Learning Module

Workshop

"Deep Learning and Its Applications in Assisting Human"

2024

Contents

CHAPTER 1
Python Programming

1.1 OBJECTIVES

1. Understand the structure of the Python programming language
2. Understand the concept of block programming using indentation
3. Understand the concept of variables
4. Be able to use the if selection statement
5. Be able to create loops using for and while `while`

1.2 PYTHON PROGRAMMING

1.2.1 First Programs

```python
print("Deep Learning and Its Applications in Assisting Human"
)
```

1.2.2 Variabel

1. Variable: is a container for storing data values.
2. Variable Declaration:
   (a) Python does not have a command for declaring a variable.
   (b) A variable is created the moment you first assign a value to it.

- Example of Variable Declaration in Python

```python
#Example of a variable declaration
Number = 123 # Integer Declarations
Name = 'john' # The variable var to be a string.
print("Example of a variable declaration")
```

```python
print(Number)
print (Name)
#Casting Example
x = str(3)     # x  string  '3'
y = int(3)     # y contains an integer number 3
z = float(3)   # z to be the number 3.0
print("Casting Example")

print(x)
print(y)
print(z)

#Example of Displaying Variable Types
print("Example of Displaying Variable Types")

print(type(x))
print(type(y))
print(type(z))
```

1.3 DATA TYPES IN PYTHON

- **Text Type:** str
- **Numeric Types:** int, float, complex
- **Sequence Types:** list, tuple, range
- **Mapping Type:** dict
- **Set Types:** set,
- **Boolean Type:** bool
- **Binary Types:** bytes, bytearray, memoryview frozenset

1.4 DATA TYPE DECLARATION

x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

1.5 OPERATORS IN PYTHON

1.5.1 Arithmetic Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor Division	x // y

```
x = 5
y = 3
print("Adding ",x + y)
print("Substraction ",x - y)
print("Multiplication ",x * y)
print("Division ",x / y)
print("Modulus ",x % y)
print("Powers of ", x ** y)
print("Floor division ", x // y)
```

1.5.2 Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
¿	Greater than	x ¿ y
¡	Less than	x ¡ y
¿=	Greater than or equal to	x ¿= y
¡=	Less than or equal to	x ¡= y

```
x = 5
y = 3

print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

1.5.3 Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x ¡ 5 and x ¡ 10
or	Returns True if one of the statements is true	x ¡ 5 or x ¡ 4
not	Reverses the result, returns False if the result is true	not (x ¡ 5 and x ¡ 10)

```
x = 5

print(x > 3 and x < 10)
print(x > 3 or x < 4)
print(not(x > 3 and x < 10))
```

1.6 LIST

A list is used to store multiple items in a single variable.

- Example of Looping through List

```
# Example of ListData variable containing data of different
    types
ListData = ("Computer", ("Monitor", 2), (40.0, 30.5))
print(ListData)
print(len(ListData))
```

1.7 PYTHON INDENTATION

Python uses indentation to indicate code blocks. The program in Listing 3.1 is an `if` block in C, which is enclosed by an opening brace "{" and a closing brace "}". In Python, the `if` block is marked by indentation, meaning that code within the same block will align with the same left margin, as shown in the program example :

- Code block in C

```c
if (5>2)
{
    printf("Five is greater than two");
}
```

- Code block in Python

```python
if 5 > 2:
    print("Five is greater than two")
```

If the if block does not follow these rules, an error message will be displayed, as shown in the figure.

- Incorrect Indentation in Python

```python
if 5 > 2:
print("Five is greater than two")
```

1.8 IF STATEMENT

- Example of If Statement

```python
a = 33
b = 200

if b > a:
    print("b is greater than a")
```

1.9 RANGE FUNCTION

The range() function returns a sequence of numbers, starting from 0 (by default) by incrementing by 1 (by default) until before the specified number.
*range(start, stop, step)
Parameters:

- start(optional). An integer number that specifies start (Default 0).

- stop (Required). Integer number specifying stop (not included).
- step (optional). An integer specifying the increment (Default 1)

```
print("Create a number sequence from 1 to 5 and print it out")
x = range(6)
for n in x:
  print(n)

print("Create a number sequence from 3 to 5 and print it out")

x = range(3, 6)
for n in x:
  print(n)

print("Create sequence numbers from 3 to 19, in increments of 2."
   )

x = range(3, 20, 2)
for n in x:
  print(n)
```

1.10 WHILE LOOP

- Example of While Loop

```
i = 1
while i < 6:
    print(i)
    i += 1
```

1.11 FOR LOOP

- Example of For Loop Starting from 0-5

```
    for x in range(6):
        print(x)
```

- Example of For Loop Starting from 2-5

```
    for x in range(2, 6):
        print(x)
```

- Example of For Loop Starting from 2 with Step 3

```
    for x in range(2, 30, 3):
        print(x)
```

1.12 FUNCTIONS

- A function is a block of code that runs when it is called.
- Data can be passed into functions as parameters.
- Functions can also return data as a result.
- Functions are defined using the `def` keyword.

1.13 ACTIVITY

1.13.1 Creat A List

“The more you learn, the more you earn.”
– Warren Buffett –

CHAPTER 2
Numpy

2.1 NUMPY LIBRARY

Numpy (Numerical Python) is a library that consists of a multidimensional array object and a collection of routines to process the array.

2.2 USING NUMPY LIBRARY ON PYTHON

- Import Numpy Library

```
import numpy as np
```

2.3 CREATE ARRAY USING NUMPY

- Create Array using Numpy

```

import numpy as np

#Create array using numpy
v = np.array ([1,2,3,4,5])

#Print content v
print(v)

#Show v dimension
print(v.shape)
```

2.4 MULTI DIMENSION ARRAY

- Create Multi Dimension Array

```
import numpy as np

#Create array using numpy
v = np.array
   ([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])

#Print content v
print (v)

#Show v dimension
print (v.shape)
```

2.5 INDEXING AND SLICING

- Indexing and Slicing

```
import numpy as np
#Create 2D array using numpy
v = np.array
   ([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
#Print content v
print ("Show v Value")
print (v)
print ("Show v dimension")
print (v.shape)

#Take the value of v in the 0th column
c = v[:,0]
print ("Show c=v[:,0]")
print (c)
print ("Show the dimension")
print (c.shape)

#Take the value of v from row 0 to 3 and column 0 to 2
c = v [0:2,0:3]
print ("Show c=v [0:2,0:0:3]")
print (c)
print ("Show the dimension",c.shape)
print (c.shape)
```

2.6 CELL MULTIPLICATION OF TWO MATRICES

Example:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{4.1}$$

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \tag{4.2}$$

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \tag{4.3}$$

$$\mathbf{C} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix} \tag{4.4}$$

- Example of Cell Multiplication of Two Matrices

```python
import numpy as np
a = np.array ([[1,2],[3,4],[5,6]])
b = np.array ([[2,4],[6,8],[10,12]])
c = a*b
print ("a=",a)
print ("b=",b)
print ("c=",c)
```

```
a= [[1 2]
 [3 4]
 [5 6]]
b= [[ 2  4]
 [ 6  8]
 [10 12]]
c= [[ 2  8]
 [18 32]
 [50 72]]
```

Figure 2.1: Output of the program

2.7 TWO MATRICES MULTIPLICATION

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{4.5}$$

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \tag{4.6}$$

$$\mathbf{C} = \text{np.matmul}(\mathbf{A}, \mathbf{B}) \tag{4.7}$$

$$\mathbf{C} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} \tag{4.8}$$

- Example of Cell Multiplication of Two Matrices

```python
import numpy as np
a = np.array ([[1,2,3],[4,5,6]])
b = np.array ([[2,4],[6,8],[10,12]])
c = np.matmul (a,b)
print ("a=\n",a)
print ("b=\n",b)
print ("c=\n",c)
```

CHAPTER 3

Basic Image Processing Operations

An image is defined as a function of two dimensions.

$$z = f(x, y)$$

where:

- x and y: Spatial coordinates (plane)
- z or f: Intensity or gray level of the image at the point x and y

Digital Image:
An image that has x, y values and limited intensity z values or has discrete values.
Digital Image Processing:
Digital image processing using a digital computer.

3.1 OPENCV LIBRARY FOR READING AND DISPLAYING IMAGES

Requirements

1. Download and Install Anaconda Navigator: Download link here
2. Install OpenCV in Anaconda Navigator:

```
!pip install opencv-python
```

3. Download the image file in Dataset directory
4. Program Link: click here

Example:

1. Display the red color by filling layer 0 and layer 1 one by one using the `for` function

- Program to Read and Display an Image

```python
import cv2
import matplotlib.pyplot as plt
# Read File
sf = "/content/Day1 - Dataset/Lung Cancer DataSet/Normal/Normal case (1).jpg"
image = cv2.imread(sf)
# Convert color to RGB
img1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Plot RGB image with Matplotlib
plt.imshow(img1)
# display the image
plt.show()
print(image.shape)
```

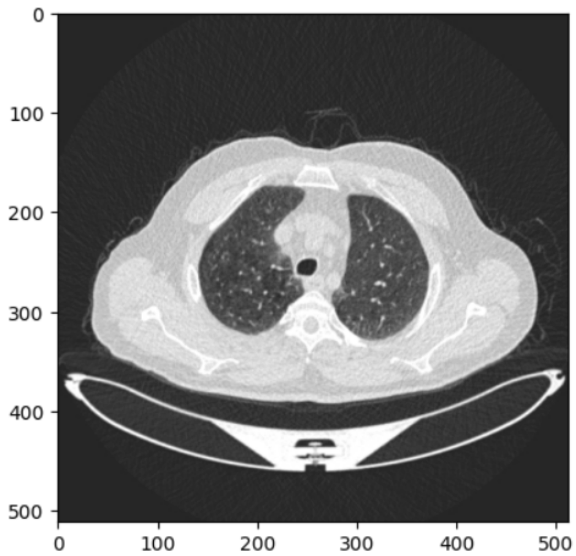

Figure 3.1: Output of the Program to Read and Display an Image

3.2 RESIZE IMAGE

To resize the image, use function
cv2.resize()
in the OpenCV library

```python
import cv2
import matplotlib.pyplot as plt
# Read File
sf = "/content/Day1 - Dataset/Lung Cancer DataSet/Normal/Normal case (1).jpg"
image = cv2.imread(sf)
# Convert color to RGB
img1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Plot RGB image with Matplotlib
```

```
plt.figure(1)
plt.imshow(img1)
width =200
height = 200
dsize = (width, height)
# resize image
img2 = cv2.resize(img1, dsize)
plt.figure(2)
plt.imshow(img2)
# display the image
plt.show()
print(img1.shape)
print(img2.shape)
```

3.3 PIXEL RELATIONSHIPS

The relationship between pixels is expressed using connectivity.

- A pixel at coordinates (x, y) has two vertical and horizontal neighbors with coordinates:

  $$(x+1, y), (x-1, y), (x, y+1), (x, y-1)$$

  This set of pixels is called the 4-neighborhood of p, denoted as $N_4(p)$.

- Four diagonal neighbors of p have coordinates:

  $$(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)$$

  which are denoted as $N_D(p)$.

- $N_D(p)$ together with $N_4(p)$ form the 8-neighborhood, denoted as $N_8(p)$.

$$N_8(p) = N_4(p) \cup N_D(p) \tag{3.1}$$

3.4 IMAGE GRADIENT

3.4.1 2D Filter

- caption=Fungsi Filter 2D

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image in grayscale mode
img = cv2.imread('/content/Image_Day1/Brain.png', cv2.
    IMREAD_GRAYSCALE)

```

```
# Sobel kernel
kernel = np.ones((5,5))/25

# Convolution
hasil_konvolusi = cv2.filter2D(img, -1, kernel)

# Display images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1, 2, 2), plt.imshow(hasil_konvolusi, cmap='gray'
    )
plt.title('Convolution Result'), plt.xticks([]), plt.yticks
    ([])
plt.tight_layout()
plt.show()
```

