GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

RECORD BOOK

GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

20MCA241 DATA SCIENCE LAB

Name: Riyas Iqbal

Branch: Master of Computer Applications

Semester: 3

Roll No: 44

CERTIFIED BONAFIDE RECORD WORK DONE BY

Reg No. ....

STAFF IN CHARGE

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Contents

Assignment 1
Review of python programming

Problem Statement

Write Python code to explore and practice with the basic data types, containers, functions, and classes of Python.

1. Start by creating variables of various numeric data types and assigning them values.
2. Print the data types and values of these variables.
3. Perform mathematical operations on these variables.
4. Update the values of these variables.
5. Create boolean variables with True or False values.
6. Print the data types of these boolean variables.
7. Perform Boolean operations on these boolean variables.
8. Create string variables with text values.
9. Print the contents and lengths of these string variables.
10. Concatenate strings.
11. Format strings with variables.
12. Use string methods to manipulate strings by capitalizing, converting to uppercase, justifying, centering, replacing substrings, and stripping whitespace.
13. Create and use Python lists. Perform tasks like appending elements, indexing, slicing, and iterating through the list.
14. Create and use Python tuples. Perform tasks like indexing, slicing, and concatenation.
15. Create and use Python sets. Perform tasks like accessing, adding, deleting set elements.
16. Create and use Python dictionaries. Perform tasks like adding, updating, and removing key-value pairs, and accessing values.
17. Define simple functions with parameters and return values.
18. Call functions with different arguments and use the returned results.
19. Write functions that accept other functions as arguments.

20. Define and use Python classes. Include tasks like creating a class, defining methods, and creating instances.
21. Implement class inheritance and method overriding.
22. Create a class with class variables and instance variables, and demonstrate their usage.

1.1 Basic data types

1.1.1 Numbers

```python
# Your Python code here
print("Hello, world!")
print(x + 1)   # Addition
print(x - 1)   # Subtraction
print(x * 2)   # Multiplication
print(x ** 2)  # Exponentiation
```

```
Hello, world! 7   5   14  49
```

1.1.2 Booleans

```python
t, f = True, False
print(type(t))
print(t and f) # Logical AND;
print(t or f)  # Logical OR;
print(not t)   # Logical NOT;
print(t != f)  # Logical XOR;
```

```
<class 'bool'>
False True False True
```

1.1.3 Strings

```python
hello = 'hello'
world = "world"
print(hello, len(hello))
hw = hello + ' ' + world  # String concatenation
print(hw)
hw12 = '{} {} {}'.format(hello, world, 12)  # string formatting
print(hw12)
s = "hello"
print(s.capitalize())
print(s.upper())
```

```
hello 5
hello world
hello world 12
Hello
```

```
print(s.rjust(7))
print(s.center(7))
print(s.replace('l', '(ell)'))
print('  world '.strip())
```

```
HELLO
  hello
 hello
he(ell)(ell)o
world
```

1.2 Containers

1.2.1 Lists

```
xs = [3, 1, 2]
print(xs, xs[2])
print(xs[-1])
xs[2] = 'foo'
print(xs)
xs.append('bar')
print(xs)
x = xs.pop()
print(x, xs)
```

```
[3, 1, 2] 2
2
[3, 1, 'foo']
[3, 1, 'foo', 'bar']
foo [3, 1]
```

1.2.2 Slicing

```
nums = list(range(5))
print(nums)
print(nums[2:4])
print(nums[2:])
print(nums[:2])
print(nums[:])
print(nums[:-1])
nums[2:4] = [8, 9] print(nums)
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

1.2.3 Loops

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

1.2.4 List comprehensions

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)
```

```
[0, 1, 4, 9, 16]
```

1.2.5 Dictionaries

```
d = {'cat': 'cute', 'dog': 'furry'}
print(d['cat'])
print('cat' in d)
d['fish'] = 'wet'
print(d['fish'])
```

```
cute
True
wet
```

1.2.6 Sets

```python
animals = {'cat', 'dog'}
print('cat' in animals)
print('fish' in animals)
animals.add('cat')
print(len(animals))
animals.remove('cat')
print(len(animals))
```

```
True
False
3
2
```

1.2.7 Tuples

```python
d = {(x, x + 1): x for x in range(10)}
t = (5, 6)
print(type(t))
print(d[t])
print(d[(1, 2)])
```

```
<class 'tuple'>
5
1
```

1.3 Functions

```python
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'
for x in [-1, 0, 1]:
    print(sign(x))
```

```
negative
zero
positive
```

1.4 Classes

```
class Greeter:
    def __init__(self, name):
        self.name = name
    def greet(self, loud=False):
        if loud:
          print('HELLO, {}'.format(self.name.upper()))
        else:
          print('Hello, {}!'.format(self.name))
g = Greeter('Fred')
g.greet()
g.greet(loud=True)
```

```
Hello, Fred!
HELLO, FRED
```

1.5 Class inheritance and method overriding.

```
class Animal:
def __init__(self, name):
self.name = name
def make_sound(self):
return "Generic animal sound"
class Cat(Animal):
def make_sound(self):
return f"{self.name} says Meow!"
class Cow(Animal):
def make_sound(self):
return f"{self.name} says Moo!"
generic_animal = Animal("Creature")
print(generic_animal.make_sound())
my_cat = Cat("Whiskers")
print(my_cat.make_sound())
my_cow = Cow("Bessie")
print(my_cow.make_sound())
```

```
Generic animal sound
Whiskers says Meow!
Bessie says Moo!
```

1.6 Class variables and instance variables

```python
class Car:
number_of_wheels = 4
def __init__(self, make, model):
self.make = make
self.model = model
def display_info(self):
print(f"Make: {self.make}, Model: {self.model}, Wheels:
{Car.number_of_wheels}")
car1 = Car("Toyota", "Camry")
car2 = Car("Honda", "Civic")
print(f"Car 1 instance variables: Make='{car1.make}', Model='{car1.model}'")
print(f"Car 2 instance variables: Make='{car2.make}', Model='{car2.model}'")
print(f"Class variable (using class name): {Car.number_of_wheels}")
print(f"Class variable (using instance car1): {car1.number_of_wheels}")
print(f"Class variable (using instance car2): {car2.number_of_wheels}")
Car.number_of_wheels = 3
print(f"\nAfter changing class variable:")
print(f"Class variable (using class name): {Car.number_of_wheels}")
print(f"Class variable (using instance car1): {car1.number_of_wheels}")
print(f"Class variable (using instance car2): {car2.number_of_wheels}")
car1.display_info()
car2.display_info()
```

```
Car 1 instance variables: Make='Toyota', Model='Camry'
Car 2 instance variables: Make='Honda', Model='Civic'
Class variable (using class name): 4
Class variable (using instance car1): 4
Class variable (using instance car2): 4
After changing class variable:
Class variable (using class name): 3
Class variable (using instance car1): 3
Class variable (using instance car2): 3
Make: Toyota, Model: Camry, Wheels: 3
Make: Honda, Model: Civic, Wheels: 3
```

Assignment 2
Vectorized Computations using Numpy

Problem Statement

Implement the following computations using NumPy:

1. Create a matrix U of shape (m, n) with input values where m and n are input positive integers.
2. Compute X as the transpose of U.
3. Create a matrix Y of shape (1, m) with random values $\in$ [0, 1].
4. Create a matrix W 1 of shape (p, n) with random values $\in$ [0, 1]

   where p is an input positive integer.
5. Create a vector B1 of shape (p, 1) with random values $\in$ [0, 1].
6. Create a vector W 2 of shape (1, p) with all zeros.
7. Create a scalar B2 with a random value $\in$ [0, 1].
8. Perform the following computations iteratively 15 times:

   (a) Z1 = W 1 $\cdot$ X + B1 (Matrix Multiplication)

   (b) A1 = f (Z1) where f is a function that returns 0 for negative values and the input value itself otherwise.

   (c) Z2 = W2 $\cdot$ A1 + B2

   (d) A2 = g(Z2) where g is a function defined as g(x) = 1/1+e^{-x}.

   (e) L = $\frac{1}{2}$ (A2 - Y)^2

   (f) dA2 = A2 - Y

   (g) dZ2 = dA2 $\circ$ gprime(Z2) where gprime (x) is a function that returns g(x) $\cdot$ (1 - g(x)) and $\circ$ indicates element-wise multiplication

   (h) dA1 = $W2^T$ $\cdot$ dZ2

   (i) dZ1 = dA1 $\circ$ fprime(Z1) where fprime is a function that returns 1 for positive values and 0 otherwise and $\circ$ indicates element-wise multiplication.

   (j) dW 1 = $\frac{1}{m}$ $\cdot$ dZ1 $\cdot$ X^T

   (k) dB1 = $\frac{1}{m}\Sigma$dZ1 (sum along the columns)

   (l) dW2 = $\frac{1}{m}\cdot$ dZ2 $\cdot$ $A1^T$

(m) dB2 = $\frac{1}{m}\Sigma$dZ2 (sum along the columns)

(n) Update and print W 1, B1, W 2, and B2 for $\alpha = 0.01$:

i. W1 = W1 - $\alpha \cdot$ dW1

ii. B1 = B1 - $\alpha \cdot$ dB1

iii. W2 = W2 - $\alpha \cdot$ dW2

iv. B2 = B2 - $\alpha \cdot$ dB2

2.1 Matrix creation

```python
import numpy as np
U = np.array([[1,2,3],[4,5,6]])
print(U)
```

```
[[1 2 3]
[4 5 6]]
```

2.2 Transpose

```python
X = U.T
print(X)
```

```
[[1 4]
[2 5]
[3 6]]
```

2.3 Matrix of shape(1,m)

```python
m=2
Y = np.random.random((1,m))
print(Y)
```

```
[[0.14143826 0.86003669]]
```

2.4 Matrix of shape(p,n)

```python
p=3
n=3
W1 = np.random.random((p,n))
print(W1)
```

```
[[0.21198769 0.43094504 0.44782404]
[0.65786203 0.71585282 0.21237249]
[0.24155719 0.39683022 0.27351944]]
```

