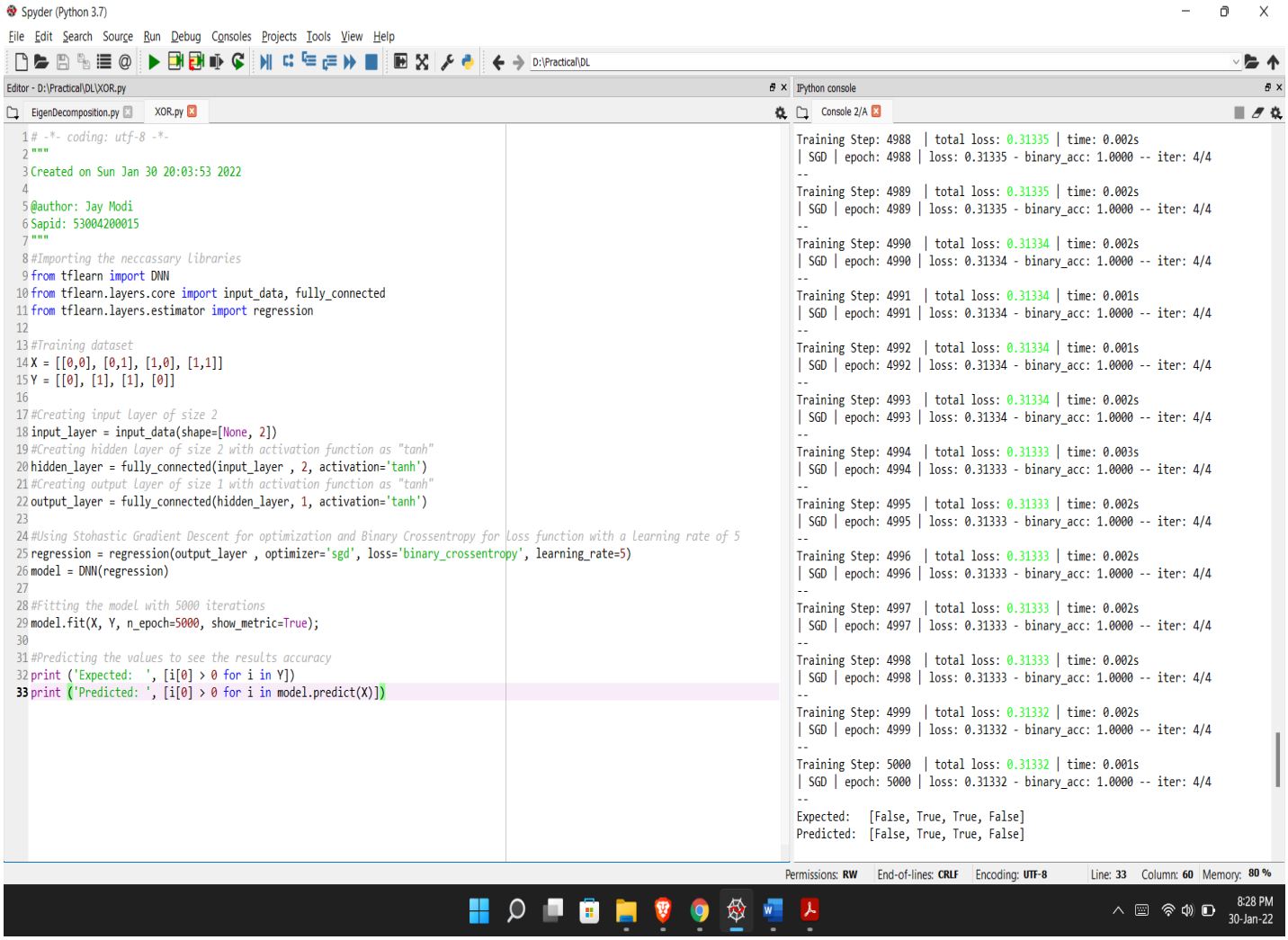
Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 30 20:03:53 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Importing the neccassary libraries
from tflearn import DNN
from tflearn.layers.core import input_data, fully_connected
from tflearn.layers.estimator import regression
#Training dataset
X = [[0,0], [0,1], [1,0], [1,1]]
Y = [[0], [1], [1], [0]]
#Creating input layer of size 2
input_layer = input_data(shape=[None, 2])
#Creating hidden layer of size 2 with activation function as "tanh"
hidden_layer = fully_connected(input_layer , 2, activation='tanh')
#Creating output layer of size 1 with activation function as "tanh"
output_layer = fully_connected(hidden_layer, 1, activation='tanh')
#Using Stohastic Gradient Descent for optimization and Binary Crossentropy for loss function with a learning rate of 5
regression = regression(output_layer , optimizer='sgd', loss='binary_crossentropy', learning_rate=5)
model = DNN(regression)
#Fitting the model with 5000 iterations
model.fit(X, Y, n_epoch=5000, show_metric=True);
#Predicting the values to see the results accuracy
```

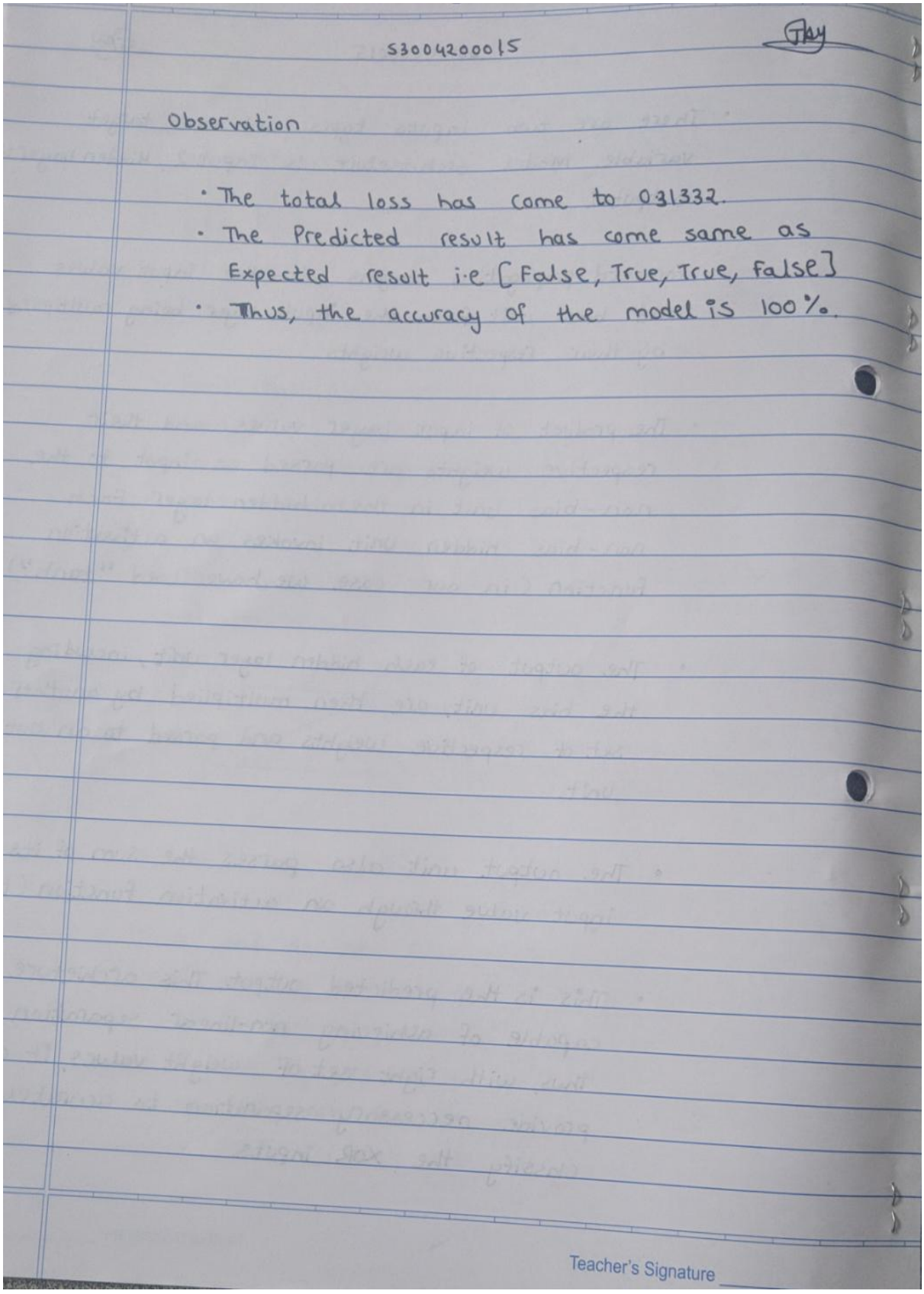
print ('Expected: ', [i[0] > 0 for i in Y])

print ('Predicted: ', [i[0] > 0 for i in model.predict(X)])

Output



Observation



53004200015

Observation

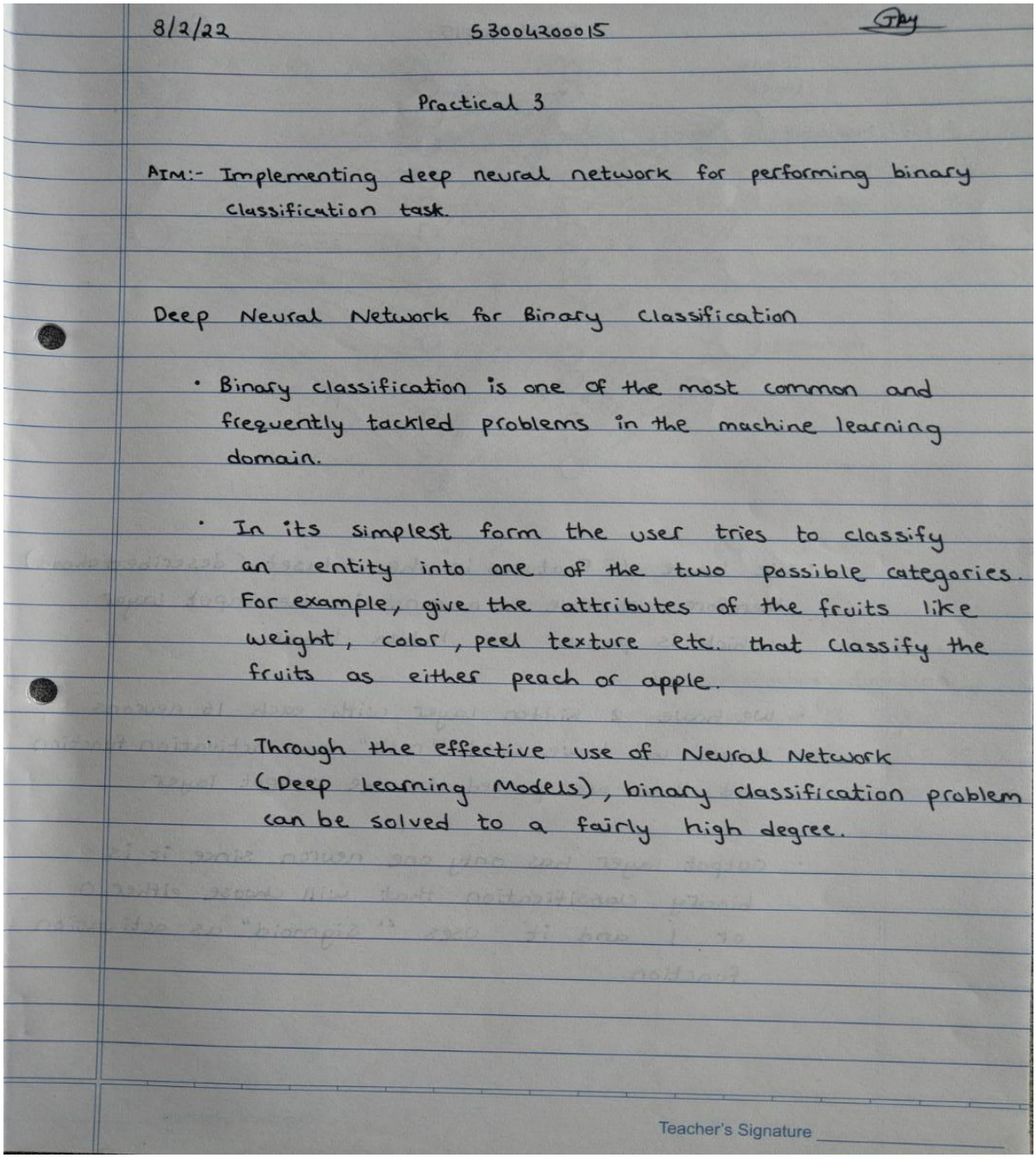
- The total loss has come to 0.31332.
- The Predicted result has come same as Expected result i.e [False, True, True, False]
- Thus, the accuracy of the model is 100%.

Teacher's Signature

Practical 2

Implementing deep neural network for performing binary classification task

Deep Neural Network for Binary classification



8/2/22 53004200015

Practical 3

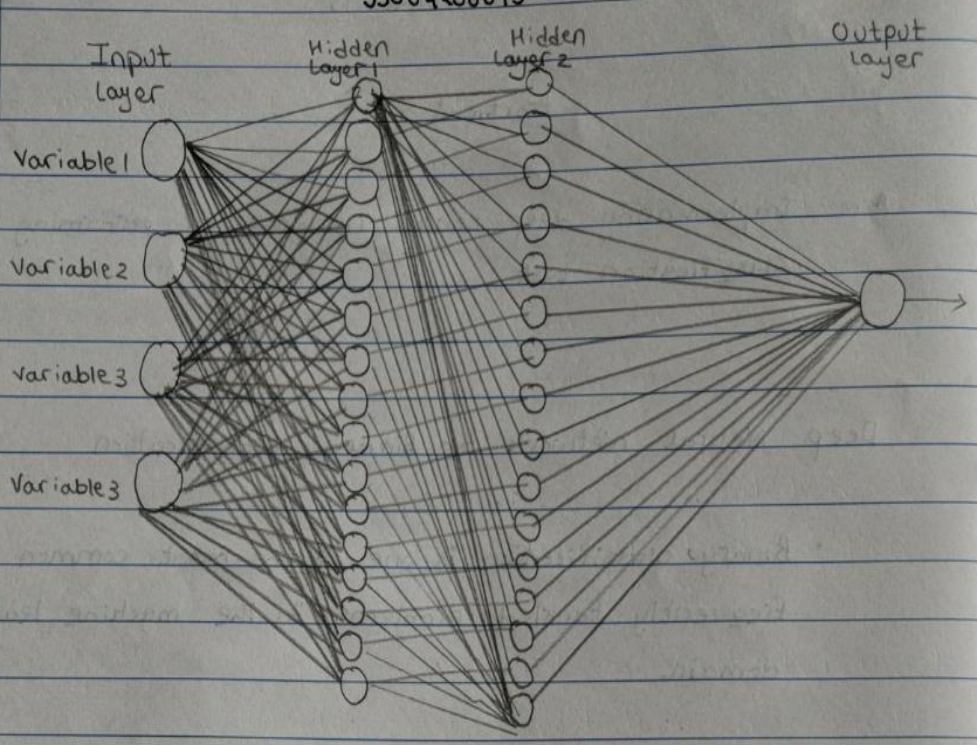
AIM:- Implementing deep neural network for performing binary classification task.

Deep Neural Network for Binary Classification

- Binary classification is one of the most common and frequently tackled problems in the machine learning domain.
- In its simplest form the user tries to classify an entity into one of the two possible categories. For example, give the attributes of the fruits like weight, color, peel texture etc. that classify the fruits as either peach or apple.
- Through the effective use of Neural Network (Deep Learning Models), binary classification problem can be solved to a fairly high degree.

Teacher's Signature

53004200015



- We have 4 Features in the dataset (described ahead) therefore we have 4 neurons in the Input layer which is passed to hidden Layer.

- We have 2 hidden layer with each 16 neurons where we have used "relu" as activation function and then are passed to the output layer

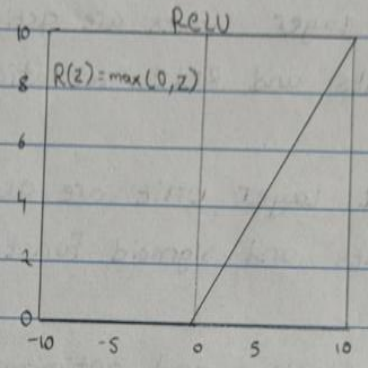
- Output layer has only one neuron since it is binary classification that will choose either 0 or 1 and it uses "sigmoid" as activation function.

Teacher's Signature

53004200015

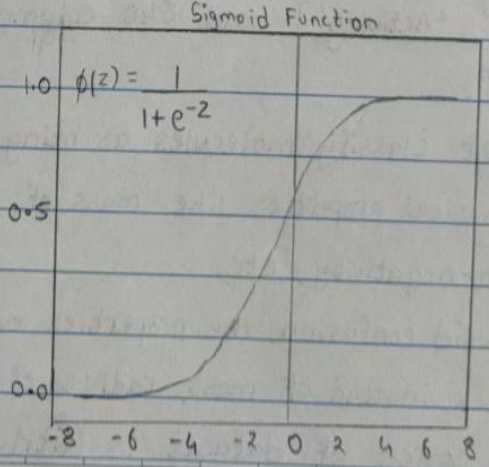
- Why Relu?

The two hidden Layer use ReLU (Rectified Linear units) as activation function because it is a half-rectified function; that is, for all the inputs less than 0 (eg:- -120, -6.7, -0.0844, 0) the value is 0 while for anything positive (eg:- 10,15,3) the value is retained.



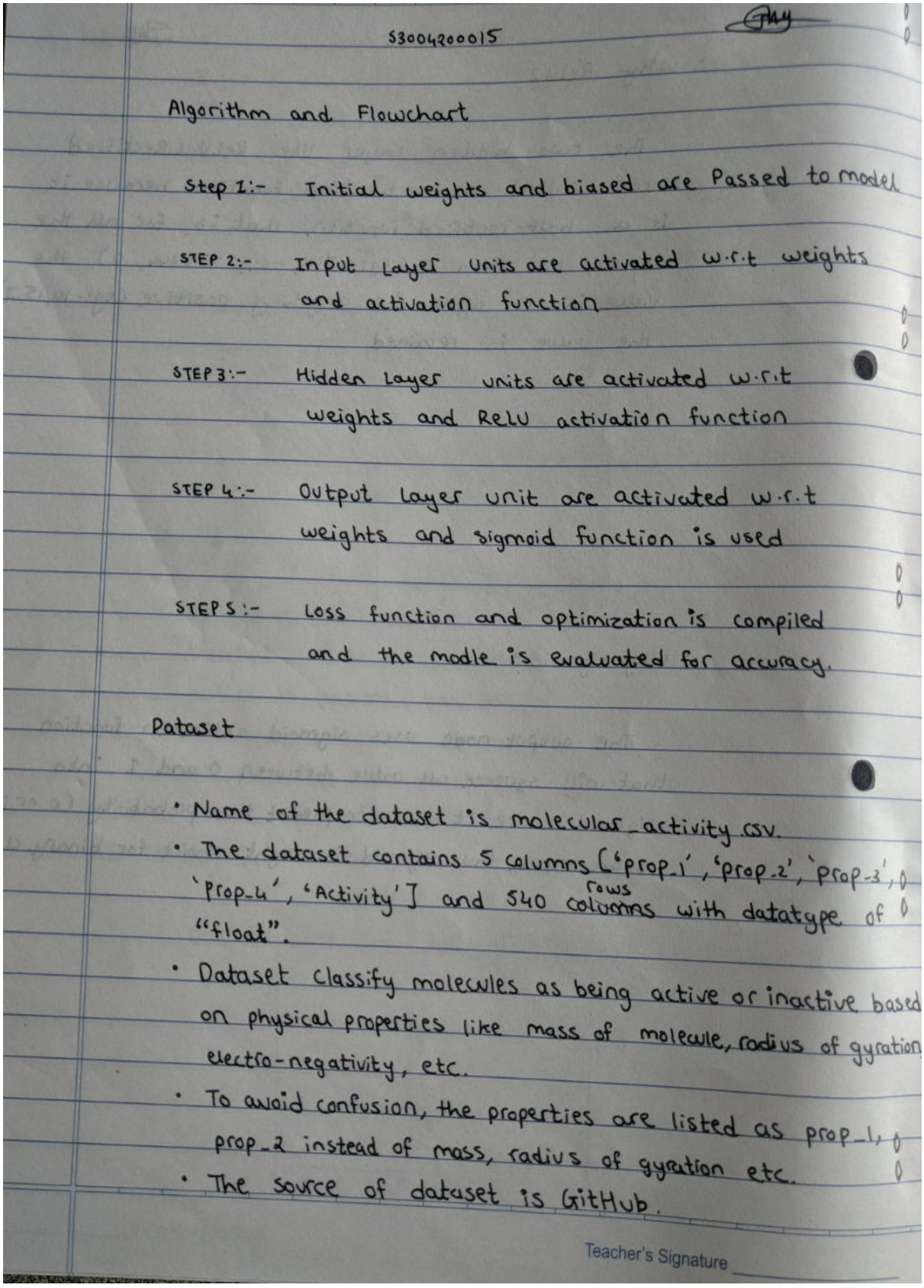
- Why sigmoid?

The output node uses sigmoid activation function that will squeeze all value between 0 and 1 into sigmoid curve. It is use to predict the probability (0 or 1). as an output, thus sigmoid is right choice for binary class.



Teacher's Signature _____

Algorithm and Dataset



53004200015

Algorithm and Flowchart

Step 1:- Initial weights and biased are Passed to model

STEP 2:- Input Layer Units are activated w.r.t weights and activation function

STEP 3:- Hidden Layer units are activated w.r.t weights and ReLU activation function

STEP 4:- Output Layer unit are activated w.r.t weights and sigmoid function is used

STEP 5:- Loss function and optimization is compiled and the modle is evaluated for accuracy.

Dataset

- Name of the dataset is molecular_activity.csv.
- The dataset contains 5 columns ['prop_1', 'prop_2', 'prop_3', 'Prop_4', 'Activity'] and 540 rows columns with datatype of "float".
- Dataset classify molecules as being active or inactive based on physical properties like mass of molecule, radius of gyration, electro-negativity, etc.
- To avoid confusion, the properties are listed as prop_1, prop_2 instead of mass, radius of gyration etc.
- The source of dataset is GitHub.

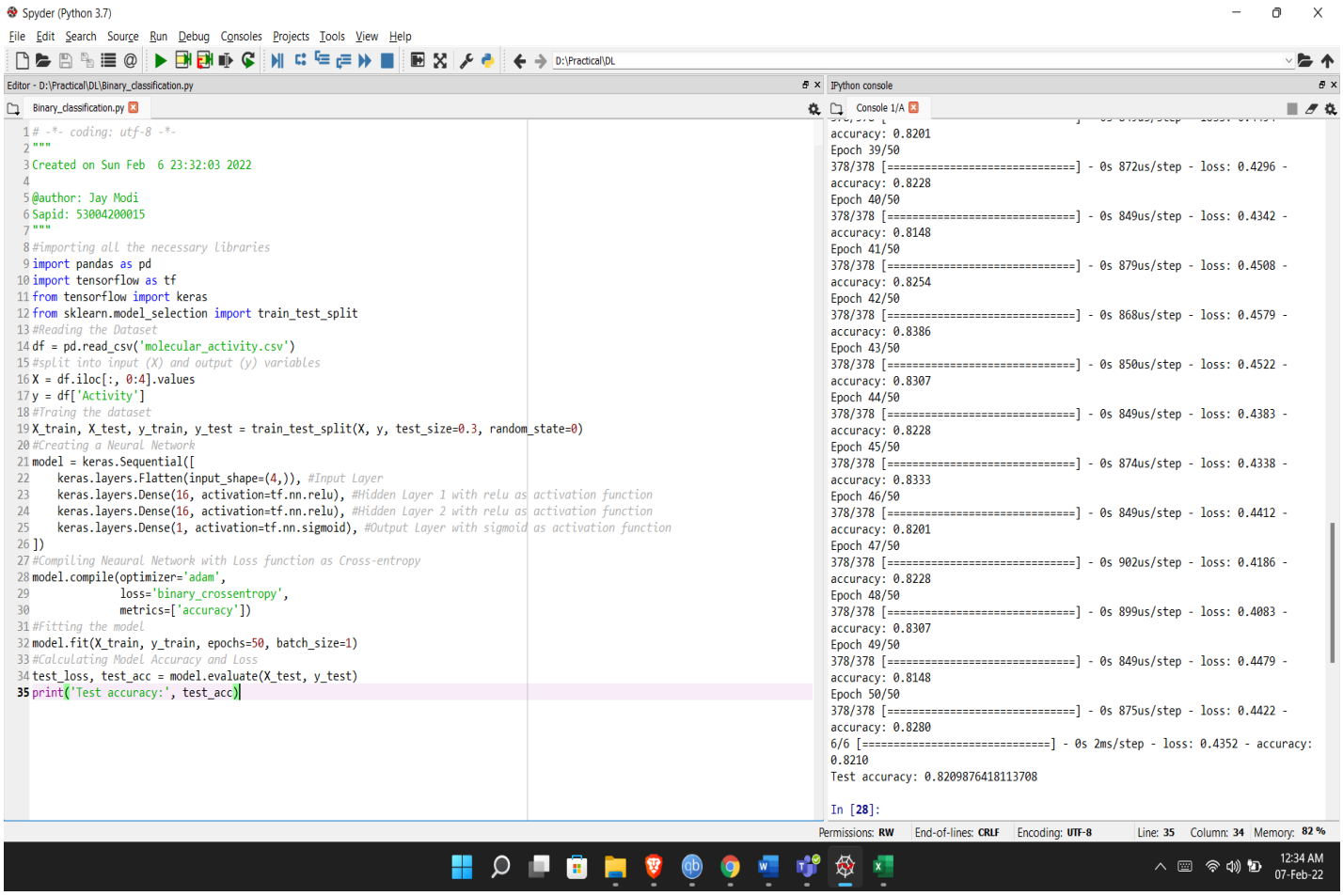
Teacher's Signature

Code

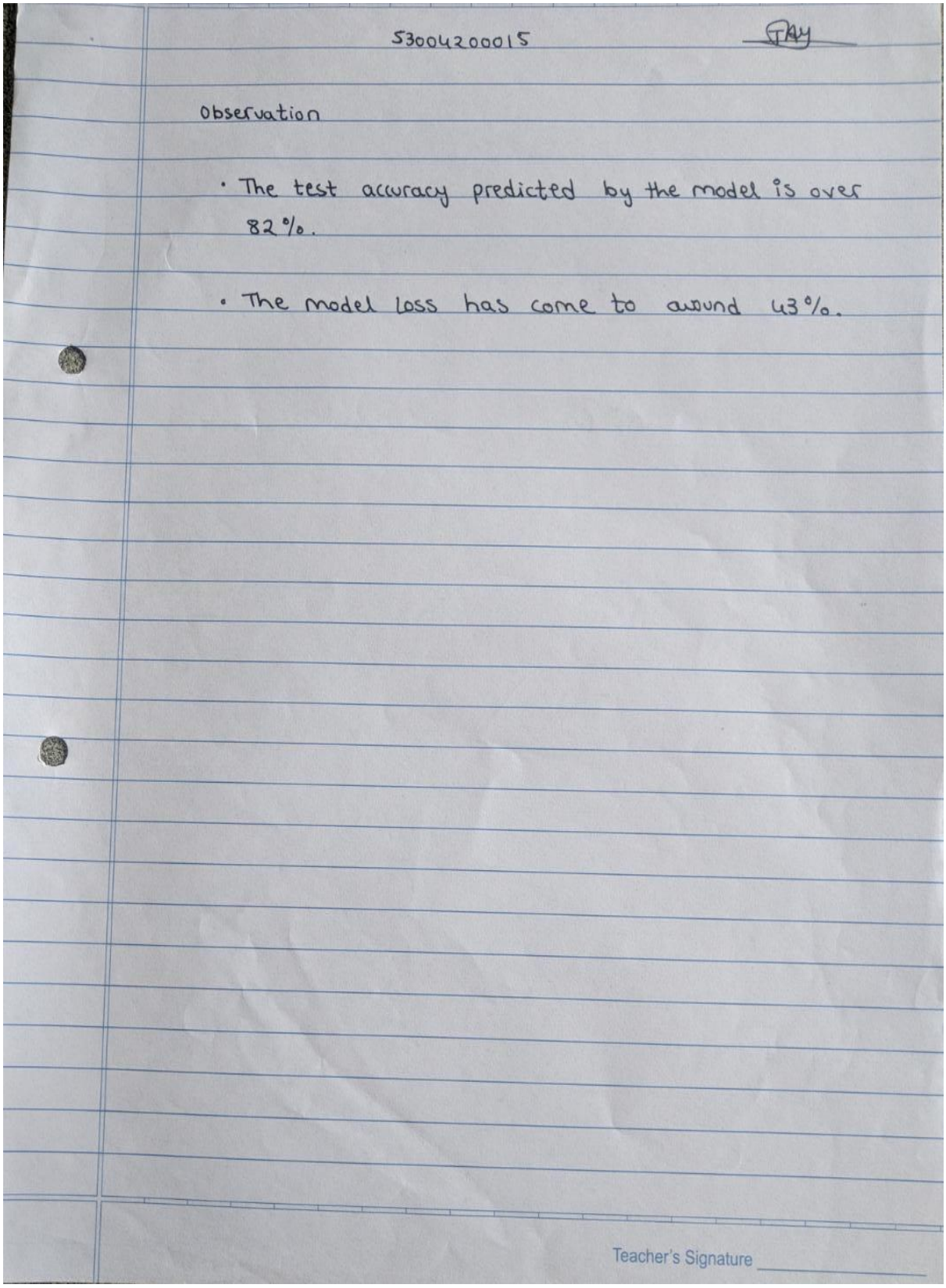
```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb  6 23:32:03 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
#Reading the Dataset
df = pd.read_csv('molecular_activity.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:4].values
y = df['Activity']
#Traing the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
#Creating a Neural Network
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(4,)), #Input Layer
    keras.layers.Dense(16, activation=tf.nn.relu), #Hidden Layer 1 with relu as activation function
        keras.layers.Dense(16, activation=tf.nn.relu), #Hidden Layer 2 with relu as activation function
```

```
    keras.layers.Dense(1, activation=tf.nn.sigmoid), #Output Layer with sigmoid as activation function
])
#Compiling Neaural Network with Loss function as Cross-entropy
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
#Fitting the model
model.fit(X_train, y_train, epochs=50, batch_size=1)
#Calculating Model Accuracy and Loss
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - D:\Practical\DL\Binary_classification.py
Binary_classification.py
IPython console
Console 1/A
-*- coding: utf-8 -*-
"""
Created on Sun Feb 6 23:32:03 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
#Reading the Dataset
df = pd.read_csv('molecular_activity.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:4].values
y = df['Activity']
#Traing the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
#Creating a Neural Network
model = keras.Sequential([
keras.layers.Flatten(input_shape=(4,)), #Input Layer
keras.layers.Dense(16, activation=tf.nn.relu), #Hidden Layer 1 with relu as activation function
keras.layers.Dense(16, activation=tf.nn.relu), #Hidden Layer 2 with relu as activation function
keras.layers.Dense(1, activation=tf.nn.sigmoid), #Output Layer with sigmoid as activation function
])
#Compiling Neaural Network with Loss function as Cross-entropy
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
#Fitting the model
model.fit(X_train, y_train, epochs=50, batch_size=1)
#Calculating Model Accuracy and Loss
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
accuracy: 0.8201
Epoch 39/50
378/378 [==========] - 0s 872us/step - loss: 0.4296 - accuracy: 0.8228
Epoch 40/50
378/378 [==========] - 0s 849us/step - loss: 0.4342 - accuracy: 0.8148
Epoch 41/50
378/378 [==========] - 0s 879us/step - loss: 0.4508 - accuracy: 0.8254
Epoch 42/50
378/378 [==========] - 0s 868us/step - loss: 0.4579 - accuracy: 0.8386
Epoch 43/50
378/378 [==========] - 0s 850us/step - loss: 0.4522 - accuracy: 0.8307
Epoch 44/50
378/378 [==========] - 0s 849us/step - loss: 0.4383 - accuracy: 0.8228
Epoch 45/50
378/378 [==========] - 0s 874us/step - loss: 0.4338 - accuracy: 0.8333
Epoch 46/50
378/378 [==========] - 0s 849us/step - loss: 0.4412 - accuracy: 0.8201
Epoch 47/50
378/378 [==========] - 0s 902us/step - loss: 0.4186 - accuracy: 0.8228
Epoch 48/50
378/378 [==========] - 0s 899us/step - loss: 0.4083 - accuracy: 0.8307
Epoch 49/50
378/378 [==========] - 0s 849us/step - loss: 0.4479 - accuracy: 0.8148
Epoch 50/50
378/378 [==========] - 0s 875us/step - loss: 0.4422 - accuracy: 0.8280
6/6 [==========] - 0s 2ms/step - loss: 0.4352 - accuracy: 0.8210
Test accuracy: 0.8209876418113708
In [28]:
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 35
Column: 34
Memory: 82 %
12:34 AM
07-Feb-22

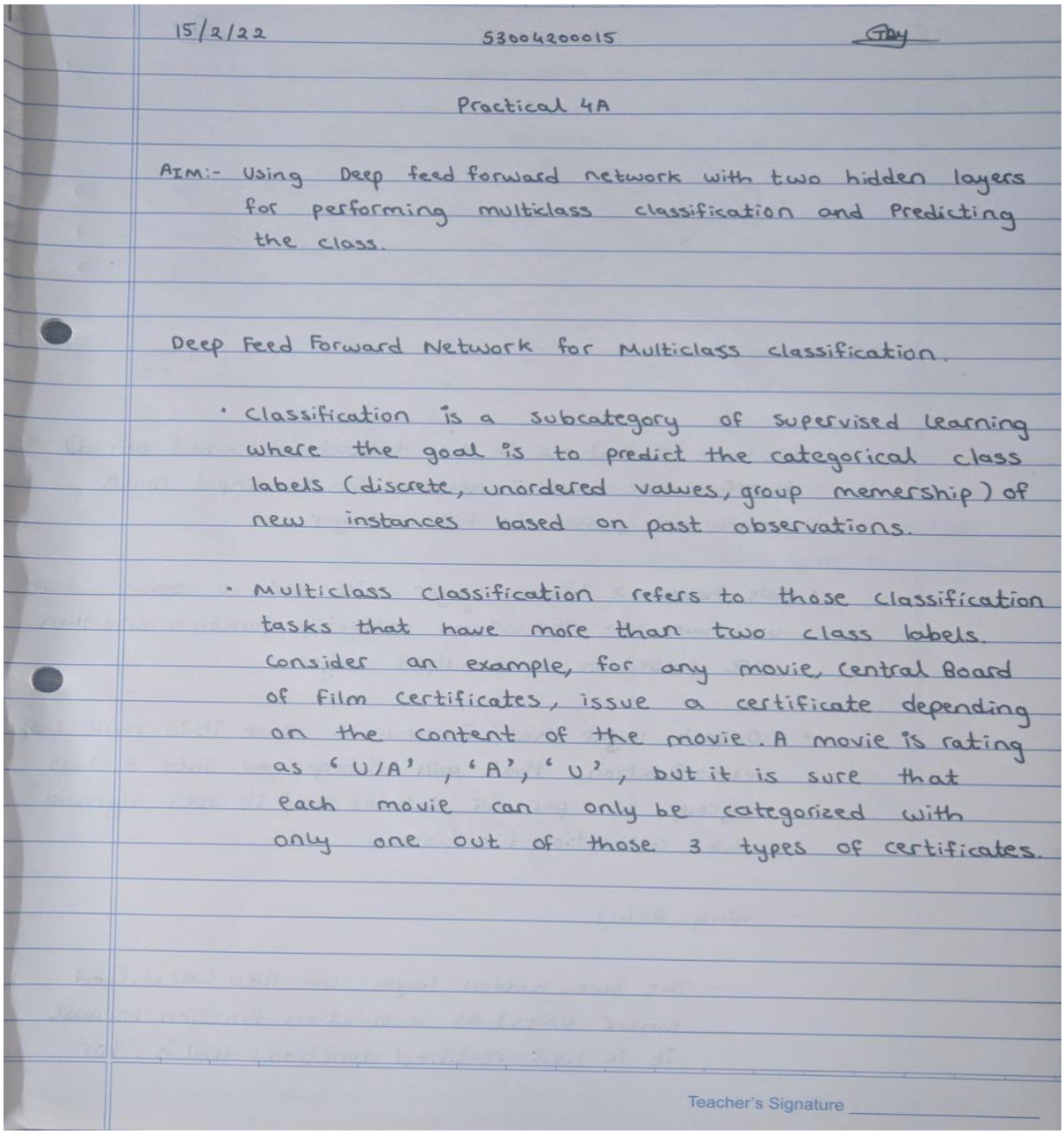
Observation



Practical 4A

Using deep feed forward network with two hidden layers for performing multiclass classification and predicting the class.