3.4.2 First Image Derivative

The first derivative in image processing is implemented from the magnitude of the gradient. The magnitude of the image f at coordinates (x, y) is defined as a two-dimensional vector:

$$\nabla f \equiv \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Thus, the magnitude of the image gradient is obtained as:

$$M(x, y) = \|\nabla f\| = \text{mag}(\nabla f) = \sqrt{q_x^2 + q_y^2}$$

or it can be approximated by:

$$M(x, y) = \|\nabla f\| = \text{mag}(\nabla f) = |q_x| + |q_y|$$

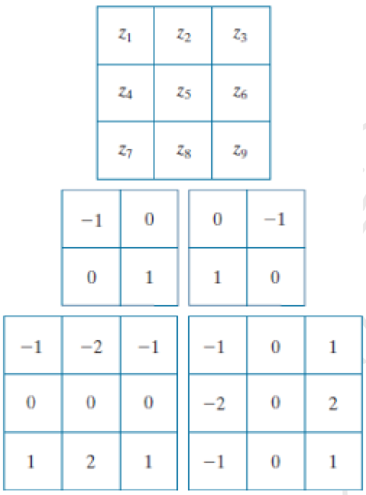

Figure 3.2: A 3x3 region of an image with z_5 as the intensity value at the center

3.4.3 Implementing Image Gradient using the Roberts Operator

- Gradient Calculation using the Roberts Operator

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def KonversiFloat2Uint8(im):
    im[im<0] = 0
    im[im>1] = 1
    return np.floor(im * 255).astype(np.uint8)

###########################
# MAIN PROGRAM
###########################

# Load image in grayscale mode (black and white)
img = np.double(cv2.imread('/content/Image_Day1/Brain.png
    ', cv2.IMREAD_GRAYSCALE))

# Normalization
img = img / 255

# Define the Roberts kernel
roberts_x = np.array([[ 0, 1],
                      [-1, 0]])
roberts_y = np.array([[ 1, 0],
                      [ 0, -1]])

# Convolve image with Roberts kernel
grad_x = cv2.filter2D(img, -1, roberts_x)
grad_y = cv2.filter2D(img, -1, roberts_y)

# Compute gradient magnitude
grad_magnitude = np.sqrt(grad_x**2 + grad_y**2)

# Display images
plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks
    ([])

# Display Result
plt.subplot(2, 2, 2), plt.imshow(grad_magnitude, cmap='
    gray')
plt.title('Gradient Magnitude'), plt.xticks([]), plt.
    yticks([])

# Display Grad X with values below 0 set to 0
plt.subplot(2, 2, 3), plt.imshow(KonversiFloat2Uint8(
    grad_x), cmap='gray')
plt.title('Gradient X'), plt.xticks([]), plt.yticks([])

```

```python
    # Display Grad Y with values below 0 set to 0
    plt.subplot(2, 2, 4), plt.imshow(KonversiFloat2Uint8(
        grad_y), cmap='gray')
    plt.title('Gradient Y'), plt.xticks([]), plt.yticks([])

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

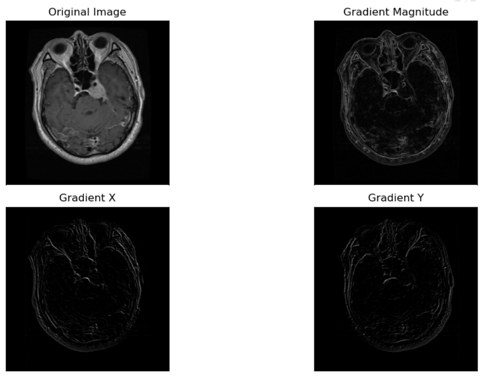

Figure 3.3

3.4.4 Highpass, Band Reject, and Bandpass Filters from Lowpass Filter

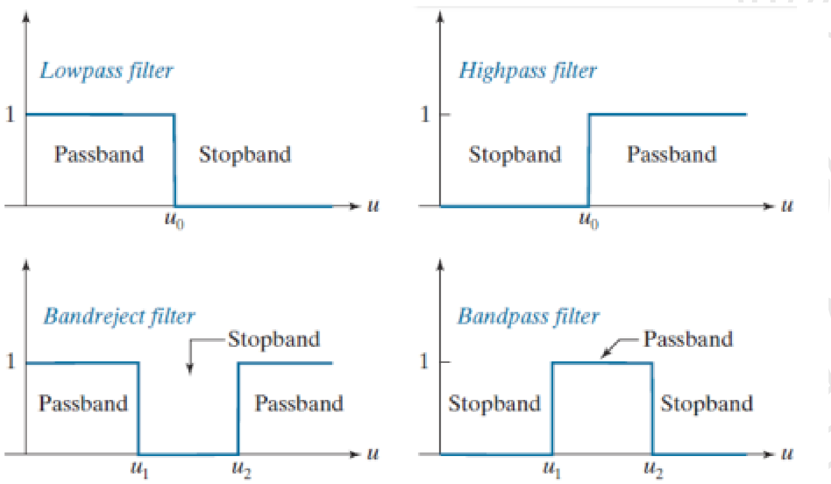

Figure 3.4: (a) Lowpass filter. (b) Highpass filter. (c) Bandreject filter. (d) Bandpass filter.

Highpass Filter Using Lowpass Filter

1. Known Image *d*: Figure 5.11
2. Lowpass Filter *lp* Using a 3x3 Box Filter: Figure 5.12
3. Highpass Filter $hp = d - lp$ Figure 5.13

Filter type	Spatial kernel in terms of lowpass kernel, lp
Lowpass	$lp(x, y)$
Highpass	$hp(x, y) = \delta(x, y) - lp(x, y)$
Bandreject	$br(x, y) = lp_1(x, y) + hp_2(x, y)$ $= lp_1(x, y) + [\delta(x, y) - lp_2(x, y)]$
Bandpass	$bp(x, y) = \delta(x, y) - br(x, y)$ $= \delta(x, y) - [lp_1(x, y) + (\delta(x, y) - lp_2(x, y))]$

Table 3.1: Filter types and their spatial kernels in terms of lowpass kernel, lp

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

Figure 3.5

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0.1	0.2	0.3	0.3	0.3	0.3	0.2	0.1	0	0
0	0	0.2	0.4	0.7	0.7	0.7	0.7	0.4	0.2	0	0
0	0	0.3	0.7	0.9	0.8	0.8	0.9	0.7	0.3	0	0
0	0	0.3	0.7	0.8	0.6	0.6	0.8	0.7	0.3	0	0
0	0	0.3	0.7	0.7	0.3	0.3	0.7	0.7	0.3	0	0
0	0	0.3	0.7	0.7	0.3	0.3	0.7	0.7	0.3	0	0
0	0	0.3	0.7	0.7	0.3	0.3	0.7	0.7	0.3	0	0
0	0	0.3	0.7	0.7	0.3	0.3	0.7	0.7	0.3	0	0
0	0	0.3	0.7	0.7	0.3	0.3	0.7	0.7	0.3	0	0
0	0	0.3	0.7	0.7	0.3	0.3	0.7	0.7	0.3	0	0
0	0	0.3	0.7	0.8	0.6	0.6	0.8	0.7	0.3	0	0
0	0	0.3	0.7	0.9	0.8	0.8	0.9	0.7	0.3	0	0
0	0	0.2	0.4	0.7	0.7	0.7	0.7	0.4	0.2	0	0
0	0	0.1	0.2	0.3	0.3	0.3	0.3	0.2	0.1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.6

0	0	0	0	0	0	0	0	0	0	0	0
0	0	-0	-0	-0	-0	-0	-0	-0	-0	0	0
0	0	-0	0.6	0.3	0.3	0.3	0.3	0.6	-0	0	0
0	0	-0	0.3	0.1	0.2	0.2	0.1	0.3	-0	0	0
0	0	-0	0.3	0.2	-1	-1	0.2	0.3	-0	0	0
0	0	-0	0.3	0.3	-0	-0	0.3	0.3	-0	0	0
0	0	-0	0.3	0.3	-0	-0	0.3	0.3	-0	0	0
0	0	-0	0.3	0.3	-0	-0	0.3	0.3	-0	0	0
0	0	-0	0.3	0.3	-0	-0	0.3	0.3	-0	0	0
0	0	-0	0.3	0.3	-0	-0	0.3	0.3	-0	0	0
0	0	-0	0.3	0.3	-0	-0	0.3	0.3	-0	0	0
0	0	-0	0.3	0.2	-1	-1	0.2	0.3	-0	0	0
0	0	-0	0.3	0.1	0.2	0.2	0.1	0.3	-0	0	0
0	0	-0	0.6	0.3	0.3	0.3	0.3	0.6	-0	0	0
0	0	-0	-0	-0	-0	-0	-0	-0	-0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.7

- Highpass filter using lowpass filter

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Function to convert float data to uint8
def KonversiFloat2Uint8(im):
    im[im<0]=0
    im[im>1]=1
    return np.floor(im*255).astype(np.uint8)

###############################################
# MAIN PROGRAM
###############################################
# Load the image in grayscale mode
img = cv2.imread('/content/Image_Day1/Brain.png', cv2.
   IMREAD_GRAYSCALE)
d = np.double(img) / 255

# Box filter kernel
kernel = np.ones((3,3))/9
# Find lowpass filter
lp = cv2.filter2D(d, -1, kernel)
# Calculate highpass filter using lowpass filter
hp = d - lp

###############################################
# Display the Resulting Highpass Image
###############################################
LowpassImage = KonversiFloat2Uint8(lp)
HighpassImage = KonversiFloat2Uint8(hp)

# Display original image
plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image d'), plt.xticks([]), plt.yticks
```

```
([])

    # Display Lowpass filter result
    plt.subplot(1, 3, 2), plt.imshow(LowpassImage, cmap='gray')
    plt.title('Lowpass filter result lp'), plt.xticks([]), plt.yticks([])

    # Display Highpass filter result (d - lp)
    plt.subplot(1, 3, 3), plt.imshow(HighpassImage, cmap='gray')
    plt.title('Highpass filter result hp=d-lp'), plt.xticks([]), plt.yticks([])

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

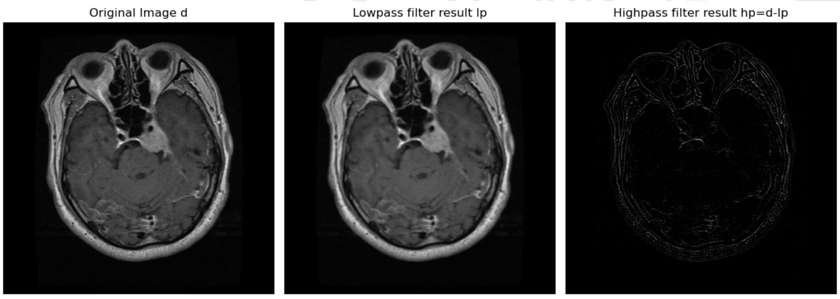

Figure 3.8: (a) Lowpass filter. (b) Highpass filter. (c) Bandreject filter. (d) Bandpass filter.

“The roots of education are bitter, but the fruit is sweet.”
– Aristotle –

CHAPTER 4
Linear Regression

4.1 PREDICTION USING LINEAR REGRESSION

It is known that Table 5.1 contains data from $X = \{x_1, x_2, \ldots, x_m\}$ as independent variables and y as the dependent variable obtained from n measurements.