2.5 Vector of shape(p,1)

```python
B1 = np.random.random((p,1))
print(B1)
```

```
[[0.21691861]
[0.38149642]
[0.0858541 ]]
```

2.6 Vector of shape(1,p)

```python
W2 = np.zeros((1,p))
print(W2)
```

```
[[0. 0. 0.]]
```

2.7 Scalar with random values

```
B2 = np.random.rand()
print(B2)
```

0.6726486910015009

2.8 Iteration

```
def f(Z1):
return np.maximum(0, Z1)
def g(Z2):
return 1 / (1 + np.exp(-Z2))
def gprime(Z2):
gz2 = g(Z2)
return gz2 * (1 - gz2)
def fprime(Z1):
return (Z1 > 0).astype(float)
num_iterations = 15
for i in range(num_iterations):
    Z1 = np.dot(W1, X) + B1
    A1 = f(Z1)
    Z2 = np.dot(W2, A1) + B2
    A2 = g(Z2)
    L = 0.5 * np.square(A2 - Y)
    dA2 = A2 - Y
    dZ2 = dA2 * gprime(Z2)
    dA1 = np.dot(W2.T, dZ2)
    dZ1 = dA1 * fprime(Z1)
    dW1 = (1/m) * np.dot(dZ1, X.T)
    dB1 = (1/m) * np.sum(dZ1, axis=1, keepdims=True)
    dW2 = (1/m) * np.dot(dZ2, A1.T)
    dB2 = (1/m) * np.sum(dZ2, axis=1, keepdims=True)
    alpha = 0.
    W1 = W1 - alpha * dW1
    B1 = B1 - alpha * dB1
    W2 = W2 - alpha * dW2
    B2 = B2 - alpha * dB2
    print(f"Iteration {i+1}:")
    print("W1:\n", W1)
    print("B1:\n", B1)
    print("W2:\n", W2)
    print("B2:\n", B2)
    print("-" * 20)
```

```
Iteration 1:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 2:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 3:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
```

```
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 4:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 5:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 6:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
```

```
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 7:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 8:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 9:
W1:
```

```
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 10:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 11:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
```

```
[[0.66731864]]
--------------------
Iteration 12:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 13:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 14:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
```

```
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
Iteration 15:
W1:
[[0.21198053 0.43094551 0.44783215]
[0.65786061 0.71585293 0.21237413]
[0.24155445 0.3968304 0.27352256]]
B1:
[[0.21692624]
[0.38149795]
[0.08585703]]
W2:
[[-0.00292056 -0.00039661 -0.00105211]]
B2:
[[0.66731864]]
--------------------
```

Assignment 3
Vectorized Computations using TensorFlow

Problem Statement

Implement the following computations using TensorFlow:

1. Create a matrix U of shape (m, n) with input values where m and n are input positive integers.
2. Compute X as the transpose of U.
3. Create a matrix Y of shape (1, m) with random values $\in$ [0, 1].
4. Create a matrix W 1 of shape (p, n) with random values $\in$ [0, 1]

   where p is an input positive integer.
5. Create a vector B1 of shape (p, 1) with random values $\in$ [0, 1].
6. Create a vector W 2 of shape (1, p) with all zeros.
7. Create a scalar B2 with a random value $\in$ [0, 1].
8. Perform the following computations iteratively 15 times:
   (a) Z1 = W 1 $\cdot$ X + B1 (Matrix Multiplication)
   (b) A1 = f (Z1) where f is a function that returns 0 for negative values and the input value itself otherwise.
   (c) Z2 = W2 $\cdot$ A1 + B2
   (d) A2 = g(Z2) where g is a function defined as g(x) = 1/1+e^{-x}.
   (e) L = $\frac{1}{2}$ (A2 - Y)^2
   (f) dA2 = A2 - Y
   (g) dZ2 = dA2 $\circ$ gprime(Z2) where gprime (x) is a function that returns g(x) $\cdot$ (1 - g(x)) and $\circ$ indicates element-wise multiplication
   (h) dA1 = $W2^T$ $\cdot$ dZ2
   (i) dZ1 = dA1 $\circ$ fprime(Z1) where fprime is a function that returns 1 for positive values and 0 otherwise and $\circ$ indicates element-wise multiplication.
   (j) dW 1 = $\frac{1}{m}$ $\cdot$ dZ1 $\cdot$ X^T
   (k) dB1 = $\frac{1}{m}\Sigma$dZ1 (sum along the columns)
   (l) dW2 = $\frac{1}{m}\cdot$ dZ2 $\cdot$ $A1^T$

(m) dB2 = $\frac{1}{m}\Sigma$dZ2 (sum along the columns)

(n) Update and print W 1, B1, W 2, and B2 for $\alpha = 0.01$:

i. W1 = W1 - $\alpha \cdot$ dW1

ii. B1 = B1 - $\alpha \cdot$ dB1

iii. W2 = W2 - $\alpha \cdot$ dW2

iv. B2 = B2 - $\alpha \cdot$ dB2

3.1 Matrix creation

```
import tensorflow as tf
m = 2 # number of examples
n = 3 # input features
p = 3 # hidden layer units
k = 10
U = tf.constant([[1, 2, 3], [4, 5, 6]], dtype=tf.float32)
print(U)
```

```
[[1 2 3]
[4 5 6]]
```

3.2 Transpose

```
X = U.T
print(X)
```

```
[[1 4]
[2 5]
[3 6]]
```

3.3 Matrix of shape(1,m)

```
Y = tf.random.uniform(shape=(1, m), minval=0, maxval=10, dtype=tf.int32)
print("Matrix Y:")
```

```
Matrix Y:
[[7 7]]
```

3.4 Matrix of shape(p,n)

```
W1=tf.Variable(tf.random.uniform((p, n), 0, 1))
print(W1)
```

```
[[0.88897145, 0.28742778, 0.6720544 ],
[0.37759733, 0.7130252 , 0.48435426],
[0.702363 , 0.24965 , 0.39652836]]
```

3.5 Vector of shape(p,1)

```
B1 =  tf.Variable(tf.random.uniform((p, 1), 0, 1))
print(B1)
```

```
[[0.39298093],
[0.20463169],
[0.7103195 ]]
```

3.6 Vector of shape(1,p)

```
W2 = tf.Variable(tf.zeros((10, p)), dtype=tf.float32)
print(W2)
```

```
[[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.]]
```

3.7 Scalar with random values

```python
B2 =  tf.Variable(tf.random.uniform((k, 1), 0, 1))
print(B2)
```

```
[[0.7250246 ],
[0.98551977],
[0.01832747],
[0.243927 ],
[0.01922894],
[0.39953363],
[0.3023355 ],
[0.33449578],
[0.09907246],
[0.34210122]]
```

