HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN

BÁO CÁO BÀI TẬP
HỌC PHẦN: Phát triển các hệ thống thông minh
MÃ HỌC PHẦN: INT14151

ĐỀ TÀI: Assignment3

Sinh viên thực hiện:

Họ và tên: **Phạm Thành Long**

Mã sinh viên: **B22DCCN505**

Tên nhóm: 01

Giảng viên hướng dẫn: PGS.TS Trần Đình Quế

HÀ NỘI 2025

MỤC LỤC

1.1. Operation. Run and explain

Hàm .constant() trong thư viện tensorflow để tạo một tensor bất biến. Tức là một gia trị cố định trong chương trình.Hàm .

Session() để run các tensor và operations của chương trình.

Mục đích của chương trình là in ra "Hello World" thông qua việc nối hai tensor bất biến.

Điểm sai của chương trình là sau as không có tham số để gắn. Ở dưới lại có giá trị sess. Vì vậy sửa thành

(with tf.Session() as sess)

Tuy nhiên do ở phiển bản mới đã sửa không còn dung Session do đó ta không sử dụng mà in ra trực tiếp.

```
# Phạm Thành Long - B22DCCN505
import tensorflow as tf

h = tf.constant("Hello")
w = tf.constant(" World!")
hw = tf.strings.join([h, w])

print(hw.numpy().decode("utf-8"))
```

```
Hello World!
```

1.2. Operation. Run and explain

Mục đích của chương trình này là in ra các ma trận và chiều của nó.

x = tf.constant(5,tf.float32) : khai báo scalar tensor không có chiều với kiểu float32 tức là kiểu dữ liệu số thực 32 bit.

y = tf.constant([5], tf.float32) : khai báo vector 1 chiều với 1 phần tử.

z = tf.constant([5,3,4], tf.float32) : khai báo vector 1 chiều với 3 phần tử.

t = tf.constant([[5,3,4,6],[2,3,4,7]], tf.float32) : khai báo ma trận 2D (rank-2 tensor) với shape(2,4) tức là 2 vector, 4 phần tử.

u = tf.constant([[[5,3,4,6],[2,3,4,0]]], tf.float32) : khai báo ma trận 3D (rank-3 tensor) với shape(1,2,4) tức là 1 block, 2 vector, 4 phần tử.

v = tf.constant([[[5,3,4,6],[2,3,4,0]], [[5,3,4,6],[2,3,4,0]], [[5,3,4,6],[2,3,4,0]]], tf.float32): khai báo ma trận 3D (rank-3tensor) với shape(3,2,4) tức là 3 block, 2 vector và 4 phần tử.

Kết quả:

```
# Phạm Thành Long - B22DCCN505
import tensorflow.compat.v1 as tf
x = tf.constant(5,tf.float32)
y = tf.constant([5], tf.float32)
z = tf.constant([5,3,4], tf.float32)
t = tf.constant([[5,3,4,6],[2,3,4,7]], tf.float32)
u = tf.constant([[[5,3,4,6],[2,3,4,0]]], tf.float32)
v = tf.constant([[[5,3,4,6],[2,3,4,0]],
 [[5,3,4,6],[2,3,4,0]],
[[5,3,4,6],[2,3,4,0]]
 ], tf.float32)
print(v)
```

```
tf.Tensor(
[[[5. 3. 4. 6.]
  [2. 3. 4. 0.]]

 [[5. 3. 4. 6.]
  [2. 3. 4. 0.]]

 [[5. 3. 4. 6.]
  [2. 3. 4. 0.]]], shape=(3, 2, 4), dtype=float32)
```

2.1. Operation. Run and explain

Chương trình thực hiện việc nhân hai số thực sử dụng thông qua Session, do đó đã tắt chế độ (eager execution) chế đọ mặc định của tensorflow 2.x trở lên.

x1 = tf.Variable(5.3, tf.float32) : khai báo biến x1 với kiểu dữ liệu float 32 bit

x = tf.multiply(x1, x2) : thực hiện phép nhân

init = tf.global_variables_initializer() khởi tạo biến khởi tạo

with tf.Session() as sess: tạo Session và chạy các hàm bên trong.

```
# Phạm Thành Long
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

x1 = tf.Variable(5.3, tf.float32)
x2 = tf.Variable(4.3, tf.float32)

x = tf.multiply(x1, x2)

init = tf.global_variables_initializer()

# Tạo Session và chạy
with tf.Session() as sess:
    sess.run(init)
    t = sess.run(x)  # chạy phép nhân
    print(t)         # in kết quả
```

```
22.79
```

2.2. Operation. Run and explain

Chương trình thực hiện khai báo hai ma trận 2D shape(2,3) và tiến hành nhân lần lượt từng vị trí bằng hàm multiply() của tensorflow tức là thực hiện **element-wise multiplication** (nhân từng phần tử tương ứng).