Table 4.1: Data of n Measurements

NoData	x_1	x_2	x_3	$\cdots$	x_m	y
1	x_{11}	x_{12}	x_{13}	$\cdots$	x_{1m}	y_1
2	x_{21}	x_{22}	x_{23}	$\cdots$	x_{2m}	y_2
3	x_{31}	x_{32}	x_{33}	$\cdots$	x_{3m}	y_3
..	..	..	..	$\cdots$	..	..
n	x_{n1}	x_{n2}	x_{n3}	$\cdots$	x_{nm}	y_n

We aim to fit a linear model based on Equation 5.1 to the data in Table 3.1:

$$y = c_0 + c_1 x_1 + c_2 x_2 + \cdots + c_m x_m \tag{4.1}$$

By substituting the data from Table 3.1 into Equation 3.1, we obtain n equations:

$$\begin{aligned}
y_1 &= c_0 + c_1 x_{11} + c_2 x_{12} + \cdots + c_m x_{1m} \\
y_2 &= c_0 + c_1 x_{21} + c_2 x_{22} + \cdots + c_m x_{2m} \\
&\vdots \\
y_n &= c_0 + c_1 x_{n1} + c_2 x_{n2} + \cdots + c_m x_{nm}
\end{aligned} \tag{4.2}$$

Thus, Equation 3.2 becomes:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \tag{4.3}$$

if

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{4.4}$$

$$\mathbf{A} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1m} \\ 1 & x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \tag{4.5}$$

and

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} \tag{4.6}$$

In matrix form, Equation 3.2 becomes

$$\mathbf{y} = \mathbf{A}\mathbf{c} \tag{4.7}$$

The coefficient vector **c** is calculated using the equation:

$$\mathbf{c} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \tag{4.8}$$

or

$$\mathbf{c} = \mathbf{A}^{+}\mathbf{y} \tag{4.9}$$

where $\mathbf{A}^{+} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is known as the pseudo-inverse.
Therefore, the predicted result $\hat{\mathbf{y}}$ from **A** is

$$\hat{\mathbf{y}} = \mathbf{A}\mathbf{c} \tag{4.10}$$

4.2 SINGLE-VARIABLE LINEAR REGRESSION PROGRAM

Next, we will try to implement it. Table 3.1 shows the results of 7 measurements of independent variable x and dependent variable y. We want to fit Equation 3.12 to Table 3.1 by finding the best values for c_0 and c_1.
Next, Table 5.1 will be fitted to Equation 5.12.

$$y = c_0 + c_1 x \tag{4.11}$$

Thus, calculate c_0 and c_1.
Answer:

Table 4.2: Sample data of x and y with function

No	x	y
1	0.1	0.4
2	1	1.2
3	2.5	2.8
4	3	3.3
5	6	6.3
6	7	7.32
7	7.8	8.13

$$\mathbf{x} = \begin{bmatrix} 0.1 \\ 1 \\ 2.5 \\ 3 \\ 6 \\ 7 \\ 7.8 \end{bmatrix} \tag{4.12}$$

$$\mathbf{y} = \begin{bmatrix} 0.4 \\ 1.2 \\ 2.8 \\ 3.3 \\ 6.3 \\ 7.32 \\ 8.13 \end{bmatrix} \tag{4.13}$$

$$\mathbf{A} = [1 \quad \mathbf{x}] \tag{4.14}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0.1 \\ 1 & 1 \\ 1 & 2.5 \\ 1 & 3 \\ 1 & 6 \\ 1 & 7 \\ 1 & 7.8 \end{bmatrix} \tag{4.15}$$

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \tag{4.16}$$

Calculating $\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$, we get:

$$\mathbf{A}^+ = \begin{bmatrix} 0.41 & 0.35 & 0.24 & 0.20 & 0 & -0.07 & -0.13 \\ -0.06 & -0.05 & -0.02 & -0.01 & 0.03 & 0.05 & 0.07 \end{bmatrix} \tag{4.17}$$

Calculating $\mathbf{c}$:

$$\mathbf{c} = \mathbf{A}^+\mathbf{y} \tag{4.18}$$

$$\mathbf{c} = \begin{bmatrix} 0.41 & 0.35 & 0.24 & 0.20 & 0 & -0.07 & -0.13 \\ -0.06 & -0.05 & -0.02 & -0.01 & 0.03 & 0.05 & 0.07 \end{bmatrix} \begin{bmatrix} 0.4 \\ 1.2 \\ 2.8 \\ 3.3 \\ 6.3 \\ 7.32 \\ 8.13 \end{bmatrix} \tag{4.19}$$

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 0.079976 \\ 1.021218 \end{bmatrix} \tag{4.20}$$

Thus, the linear equation obtained is:

$$y = 0.079976 + 1.021218x \tag{4.21}$$

Next, the implementation in Python code is as follows:

- Simple Linear Regression Program

```python
import numpy as np
import matplotlib.pyplot as plt

# Data sampling
x= np.array([[0.1], [1], [2.5], [3], [6], [7], [7.8]])
y= np.array([[0.4], [1.2], [2.8], [3.3], [6.3], [7.32], [8.13]])

# Relationship between x and y satisfying the equation
# y = c0 + c1*x

br, col = x.shape
# Building Matrix A= [1 x]
A = np.ones((br, 1))
A = np.insert(A, [1], x, axis=1)

# Calculating Pseudo Inverse Matrix A with np.linalg.pinv function and storing in variable Ap
Ap = np.linalg.pinv(A)

# Calculating Parameter c  c=Ap*y
c = np.matmul(Ap, y)

# Applying obtained c to predict y values
# yp = A * c  -> yp = predicted values
yp = np.matmul(A, c)

# Displaying sample points
plt.scatter(x, y)

# Displaying the prediction result graph
plt.plot(x, yp)
```

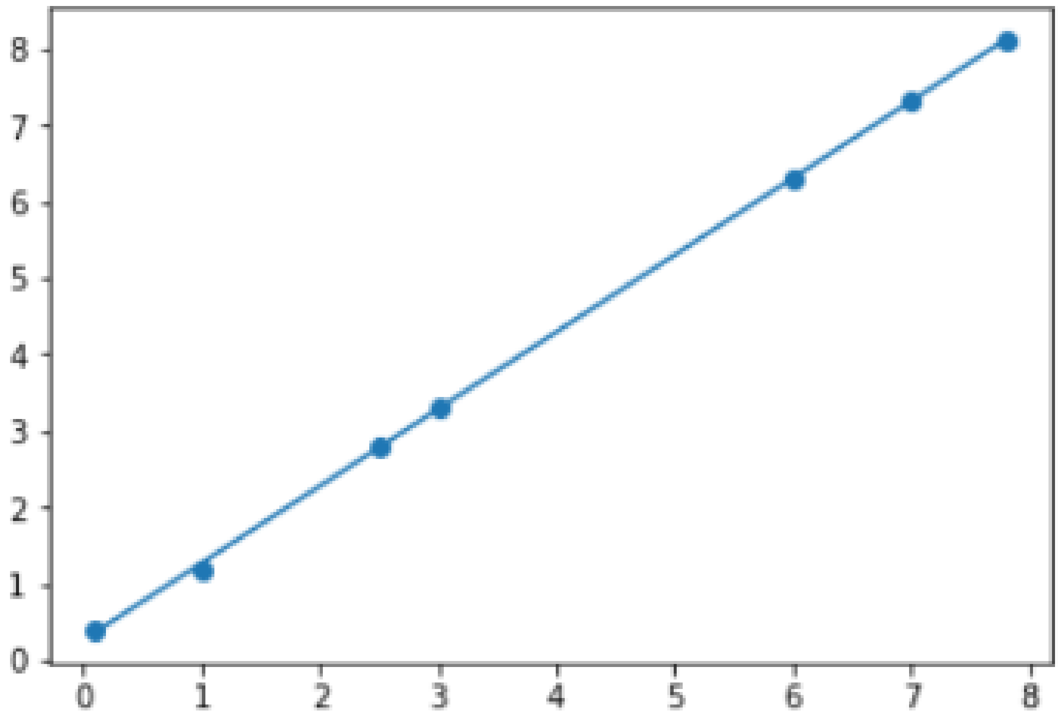

Figure 4.1: Result of Linear Regression Equation

4.3 LINEAR REGRESSION WITH TWO VARIABLES

The measurement data is given as shown in Table 3.3

x_1	x_2	y
1	4	24
4	3	27
3	1	17
5	3	33

We aim to fit Equation 5.23 to Table 5.3.

$$y = c_0 + c_1 x_1 + c_2 x_2 \tag{4.22}$$

For n data points, the relationship between x_1, x_2, and y is:

$$y_1 = c_0 + c_1 x_{11} + c_2 x_{12} \tag{4.23}$$
$$y_2 = c_0 + c_1 x_{21} + c_2 x_{22} \tag{4.24}$$
$$\vdots \tag{4.25}$$
$$y_n = c_0 + c_1 x_{n1} + c_2 x_{n2} \tag{4.26}$$

If written in matrix form, Equation 5.24 becomes:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} \tag{4.27}$$

Thus, we obtain

$$\mathbf{y} = \begin{bmatrix} 24 \\ 27 \\ 17 \\ 33 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 5 \end{bmatrix}, \quad \mathbf{x}_2 = \begin{bmatrix} 4 \\ 3 \\ 1 \\ 3 \end{bmatrix} \tag{4.28}$$

$$\mathbf{A} = [1 \quad \mathbf{x}_1 \quad \mathbf{x}_2] \tag{4.29}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 4 \\ 1 & 3 & 3 \\ 1 & 4 & 1 \\ 1 & 5 & 3 \end{bmatrix} \tag{4.30}$$

Calculating the pseudo-inverse of matrix $\mathbf{A}$:

$$\mathbf{A}^+ = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T \tag{4.31}$$

$$\mathbf{A}^+ = \begin{bmatrix} 1.238 & -0.611 & 1.507 & -1.134 \\ -0.333 & 0.055 & -0.111 & 0.388 \\ -0.047 & 0.222 & -0.301 & 0.126 \end{bmatrix} \tag{4.32}$$

Calculating $\mathbf{c}$ with $\mathbf{c} = \mathbf{A}^+\mathbf{y}$:

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -23.365 \\ 11.111 \\ 4.873 \end{bmatrix} \tag{4.33}$$

- Implementation of Linear Regression with Two Independent Variables in Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Sample Data for x1, x2, and y
x1= np.array([[1], [2], [3], [4]])
x2= np.array([[4], [5], [1], [3]])
y= np.array([[4], [27], [17], [33]])

# Relationship between x1, x2, and y satisfying the
   equation
# y = c0 + c1*x1 + c2*x2

br, col = x1.shape
# Building Matrix A= [1 x1 x2]
A = np.ones((br, 1))
A = np.insert(A, [1], x1, axis=1)
A = np.insert(A, [2], x2, axis=1)
```

```
    # Calculating the Pseudo Inverse of Matrix A using np.
        linalg.pinv and storing in variable Ap
    Ap = np.linalg.pinv(A)