3.8 Iteration

```
def relu(z):
return tf.maximum(0.0, z)
def relu_prime(z):
return tf.cast(z > 0, tf.float32)
def softmax(z):
expz = tf.exp(z - tf.reduce_max(z, axis=1, keepdims=True))
return expz / tf.reduce_sum(expz, axis=1, keepdims=True)
alpha = tf.constant(0.01)
num_iters = 15
for i in range(num_iters):
Z1 = tf.matmul(W1, X) + B1
A1 = relu(Z1)
Z2 = tf.matmul(W2, A1) + B2
Z2 = tf.transpose(Z2)
A2 = softmax(Z2)
Y_reshaped = tf.reshape(Y, [-1])
Y_onehot = tf.one_hot(Y_reshaped, depth=k)
dZ2 = A2 - Y_onehot
dW2 = (1. / tf.cast(m, tf.float32)) * tf.matmul(tf.transpose(dZ2), tf.
  transpose(A1))
dB2 = (1. / tf.cast(m, tf.float32)) * tf.reduce_sum(tf.transpose(dZ2),
  axis=1, keepdims=True)
dA1 = tf.matmul(tf.transpose(W2), tf.transpose(dZ2))
dZ1 = dA1 * relu_prime(Z1)
dW1 = (1. / tf.cast(m, tf.float32)) * tf.matmul(dZ1, tf.transpose(X))
dB1 = (1. / tf.cast(m, tf.float32)) * tf.reduce_sum(dZ1, axis=1,
  keepdims=True)
W1.assign_sub(alpha * dW1)
B1.assign_sub(alpha * dB1)
W2.assign_sub(alpha * dW2)
B2.assign_sub(alpha * dB2)
tf.print("\nIteration", i + 1)
tf.print("W1:\n", W1)
tf.print("B1:\n", B1)
tf.print("W2:\n", W2)
tf.print("B2:\n", B2)
```

```
Iteration 1
W1:
[[0.888971448 0.287427783 0.67205441]
[0.377597332 0.713025212 0.484354258]
[0.702363 0.24965 0.396528363]]
B1:
[[0.392980933]
[0.204631686]
[0.710319519]]
W2:
[[-0.00926425308 -0.00811857637 -0.00714355102]
[-0.0120210405 -0.0105344411 -0.0092692757]
4
[-0.00456978474 -0.00400465587 -0.00352370483]
...
[0.0601874068 0.0527442433 0.0464097634]
[-0.0049540787 -0.00434142537 -0.00382002885]
[-0.0063169715 -0.00553577393 -0.0048709386]]
B2:
[[0.723630548]
[0.983710885]
[0.01763984]
...
[0.34355244]
[0.098326996]
[0.341150671]]
Iteration 2
W1:
[[0.890173197 0.289141744 0.674280584]
[0.378650486 0.71452719 0.486305118]
[0.703289688 0.250971615 0.398244917]]
B1:
[[0.393493116]
[0.205080539]
[0.710714459]]
W2:
[[-0.0164473131 -0.0144220237 -0.0126971845]
[-0.0208942778 -0.0183228273 -0.0161326509]
```

```
[-0.00842497498 -0.00738648744 -0.0065022083]
...
[0.10841991 0.0950639695 0.0836901441]
[-0.00910457 -0.00798241049 -0.00702687]
[-0.0114808669 -0.0100662485 -0.00886161439]]
B2:
[[0.722495854]
[0.982298434]
[0.0170386694]
...
[0.351132214]
[0.0976790935]
[0.340341538]]
Iteration 3
W1:
[[0.891573489 0.29123497 0.677066743]
[0.379878312 0.716362536 0.488748044]
[0.704370618 0.252587408 0.400395572]]
B1:
[[0.39418605]
[0.205688104]
5
[0.711249352]]
W2:
[[-0.0212540366 -0.0186534058 -0.0164362434]
[-0.0266922172 -0.0234287959 -0.0206462033]
[-0.0111352131 -0.00977063924 -0.00860757]
...
[0.141367972 0.124059714 0.109305106]
[-0.0120093394 -0.0105378404 -0.00928360783]
[-0.015040189 -0.0131981652 -0.0116279963]]
B2:
[[0.721656382]
[0.981273413]
[0.0165757891]
...
[0.356832981]
[0.0971820429]
```

```
[0.339728445]]
Iteration 4
W1:
[[0.892793298 0.293163419 0.679703832]
[0.380948782 0.718054891 0.491062254]
[0.705313802 0.254078478 0.402434558]]
B1:
[[0.39489466]
[0.206309959]
[0.711797237]]
W2:
[[-0.0246474296 -0.0216508675 -0.0190934166]
[-0.0307533015 -0.0270177294 -0.0238290895]
[-0.0130870435 -0.0114932079 -0.0101333242]
...
[0.164828211 0.144774839 0.127662078]
[-0.0140970703 -0.0123804882 -0.0109158382]
[-0.0175817125 -0.0154419318 -0.0136160348]]
B2:
[[0.721002221]
[0.980480075]
[0.016208984]
...
[0.361307055]
[0.0967888162]
[0.339246035]]
Iteration 5
W1:
[[0.893847048 0.294910431 0.682144046]
[0.381874353 0.719589353 0.493205607]
6
[0.706129968 0.255431563 0.404324561]]
B1:
[[0.395587891]
[0.206918865]
[0.712334156]]
W2:
[[-0.0272918567 -0.0239929929 -0.0211747941]
```

```
[-0.0339099243 -0.029814804 -0.0263158437]
[-0.0146204103 -0.0128500331 -0.0113380654]
...
[0.183174655 0.161017537 0.1420912]
[-0.015735738 -0.0138306106 -0.0122035183]
[-0.0195710044 -0.0172028299 -0.0151800849]]
B2:
[[0.72045517]
[0.979819]
[0.0158995446]
...
[0.365062356]
[0.0964573845]
[0.338840634]]
Iteration 6
W1:
[[0.894781232 0.2965177 0.68442446]
[0.382695526 0.721002221 0.495210171]
[0.706854641 0.256678373 0.406093508]]
B1:
[[0.396261]
[0.207510561]
[0.712856293]]
W2:
[[-0.029488869 -0.0259426255 -0.0229104795]
[-0.036529284 -0.0321402363 -0.0283869132]
[-0.0158997886 -0.013984357 -0.0123470919]
...
[0.19844532 0.174563617 0.154146552]
[-0.0171023011 -0.0150423311 -0.013281472]
[-0.0212274622 -0.0186719969 -0.0164873917]]
B2:
[[0.719978392]
[0.979244351]
[0.0156281088]
...
[0.368344277]
[0.0961668491]
```

```
[0.338486]]
7
Iteration 7
W1:
[[0.895627916 0.298016667 0.686575651]
[0.383440346 0.722320795 0.497102499]
[0.707512319 0.25784272 0.407764524]]
B1:
[[0.39691326]
[0.208084315]
[0.713362932]]
W2:
[[-0.0313863 -0.0276287775 -0.0244135391]
[-0.0387892909 -0.0341493674 -0.0301785152]
[-0.0170081463 -0.0149685014 -0.0132237198]
...
[0.211651474 0.186295152 0.164600849]
[-0.0182857718 -0.0160932485 -0.0142176431]
[-0.0226604287 -0.0199447889 -0.0176214725]]
B2:
[[0.719552934]
[0.978732765]
[0.0153845008]
...
[0.371280253]
[0.0959062427]
[0.338168442]]
Iteration 8
W1:
[[0.896407723 0.299427748 0.688618064]
[0.384126723 0.723562837 0.498900235]
[0.708118737 0.25894013 0.409352899]]
B1:
[[0.397544563]
[0.20864]
[0.713853896]]
W2:
[[-0.0330657437 -0.0291227493 -0.025746543]
```

```
[-0.0407876261 -0.0359276198 -0.0317656659]
[-0.0179919321 -0.0158430021 -0.0140034771]
...
[0.223354667 0.1967026 0.173884258]
[-0.019335907 -0.0170267932 -0.0150500992]
[-0.0239307377 -0.0210743211 -0.0186289046]]
B2:
[[0.71916759]
[0.978270531]
[0.0151627315]
...
8
[0.373945057]
[0.0956691206]
[0.337880015]]
Iteration 9
W1:
[[0.897133946 0.300764471 0.690565288]
[0.38476631 0.724740088 0.50061512]
[0.708684146 0.259980798 0.410868853]]
B1:
[[0.398155063]
[0.209177643]
[0.714329183]]
W2:
[[-0.0345769 -0.0304680467 -0.026947733]
[-0.0425837114 -0.0375270471 -0.0331941545]
[-0.0188796259 -0.0166327506 -0.0147082079]
...
[0.233897954 0.20608604 0.182260379]
[-0.0202831943 -0.0178696122 -0.0158022307]
[-0.0250755697 -0.0220931098 -0.0195382405]]
B2:
[[0.718815207]
[0.977848709]
[0.0149589069]
...
[0.376387328]
```

```
[0.0954512879]
[0.33761546]]
Iteration 10
W1:
[[0.897815764 0.302036226 0.692427]
[0.385367036 0.725860596 0.50225544]
[0.709215403 0.260971785 0.412319541]]
B1:
[[0.39874503]
[0.209697455]
[0.714788914]]
W2:
[[-0.0359525271 -0.0316933952 -0.0280423984]
[-0.044216726 -0.0389820449 -0.034494292]
[-0.0196900293 -0.0173542034 -0.0153523758]
...
[0.243507594 0.214643732 0.189903632]
[-0.0211477522 -0.0186393186 -0.0164895188]
[-0.0261194427 -0.0230226293 -0.0203683693]]
B2:
9
[[0.71849066]
[0.97746104]
[0.0147702899]
...
[0.378641307]
[0.0952498]
[0.337371111]]
Iteration 11
W1:
[[0.898459554 0.303249836 0.694210351]
[0.385934532 0.72693032 0.503827453]
[0.709717453 0.261918247 0.413710356]]
B1:
[[0.399314821]
[0.210199714]
[0.715233266]]
W2:
```

```
[[-0.0372156 -0.0328189805 -0.0290483478]
[-0.0457142107 -0.040316835 -0.0356874615]
[-0.020436313 -0.0180189069 -0.015946148]
...
[0.252342194 0.222514912 0.196936741]
[-0.0219436735 -0.0193482712 -0.0171228461]
[-0.0270795356 -0.023877956 -0.0211325735]]
B2:
[[0.718190074]
[0.977102816]
[0.0145948352]
...
[0.380732626]
[0.095062457]
[0.337144256]]
Iteration 12
W1:
[[0.899070144 0.304410577 0.695921242]
[0.38647294 0.727953851 0.505336106]
[0.710194 0.262824118 0.415045589]]
B1:
[[0.399864972]
[0.210684836]
[0.715662599]]
W2:
[[-0.0383830667 -0.0338597223 -0.0299787614]
[-0.0470965207 -0.0415493511 -0.0367895253]
[-0.021128159 -0.018635368 -0.0164970253]
...
10
[0.26051864 0.229802355 0.203450441]
[-0.0226813219 -0.0200055763 -0.0177102461]
[-0.0279684886 -0.0246702023 -0.0218406599]]
B2:
[[0.717910528]
[0.976770282]
[0.0144309448]
...
```

```
[0.38268131]
[0.0948875323]
[0.336932719]]
Iteration 13
W1:
[[0.89965117 0.30552271 0.697564483]
[0.386985481 0.728934884 0.506785572]
[0.710647762 0.263692647 0.416328847]]
B1:
[[0.400396079]
[0.211153314]
[0.716077328]]
W2:
[[-0.0394679308 -0.0348270871 -0.0308437888]
[-0.0483793132 -0.0426934175 -0.0378127284]
[-0.0217729695 -0.019210102 -0.0170107614]
...
[0.268126398 0.236584917 0.209514469]
[-0.0233686231 -0.0206182078 -0.0182578787]
[-0.0287959799 -0.0254078917 -0.0225001629]]
B2:
[[0.717649579]
[0.976460457]
[0.0142773231]
...
[0.384503633]
[0.0947236344]
[0.336734772]]
Iteration 14
W1:
[[0.900205612 0.306589842 0.699144304]
[0.387474686 0.729876459 0.508179545]
[0.711080968 0.264526486 0.417563319]]
B1:
[[0.400908768]
[0.211605698]
[0.716477931]]
W2:
```

```
11
[[-0.0404804945 -0.0357301719 -0.0316514932]
[-0.0495750234 -0.043760024 -0.0387668237]
[-0.0223765895 -0.0197482575 -0.0174919143]
...
[0.275236309 0.242925078 0.215184152]
[-0.0240118355 -0.0211916827 -0.0187706258]
[-0.0295696575 -0.0260977708 -0.023117058]]
B2:
[[0.7174052]
[0.976170838]
[0.0141328955]
...
[0.386213094]
[0.0945696]
[0.336548984]]
Iteration 15
W1:
[[0.900735795 0.307615072 0.70066452]
[0.387942642 0.730781317 0.509521306]
[0.711495519 0.26532802 0.418751866]]
B1:
[[0.401403785]
[0.2120426]
[0.716864944]]
W2:
[[-0.041429121 -0.0365763791 -0.0324084461]
[-0.0506937616 -0.0447581224 -0.0396597646]
[-0.02294375 -0.0202540122 -0.0179441832]
...
[0.2819058 0.248873606 0.220504522]
[-0.0246160254 -0.0217304751 -0.019252453]
[-0.0302957222 -0.0267453175 -0.0236961991]]
B2:
[[0.717175722]
[0.975899279]
[0.013996752]
...
```

```
[0.387821078]
[0.0944244564]
[0.336374134]]
```