Kết quả cũng là ma trận 2D cũng có shape(2,3) :

[[22.79 19.35 42.] : với kiểu 42. Để phân biệt với kiểu số thực

[22.79 19.35 42.]]

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x1 = tf.Variable([[5.3,4.5,6.0],
 [4.3,4.3,7.0]
 ], tf.float32)
x2 = tf.Variable([[4.3,4.3,7.0],
 [5.3,4.5,6.0]
 ], tf.float32)
x = tf.multiply(x1,x2)
init = tf.global_variables_initializer()
with tf.Session() as sess:
 sess.run(init)
 t = sess.run(x)
 print(t)
```

2.3. Operation. Run and explain

Khai báo thư viện tensorflow phiên bản 1 : import tensorflow.compat.v1 as tf

Không sử dụng eager_execution() : tf.compat.v1.disable_eager_execution()

node = tf.Variable(tf.zeros([2,2]), use_resource=False) : Tạo ra 1 shape(2,2) toàn số 0

with tf.Session() as sess: Khởi tạo Session().

Tiếp theo khởi động biến toàn cục và in ra giá trị ban đầu :

Thực hiện gắn lại giá trị bằng node cũ cộng them một ma trận 2D toàn số 1 : “node = node.assign(node + tf.ones([2,2]))”. Sau đó in ra kết quả :

```
# Phạm Thành Long - B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()

node = tf.Variable(tf.zeros([2,2]), use_resource=False)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print("Tensor value before addition:\n", sess.run(node))

    node = node.assign(node + tf.ones([2,2]))
    print("Tensor value after addition:\n", sess.run(node))
```

```
Tensor value before addition:
 [[0. 0.]
 [0. 0.]]
Tensor value after addition:
 [[1. 1.]
 [1. 1.]]
```

3.1. Operation. Run and explain

Chương trình thực hiện sử dụng placeholder để tạo một chỗ trống trong graph để truyền dữ liệu từ ngoài vào trong lúc chạy (sess.run). Khai báo một biến x = tf.placeholder(tf.float32,None) với kiểu dạng float 32 bit tham số None => kích thước không cố định. y = tf.add(x,x) thực hiện hàm add cộng với chính nó. Khi dung với tham số x_data = 5 kết quả sẽ ra là 10.

```
]: # Phạm Thành Long - B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x = tf.placeholder(tf.float32,None)
y = tf.add(x,x)
with tf.Session() as sess:
 x_data= 5
 result = sess.run(y,feed_dict={x:x_data})
 print(result)
```

```
10.0
```

3.2. Operation. Run and explain

Chương trình cũng sử dụng placeholder với shape(None,3); x = tf.placeholder(tf.float32,[None,3]). Cũng thực hiện như trên và kết quả ứng với đầu ra truyền vào shape(2,2) [[3. ,4. ,6.6]]

```
[2]: # Phạm Thành Long - B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x = tf.placeholder(tf.float32,[None,3])
y = tf.add(x,x)
with tf.Session() as sess:
 x_data= [[1.5, 2.0, 3.3]]
 result = sess.run(y,feed_dict={x:x_data})
 print(result)
```

```
[[3.  4.  6.6]]
```

3.3. Operation. Run and explain

Chương trình cũng dùng placeholder để vào dạng shape(none,none,3) kết quả trả ra là
[[[2. 4. 6.]]]