    # Calculating Parameter c  c = Ap * y
    c = np.matmul(Ap, y)

    # Applying obtained c to predict y values
    # yp = A * c  -> yp = predicted values
    yp = np.matmul(A, c)

    # Displaying x1, x2, and y in 3D
    ax = plt.axes(projection="3d")

    # Displaying Sample Points
    ax.scatter3D(x1, x2, y, color = "red")
    ax.scatter3D(x1, x2, yp, color = "green")
    plt.show()
```

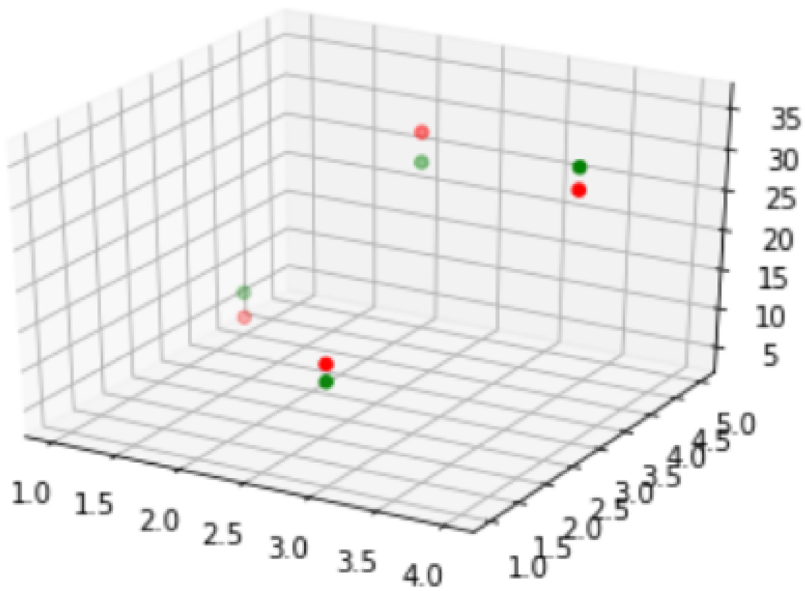

Figure 4.2: Result of Linear Regression Implementation with Two Variables

4.4 EXAMPLE OF SECOND-DEGREE POLYNOMIAL REGRESSION

The function below is fitted to the table above:

$$y = c_0 + c_1 x + c_2 x^2 \tag{4.34}$$

Table 4.3: Data Pairs from Measurement Results

	x	y
1	-5	73
2	-3	25
3	2	23
4	3	37
5	4	63

$$\mathbf{y} = \begin{bmatrix} 73 \\ 25 \\ 23 \\ 37 \\ 63 \end{bmatrix} \tag{4.35}$$

$$\mathbf{A} = \begin{bmatrix} 1 & -5 & 25 \\ 1 & -3 & 9 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix} \tag{4.36}$$

Calculating the pseudo-inverse of matrix **A**:

$$\mathbf{A}^{+} = \begin{bmatrix} -0.237 & 0.662 & 0.632 & 0.2351 & -0.292 \\ -0.046 & -0.088 & -0.007 & 0.041 & 0.099 \\ 0.035 & -0.035 & -0.034 & -0.003 & 0.0375 \end{bmatrix} \tag{4.37}$$

Calculating $\hat{\mathbf{c}} = \mathbf{A}^{+}\mathbf{y}$:

$$\hat{\mathbf{c}} = \begin{bmatrix} 4.0705 \\ 2.0947 \\ 3.1516 \end{bmatrix} \tag{4.38}$$

Thus, the resulting second-degree polynomial model is:

$$\hat{y} = 4.0705 + 2.0947X + 3.1516X^2 \tag{4.39}$$

Quadratic regression represents the points obtained from Table 3.4 alongside the graph derived from Equation 3.31

- Second-Degree Polynomial Regression Program

```
import numpy as np
import matplotlib.pyplot as plt

# Preparing Training Data
x=np.array([[1],[3],[4],[5],[7]])
y=np.array([[1],[9],[16],[26],[50]])
br, col = x.shape

# Building Matrix A= [1 X X^2]
A=np.ones((br,1))
A=np.insert(A,[1],x,axis=1)
A=np.insert(A,[2],np.power(x,2),axis=1)
```

```
# Calculating Pseudo Inverse Matrix X using np.linalg.
   pinv function
Ap=np.linalg.pinv(A)

# Calculating Parameter c  c=Xp*y
c=np.matmul(Ap,y)

# Applying obtained c to predict y values
yp = np.matmul(A,c)

# Displaying sample points
plt.scatter(x, y)

# Displaying the prediction result graph
plt.plot(x, yp)
```

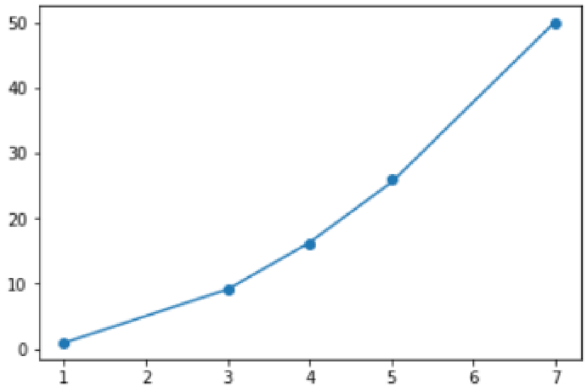

Figure 4.3: Graph of Predicted Results from the Polynomial Equation

4.5 LINEAR REGRESSION FOR RAINFALL PREDICTION

Next, we will try linear regression to predict rainfall based on rainfall data over 6 years, from 2001 to 2006.

Table 4.4: Monthly Rainfall Data from 2001 to 2006

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2001	1437	1076	682	416	248	189	124	137	122	582	733	829
2002	1251	873	577	214	226	60	95	89	102	562	786	840
2003	970	484	515	473	348	203	36	35	89	563	251	295
2004	584	373	300	165	128	91	16	3	324	372	376	209
2005	786	758	643	328	117	0	0	0	47	96	138	0
2006	1420	1230	920	563	294	0	15	0	0	86	222	0

A model is desired that can predict monthly rainfall as closely as possible to the historical data we already have. For that purpose, we choose the model to be an 8th-degree polynomial equation as follows:

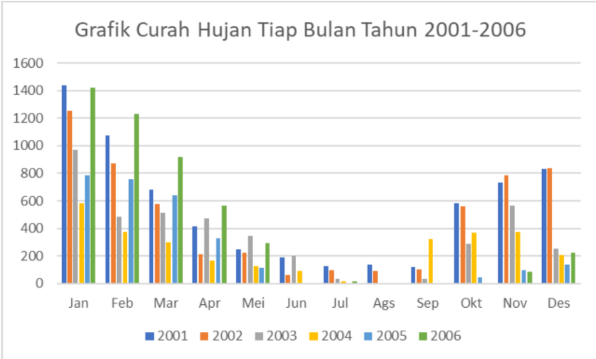

Figure 4.4: Monthly Rainfall Data in Bar Chart Form

$$y = c_0 + c_1x^1 + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6 + c_7x^7 + c_8x^8 \tag{4.40}$$

where x represents the month, and y represents the rainfall.
The first step is to prepare the data that will be used for training, in the format Year, Month, and Rainfall, separated by commas, as shown in the data below.

- Format of DataCurahHujan.csv file

```
    2001.00 ,1.00 ,1437.00
    2001.00 ,2.00 ,1076.00
    2001.00 ,3.00 ,682.00
    2001.00 ,4.00 ,416.00
    2001.00 ,5.00 ,248.00
    2001.00 ,6.00 ,189.00
    2001.00 ,7.00 ,124.00
    2001.00 ,8.00 ,137.00
    2001.00 ,9.00 ,122.00
    2001.00 ,10.00 ,582.00
    2001.00 ,11.00 ,733.00
    2001.00 ,12.00 ,829.00
    2002.00 ,1.00 ,1251.00
    2002.00 ,2.00 ,873.00
    2002.00 ,3.00 ,577.00
    2002.00 ,4.00 ,214.00
    2002.00 ,5.00 ,226.00
    2002.00 ,6.00 ,60.00
    2002.00 ,7.00 ,95.00
    2002.00 ,8.00 ,89.00
    2002.00 ,9.00 ,102.00
    2002.00 ,10.00 ,562.00
    2002.00 ,11.00 ,786.00
    2002.00 ,12.00 ,840.00
    2003.00 ,1.00 ,970.00
    2003.00 ,2.00 ,484.00
    2003.00 ,3.00 ,515.00
    2003.00 ,4.00 ,473.00
    2003.00 ,5.00 ,348.00
    2003.00 ,6.00 ,203.00
    2003.00 ,7.00 ,36.00
    2003.00 ,8.00 ,0.00
```

```
    2003.00 ,9.00 ,35.00
    2003.00 ,10.00 ,289.00
    2003.00 ,11.00 ,563.00
    2003.00 ,12.00 ,251.00
    2004.00 ,1.00 ,584.00
    2004.00 ,2.00 ,373.00
    2004.00 ,3.00 ,300.00
    2004.00 ,4.00 ,165.00
    2004.00 ,5.00 ,128.00
    2004.00 ,6.00 ,91.00
    2004.00 ,7.00 ,16.00
    2004.00 ,8.00 ,3.00
    2004.00 ,9.00 ,324.00
    2004.00 ,10.00 ,372.00
    2004.00 ,11.00 ,376.00
    2004.00 ,12.00 ,209.00
    2005.00 ,1.00 ,786.00
    2005.00 ,2.00 ,758.00
    2005.00 ,3.00 ,643.00
    2005.00 ,4.00 ,328.00
    2005.00 ,5.00 ,117.00
    2005.00 ,6.00 ,0.00
    2005.00 ,7.00 ,0.00
    2005.00 ,8.00 ,0.00
    2005.00 ,9.00 ,0.00
    2005.00 ,10.00 ,47.00
    2005.00 ,11.00 ,96.00
    2005.00 ,12.00 ,138.00
    2006.00 ,1.00 ,1420.00
    2006.00 ,2.00 ,1230.00
    2006.00 ,3.00 ,920.00
    2006.00 ,4.00 ,563.00
    2006.00 ,5.00 ,294.00
    2006.00 ,6.00 ,0.00
    2006.00 ,7.00 ,15.00
    2006.00 ,8.00 ,0.00
    2006.00 ,9.00 ,0.00
    2006.00 ,10.00 ,0.00
    2006.00 ,11.00 ,86.00
    2006.00 ,12.00 ,222.00
```

- Linear Regression for Rainfall Prediction

```
###################################
#Rainfall Data Set Consists of Three Columns: Year, Month,
   Rainfall
#Index numpy array starting from 0 to N-1 where N is the
   amount of data
#Year : Column 0, Month Column 1, Rainfall Column 2
#Model is a polynomial equation of degree 6
# y= c0+c1x^1+c2x^2+c3x^3+c4x^4+c5x^5+c6x^6
###################################

```

```python
import numpy as np
import matplotlib.pyplot as plt

def LearningCurahHujan(x,y):
    JumlahData = x.shape[0]
    # membangun matrix untuk persamaan:
    # y= c0*x^0+c1*x^1+c2*x^2+c3*x^3+c4*x^4+c5*x^5+c6*x^6
    A=np.ones([JumlahData,1])
    A=np.insert(A,[1],x,axis=1)
    A=np.insert(A,[1],np.power(x,2),axis=1)
    A=np.insert(A,[1],np.power(x,3),axis=1)
    A=np.insert(A,[1],np.power(x,4),axis=1)
    A=np.insert(A,[1],np.power(x,5),axis=1)
    A=np.insert(A,[1],np.power(x,6),axis=1)
    A=np.insert(A,[1],np.power(x,7),axis=1)
    A=np.insert(A,[1],np.power(x,8),axis=1)
    # Menghitung Pseudo Inverse Matrix A dengan fungsi np
       .linalg.pinv dan disimpan dalam variabel Ap
    Ap=np.linalg.pinv(A)
    # Menghitung Parameter c  c=Ap*y
    c=np.matmul(Ap,y)
    return c

def PrediksiCurahHujan(x,c):
    br = x.shape[0]
    A= np.ones([br,1])
    A=np.insert(A,[1],x,axis=1)
    A=np.insert(A,[1],np.power(x,2),axis=1)
    A=np.insert(A,[1],np.power(x,3),axis=1)
    A=np.insert(A,[1],np.power(x,4),axis=1)
    A=np.insert(A,[1],np.power(x,5),axis=1)
    A=np.insert(A,[1],np.power(x,6),axis=1)
    A=np.insert(A,[1],np.power(x,7),axis=1)
    A=np.insert(A,[1],np.power(x,8),axis=1)
    yp=np.matmul(A,c)
    return yp

