

**DEPARTMENT
OF
COMPUTER ENGINEERING**

**LAB MANUAL
Lab Practices V**

**Course: Deep Learning
Final Year Semester – II**

Prepared by : Dr Amol Dhakne



Institute Vision

To strive for excellence by providing quality technical education and facilitate research for the welfare of society

Institute Mission

1. To educate students with strong fundamentals by providing conducive environment
2. To inculcate research with creativity & innovation
3. To strengthen leadership, team work, professional & communication skills and ethical standards
4. To promote Industry Institute collaboration & prepare students for life long learning in context of technological change.

Department Vision

To produce quality computer professionals and fostering research aptitude for dispensing service to society

Department Mission

1. To promote growth of an individual by imparting comprehensive knowledge of tools and technologies.
2. To facilitate research and innovation by engaging faculty and students in research activities.
3. To enrich industry-institute interaction in order to provide a platform to know industry demands and motivation for self-employment.
4. To bring forth a conducive environment to enhance soft skills and professional skills to cater needs of society

Program Specific Outcomes

1. Professional Skills: The ability to comprehend, analyze and develop software and hardware systems and applications through research, in varying domains.
2. Problem-Solving Skills: The ability to apply standard paradigms and strategies in software project development using open-ended programming environments to deliver a quality product.
3. Successful Career and Entrepreneurship: Adaptation of modern practical and systematic approaches in creating innovative solutions for a successful career, entrepreneurship, and a zest for higher studies.

Course Objective

1. To understand and implement searching and sorting algorithms.
2. To learn the fundamentals of GPU Computing in the CUDA environment.
3. To illustrate the concepts of Artificial Intelligence/Machine Learning(AI/ML).
4. To understand Hardware acceleration.
5. To implement different deep learning models.

Course Outcome

After completion of the course, students will be able to :

CO1: Analyze and measure performance of sequential and parallel algorithms.

CO2: Design and Implement solutions for multicore/Distributed/parallel environment.

CO3: Identify and apply the suitable algorithms to solve AI/ML problems.

CO4: Apply the technique of Deep Neural network for implementing Linear regression and classification.

CO5: Apply the technique of Convolution (CNN) for implementing Deep Learning models.

CO6: Design and develop Recurrent Neural Network (RNN) for prediction

@The CO-PO Mapping
Matrix

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	1	-	1	1	-	2	1	-	-	-	-	-
CO2	1	2	1	-	-	1	-	-	-	-	-	1
CO3	-	1	1	1	1	1	-	-	-	-	-	-
CO4	3	3	3	-	3	-	-	-	-	-	-	-
CO5	3	3	3	3	3	-	-	-	-	-	-	-
CO6	3	3	3	3	3	-	-	-	-	-	-	-
CO7	3	3	3	3	3		-	-	-	-	-	-

Guidelines for Students

The laboratory assignments are to be submitted by student in the form of journal. Journal may home Home Faculty of Engineering Savitribai Phule Pune University Syllabus for Fourth Year of Computer Engineering ` #106/128 consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, Algorithm/Database design, test cases, conclusion/analysis). Program codes with sample output of all performed assignments are to be submitted as softcopy.

Guidelines for Laboratory /Term Work Assessment

Continuous assessment of laboratory work is to be done based on overall performance and lab assignments performance of student. Each lab assignment assessment will assign grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each lab assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness reserving weightage for successful mini-project completion and related documentation.

Table of Contents

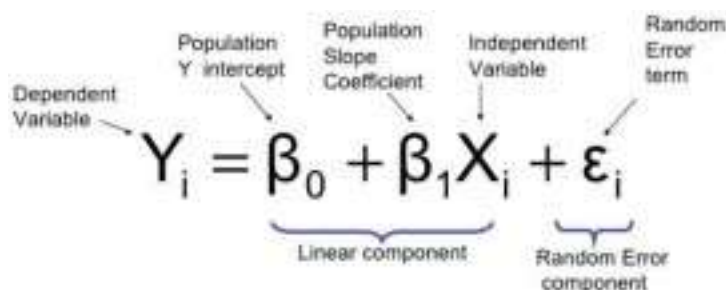
Sr.No		Title of the Experiment	Page No
High Performance Computing			
1			
2			
3			
4			
5			
Deep Learning			
1			
2			
3			
4			
5			

Assignment No. 1

- **Title:** Linear regression by using Deep Neural network
- **Date of Completion:**
- **Objective:** Study and understand neural network by using Linear Regression for predicting house prices.
- **Problem Statement:** Implement Boston housing price prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.
- **Theory:**

Linear Regression

Linear Regression is a supervised learning technique that involves learning the relationship between the features and the target. The target values are continuous, which means that the values can take any values between an interval. Use-cases of regression include stock market price prediction, house price prediction, sales prediction, and etc.



The diagram illustrates the Linear Regression equation:
$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$
 The components are labeled as follows:

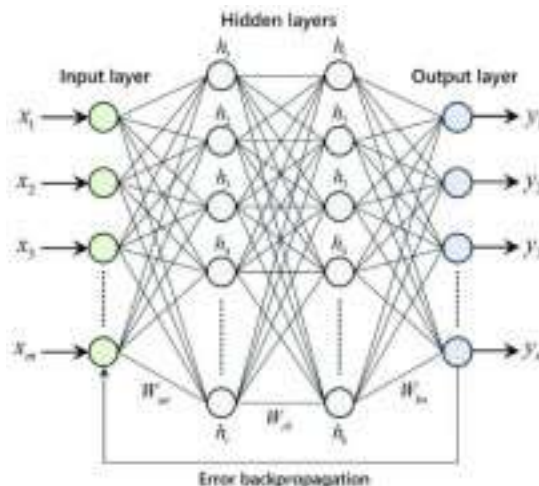
- Dependent Variable:** Y_i
- Population Y intercept:** β_0
- Population Slope Coefficient:** β_1
- Independent Variable:** X_i
- Random Error term:** ϵ_i

The equation is also grouped into two main components:

- Linear component:** $\beta_0 + \beta_1 X_i$
- Random Error component:** ϵ_i

Neural network

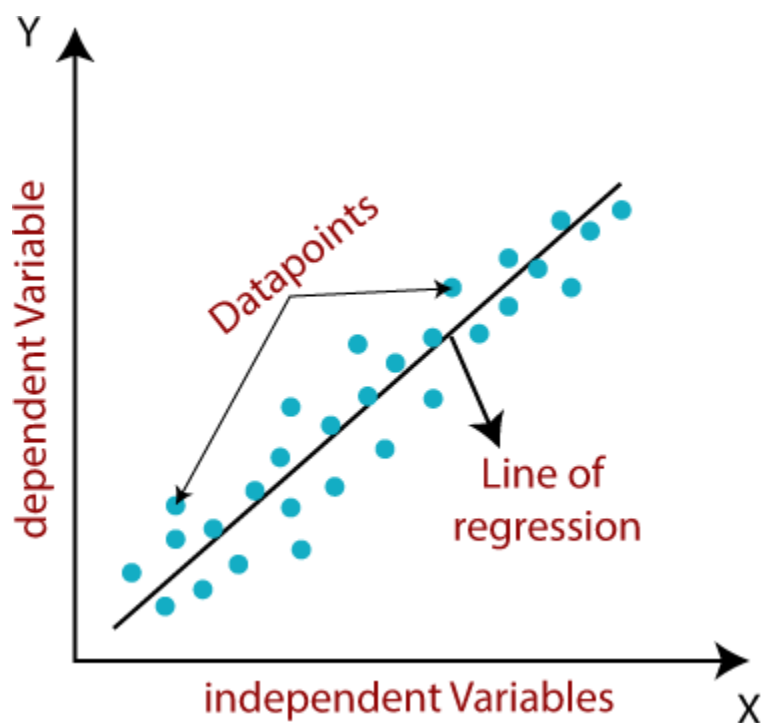
Neural networks are formed when multiple neural layers combine with each other to give out a network, or we can say that there are some layers whose outputs are inputs for other layers.