Deep Feed Forward Network for Multiclass classification



15/2/22 53004200015 Today

Practical 4A

AIM:- Using Deep feed forward network with two hidden layers for performing multiclass classification and Predicting the class.

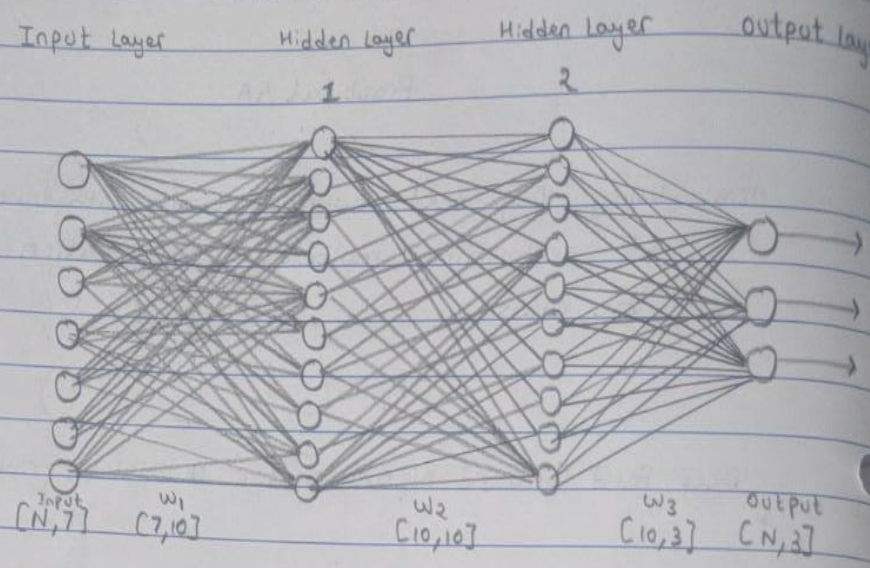
Deep Feed Forward Network for Multiclass classification.

- Classification is a subcategory of supervised learning where the goal is to predict the categorical class labels (discrete, unordered values, group memership) of new instances based on past observations.

- Multiclass classification refers to those classification tasks that have more than two class labels. Consider an example, for any movie, Central Board of Film Certificates, issue a certificate depending on the content of the movie. A movie is rating as 'U/A', 'A', 'U', but it is sure that each movie can only be categorized with only one out of those 3 types of certificates.

Teacher's Signature _____

53004200015



- We have 7 features in the dataset (described ahead) therefore we have 7 neurons in the Input Layer. which are passed to hidden Layer.

- We have 2 hidden Layer with each 10 neurons where we have use "relu" as activation function and then are passed to the output layer.

- Output layer has 3 neurons since it is multiclass classification that will categorized into 3 class group as per our dataset and it uses "sigmoid" as activation function.

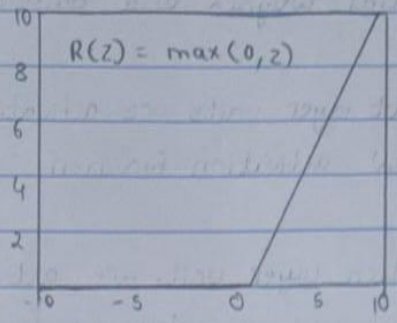
- Why Relu?

  - The two hidden layers use RdU (Rectified Linear units) as activation function because it is half-rectified function; that is, for

Teacher's Signature

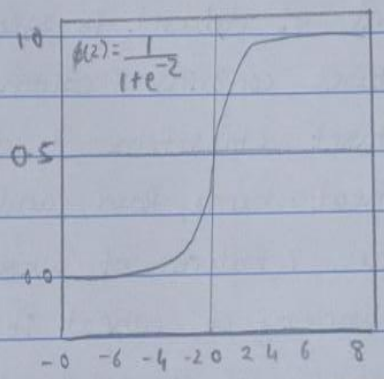
53004200015

all the input less than 0 (eg:- -120, -6.7, -0.03, 0) the value is 0 while for anything positive (10, 15, 34) the value is retained.



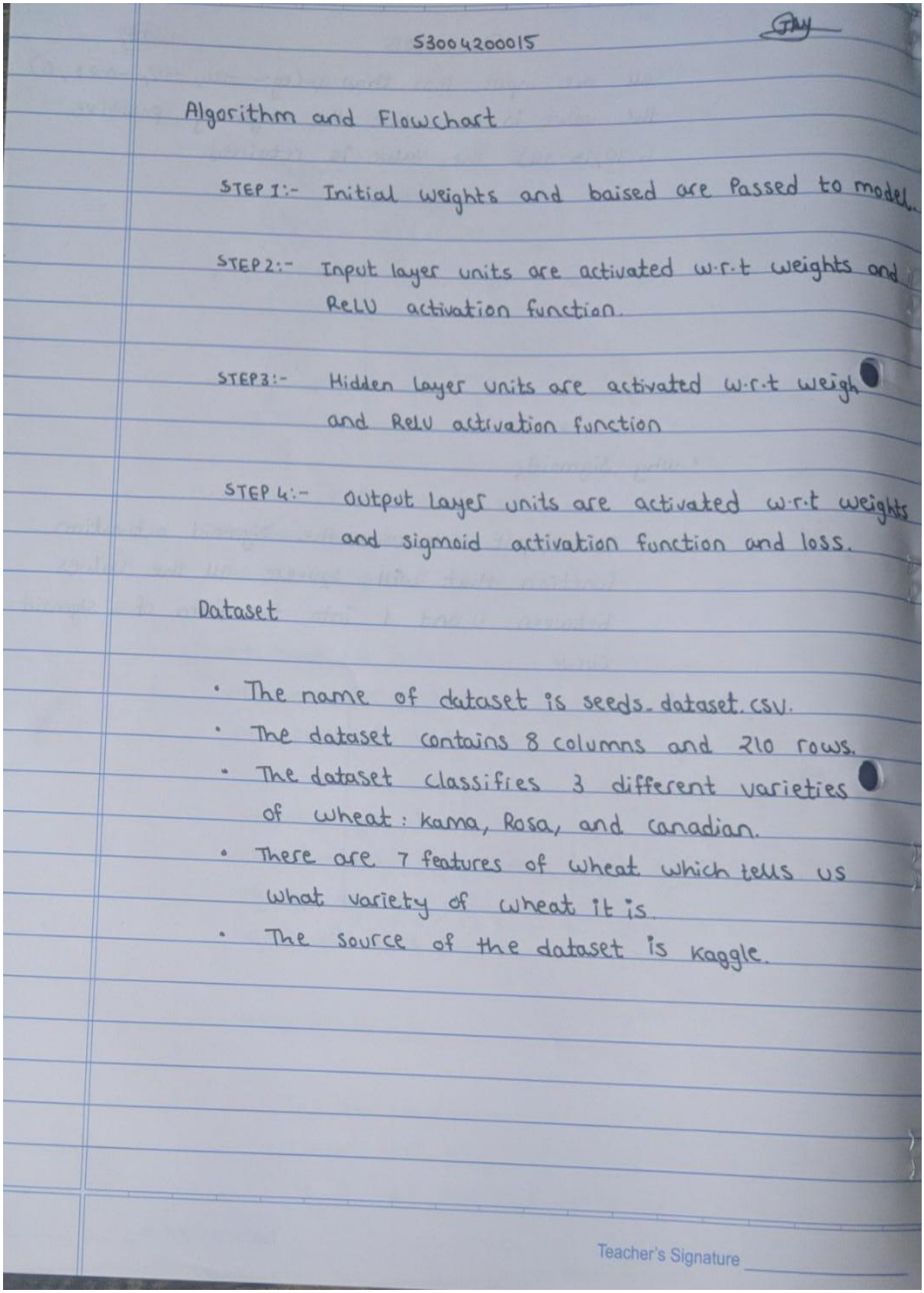
- Why Sigmoid?

- The output node uses the sigmoid activation function that will squeeze all the values between 0 and 1 into the form of a sigmoid curve.



Teacher's Signature

Algorithm and Dataset



53004200015

Algorithm and Flowchart

STEP 1:- Initial weights and baised are Passed to model.

STEP 2:- Input layer units are activated w.r.t weights and ReLU activation function.

STEP 3:- Hidden layer units are activated w.r.t weigh and ReLU activation function

STEP 4:- Output layer units are activated w.r.t weights and sigmoid activation function and loss.

Dataset

- The name of dataset is seeds_dataset.csv.
- The dataset contains 8 columns and 210 rows.
- The dataset classifies 3 different varieties of wheat: Kama, Rosa, and Canadian.
- There are 7 features of wheat which tells us what variety of wheat it is.
- The source of the dataset is Kaggle.

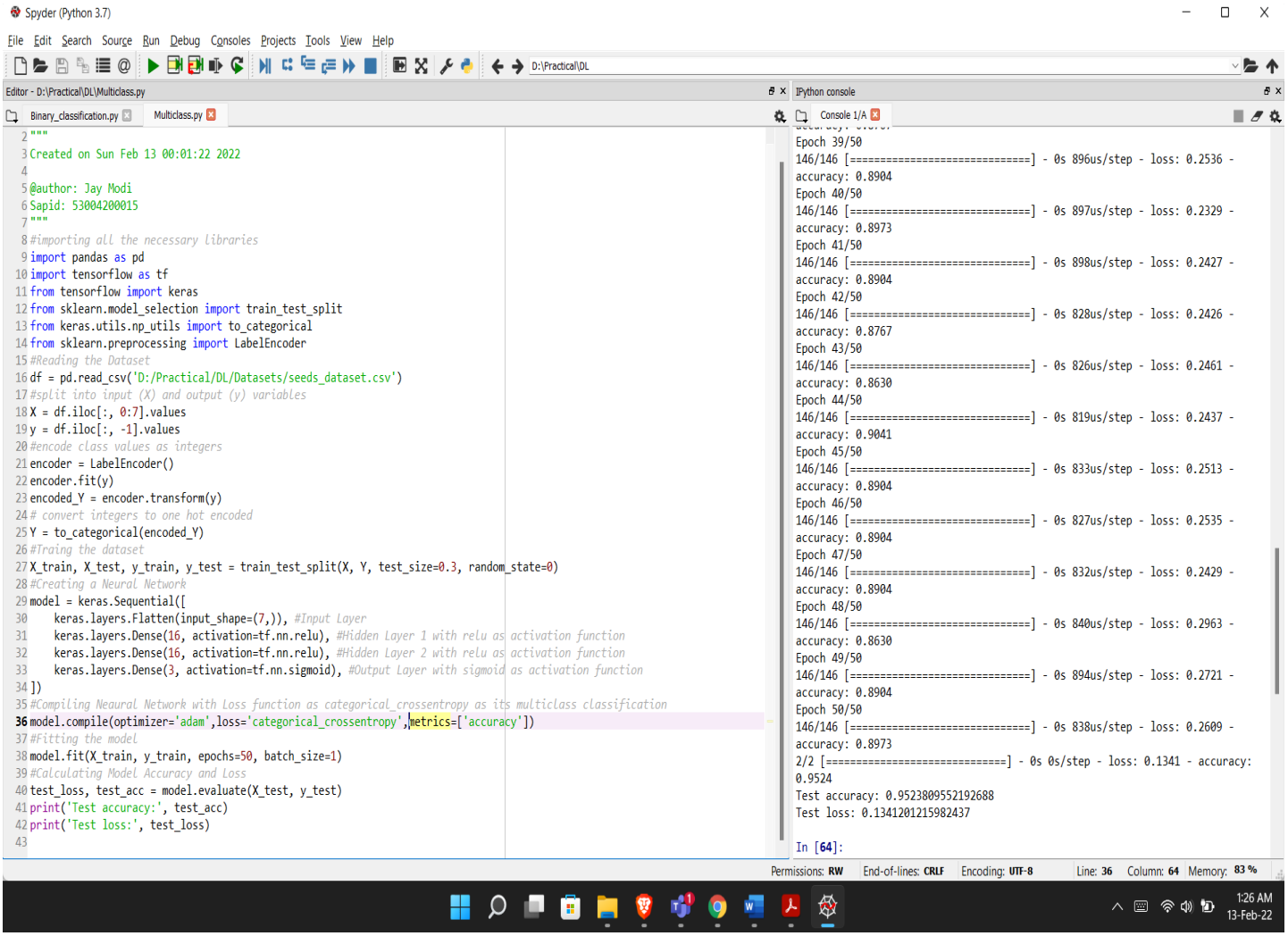
Teacher's Signature

Code

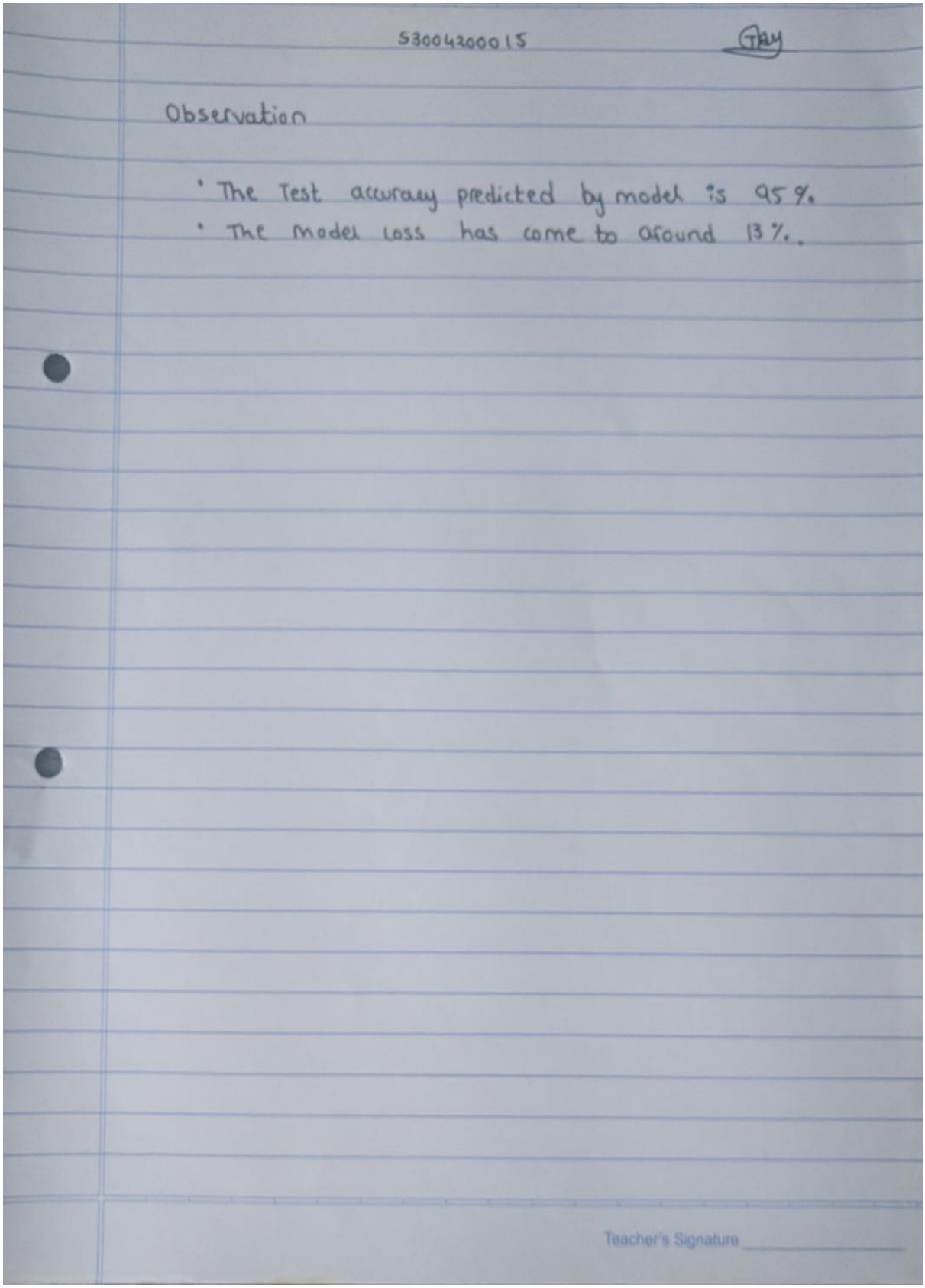
```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 13 00:01:22 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/seeds_dataset.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:7].values
y = df.iloc[:, -1].values
#encode class values as integers
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to one hot encoded
Y = to_categorical(encoded_Y)
#Traing the dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

#Creating a Neural Network

model = keras.Sequential([

    keras.layers.Flatten(input_shape=(7,)), #Input Layer

    keras.layers.Dense(10, activation=tf.nn.relu), #Hidden Layer 1 with relu as activation function

        keras.layers.Dense(10, activation=tf.nn.relu), #Hidden Layer 2 with relu as activation function

    keras.layers.Dense(3, activation=tf.nn.sigmoid), #Output Layer with sigmoid as activation function

])

#Compiling Neural Network with Loss function as categorical_crossentropy as its multiclass classification

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

#Fitting the model

model.fit(X_train, y_train, epochs=50, batch_size=1)

#Calculating Model Accuracy and Loss

test_loss, test_acc = model.evaluate(X_test, y_test)

print('Test accuracy:', test_acc)

print('Test loss:', test_loss)
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - D:\Practical\DL\Multiclass.py
Binary_classification.py
Multiclass.py
"""
Created on Sun Feb 13 00:01:22 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/seeds_dataset.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:7].values
y = df.iloc[:, -1].values
#encode class values as integers
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to one hot encoded
Y = to_categorical(encoded_Y)
#Traing the dataset
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
#Creating a Neural Network
model = keras.Sequential([
keras.layers.Flatten(input_shape=(7,)), #Input Layer
keras.layers.Dense(16, activation=tf.nn.relu), #Hidden Layer 1 with relu as activation function
keras.layers.Dense(16, activation=tf.nn.relu), #Hidden Layer 2 with relu as activation function
keras.layers.Dense(3, activation=tf.nn.sigmoid), #Output Layer with sigmoid as activation function
])
#Compiling Neaural Network with Loss function as categorical_crossentropy as its multiclass classification
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
#Fitting the model
model.fit(X_train, y_train, epochs=50, batch_size=1)
#Calculating Model Accuracy and Loss
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)
IPython console
Console 1/A
Epoch 39/50
146/146 [==========] - 0s 896us/step - loss: 0.2536 - accuracy: 0.8904
Epoch 40/50
146/146 [==========] - 0s 897us/step - loss: 0.2329 - accuracy: 0.8973
Epoch 41/50
146/146 [==========] - 0s 898us/step - loss: 0.2427 - accuracy: 0.8904
Epoch 42/50
146/146 [==========] - 0s 828us/step - loss: 0.2426 - accuracy: 0.8767
Epoch 43/50
146/146 [==========] - 0s 826us/step - loss: 0.2461 - accuracy: 0.8630
Epoch 44/50
146/146 [==========] - 0s 819us/step - loss: 0.2437 - accuracy: 0.9041
Epoch 45/50
146/146 [==========] - 0s 833us/step - loss: 0.2513 - accuracy: 0.8904
Epoch 46/50
146/146 [==========] - 0s 827us/step - loss: 0.2535 - accuracy: 0.8904
Epoch 47/50
146/146 [==========] - 0s 832us/step - loss: 0.2429 - accuracy: 0.8904
Epoch 48/50
146/146 [==========] - 0s 840us/step - loss: 0.2963 - accuracy: 0.8630
Epoch 49/50
146/146 [==========] - 0s 894us/step - loss: 0.2721 - accuracy: 0.8904
Epoch 50/50
146/146 [==========] - 0s 838us/step - loss: 0.2609 - accuracy: 0.8973
2/2 [==========] - 0s 0s/step - loss: 0.1341 - accuracy: 0.9524
Test accuracy: 0.9523809552192688
Test loss: 0.1341201215982437
In [64]:
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 36
Column: 64
Memory: 83 %
1:26 AM
13-Feb-22

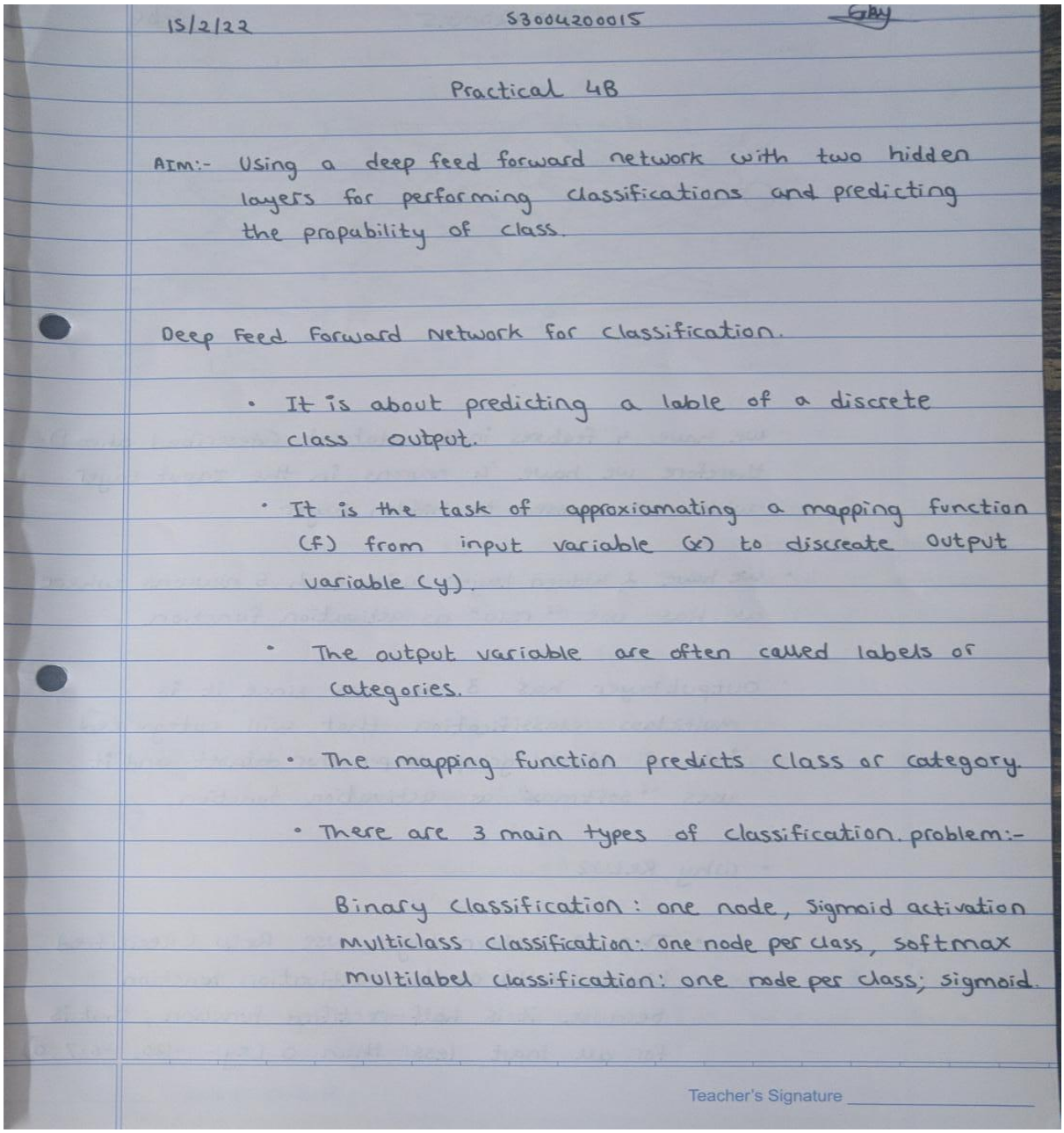
Observation



Practical 4B

Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class.

Deep feed forward network for Classification



15/2/22 53004200015

Practical 4B

AIM:- Using a deep feed forward network with two hidden layers for performing classifications and predicting the propability of class.

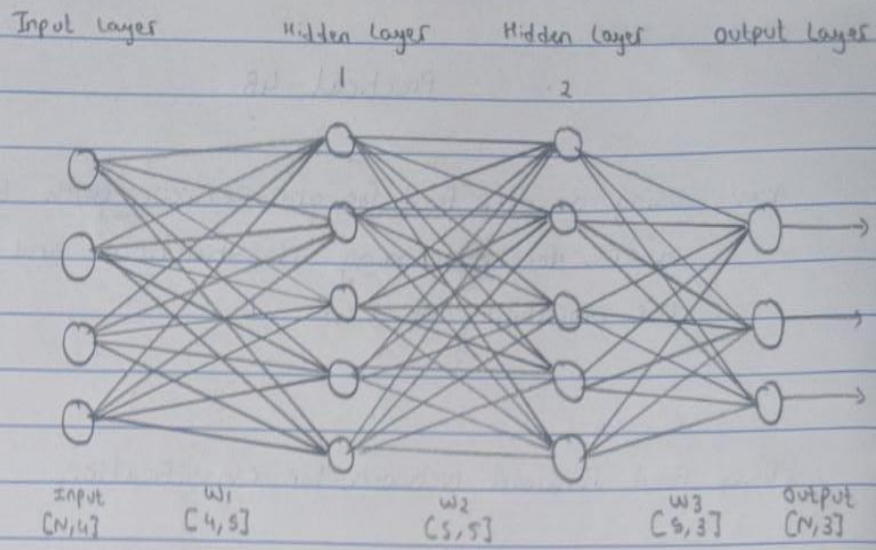
Deep Feed Forward Network for classification.

- It is about predicting a lable of a discrete class output.
- It is the task of approxiamating a mapping function (f) from input variable (x) to discreate output variable (y).
- The output variable are often called labels or categories.
- The mapping function predicts class or category.
- There are 3 main types of classification problem:-

Binary classification : one node, sigmoid activation
Multiclass classification: one node per class, softmax
Multilabel classification: one node per class; sigmoid.

Teacher's Signature _____

53004200015



- we have 4 features in the dataset (described ahead) therefore we have 4 neurons in the Input layer which are passed to hidden layer.

- we have 2 hidden layers with each 5 neurons where we has use "relu" as activation function.

- Output layer has 3 neurons since it is multiclass classification that will categorized into 3 class group as per our dataset and it uses "softmax" as activation function.

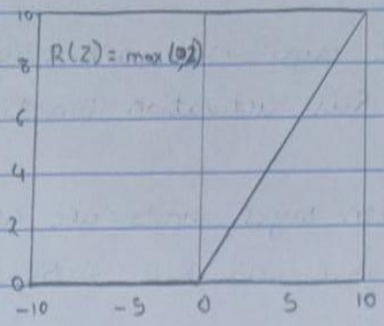
- Why ReLU?

  - The 2 hidden layer use ReLU (Recitified Linear Unit) as the activation function because it is half-rectified function; that is for all input less than 0 (eg:- -120, -6.7, 0)

Teacher's Signature

5300420015

the value is 0 while for anything positive (eg:- 10, 15, 34) the value is retained.



- Why Softmax?

  - The softmax function output a vector of values that sum to 1.0 that can be interpreted as probabilities of a class membership. It is related to argmax function that output a 0 for all options and 1 for the chosen option.

  - Softmax is a "softer" version of argmax that allow a probability-like output of a winner take-all function.

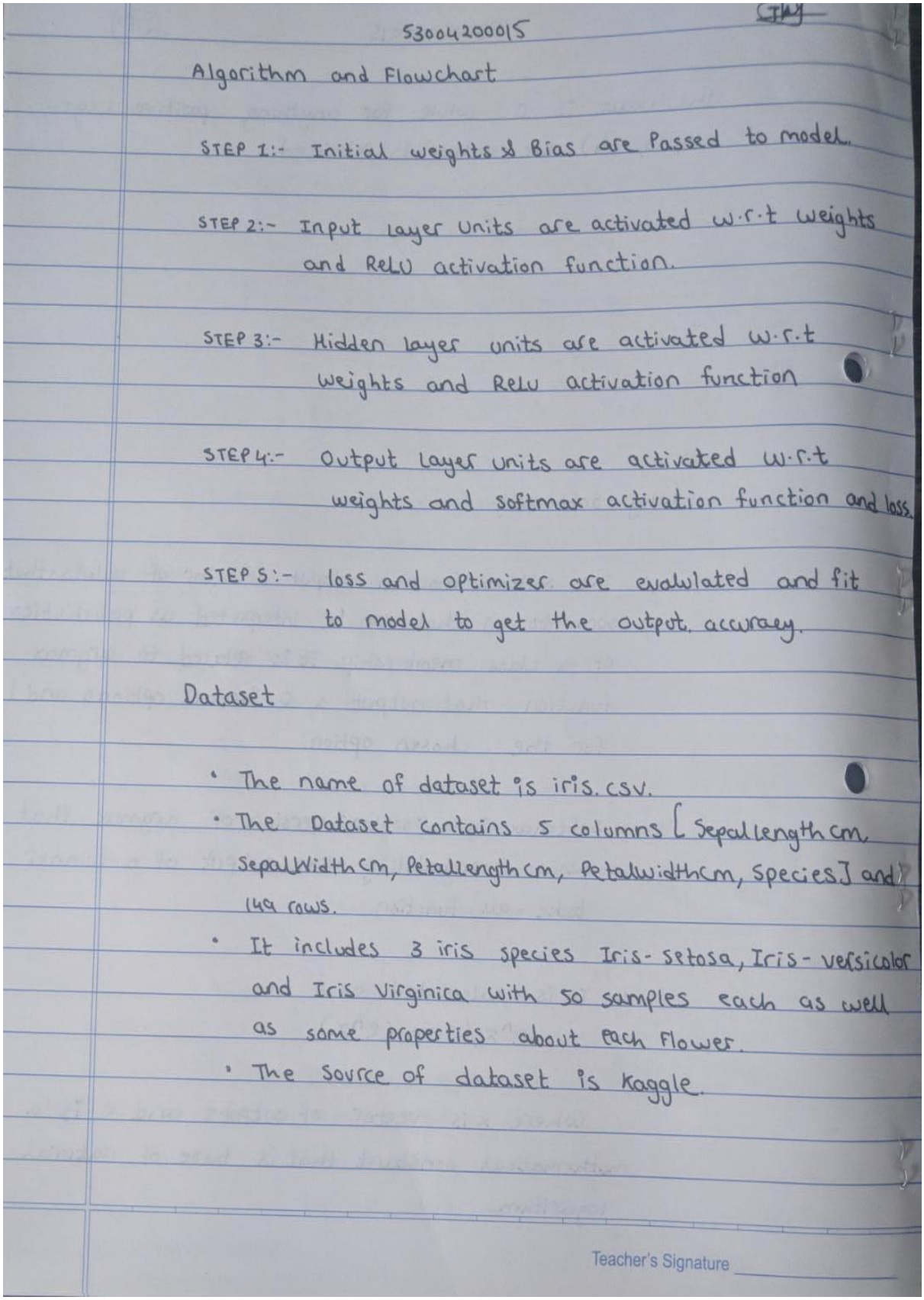
  - It is calculated as:-

    $e^x / \text{sum}(e^x)$

    where x is vector of outputs and e is a mathematical constant that is base of natural logarithm.

Teacher's Signature

Algorithm and Dataset



53004200015

Algorithm and Flowchart

STEP 1:- Initial weights & Bias are Passed to model.

STEP 2:- Input layer units are activated w.r.t weights and ReLU activation function.

STEP 3:- Hidden layer units are activated w.r.t weights and ReLU activation function

STEP 4:- Output layer units are activated w.r.t weights and softmax activation function and loss.

STEP 5:- loss and optimizer are evaluated and fit to model to get the output. accuracy.

Dataset

- The name of dataset is iris.csv.
- The Dataset contains 5 columns [SepalLength cm, SepalWidth cm, PetalLength cm, PetalWidth cm, Species] and 149 rows.
- It includes 3 iris species Iris-setosa, Iris-versicolor and Iris Virginica with 50 samples each as well as some properties about each flower.
- The source of dataset is Kaggle.

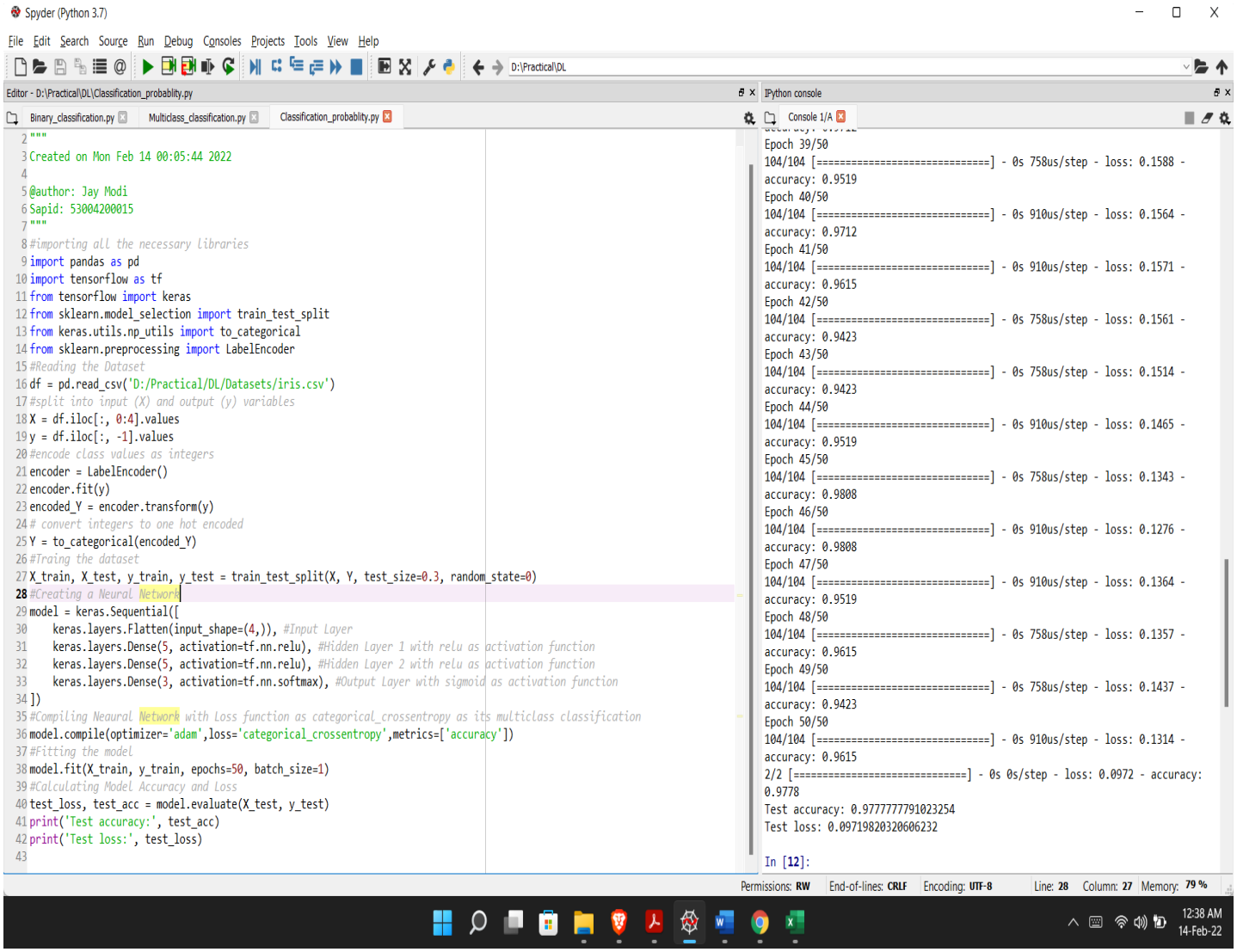
Teacher's Signature

Code

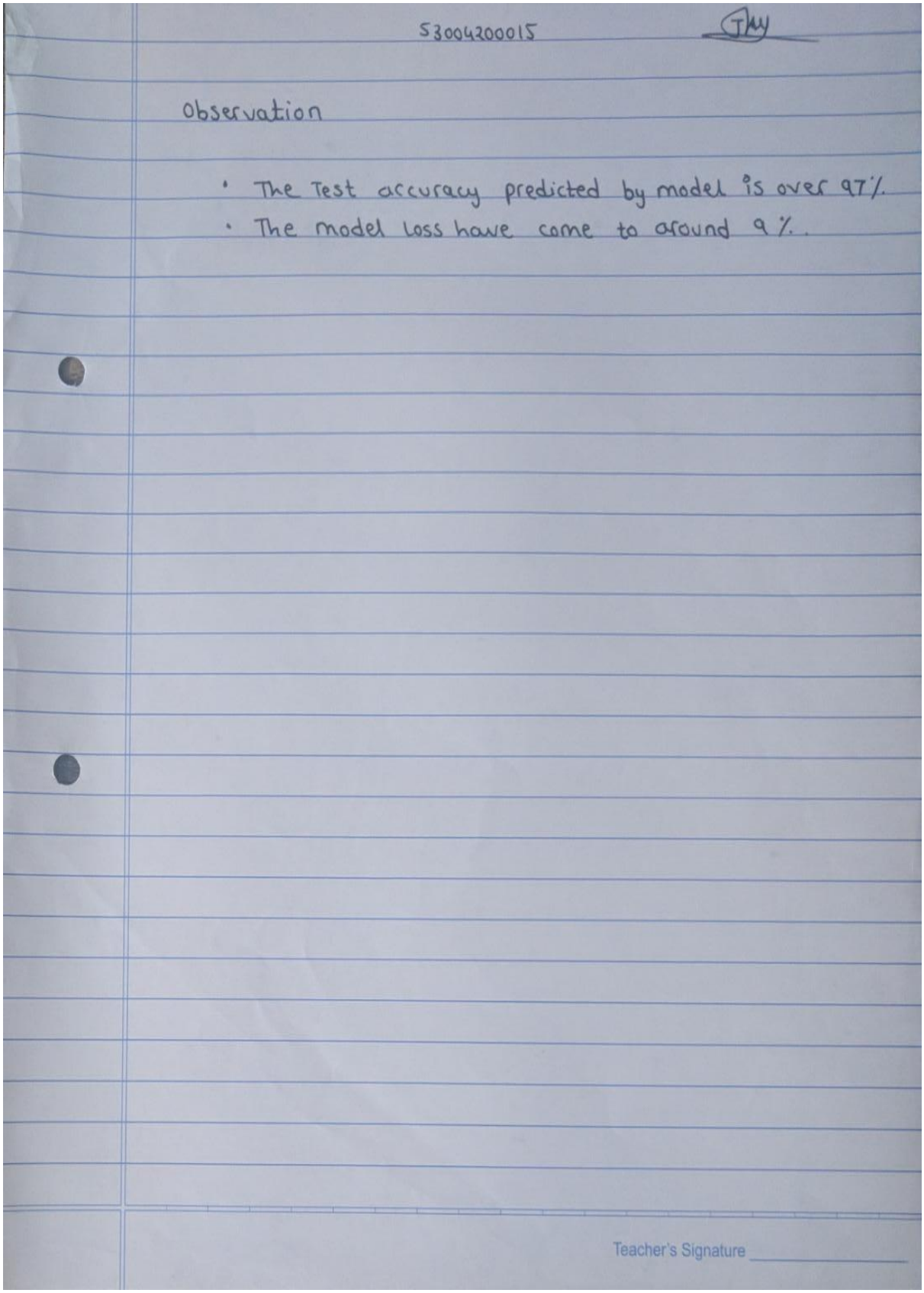
```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 14 00:05:44 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/iris.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:4].values
y = df.iloc[:, -1].values
#encode class values as integers
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to one hot encoded
Y = to_categorical(encoded_Y)
#Traing the dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)