#########################################
# Main Program
#########################################
# Membaca file DataCurahHujan
Data=np.loadtxt("DataCurahHujan.csv",delimiter=",")
X=Data[:, 1:2]
Y=Data[:, 2:3]

c=LearningCurahHujan(X,Y)

# Membuat Data Tes
X_tes =np.r_[1:12:0.1]
JumlahData = len(X_tes)
X_tes=np.reshape(X_tes,(JumlahData,1))
X_tes=PrediksiCurahHujan(X_tes,c)
plt.figure(1)
```

```
    X= [0:12]
    Y1= [Y1:0:12*1,:]
    Y2= [Y2:12*1:12*2,:]
    Y3= [Y3:12*2:12*3,:]
    Y4= [Y4:12*3:12*4,:]
    Y5= [Y5:12*4:12*5,:]
    Y6= [Y6:12*5:12*6,:]

    plt.plot(X_tes,Y_tes,linewidth=6)
    plt.plot(X,Y1)
    plt.plot(X,Y2)
    plt.plot(X,Y3)
    plt.plot(X,Y4)
    plt.plot(X,Y5)
    plt.plot(X,Y6)
    plt.show()
```

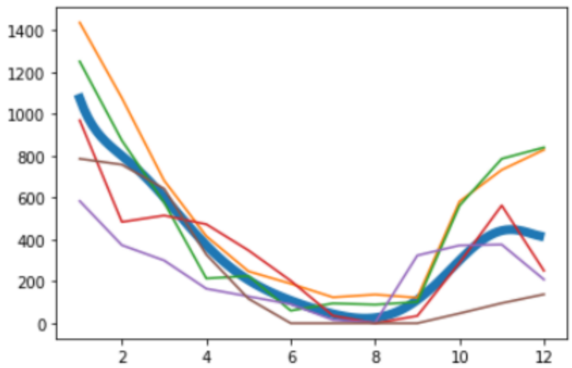

Figure 4.5: The fitting results of the data to the polynomial model shown by the thick blue line in the graph.

“The roots of education are bitter, but the fruit is sweet.”

– Aristotle –

CHAPTER 5

Artificial Neural Network

5.1 LINEAR REGRESSION VS ARTIFICIAL NEURAL NETWORK

1. *L*inear Regression: Prediction of Continuous Time series Data
2. Artificial Neural Network: Prediction of Discrete Data

5.2 REPRESENTATION OF LINEAR EQUATIONS IN DIAGRAM FORM

$$y = c_0 + c_1 x_1 + c_2 x_2 + \cdots + c_m x_m \tag{5.1}$$

can be represented in diagram form

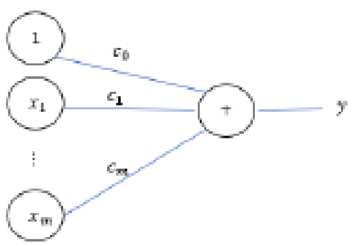

Figure 5.1: Linear Equation

If a linear activation function is added, it becomes

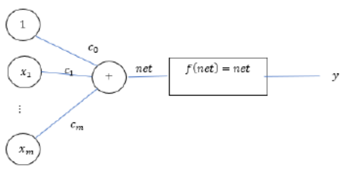

Figure 5.2: Linear Equation

5.3 PERCEPTRON NEURAL NETWORK

There are three elements of the simplest ann consisting of:

1. Links or synapses:
   (a) Links or synapses connect the input to the summing machine.
   (b) Each such link carries a weight w or gain to scale the corresponding input.
2. Neurons: To sum the inputs that have been multiplied by the weights.
3. Activation: To limit the output of the neuron.

No	x_1	x_2	Target (y)
1	x_{11}	x_{12}	y_1
2	x_{21}	x_{22}	y_2

Table 5.1: Table

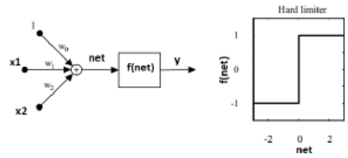

Figure 5.3: Perceptron Network

Example
The image above is a perceptron network with:

- One output node.
- Three input vectors x_1, x_2 and an input that is always one.
- Three weights w_2, w_1 and w_0.
- One hard limit activation function to limit the output of the perceptron network.

The steps to calculate the output of a perceptron network are:

1. Suppose the weights of the network are: $w_0 = 0.5$, $w_1 = 0.5$ and $w_2 = -0.4$.
2. Input values are $x_1 = 0.5$, $x_2 = 0.5$.
3. The activation function is a hard limit.

Then the output of the perceptron network can be calculated as follows:
1. Calculate the sum of all inputs that have been weighted.

$$net = w_0 + x_1 w_1 + x_2 w_2$$

$$= 0.5 + 0.5 \times 0.5 - 0.4 \times 0.5$$

$$= 0.5 + 0.25 - 0.2 = 0.55$$

2. Calculate the network output $y = f(\text{net})$. Figure **??** applies the Hard Limit activation function to obtain the network output.

$$f(\text{net}) = \begin{cases} 0 & \text{if net} \leq 0 \\ 1 & \text{if net} \geq 0 \end{cases}$$

Then we find

$$y = f(0.55) = 1$$

5.4 ACTIVATION FUNCTION

1. Sigmoid : Output ranges from 0 to 1

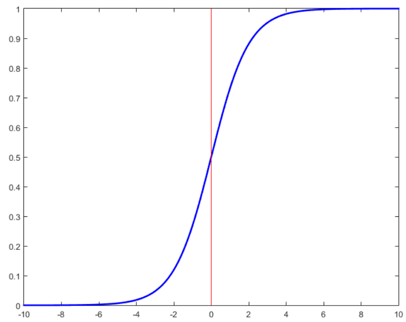

Figure 5.4: Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

2. Rectified Linear units(ReLU) Function Rectified Linear units or ReLU is widely used as an activation function because it converges faster.

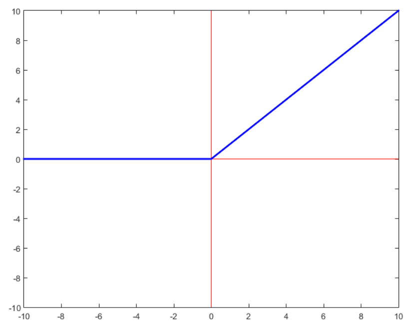

Figure 5.5: Relu

$$f(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{5.3}$$

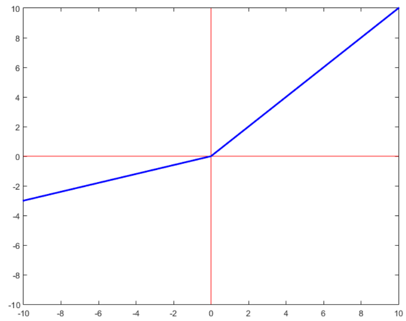

Figure 5.6: Leakyrelu

3. LeakyReLU

$$f(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases} \tag{5.4}$$

4. Hyperbolic Tangent Function

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{5.5}$$

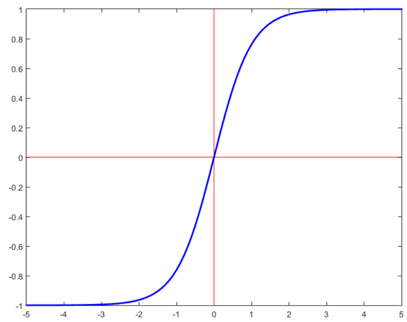

Figure 5.7: Tangen Hiperbolic

5. Softplus

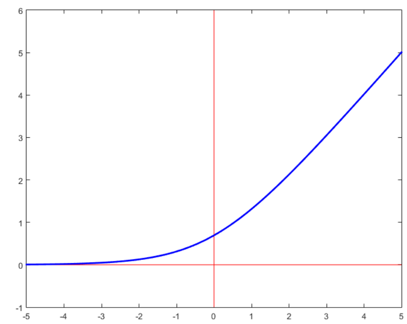

Figure 5.8: Softplus

$$f(x) = In(1 + e^x) \tag{5.6}$$

5.5 TRAINING ANN PERCEPTRON

ANN learning is the tuning of weights based on historical data until it reaches the expected output.

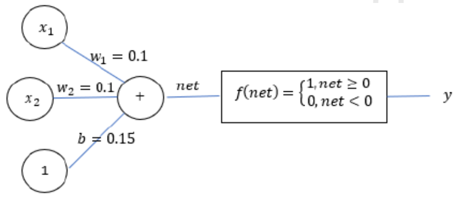

Figure 5.9: Training ANN Perceptron

The steps are as follows:

1. For Data No 1: $x_1 = x_{11}$, $x_2 = x_{12}$, and $t = t_1$
   - Calculate the weighted sum of inputs:
   $$\text{net}_1 = b + w_1 x_{11} + w_2 x_{12}$$
   - Calculate the output by applying the activation function to net_1:
   $$y_1 = f(\text{net}_1)$$
   - Calculate the propagation error:
   $$e_1 = t_1 - y_1$$
   - Fixing the weights:
   $$\Delta w_1 = \alpha x_{11} e$$
   $$\Delta w_2 = \alpha x_{12} e$$
   $$\Delta b = \alpha e_1$$
   - Applying the new weights:
   $$w_1 = w_1 + \Delta w_1$$
   $$w_2 = w_2 + \Delta w_2$$
   $$b = b + \Delta b$$
2. For Data No 2: $x_1 = x_{21}$, $x_2 = x_{22}$, and $t = t_2$
   - Calculate the weighted sum of inputs:
   $$\text{net}_2 = b + w_1 x_{21} + w_2 x_{22}$$

$$\Delta w_1 = \alpha x_{21} e2$$
$$\Delta w_2 = \alpha x_{22} e2$$
$$\Delta b = \alpha e_2$$

- Calculate the output by applying the activation function on net$_2$:

$$y_2 = f(\text{net}_2)$$

- Calculating the propagation error:

$$e_2 = t_2 - y_2$$

- Fixing the weights:
- Applying the new weights:

$$w_1 = w_1 + \Delta w_1 1$$
$$w_2 = w_2 + \Delta w_2 1$$
$$b = b + \Delta b$$

3. Calculating Loss:

$$\text{Loss} = |e_1| + |e_2|$$

4. Repeat step 1 until Loss $= 0$

```
import matplotlib.pyplot as plt
import numpy as np
x1 = np.array([1 ,0])
x2 = np.array([0 ,1])
t  = np.array([0 ,1])
#Determining the Learning Rate
alpha = 0.5
#weight initialization
w1 =0.5
w2 =0.5
b=0.2
loss =[];
for epoh in range(5):
    loss.append(0)
    for NoData in range(2):
        #Calculating the weighted sum of inputs
        net = x1[NoData]*w1+x2[NoData]*w2+b
        #Apply the activation function to obtain the output y
        if net>=0:
            y=1
        else:
            y=0
        #Calculating Propagation Error
        e=t[NoData]-y
        #Calculating Weight Change
```

```
        dw1 = e*alpha*x1[NoData]
        dw2 = e*alpha*x2[NoData]
        db = e*alpha