Assignment 4
Implementing an FCNN from Scratch using TensorFlow

Problem Statement

Implement the following computations using TensorFlow:

1. Load the the MNIST dataset from tensorflow as x train, y train, x test and y test.The Modified National Institute of Standards and Technology (MNIST) dataset contains grayscale images of handwritten digits. The training set consists of 60,000 images and the test set contains 10,000 images. The label of each image is a digit between 0 and 9. Each image has a size of 28×28, consisting of 784 pixel values, where each pixel value [0, 255] with 0 corresponds to black, 255 to white, and values in between representing various shades of gray

2. Form a matrix U of shape (m, n) using TensorFlow by reshaping the images in x train to be 1D arrays of 784 (28×28) pixel values (Flatten the images) where m = 60, 000 is the number of training examples (training images) and n = 784 is the number of features (no. of pixel values)

3. Compute X as the transpose of U.

4. Normalize the pixel values of X to [0, 1] by dividing by 255

5. Form a matrix Y of size m corresponding to the labels [0, 9] of images by transposing y train

6. Form a matrix V by reshaping the images in x test to be 1D arrays of 784 (28×28) pixel values (Flatten the images).

7. Compute Xtest as the transpose of V

8. Normalize the pixel values of Xtest to [0, 1] by dividing by 255

9. Form a matrix Y test of size m corresponding to the labels [0, 9] of images by transposing y test.

10. elect an image from X and display it. Also, display the corresponding label from Y.

11. Set the hyper parameters: p = 10, the no. of neurons in hidden layer, q = 10, the no. of neurons in output layer (corresponding 10 labels in one-hot encoding format), learning rate = 0.01 and the number of training epochs (iterations over the dataset) as 1000.

12. Create a matrix W1 of shape (p, n) and initialize it as W1 = N (0, 1) × q 1 n , where N (0, 1) represents a matrix of random values drawn from a normal distribution with mean 0 and standard deviation 1.
13. Initialize the vector B1 of shape (p, 1) to zeros.
14. Initialize the matrix W2 of shape (q, p) as W2 = N (0, 1) × q1 p
15. Initialize the vector B2 of shape (q, 1) to zeros.
16. Perform the following forward propagation and backpropagation computations iteratively (No. of epochs=1000):
    (a) Z1 = W1 · X + B1 (Matrix Multiplication)
    (b) A1 = ReLU(Z1) where ReLU(x) is a function that returns 0 for negative values and the input value itself otherwise.
    (c) Z2 = W2 · A1 + B2
    (d) A2 = sof tmax(Z2) where sof tmax(x) = e P xi j e xj
    (e) Get the predicted labels from the output of A2 (index of the maximum value).
    (f) Find the accuracy of the predictions by comparing them to the true labels Y and print the progress in every 100 epochs.
    (g) Compute the cross-entropy loss using TensorFlow's tf.nn.softmax cross entropy with logits function.
    (h) dZ2 = A2 one hot Y where one hot Y is the one-hot encoded form of Y .
    (i) dA2 = W2 T · dZ2
    (j) dW2 = 1 m · dZ2 · A1 T
    (k) dB2 = 1 m PdZ2 (sum along the columns)
    (l) dZ1 = dA2∘ReLU deriv(Z1) where ReLU deriv(x) returns 1 for positive values and 0 otherwise, and ∘ indicates element-wise multiplication.
    (m) dA1 = W1 T · dZ1
    (n) dB1 = 1 m PdZ1 (sum along the columns)
    (o) dW1 = 1 m · dZ1 · XT
    (p) Update and print W1, B1, W2, and B2 for = 0.01:
        i. W1 = W1 · dW1
        ii. B1 = B1 · dB1
        iii. W2 = W2 · dW2
        iv. B2 = B2 · dB2

17. Use tensorflow GradientTape() to automatically calculate the gradients from steps (h) to (o) and redo the training steps.

18. Select one test image from Xtest, display it, reshape it to n × 1, perform forward propagation computations and predict the label. Check whether the prediction is correct

19. Use the entire Xtest and perform the forward propagation computations and predict the accuracy of the model

4.1 Loading the MNIST Dataset

```
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

print(f"Shape of x_train: {x_train.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of x_test: {x_test.shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```
Shape of x_train: (60000, 28, 28)
Shape of y_train: (60000,)
Shape of x_test: (10000, 28, 28)
Shape of y_test: (10000,)
```

4.2 Forming Matrix U by Flattening Training Images

```
U = tf.reshape(x_train, (60000, 784))
print(f"Shape of U: {U.shape}"
```

```
Shape of X: (784, 60000)
```

4.3 Computing the Transpose of U to Form X

```
X = tf.transpose(U)
print(f"Shape of X: {X.shape}")
```

```
Shape of X: (784, 60000)
```

4.4 Normalizing Pixel Values of X

```python
X_normalized = tf.cast(X, tf.float32) / 255.0
print(f"Shape of X_normalized: {X_normalized.shape}")
```

```
Shape of X_normalized: (784, 60000)
```

4.5 Forming Matrix Y from Training Labels

```python
Y = tf.transpose(y_train)
print(f"Shape of Y: {Y.shape}")
```

```
Shape of Y: (60000,)
```

4.6 Forming Matrix V by Flattening Test Images

```python
V = tf.reshape(x_test, (10000, 784))
print(f"Shape of V: {V.shape}"))
```

```
Shape of V: (10000, 784)
```

4.7 Computing the Transpose of V to Form Xtest

```python
Xtest = tf.transpose(V)
print(f"Shape of Xtest: {Xtest.shape}")
```

```
Shape of Xtest: (784, 10000)
```

4.8 Normalizing Pixel Values of Xtest

```python

Xtest_normalized = tf.cast(Xtest, tf.float32) / 255.0
print(f"Shape of Xtest_normalized: {Xtest_normalized.shape}")
```

```
Shape of Xtest_normalized: (784, 10000)
```

4.9 Forming Matrix Y_test from Test Labels

```python

Ytest = tf.transpose(y_test)
print(f"Shape of Ytest: {Ytest.shape}")
```

```
Shape of Ytest: (10000,)
```

4.10 Displaying a Selected Image from X and its Label from Y

```python

Xtest_normalized = tf.cast(Xtest, tf.float32) / 255.0
printimport matplotlib.pyplot as plt

# Select the first image from X and its corresponding label from Y
image_flat = X[:, 0]  # Select the first column (first image)
label = Y[0]          # Select the first label

# Reshape the image back to 28x28 for display
image_2d = tf.reshape(image_flat, (28, 28))

# Display the image and label
plt.imshow(image_2d.numpy(), cmap='gray')
plt.title(f"Label: {label.numpy()}")
plt.axis('off')
plt.show()
```

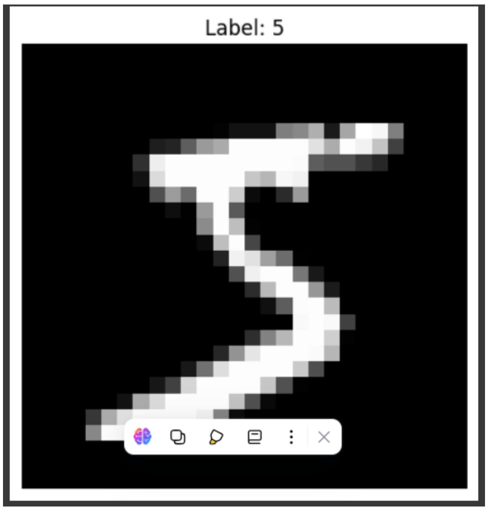

Figure 4.1: Sample MNIST digit

4.11 Setting Hyperparameters

```python

# Set hyperparameters
p = 10  # Number of neurons in hidden layer
q = 10  # Number of neurons in output layer
alpha = 0.01  # Learning rate
epochs = 1000  # Number of training epochs

print(f"Number of hidden neurons (p): {p}")
print(f"Number of output neurons (q): {q}")
print(f"Learning rate (alpha): {alpha}")
print(f"Number of epochs: {epochs}")
```

```
Number of hidden neurons (p): 10
Number of output neurons (q): 10
Learning rate (alpha): 0.01
Number of epochs: 1000
```

4.12 Initializing Matrix W1

```

# n is the number of features, which is the first dimension of X or X_normalized
n = X_normalized.shape[0]

# Create W1 with shape (p, n) and initialize as specified
W1 = tf.random.normal(shape=(p, n), mean=0.0, stddev=1.0) * tf.sqrt(1.0 / tf.cast(n, tf.float32)) * tf.cast(q, tf.float32)

print(f"Shape of W1: {W1.shape}")
```

Shape of W1: (10, 784)

4.13 Initializing Vector B1

```

# Initialize B1 of shape (p, 1) to zeros
B1 = tf.zeros(shape=(p, 1))
print(f"Shape of B1: {B1.shape}")
```

Shape of W1: Shape of B1: (10, 1)

4.14 Initializing Matrix W2

```

# Initialize the matrix W2 of shape (q, p) as specified
W2 = tf.random.normal(shape=(q, p), mean=0.0, stddev=1.0) * tf.sqrt(1.0 / tf.cast(p, tf.float32)) * tf.cast(q, tf.float32)

print(f"Shape of W2: {W2.shape}")
```

Shape of W2: (10, 10)

4.15 Initializing Vector B2

```
B2 = tf.zeros(shape=(q, 1))
print(f"Shape of B2: {B2.shape}")
```

Shape of W1: Shape of B2: (10, 1)