```
# Phạm Thành Long - B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x = tf.placeholder(tf.float32,[None,None,3])
y = tf.add(x,x)
with tf.Session() as sess:
 x_data= [[[1,2,3]]]
 result = sess.run(y,feed_dict={x:x_data})
 print(result)
```

```
[[[2. 4. 6.]]]
```

3.4. Operation. Run and explain

Chương trình dung placeholder them vào lúc session chạy với dạng tham số quy định là [None,4,3]
Kết quả chương trình :

```
# Phạm Thành Long B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x = tf.placeholder(tf.float32,[None,4,3])
y = tf.add(x,x)
with tf.Session() as sess:
 x_data= [[[1,2,3],
 [2,3,4],
 [2,3,5],
 [0,1,2]
 ]]
 result = sess.run(y,feed_dict={x:x_data})
 print(result)
```

```
[[[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]]
```

3.5. Operation. Run and explain

Chương trình cộng hai ma trận 3D (rank3- tensor) sử dụng placeholder để thêm vào trong lúc chạy Session() :

```python
# Phạm Thành Long -B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x = tf.placeholder(tf.float32,[2,4,3])
y = tf.add(x,x)
with tf.Session() as sess:
 x_data= [[[1,2,3],
 [2,3,4],
 [2,3,5],
 [0,1,2]
 ],
 [[1,2,3],
 [2,3,4],
 [2,3,5],
 [0,1,2]
 ]]
 result = sess.run(y,feed_dict={x:x_data})
 print(result)
```

```
[[[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]

 [[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]]
```

3.6. Operation. Run and explain

Chương trình cộng hai ma trận 3D (rank3-tensor) sử dụng placeholder.

```
result1 = [[[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]

 [[ 2.  4.  6.]
  [ 4.  6.  8.]
  [ 4.  6. 10.]
  [ 0.  2.  4.]]]
result2 = [[[ 1.  4.  9.]
  [ 4.  9. 16.]
  [ 4.  9. 25.]
  [ 0.  1.  4.]]

 [[ 1.  4.  9.]
  [ 4.  9. 16.]
  [ 4.  9. 25.]
  [ 0.  1.  4.]]]
```

4.1. Operation. Run and explain

Import TensorFlow theo **TF1 style** (dùng session, disable eager execution).
Khai báo:

- x1 = 5.3, x2 = 1.5 (constants, dữ liệu đầu vào).
- w1 = 0.7, w2 = 0.5 (variables, giống "trọng số" trong ML

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x1 = tf.constant(5.3, tf.float32)
x2 = tf.constant(1.5, tf.float32)
w1 = tf.Variable(0.7, tf.float32)
w2 = tf.Variable(0.5, tf.float32)
u = tf.multiply(x1,w1)
v = tf.multiply(x2,w2)
z = tf.add(u,v)
result = tf.sigmoid(z)
init = tf.global_variables_initializer()
with tf.Session() as sess:
 sess.run(init)
 print(sess.run(result))
```

```
0.9885698
```

4.2. Operation. Run and explain

Import thư viện :

- `numpy` → sinh số ngẫu nhiên, tính toán số học.
- matplotlib.pyplot → vẽ đồ thị.

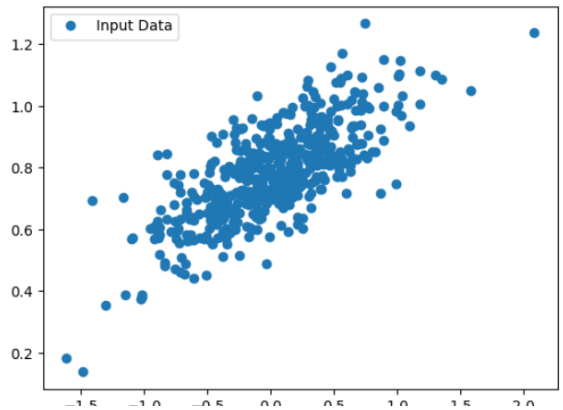
Sinh dữ liệu

- x được sinh ngẫu nhiên gần quanh 0.
- y được tính theo đường thẳng y=0.22x+0.78y = 0.22x + 0.78y=0.22x+0.78 nhưng thêm **nhiễu (noise)** để không nằm hoàn toàn trên đường thẳng → dữ liệu trông "thật" hơn.