#Creating a Neural Network

model = keras.Sequential([

    keras.layers.Flatten(input_shape=(4,)), #Input Layer

    keras.layers.Dense(5, activation=tf.nn.relu), #Hidden Layer 1 with relu as activation function

        keras.layers.Dense(5, activation=tf.nn.relu), #Hidden Layer 2 with relu as activation function

    keras.layers.Dense(3, activation=tf.nn.softmax), #Output Layer with sigmoid as activation function

])

#Compiling Neaural Network with Loss function as categorical_crossentropy as its multiclass classification

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

#Fitting the model

model.fit(X_train, y_train, epochs=50, batch_size=1)

#Calculating Model Accuracy and Loss

test_loss, test_acc = model.evaluate(X_test, y_test)

print('Test accuracy:', test_acc)

print('Test loss:', test_loss)
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - D:\Practical\DL\Classification_probablity.py
Binary_classification.py
Multiclass_classification.py
Classification_probablity.py
"""
Created on Mon Feb 14 00:05:44 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/iris.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:4].values
y = df.iloc[:, -1].values
#encode class values as integers
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to one hot encoded
Y = to_categorical(encoded_Y)
#Traing the dataset
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
#Creating a Neural Network
model = keras.Sequential([
keras.layers.Flatten(input_shape=(4,)), #Input Layer
keras.layers.Dense(5, activation=tf.nn.relu), #Hidden Layer 1 with relu as activation function
keras.layers.Dense(5, activation=tf.nn.relu), #Hidden Layer 2 with relu as activation function
keras.layers.Dense(3, activation=tf.nn.softmax), #Output Layer with sigmoid as activation function
])
#Compiling Neaural Network with Loss function as categorical_crossentropy as its multiclass classification
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
#Fitting the model
model.fit(X_train, y_train, epochs=50, batch_size=1)
#Calculating Model Accuracy and Loss
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)
IPython console
Console 1/A
Epoch 39/50
104/104 [==========] - 0s 758us/step - loss: 0.1588 - accuracy: 0.9519
Epoch 40/50
104/104 [==========] - 0s 910us/step - loss: 0.1564 - accuracy: 0.9712
Epoch 41/50
104/104 [==========] - 0s 910us/step - loss: 0.1571 - accuracy: 0.9615
Epoch 42/50
104/104 [==========] - 0s 758us/step - loss: 0.1561 - accuracy: 0.9423
Epoch 43/50
104/104 [==========] - 0s 758us/step - loss: 0.1514 - accuracy: 0.9423
Epoch 44/50
104/104 [==========] - 0s 910us/step - loss: 0.1465 - accuracy: 0.9519
Epoch 45/50
104/104 [==========] - 0s 758us/step - loss: 0.1343 - accuracy: 0.9808
Epoch 46/50
104/104 [==========] - 0s 910us/step - loss: 0.1276 - accuracy: 0.9808
Epoch 47/50
104/104 [==========] - 0s 910us/step - loss: 0.1364 - accuracy: 0.9519
Epoch 48/50
104/104 [==========] - 0s 758us/step - loss: 0.1357 - accuracy: 0.9615
Epoch 49/50
104/104 [==========] - 0s 758us/step - loss: 0.1437 - accuracy: 0.9423
Epoch 50/50
104/104 [==========] - 0s 910us/step - loss: 0.1314 - accuracy: 0.9615
2/2 [==========] - 0s 0s/step - loss: 0.0972 - accuracy: 0.9778
Test accuracy: 0.9777777791023254
Test loss: 0.09719820320606232
In [12]:
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 28
Column: 27
Memory: 79 %
12:38 AM
14-Feb-22

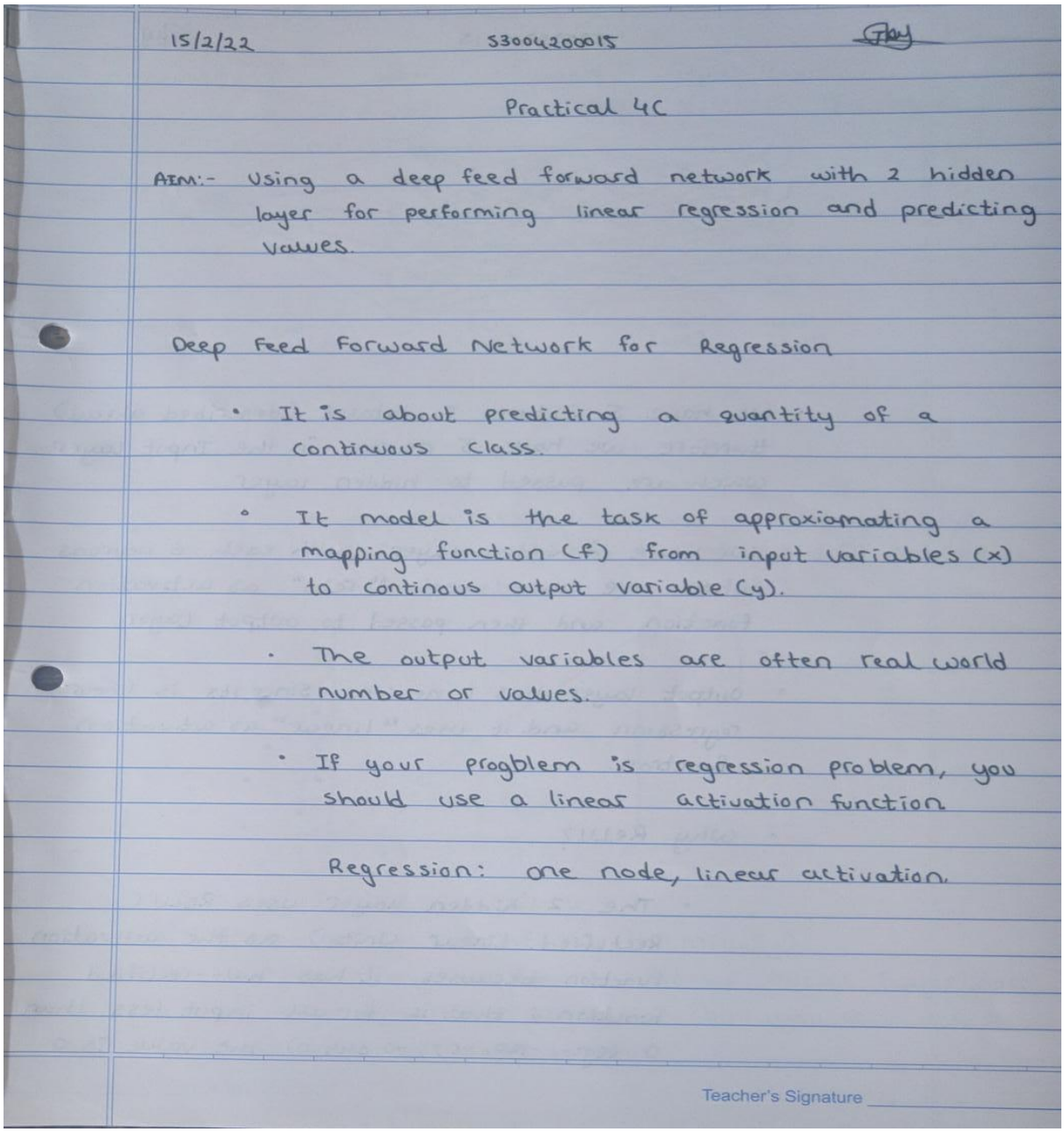
Observation



Practical 4C

Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.

Deep feed forward network for Regression



15/2/22 53004200015

Practical 4C

AIM:- Using a deep feed forward network with 2 hidden layer for performing linear regression and predicting values.

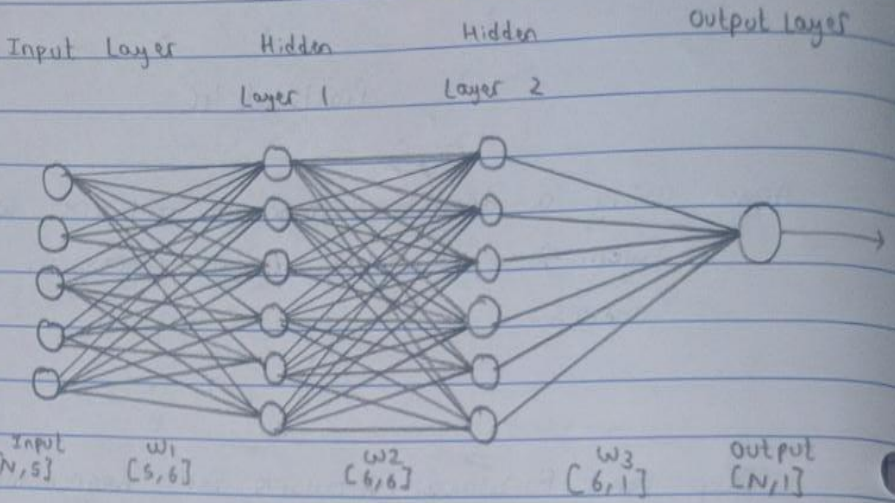
Deep Feed Forward Network for Regression

- It is about predicting a quantity of a continuous class.
- It model is the task of approximating a mapping function (f) from input variables (x) to continous output variable (y).
- The output variables are often real world number or values.
- If your progblem is regression problem, you should use a linear activation function.

Regression: one node, linear activation.

Teacher's Signature

53004200015



- We have 5 features in dataset (described ahead) therefore we have 5 neurons in the Input layer which are passed to hidden layer.

- We have 2 hidden layer with each 6 neurons where we have use "relu" as activation function and then passed to output layer.

- Output layer has 1 neurons since its is linear regression and it uses "linear" as activation function.

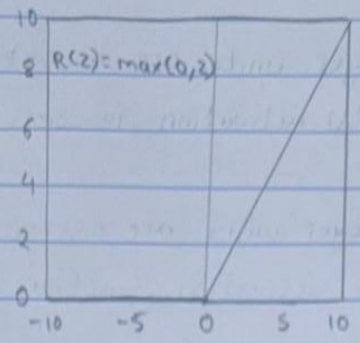
- Why ReLU?

  - The 2 hidden layer uses ReLU (Rectified Linear Units) as the activation function becauses it has half-rectified function ; that is for all input less than 0 (eg:- -120, -6.7, -0.344, 0) the value is 0

Teacher's Signature

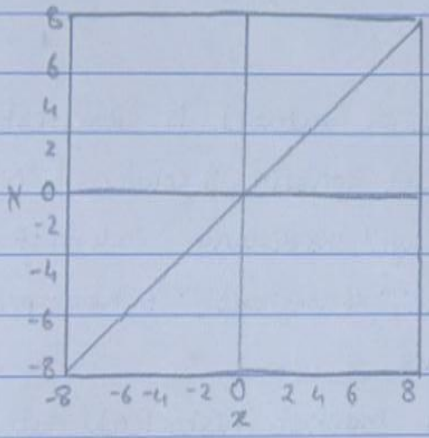
53004200015

while for any positive (eg:- 10, 15, 34) the value is retained.



- Why Linear ?

  - As you can see the function is Linear or line. Therefore, the output of function will not be confined between any range.



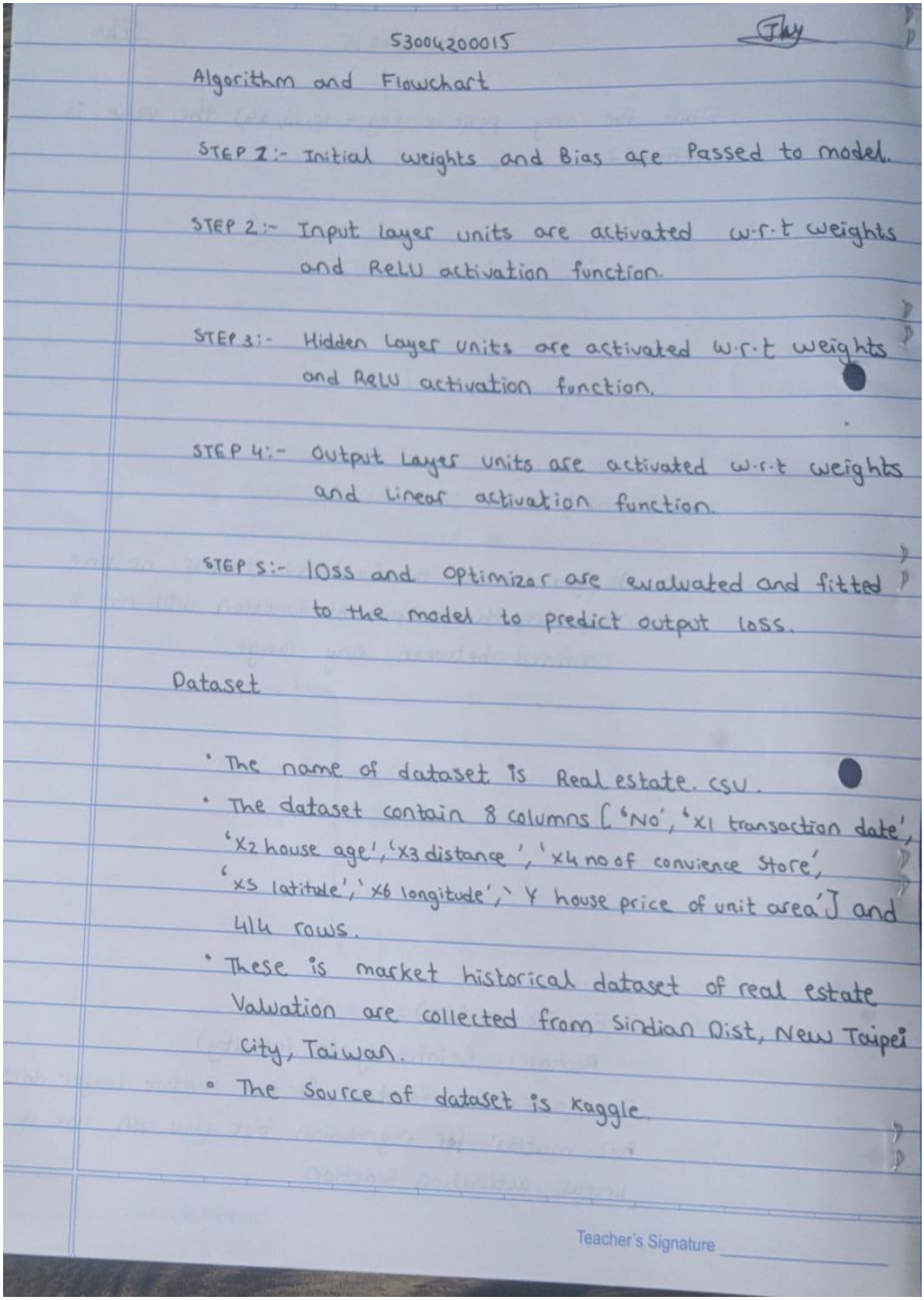
Equation:- $f(x) = x$

Range:- (-infinity to infinity)

- The activation function for the output layer does not matter for regression but you can use a linear activation function.

Teacher's Signature

Algorithm and Dataset



53004200015

Algorithm and Flowchart

STEP 1:- Initial weights and Bias are Passed to model.

STEP 2:- Input layer units are activated w.r.t weights and ReLU activation function.

STEP 3:- Hidden layer units are activated w.r.t weights and ReLU activation function.

STEP 4:- Output Layer units are activated w.r.t weights and linear activation function.

STEP 5:- loss and optimizer are evaluated and fitted to the model to predict output loss.

Dataset

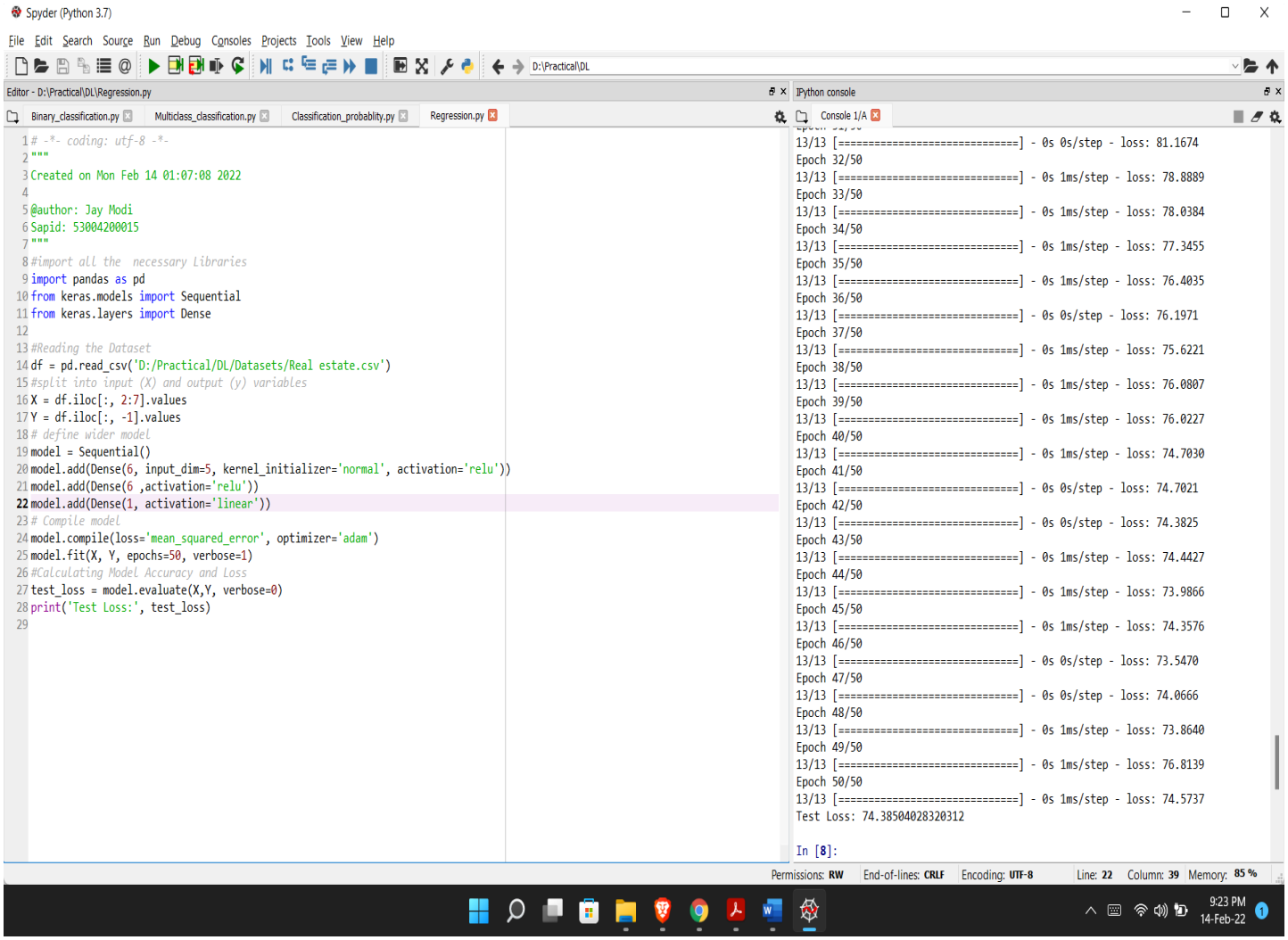
- The name of dataset is Real estate.csv.
- The dataset contain 8 columns ['No', 'X1 transaction date', 'X2 house age', 'X3 distance', 'X4 no of convience store', 'X5 latitude', 'X6 longitude', 'Y house price of unit area'] and 414 rows.
- These is market historical dataset of real estate Valuation are collected from Sindian Dist, New Taipei City, Taiwan.
- The source of dataset is Kaggle.