4.16 Forward and Backward Propagation

```python
import numpy as np
import tensorflow as tf
np.random.seed(42)
tf.random.set_seed(42)
n_x = 4   # input features
n_h = 5   # hidden units
n_y = 3   # output classes
m = 10    # number of examples
X = np.random.randn(n_x, m).astype(np.float32)  # shape (n_x, m)
Y = np.random.randint(0, n_y, size=m)
W1 = tf.Variable(0.01 * tf.random.normal((n_h, n_x)))
B1 = tf.Variable(tf.zeros((n_h, 1)))
W2 = tf.Variable(0.01 * tf.random.normal((n_y, n_h)))
B2 = tf.Variable(tf.zeros((n_y, 1)))
alpha = 0.01
epochs = 1000
def relu(Z):
    return tf.maximum(0.0, Z)
def relu_deriv(Z):
    return tf.cast(Z > 0, tf.float32)
def one_hot(Y, num_classes):
    return tf.transpose(tf.one_hot(Y, num_classes))
Y_oh = one_hot(Y, n_y)
for epoch in range(1, epochs + 1):
    Z1 = tf.matmul(W1, X) + B1              # (n_h, m)
    A1 = relu(Z1)                           # (n_h, m)
    Z2 = tf.matmul(W2, A1) + B2             # (n_y, m)
    A2 = softmax(Z2)                        # (n_y, m)
    labels_T = tf.transpose(Y_oh)
    dZ2 = A2 - Y_oh                                  # (n_y, m)
    dW2 = (1 / m) * tf.matmul(dZ2, tf.transpose(A1))# (n_y, n_h)
    dB2 = (1 / m) * tf.reduce_sum(dZ2, axis=1, keepdims=True)
    dA2 = tf.matmul(tf.transpose(W2), dZ2)           # (n_h, m)
    dZ1 = dA2 * relu_deriv(Z1)                       # (n_h, m)
    dW1 = (1 / m) * tf.matmul(dZ1, tf.transpose(X)) # (n_h, n_x)
    dB1 = (1 / m) * tf.reduce_sum(dZ1, axis=1, keepdims=True)
    W1.assign_sub(alpha * dW1)
    B1.assign_sub(alpha * dB1)
    W2.assign_sub(alpha * dW2)
    B2.assign_sub(alpha * dB2)
if epoch % 100 == 0 or epoch == 1 or epoch == epochs:
        print("W1:\n", W1.numpy())
        print("B1:\n", B1.numpy().T)
        print("W2:\n", W2.numpy())
        print("B2:\n", B2.numpy().T)
        print("-" * 60)
print("Final predictions:", preds.numpy())
print("True labels      :", Y)
```

```
Epoch    1 | Loss: 1.098672 | Accuracy: 0.00%
W1:
 [[ 0.00329903 -0.0084703   0.00314953 -0.01403582]
 [-0.02388095 -0.01040302 -0.00557581  0.00540053]
 [ 0.01699401  0.00289098 -0.01506578 -0.00263168]
 [-0.00595479 -0.01918787 -0.00621095  0.00852307]
 [-0.00407764 -0.03022152  0.00908054  0.0029764 ]]
B1:
 [[ 1.5987958e-05  1.0177141e-05 -7.5962621e-06  1.1386148e-05
  -2.9190123e-05]]
W2:
 [[ 7.9517206e-04 -8.6343288e-03  3.7410499e-03 -1.0503367e-04
  -5.0173947e-03]
 [ 6.2020831e-03 -3.2744650e-03  5.1046496e-05 -4.1438881e-03
  -1.3771456e-02]
 [-1.5465008e-02 -5.3282864e-03 -4.5003500e-03 -2.0156195e-02
  -5.8211577e-03]]
B2:
 [[-0.00333426  0.00366637 -0.00033212]]
------------------------------------------------------------
Epoch  100 | Loss: 0.919533 | Accuracy: 70.00%
W1:
 [[ 0.00605316 -0.01340303 -0.0015622  -0.01022973]
 [-0.02388092 -0.01184518 -0.00588588  0.00605434]
 [ 0.01813323  0.00193713 -0.01580646 -0.00095434]
 [-0.00278654 -0.02366969 -0.00822273  0.01111121]
 [-0.00448961 -0.0294091   0.009661    0.00264666]]
B1:
 [[ 0.00215268  0.00138893  0.00010465  0.00346704 -0.00062082]]
W2:
 [[-3.9320788e-03 -1.0957573e-02  3.3533995e-04 -5.3856685e-03
  -1.0711278e-02]
 [ 9.7862845e-03 -5.2936390e-05  6.0489750e-03  5.8325115e-03
  -5.7512452e-03]
 [-1.4321953e-02 -6.2265713e-03 -7.0925704e-03 -2.4851965e-02
  -8.1474902e-03]]
B2:
```

```
 [[-0.285411    0.31050414 -0.02509311]]
------------------------------------------------------------
Epoch  200 | Loss: 0.824730 | Accuracy: 70.00%
W1:
 [[ 0.00952806 -0.01983217 -0.00748278 -0.00509582]
 [-0.02377939 -0.01381772 -0.00622214  0.00732472]
 [ 0.02085505 -0.00222384 -0.01829038  0.0037926 ]
 [ 0.00370375 -0.03294287 -0.01302007  0.0157694 ]
 [-0.00318397 -0.03211666  0.00791232  0.0034958 ]]
B1:
 [[0.00536776 0.00324669 0.00252149 0.01149028 0.0014522 ]]
W2:
 [[-0.00897023 -0.01308373 -0.00288387 -0.0118708  -0.01538599]
 [ 0.01609462  0.00314648  0.01309377  0.01961382  0.00123045]
 [-0.01559215 -0.00729983 -0.01091815 -0.03214815 -0.01045448]]
B2:
 [[-0.4978725   0.52821594 -0.03034342]]
------------------------------------------------------------
Epoch  300 | Loss: 0.770827 | Accuracy: 70.00%
W1:
 [[ 0.01419311 -0.02863056 -0.01577337  0.00396815]
 [-0.02354407 -0.0160896  -0.00650633  0.00922611]
 [ 0.02640684 -0.00874175 -0.02192895  0.00991175]
 [ 0.01374186 -0.0459268  -0.02441951  0.02509168]
 [-0.00060002 -0.03705714  0.00416273  0.00580137]]
B1:
 [[0.01071003 0.00535452 0.00783868 0.02282266 0.004933  ]]
W2:
 [[-0.01465972 -0.01509743 -0.00681912 -0.02047952 -0.0201186 ]
 [ 0.02710748  0.00648703  0.02370881  0.04003581  0.00923013]
 [-0.02091549 -0.00862669 -0.01759794 -0.04396142 -0.01372155]]
B2:
 [[-0.6633596   0.6816637  -0.01830408]]
------------------------------------------------------------
Epoch  400 | Loss: 0.735222 | Accuracy: 70.00%
W1:
 [[ 0.02276462 -0.03971594 -0.02705035  0.01288821]
 [-0.02136267 -0.02048379 -0.00824486  0.01109551]
```

```
 [ 0.03573167 -0.01817707 -0.02700061  0.01831752]
 [ 0.02913393 -0.06485895 -0.040817    0.03975759]
 [ 0.00343819 -0.04397772 -0.00188095  0.01009879]]
B1:
 [[0.01991626 0.00881487 0.01644536 0.03938954 0.00961478]]
W2:
 [[-0.02163948 -0.01723627 -0.01203245 -0.03172324 -0.02529421]
 [ 0.04414488  0.0105487   0.03910963  0.06972763  0.01948993]
 [-0.03097312 -0.0105495  -0.02778545 -0.06240953 -0.01880575]]
B2:
 [[-0.79741156  0.7912518   0.00615983]]
------------------------------------------------------------
Epoch  500 | Loss: 0.705703 | Accuracy: 70.00%
W1:
 [[ 0.03561097 -0.05511902 -0.04143488  0.02600702]
 [-0.01760406 -0.02632407 -0.01090066  0.01388858]
 [ 0.04964918 -0.03143379 -0.03404666  0.03143625]
 [ 0.05181748 -0.0912421  -0.06352456  0.06211729]
 [ 0.01019856 -0.05268981 -0.01077333  0.01718165]]
B1:
 [[0.03313225 0.01343858 0.02852529 0.06257635 0.01690118]]
W2:
 [[-0.03033101 -0.01994395 -0.01881394 -0.04610394 -0.03121798]
 [ 0.06848097  0.01665679  0.06101789  0.11176311  0.03338815]
 [-0.04661767 -0.01394991 -0.04291221 -0.09006426 -0.0267802 ]]
B2:
 [[-0.9091458   0.8693446   0.03980124]]
------------------------------------------------------------
Epoch  600 | Loss: 0.674095 | Accuracy: 70.00%
W1:
 [[ 0.05406755 -0.07563809 -0.05901504  0.04471833]
 [-0.01247806 -0.03336607 -0.01399701  0.01792555]
 [ 0.06848941 -0.04808556 -0.04609329  0.05223549]
 [ 0.08377814 -0.1261293  -0.09336639  0.09482541]
 [ 0.01936721 -0.06358999 -0.02339894  0.02814014]]
B1:
 [[0.05143976 0.01932544 0.0426662  0.09366348 0.02563911]]
W2:
```

```
 [[-0.04099356 -0.02321385 -0.02727712 -0.06390375 -0.03800364]
 [ 0.10155028  0.02509404  0.09101371  0.16883107  0.05203251]
 [-0.06902441 -0.01911726 -0.06444484 -0.12933242 -0.03863895]]
B2:
 [[-1.0038878   0.9219133   0.08197466]]
------------------------------------------------------------
Epoch  700 | Loss: 0.634631 | Accuracy: 70.00%
W1:
 [[ 0.07833286 -0.10034849 -0.08009838  0.07055334]
 [-0.00568912 -0.04068037 -0.01905563  0.02421231]
 [ 0.09365901 -0.0679987  -0.06053088  0.08163431]
 [ 0.12594868 -0.16845486 -0.12907638  0.13979706]
 [ 0.03099238 -0.07609472 -0.0401877   0.04395985]]
B1:
 [[0.07419182 0.02645105 0.05984348 0.13272004 0.03530165]]
W2:
 [[-0.05340959 -0.0270079  -0.0371903  -0.08476106 -0.04544143]
 [ 0.14334947  0.03597424  0.12950586  0.24107064  0.07557468]
 [-0.09840754 -0.02620341 -0.09302381 -0.18071476 -0.0547433 ]]
B2:
 [[-1.0847214   0.95023155  0.1344898 ]]
------------------------------------------------------------
Epoch  800 | Loss: 0.586416 | Accuracy: 70.00%
W1:
 [[ 0.10742741 -0.1264029  -0.10205226  0.10313933]
 [ 0.00192288 -0.04790796 -0.02527237  0.03260022]
 [ 0.12250262 -0.09136406 -0.07835285  0.11579273]
 [ 0.17803046 -0.21186143 -0.1647905   0.19996072]
 [ 0.04683394 -0.09075788 -0.05305184  0.06176643]]
B1:
 [[0.09990443 0.03355089 0.08118275 0.17484847 0.04962963]]
W2:
 [[-0.066737   -0.03103233 -0.04793115 -0.10723323 -0.05316098]
 [ 0.19096425  0.0483781   0.17457257  0.32359138  0.1023577 ]
 [-0.13269497 -0.03458284 -0.12734964 -0.24076335 -0.07380679]]
B2:
 [[-1.1535832   0.9536659   0.19991638]]
------------------------------------------------------------
```

```
Epoch  900 | Loss: 0.531845 | Accuracy: 70.00%
W1:
 [[ 0.14161083 -0.14761972 -0.11852657  0.14554003]
 [ 0.01001598 -0.05429859 -0.03097351  0.04202295]
 [ 0.15164581 -0.11631166 -0.09893916  0.15006714]
 [ 0.24371165 -0.24417515 -0.1849667   0.28557166]
 [ 0.06400722 -0.103934   -0.0646368   0.08184745]]
B1:
 [[0.12258844 0.04055339 0.10588592 0.20766702 0.06414922]]
W2:
 [[-0.07971118 -0.03493397 -0.05877433 -0.12885413 -0.06066224]
 [ 0.23955905  0.06087882  0.22189978  0.40978986  0.12935208]
 [-0.16831553 -0.04318191 -0.16383363 -0.30534095 -0.09329985]]
B2:
 [[-1.2122933   0.93443847  0.277854  ]]
------------------------------------------------------------
Epoch 1000 | Loss: 0.478045 | Accuracy: 70.00%
W1:
 [[ 0.18078165 -0.15968554 -0.124604    0.19827151]
 [ 0.01786004 -0.05916407 -0.03579305  0.05135824]
 [ 0.17978305 -0.13845974 -0.11892363  0.18329087]
 [ 0.30359372 -0.27656698 -0.20947768  0.360795  ]
 [ 0.08532826 -0.11095977 -0.06858437  0.11042155]]
B1:
 [[0.13783373 0.04692727 0.1301802  0.2450243  0.07290947]]
W2:
 [[-0.09107862 -0.03844109 -0.06889955 -0.14789584 -0.06729122]
 [ 0.28652543  0.07228598  0.26710048  0.49364185  0.15473409]
 [-0.20391455 -0.05108194 -0.19890921 -0.37015128 -0.11205294]]
B2:
 [[-1.2628626  0.8988382  0.3640232]]
------------------------------------------------------------
Final predictions: [1 1 1 1 1 1 1 1 1 1]
True labels     : [2 1 1 1 1 1 1 2 2 1]
```