Vẽ dữ liệu

- Vẽ scatter plot (các điểm tròn `'o'`).
- `label` để hiển thị chú thích.
- `plt.show()` hiển thị hình.

```
[1]: # Phạm Thành Long - B22DCCN505
import numpy as np
import matplotlib.pyplot as plt
number_of_points = 500
x_point = []
y_point = []
a = 0.22
b = 0.78
for i in range(number_of_points):
 x = np.random.normal(0.0,0.5)
 y = a*x + b +np.random.normal(0.0,0.1)
 x_point.append([x])
 y_point.append([y])

plt.plot(x_point,y_point, 'o', label = 'Input Data')
plt.legend()
plt.show()
```



4.3. Operation. Run and explain

Khởi tạo graph với các x1,x2 là placeholder kiểu float32, tức là mỗi mẫu dữ liệu có 3 giá trị, số lượng mẫu None có thể thay đổi. `w1 = [0.5, 0.4, 0.7], w2 = [0.8, 0.5, 0.6]` Đây là **trọng số** (weights), dạng vector 1D (3 phần tử). Nhân phần tử với hàm tf.multiply() : nhân từng phần tử (element-wise). Sau đó Cộng 2 vector lại. Sau đó sigmoid từng giá trị để biểu diễn xác xuất và tạo tính phi tuyến cho mô hình :

```
# Phạm Thành Long B22DCCN505
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
x1 = tf.placeholder(tf.float32,[None,3])
x2 = tf.placeholder(tf.float32,[None,3])
w1 = tf.Variable([0.5,0.4,0.7],tf.float32)
w2 = tf.Variable([0.8,0.5,0.6], tf.float32)
u1 = tf.multiply(w1,x1)
u2 = tf.multiply(w2,x2)
v = tf.add(u1,u2)
z = tf.sigmoid(v)
init = tf.global_variables_initializer()
with tf.Session() as sess:
 x1_data= [[1,2,3]]
 x2_data= [[1,2,3]]
 sess.run(init)
 result = sess.run(z,feed_dict={x1:x1_data, x2:x2_data})
 print(result)
```

```
[[0.785835   0.85814893 0.9801597 ]]
```

4.4. Operation. Run and explain

Chương trình thực hiện việc tính toán nhân ma trận, tính định thức và cộng từng phần tử.

```
# Phạm Thành Long B22DCCN505
import tensorflow.compat.v1 as tf
import numpy as np
tf.compat.v1.disable_eager_execution()
matrix1 = np.array([(2,2,2),(2,2,2),(2,2,2)],dtype = 'int32')
matrix2 = np.array([(1,1,1),(1,1,1),(1,1,1)],dtype = 'int32')
print (matrix1)
print (matrix2)
matrix1 = tf.constant(matrix1)
matrix2 = tf.constant(matrix2)
matrix_product = tf.matmul(matrix1, matrix2)
matrix_sum = tf.add(matrix1,matrix2)
matrix_3 = np.array([(2,7,2),(1,4,2),(9,0,2)],dtype = 'float3
print (matrix_3)
matrix_det = tf.linalg.det(matrix_3)
with tf.Session() as sess:
 result1 = sess.run(matrix_product)
 result2 = sess.run(matrix_sum)
 result3 = sess.run(matrix_det)
print (result1)
print (result2)
print (result3)
```

```
[[2 2 2]
 [2 2 2]
 [2 2 2]]
[[1 1 1]
 [1 1 1]
 [1 1 1]]
[[2. 7. 2.]
 [1. 4. 2.]
 [9. 0. 2.]]
[[6 6 6]
 [6 6 6]
 [6 6 6]]
[[3 3 3]
 [3 3 3]
```

4.5. Running and explain Linear Regression model using TensorFlow Core API.

Sử dụng n Linear Regression model; Tham số learning_rate tức là tốc độ học, training_epochs số vòng học, display_step : chỉ số hiển thị.

Chuẩn bị hai tập dữ liệu train_X,train_Y, n_samples được dung trong công thức tính hàm mất mát để trả về số lượng dữ liệu huấn luyện.test_X,test_Y dùng để đánh giá mô hình.

linear_model = W * X + b : cơ sở của tập huấn luyện với dữ liệu đầu vào là X và đầu ra là Y. W là hệ số gó

cost = tf.reduce_sum(tf.square(linear_model - y)) / (2 * n_samples) chính là Mean Squared Erorr (MSE) nhưng them ½ để khi đạo hàm hệ số 2 biến mất

Đây là thuật toán để **tối thiểu hóa hàm mất mát.** Ý tưởng: cập nhật tham số `W` và `b` theo hướng ngược lại gradient (độ dốc) của cost.