The purpose of using Artificial Neural Networks for Regression over Linear Regression is that the linear regression can only learn the linear relationship between the features and target and therefore cannot learn the complex non-linear relationship. In order to learn the complex non-linear relationship between the features and target, we are in need of other techniques. One of those techniques is to use Artificial Neural Networks. Artificial Neural Networks have the ability to learn the complex relationship between the features and target due to the presence of activation function in each layer. Artificial Neural Networks are one of the deep learning algorithms that simulate the workings of neurons in the human brain.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$$y = a_0 + a_1x + \varepsilon$$

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a_0 = intercept of the line (Gives an additional degree of freedom)

a_1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

- Simple Linear Regression:

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- Multiple Linear regression:

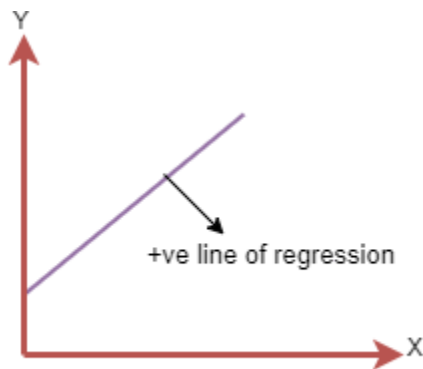
If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

A linear line showing the relationship between the dependent and independent variables is called a regression line. A regression line can show two types of relationship:

- Positive Linear Relationship:

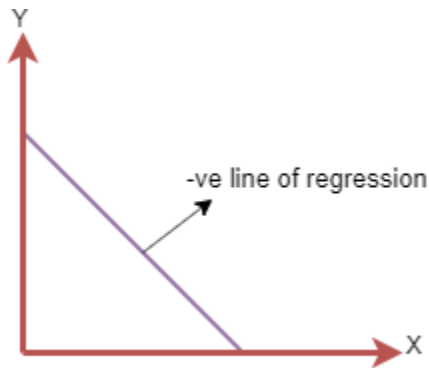
If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y = a_0 + a_1X$

- Negative Linear Relationship:

If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y = -a_0 + a_1X$

- **Dataset :** Boston Housing Dataset
- **References/ Available link :**
 1. <https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155>
 2. <https://www.kaggle.com/code/shreayan98c/boston-house-price-prediction>
- **Questions and answer :**
 - Q1: What is Linear Regression?
 - Q2: How would you detect Overfitting in Linear Models?
 - Q3: What are types of Linear Regression?
 - Q4: How can you check if the Regression model fits the data well?
 - Q5: Define Linear Regression and its structure ?
 - Q6: What is the difference between Mean Absolute Error (MAE) vs Mean Squared Error (MSE)?
 - Q7: What's the difference between Covariance and Correlation?
 - Q8: Explain the intuition behind Gradient Descent algorithm ?
 - Q9: How is the Error calculated in a Linear Regression model?
 - Q10: Why use Root Mean Squared Error (RMSE) instead of Mean Absolute Error (MAE)?
- **Mcqs:**
 1. The best fit line method for data in Linear Regression?
 - A) Least Square Error
 - B) Maximum Likelihood
 - C) Logarithmic Loss
 - D) Both A and B

2. Which of the following metrics can be used to evaluate a model with a continuous output variable?

- A) Precision-Recall Curve
- B) Accuracy
- C) Logloss
- D) Mean-Squared-Error

3. Which statement is true about outliers in Linear regression?

- A) Linear regression model is not sensitive to outliers
- B) Linear regression model is sensitive to outliers
- C) Can't say
- D) None of these

- **Conclusion :**

We have successfully solved Boston house price prediction problem by Linear Regression using deep neural networks.

In [1]:

```
# Importing the pandas for data processing and numpy for numerical computing
import numpy as np
import pandas as pd
```

In [2]:

```
# Importing the Boston Housing dataset from the sklearn
from sklearn.datasets import load_boston
boston = load_boston()
```

In [3]:

```
# Converting the data into pandas dataframe
data = pd.DataFrame(boston.data)
```

In [4]:

```
data.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [5]:

```
# Adding the feature names to the dataframe
data.columns = boston.feature_names
```

In [6]:

```
# Adding the target variable to the dataset
data['PRICE'] = boston.target
```

In [7]:

```
# Looking at the data with names and target variable  
data.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [8]:

```
# Checking the null values in the dataset  
data.isnull().sum()
```

Out[8]:

```
CRIM      0  
ZN        0  
INDUS     0  
CHAS      0  
NOX       0  
RM        0  
AGE       0  
DIS       0  
RAD       0  
TAX       0  
PTRATIO   0  
B         0  
LSTAT     0  
PRICE     0  
dtype: int64
```

In [9]:

```
# Checking the statistics of the data
data.describe()
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

In [10]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   CRIM        506 non-null    float64
 1   ZN          506 non-null    float64
 2   INDUS       506 non-null    float64
 3   CHAS        506 non-null    float64
 4   NOX         506 non-null    float64
 5   RM          506 non-null    float64
 6   AGE         506 non-null    float64
 7   DIS         506 non-null    float64
 8   RAD         506 non-null    float64
 9   TAX         506 non-null    float64
10  PTRATIO     506 non-null    float64
11  B           506 non-null    float64
12  LSTAT       506 non-null    float64
13  PRICE      506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

In [11]:

```
# X = data[['LSTAT', 'RM', 'PTRATIO']]
X = data.iloc[:, :-1]
y = data.PRICE
```

In [12]:

```
# Splitting the data into train and test for building the model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state =
```

In [13]:

```
# Linear Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [14]:

```
# Fitting the model
regressor.fit(X_train,y_train)
```

Out[14]:

LinearRegression()

In [15]:

```
# Prediction on the test dataset
y_pred = regressor.predict(X_test)
```

In [16]:

```
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

5.041784121402041

In [17]:

```
# Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [18]:

```
# Creating the neural network model
import keras
from keras.layers import Dense, Activation, Dropout
from keras.models import Sequential

model = Sequential()

model.add(Dense(128, activation = 'relu', input_dim = 13))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(32, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [19]:

```
model.fit(X_train, y_train, epochs = 100)
```

```
Epoch 24/100
13/13 [=====] - 0s 2ms/step - loss: 10.8369
Epoch 25/100
13/13 [=====] - 0s 3ms/step - loss: 10.7921
Epoch 26/100
13/13 [=====] - 0s 2ms/step - loss: 10.3413
Epoch 27/100
13/13 [=====] - 0s 3ms/step - loss: 10.2014
Epoch 28/100
13/13 [=====] - 0s 2ms/step - loss: 9.9532
Epoch 29/100
13/13 [=====] - 0s 2ms/step - loss: 9.8095
Epoch 30/100
13/13 [=====] - 0s 3ms/step - loss: 9.7494
Epoch 31/100
13/13 [=====] - 0s 4ms/step - loss: 9.6172
Epoch 32/100
13/13 [=====] - 0s 3ms/step - loss: 9.2688
Epoch 33/100
13/13 [=====] - 0s 3ms/step - loss: 9.2943
```

In [20]:

```
y_pred = model.predict(X_test)
```

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

In [27]:

```
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
```

```
3.1410293739926494
```


Assignment No. 2

- **Title:** Classification using Deep neural network
- **Date of Completion:**
- **Objective:** To study and understand how to solve classification problem using deep neural network.
- **Problem Statement:** Binary classification using Deep Neural Networks Example: Classify movie reviews into "positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset.

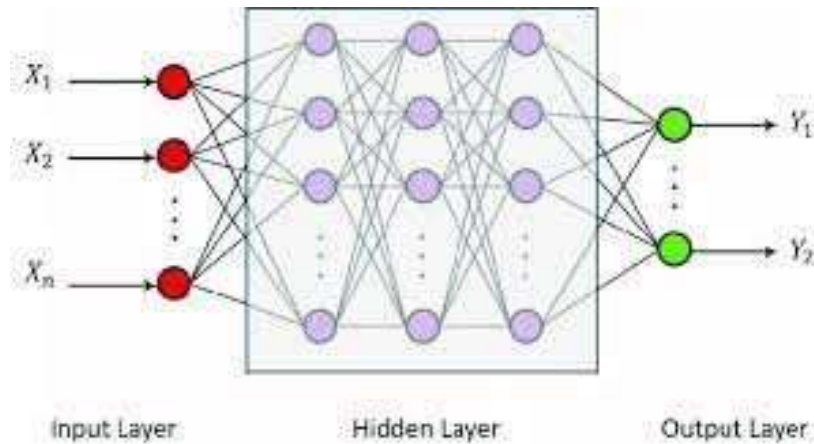
- **Theory:**

Binary classification refers to those **classification tasks that have two class labels**. Examples include: Email spam detection (spam or not). Churn prediction (churn or not).

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.

Deep learning (DL) is a subfield of machine learning based on learning multiple levels of representations by making a hierarchy of features where the higher levels are defined from the lower levels and the same lower level features can help in defining many higher level features. DL structure extends the traditional neural networks by adding more hidden layers to the network architecture between the input and output layers to model more complex and nonlinear relationships.

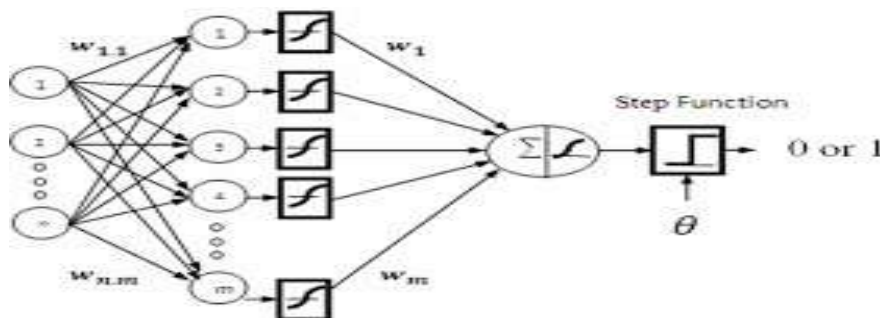
Deep Neural Network is another DL architecture that is widely used for classification or regression with success in many areas. It's a typical feedforward network which the input flows from the input layer to the output layer through number of hidden layers which are more than two layers.



The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification.

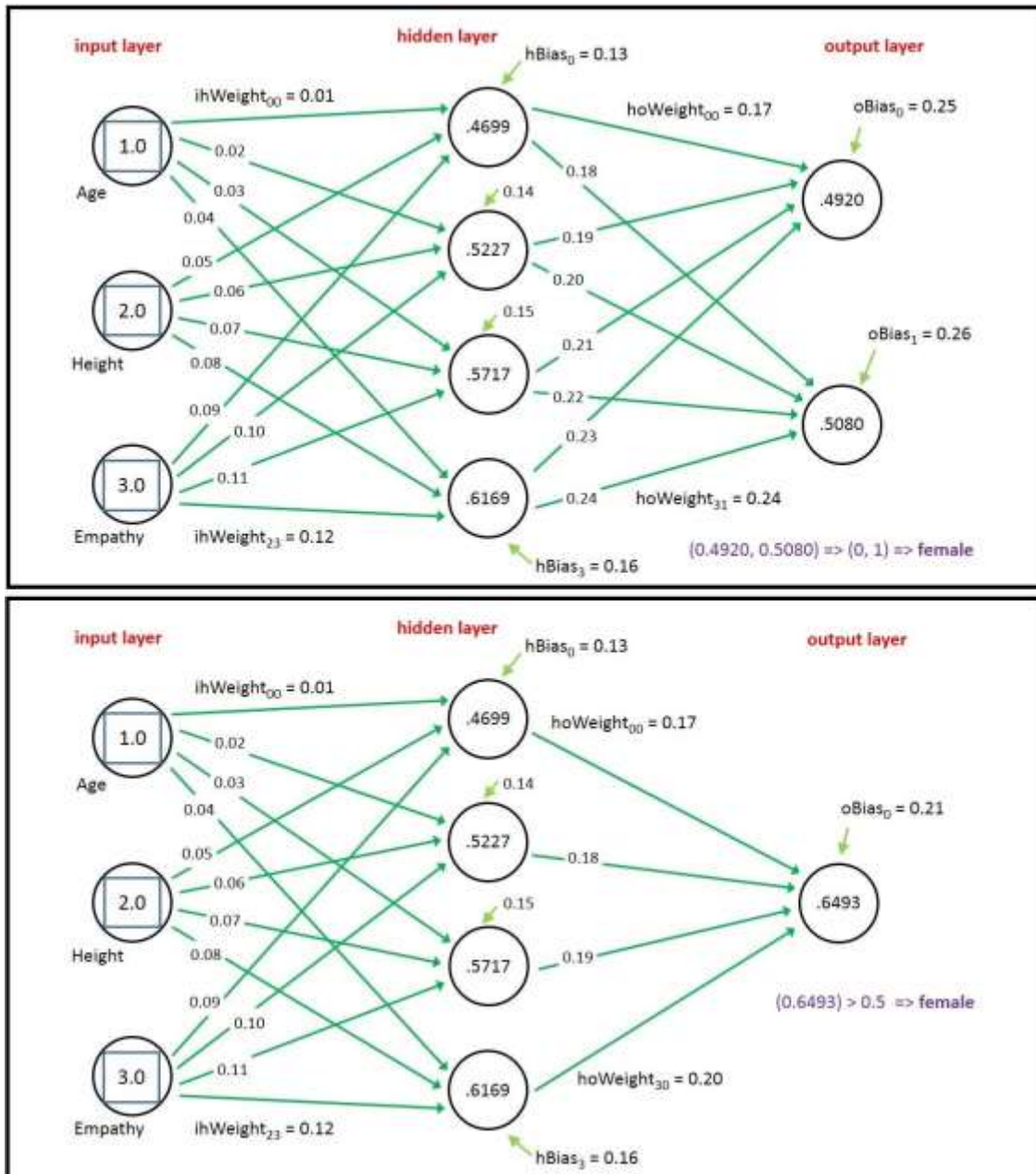
The IMDB sentiment classification dataset consists of 50,000 movie reviews from IMDB users that are labeled as either positive (1) or negative (0). The reviews are preprocessed and each one is encoded as a sequence of word indexes in the form of integers. The words within the reviews are indexed by their overall frequency within the dataset.

Building a neural network that performs binary classification involves making two simple changes:



- Add an activation function – specifically, the *sigmoid* activation function – to the output layer. Sigmoid reduces the output to a value from 0.0 to 1.0 representing a probability. For a reminder of what a sigmoid function does, see my post on binary classification.

- Change the loss function to *binary_crossentropy*, which is purpose-built for binary classifiers. Accordingly, change *metrics* to 'accuracy' so accuracies computed by the loss function are captured in the history object returned by *fit*.



- **Dataset :** IMDB dataset

- **References/ Available link :**

1. <https://towardsdatascience.com/binary-classification-of-imdb-movie-reviews-648342bc70dd>

2. <https://vsokolov.org/courses/41000/notes/nlp-imdb.html>

- **Questions and answer :**

1. What is Deep Learning?
2. What is a Neural network?
3. What is Binary Classification?
4. What types of Classification Algorithms do you know?
5. Can you choose a classifier based on the size of the training set?
6. Could you convert Regression into Classification and vice versa?
7. How do you use a supervised Logistic Regression for Classification?
8. How does ROC curve and AUC value help measure how good a model is?
9. Name some classification metrics and when would you use each one?
10. What's the difference between Generative Classifiers and Discriminative Classifiers? Name some examples of each one?

- **Mcqs:**

1. Following are the types of supervised learning_____
 - a. regression
 - b. classification
 - c. subgroup discovery
 - d. All of above
2. Following are the descriptive models_____
 - a. classification
 - b. clustering
 - c. association rule
 - d. Both 1 and 2
3. In binary classification number of classes must be_____
 - a. equals to two
 - b. less than two
 - c. greater than two
 - d. None

- **Conclusion :** We have successfully classified movie reviews using deep neural networks.

In [34]:

```
from keras.datasets import imdb

# Load the data, keeping only 10,000 of the most frequently occurring words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 10000)
```

In [35]:

```
# Here is a list of maximum indexes in every review
print(type([max(sequence) for sequence in train_data]))