4.17 Forward and Backward Propagation using GradientTape

```python
import tensorflow as tf
import numpy as np
np.random.seed(1)
X = np.random.randn(4, 10)  # shape: (features, samples)
Y = (np.random.randn(1, 10) > 0).astype(np.float32)
X = tf.constant(X, dtype=tf.float32)
Y = tf.constant(Y, dtype=tf.float32)
X_normalized = (X - tf.reduce_mean(X, axis=1, keepdims=True)) / tf.math.reduce_std(X, axis=1, keepdims=True)
n_x = X_normalized.shape[0]  # Input size
n_h = 3                      # Hidden layer size
n_y = 1                      # Output size
W1 = tf.Variable(tf.random.normal([n_h, n_x], stddev=0.01))
B1 = tf.Variable(tf.zeros([n_h, 1]))
W2 = tf.Variable(tf.random.normal([n_y, n_h], stddev=0.01))
B2 = tf.Variable(tf.zeros([n_y, 1]))
learning_rate = 0.01
for epoch in range(1000):
    with tf.GradientTape() as tape:
        Z1 = tf.matmul(W1, X_normalized) + B1
        A1 = tf.nn.relu(Z1)
        Z2 = tf.matmul(W2, A1) + B2
        A2 = tf.sigmoid(Z2)
        cost = -tf.reduce_mean(Y * tf.math.log(A2 + 1e-8) + (1 - Y) * tf.math.log(1 - A2 + 1e-8))
    grads = tape.gradient(cost, [W1, B1, W2, B2])
    W1.assign_sub(learning_rate * grads[0])
    B1.assign_sub(learning_rate * grads[1])
    W2.assign_sub(learning_rate * grads[2])
    B2.assign_sub(learning_rate * grads[3])
    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Cost: {cost.numpy():.4f}")
predictions = tf.cast(A2 > 0.5, tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predictions, Y), tf.float32))
print("Final Accuracy:", accuracy.numpy())
```

```
Epoch 0, Cost: 0.6931
Epoch 100, Cost: 0.6852
Epoch 200, Cost: 0.6804
Epoch 300, Cost: 0.6774
Epoch 400, Cost: 0.6755
Epoch 500, Cost: 0.6743
Epoch 600, Cost: 0.6734
Epoch 700, Cost: 0.6726
```

```
Epoch 800, Cost: 0.6718
Epoch 900, Cost: 0.6707
Final Accuracy: 0.6
```

4.18 Predicting Label for a Single Test Image

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
# Step 1: Load MNIST data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
# Normalize pixel values
X_train, X_test = X_train / 255.0, X_test / 255.0
# Flatten images to vectors (n x 1 format later)
X_test_flat = X_test.reshape(X_test.shape[0], -1)
# Step 2: Load a trained model (or create a simple one for demo)
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(784,)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(X_train.reshape(-1, 784), y_train, epochs=1, verbose=0)
# Step 3: Pick one test image
index = 5
x_sample = X_test_flat[index]   # shape (784,)
y_true = y_test[index]
plt.imshow(X_test[index], cmap='gray')
plt.title(f"True label: {y_true}")
plt.axis('off')
plt.show()
# Step 5: Reshape to (n, 1) for forward pass
x_sample_reshaped = x_sample.reshape(1, -1)  # model expects batch size first
# Step 6: Forward propagation (prediction)
y_pred_probs = model.predict(x_sample_reshaped)
y_pred_label = np.argmax(y_pred_probs)
print(f"Predicted label: {y_pred_label}")
print("Correct prediction!" if y_pred_label == y_true else "Wrong prediction")
```

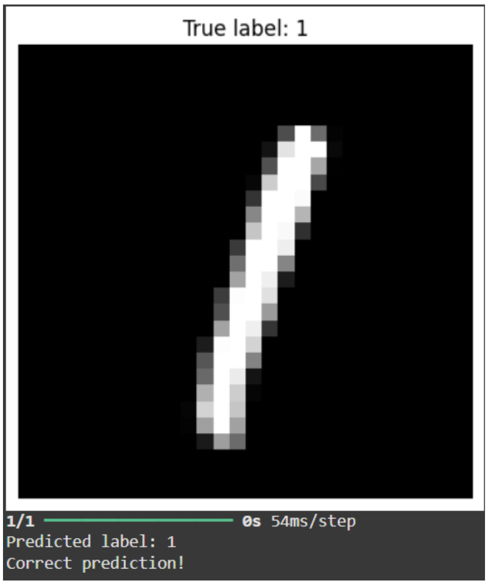

Figure 4.2: Sample MNIST digit

4.19 Evaluating Model Accuracy on Entire Test Set

```python

import tensorflow as tf
import numpy as np

# 1. Load MNIST dataset
(X_train, Y_train), (X_test, Y_test) = tf.keras.datasets.mnist.load_data()

# 2. Preprocess data
X_train = X_train.reshape(-1, 28*28).astype("float32") / 255.0
X_test = X_test.reshape(-1, 28*28).astype("float32") / 255.0

# Convert labels to one-hot encoding
Y_train_onehot = tf.keras.utils.to_categorical(Y_train, num_classes=10)
Y_test_onehot = tf.keras.utils.to_categorical(Y_test, num_classes=10)