Teacher's Signature

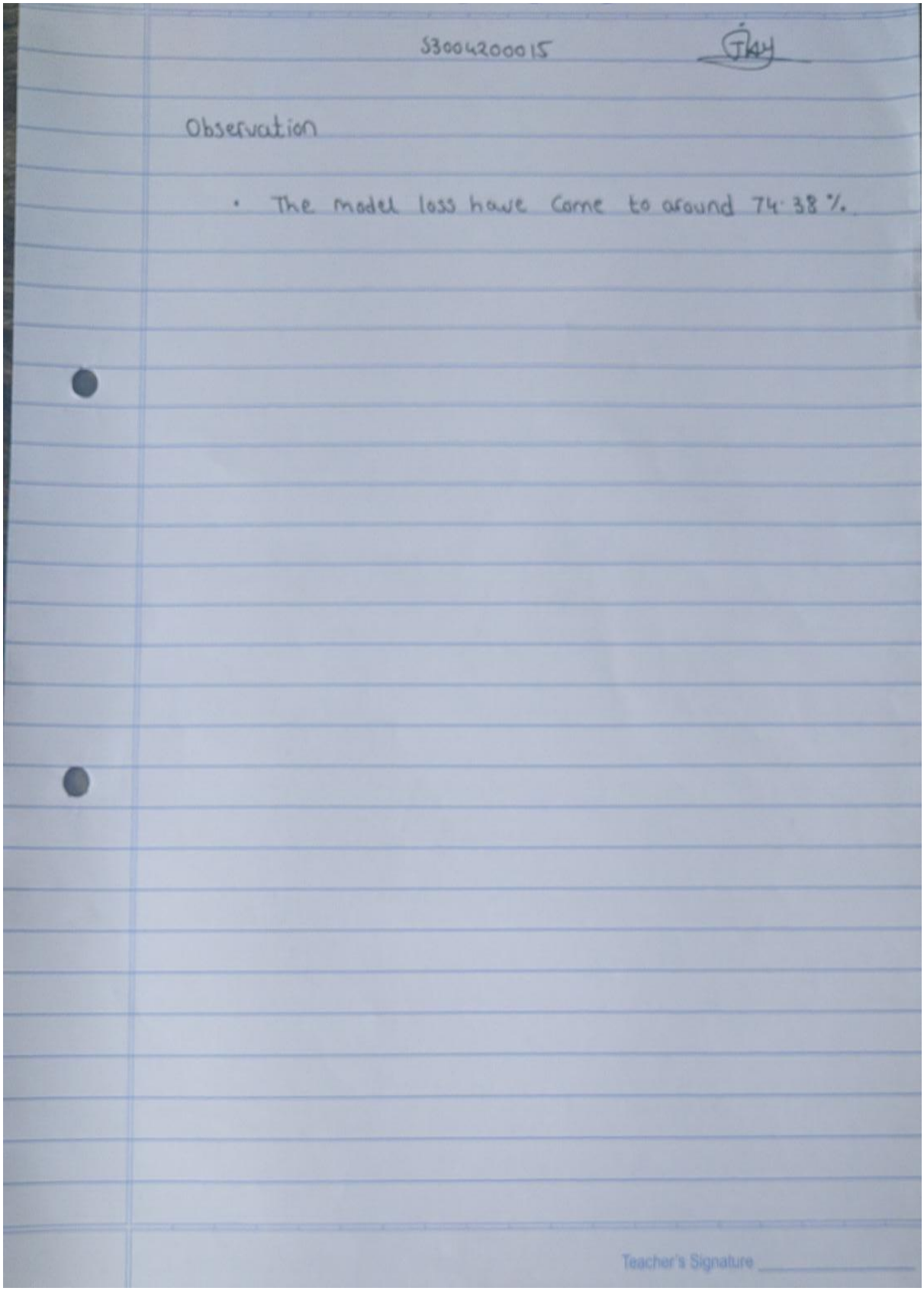
Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 14 01:07:08 2022
@author: Jay Modi
Sapid: 53004200015
"""
#import all the  necessary Libraries
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/Real estate.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 2:7].values
Y = df.iloc[:, -1].values
# define wider model
model = Sequential()
model.add(Dense(6, input_dim=5, kernel_initializer='normal', activation='relu'))
model.add(Dense(6 ,activation='relu'))
model.add(Dense(1, activation=‘linear’))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X, Y, epochs=50, verbose=1)
#Calculating Model Accuracy and Loss
test_loss = model.evaluate(X,Y, verbose=0)
print('Test Loss:', test_loss)
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Practical\DL
Editor - D:\Practical\DL\Regression.py
Binary_classification.py
Multiclass_classification.py
Classification_probablity.py
Regression.py
-*- coding: utf-8 -*-
"""
Created on Mon Feb 14 01:07:08 2022
@author: Jay Modi
Sapid: 53004200015
"""
#import all the necessary Libraries
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/Real estate.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 2:7].values
Y = df.iloc[:, -1].values
# define wider model
model = Sequential()
model.add(Dense(6, input_dim=5, kernel_initializer='normal', activation='relu'))
model.add(Dense(6 ,activation='relu'))
model.add(Dense(1, activation='linear'))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X, Y, epochs=50, verbose=1)
#Calculating Model Accuracy and Loss
test_loss = model.evaluate(X,Y, verbose=0)
print('Test Loss:', test_loss)
IPython console
Console 1/A
13/13 [==============================] - 0s 0s/step - loss: 81.1674
Epoch 32/50
13/13 [==============================] - 0s 1ms/step - loss: 78.8889
Epoch 33/50
13/13 [==============================] - 0s 1ms/step - loss: 78.0384
Epoch 34/50
13/13 [==============================] - 0s 1ms/step - loss: 77.3455
Epoch 35/50
13/13 [==============================] - 0s 1ms/step - loss: 76.4035
Epoch 36/50
13/13 [==============================] - 0s 0s/step - loss: 76.1971
Epoch 37/50
13/13 [==============================] - 0s 1ms/step - loss: 75.6221
Epoch 38/50
13/13 [==============================] - 0s 1ms/step - loss: 76.0807
Epoch 39/50
13/13 [==============================] - 0s 1ms/step - loss: 76.0227
Epoch 40/50
13/13 [==============================] - 0s 1ms/step - loss: 74.7030
Epoch 41/50
13/13 [==============================] - 0s 0s/step - loss: 74.7021
Epoch 42/50
13/13 [==============================] - 0s 0s/step - loss: 74.3825
Epoch 43/50
13/13 [==============================] - 0s 1ms/step - loss: 74.4427
Epoch 44/50
13/13 [==============================] - 0s 1ms/step - loss: 73.9866
Epoch 45/50
13/13 [==============================] - 0s 1ms/step - loss: 74.3576
Epoch 46/50
13/13 [==============================] - 0s 0s/step - loss: 73.5470
Epoch 47/50
13/13 [==============================] - 0s 0s/step - loss: 74.0666
Epoch 48/50
13/13 [==============================] - 0s 1ms/step - loss: 73.8640
Epoch 49/50
13/13 [==============================] - 0s 1ms/step - loss: 76.8139
Epoch 50/50
13/13 [==============================] - 0s 1ms/step - loss: 74.5737
Test Loss: 74.38504028320312
In [8]:
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 22
Column: 39
Memory: 85 %
9:23 PM
14-Feb-22

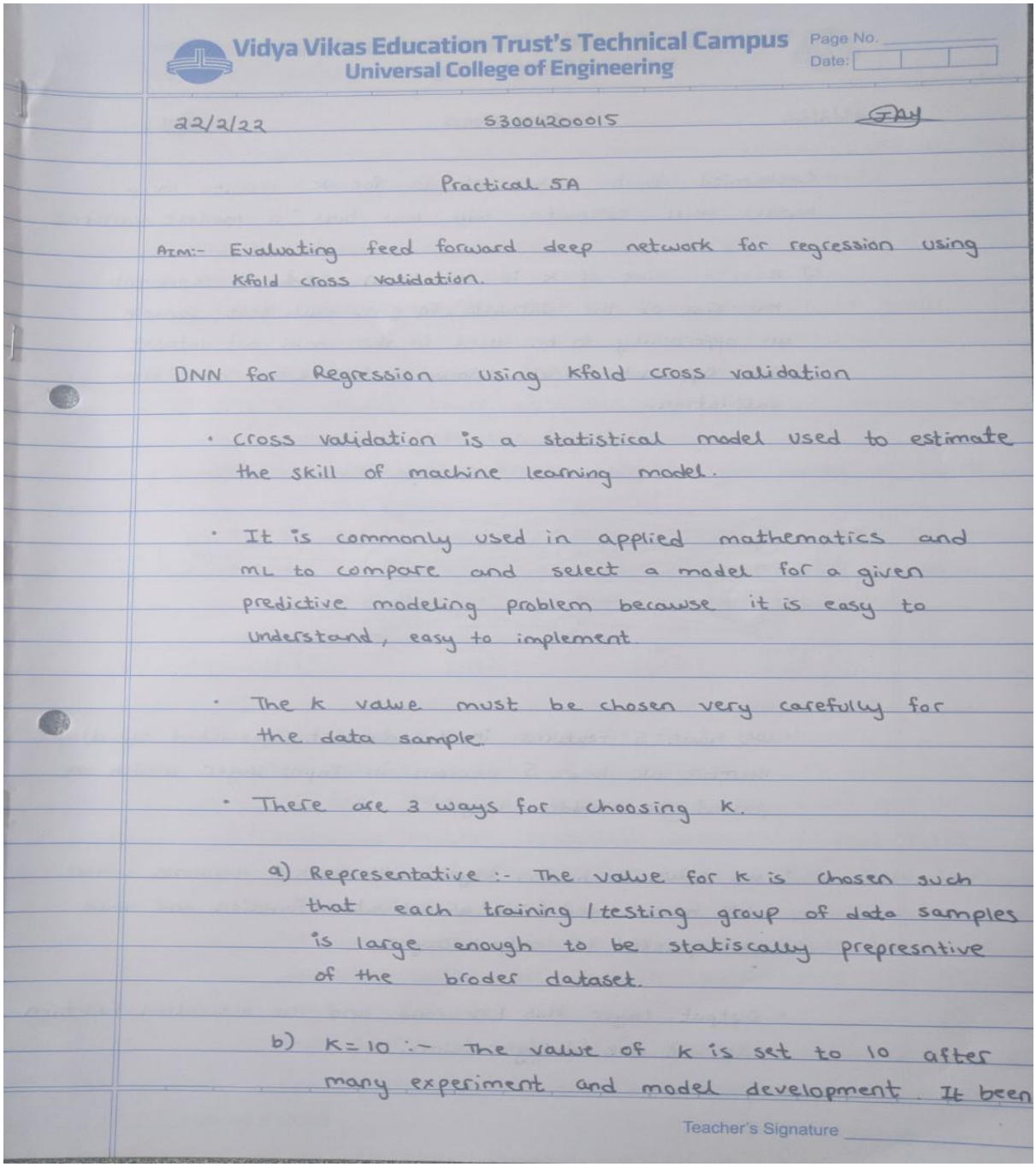
Observation



Practical 5A

Evaluating feed forward deep network for regression using KFold cross validation.

DNN for Regression using KFold cross validation



Vidya Vikas Education Trust's Technical Campus
Universal College of Engineering

Page No.
Date:

22/2/22 53004200015 FAY

Practical 5A

AIM:- Evaluating feed forward deep network for regression using Kfold cross validation.

DNN for Regression using Kfold cross validation

- Cross validation is a statistical model used to estimate the skill of machine learning model.
- It is commonly used in applied mathematics and ML to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement.
- The k value must be chosen very carefully for the data sample.
- There are 3 ways for choosing K.

a) Representative :- The value for K is chosen such that each training / testing group of data samples is large enough to be statiscally prepresntive of the broder dataset.

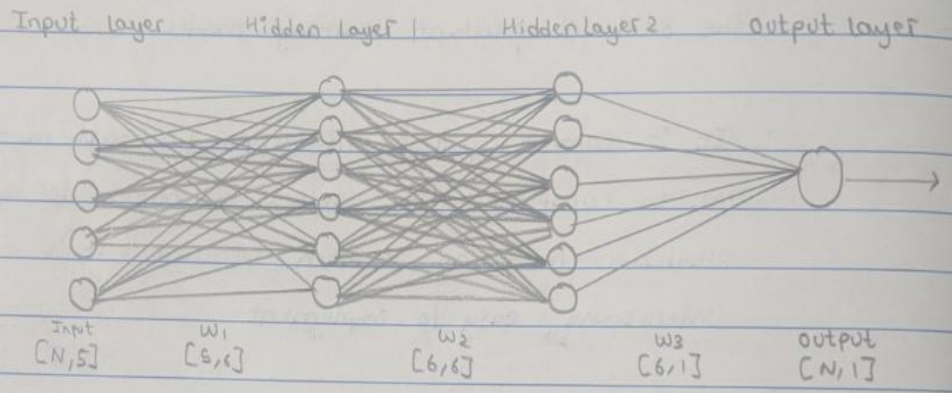
b) K=10 :- The value of K is set to 10 after many experiment and model development. It been

Teacher's Signature

22/2/22 S3004200015

exclamied that value of 10 for K results in a model skill estimate will low bias a modest variance.

c) K=n :- size of K is set to n, and it represent the size of the datasets. To give each test sample an opportunity to be used in the hold out dataset. This approach is also known as 'leave one out' cross validation.



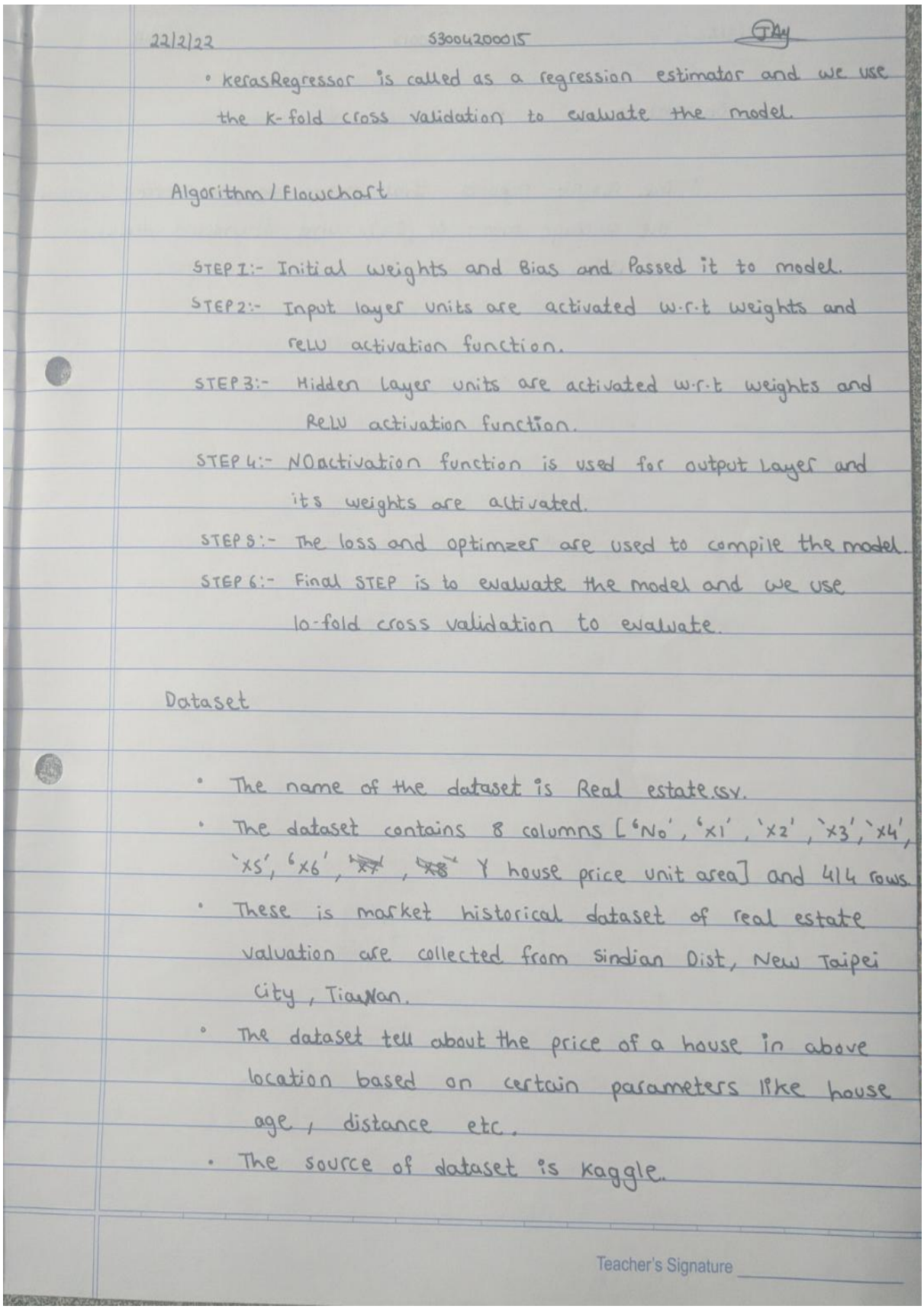
- We have 5 features in the dataset (described ahead) therefore we have 5 neurons in Input layer which are passed to hidden layer.

- We have 2 hidden layer with each 6 neuronos where we have used 'relu' as activation function and then are passed to output layer.

- Output layer has 1 neurons and no activation function as we are doing ression.

Teacher's Signature

Algorithm and Dataset



22/2/22 53004200015

- KerasRegressor is called as a regression estimator and we use the K-fold cross validation to evaluate the model.

Algorithm / Flowchart

STEP 1:- Initial weights and Bias and Passed it to model.

STEP 2:- Input layer units are activated w.r.t weights and ReLU activation function.

STEP 3:- Hidden layer units are activated w.r.t weights and ReLU activation function.

STEP 4:- NO activation function is used for output layer and its weights are activated.

STEP 5:- The loss and optimzer are used to compile the model.

STEP 6:- Final STEP is to evaluate the model and we use 10-fold cross validation to evaluate.

Dataset

- The name of the dataset is Real estate.csv.
- The dataset contains 8 columns ['No', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', ~~'x7'~~, ~~'x8'~~ Y house price unit area] and 414 rows.
- These is market historical dataset of real estate valuation are collected from Sindian Dist, New Taipei City, Taiwan.
- The dataset tell about the price of a house in above location based on certain parameters like house age, distance etc.
- The source of dataset is Kaggle.