# Find the maximum of all max indexes
max([max(sequence) for sequence in train_data])
```

<class 'list'>

Out[35]:

9999

In [36]:

```
# step 1: Load the dictionary mappings from word to integer index
word_index = imdb.get_word_index()

# step 2: reverse word index to map integer indexes to their respective words
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

# Step 3: decode the review, mapping integer indices to words
#
# indices are off by 3 because 0, 1, and 2 are reserved indices for "padding", "Start of
decoded_review = ' '.join([reverse_word_index.get(i-3, '?') for i in train_data[0]])

decoded_review
```

Out[36]:

"? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children are often left out of the ? list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

In [37]:

```
# Vectorize input data

import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension)) # Creates an all zero matrix of shape (len(sequences), dimension)
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1 # Sets specific indices of results to 1
    return results

# Vectorize training Data
X_train = vectorize_sequences(train_data)

# Vectorize testing Data
X_test = vectorize_sequences(test_data)
```

In [38]:

```
X_train[0]
```

Out[38]:

```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [39]:

```
X_train.shape
```

Out[39]:

```
(25000, 10000)
```

In [40]:

```
# Vectorize labels
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

In [41]:

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [42]:

```
from keras import optimizers
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001), loss = losses.binary_crossentropy,
              /usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/rmsprop.py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
              super(RMSprop, self).__init__(name, **kwargs)
```

In [43]:

```
# Input for Validation
X_val = X_train[:10000]
partial_X_train = X_train[10000:]

# Labels for validation
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```


In [44]:

```
history = model.fit(partial_X_train, partial_y_train, epochs=20, batch_size=512, validati
```

```
Epoch 1/20
30/30 [=====] - 2s 56ms/step - loss: 0.4878 - binary_accuracy: 0.7969 - val_loss: 0.3705 - val_binary_accuracy: 0.8656
Epoch 2/20
30/30 [=====] - 1s 45ms/step - loss: 0.2905 - binary_accuracy: 0.9047 - val_loss: 0.3232 - val_binary_accuracy: 0.8687
Epoch 3/20
30/30 [=====] - 2s 54ms/step - loss: 0.2151 - binary_accuracy: 0.9300 - val_loss: 0.2884 - val_binary_accuracy: 0.8847
Epoch 4/20
30/30 [=====] - 1s 42ms/step - loss: 0.1678 - binary_accuracy: 0.9447 - val_loss: 0.2873 - val_binary_accuracy: 0.8839
Epoch 5/20
30/30 [=====] - 2s 61ms/step - loss: 0.1391 - binary_accuracy: 0.9546 - val_loss: 0.2910 - val_binary_accuracy: 0.8835
Epoch 6/20
30/30 [=====] - 2s 63ms/step - loss: 0.1157 - binary_accuracy: 0.9634 - val_loss: 0.2987 - val_binary_accuracy: 0.8840
Epoch 7/20
30/30 [=====] - 1s 41ms/step - loss: 0.0975 - binary_accuracy: 0.9701 - val_loss: 0.3287 - val_binary_accuracy: 0.8774
Epoch 8/20
30/30 [=====] - 1s 34ms/step - loss: 0.0750 - binary_accuracy: 0.9795 - val_loss: 0.3497 - val_binary_accuracy: 0.8790
Epoch 9/20
30/30 [=====] - 1s 35ms/step - loss: 0.0700 - binary_accuracy: 0.9807 - val_loss: 0.3777 - val_binary_accuracy: 0.8768
Epoch 10/20
30/30 [=====] - 1s 34ms/step - loss: 0.0592 - binary_accuracy: 0.9826 - val_loss: 0.3836 - val_binary_accuracy: 0.8769
Epoch 11/20
30/30 [=====] - 1s 35ms/step - loss: 0.0467 - binary_accuracy: 0.9881 - val_loss: 0.4225 - val_binary_accuracy: 0.8697
Epoch 12/20
30/30 [=====] - 1s 35ms/step - loss: 0.0396 - binary_accuracy: 0.9909 - val_loss: 0.4334 - val_binary_accuracy: 0.8740
Epoch 13/20
30/30 [=====] - 1s 33ms/step - loss: 0.0306 - binary_accuracy: 0.9935 - val_loss: 0.4850 - val_binary_accuracy: 0.8631
Epoch 14/20
30/30 [=====] - 1s 33ms/step - loss: 0.0266 - binary_accuracy: 0.9948 - val_loss: 0.4939 - val_binary_accuracy: 0.8720
Epoch 15/20
30/30 [=====] - 1s 35ms/step - loss: 0.0208 - binary_accuracy: 0.9965 - val_loss: 0.5268 - val_binary_accuracy: 0.8700
Epoch 16/20
30/30 [=====] - 1s 34ms/step - loss: 0.0172 - binary_accuracy: 0.9977 - val_loss: 0.5638 - val_binary_accuracy: 0.8683
Epoch 17/20
30/30 [=====] - 1s 48ms/step - loss: 0.0141 - binary_accuracy: 0.9980 - val_loss: 0.5940 - val_binary_accuracy: 0.8655
Epoch 18/20
30/30 [=====] - 2s 52ms/step - loss: 0.0117 - binary_accuracy: 0.9983 - val_loss: 0.6302 - val_binary_accuracy: 0.8639
Epoch 19/20
30/30 [=====] - 1s 40ms/step - loss: 0.0102 - binary_accuracy: 0.9985 - val_loss: 0.6597 - val_binary_accuracy: 0.8657
Epoch 20/20
30/30 [=====] - 1s 34ms/step - loss: 0.0080 - binary_accuracy: 0.9989 - val_loss: 0.6970 - val_binary_accuracy: 0.8651
```

In [45]:

```
history_dict = history.history  
history_dict.keys()
```

Out[45]:

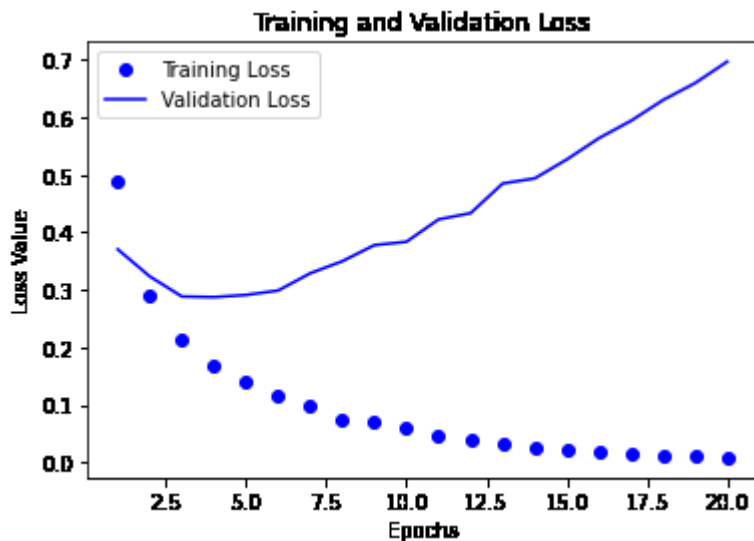
```
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

In [46]:

```
import matplotlib.pyplot as plt  
%matplotlib inline
```

In [47]:

```
# Plotting Losses  
loss_values = history_dict['loss']  
val_loss_values = history_dict['val_loss']  
  
epochs = range(1, len(loss_values) + 1)  
  
plt.plot(epochs, loss_values, 'bo', label="Training Loss")  
plt.plot(epochs, val_loss_values, 'b', label="Validation Loss")  
  
plt.title('Training and Validation Loss')  
plt.xlabel('Epochs')  
plt.ylabel('Loss Value')  
plt.legend()  
  
plt.show()
```



In [48]:

Training and Validation Accuracy

```

acc_values = history_dict['binary_accuracy']
val_acc_values = history_dict['val_binary_accuracy']