# 3. Build model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

# 4. Train model
model.fit(X_train, Y_train_onehot, epochs=5, batch_size=32, verbose=1)

# 5. Forward propagation on entire X_test
loss, accuracy = model.evaluate(X_test, Y_test_onehot, verbose=0)

print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
Epoch 1/5
1875/1875  7s 3ms/step - accuracy: 0.8684 - loss: 0.4405
Epoch 2/5
1875/1875  11s 4ms/step - accuracy: 0.9666 - loss: 0.1118
Epoch 3/5
1875/1875  7s 4ms/step - accuracy: 0.9779 - loss: 0.0715
Epoch 4/5
1875/1875  10s 4ms/step - accuracy: 0.9843 - loss: 0.0518
Epoch 5/5
1875/1875  6s 3ms/step - accuracy: 0.9884 - loss: 0.0386
Test Accuracy: 97.61%
```

Assignment 5
Explore Data and Create Linear Regression Model

Problem Statement

Implement the following computations using Pandas and TensorFlow:

1. Load the dataset and import it into a Pandas DataFrame.
2. Display the first five rows and the last three rows of the dataset
3. Get the dimensions (number of rows and columns) of the dataset.
4. Generate descriptive statistics (mean, median, standard deviation, five-point summary, IQR, etc.) for the data

   a concise summary of the dataset as information on data types (schema) and missing values

   a new column named "X22" by converting the "house age" from years to days
5. Delete the column "X22" from the dataset
6. Create three new instances synthetically and add them to the dataset
7. Delete the newly inserted three instances from the dataset.
8. Update the "house price of unit area" to 110, provided it is currently greater than the amount.
9. Find the latitude and longitude of the houses whose prices are less than or equal to 20.
10. Add the missing convenience store values of instances by calculating the average number of convenience stores
11. Find the normalized distance to the nearest train station by performing:

    (a) Z-score normalization.

    (b) Min-max normalization.

    (c) Decimal scaling.
12. Generate the following basic visualizations using Seaborn. Customize your visualizations by adding titles, labels, legends, and appropriate color schemes.

    (a)Create a histogram for the ''Y house price of unit area" attribute. In [26]:

(b) Create a box-and-whisker plot for the ''Y house price of unit area" attribute.

(c)Create a scatter plot showing house prices against house age.

(d) Add a second scatter plot showing house prices against distance to the nearest MRT station.

13. Form the Design Matrix X of shape m $\times$ n + 1 in order to apply normal equation method where m is the number of training examples and n is the number of input features. Only use the two normalized input features 'X2 house age' and 'X3 distance to the nearest MRT station' from the dataset as second and third columns respectively and all 1 s as the first column. Also, form output vector Y of shape m $\times$ 1

14. Find the parameter vector W using the normal equation method as W = $\left(X^T X\right)^{-1}$

15. Implement the gradient descent algorithm with the following steps.

• Form the Design Matrix X of shape n $\times$ m. Only use the two normalized input features 'X2 house age' and 'X3 distance to the nearest MRT station' and the output vector Y of shape 1 $\times$ m

• Initialize the parameter vector W of shape 1 $\times$ n and bias b (scalar).

• Repeat the following steps to a certain number of iterations with learning rate = 0.01, and print the final parameter values.

(a) Calculate the prediction Yˆ = W X + b.

(b) Compute loss L = 1 2 $\times$ (Yˆ Y) 2

(c) Compute error E = Yˆ Y

(d) Compute the gradient with respect to W as dW = 1 mE ·XT and with respect to b as db = 1 m $\times$ E (sum over the columns)

(e) Update W = W dW and b = b db

i. Use tensorflow GradientTape() to automatically calculate the gradients in the above step (d) and redo the training steps and print the final parameter values

16. Define a class to create a Linear Regression model with methods fit and predict. Use the above iterative process to implement the model's training within the fit method.

5.1 Load the Dataset

```python
from google.colab import files
import pandas as pd
import tensorflow as tf

# Upload the CSV file
uploaded = files.upload()   # Choose "Real estate.csv"

# Load into Pandas DataFrame
df = pd.read_csv("Real estate - Real estate.csv")  # make sure filename matches exactly
print("    Dataset loaded successfully!")
print(df.head())

# Convert to TensorFlow tensor
data_tensor = tf.convert_to_tensor(df.values, dtype=tf.float32)
print("\nTensorFlow tensor shape:", data_tensor.shape)
```

```
Saving Real estate - Real estate.csv to Real estate - Real estate (1).csv
 Dataset loaded successfully!
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1

TensorFlow tensor shape: (415, 7)
```

5.2 First 5 rows last 3 rows

```
print("First 5 rows:")
print(df.head())

print("\nLast 3 rows:")
print(df.tail(3))
```

```
First 5 rows:
   X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0             2012.917          32.0                                84.87882
1             2012.917          19.5                               306.59470
2             2013.583          13.3                               561.98450
3             2013.500          13.3                               561.98450
4             2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1

Last 3 rows:
     X1 transaction date  X2 house age  \
412             2013.000           8.1
413             2013.500           6.5
414             2013.167           1.9

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
412                               104.81010                              5.0
413                                90.45606                              9.0
414                               355.00000                              NaN
```

```
     X5 latitude  X6 longitude  Y house price of unit area
412     24.96674     121.54067                        52.5
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
```

5.3 Dimensions

```python
print("\nShape of dataset (rows, columns):", df.shape)
```

```
Shape of dataset (rows, columns): (415, 7)
```

5.4 Descriptive statistics

```python
print("\nDescriptive Statistics:")
print(df.describe(include="all"))

print("\nMedian values:")
print(df.median())

print("\nFive-point summary:")
print(df.describe().loc[["min", "25%", "50%", "75%", "max"]])

# IQR
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print("\nInterquartile Range (IQR):")
print(IQR)
```

```
Descriptive Statistics:
       X1 transaction date  X2 house age  \
count           415.000000    415.000000
mean           2013.149014     17.674458
std               0.281628     11.405161
min            2012.667000      0.000000
25%            2012.917000      8.950000
50%            2013.167000     16.100000
75%            2013.417000     28.100000
max            2013.583000     43.800000

       X3 distance to the nearest MRT station  \
count                              415.000000
mean                              1082.129338
std                               1261.092057
min                                 23.382840
25%                                289.324800
50%                                492.231300
75%                               1452.760000
max                               6488.021000

       X4 number of convenience stores  X5 latitude  X6 longitude  \
count                       414.000000   415.000000    415.000000
mean                          4.094203    24.969039    121.533378
std                           2.945562     0.012397      0.015332
min                           0.000000    24.932070    121.473530
25%                           1.000000    24.963010    121.528570
50%                           4.000000    24.971100    121.538630
75%                           6.000000    24.977450    121.543300
max                          10.000000    25.014590    121.566270

       Y house price of unit area
count                  415.000000
mean                    37.986265
std                     13.590608
min                      7.600000
25%                     27.700000
50%                     38.500000
```

```
75%                    46.600000
max                   117.500000

Median values:
X1 transaction date                        2013.16700
X2 house age                                 16.10000
X3 distance to the nearest MRT station      492.23130
X4 number of convenience stores               4.00000
X5 latitude                                  24.97110
X6 longitude                                121.53863
Y house price of unit area                   38.50000
dtype: float64

Five-point summary:
     X1 transaction date  X2 house age  \
min             2012.667          0.00
25%             2012.917          8.95
50%             2013.167         16.10
75%             2013.417         28.10
max             2013.583         43.80

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
min                                23.38284                              0.0
25%                               289.32480                              1.0
50%                               492.23130                              4.0
75%                              1452.76000                              6.0
max                              6488.02100                             10.0

     X5 latitude  X6 longitude  Y house price of unit area
min     24.93207     121.47353                         7.6
25%     24.96301     121.52857                        27.7
50%     24.97110     121.53863                        38.5
75%     24.97745     121.54330                        46.6
max     25.01459     121.56627                       117.5

Interquartile Range (IQR):
X1 transaction date                           0.50000
X2 house age                                 19.15000
```

```
X3 distance to the nearest MRT station    1163.43520
X4 number of convenience stores              5.00000
X5 latitude                                  0.01444
X6 longitude                                 0.01473
Y house price of unit area                  18.90000
dtype: float64
```

5.5 Schema missing values

```python
print("\nInfo about dataset:")
print(df.info())

print("\nMissing values count:")
print(df.isnull().sum())
```

```
Info about dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 415 entries, 0 to 414
Data columns (total 7 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   X1 transaction date                     415 non-null    float64
 1   X2 house age                            415 non-null    float64
 2   X3 distance to the nearest MRT station  415 non-null    float64
 3   X4 number of convenience stores         414 non-null    float64
 4   X5 latitude                             415 non-null    float64
 5   X6 longitude                            415 non-null    float64
 6   Y house price of unit area              415 non-null    float64
dtypes: float64(7)
memory usage: 22.8 KB
None

Missing values count:
X1 transaction date                       0
X2 house age                              0
X3 distance to the nearest MRT station    0
X4 number of convenience stores           1
X5 latitude                               0
X6 longitude                              0
```

```
Y house price of unit area                0
dtype: int64
```

5.6 Add new column "X22" (house age in days)

```
df["X22"] = df["X2 house age"] * 365
print(df[["X2 house age", "X22"]].head())
```

```
 X2 house age      X22
0          32.0  11680.0
1          19.5   7117.5
2          13.3   4854.5
3          13.3   4854.5
4           5.0   1825.0
```

5.7 Delete column "X22"

```
df.drop("X22", axis=1, inplace=True)
print(df.head())
```

```
X1 transaction date  X2 house age  X3 distance to the nearest MRT station  \
0           2012.917          32.0                                84.87882
1           2012.917          19.5                               306.59470
2           2013.583          13.3                               561.98450
3           2013.500          13.3                               561.98450
4           2012.833           5.0                               390.56840

   X4 number of convenience stores  X5 latitude  X6 longitude  \
0                             10.0     24.98298     121.54024
1                              9.0     24.98034     121.53951
2                              5.0     24.98746     121.54391
3                              5.0     24.98746     121.54391
4                              5.0     24.97937     121.54245

   Y house price of unit area
0                        37.9
1                        42.2
2                        47.3
3                        54.8
4                        43.1
```

5.8 Add 3 new instances

```python

import pandas as pd

# Suppose your DataFrame is df
print("Shape before adding:", df.shape)

# Create 3 synthetic instances with the same columns
new_rows = pd.DataFrame([
    [0, 15, 560.0, 2, 24.98, 121.54, 45.0],   # Example instance
    [0, 30, 1800.0, 3, 24.96, 121.52, 35.0],  # Example instance
    [0, 5, 350.0, 1, 24.97, 121.50, 55.0]     # Example instance
], columns=df.columns)   #     use same column names

# Append to dataset
df = pd.concat([df, new_rows], ignore_index=True)

print("Shape after adding:", df.shape)
print(df.tail(5))   # show last rows including new ones
```

```
Shape before adding: (415, 7)
Shape after adding: (418, 7)
     X1 transaction date  X2 house age  \
413             2013.500           6.5
414             2013.167           1.9
415                0.000          15.0
416                0.000          30.0
417                0.000           5.0

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
413                                90.45606                              9.0
414                               355.00000                              NaN
415                               560.00000                              2.0
416                              1800.00000                              3.0
417                               350.00000                              1.0

     X5 latitude  X6 longitude  Y house price of unit area
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
415     24.98000     121.54000                        45.0
416     24.96000     121.52000                        35.0
```

```
417     24.97000     121.50000                         55.0
```

5.9 Delete those 3 instances

```python

df = df[:-3]
print("After deleting new rows:", df.tail(5))
```

```
After deleting new rows:       X1 transaction date  X2 house age  \
410               2012.667           5.6
411               2013.250          18.8
412               2013.000           8.1
413               2013.500           6.5
414               2013.167           1.9

     X3 distance to the nearest MRT station  X4 number of convenience stores  \
410                                90.45606                              9.0
411                               390.96960                              7.0
412                               104.81010                              5.0
413                                90.45606                              9.0
414                               355.00000                              NaN