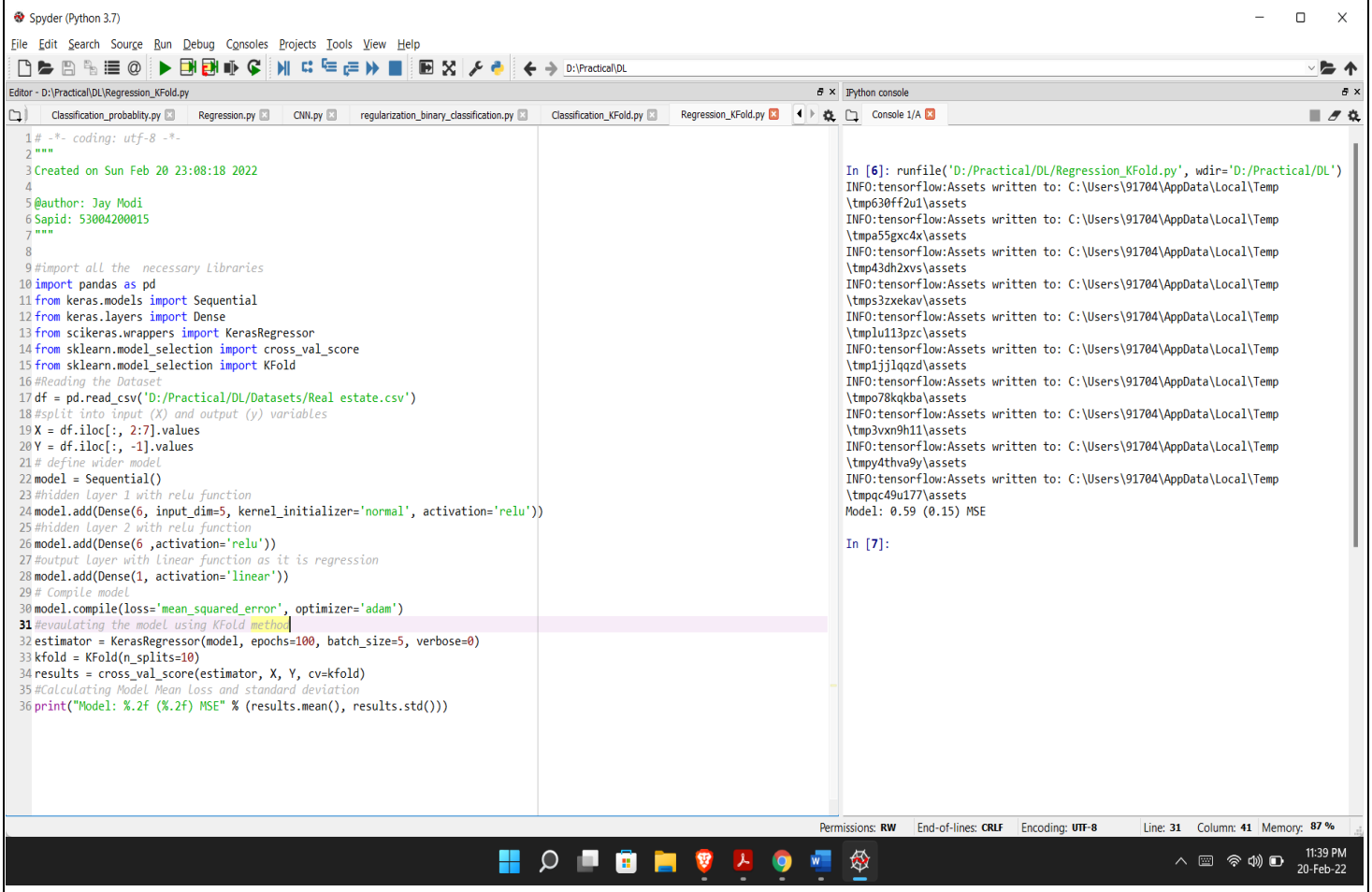
Teacher's Signature _____

Code

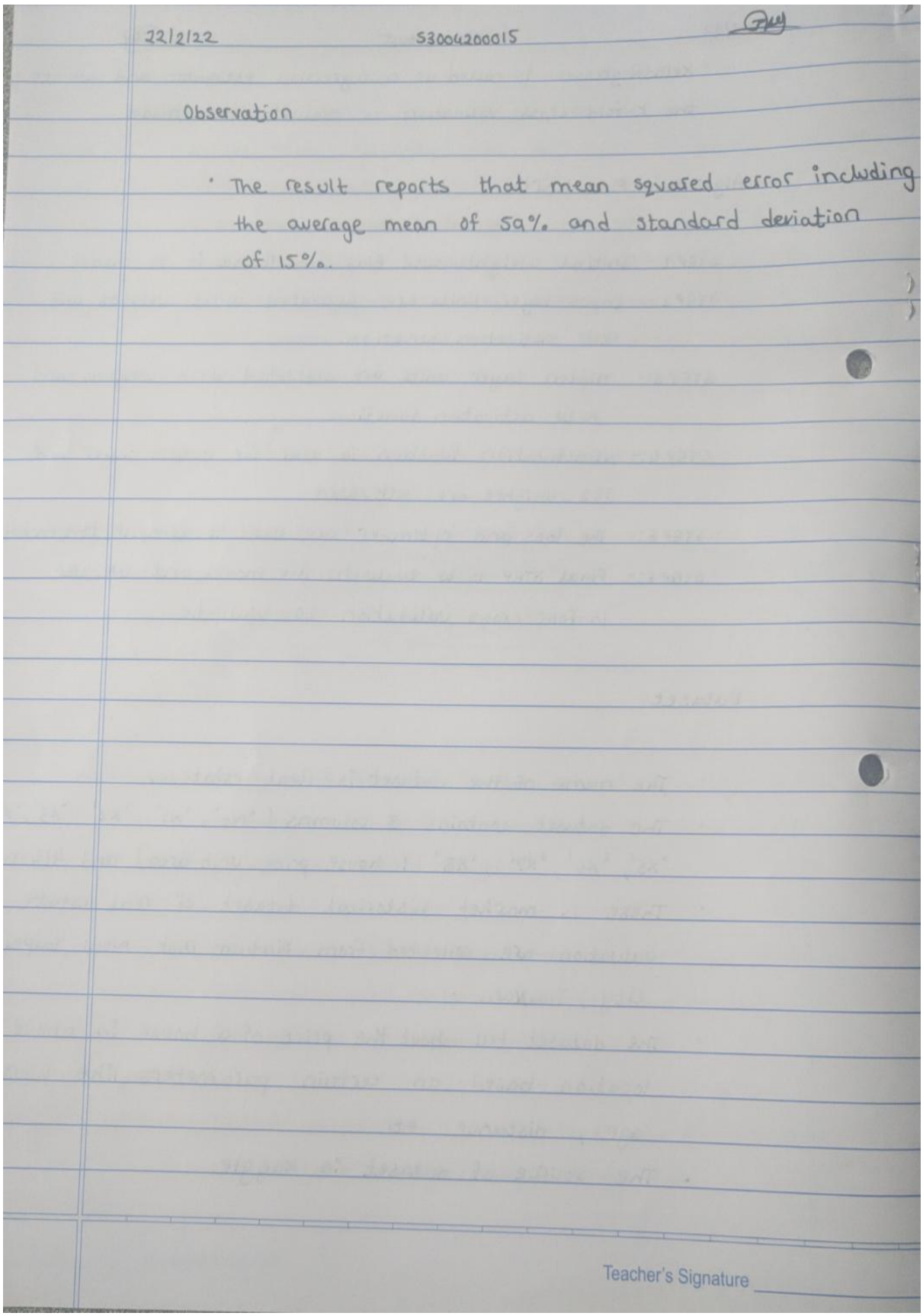
```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 20 23:08:18 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Import all the necessary Libraries
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/Real estate.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 2:7].values
Y = df.iloc[:, -1].values
# define wider model
model = Sequential()
#hidden layer 1 with relu function
model.add(Dense(6, input_dim=5, kernel_initializer='normal', activation='relu'))
#hidden layer 2 with relu function
model.add(Dense(6 ,activation='relu'))
#output layer with linear function as it is regression
model.add(Dense(1, activation='linear'))
```

```
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam')
#evaulating the model using KFold method
estimator = KerasRegressor(model, epochs=100, batch_size=5, verbose=0)
kfold = KFold(n_splits=10)
results = cross_val_score(estimator, X, Y, cv=kfold)
#Calculating Model Mean loss and standard deviation
print("Model: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Output



Observation



22/2/22 53004200015

Observation

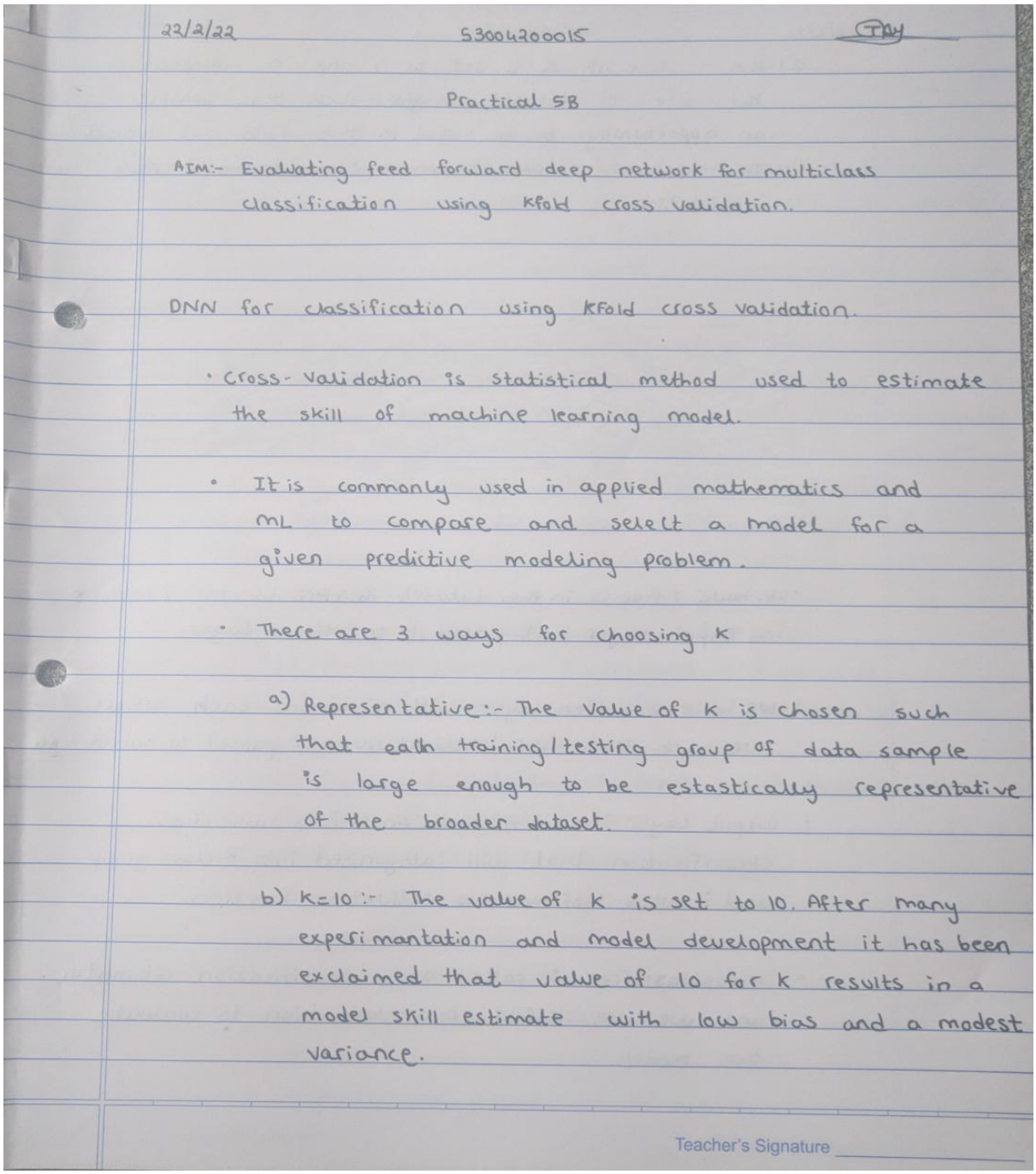
- The result reports that mean squared error including the average mean of 59% and standard deviation of 15%.

Teacher's Signature

Practical 5B

Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.

DNN for Classification using KFold cross validation



22/2/22 53004200015 TAY

Practical 5B

AIM:- Evaluating feed forward deep network for multiclass classification using Kfold cross validation.

DNN for classification using Kfold cross validation.

- Cross-validation is statistical method used to estimate the skill of machine learning model.
- It is commonly used in applied mathematics and ML to compare and select a model for a given predictive modeling problem.
- There are 3 ways for choosing k

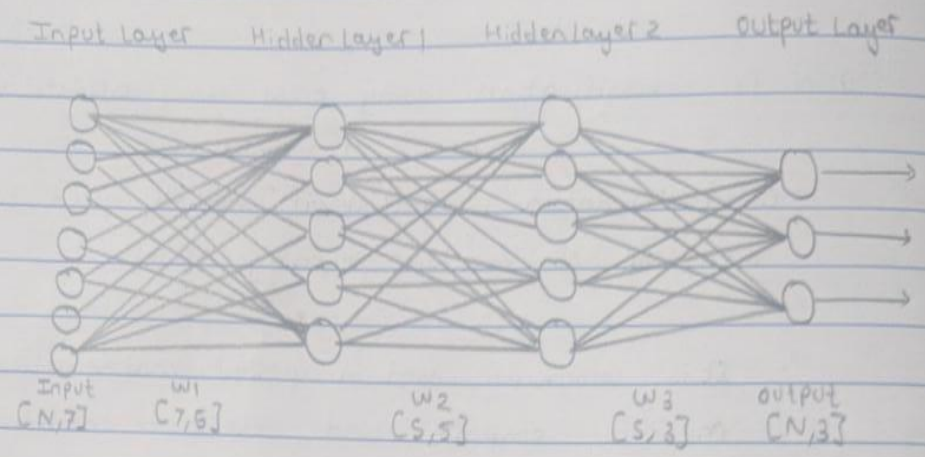
a) Representative :- The value of k is chosen such that each training/testing group of data sample is large enough to be estastically representative of the broader dataset.

b) k=10 :- The value of k is set to 10. After many experimantation and model development it has been exclaimed that value of 10 for k results in a model skill estimate with low bias and a modest variance.

Teacher's Signature _____

22/2/22 53004200015 JKy

c) K=n :- Size of K is set to n and n represents the size of dataset. To give each test sample an opportunity to be used in the hold out dataset This approach is also known as 'leave one out' cross validation.



- we have 7 feaures in the dataset, therefore we use 7 neurons in Input layer and passed it to Hidden Layers.

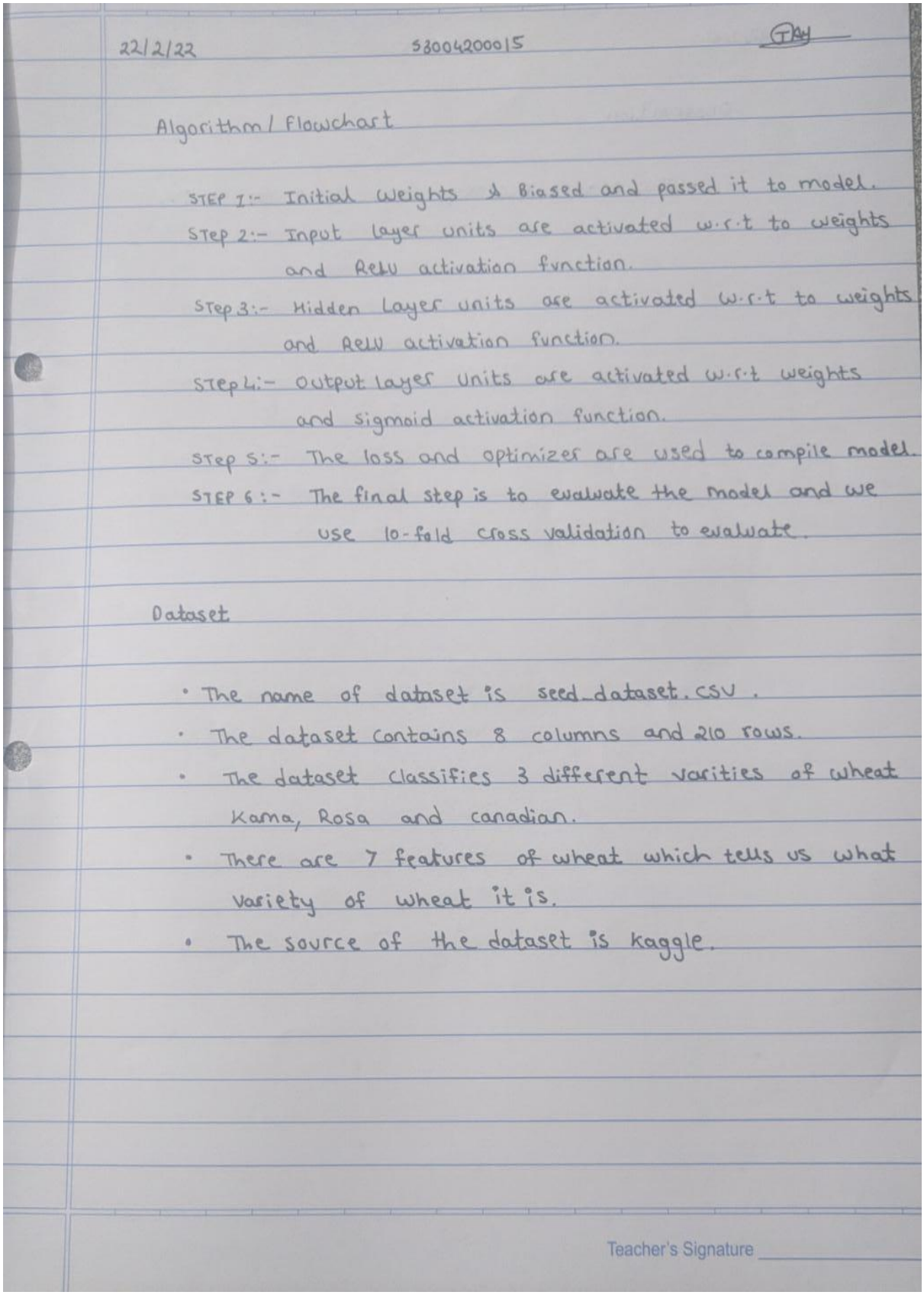
- We have 2 hidden layer with 5 neurons each, where we have used 'relu' activation and passed to output layer

- Output Layer has 3 neurons since it is multiclass classification that will categorized into 3 class group and it uses 'softmax' as activation function.

- KerasClassifier is called as a classification estimator and we use K-fold cross validation to evaluate the model.

Teacher's Signature

Algorithm and Dataset



22/2/22 53004200015 RAY

Algorithm / Flowchart

STEP 1:- Initial weights & Biased and passed it to model.
STEP 2:- Input layer units are activated w.r.t to weights and ReLU activation function.
STEP 3:- Hidden Layer units are activated w.r.t to weights and ReLU activation function.
STEP 4:- Output layer units are activated w.r.t weights and sigmoid activation function.
STEP 5:- The loss and optimizer are used to compile model.
STEP 6:- The final step is to evaluate the model and we use 10-fold cross validation to evaluate.

Dataset

- The name of dataset is seed_dataset.csv.
- The dataset contains 8 columns and 210 rows.
- The dataset classifies 3 different varities of wheat Kama, Rosa and canadian.
- There are 7 features of wheat which tells us what variety of wheat it is.
- The source of the dataset is kaggle.

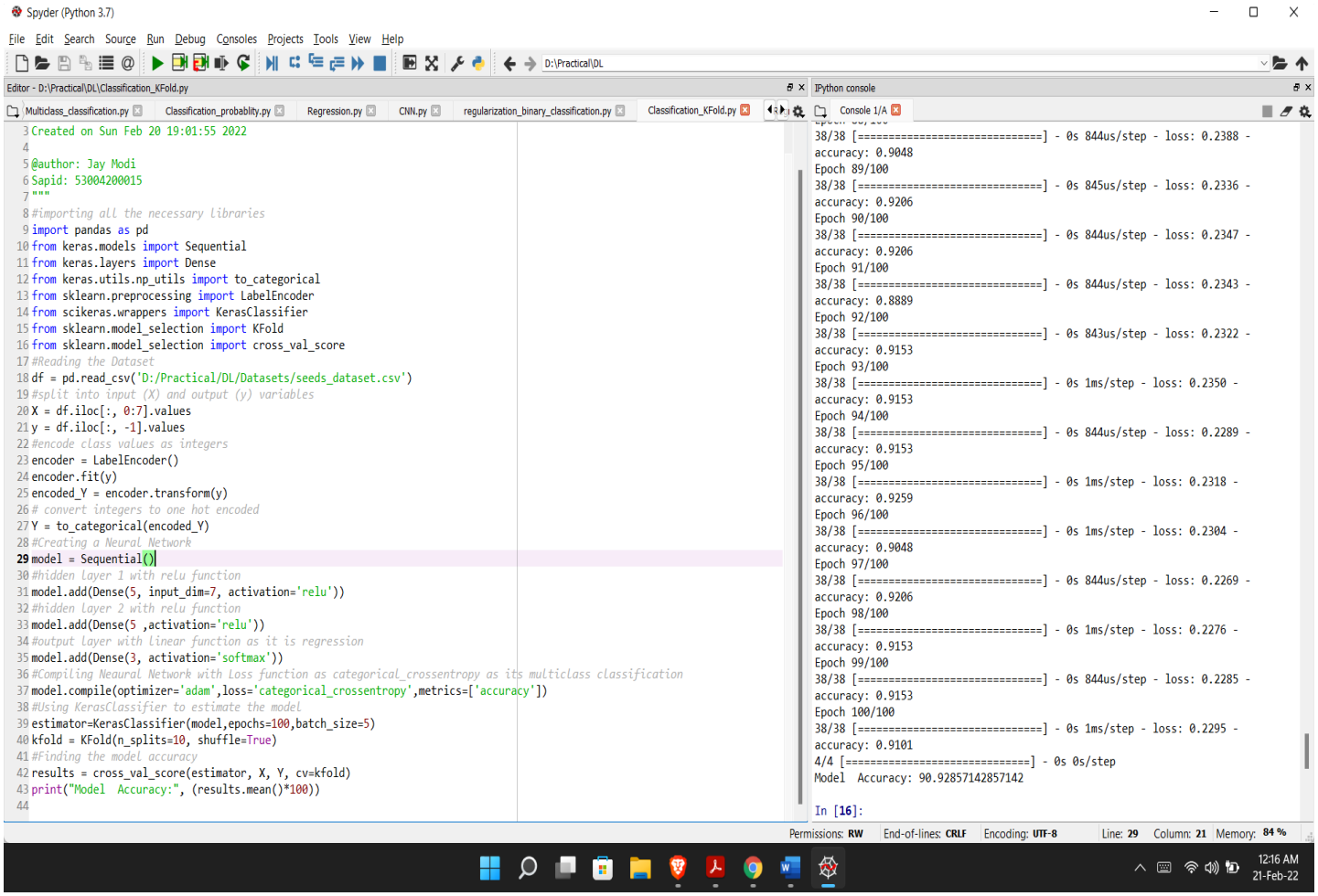
Teacher's Signature

Code

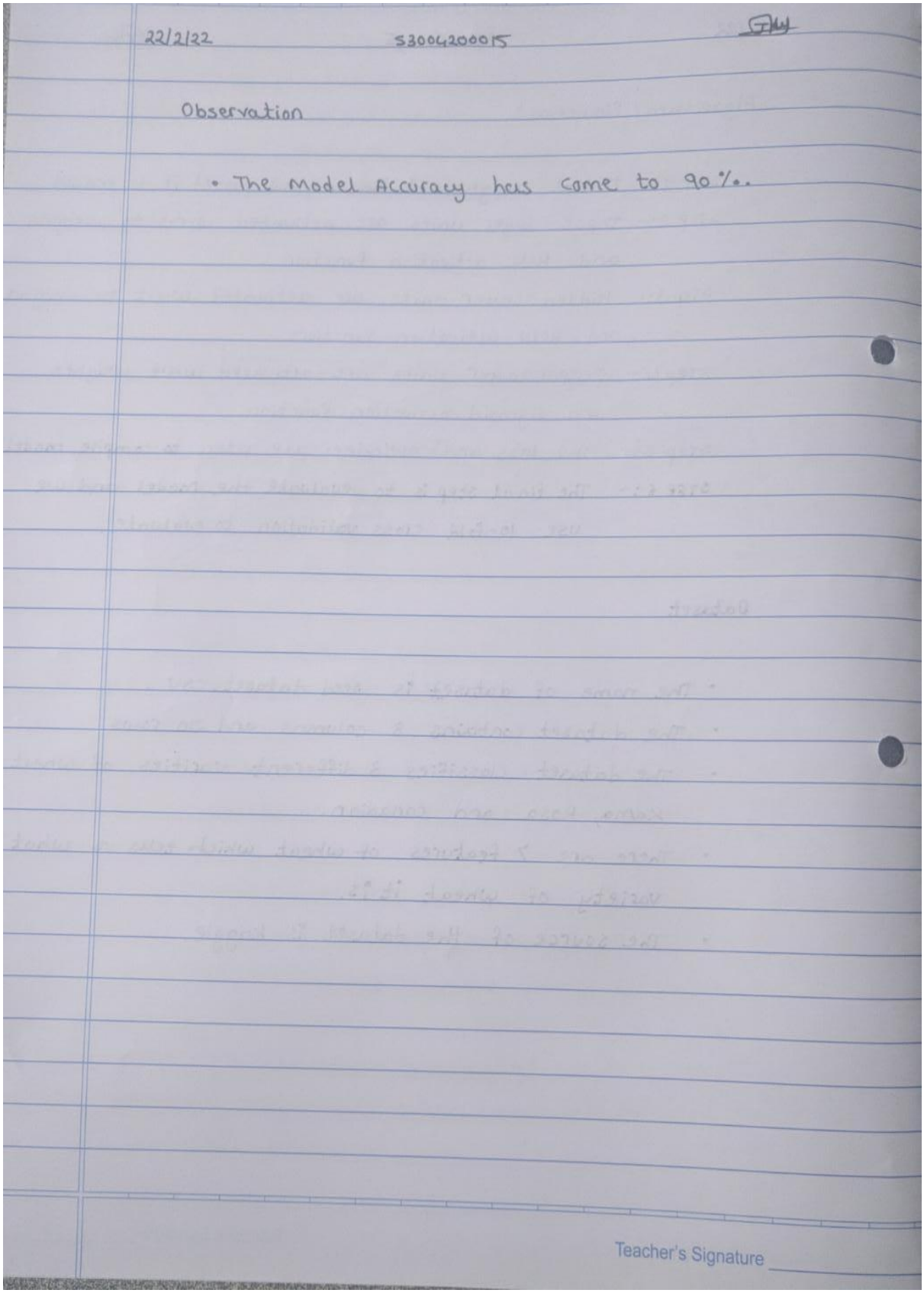
```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 20 19:01:55 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import LabelEncoder
from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/seeds_dataset.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:7].values
y = df.iloc[:, -1].values
#encode class values as integers
encoder = LabelEncoder()
encoder.fit(y)
encoded_Y = encoder.transform(y)
# convert integers to one hot encoded
Y = to_categorical(encoded_Y)
```

```
#Creating a Neural Network
model = Sequential()
#hidden layer 1 with relu function
model.add(Dense(5, input_dim=7, activation='relu'))
#hidden layer 2 with relu function
model.add(Dense(5 ,activation='relu'))
#output layer with linear function as it is regression
model.add(Dense(3, activation='softmax'))
#Compiling Neaural Network with Loss function as categorical_crossentropy as its multiclass classification
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
#Using KerasClassifier to estimate the model
estimator=KerasClassifier(model,epochs=100,batch_size=5)
kfold = KFold(n_splits=10, shuffle=True)
#Finding the model accuracy
results = cross_val_score(estimator, X, Y, cv=kfold)
print("Model  Accuracy:", (results.mean()*100))
```

Output


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - D:\Practical\DL\Classification_KFold.py
Multiclass_classification.py
Classification_probablity.py
Regression.py
CNN.py
regularization_binary_classification.py
Classification_KFold.py
IPython console
Console 1/A
3 Created on Sun Feb 20 19:01:55 2022
4
5 @author: Jay Modi
6 Sapid: 53004200015
7 """
8 #importing all the necessary libraries
9 import pandas as pd
10 from keras.models import Sequential
11 from keras.layers import Dense
12 from keras.utils.np_utils import to_categorical
13 from sklearn.preprocessing import LabelEncoder
14 from scikeras.wrappers import KerasClassifier
15 from sklearn.model_selection import KFold
16 from sklearn.model_selection import cross_val_score
17 #Reading the Dataset
18 df = pd.read_csv('D:/Practical/DL/Datasets/seeds_dataset.csv')
19 #split into input (X) and output (y) variables
20 X = df.iloc[:, 0:7].values
21 y = df.iloc[:, -1].values
22 #encode class values as integers
23 encoder = LabelEncoder()
24 encoder.fit(y)
25 encoded_Y = encoder.transform(y)
26 # convert integers to one hot encoded
27 Y = to_categorical(encoded_Y)
28 #Creating a Neural Network
29 model = Sequential()
30 #hidden layer 1 with relu function
31 model.add(Dense(5, input_dim=7, activation='relu'))
32 #hidden layer 2 with relu function
33 model.add(Dense(5 ,activation='relu'))
34 #output layer with linear function as it is regression
35 model.add(Dense(3, activation='softmax'))
36 #Compiling Neaural Network with Loss function as categorical_crossentropy as its multiclass classification
37 model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
38 #Using KerasClassifier to estimate the model
39 estimator=KerasClassifier(model,epochs=100,batch_size=5)
40 kfold = KFold(n_splits=10, shuffle=True)
41 #Finding the model accuracy
42 results = cross_val_score(estimator, X, Y, cv=kfold)
43 print("Model Accuracy:", (results.mean()*100))
44
38/38 [==============================] - 0s 844us/step - loss: 0.2388 -
accuracy: 0.9048
Epoch 89/100
38/38 [==============================] - 0s 845us/step - loss: 0.2336 -
accuracy: 0.9206
Epoch 90/100
38/38 [==============================] - 0s 844us/step - loss: 0.2347 -
accuracy: 0.9206
Epoch 91/100
38/38 [==============================] - 0s 844us/step - loss: 0.2343 -
accuracy: 0.8889
Epoch 92/100
38/38 [==============================] - 0s 843us/step - loss: 0.2322 -
accuracy: 0.9153
Epoch 93/100
38/38 [==============================] - 0s 1ms/step - loss: 0.2350 -
accuracy: 0.9153
Epoch 94/100
38/38 [==============================] - 0s 844us/step - loss: 0.2289 -
accuracy: 0.9153
Epoch 95/100
38/38 [==============================] - 0s 1ms/step - loss: 0.2318 -
accuracy: 0.9259
Epoch 96/100
38/38 [==============================] - 0s 1ms/step - loss: 0.2304 -
accuracy: 0.9048
Epoch 97/100
38/38 [==============================] - 0s 844us/step - loss: 0.2269 -
accuracy: 0.9206
Epoch 98/100
38/38 [==============================] - 0s 1ms/step - loss: 0.2276 -
accuracy: 0.9153
Epoch 99/100
38/38 [==============================] - 0s 844us/step - loss: 0.2285 -
accuracy: 0.9153
Epoch 100/100
38/38 [==============================] - 0s 1ms/step - loss: 0.2295 -
accuracy: 0.9101
4/4 [==============================] - 0s 0s/step
Model Accuracy: 90.92857142857142
In [16]:
Permissions: RW
End-of-lines: CRLF
Encoding: UTF-8
Line: 29
Column: 21
Memory: 84 %
12:16 AM
21-Feb-22

Observation


22/2/22 53004200015

Observation

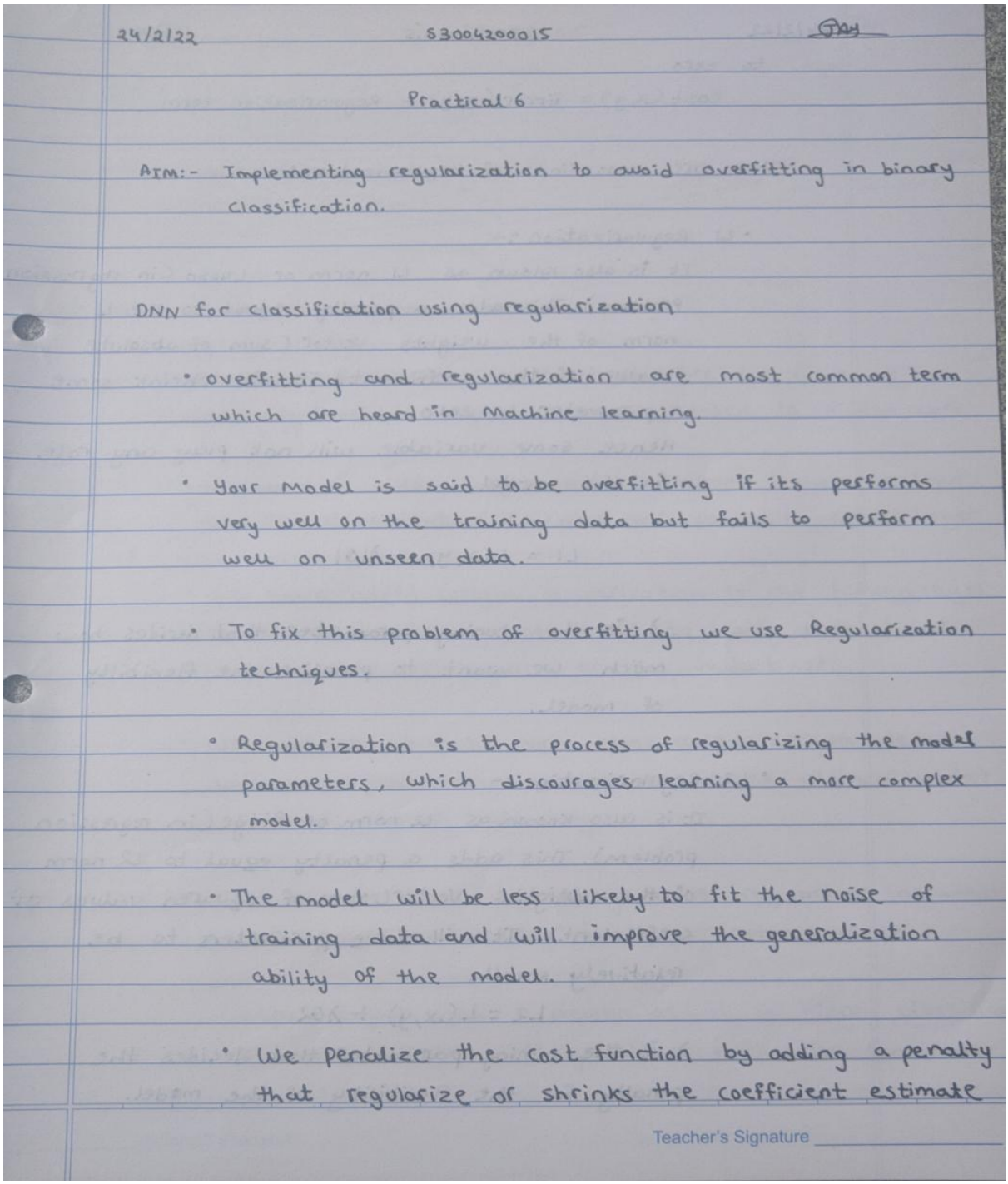
- The Model Accuracy has come to 90%.

Teacher's Signature

Practical 6

Implementing regularization to avoid overfitting in binary classification.

DNN for Classification using regularization



24/2/22 53004200015 Jay

Practical 6

AIM:- Implementing regularization to avoid overfitting in binary classification.

DNN for classification using regularization

- overfitting and regularization are most common term which are heard in machine learning.
- Your Model is said to be overfitting if its performs very well on the training data but fails to perform well on unseen data.
- To fix this problem of overfitting we use Regularization techniques.
- Regularization is the process of regularizing the model parameters, which discourages learning a more complex model.
- The model will be less likely to fit the noise of training data and will improve the generalization ability of the model.
- We penalize the cost function by adding a penalty that regularize or shrinks the coefficient estimate

Teacher's Signature

24/2/22 53004200015 Day

to zero.

Cost(x,y) = Error(x,y) + Regularization term

* There are two kinds of Regularization :-

* L1 Regularization :-

It is also known as L1 norm or Lasso (in regression problem). This adds a penalty equal to the L1 norm of the weights vector (sum of absolute value of the coefficient). It will shrink some parameter to zero.

Hence some variable will not play any role in the model.

$$L1 = L(x,y) + \lambda|\theta|$$

λ is the tuning parameter that decides how much we want to penalize the flexibilty of model.

* L2 Regularization :-

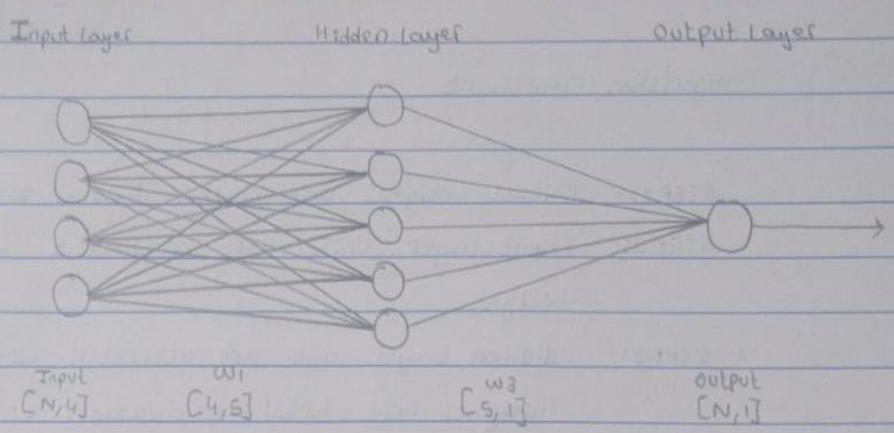
It is also known as L2 norm or Ridge (in regression problem). This adds a penalty equal to L2 norm of the weights vector (sum of squared values of coefficient). It will force parameters to be relatively small.

$$L2 = L(x,y) + \lambda\theta 2$$

λ is the tuning parameter that decides the penalty for the flexibility of the model.

Teacher's Signature

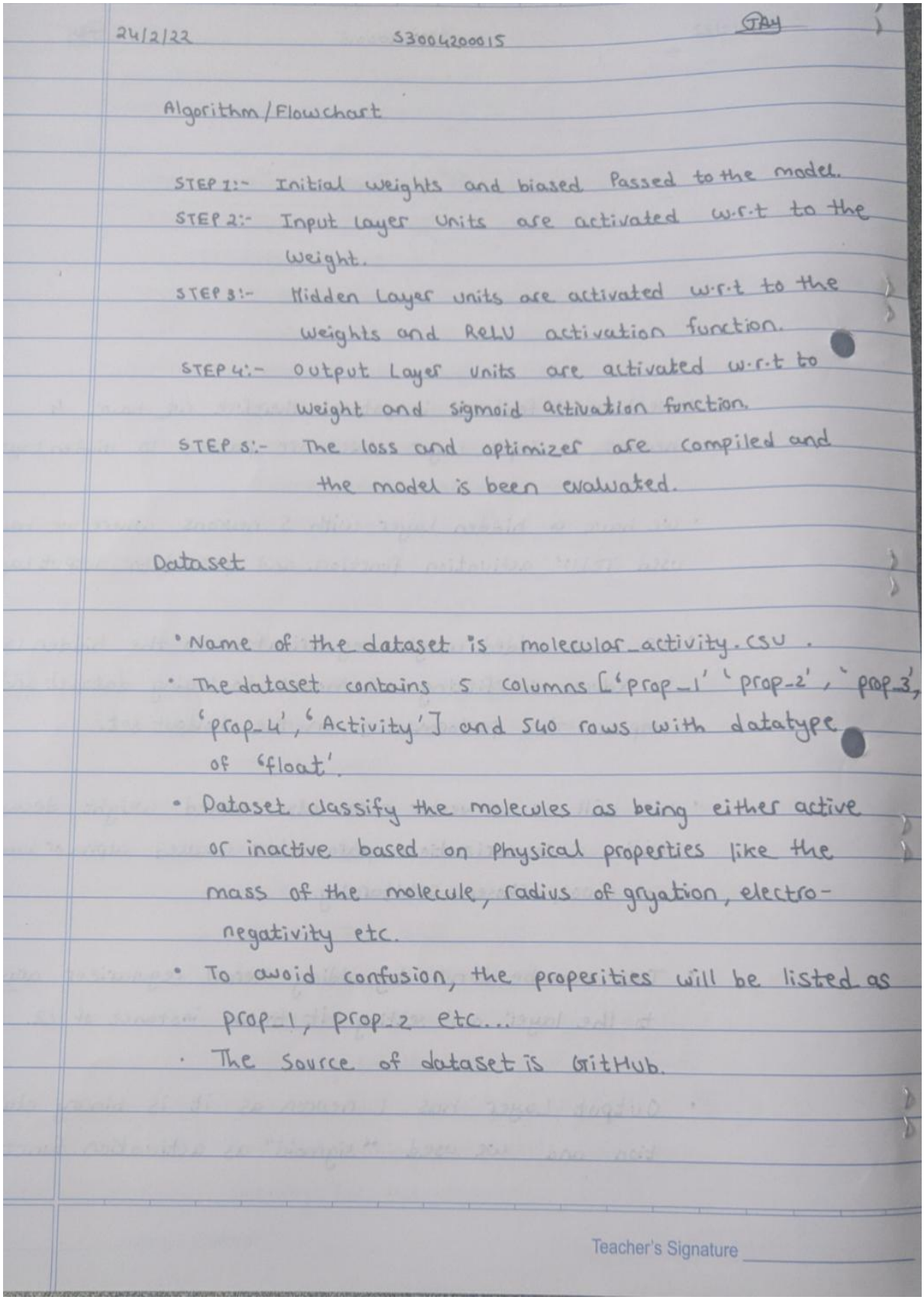
24/2/22 53004200015



- we have 4 features in dataset therefore we have 4 neurons in Input Layer which are passed to Hidden Layer.
- we have a hidden Layer with 5 neurons where we have used 'reLU' activation function. and passed to output Layer.
- we have added weight regularization to the hidden layer to reduce overfitting of model to traing dataset and improve the performance on the holdout set.
- we will use L2 vector norm also called weight decay with a regularization parameter (called alpha or lamda) of 0.001, Chosen arbitarally.
- This can be done by adding kernel_regularizer argument to the layer and setting it to an instance of L2.
- Output Layer has 1 neuron as it is binary classification and we used "sigmoid" as activation function

Teacher's Signature _____

Algorithm and Dataset



24/2/22 53004200015 JAY

Algorithm / Flowchart

STEP 1:- Initial weights and biased passed to the model.

STEP 2:- Input layer units are activated w.r.t to the weight.

STEP 3:- Hidden layer units are activated w.r.t to the weights and ReLU activation function.

STEP 4:- Output Layer units are activated w.r.t to weight and sigmoid activation function.

STEP 5:- The loss and optimizer are compiled and the model is been evaluated.

Dataset

- Name of the dataset is molecular_activity.csv.
- The dataset contains 5 columns ['prop_1' 'prop_2', 'prop_3', 'prop_4', 'Activity'] and 540 rows with datatype of 'float'.
- Dataset classify the molecules as being either active or inactive based on physical properties like the mass of the molecule, radius of gryation, electro-negativity etc.
- To avoid confusion, the properities will be listed as prop_1, prop_2 etc...
- The source of dataset is GitHub.

Teacher's Signature _____

Code

```
"""
Created on Sun Feb 20 17:56:08 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.regularizers import l2
from matplotlib import pyplot
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/molecular_activity.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:4].values
y = df['Activity']
#Traing the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)
#Creating a Neural Network
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(4,)), #Input Layer
    keras.layers.Dense(16, activation=tf.nn.relu,kernel_regularizer=l2(0.001)),
#Hidden Layer 1 with relu as activation function
```

```
        keras.layers.Dense(16,
activation=tf.nn.relu,kernel_regularizer=l2(0.001)), #Hidden Layer 2 with relu
as activation function

   keras.layers.Dense(1, activation=tf.nn.sigmoid), #Output Layer with sigmoid
as activation function

])

#Compiling Neaural Network with Loss function as Cross-entropy

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

#Fitting the model

history=model.fit(X_train, y_train,validation_data=(X_train, y_train),
epochs=50, batch_size=1)

#Calculating Model Accuracy and Loss

test_loss, test_acc = model.evaluate(X_test, y_test)

print('Test accuracy:', test_acc)

print('Test loss:', test_loss)

#Plotting the accuracy graph

pyplot.plot(history.history['accuracy'],label='train')

pyplot.plot(history.history['val_accuracy'],label='test')

pyplot.legend()

pyplot.show()
```

Output

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

D:\Practical\DL

Editor - D:\Practical\DL\regularization_binary_classification.py

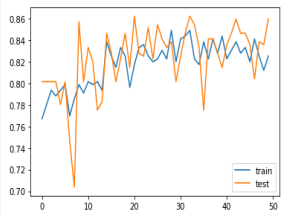
Binary_classification.py | Multiclass_classification.py | Classification_probablity.py | Regression.py | CNN.py | regularization_binary_classification.py

```
"""
Created on Sun Feb 20 17:56:08 2022

@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from keras.regularizers import l2
from matplotlib import pyplot
#Reading the Dataset
df = pd.read_csv('D:/Practical/DL/Datasets/molecular_activity.csv')
#split into input (X) and output (y) variables
X = df.iloc[:, 0:4].values
y = df['Activity']
#Traing the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
#Creating a Neural Network
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(4,)), #Input Layer
    keras.layers.Dense(16, activation=tf.nn.relu,kernel_regularizer=l2(0.001)), #Hidden Layer 1 with relu as activation fu
    keras.layers.Dense(16, activation=tf.nn.relu,kernel_regularizer=l2(0.001)), #Hidden Layer 2 with relu as activation fu
    keras.layers.Dense(1, activation=tf.nn.sigmoid), #Output Layer with sigmoid as activation function
])
#Compiling Neaural Network with Loss function as Cross-entropy
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
#Fitting the model
history=model.fit(X_train, y_train,validation_data=(X_train, y_train), epochs=50, batch_size=1)
#Calculating Model Accuracy and Loss
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
print('Test loss:', test_loss)

pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

IPython console

Console 1/A

```
378/378 [==============================] - 1s 2ms/step - loss: 0.4244 -
accuracy: 0.8386 - val_loss: 0.4191 - val_accuracy: 0.8598
Epoch 44/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4292 -
accuracy: 0.8280 - val_loss: 0.4205 - val_accuracy: 0.8466
Epoch 45/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4192 -
accuracy: 0.8333 - val_loss: 0.4188 - val_accuracy: 0.8466
Epoch 46/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4309 -
accuracy: 0.8201 - val_loss: 0.4086 - val_accuracy: 0.8360
Epoch 47/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4104 -
accuracy: 0.8413 - val_loss: 0.4710 - val_accuracy: 0.8042
Epoch 48/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4301 -
accuracy: 0.8254 - val_loss: 0.3970 - val_accuracy: 0.8386
Epoch 49/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4280 -
accuracy: 0.8122 - val_loss: 0.4275 - val_accuracy: 0.8360
Epoch 50/50
378/378 [==============================] - 1s 2ms/step - loss: 0.4105 -
accuracy: 0.8254 - val_loss: 0.4204 - val_accuracy: 0.8598
6/6 [==============================] - 0s 0s/step - loss: 0.4241 - accuracy:
0.8333
Test accuracy: 0.8333333134651184
Test loss: 0.42411738634109497
```

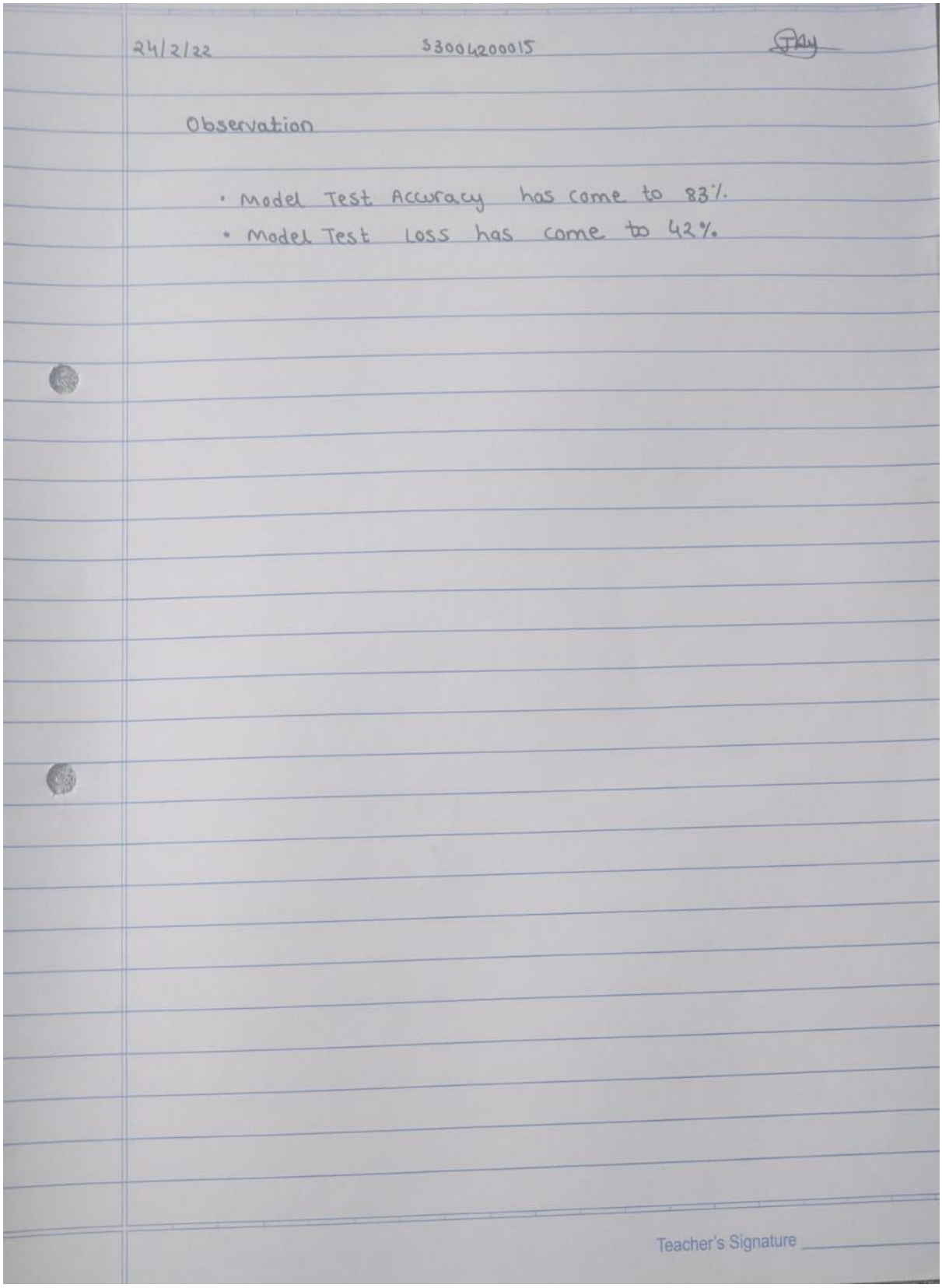



```
In [17]:
```

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 30 | Column: 60 | Memory: 80 %

6:57 PM
20-Feb-22

Observation



Practical 7

Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

Recurrent neural network for stock price analysis.