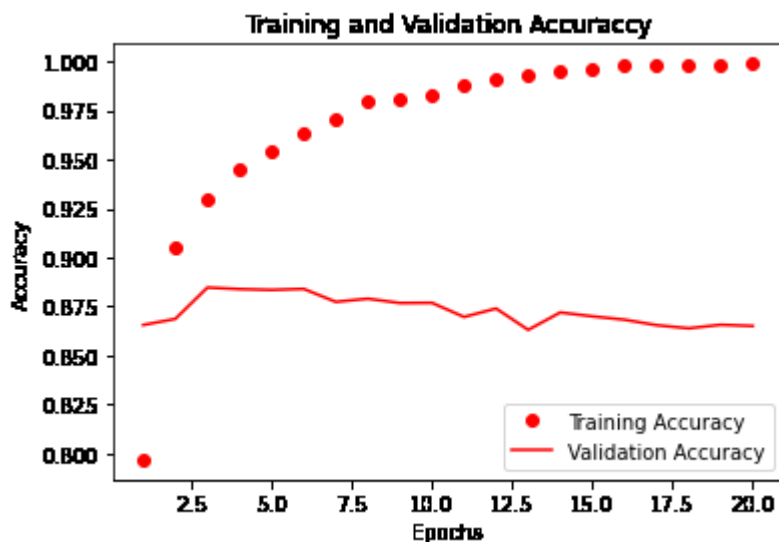
epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, acc_values, 'ro', label="Training Accuracy")
plt.plot(epochs, val_acc_values, 'r', label="Validation Accuracy")

plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()

```



In [49]:

```
model.fit(partial_X_train, partial_y_train, epochs=3, batch_size=512, validation_data=(X_
```

Epoch 1/3

```
30/30 [=====] - 2s 61ms/step - loss: 0.0040 - binary_accuracy: 0.9999 - val_loss: 0.7310 - val_binary_accuracy: 0.8648
```

Epoch 2/3

```
30/30 [=====] - 1s 33ms/step - loss: 0.0061 - binary_accuracy: 0.9988 - val_loss: 0.7557 - val_binary_accuracy: 0.8636
```

Epoch 3/3

```
30/30 [=====] - 1s 34ms/step - loss: 0.0039 - binary_accuracy: 0.9995 - val_loss: 0.8755 - val_binary_accuracy: 0.8475
```

Out[49]:

<keras.callbacks.History at 0x7f1cc77b7880>

In [50]:

```
# Making Predictions for testing data
np.set_printoptions(suppress=True)
result = model.predict(X_test)
```

782/782 [=====] - 2s 2ms/step

In [51]:

```
result
```

Out[51]:

```
array([[0.00190504],
       [1.         ],
       [0.09390965],
       ...,
       [0.00027408],
       [0.00282606],
       [0.3104798 ]], dtype=float32)
```

In [52]:

```
y_pred = np.zeros(len(result))
for i, score in enumerate(result):
    y_pred[i] = 1 if score > 0.5 else 0
```

In [53]:

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_pred, y_test)
```

In [54]:

```
# error
mae
```

Out[54]:

0.16668

Assignment No. 3

- **Title:**Convolutional neural network (CNN)
- **Date of Completion:**
- **Objective:** Study and understand Convolutional Neural Network by creating a classifier.
- **Problem Statement:**Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

- **Theory:**

CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

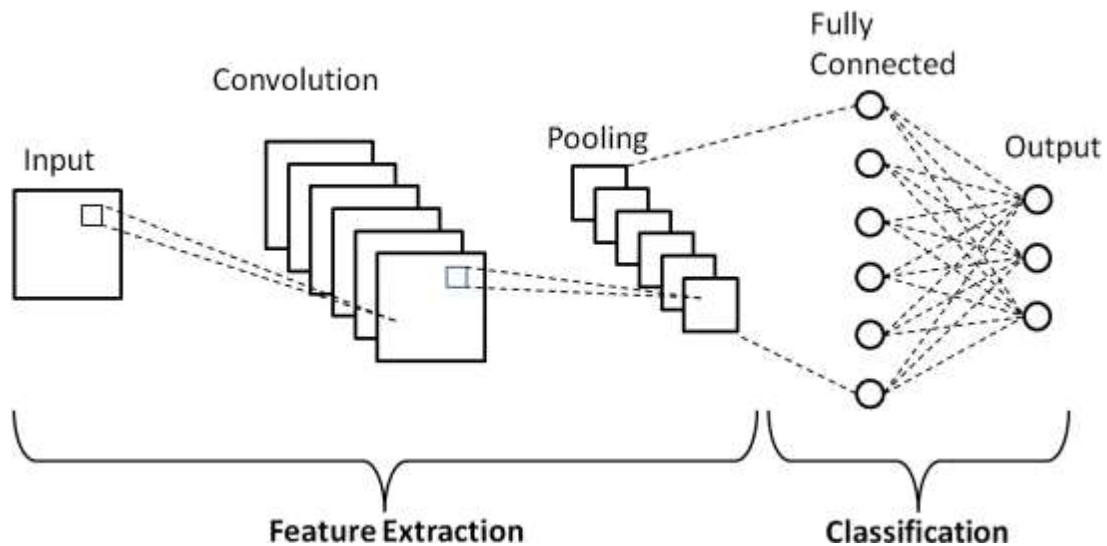
CNN has high accuracy, and because of the same, it is useful in image recognition. Image recognition has a wide range of uses in various industries such as medical image analysis, phone, security, recommendation systems, etc.

The term ‘Convolution’ in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.

Basic Architecture

There are two main parts to a CNN architecture

- A convolution tool that separates and identifies the various features of the image for analysis in a process called Feature Extraction.
- The network of feature extraction consists of many pairs of convolutional or pooling layers.
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
- This CNN model of feature extraction aims to reduce the number of features present in a dataset. It creates new features which summarizes the existing features contained in an original set of features. There are many CNN layers as shown in the CNN architecture diagram.

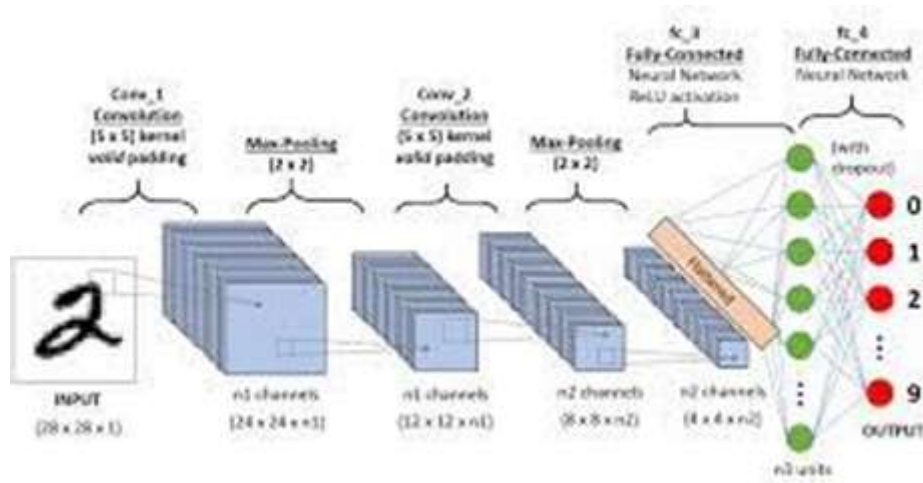


Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

Convolutional layer

Pooling layer

Fully-connected (FC) layer



1. Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size $M \times M$. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter ($M \times M$).

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The convolution layer in CNN passes the result to the next layer once applying the convolution operation in the input. Convolutional layers in CNN benefit a lot as they ensure the spatial relationship between the pixels is intact.

2. Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. It basically summarises the features generated by a convolution layer.

In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

This CNN model generalises the features extracted by the convolution layer, and helps the networks to recognise the features independently. With the help of this, the computations are also reduced in a network.

3. Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place. The reason two layers are connected is that two fully connected layers will perform better than a single connected layer. These layers in CNN reduce the human supervision

There are two main types of pooling:

Max pooling: As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.

Average pooling: As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

The name of the full-connected layer aptly describes itself. The pixel values of the input image are not directly connected to the output layer in partially connected layers.