        #Updating the weight
        w1 =w1+dw1
        w2 =w2+dw2
        b=b+db
        loss[epoh]=loss[epoh]+np.abs(e)
#Show Loss Graph
plt.plot(loss)
```

5.6 MULTIPLE OUTPUT PERCEPTRON TRAINING AND PREDICTION

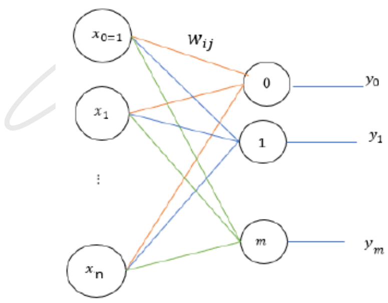

Figure 5.10: Multi output perceptron ANN architecture

No	x_0	x_1	x_2	...	x_n	t_1	t_2	...	t_m
1	x_{10}	x_{11}	x_{12}	...	x_{1n}	t_{11}	t_{12}	...	t_{1m}
2	x_{20}	x_{21}	x_{22}	...	x_{2n}	t_{21}	t_{22}	...	t_{2m}
3	x_{30}	x_{31}	x_{32}	...	x_{3n}	t_{31}	t_{32}	...	t_{3m}
...	...	...	...	...	...	...	...	...	...
N	x_{N0}	x_{N1}	x_{N2}	...	x_{Nn}	t_{N1}	t_{N2}	...	t_{Nm}

Table 5.2: Data Table

The representation of the Multiple Output Perceptron ANN in matrix form is:
1. **Input Matrix** X:

$$X = \begin{bmatrix} x_{00} & x_{10} & \cdots & x_{N0} \\ x_{01} & x_{11} & \cdots & x_{N1} \\ x_{02} & x_{12} & \cdots & x_{N2} \\ \cdots & \cdots & \cdots & \cdots \\ x_{0n} & x_{1n} & \cdots & x_{Nn} \end{bmatrix}$$

or

$$x_i = \begin{bmatrix} x_{i0} & x_{i1} & x_{i2} & \cdots & x_{in} \end{bmatrix}$$

2. **Output Matrix** T:

$$T = \begin{bmatrix} t_{00} & t_{10} & \cdots & t_{N0} \\ t_{01} & t_{11} & \cdots & t_{N1} \\ t_{02} & t_{12} & \cdots & t_{N2} \\ \cdots & \cdots & \cdots & \cdots \\ t_{0m} & t_{1m} & \cdots & t_{Nm} \end{bmatrix}$$

or

$$t_i = \begin{bmatrix} t_{i0} & t_{i1} & t_{i2} & \cdots & t_{im} \end{bmatrix}$$

where N is the amount of data.

3. **Weight Matrix** W:

$$W = \begin{bmatrix} w_{00} & w_{10} & w_{20} & \cdots & w_{m0} \\ w_{01} & w_{11} & w_{21} & \cdots & w_{m1} \\ w_{02} & w_{12} & w_{22} & \cdots & w_{m2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{0n} & w_{1n} & w_{2n} & \cdots & w_{mn} \end{bmatrix}$$

5.6 MULTIPLE OUTPUT PERCEPTRON LEARNING

Furthermore, in this example, a sigmoid function will be applied to obtain the output. Then learning a multi-output perceptron ANN follows the following steps:

1. Calculating the sum of multiplication of input i with weights W:

$$\text{net}_i = x_i W$$

2. Calculating y by applying the sigmoid function:

$$y_i = \frac{1}{1 + e^{-\text{net}}}$$

3. Calculating the propagation error:

$$e_i = T_i - y_i$$

4. Calculating the weight change based on the propagation error: where α is the learning

$$\Delta \mathbf{w} = \alpha \mathbf{x}_i^T e_i$$

rate.

5. Weight Update:

$$\mathbf{W} = \mathbf{W} + \Delta \mathbf{w}$$

6. Repeat steps 1-5 until Propagation error $= 0$.

Predict Data Using Weights obtained from ANN training:

- Multiplication between input vector and weights:

$$\text{net} = XW$$

- Calculate the output y by applying the sigmoid function:

$$y = \frac{1}{1 + e^{-\text{net}}}$$

In the following example, two functions are created for:

- Perceptron_Training(X, Y) function
- Prediction(xp, yp) function

```
###############################################
## Module
###############################################
import matplotlib.pyplot as plt
import numpy as np
#Create a Function for Training Perceptron with Multiple Input
   and Output vectors
def TrainingPerceptron(x,T):
    #Steps :
    #  1. Data Initialization
    TotalData=x.shape[0]
    TotalInputVector =x.shape[1]
    TotalOutputVector= T.shape[1]
    #2.  Weight Initialization With a random number with a
       maximum value of 0.5
    w=np.random.rand(TotalInputVector,TotalOutputVector)*0.5
    #3.  Specify a Learning Rate of 0.5
    alpha = 0.5
    loss = []
    #4. Iterate for weight tuning 1000 times
    for epoh  in range(1000):
        loss.append(0)
    #5. Weight Tuning For each data
        for NData in range(JumlahData):
            xi = x[NData:NData+1,:]
            net = np.matmul(xi,w)
            #Apply sigmoid activation function to find output
            y = 1/(1 + np.exp(-net))
            #Calculating Propagation Error
            e = T[NData:NData+1,:]-y
            #Calculating Weight Change
            dw = alpha*np.matmul(xi.transpose(),e)
            w=w+dw
            #Calculating Loss with sum error function
            loss[epoh]=loss[epoh] +np.sum(np.abs(e))
    return w,loss
```

```

#Create data prediction function xp based on weight w of
    ANNlearning result
def Prediction(xp,w):
    net = np.matmul(xp,w)
    #Apply sigmoid activation function
    yp =np.array( 1/(1 + np.exp(-net)))
    return yp;
```

Next, we make the above program into a module so that it can be reused to predict other data.

```

##############################
# Main Program
##############################

X =np.array([[1,1,0,0,1],#Kelas [0,1]
             [1,1,0,0,0],#Kelas [1,0]
             [1,0,1,1,0],#Kelas [0,1]
             [1,0,0,1,0],#Kelas [1,0]
             [1,0,0,1,1],#Kelas [0,1]
             [1,0,1,0,0],#Kelas [1,0]
             [1,0,1,1,0]])#Kelas [0,1]
T =np.array([[0,1],
            [1,0],
            [0,1],
            [1,0],
            [0,1],
            [1,0],
            [0,1]])

#Performing Perceptron Learning
w,loss = TrainingPerceptron(X,T)

#Display Loss Graphics
plt.plot(loss)
```

```
#***********************************************
#2. Prediction based on training result weight
#***********************************************
xp=np.array([[1,0,1,1,1]])
yp = Prediction(xp,w)

print("Prediction Result")
print(yp)
yp = yp>0.5
yp=np.array(yp, dtype=int)
print(yp)
```

```
x=np.array([[1,1,0],
            [1,0,1],
            [1,0,0]])
```

```
y=np.array([[1,0],[1,0],[0,1]])
w,loss =TrainingPerceptron(x,y)
plt.plot(loss)

xp =np.array([[1,0,0]])
yp = Prediction(xp,w)
print(yp)
```

“The roots of education are bitter, but the fruit is sweet.”

– Aristotle –

CHAPTER 6

Keras Python Library for Artificial Neural Networks

6.1 KERAS MODEL

1. *R*epresents the actual neural network model.
2. This model groups layers into objects.
    (a) There are two types of Keras Models:

- Perceptron_Training(X, Y) function
- Prediction(xp, yp) function

6.1.1 Keras Sequential Model

The Sequential model is suitable for regular layers where each layer has exactly one input and one output.

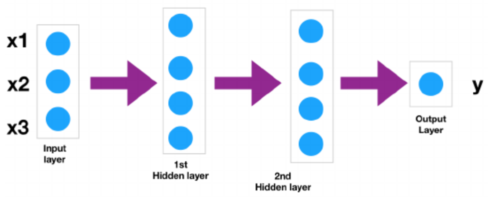

Figure 6.1: Keras Sequential Model

Creating a Sequential Model
Next, create a sequential model according to the figure above.

1. **Input layer**: input vector three (x_1, x_2, x_3)
2. **Hidden layer**: 2 each with 4 nodes
3. **Output layer**: output vector 1 (y)

```python
#Preparing keras library to create Sequential models
#Preparing keras library for Dense layer
# The dense layer is used to create the hidden layer

from keras.models import Sequential
from keras.layers import Dense

## Building a Neural Network
model = Sequential()
## Define input and two hidden layers
model.add(Dense(4, input_dim=3,activation='relu'))
model.add(Dense(4, activation='relu'))
## Define the output layer
model.add(Dense(1, activation='sigmoid'))
# summarize layers
print(model.summary())
```

Result Model "Sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 4)	16
dense_1 (Dense)	(None, 4)	20
dense_2 (Dense)	(None, 1)	5

- Total params: 41 (164.00 B)
- Trainable params: 41 (164.00 B)
- Non-trainable params: 0 (0.00 B)

6.1.2 Functional Keras Model

The functional Keras model is more flexible than the sequential model because it can be used to build more complex models with many inputs and outputs.

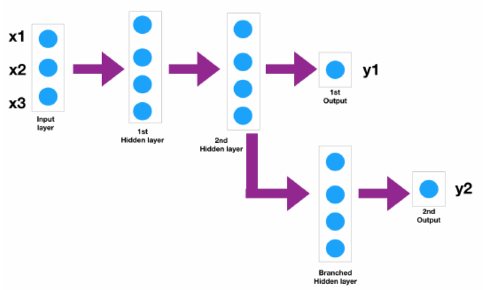

Figure 6.2: Keras Sequential Model

```python
from keras.layers import Input
from keras.layers import Dense

#Building a neural network
#Using 2 hidden layers and one branching layer each with 10 neurons
##Define the input layer
```

```
input_layer = Input(shape=(3,),name='input_layer')
##Define two hidden layers
Layer_1 = Dense(10, activation="relu",name='Layer_1')(input_layer
    )
Layer_2 = Dense(10, activation="relu",name='Layer_2')(Layer_1)
##Define output layer y1
y1_output= Dense(1, activation="linear",name='y1_output')(Layer_2
    )
##Define the branch layer
Branched_layer=Dense(10, activation="relu",name='Branched_layer')
    (Layer_2)
##Define the second output layer
y2_output= Dense(1, activation="linear",name='y2_output')(
    Branched_layer)

## Define the model by specifying input and output layers

model = Model(inputs=input_layer,outputs=[y1_output,y2_output])

# summarize layers
print(model.summary())
```

Result Model "Functional3"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 4)	16
dense_1 (Dense)	(None, 4)	20
dense_2 (Dense)	(None, 1)	5

Table 6.1: Model Summary

Total Parameters

- Total params: 41 (164.00 B)
- Trainable params: 41 (164.00 B)
- Non-trainable params: 0 (0.00 B)

6.2 ANN TRAINING AND PREDICTION USING KERAS MODEL

The steps in training and predicting the keras model are as follows:

1. *P*repare Input and Output datasets

```
# ####################################################
# Setting up the Data Set
# Input X  dan Target Y
# ====================================================
import numpy as np
import matplotlib.pyplot as plt
```

```
from keras.models import Sequential
from keras.layers import Dense
X=np.array([[0,0],
            [0,1],
            [1,0],
            [1,1]])
Y=np.array([[1],
            [0],
            [0],
            [0]])
print(X.shape)
print(Y.shape)
```

(a) Defining the ANN model:

```
model = Sequential()
#. Adding Dense Layer to Model
NNode = 1 # Number of Node.
NInput = 2# Number of input vectors
model.add(Dense(NNode, input_dim=NInput, activation='
   sigmoid',use_bias=True))
print(model.summary())
    \end
        \end{enumerate}
    \item Defining the ANN model:
        \begin{enumerate}
\begin{lstlisting}
model = Sequential()
#. Menambahkan Layer Dense ke Model
NNode = 1 # Number of Node.
NInput = 2# Number of input vectors
model.add(Dense(NNode, input_dim=NInput, activation='
   sigmoid',use_bias=True))
print(model.summary())
```

Total params: 3 (12.00 B)
Trainable params: 3 (12.00 B)
Non-trainable params: 0 (0.00 B)

Compiling the model :

i.
- **Loss**: Objective function to optimize the score.
- **Optimizer**: A method used to change the attributes of the deep learning model such as weights, learning rate, to reduce the loss and get faster results.
- **Metrics**: A function used to assess the performance of the model.