3/3/22 S3004200015

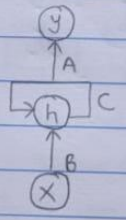
Practical 7

AIM:- Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

Recurrent neural network for stock price analysis.

- Recurrent Neural Network (RNN) are a type of Neural Network where the output from previous step are fed as input to the current step.
- An RNN can handle sequential data, accepting the current input data and previously received inputs. RNNs can memorize previous input due to their internal memory.

Recurrent Neural Network



- Here, "X" is the input layer, "h" is hidden layer, and "y" is the output layer. A, B and C are network parameters used to improve the output of the model.

Teacher's Signature _____

3/3/22 53004200015 TRY

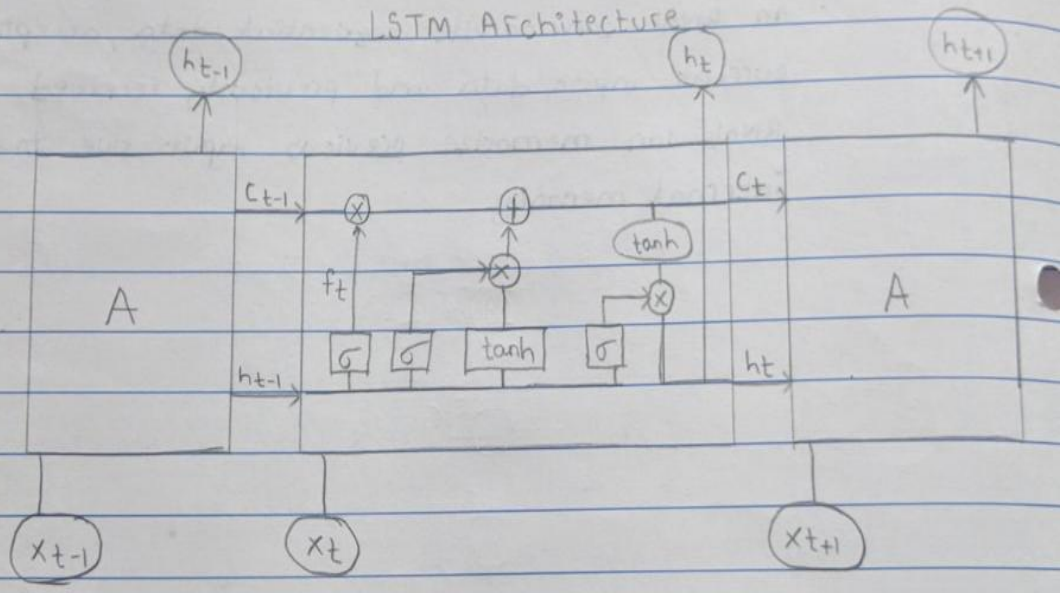
- There are main two issues with standard RNNs
  - Vanishing Gradient Problem
  - Exploding Gradient Problem

Long training time, poor performance, and bad accuracy are the major issues in gradient problem.

- To deal with gradient problem we use Long-Short Term Memory Network. (LSTMs)

- LSTMs are special kind of RNN - capable of learning long-term dependencies by remembering information for long periods is the default behaviour.

LSTM Architecture



Teacher's Signature _____

3/3/22 53004200015 JKW

- LSTM work in a 3-step process

STEP 1:- Decide How Much Past Data It Should Remember

The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at the previous state (h_{t-1}) along with the current input x_t and computes the function.

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f)$$

f_t = forget gate which decides which information to delete that is not important from previous time step.

STEP 2:- Decide How Much This Unit Adds to Current State

In the second layer, there are two parts. One is the sigmoid function, and other is the tanh function. In the sigmoid function, it decides which values to let through (0 or 1). tanh function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

Teacher's Signature _____

3/3/22 53004200015

$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i)$

$C_t = \tanh(w_C \cdot [h_{t-1}, x_t] + b_C)$

i_t = input gate which determines information to let through based on its significance in the current time step.

STEP 3:- Decide What Part of the Current Cell State Makes It to the Output.

The third step is to decide what output will be. First, we run a sigmoid layer, which decides what part of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

$O_t = \sigma(w_o [h_{t-1}, x_t] + b_o)$

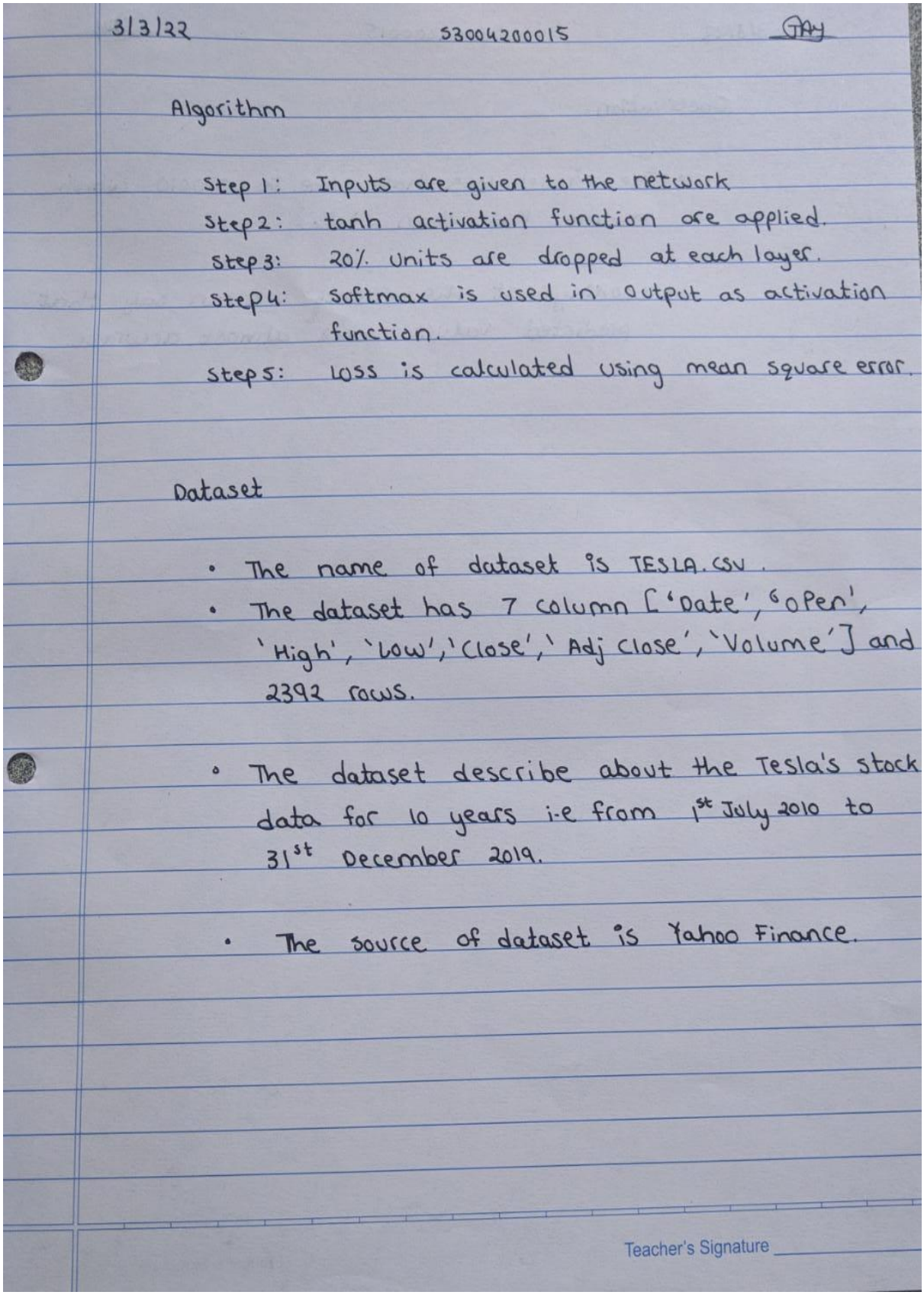
$h_t = O_t * \tanh(C_t)$

O_t = output gate allows the passed info to impact output in current time step.

- LSTM Network are popularly used on time-series data for classification, processing and making predictions

Teacher's Signature

Algorithm and Dataset



3/3/22 53004200015

Algorithm

Step 1: Inputs are given to the network
Step 2: tanh activation function are applied.
Step 3: 20% units are dropped at each layer.
Step 4: Softmax is used in output as activation function.
Step 5: Loss is calculated using mean square error.

Dataset

- The name of dataset is TESLA.csv.
- The dataset has 7 column ['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'] and 2392 rows.
- The dataset describe about the Tesla's stock data for 10 years i.e from 1st July 2010 to 31st December 2019.
- The source of dataset is Yahoo Finance.

Teacher's Signature

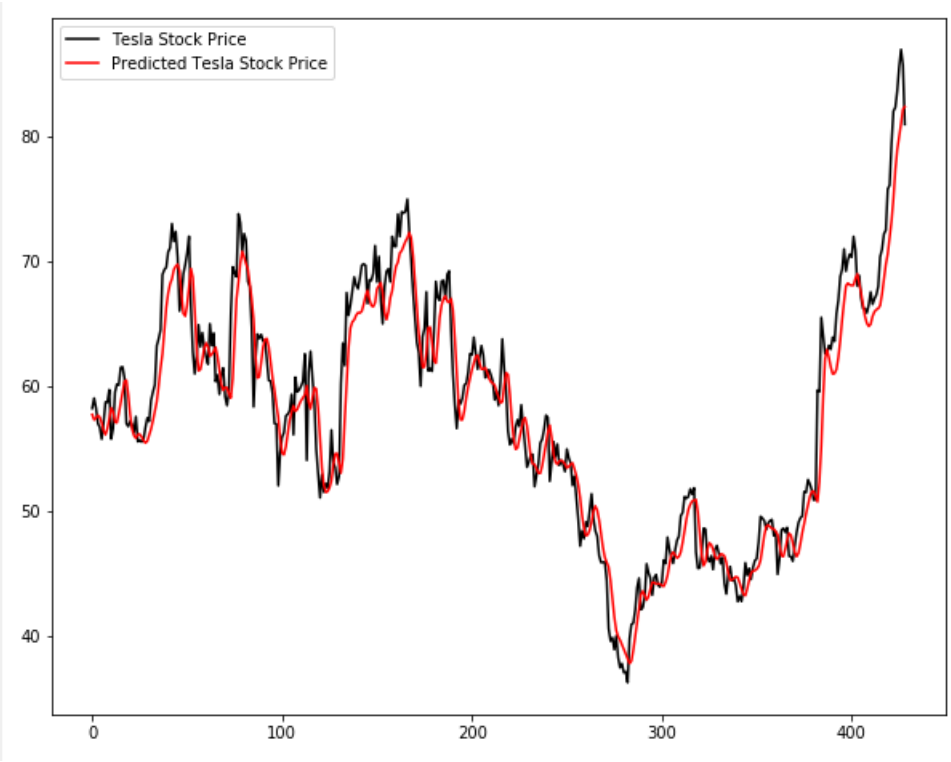
Code

```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 26 23:31:10 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Import all the neccessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
#Reading the dataset
df = pd.read_csv('D:/Practical/DL/Datasets/TESLA.csv')
df.shape
df = df['Open'].values
df = df.reshape(-1, 1)
#Split the data into training and testing sets
dataset_train = np.array(df[:int(df.shape[0]*0.8)])
dataset_test = np.array(df[int(df.shape[0]*0.8):])
#Scaling our data between zero and one using MinMaxScaler.
scaler = MinMaxScaler(feature_range=(0,1))
dataset_train = scaler.fit_transform(dataset_train)
```

```
dataset_test = scaler.transform(dataset_test)
#Function for creating datasets
def create_dataset(df):
    x = []
    y = []
    for i in range(50, df.shape[0]):
        x.append(df[i-50:i, 0])
        y.append(df[i, 0])
    x = np.array(x)
    y = np.array(y)
    return x,y
#Creating our training and testing data
x_train, y_train = create_dataset(dataset_train)
x_test, y_test = create_dataset(dataset_test)
#Reshaping our data to make it a 3D array in order to use it in LSTM Layer.
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
#Building Model
model = Sequential()
model.add(LSTM(units=96, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=96,return_sequences=True))
model.add(Dropout(0.2))
```

```
model.add(LSTM(units=96,return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=96))
model.add(Dropout(0.2))
model.add(Dense(units=1))
#Compiling the model
model.compile(loss='mean_squared_error', optimizer='adam')
#Fitting the model
model.fit(x_train, y_train, epochs=50, batch_size=32)
#Getting the data ready for actucal and predicted output
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
y_test_scaled = scaler.inverse_transform(y_test.reshape(-1, 1))
#Data Visualization
plt.figure(figsize=(10,8))
plt.plot(y_test_scaled, color = 'black', label = 'Tesla Stock Price')
plt.plot(predictions, color = 'red', label = 'Predicted Tesla Stock Price')
plt.legend()
```

Output



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - D:\Practical\DL\RNN.py

Classification_probablity.py | Regression.py | Regression_KFold.py | Classification_KFold.py | regularization_binary_classification.py | RNN.py*

```
# -*- coding: utf-8 -*-
"""
Created on Sat Feb 26 23:31:10 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Import all the neccessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
#Reading the dataset
df = pd.read_csv('D:/Practical/DL/Datasets/TESLA.csv')
df.shape
df = df['Open'].values
df = df.reshape(-1, 1)
#Split the data into training and testing sets
dataset_train = np.array(df[:int(df.shape[0]*0.8)])
dataset_test = np.array(df[int(df.shape[0]*0.8):])
#Scaling our data between zero and one using MinMaxScaler.
scaler = MinMaxScaler(feature_range=(0,1))
dataset_train = scaler.fit_transform(dataset_train)
dataset_test = scaler.transform(dataset_test)
#Function for creating datasets
def create_dataset(df):
    x = []
    y = []
    for i in range(50, df.shape[0]):
        x.append(df[i-50:i, 0])
        y.append(df[i, 0])
    x = np.array(x)
    y = np.array(y)
    return x,y
#Creating our training and testing data
x_train, y_train = create_dataset(dataset_train)
x_test, y_test = create_dataset(dataset_test)
#Reshaping our data to make it a 3D array in order to use it in LSTM Layer.
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
#Building Model
```

IPython console — Console 1/A

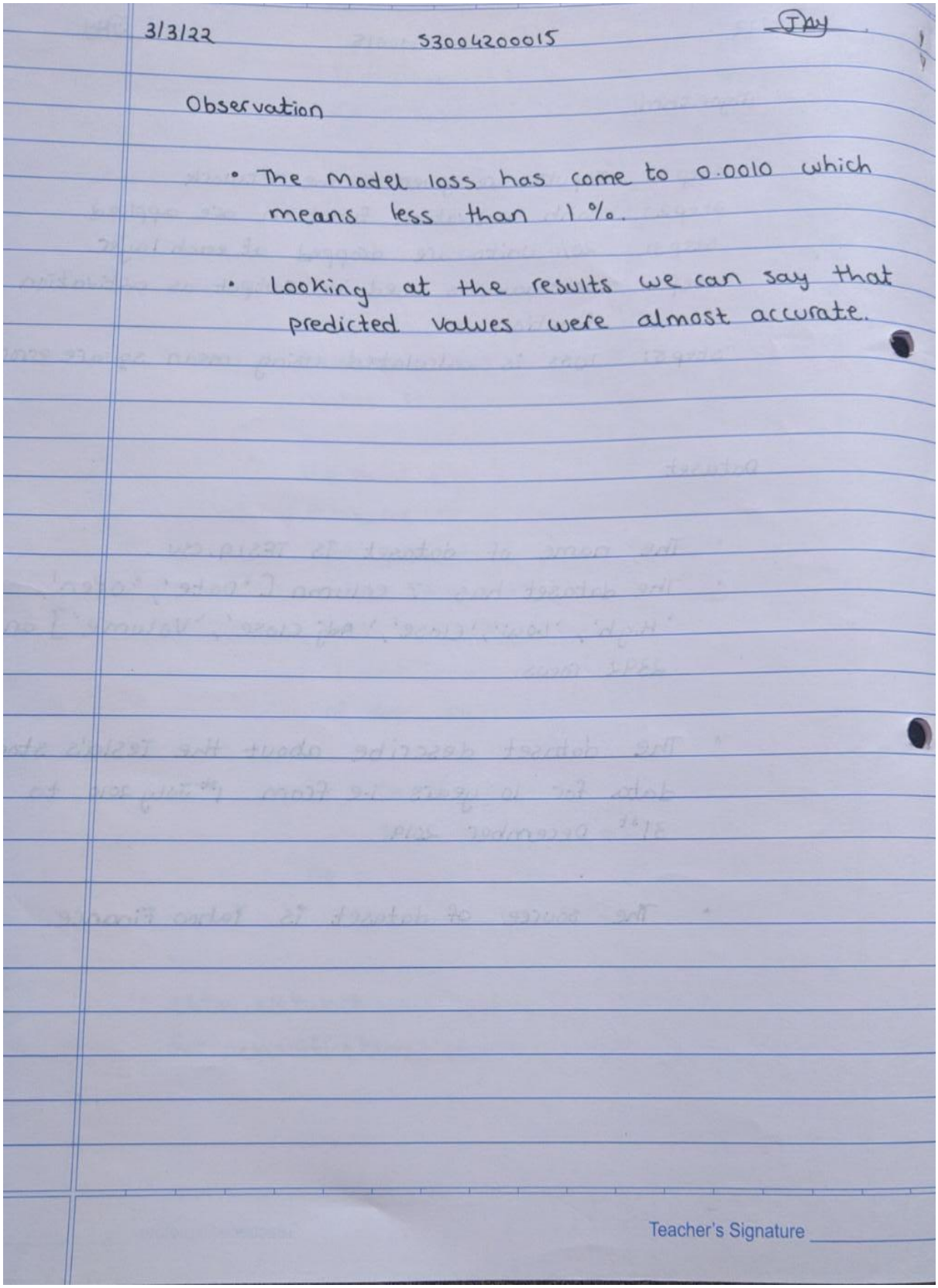
```
59/59 [==============================] - 8s 131ms/step - loss: 0.0011
Epoch 44/50
59/59 [==============================] - 7s 124ms/step - loss: 0.0011
Epoch 45/50
59/59 [==============================] - 6s 100ms/step - loss: 0.0012
Epoch 46/50
59/59 [==============================] - 6s 98ms/step - loss: 0.0011
Epoch 47/50
59/59 [==============================] - 6s 103ms/step - loss:
9.7437e-04
Epoch 48/50
59/59 [==============================] - 6s 101ms/step - loss: 0.0011
Epoch 49/50
59/59 [==============================] - 6s 101ms/step - loss: 0.0011
Epoch 50/50
59/59 [==============================] - 6s 102ms/step - loss: 0.0010
```


```
In [4]:
```

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 62 | Column: 27 | Memory: 83 %

12:32 AM
28-Feb-22

Observation



3/3/22 53004200015 Day

Observation

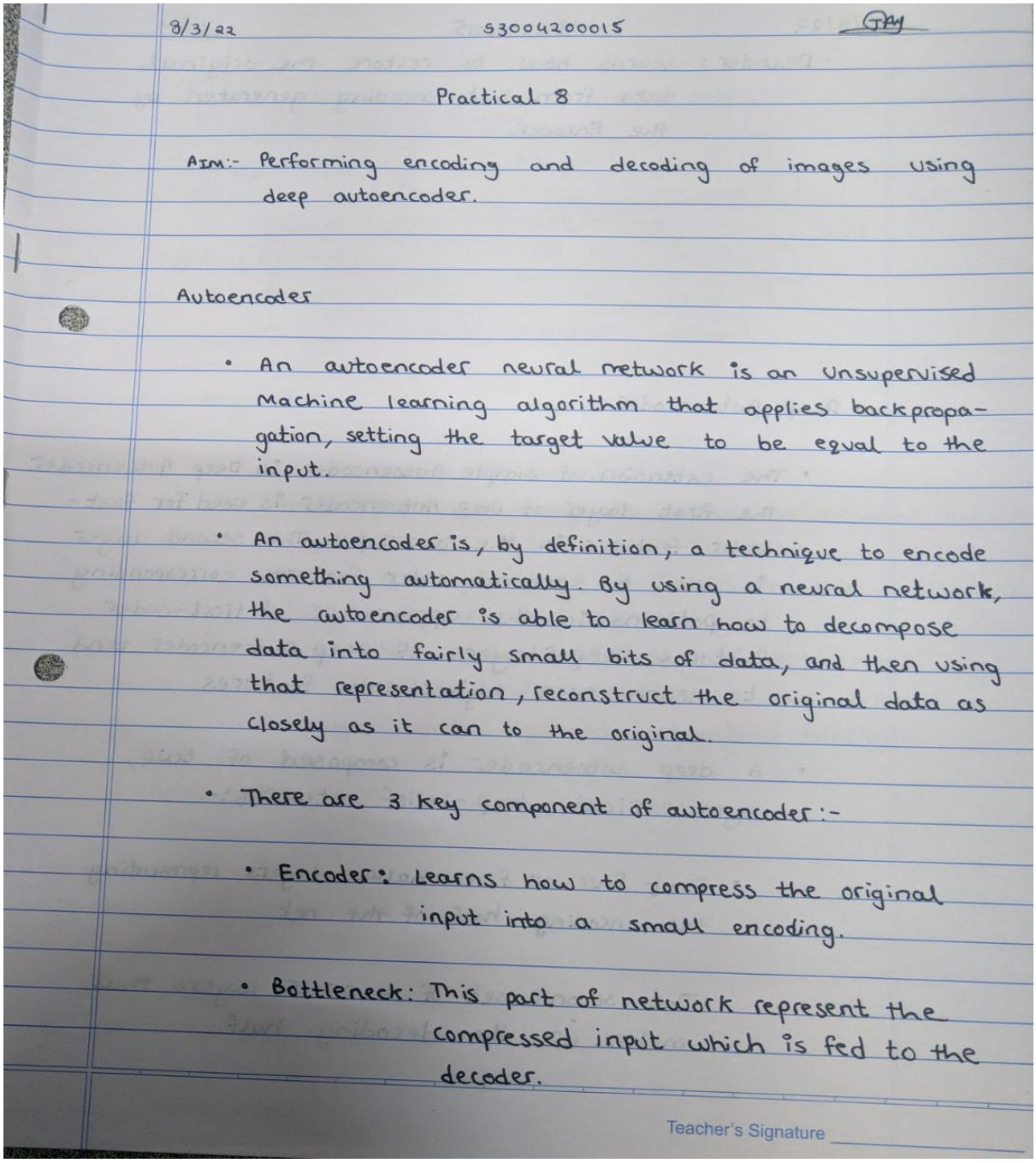
- The model loss has come to 0.0010 which means less than 1%.
- Looking at the results we can say that predicted values were almost accurate.

Teacher's Signature

Practical 8

Performing encoding and decoding of images using deep autoencoder.

Autoencoder



8/3/22 53004200015

Practical 8

Aim:- Performing encoding and decoding of images using deep autoencoder.

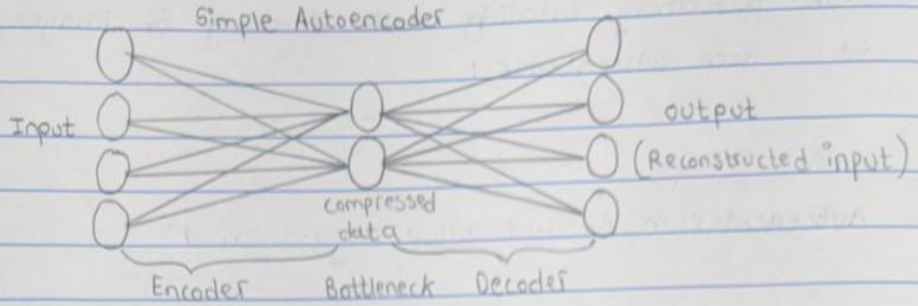
Autoencoder

- An autoencoder neural network is an unsupervised machine learning algorithm that applies backpropagation, setting the target value to be equal to the input.
- An autoencoder is, by definition, a technique to encode something automatically. By using a neural network, the autoencoder is able to learn how to decompose data into fairly small bits of data, and then using that representation, reconstruct the original data as closely as it can to the original.
- There are 3 key component of autoencoder:-
  - Encoder: Learns how to compress the original input into a small encoding.
  - Bottleneck: This part of network represent the compressed input which is fed to the decoder.

Teacher's Signature

8/3/22 53004200015 Jay

- Decoder: Learns how to restore the original data from that encoding generated by the Encoder.



- Deep Autoencoder

  - The extension of simple Autoencoder is Deep Autoencoder. The first layer of Deep Autoencoder is used for first-order features in the raw input. The second layer is used for second-order features corresponding to patterns in the appearance of first-order features. Deeper layers of Deep Autoencoder tend to learn even higher-order features.

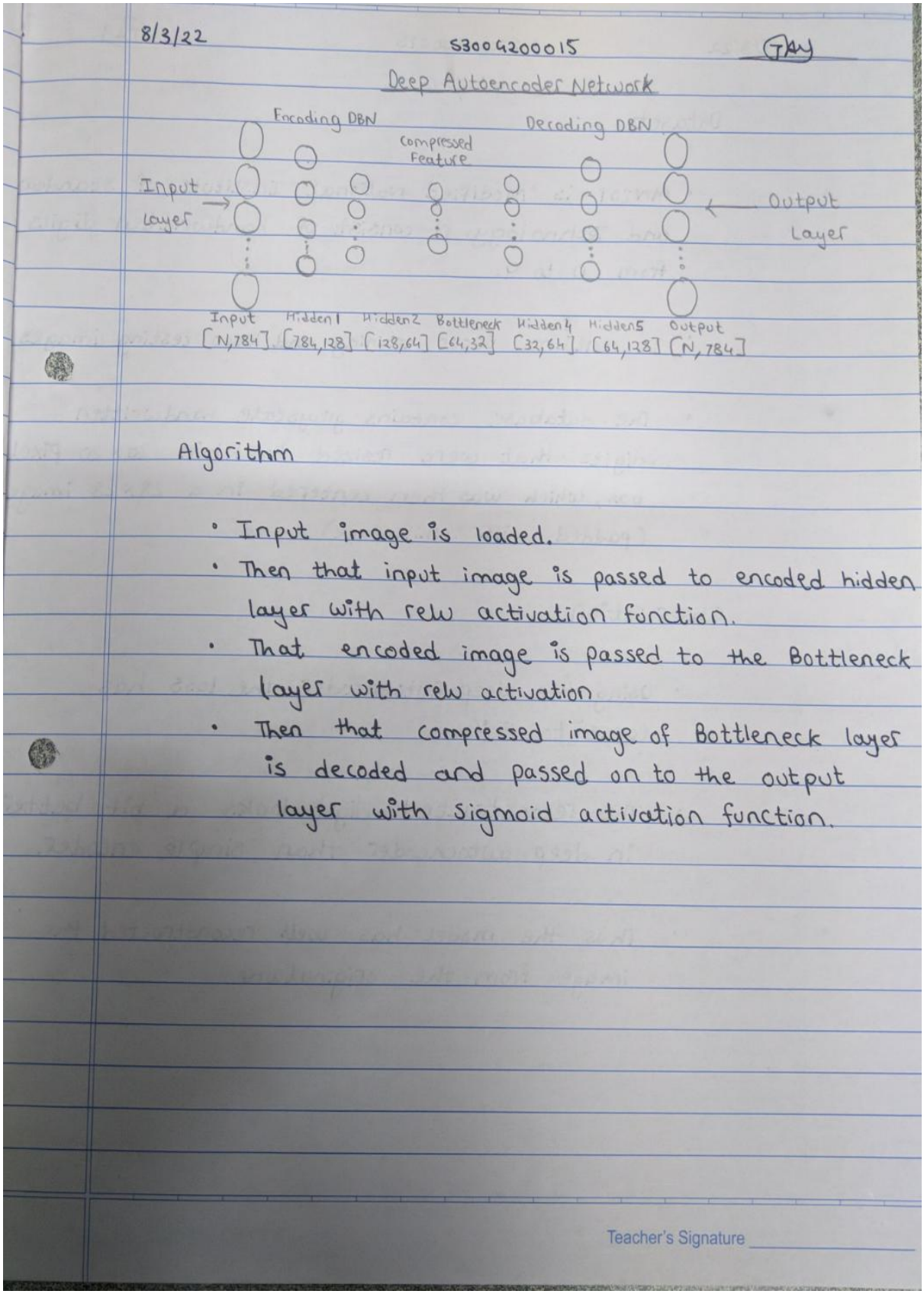
  - A deep autoencoder is composed of two, symmetrical deep-belief networks:-

    - First four or five shallow layers representing the encoding half of the net.