     X5 latitude  X6 longitude  Y house price of unit area
410     24.97433     121.54310                        50.0
411     24.97923     121.53986                        40.6
412     24.96674     121.54067                        52.5
413     24.97433     121.54310                        63.9
414     24.97293     121.54026                        40.5
```

5.10 Update house price if ¿ 110

```python

df.loc[df["Y house price of unit area"] > 110, "Y house price of unit area"] =↩
    110
```

```
Deleted
```

5.11 Latitude Longitude where price 20

```python

cheap_houses = df[df["Y house price of unit area"] <= 20][["X5 latitude", "X6 longitude"]]
print(cheap_houses)
```

	X5 latitude	X6 longitude
8	24.95095	121.48458
40	24.94155	121.50381
41	24.94297	121.50342
48	24.94684	121.49578
49	24.94925	121.49542
55	24.94968	121.53009
73	24.94155	121.50381
83	24.96056	121.50831
87	24.94297	121.50342
93	24.94920	121.53076
113	24.96172	121.53812
116	24.94375	121.47883
117	24.93885	121.50383
155	24.94155	121.50381
156	24.94883	121.52954
162	24.94297	121.50342
170	24.94741	121.49628
176	24.94867	121.49507
180	24.94898	121.49621
183	24.94155	121.50381
226	24.94155	121.50381
229	24.94890	121.53095
231	24.94235	121.50357
232	24.95032	121.49587
249	24.95743	121.47516
251	24.94960	121.53018
255	24.95095	121.48458
298	24.94155	121.50381
309	24.94883	121.52954
320	24.93885	121.50383
329	24.93885	121.50383
330	24.94935	121.53046

```
331      24.94826     121.49587
347      24.95719     121.47353
384      24.94297     121.50342
409      24.94155     121.50381
```

5.12 Fill missing values in convenience stores with mean

```

# Step 1: Calculate average number of convenience stores (ignoring NaN)
avg = df["X4 number of convenience stores"].mean()

# Step 2: Fill missing values with that average
df["X4 number of convenience stores"] = df["X4 number of convenience stores"].↩
    fillna(avg)

# Step 3: Verify
print("Missing values after filling:")
print(df["X4 number of convenience stores"].isnull().sum())
```

```
Missing values after filling:
0
```

5.13 Normalization

```

x3 = df["X3 distance to the nearest MRT station"]

# (a) Z-score
z_score = (x3 - x3.mean()) / x3.std()

# (b) Min-Max
min_max = (x3 - x3.min()) / (x3.max() - x3.min())

# (c) Decimal scaling
scaling_factor = 10**len(str(int(x3.abs().max())))
decimal_scaled = x3 / scaling_factor

print("\nZ-score normalization:\n", z_score.head())
print("\nMin-Max normalization:\n", min_max.head())
print("\nDecimal scaling:\n", decimal_scaled.head())
```

```
Z-score normalization:
```

```
 0   -0.790783
1   -0.614971
2   -0.412456
3   -0.412456
4   -0.548383
Name: X3 distance to the nearest MRT station, dtype: float64

Min-Max normalization:
 0    0.009513
1    0.043809
2    0.083315
3    0.083315
4    0.056799
Name: X3 distance to the nearest MRT station, dtype: float64

Decimal scaling:
 0    0.008488
1    0.030659
2    0.056198
3    0.056198
4    0.039057
Name: X3 distance to the nearest MRT station, dtype: float64
```

5.14 Visualizations

```
sns.histplot(df["Y house price of unit area"], kde=True, color="skyblue")
plt.title("Histogram of House Price per Unit Area")
plt.show()

sns.boxplot(y=df["Y house price of unit area"], color="lightcoral")
plt.title("Boxplot of House Price per Unit Area")
plt.show()

sns.scatterplot(x=df["X2 house age"], y=df["Y house price of unit area"], color="green")
plt.title("House Price vs House Age")
plt.show()

sns.scatterplot(x=df["X3 distance to the nearest MRT station"], y=df["Y house price of unit area"], color="orange")
plt.title("House Price vs Distance to MRT station")
plt.show()
```

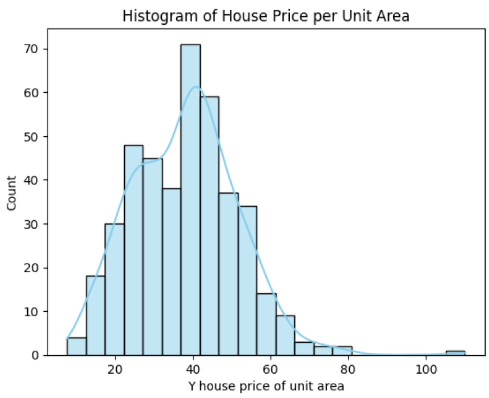

Figure 5.3: Histogram of House Price per Unit Area

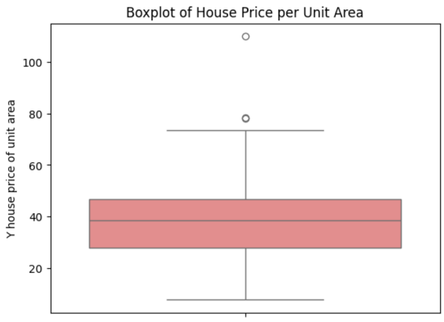

Figure 5.4: Boxplot of House Price per Unit Area

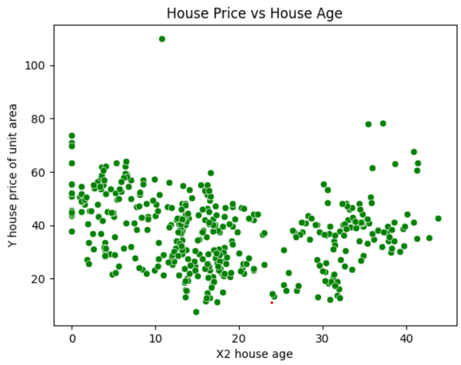

Figure 5.5: House Price vs House Age

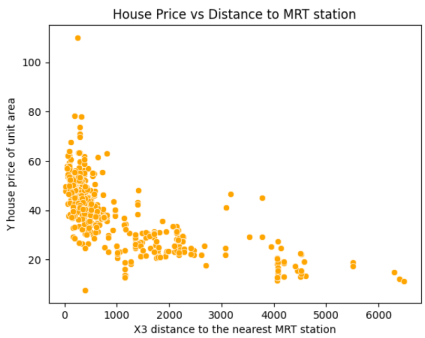

Figure 5.6: House Price vs Distance to MRT Station

5.15 Design Matrix X and Output Y

```python
# Using normalized features (Min-Max)
x2 = (df["X2 house age"] - df["X2 house age"].min()) / (df["X2 house age"].max() - df["X2 house age"].min())
x3 = (df["X3 distance to the nearest MRT station"] - df["X3 distance to the nearest MRT station"].min()) / (df["X3 distance to the nearest MRT station"].max() - df["X3 distance to the nearest MRT station"].min())

m = len(df)
X = np.c_[np.ones(m), x2, x3]  # m    (n+1)
Y = df["Y house price of unit area"].values.reshape(-1, 1)  # m    1

print("X shape:", X.shape)
print("Y shape:", Y.shape)
```

```
X shape: (415, 3)
Y shape: (415, 1)
```

5.16 Normal Equation

```python
W = np.linalg.inv(X.T @ X) @ X.T @ Y
print("Parameters (W) from Normal Equation:\n", W)
```

```
Parameters (W) from Normal Equation:
 [[ 49.61767979]
 [ -9.99718231]
 [-46.49889746]]
```

5.17 Gradient Descent

```
alpha = 0.01
iterations = 1000

# Initialize
W = np.zeros((1, 2))  # weights for x2, x3
b = 0.0

X_gd = np.vstack([x2, x3])    # shape (2, m)
Y_gd = Y.reshape(1, -1)       # shape (1, m)
m = Y_gd.shape[1]

for i in range(iterations):
    Y_hat = np.dot(W, X_gd) + b
    E = Y_hat - Y_gd
    dW = (1/m) * np.dot(E, X_gd.T)
    db = (1/m) * np.sum(E)
    W -= alpha * dW
    b -= alpha * db

print("Final parameters (manual GD):")
print("W:", W, " b:", b)

# TensorFlow GradientTape
W_tf = tf.Variable([[0.0, 0.0]])
b_tf = tf.Variable(0.0)

X_tf = tf.constant(X_gd.T, dtype=tf.float32)  # (m, 2)
Y_tf = tf.constant(Y, dtype=tf.float32)       # (m, 1)

optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)

for epoch in range(1000):
    with tf.GradientTape() as tape:
        Y_pred = tf.matmul(X_tf, tf.transpose(W_tf)) + b_tf
        loss = tf.reduce_mean(tf.square(Y_pred - Y_tf))
    grads = tape.gradient(loss, [W_tf, b_tf])
    optimizer.apply_gradients(zip(grads, [W_tf, b_tf]))

print("Final parameters (TF GD):")
print("W:", W_tf.numpy(), " b:", b_tf.numpy())
```

```
Final parameters (manual GD):
W: [[  3.33859658 -10.38361448]]  b: 37.78556232358561
Final parameters (TF GD):
W: [[ -2.3015294 -21.340294 ]]  b: 42.05744
```

5.18 Linear Regression Class

```
class MyLinearRegression:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs

    def fit(self, X, Y):
        m, n = X.shape
        self.W = np.zeros((1, n))
        self.b = 0.0
        for i in range(self.epochs):
            Y_hat = np.dot(self.W, X.T) + self.b
            E = Y_hat - Y.T
            dW = (1/m) * np.dot(E, X)
            db = (1/m) * np.sum(E)
            self.W -= self.lr * dW
            self.b -= self.lr * db

    def predict(self, X):
        return np.dot(self.W, X.T) + self.b

# Example usage
model = MyLinearRegression(lr=0.01, epochs=1000)
model.fit(X[:,1:], Y)  # use only features (skip bias column)
preds = model.predict(X[:,1:])
print("Predictions shape:", preds.shape)
```

```
Predictions shape: (1, 415)
```