```
    #3. Compile model
    #Parameter Training
    # Fungsi loss = mse
    # Optimiser : SGD Stochastic Gradien Descent
    # metrics :accuracy
model.compile(loss='mse',optimizer='SGD',metrics=['
   accuracy'])
```

- Training and Prediction Keras Model
- Model learning is performed after the model is compiled in a NumPy array. The array consists of inputs and labels
- Keras, learning uses the fit() method
- In Keras, learning uses the fit() method

```
#4. Training Model using 200 epochs
# X : Input Data
# Y : Target
# epoch : Number of iterations = 200
His=model.fit(X, Y,epochs=200)
#Display Loss Graphics
plt.plot(His.history['loss'])
#5. Prediction
yp=model.predict(X)
#Printing the prediction result
print("Prediction Result X")
print(yp)
```

6.3 MULTILAYER PERCEPTRON

```
###################################################
# File Name :Single Output Perceptron With Keras
#==================================================
# This example program requires functions contained in
    several libraries
# library numpy: array function to turn the list into an
    array
# Keras.model library: sequential() function
#                      : add() function to add layers to
    the model
# Keras Layer library: dense() function to create a dense
     layer on the model
#--------------------------------------------------
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
###################################################
# Preparing the DataSet
# Input X and Target T
#===================================================
#Step 1: prepare input and output datasets
X=np.array([[0,0],
            [0,1],
            [1,0],
            [1,1]])

T=np.array([[1],
```

```
              [0],
              [0],
              [0]])

#####################################################
# Defining ANN Model with Output 1 and input 2
# In ANN We applied sigmoid activation function
#On the node in the ANN is added bias
#Number of hidden layers 3 with each node 1000
#Relu activation unit except the last node sigmoid to
   limit the output between 0 and 1
#==================================================
#Step 2: define the ANN model
model = Sequential()

NInput = X.shape[1] # Total of input vectors
NNOutput = T.shape[1] # Total of Nodes.
#Adding a Dense Layer to the Model
model.add(Dense(1000, input_dim=NInput, activation='relu'
   ,use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(NNOutput,  activation='sigmoid',use_bias=
   True))

#3. Compile model
#Parameter Training
# Loss Function = mse
# Optimiser : SGD Stochastic Gradien Descent
# metrics :accuracy
model.compile(loss='mse',optimizer='SGD',metrics=['
   accuracy'])


#4. Training Model using 1000 epochs
# X : Input Data
# T : Target
# epoch : Number of iterations = 1000
His=model.fit(X, T,epochs=1000)
#Display Loss Graphics
plt.plot(His.history['loss'])
print(model.summary())

#5. Predictions
yp=model.predict(X)
#printing the prediction result
print(yp)
```

6.4 ACTIVITY CREATE MLP TRAINING PREDICTION USING KERAS

6.4.1 Trying MLP with Keras

Model design for prediction of the following data pairs

No Data	x_1	x_2	x_3	x_4	y_1	y_2	y_3
1	1	1	0	0	0	1	0
2	0	1	1	0	0	1	0
3	0	0	1	1	0	1	0
4	0	1	1	1	0	0	1
5	1	1	1	0	1	0	0
6	0	1	1	1	0	0	1
7	1	0	1	1	0	0	1
8	0	1	0	1	0	1	0
9	1	0	1	0	1	0	0
10	0	0	1	0	1	0	0

The data table above classifies $X = \{x_1, x_2, x_3, x_4\}$ into three classes $Y = \{y_1, y_2, y_3\}$ with the following conditions:

i. If only one of x's is 1, then it will be in class 1 $(1, 0, 0)$.
ii. If any two of x's are 1, then it will be in class 2 $(0, 1, 0)$.
iii. If any three of x's are 1, then it will be in class 3 $(0, 0, 1)$.

The desired architecture of our MLP is as follows:

i. Input: X and Output: Y.
ii. Number of hidden layers: 5.
iii. The number of nodes in each hidden layer: 100.
iv. The activation function used in the hidden layers: ReLU.
v. The activation function used in the output layer: Sigmoid.
vi. The number of epochs during training: 1000.
vii. During training, use the weights that have been trained.
viii. After training, predict for $x = [[0, 1, 1, 0]]$.

Setting up input and output datasets

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
from keras.models import load_model

#####################################################
# Preparing the Datasets
# Input X and Target T
#====================================================
#Step 1: prepare input and output datasets
##############################################
## Input and Target here
```

```
#############################################
X=np.array([[?,?,?,?],
            [?,?,?,?],
            [?,?,?,?],
            .........
            [?,?,?,?]])

T=np.array([[?,?,?,?],
            [?,?,?,?],
            [?,?,?,?],
            .........
            [?,?,?,?]])
```

Creating the ANN Model

```
#Step 2: Model the ANN model
model = Sequential()
NInput = X.shape[1] # Number of input vectors
NNOutput = T.shape[1] # Number of Node.
#Add a Dense Layer to the Model
model.add(Dense(1000, input_dim=NInput, activation='relu'
   ,use_bias=True))
###################################################
## Add a hidden layer here
##################################################
?
?
?
?
##model.add(Dense(1000,  activation='relu',use_bias=True)
   )
model.add(Dense(NNOutput,  activation='sigmoid',use_bias=
   True))

#3. Compile the model
model.compile(loss='mse',optimizer='SGD',metrics=['
   accuracy'],shuffle=True)
print(model.summary())
```

Training the ANN

```
#4. Training Model using 1000 epoch
# X : Input Data
# T : Target
# epoh : Number of iterations = 1000
His=model.fit(X, T,epochs=1000)
#Save the model and weights to a file with the name
     weights . h 5
model.save("weights.h5")
#Display Loss Graph
plt.plot(His.history['loss'])
```

Making Predictions

```
#5. Prediction
```

```
model=load_model("wights.h5")
yp=model.predict(X)
#printing the prediction result
print(yp)
```

6.5 CREATING A DEEP LEARNING MODULE FOR ANN MLP TRAINING AND PREDICTION

Next, we will make the program that we have created into a module so that it can be used many times.

6.5.1 Creating ModuleDeepLearningMLP

Create this program and save it with the name ModuleDeepLearningMLP.py
The program can be downloaded at here

```
#Save this program with the name "ModulDeepLearningMLP.py
   "
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
from keras.models import load_model

def Training(X,T, JumEpoh,NamaFileBobot):
    #==================================================
    #Step 2: defining the ANN model
    model = Sequential()

    NInput = X.shape[1] # Number of input vectors
    NNOutput = T.shape[1] # Number of Nodes.
    #Adding a Dense Layer to the Model
    model.add(Dense(1000, input_dim=NInput, activation='
       relu',use_bias=True))
    model.add(Dense(1000,  activation='relu',use_bias=
       True))
    model.add(Dense(1000,  activation='relu',use_bias=
       True))
    model.add(Dense(NNOutput,  activation='sigmoid',
       use_bias=True))
    model.compile(loss='mse',optimizer='SGD',metrics=['
       accuracy'])
    His=model.fit(X, T,epochs=JumEpoh)
    plt.plot(His.history['loss'])
    print(model.summary())
    model.save(NamaFileBobot)
    return model,His

```

```
def Prediction(X):
    model=load_model(NamaFileBobot)
    yp=model.predict(X)
    return yp
```

6.5.2 Using DeepLearning MLP Module

Make sure MoudlDeepLarningMLP.py has been uploaded to google colab If you haven't already, download it at here then uploaded to google colab, and If it has been uploaded it will appear in the google colab directory

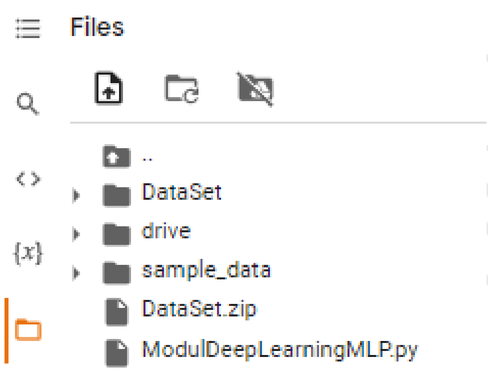

Figure 6.3: Google Colab Directory

```
import ModulDeepLearningMLP as DM
import numpy as np
X= np.array([[1,0,0],
             [0,1,0],
             [0,0,1],
             [1,1,0],
             [0,1,1]])
T = np.array([[1,0],
              [1,0],
              [1,0],
              [0,1],
              [0,1]])
DM.Prediction(X,T,1000,"Bobot_752.h5")
```

```
import ModulDeepLearningMLP as DM
import numpy as np
X= np.array([[1,0,0],
             [0,1,0],
             [0,0,1],
             [1,1,0],
             [0,1,1]])
DM.Prediction(X,"Bobot_752.h5")
```

CHAPTER 7

Feed forward Deeplearning Using Keras Library Keras

Feedforward neural networks or Multi-layered Network of Neurons (MLN).
Deep learning: Number of Hidden layers equal or more than three

7.1 FEEDFORWARD NEURAL NETWORK

- It is called feedforward because the information only moves forward through the input nodes, then forwarded to the hidden layer, and finally to the output nodes.
- There is no feedback connection.

There are three layers in a feedforward neural network:

- **Input Layer**: Input data in the form of raw data.
- **Hidden Layer**: The layer between the input layer and the output layer.
- **Output Layer**: The last layer.

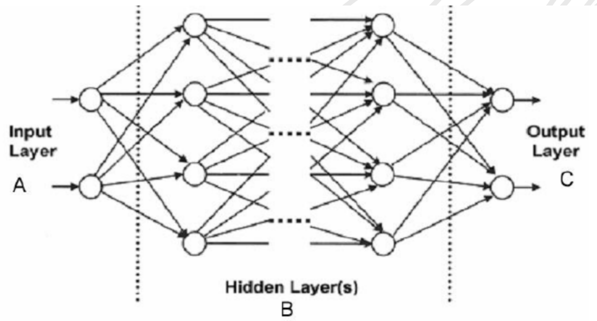

Figure 7.1: MLP

7.2 CREATE MODEL FEED FORWARD DEEP LEARNING

7.2.1 Regular feed forward

The example below is an MLP with two hidden layers. Adding a hidden layer will increase the number of features

```
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.layers import Dense
InputVectorNumber =10
OutputVectorNumber =10
model = Sequential()
model.add(Dense(1000, input_dim=InputVectorNumber, activation='linear'))
model.add(Dense(1000,  activation='linear'))
model.add(Dense(OutputVectorNumber, activation='sigmoid'))
model.compile(loss='categorical_crossentropy',optimizer='adam', metrics=['accuracy'])
print(model.summary())
plot_model(model, show_layer_activations=True)
```

7.2.2 Feed forward with deep learning

Below is a modification of the previous program. The input and output vectors have the same amount as the previous program, the number of hidden layers is increased to 5.

```
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.layers import Dense
InputVectorNumber =10
OutputVectorNumber =10
model = Sequential()
model.add(Dense(1000, input_dim=InputVectorNumber, activation='linear'))
model.add(Dense(1000,  activation='linear'))
model.add(Dense(1000,  activation='linear'))
model.add(Dense(1000,  activation='linear'))
model.add(Dense(1000,  activation='linear'))
model.add(Dense(OutputVectorNumber, activation='sigmoid'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

plot_model(model, show_layer_activations=True)
```

7.3 MLP APPLICATION FOR DIABETES CLASSIFICATION

Next we will apply Deep learning for diabetes classification. For that, make sure that the data set has been uploaded.
The files to be used are stored in the directory *DataSetDiabetes.csv*