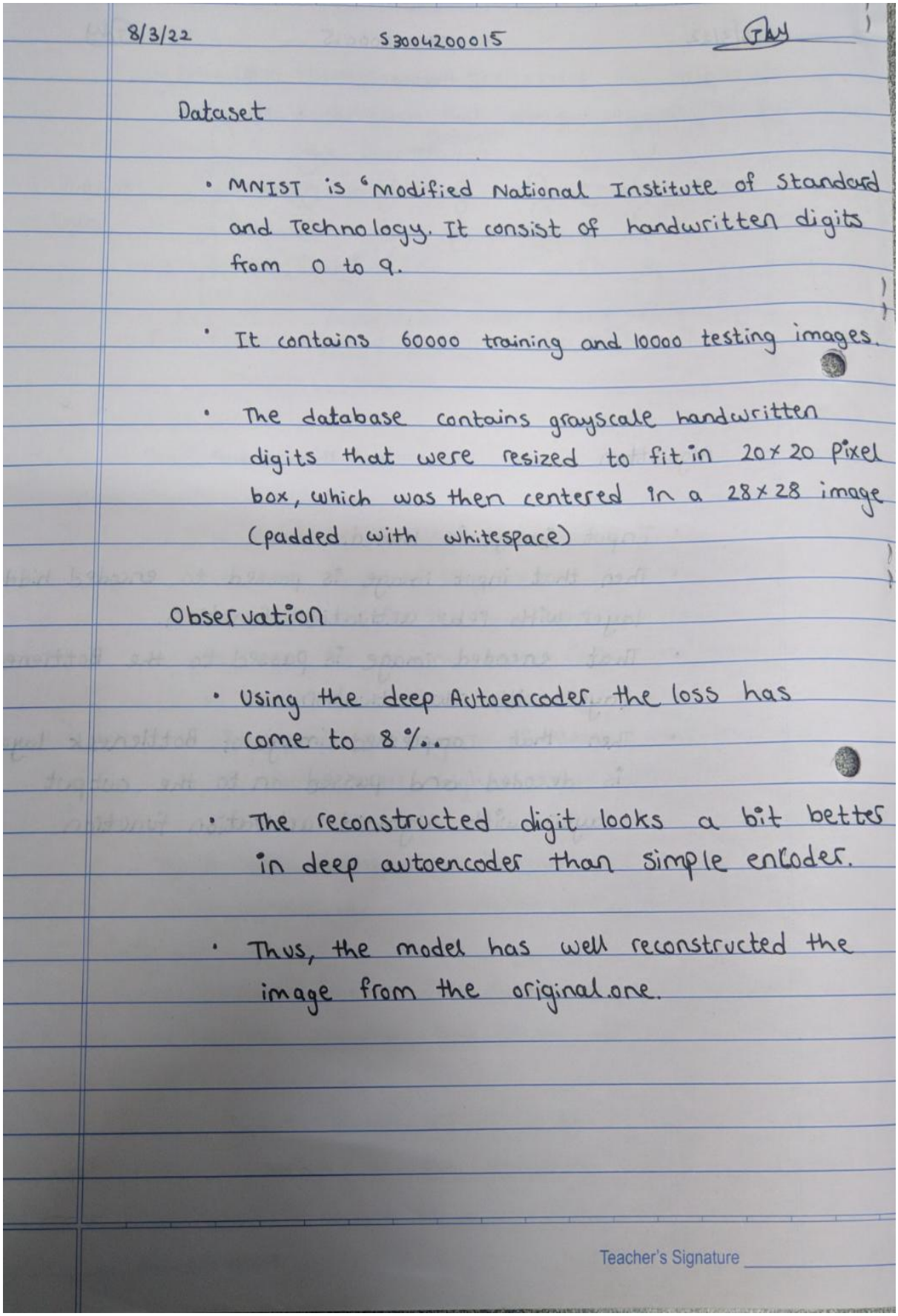
    - The second set of 4 or 5 layers that make up the decoding half.

Teacher's Signature _____

Algorithm


8/3/22
53004200015
Deep Autoencoder Network
Encoding DBN
Decoding DBN
Compressed Feature
Input layer
Output Layer
Input [N,784]
Hidden1 [784,128]
Hidden2 [128,64]
Bottleneck [64,32]
Hidden4 [32,64]
Hidden5 [64,128]
Output [N,784]
Algorithm
• Input image is loaded.
• Then that input image is passed to encoded hidden layer with relu activation function.
• That encoded image is passed to the Bottleneck layer with relu activation
• Then that compressed image of Bottleneck layer is decoded and passed on to the output layer with sigmoid activation function.
Teacher's Signature

Dataset



8/3/22 53004200015

Dataset

- MNIST is 'Modified National Institute of Standard and Technology. It consist of handwritten digits from 0 to 9.
- It contains 60000 training and 10000 testing images.
- The database contains grayscale handwritten digits that were resized to fit in 20×20 pixel box, which was then centered in a 28×28 image (padded with whitespace)

Observation

- Using the deep Autoencoder the loss has come to 8%..
- The reconstructed digit looks a bit better in deep autoencoder than simple encoder.
- Thus, the model has well reconstructed the image from the original one.

Teacher's Signature

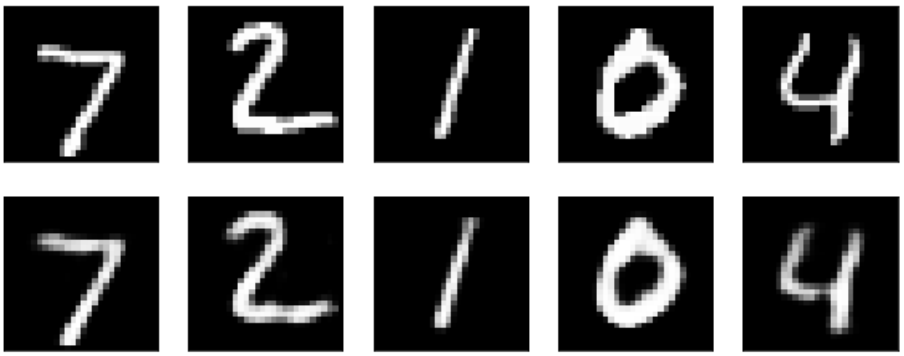
Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  6 20:33:17 2022
@author: Jay Modi
Sapid: 43004200015
"""
#Import neccessary libaries
import keras
from keras import layers
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
# This is the size of encoded representations
encoding_dim=32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
#this is our input image
input_img=keras.Input(shape=(784,))
#"encoded" is the encoded representation of the input
encoded=layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(32, activation='relu')(encoded)
#"decoded" is the lossy reconstruction of the input
decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded=layers.Dense(784, activation='sigmoid')(encoded)
#creating autoencoder model
```

```
autoencoder=keras.Model(input_img,decoded)
#create the encoder model
encoder=keras.Model(input_img,encoded)
encoded_input=keras.Input(shape=(encoding_dim,))
#Retrive the last layer of the autoencoder model
decoder_layer=autoencoder.layers[-1]
#create the decoder model
decoder=keras.Model(encoded_input,decoder_layer(encoded_input))
#Compiling the autoencoder model with binary crossentropy loss, and the
Adam optimizer
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
#scale and make train and test dataset
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=X_train.reshape((len(X_train),np.prod(X_train.shape[1:])))
X_test=X_test.reshape((len(X_test),np.prod(X_test.shape[1:])))
print(X_train.shape)
print(X_test.shape)
#train autoencoder with training dataset with 50 epochs and using minibatch
approach
autoencoder.fit(X_train,X_train,epochs=50,batch_size=256,shuffle=True,valida
tion_data=(X_test,X_test))
# Encode and decode some digits
encoded_imgs=encoder.predict(X_test)
decoded_imgs=decoder.predict(encoded_imgs)
# How many digits we will display
n = 5
```

```
plt.figure(figsize=(10, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(X_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Output



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

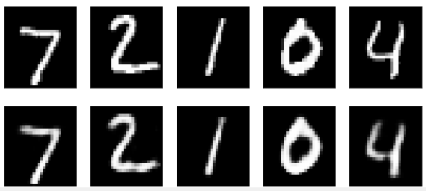
Editor - D:\Practical\DL\Deep_Autoencoder.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  6 20:33:17 2022

@author: Jay Modi
Sapid: 43004200015
"""
#Import neccessary libaries
import keras
from keras import layers
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
# This is the size of encoded representations
encoding_dim=32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
#this is our input image
input_img=keras.Input(shape=(784,))
#"encoded" is the encoded representation of the input
encoded=layers.Dense(128, activation='relu')(input_img)
encoded = layers.Dense(64, activation='relu')(encoded)
encoded = layers.Dense(32, activation='relu')(encoded)
#"decoded" is the lossy reconstruction of the input
decoded = layers.Dense(64, activation='relu')(encoded)
decoded = layers.Dense(128, activation='relu')(decoded)
decoded=layers.Dense(784, activation='sigmoid')(encoded)
#creating autoencoder model
autoencoder=keras.Model(input_img,decoded)
#create the encoder model
encoder=keras.Model(input_img,encoded)
encoded_input=keras.Input(shape=(encoding_dim,))
#Retrive the last layer of the autoencoder model
decoder_layer=autoencoder.layers[-1]
#create the decoder model
decoder=keras.Model(encoded_input,decoder_layer(encoded_input))
#Compiling the autoencoder model with binary crossentropy loss, and the Adam optimizer
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
#scale and make train and test dataset
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=X_train.reshape((len(X_train),np.prod(X_train.shape[1:])))
X_test=X_test.reshape((len(X_test),np.prod(X_test.shape[1:])))
```

IPython console

Console 1/A

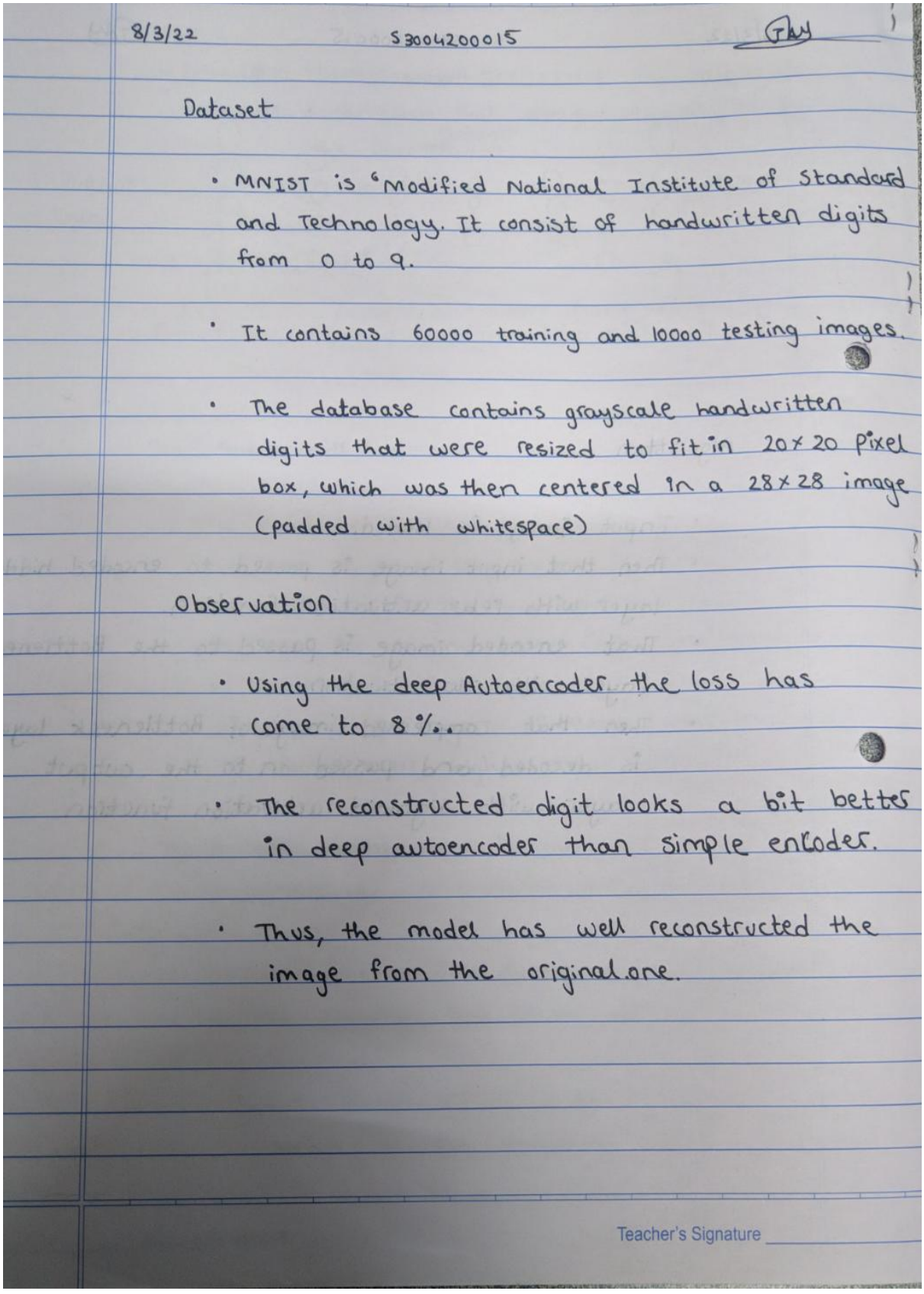
```
235/235 [==============================] - 2s 6ms/step - loss: 0.0871 - val_loss: 0.0862
Epoch 37/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0869 - val_loss: 0.0861
Epoch 38/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0868 - val_loss: 0.0860
Epoch 39/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0868 - val_loss: 0.0860
Epoch 40/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0867 - val_loss: 0.0859
Epoch 41/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0866 - val_loss: 0.0856
Epoch 42/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0865 - val_loss: 0.0857
Epoch 43/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0864 - val_loss: 0.0855
Epoch 44/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0864 - val_loss: 0.0854
Epoch 45/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0863 - val_loss: 0.0857
Epoch 46/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0863 - val_loss: 0.0855
Epoch 47/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0862 - val_loss: 0.0853
Epoch 48/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0861 - val_loss: 0.0855
Epoch 49/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0861 - val_loss: 0.0852
Epoch 50/50
235/235 [==============================] - 2s 7ms/step - loss: 0.0860 - val_loss: 0.0853
```



Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 60 Column: 1 Memory: 89 %

12:24 AM 07-Mar-22

Observation



8/3/22 53004200015

Dataset

- MNIST is 'Modified National Institute of Standard and Technology. It consist of handwritten digits from 0 to 9.
- It contains 60000 training and 10000 testing images.
- The database contains grayscale handwritten digits that were resized to fit in 20×20 pixel box, which was then centered in a 28×28 image (padded with whitespace)

Observation

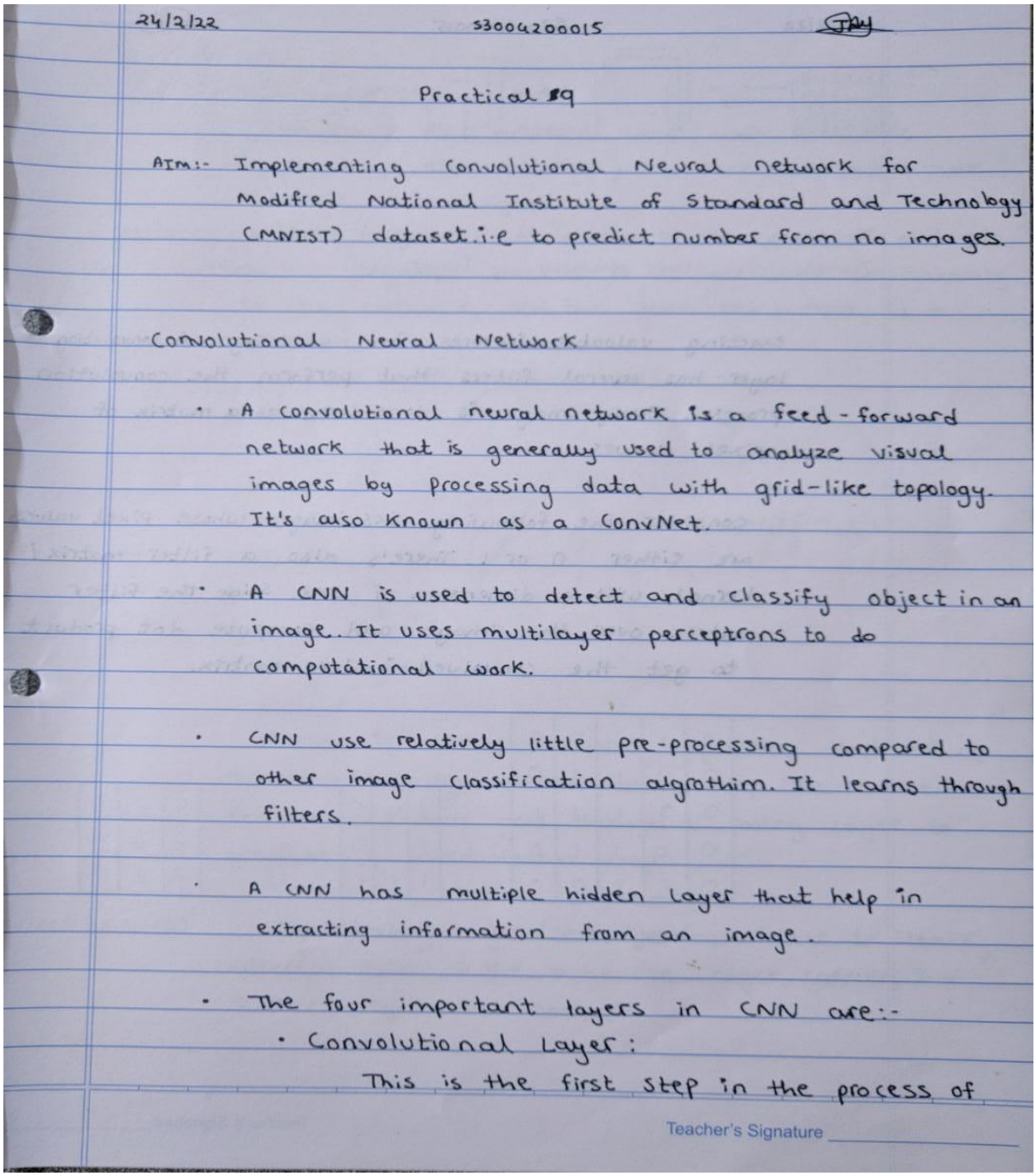
- Using the deep Autoencoder the loss has come to 8%..
- The reconstructed digit looks a bit better in deep autoencoder than simple encoder.
- Thus, the model has well reconstructed the image from the original one.

Teacher's Signature

Practical 9

Implementing convolutional neural network to predict numbers from number images.

Convolutional neural network



24/2/22 53004200015

Practical 9

AIM:- Implementing Convolutional Neural network for Modified National Institute of Standard and Technology (MNIST) dataset i.e to predict number from no images.

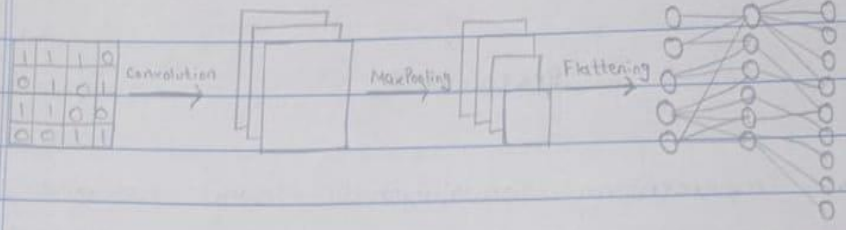
Convolutional Neural Network

- A convolutional neural network is a feed-forward network that is generally used to analyze visual images by processing data with grid-like topology. It's also known as a ConvNet.
- A CNN is used to detect and classify object in an image. It uses multilayer perceptrons to do computational work.
- CNN use relatively little pre-processing compared to other image classification algorithm. It learns through filters.
- A CNN has multiple hidden layer that help in extracting information from an image.
- The four important layers in CNN are:-
  - Convolutional Layer:

    This is the first step in the process of

Teacher's Signature

24/2/22 53004200015 JAY



Input Image | Convolutional Layer with ReLU | Pooling Layer | Fully connected Network layer

exacting valuable features from an image. A convolution layer has several filters that perform the convolution process. Every image is considered as a matrix of pixel values.

Consider the following 5x5 image whose pixel values are either 0 or 1. There's also a filter matrix / kernal with a dimension of 3x3. Slide the filter matrix over the image and compute dot product to get the convolved feature matrix.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image Pixel

1	0	1
0	1	0
1	0	1

kernel/Filter

4	3	4
2	4	3
2	3	4

Convolved Feature

Teacher's Signature _____

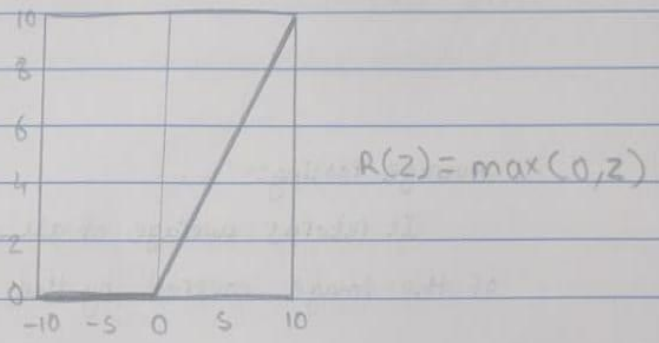
24/2/22 53004200015

- ReLU Layer:

It stands for rectified linear unit. Once the feature maps are extracted, the next step is to move them to ReLU layer.

ReLU performs an element-wise operation and sets all the negative pixel at 0. It introduces non-linearity to the network, and the generated output is a rectified feature map.

Below is the graph of ReLU function.



- Pooling Layer

Pooling is a down-sampling operations that reduce the dimensionality of the feature map. The rectified feature map now goes through a pooling layer to generate a pooled feature map.

The pooling layer uses various filter to identify different parts of the image like edges, corner, body feathers, eyes and beak.

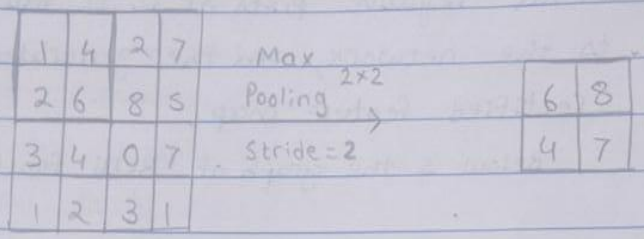
Teacher's Signature _____

24/2/22 53004200015

There are two types of Pooling

Max Pooling:-

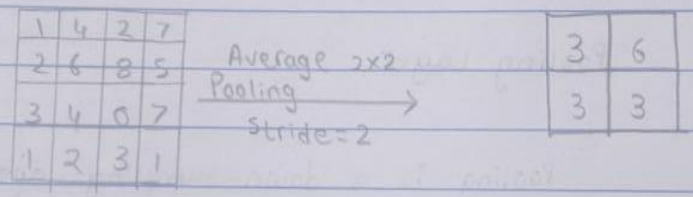
It returns the maximum value from the portion of the image covered by kernel.



Rectified Feature map Pooled Feature map

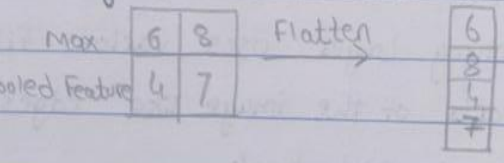
Average Pooling:-

It returns average of all values from the portion of the image covered by the kernel.



- Flatten Layer

The next step in the process is called Flattening. It is used to convert all resultant 2-D array to single long continuous liner vector.



- The flattened matrix is fed as input to the fully connected layer to classify the image.

Teacher's Signature

Algorithm and Flowchart

24/2/22 53004200015 Thu

Algorithm

Step 1:- Input Image is loaded

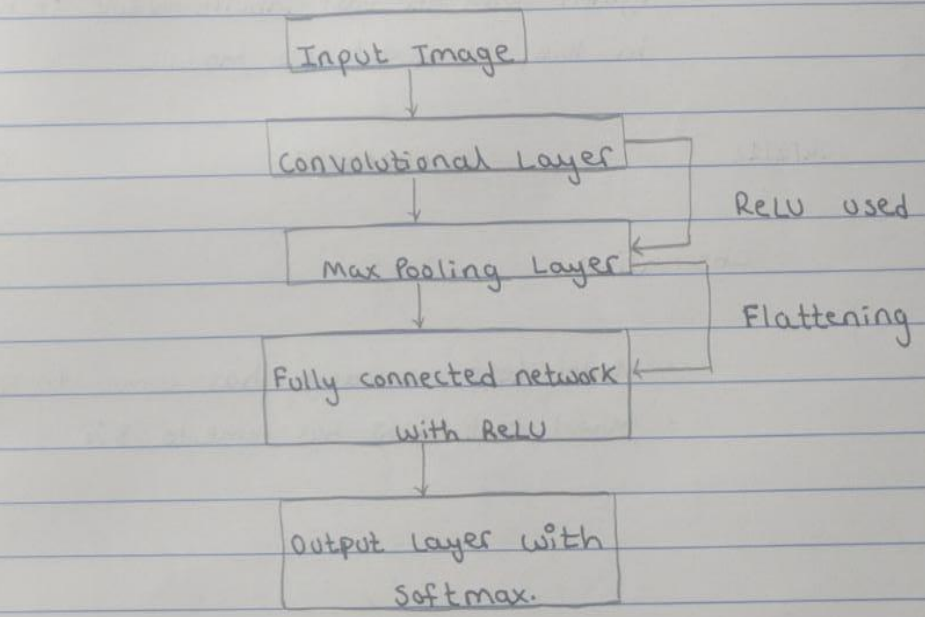
Step 2:- The pixel from the image are fed to convolutional layer that performs convolutional operation.

Step 3:- The convolved map is applied to a ReLU function to generate a rectified feature map.

Step 4:- Max Pooling is applied to the recified feature map

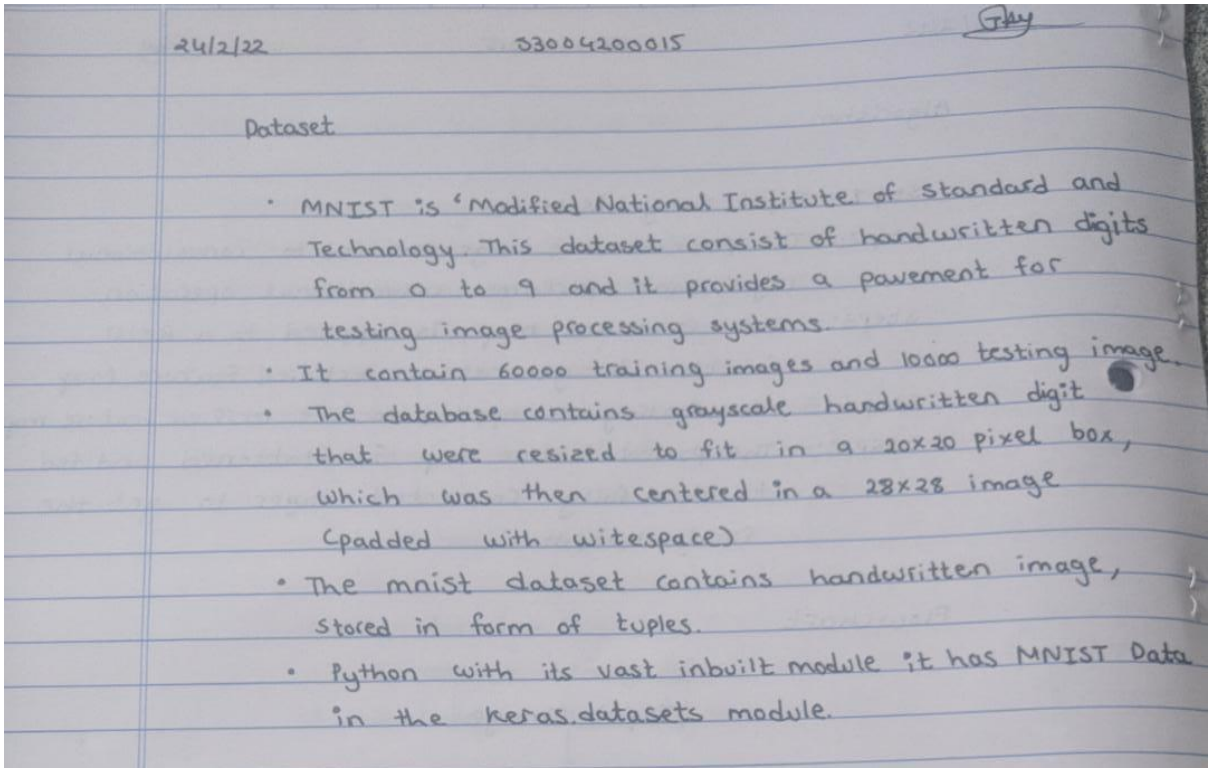
STEP 5:- The pooled feature map is flattened and fed to the fully connected layer to get the final output.

Flowchart



Teacher's Signature _____

Dataset



24/2/22 53004200015

Dataset

- MNIST is 'Modified National Institute of standard and Technology. This dataset consist of handwritten digits from 0 to 9 and it provides a pavement for testing image processing systems.
- It contain 60000 training images and 10000 testing image.
- The database contains grayscale handwritten digit that were resized to fit in a 20×20 pixel box, which was then centered in a 28×28 image (padded with witespace)
- The mnist dataset contains handwritten image, stored in form of tuples.
- Python with its vast inbuilt module it has MNIST Data in the keras.datasets module.

Code

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 20 01:47:01 2022
@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
from keras.datasets import mnist
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten
import matplotlib.pyplot as plt
#download mnist data and split into train and test sets
(X_train,Y_train),(X_test,Y_test)=mnist.load_data()
#plot the first image in the dataset
plt.imshow(X_train[0])
plt.show()
print(X_train[0].shape)
# reshape dataset to have a single channel
X_train=X_train.reshape(60000,28,28,1)
X_test=X_test.reshape(10000,28,28,1)
# one hot encode target values
Y_train=to_categorical(Y_train)
Y_test=to_categorical(Y_test)
Y_train[0]
print(Y_train[0])
```

```
#Creating a CNN Model
model=Sequential()
#Creating Convolutional Layer with relu function
model.add(Conv2D(64,kernel_size=3,activation='relu',input_shape=(28,28,1)))
#Adding MaxPooling Layer of 2*2
model.add(MaxPooling2D((2, 2)))
#Flattening layer which acts as input to fully connected network
model.add(Flatten())
#Fully connected network with relu function
model.add(Dense(100, activation='relu'))
#Output Layer with softmax function
model.add(Dense(10,activation='softmax'))
#Compiling Model with Loss function as categorical_crossentropy as its multiclass classification
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
#fitting the model
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=3)
#Evaluate the results
score = model.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

Editor - D:\Practical\DL\CNN.py

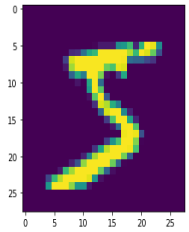
Binary_classification.py | Multiclass_classification.py | Classification_probablity.py | Regression.py | CNN.py

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 20 01:47:01 2022

@author: Jay Modi
Sapid: 53004200015
"""
#importing all the necessary libraries
from keras.datasets import mnist
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten
import matplotlib.pyplot as plt
#download mnist data and split into train and test sets
(X_train,Y_train),(X_test,Y_test)=mnist.load_data()
#plot the first image in the dataset
plt.imshow(X_train[0])
plt.show()
print(X_train[0].shape)
# reshape dataset to have a single channel
X_train=X_train.reshape(60000,28,28,1)
X_test=X_test.reshape(10000,28,28,1)
# one hot encode target values
Y_train=to_categorical(Y_train)
Y_test=to_categorical(Y_test)
Y_train[0]
print(Y_train[0])
#Creating a CNN Model
model=Sequential()
#Creating Convolutional Layer with relu function
model.add(Conv2D(64,kernel_size=3,activation='relu',input_shape=(28,28,1)))
#Adding MaxPooling Layer of 2*2
model.add(MaxPooling2D((2, 2)))
#Flattening layer which acts as input to fully connected network
model.add(Flatten())
#Fully connected network with relu function
model.add(Dense(100, activation='relu'))
#Output Layer with softmax function
model.add(Dense(10,activation='softmax'))
#Compiling Model with Loss function as categorical_crossentropy as its multiclass classification
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
#fitting the model
```

IPython console

Console 1/A

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.6.1 -- An enhanced Interactive Python.