However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer. This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While

convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

- **Dataset :** MNIST Fashion Dataset.
- **References/ Available link :**
 1. <https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-classification/>
 2. <https://www.kaggle.com/code/anindya2906/fashion-mnist-classification>
- **Questions and answer :**
 1. What do you mean by Convolutional Neural Network?
 2. Why do we prefer Convolutional Neural networks (CNN) over Artificial Neural networks (ANN) for image data as input?
 3. Explain the different layers in CNN?
 4. Explain the significance of the RELU Activation function in Convolution Neural Network?
 5. Why do we use a Pooling Layer in a CNN?
 6. What is the size of the feature map for a given input size image, Filter Size, Stride, and Padding amount?
 7. Explain the terms “Valid Padding” and “Same Padding” in CNN?
 8. What are the different types of Pooling? Explain their characteristics?
 9. Does the size of the feature map always reduce upon applying the filters? Explain why or why not?
 10. What is Stride? What is the effect of high Stride on the feature map?
- **Mcqs:**
 1. Which is the most direct application of neural networks?
 - a) vector quantization
 - b) pattern mapping
 - c) pattern classification
 - d) control applications
 2. What are pros of neural networks over computers?
 - a) they have ability to learn b examples
 - b) they have real time high computational rates
 - c) they have more tolerance
 - d) all of the mentioned

3. which of the following is false?

- a) neural networks are artificial copy of the human brain
- b) neural networks have high computational rates than conventional computers
- c) neural networks learn by examples
- d) none of the mentioned

- **Conclusion :**

We have successfully created a classifier using Convolutional neural network.

In [25]:

```
from __future__ import absolute_import, division, print_function

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
```

In [26]:

```
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

In [27]:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [28]:

```
train_images.shape
```

Out[28]:

```
(60000, 28, 28)
```

In [29]:

```
len(train_labels)
```

Out[29]:

```
60000
```

In [30]:

```
train_labels
```

Out[30]:

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

In [31]:

```
test_images.shape
```

Out[31]:

```
(10000, 28, 28)
```

In [32]:

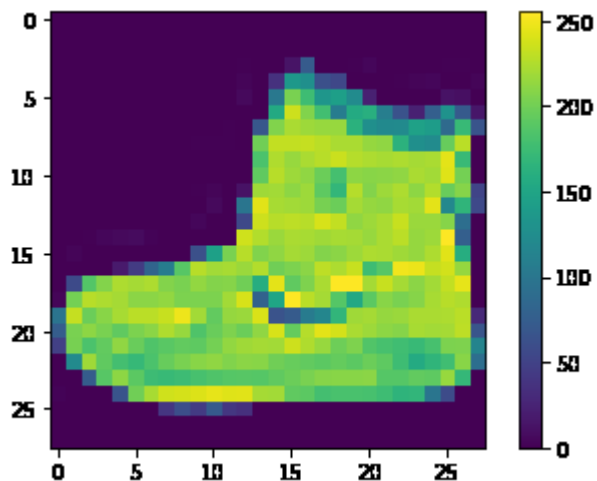
```
len(test_labels)
```

Out[32]:

10000

In [33]:

```
plt.figure()  
plt.imshow(train_images[0])  
plt.colorbar()  
plt.grid(False)  
plt.show()
```



In [34]:

```
train_images = train_images / 255.0  
test_images = test_images / 255.0
```

In [35]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



In [36]:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

In [37]:

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

In [38]:

```
model.fit(train_images, train_labels, epochs=5)
```

```
Epoch 1/5  
1875/1875 [=====] - 7s 4ms/step - loss: 0.4971 -  
accuracy: 0.8257  
Epoch 2/5  
1875/1875 [=====] - 6s 3ms/step - loss: 0.3751 -  
accuracy: 0.8648  
Epoch 3/5  
1875/1875 [=====] - 7s 4ms/step - loss: 0.3338 -  
accuracy: 0.8780  
Epoch 4/5  
1875/1875 [=====] - 7s 3ms/step - loss: 0.3104 -  
accuracy: 0.8856  
Epoch 5/5  
1875/1875 [=====] - 8s 4ms/step - loss: 0.2946 -  
accuracy: 0.8906
```

Out[38]:

```
<keras.callbacks.History at 0x7f07800931f0>
```

In [39]:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)  
  
print('Test accuracy:', test_acc)
```

```
313/313 [=====] - 1s 2ms/step - loss: 0.3542 - ac  
curacy: 0.8736  
Test accuracy: 0.8736000061035156
```

In [40]:

```
predictions = model.predict(test_images)
```

```
313/313 [=====] - 1s 2ms/step
```

In [41]:

```
predictions[0]
```

Out[41]:

```
array([4.9921914e-06, 3.3841974e-07, 1.8965457e-07, 1.1019566e-09,  
       1.3401749e-06, 4.8021390e-03, 1.4439345e-06, 5.2010346e-02,  
       2.6271462e-05, 9.4315302e-01], dtype=float32)
```

In [42]:

```
np.argmax(predictions[0])
```

Out[42]:

9

In [43]:

```
test_labels[0]
```

Out[43]:

9

In [44]:

```
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel("{} {:.2f}% ({}).format(class_names[predicted_label],
                                      100*np.max(predictions_array),
                                      class_names[true_label]),
              color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

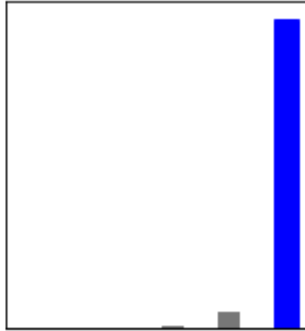
    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

In [45]:

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



Ankle boot 94% (Ankle boot)

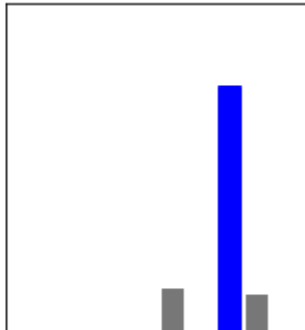


In [46]:

```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions, test_labels)
plt.show()
```



Sneaker 75% (Sneaker)

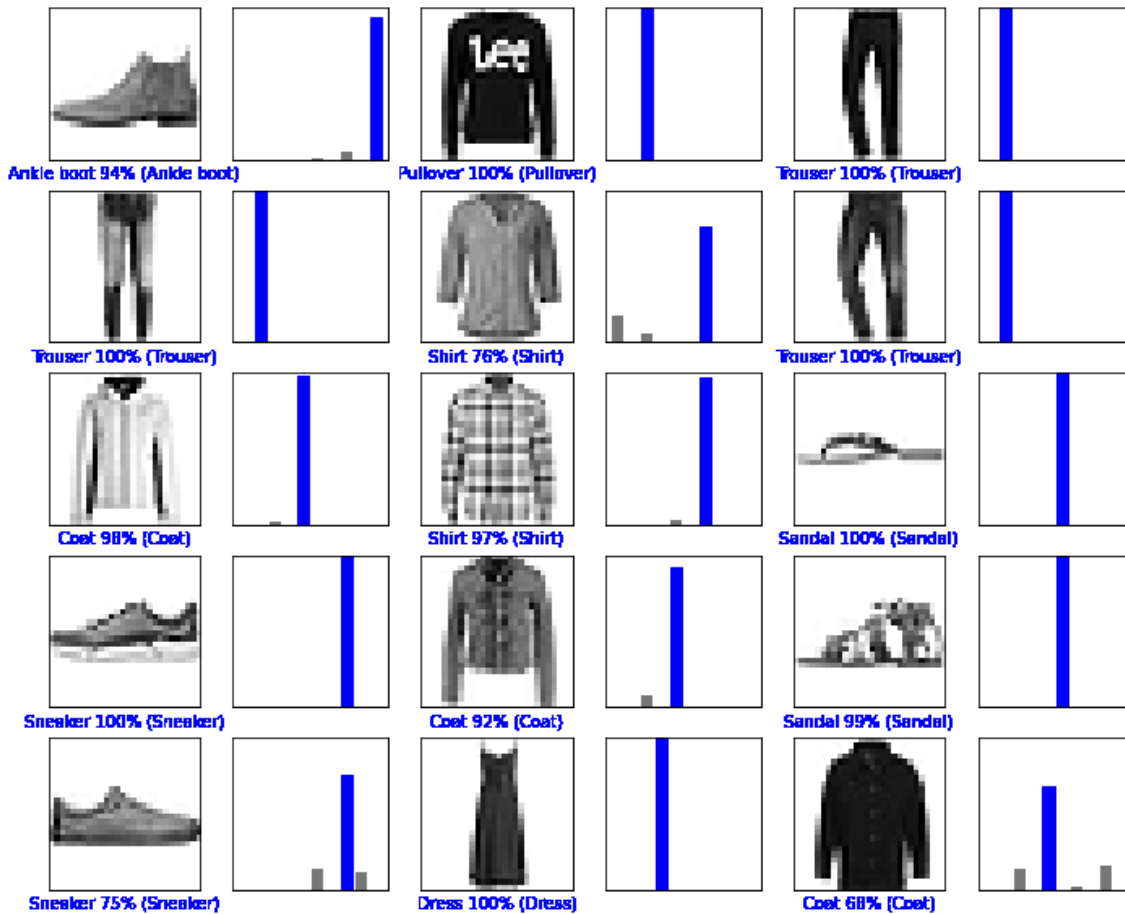


In [47]:

```

# Plot the first X test images, their predicted label, and the true label
# Color correct predictions in blue, incorrect predictions in red
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
plt.show()

```



In [48]:

```

# Grab an image from the test dataset
img = test_images[0]

print(img.shape)

```

(28, 28)

In [49]:

```
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))

print(img.shape)

(1, 28, 28)
```

In [50]:

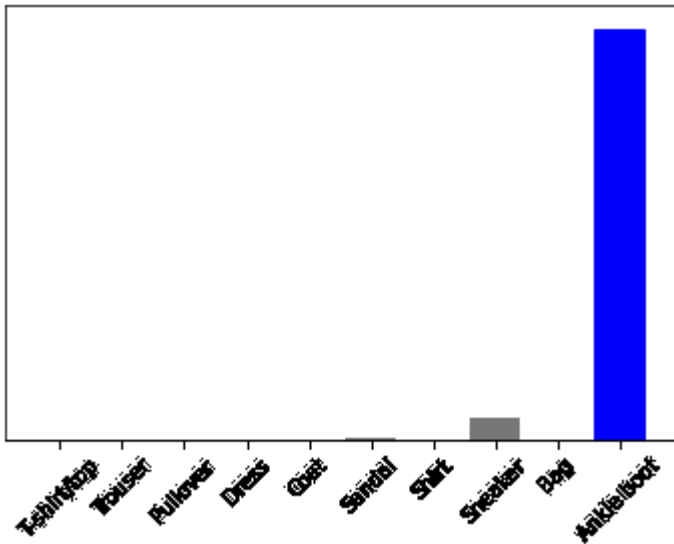
```
predictions_single = model.predict(img)

print(predictions_single)

1/1 [=====] - 0s 25ms/step
[[4.9921905e-06 3.3841968e-07 1.8965453e-07 1.1019544e-09 1.3401745e-06
 4.8021362e-03 1.4439315e-06 5.2010350e-02 2.6271480e-05 9.4315284e-01]]
```

In [51]:

```
plot_value_array(0, predictions_single, test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```



In [52]:

```
np.argmax(predictions_single[0])
```

Out[52]:

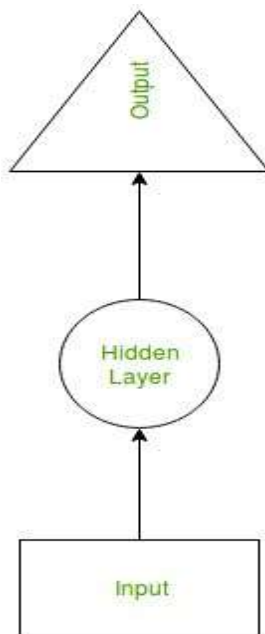
9

Assignment No. 4

- **Title:** Recurrent neural network (RNN) .
- **Date of Completion:**
- **Objective:** To study and understand Recurrent Neural Network by doing analysis and designing prediction systems.
- **Problem Statement:** Use the Google stock prices dataset and design a time series analysis and prediction system using RNN.

- **Theory:**

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.



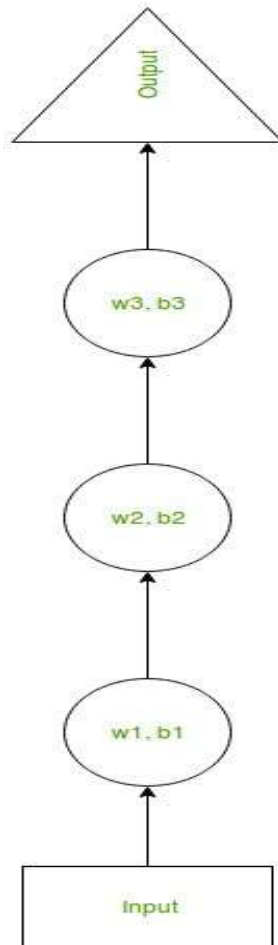
RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden

layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

How RNN works

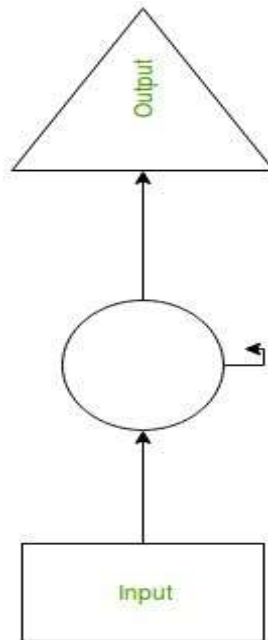
The working of an RNN can be understood with the help of the below example:

Example: Suppose there is a deeper network with one input layer, three hidden layers, and one output layer. Then like other neural networks, each hidden layer will have its own set of weights and biases, let's say, for hidden layer 1 the weights and biases are (w_1, b_1) , (w_2, b_2) for the second hidden layer, and (w_3, b_3) for the third hidden layer. This means that each of these layers is independent of the other, i.e. they do not memorize the previous outputs.



Now the RNN will do the following:

- RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer.
- Hence these three layers can be joined together such that the weights and bias of all the hidden layers are the same, in a single recurrent layer.



$$h_t = f(h_{t-1}, x_t)$$

The formula for calculating current state:

where:

h_t -> current state

h_{t-1} -> previous state

x_t -> input state

Formula for applying Activation function(tanh):

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

where:

W_{hh} -> weight at recurrent neuron

W_{xh} -> weight at input neuron

The formula for calculating output:

$$y_t = W_{hy}h_t$$

Y_t -> output

W_{hy} -> weight at output layer

Training through RNN

1. A single-time step of the input is provided to the network.
2. Then calculate its current state using a set of current input and the previous state.
3. The current h_t becomes h_{t-1} for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.

5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

Advantages of Recurrent Neural Network

1. An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of Recurrent Neural Network

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

Applications of Recurrent Neural Network

1. Language Modelling and Generating Text
2. Speech Recognition
3. Machine Translation
4. Image Recognition, Face detection
5. Time series Forecasting

Long short-term memory (LSTM)

This is a popular RNN architecture as a solution to vanishing gradient problem. It addresses the problem of long-term dependencies. That is, if the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state. LSTMs have “cells” in the hidden layers of the neural network, which have three gates—an input gate, an output gate, and a forget gate. These gates control the flow of information which is needed to predict the output in the network.

- **Dataset :** Google stock prices dataset
- **References/ Available link**
 - 1.<https://towardsdatascience.com/lstm-for-google-stock-price-prediction-e35f5cc84165>
 - 2.<https://medium.com/linkit-intecs/developing-a-recurrent-neural-network-rnn-for-a-google-stock-price-dataset-having-continuous-data-97b6e4e724a3>
- **Questions and answer :**

1. What's the difference between Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) and in which cases would use each one?
2. How many dimensions must the inputs of an RNN layer have?
3. What are the main difficulties when training RNNs? How can you handle them?
4. What are the uses of using RNN in NLP?
5. What types of Recurrent Neural Networks (RNN) do you know?
6. What's the difference between Stateful RNN vs Stateless RNN? What are their pros and cons?
7. What's the difference between Traditional Feedforward Networks and Recurrent Neural Networks?
8. What's the difference between Recurrent Neural Networks and Recursive Neural Networks?
9. When would you use MLP, CNN, and RNN?
10. Why are RNNs (Recurrent Neural Network) better than MLPs at predicting Time Series Data?