7.3.1 Diabetes Dataset

The dataset consists of eight input variables and one output variable in the last column. A learning model will be performed to map the input variable X to the output variable y so that a function is obtained:

$$y = f(X)$$

where X is the input variable and y is the output variable.
Input variable (X):

i. x_0: Number of times pregnant
ii. x_1: 2-hour plasma glucose concentration in oral glucose tolerance test
iii. x_2: Diastolic blood pressure (mm Hg)
iv. x_3: Triceps skinfold thickness (mm)
v. x_4: 2-hour serum insulin (μU/ml)
vi. x_5: Body mass index (weight in kg/(height in m)^2)
vii. x_6: Family diabetes pedigree
viii. x_7: Age (years)

Output Variable (y): Class Variable (0 or 1)

7.3.2 Diabetes Disease Prediction Steps

1. Read Dataset
make sure the dataset has been uploaded

```
import numpy as np
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt


sf ="/content/DataSet/DataSetDiabetes/DataSetDiabetes.csv
    "
dataset = np.loadtxt(sf, delimiter=',')
```

2. Normalization of Input Data
Data for each column has a different range

X0	X1	X2	X3	X4	X5	X6	X7	Clm1
0.708333	8.291667	122	4.125	35.25	2.792361	0.1125	81	

Normalize the input data so that the maximum value of the input parameter is one.
From the data set, it is known that the maximum value for each column is : Then the data in each column must be divided by the maximum value in each column.

```
X = dataset[:,0:8]
print("Maximum value of each column before the data is normalized to the value of 1")
print(np.max(X,axis=0))
####Input normalization
X[:,0] =X[:,0]/17.0;
X[:,1] =X[:,1]/199.0;
X[:,2] =X[:,2]/122;
X[:,3] =X[:,3]/99;
X[:,4] =X[:,4]/846;
X[:,5] =X[:,5]/67.1;
X[:,6] =X[:,6]/2.42;
X[:,7] =X[:,7]/81;
print("====================================")

print("Maximum value of each column after the data is normalized to value 1")
print(np.max(X,axis = 0))
```

3. Membuat model MLP

```
#2. Creating Keras Model
print( Creating  a Model )
model = Sequential()
model.add(Dense(1000, input_dim=8, activation='relu', use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))

model.add(Dense(8,  activation='relu',use_bias=True))
model.add(Dense(1, activation='sigmoid'))
```

4. Compile the model Compile the model The parameters selected are:

i. loss=binary_crossentropy
   - ”loss binary_crossentropy” is chosen because there are two expected output conditions, namely diabetes and non-diabetes.

ii. optimizer=”adam”
   - ”adaptive moment estimation”: uses the first and second moment gradient estimates to adapt the learning rate for each neural network weight.

iii. metric = ”accuracy”

```
#3.  Compiling the Hard Model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

5. Training Data

```
#4.  Training keras model
His=model.fit(X, T, epochs=250)
```

6. Display Prediction Results

```
#5. Making Predictions
HasilPrediksi = model.predict(X)
```

7.3.3 Complete Program for Diabetes Disease Prediction

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt
##################################
##1.  Read dataset
#################################
sf ="/content/DataSet/DataSetDiabetes/DataSetDiabetes.csv
   "
dataset = loadtxt(sf, delimiter=',')
## Split the dataset into input (X) and Target (T)
   variables
X = dataset[:,0:8]
T = dataset[:,8]
#################################
##2. Normalization of input parameters
#################################
XMax = np.max(X,axis=0)
X[:,0] =X[:,0]/17;
X[:,1] =X[:,1]/199;
X[:,2] =X[:,2]/122;
X[:,3] =X[:,3]/99;
X[:,4] =X[:,4]/846;
X[:,5] =X[:,5]/67.1;
X[:,6] =X[:,6]/2.42;
X[:,7] =X[:,7]/81;
##############################
##3. Creating a Sequential Model
##############################
model = Sequential()
model.add(Dense(1000, input_dim=8, activation='relu',
   use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(1000,  activation='relu',use_bias=True))
model.add(Dense(8,  activation='relu',use_bias=True))
model.add(Dense(1, activation='sigmoid'))
###############################
#4.  Compiling the Keras Model
###############################
```

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

##############################
#5.  Training model
##############################
His=model.fit(X, T,validation_split=0.33, epochs=150)

##############################
#Display Loss Graphics
##############################
plt.plot(His.history['loss'])
plt.plot(His.history['acc'])

##############################
#6. Making Predictions
##############################
HasilPrediksi = model.predict(X)

##############################
#Display Results
##############################
Prediction = (ResultPrediction > 0.5).astype(int)

for i in range(50):
    print('%s => %d (Expectation %d)' % (X[i].tolist(),
       Prediction[i], T[i]))
```

7.4 APPLICATION OF MLP FOR SPEECH RECOGNITION

7.4.1 Voice Dataset

i. Sound in the form of *.wav* files.
ii. Classified in two classes: “Open” and “Close”.
   - The sound dataset for the word “Open” is stored in the directory `SoundDataset\Open`.
   - The sound dataset for the word “Close” is stored in the directory `SoundDataset\Close`.

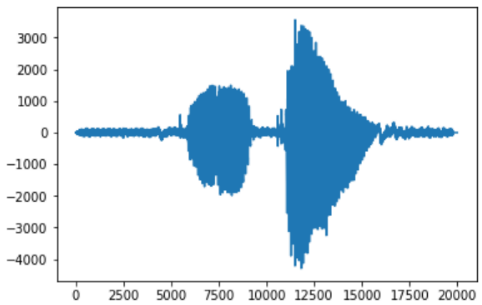

Figure 7.2: Signal form for the word “Open”

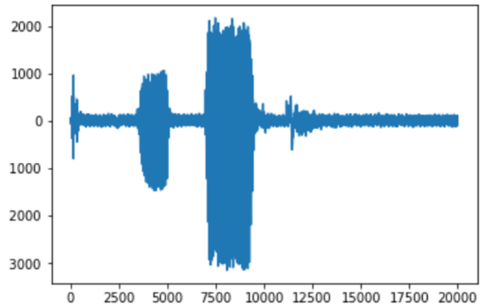

Figure 7.3: Close Word Form

7.4.2 Reading Voice Data Files with WAV Format

```python
import matplotlib.pyplot as plt
from scipy.io import wavfile
sNamaFileWav ="/content/DataSet/VoiceDataSet/open/Open
    (2).wav"
samplerate,data=wavfile.read(sNamaFileWav)
plt.plot(data)
```

7.4.3 Voice Dataset Training

Step 1: Deeplearning Architecture Design Step 2: Voice Data Training The

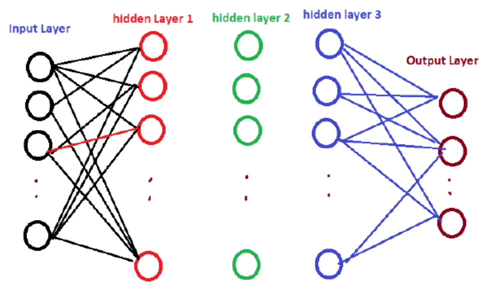

Figure 7.4: Deeplearning Architecture Design

steps taken at this stage are:

i. **Preparing the Dataset for Training**
   In this example, the data set to be trained is divided into two classes:
   - **Open Class**: Sound files in `.wav` format are stored in the `SoundDirectory\Open` directory.
   - **Close Class**: Sound files in `.wav` format are stored in the `Sound\Close` directory.

ii. **Loading Training Data to Variables**
   Loading training data to variables is done in two stages:
   - The first stage is to read the sound and convert it into a vector with the number of input vectors according to the MLP model design.
     – Next, arrange the vector into training data.
     – **Define the MLP Model**: The model consists of 5 layers consisting of:

- One input layer
- Three hidden layers
- One output layer

iii. **Performing Training**
Training is done with the `fit()` function with parameters *X* (training data), *T* (training target), and `epochs` (the number of iterations for training).

```
##############################################
#File Name: TrainingMLPSuara.by
#Dataset Directory : DatasetSuara

import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.io import wavfile
from keras.models import Sequential
from keras.layers import Dense
def ReadFiledWav( sName,InputVectorNumber):
    print(sName)
    samplerate, data = wavfile.read(sName)
    JumData = data.shape[0]
    ########################
    # Resize data to SumVectorInput
    #------------------------
    Vi =np.zeros([InputVectorNumber])
    st=JumData / InputVectorNumber;
    for i in range( InputVectorNumber):
        n= np.int(i*st)
        Vi[i]= data[n]
       #-----end for----------
    return Vi,data,samplerate

def LoadingDataset(sDir,LabelClass,InputVectorNumber):
  n=len(LabelClass)
  TargetClass = np.eye(n)
  X=[]
  T=[]
  for i in range(n):
    DirClass = os.path.join(sDir, LabelClass[i])
    files = os.listdir(DirClass)
    sd =DirClass

    for sName in files:
        ff=sName.lower()
        if (ff.endswith('.wav')):
            sf = sd+"/"+sName

            Vi, data,samplerate = ReadFiledWav(sf,
               InputVectorNumber)
            X.append(Vi)
            T.append(TargetClass[i])
  #Normalize the maximum X to 1
  X = np.array(X)/np.max(np.abs(X))
```

```
  T =np.array(T)
  return X,T

def MembuatModelMLP(InputVectorNumber,NumberofClasses):
    model = Sequential()
    model.add(Dense(1000, input_dim=InputVectorNumber,
       activation='linear'))
    model.add(Dense(1500,  activation='linear'))
    model.add(Dense(1000,  activation='linear'))
    model.add(Dense(JumlahKelas, activation='sigmoid'))
    model.compile(loss='categorical_crossentropy',
       optimizer='adam', metrics=['accuracy'])
    return model


##################################################
# Main Program
#--------------------------------------------------
#1 Training voice data

#Specify the Yant Dataset Directory to be placed in the
   Sound Dataset directory
sDir = "/content/DataSet/DataSetSuara"
LabelClass=("buka","tutup")
NumberofClasses = 2
#Determining the Number of Input Vectors
InputVectorNumber = 10000
#Loading Data Set
X,T = LoadingDataset(sDir,LabelClass,InputVectorNumber)

#############################################
#Training Model
#--------------------------------------------
model = CreatingMLPModels(InputVectorNumber,
   NumberofClasses)
his=model.fit(X, T,shuffle=True, epochs=10)
#Save the model to file
model.save("WeightVoice.h5")
plt.figure(1)
plt.plot(his.history['loss'])

plt.ylabel('loss/acc')
plt.xlabel('epoch')
plt.show()
```

7.4.4 Classification of Voice Data

```
import numpy as np
from scipy.io import wavfile
from keras.models import load_model
import os

```

```
def ReadFiledWav( sName,InputVectorNumber):
    samplerate, data = wavfile.read(sName)
    JumData = data.shape[0]
    ########################
    # Resize data to SumVectorInput
    #-----------------------
    Vi =np.zeros([InputVectorNumber])
    st=JumData / InputVectorNumber;
    for i in range(InputVectorNumber):
        n= np.int(i*st)
        Vi[i]= data[n]
      #-----end for----------
    return Vi,data,samplerate
def LoadingDataset(sDir,DirektoryDataTes,
   InputVectorNumber):
    DirKelas = os.path.join(sDir, DirectoryDataTest)
    files = os.listdir(DirKelas)
    sd =DirClass
    X=[]
    for sName in files:
        ff=sName.lower()
        if (ff.endswith('.wav')):
            sf = sd+"/"+sName
            print(sf)
            Vi, data,samplerate = MembacaFileWav(sf,
               InputVectorNumber)
            X.append(Vi)

  #Normalize the maximum X to 1
    X = np.array(X)/np.max(np.abs(X))

    return X
########################################
# Performing Voice Classification Prediction
#--------------------------------------
sDir = "/content/VoiceDataSet"
JumlahVektorInput = 10000

model=load_model('WeightVoice.h5"')
X = LoadTestData(sDir,"tes",InputVectorNumber)

####################################
# Voice Classification Results
hs=model.predict(X)
print(hs)
```