In [1]: runfile('D:/Practical/DL/CNN.py', wdir='D:/Practical/DL')
```

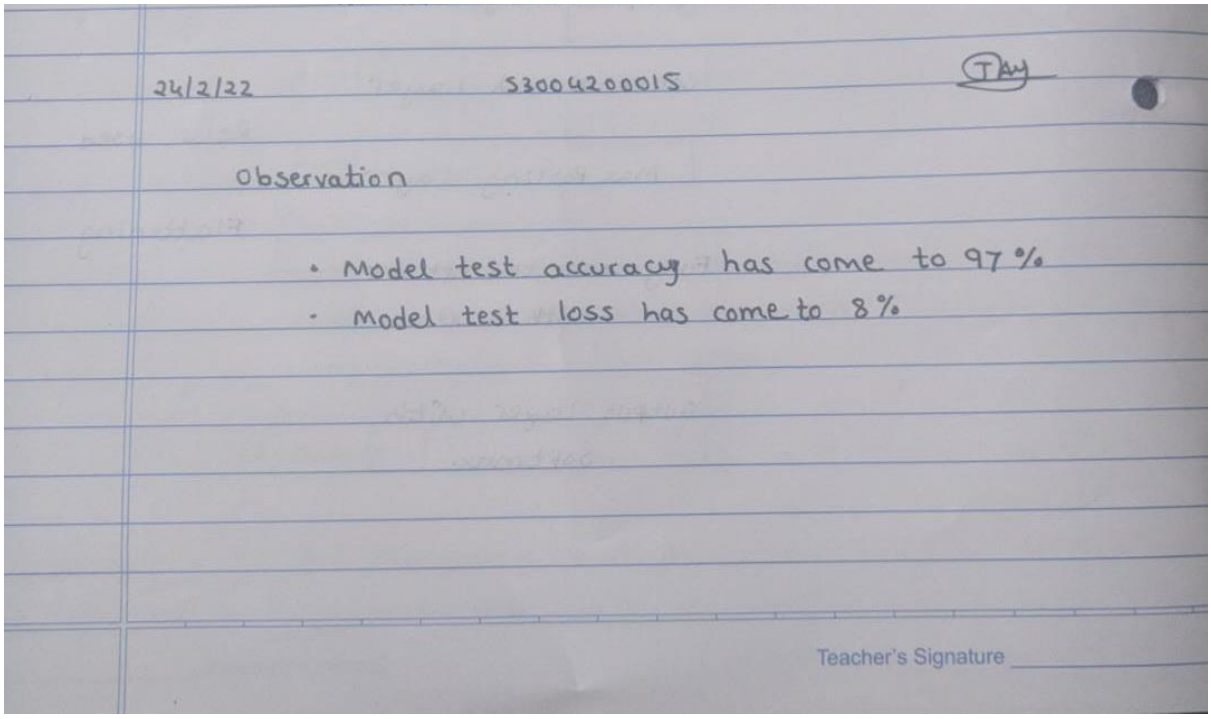



```
(28, 28)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Epoch 1/3
1875/1875 [==============================] - 36s 19ms/step - loss: 0.6342 -
accuracy: 0.9390 - val_loss: 0.1015 - val_accuracy: 0.9708
Epoch 2/3
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0802 -
accuracy: 0.9756 - val_loss: 0.1097 - val_accuracy: 0.9665
Epoch 3/3
1875/1875 [==============================] - 35s 19ms/step - loss: 0.0594 -
accuracy: 0.9818 - val_loss: 0.0870 - val_accuracy: 0.9783
313/313 [==============================] - 2s 6ms/step - loss: 0.0870 -
accuracy: 0.9783
Test loss: 0.08699119091033936
Test accuracy: 0.9782999753952026

In [2]:
```

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 42 | Column: 1 | Memory: 94 %

5:50 PM
20-Feb-22

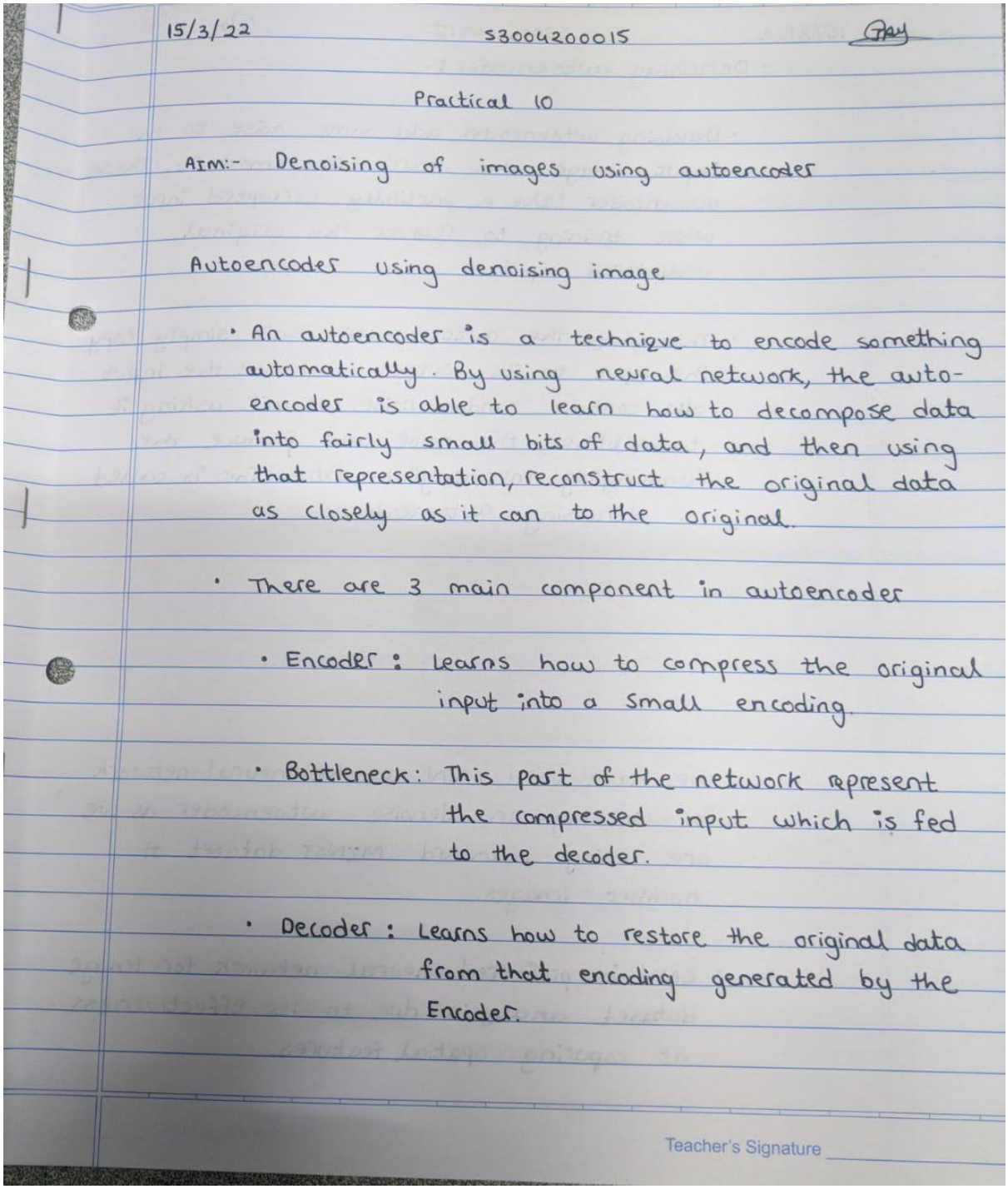
Observation



Practical 10

Denoising of images using autoencoder.

Autoencoder using denoising image



15/3/22 53004200015

Practical 10

Aim:- Denoising of images using autoencoder

Autoencoder using denoising image

- An autoencoder is a technique to encode something automatically. By using neural network, the auto-encoder is able to learn how to decompose data into fairly small bits of data, and then using that representation, reconstruct the original data as closely as it can to the original.

- There are 3 main component in autoencoder

  - Encoder : Learns how to compress the original input into a small encoding.

  - Bottleneck: This part of the network represent the compressed input which is fed to the decoder.

  - Decoder : Learns how to restore the original data from that encoding generated by the Encoder.

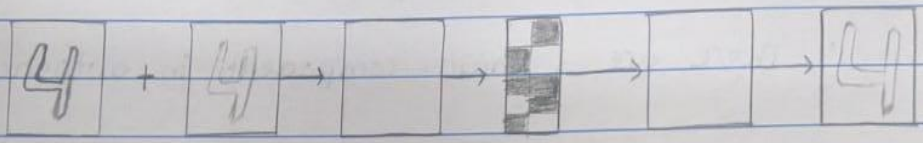
Teacher's Signature

15/3/22 53004200015

- Denoising autoencoders :-

- Denoising autoencoders add some noise to the input image and learn to remove it. These autoencoders take a partially corrupted input while training to recover the original undistored input.

- This way the autoencoder can't simply copy the input to its output because the input also contain random noise. We are asking it to subtract the noise and produce the underlying meaningful data. This is called a denoising autoencoder.



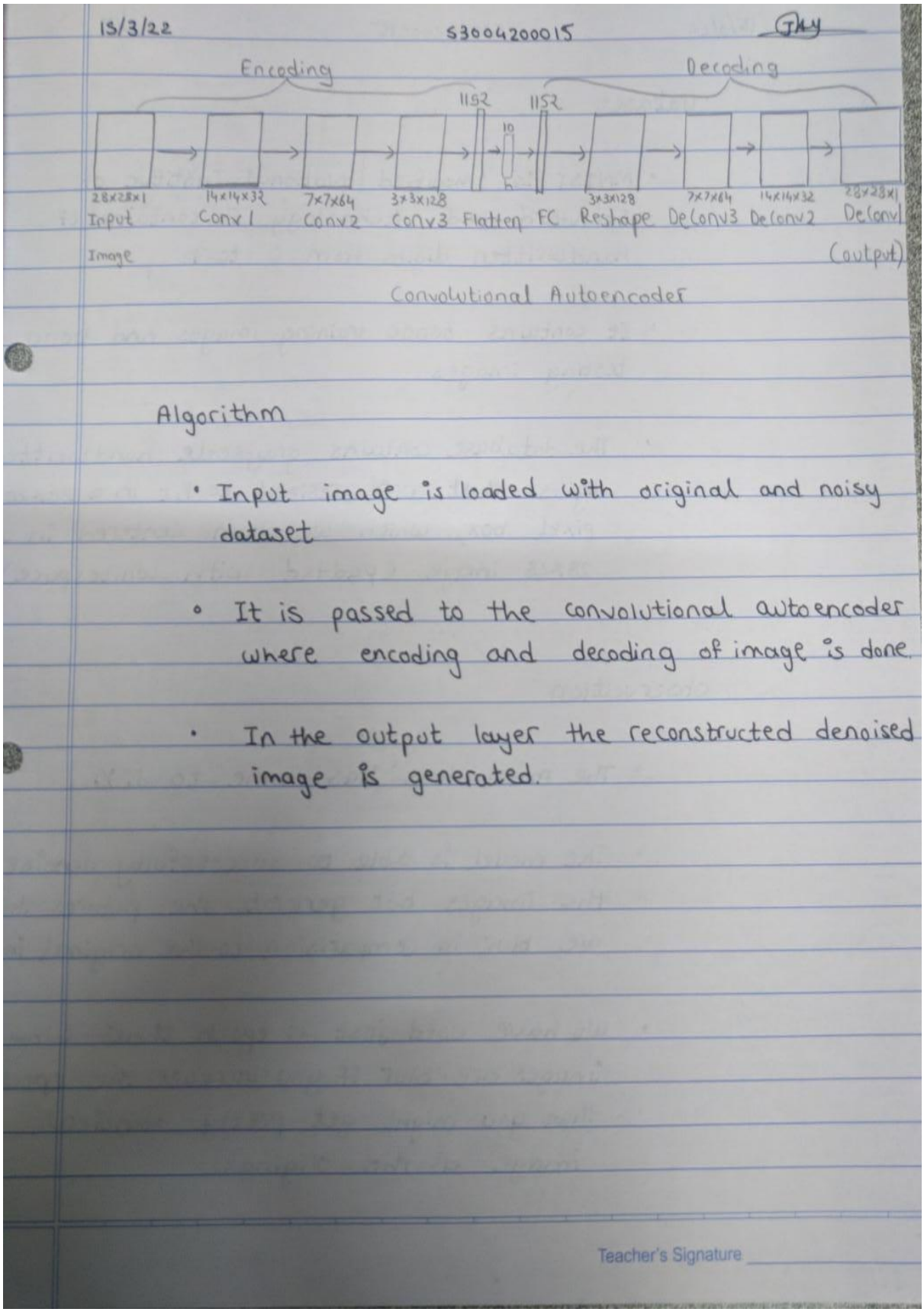
Original image | Noisy Image | Encoder | Code | Decoder | Output image

- we have used CNN as the neural network for working on denoise autoencoder as we are using standard MINST dataset of number images.

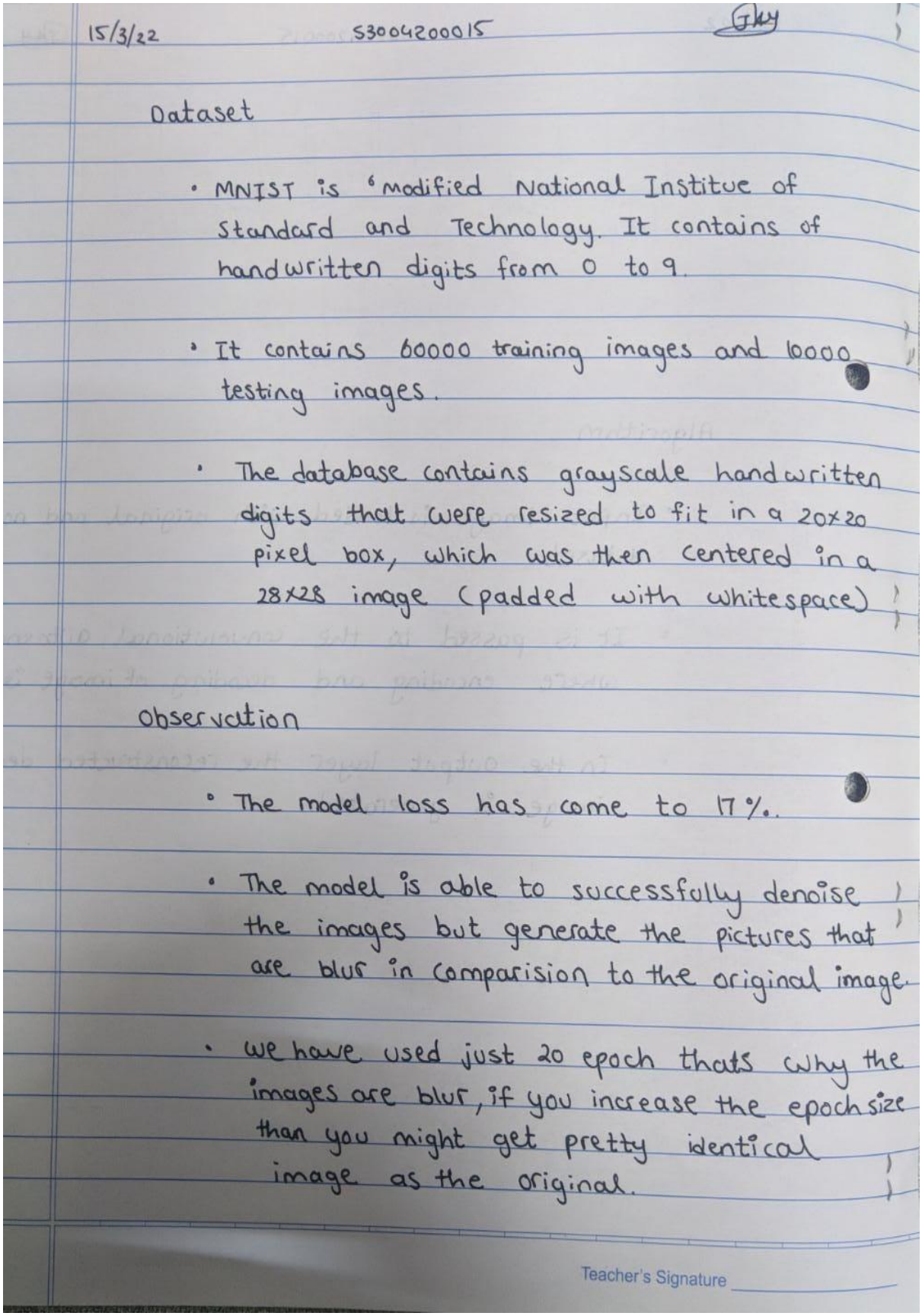
- CNN is preferred neural network for image dataset analysis due to its effectiveness at capuring spatial features.

Teacher's Signature

Algorithm


15/3/22
53004200015
JKY
Encoding
Decoding
1152
1152
10
28x28x1
Input Image
14x14x32
Conv1
7x7x64
Conv2
3x3x128
Conv3
Flatten
FC
3x3x128
Reshape
7x7x64
DeConv3
14x14x32
DeConv2
28x28x1
DeConv1
(output)
Convolutional Autoencoder
Algorithm
• Input image is loaded with original and noisy dataset
• It is passed to the convolutional autoencoder where encoding and decoding of image is done.
• In the output layer the reconstructed denoised image is generated.
Teacher's Signature

Dataset



15/3/22 53004200015

Dataset

- MNIST is 'Modified National Institue of Standard and Technology. It contains of handwritten digits from 0 to 9.
- It contains 60000 training images and 10000 testing images.
- The database contains grayscale handwritten digits that were resized to fit in a 20×20 pixel box, which was then centered in a 28×28 image (padded with whitespace)

Observation

- The model loss has come to 17%.
- The model is able to successfully denoise the images but generate the pictures that are blur in comparision to the original image.
- we have used just 20 epoch thats why the images are blur, if you increase the epoch size than you might get pretty identical image as the original.

Teacher's Signature

Code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  7 00:42:04 2022
@author: Jay Modi
Sapid: 53004200015
"""
#Import neccessary libaries
import keras
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
from keras.layers import Input, Conv2D, MaxPool2D, UpSampling2D
#scale and make train and test dataset
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=np.reshape(X_train,(len(X_train),28,28,1))
X_test=np.reshape(X_test,(len(X_test),28,28,1))
print(X_train.shape)
print(X_test.shape)
#Adding Noisy data
noise_factor = 0.5
X_train_noisy = X_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)
X_test_noisy = X_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)
# to make values in the range of 0 to 1,
```

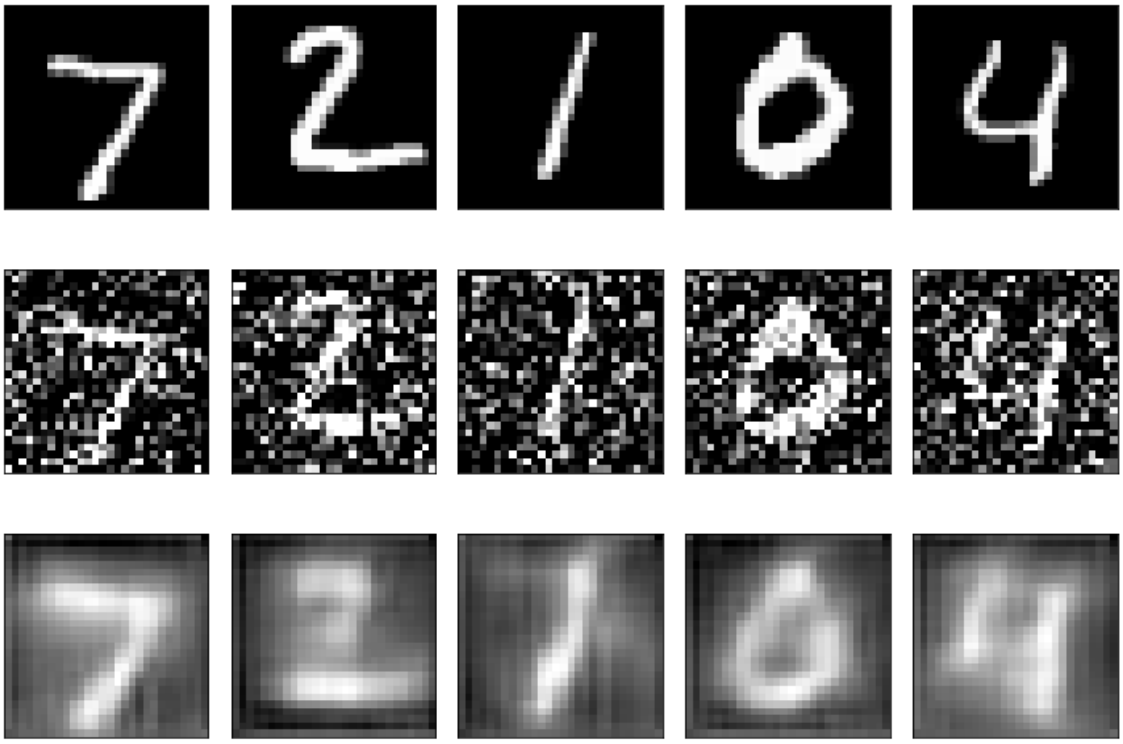
```
#if values < 0 then they will be equal to 0 and
#if values > 1 then they will be equal to 1.
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
input_img = Input(shape=(28, 28, 1))
# encoding architecture
x1 = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x1 = MaxPool2D( (2, 2), padding='same')(x1)
x2 = Conv2D(32, (3, 3), activation='relu', padding='same')(x1)
x2 = MaxPool2D( (2, 2), padding='same')(x2)
x3 = Conv2D(16, (3, 3), activation='relu', padding='same')(x2)
encoded    = MaxPool2D( (2, 2), padding='same')(x3)
# decoding architecture
x3 = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x3 = UpSampling2D((2, 2))(x3)
x2 = Conv2D(32, (3, 3), activation='relu', padding='same')(x3)
x2 = UpSampling2D((2, 2))(x2)
x1 = Conv2D(64, (3, 3), activation='relu')(x2)
x1 = UpSampling2D((2, 2))(x1)
decoded   = Conv2D(1, (3, 3), padding='same')(x1)
#creating autoencoder model
autoencoder=keras.Model(input_img,decoded)
#Compiling the autoencoder model with binary crossentropy loss, and the
Adam optimizer
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
#train autoencoder with training dataset with 20 epochs and using minibatch
approach
```

```
autoencoder.fit(X_train_noisy,X_train,epochs=20,batch_size=256,shuffle=True
,validation_data=(X_test_noisy,X_test))
# to predict the reconstructed images for the original images...
pred = autoencoder.predict(X_test_noisy)
plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.xticks([]) # to remove x-axis  the [] empty list indicates this
    plt.yticks([]) # to remove y-axis
    plt.grid(False) # to remove grid
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray') #display the image
plt.tight_layout() # to have a proper space in the subplots
plt.show()

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.xticks([]) # to remove x-axis  the [] empty list indicates this
    plt.yticks([]) # to remove y-axis
    plt.grid(False) # to remove grid
    plt.imshow(X_test_noisy[i].reshape(28, 28), cmap='gray') #display the image
plt.tight_layout() # to have a proper space in the subplots
plt.show()

# to visualize reconstructed images(output of autoencoder)
plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(1, 5, i+1)
```

```
    plt.xticks([]) # to remove x-axis  the [] empty list indicates this
    plt.yticks([]) # to remove y-axis
    plt.grid(False) # to remove grid
    plt.imshow(pred[i].reshape(28, 28), cmap='gray') #display the image
plt.tight_layout() # to have a proper space in the subplots
plt.show()
```

53004200015

Output



Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

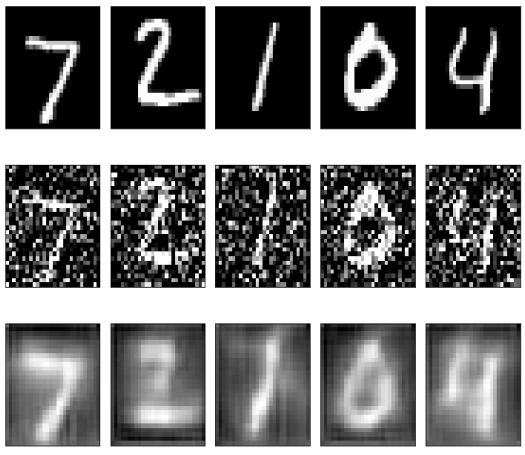
Editor - D:\Practical\DL\Denois_autoencoder.py

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  7 00:42:04 2022

@author: Jay Modi
Sapid: 53004200015
"""
#Import neccessary libaries
import keras
from keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
from keras.layers import Input, Conv2D, MaxPool2D, UpSampling2D
#scale and make train and test dataset
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=np.reshape(X_train,(len(X_train),28,28,1))
X_test=np.reshape(X_test,(len(X_test),28,28,1))
print(X_train.shape)
print(X_test.shape)
#Adding Noisy data
noise_factor = 0.5
X_train_noisy = X_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)
X_test_noisy = X_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)
# to make values in the range of 0 to 1,
#if values < 0 then they will be equal to 0 and
#if values > 1 then they will be equal to 1.
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
input_img = Input(shape=(28, 28, 1))
# encoding architecture
x1 = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x1 = MaxPool2D( (2, 2), padding='same')(x1)
x2 = Conv2D(32, (3, 3), activation='relu', padding='same')(x1)
x2 = MaxPool2D( (2, 2), padding='same')(x2)
x3 = Conv2D(16, (3, 3), activation='relu', padding='same')(x2)
encoded    = MaxPool2D( (2, 2), padding='same')(x3)
# decoding architecture
x3 = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x3 = UpSampling2D((2, 2))(x3)
```

IPython console

Console 1/A

```
Epoch 13/20
235/235 [==============================] - 136s 580ms/step - loss: 0.2355 - val_loss: 0.2201
Epoch 14/20
235/235 [==============================] - 134s 572ms/step - loss: 0.2652 - val_loss: 0.2337
Epoch 15/20
235/235 [==============================] - 134s 570ms/step - loss: 0.2227 - val_loss: 0.2078
Epoch 16/20
235/235 [==============================] - 134s 570ms/step - loss: 0.2050 - val_loss: 0.1958
Epoch 17/20
235/235 [==============================] - 134s 571ms/step - loss: 0.1995 - val_loss: 0.2117
Epoch 18/20
235/235 [==============================] - 133s 565ms/step - loss: 0.1927 - val_loss: 0.1885
Epoch 19/20
235/235 [==============================] - 134s 569ms/step - loss: 0.1883 - val_loss: 0.1811
Epoch 20/20
235/235 [==============================] - 141s 600ms/step - loss: 0.1789 - val_loss: 0.1791
```

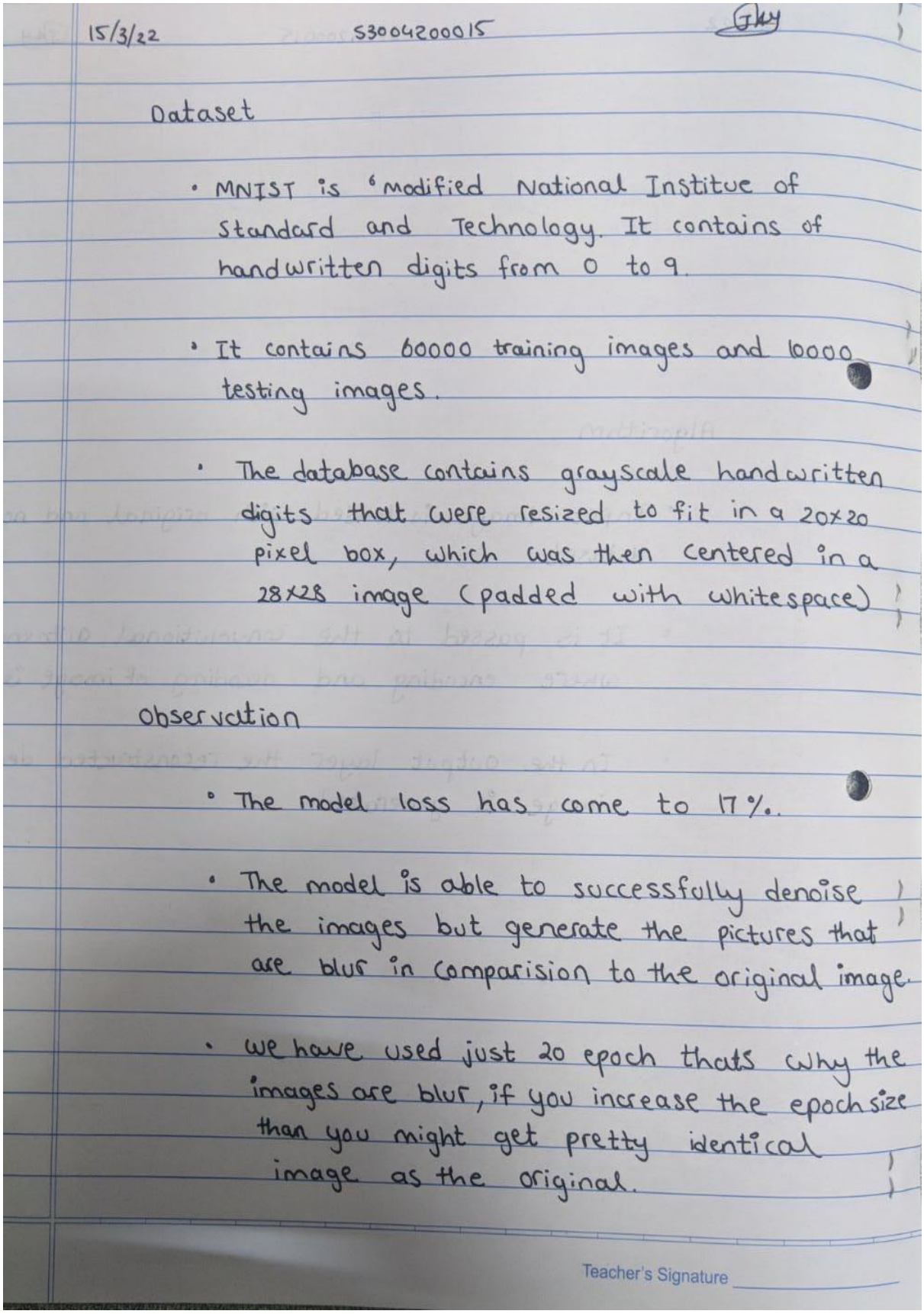


```
In [3]:
```

Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 53 Column: 63 Memory: 90 %

5:01 PM
08-Mar-22

Observation



15/3/22 53004200015

Dataset

- MNIST is 'Modified National Institue of Standard and Technology. It contains of handwritten digits from 0 to 9.
- It contains 60000 training images and 10000 testing images.
- The database contains grayscale handwritten digits that were resized to fit in a 20×20 pixel box, which was then centered in a 28×28 image (padded with whitespace)

Observation

- The model loss has come to 17%.
- The model is able to successfully denoise the images but generate the pictures that are blur in comparision to the original image.
- We have used just 20 epoch thats why the images are blur, if you increase the epochsize than you might get pretty identical image as the original.

Teacher's Signature