- **Mcqs:**

1. RNN Stands for?
 - a) Recursive Neural Network
 - b) Recurrent Neural Network
 - c). Removable Neural Network
 - d).Recurring Neural Network
2. The main and most important feature of RNN is_____.
 - a) Visible State
 - b) Hidden State
 - c) Present State
 - d) none of above
3. RNN remembers each and every information through_____.
 - a) Work
 - b) Time
 - c) Hours
 - d) Memory

- **Conclusion :** We have successfully implemented a recurrent neural network to create a classifier.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
```

In [2]:

```
data = pd.read_csv('Google_train_data.csv')
data.head()
```

Out[2]:

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7,380,500
1	1/4/2012	331.27	333.87	329.08	666.45	5,749,400
2	1/5/2012	329.83	330.75	326.89	657.21	6,590,300
3	1/6/2012	328.34	328.77	323.68	648.24	5,405,900
4	1/9/2012	322.04	322.29	309.46	620.76	11,688,800

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  -
 0   Date        1258 non-null   object  
 1   Open        1258 non-null   float64  
 2   High        1258 non-null   float64  
 3   Low         1258 non-null   float64  
 4   Close       1258 non-null   object  
 5   Volume      1258 non-null   object  
dtypes: float64(3), object(3)
memory usage: 59.1+ KB
```

In [4]:

```
data["Close"] = pd.to_numeric(data.Close, errors='coerce')
data = data.dropna()
trainData = data.iloc[:, 4:5].values
```


In [5]:

data.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1149 entries, 0 to 1257
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   Date    1149 non-null    object
 1   Open    1149 non-null    float64
 2   High    1149 non-null    float64
 3   Low     1149 non-null    float64
 4   Close   1149 non-null    float64
 5   Volume  1149 non-null    object
dtypes: float64(4), object(2)
memory usage: 62.8+ KB
```

In [6]:

```
sc = MinMaxScaler(feature_range=(0,1))
trainData = sc.fit_transform(trainData)
trainData.shape
```

Out[6]:

(1149, 1)

In [7]:

```
X_train = []
y_train = []

for i in range (60,1149): #60 : timestep // 1149 : Length of the data
    X_train.append(trainData[i-60:i,0])
    y_train.append(trainData[i,0])

X_train,y_train = np.array(X_train),np.array(y_train)
```

In [8]:

```
X_train = np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1)) #adding the batch_size
X_train.shape
```

Out[8]:

(1089, 60, 1)

In [9]:

```
model = Sequential()

model.add(LSTM(units=100, return_sequences = True, input_shape =(X_train.shape[1],1)))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences = False))
model.add(Dropout(0.2))

model.add(Dense(units =1))
model.compile(optimizer='adam',loss="mean_squared_error")
```

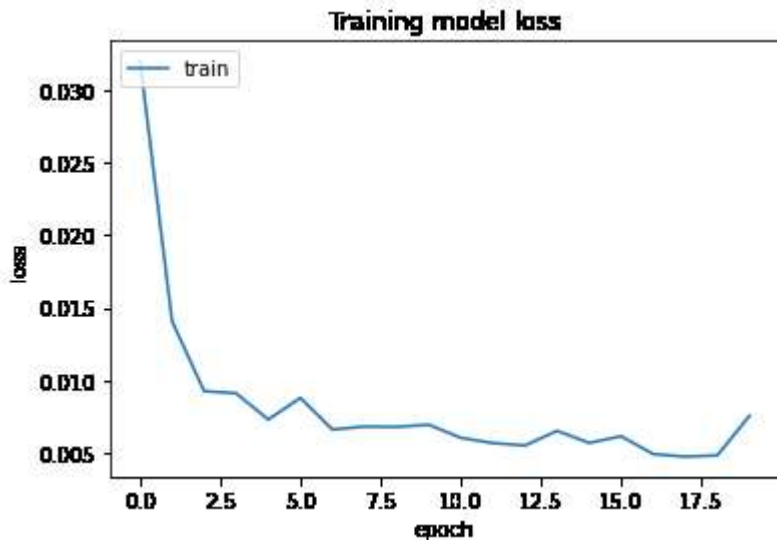
In [10]:

```
hist = model.fit(X_train, y_train, epochs = 20, batch_size = 32, verbose=2)
```

```
Epoch 1/20
35/35 - 16s - loss: 0.0320 - 16s/epoch - 459ms/step
Epoch 2/20
35/35 - 12s - loss: 0.0141 - 12s/epoch - 335ms/step
Epoch 3/20
35/35 - 9s - loss: 0.0093 - 9s/epoch - 244ms/step
Epoch 4/20
35/35 - 7s - loss: 0.0091 - 7s/epoch - 214ms/step
Epoch 5/20
35/35 - 10s - loss: 0.0073 - 10s/epoch - 285ms/step
Epoch 6/20
35/35 - 9s - loss: 0.0088 - 9s/epoch - 250ms/step
Epoch 7/20
35/35 - 7s - loss: 0.0066 - 7s/epoch - 212ms/step
Epoch 8/20
35/35 - 9s - loss: 0.0068 - 9s/epoch - 243ms/step
Epoch 9/20
35/35 - 9s - loss: 0.0068 - 9s/epoch - 253ms/step
Epoch 10/20
35/35 - 7s - loss: 0.0069 - 7s/epoch - 213ms/step
Epoch 11/20
35/35 - 9s - loss: 0.0060 - 9s/epoch - 247ms/step
Epoch 12/20
35/35 - 10s - loss: 0.0057 - 10s/epoch - 284ms/step
Epoch 13/20
35/35 - 7s - loss: 0.0055 - 7s/epoch - 214ms/step
Epoch 14/20
35/35 - 9s - loss: 0.0065 - 9s/epoch - 252ms/step
Epoch 15/20
35/35 - 8s - loss: 0.0057 - 8s/epoch - 242ms/step
Epoch 16/20
35/35 - 7s - loss: 0.0061 - 7s/epoch - 210ms/step
Epoch 17/20
35/35 - 9s - loss: 0.0049 - 9s/epoch - 243ms/step
Epoch 18/20
35/35 - 10s - loss: 0.0047 - 10s/epoch - 282ms/step
Epoch 19/20
35/35 - 7s - loss: 0.0048 - 7s/epoch - 211ms/step
Epoch 20/20
35/35 - 8s - loss: 0.0076 - 8s/epoch - 243ms/step
```

In [11]:

```
plt.plot(hist.history['loss'])
plt.title('Training model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```



In [12]:

```
testData = pd.read_csv('Google_test_data.csv')
testData["Close"] = pd.to_numeric(testData.Close, errors='coerce')
testData = testData.dropna()
testData = testData.iloc[:, 4:5]
y_test = testData.iloc[60:, 0].values
#input array for the model
inputClosing = testData.iloc[:, 0].values
inputClosing_scaled = sc.transform(inputClosing)
inputClosing_scaled.shape
X_test = []
length = len(testData)
timestep = 60
for i in range(timestep, length):
    X_test.append(inputClosing_scaled[i-timestep:i, 0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
X_test.shape
```

Out[12]:

(192, 60, 1)

In [13]:

```
y_pred = model.predict(X_test)
y_pred
```

6/6 [=====] - 2s 72ms/step

Out[13]:

```
array([[1.16179 ],
       [1.1631088],
       [1.1724331],
       [1.1868168],
       [1.1983689],
       [1.1982452],
       [1.1870549],
       [1.1716752],
       [1.1618347],
       [1.1591649],
       [1.1531832],
       [1.1432477],
       [1.1342392],
       [1.1257831],
       [1.1235965],
       [1.127025 ]])
```

In [14]:

```
predicted_price = sc.inverse_transform(y_pred)
```

In [15]:

```
plt.plot(y_test, color = 'red', label = 'Actual Stock Price')
plt.plot(predicted_price, color = 'green', label = 'Predicted Stock Price')
plt.title('Google stock price prediction')
plt.xlabel('Time')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```