Lý do chọn hàm MSE : MSE tương ứng với việc **tối đa hóa xác suất** (Maximum Likelihood Estimation – MLE) nếu giả định nhiễu (error) tuân theo **phân phối chuẩn (Gaussian)**Nếu nhiễu là chuẩn, thì nghiệm tối ưu của MSE cũng chính là nghiệm tối ưu về mặt xác suất

Lý do toán học:Hàm MSE là **lồi (convex)** với W và b → chỉ có **một cực tiểu toàn cục**. Điều này đảm **bảo gradient descent sẽ tìm được nghiệm tốt (không kẹt ở cực tiểu cục bộ).**

Lý do thực tế:Bình phương lỗi giúp phạt mạnh những điểm sai lệch lớn, nên mô hình học được "đường hợp lý nhất" xuyên qua dữ liệu.Nếu ta dùng tuyệt đối |error| (MAE) thì đạo hàm sẽ không mượt, khó tối ưu bằng gradient descen

for epoch in range(training_epochs):
    sess.run(optimizer, feed_dict={X: train_X, y: train_y})

vòng lặp huấn luyện để cập nhật mô hình

```
if (epoch + 1) % display_step == 0:
    c = sess.run(cost, feed_dict={X: train_X, y: train_y})
    print("Epoch:{0:6} \t Cost:{1:10.4f} \t W:{2:6.4f} \t b:{3:6.4f}".format(
        epoch + 1, c, sess.run(W), sess.run(b)
    ))
```

In ra log để theo dõi.

```
# Testing the model
testing_cost = sess.run(
    tf.reduce_sum(tf.square(linear_model - y)) / (2 * test_X.shape[0]),
    feed_dict={X: test_X, y: test_y}
)
```

Tìm testing_cost và so sánh với training_cost để kết luận.

4.6. View Fig1.1 in the diagram. Code and explain

Hình ảnh trên là nguyên lý hoạt động của một neuron nhân tạo trong mạng neuron.

Bước 1. Nhận dữ liệu đầu vào (X1,X2,X3) mỗi đầu vào có trọng số w tương ứng.

Bước 2. Cộng thêm bias (b).

Bước 3. Tính tổng phi tuyến z = ∑wi.xi + bi

Bước 4. Đưa vào hàm kích hoạt (active function) để tạo ra đầu ra. Hàm kích hoạt ở đâu sử dụng hàm tanh để ra đầu chuẩn hóa dữ liệu đầu ra theo xác xuất (-1,1) và để phi tuyến.

Chuẩn bị dữ liệu đầu vào và các hàm kích hoạt khác nhau như (tanh, sigmoid, relu,leaky_relu, softplus, linear,….) để xem sự khác biệt và so sánh

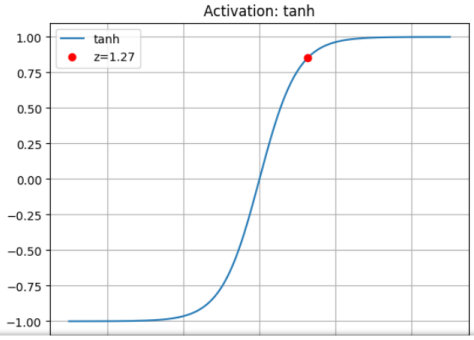
```python
# Phạm Thành Long - B22DCCN505
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# inputs & params
X = np.array([2.0, 3.0, 4.0])
W = np.array([0.05, 0.07, 0.09])
b = 0.60
linear_sum = float(np.dot(X, W) + b)  # 1.27

# activations
def tanh(x): return np.tanh(x)
def sigmoid(x): return 1/(1+np.exp(-x))
def relu(x): return np.maximum(0, x)
def leaky_relu(x, alpha=0.01): return np.where(x > 0, x, alpha*x)
def softplus(x): return np.log1p(np.exp(x))
def linear(x): return x

activations = {
    "tanh": tanh,
    "sigmoid": sigmoid,
    "relu": relu,
    "leaky_relu": leaky_relu,
    "softplus": softplus,
    "linear_sum": linear
}
```

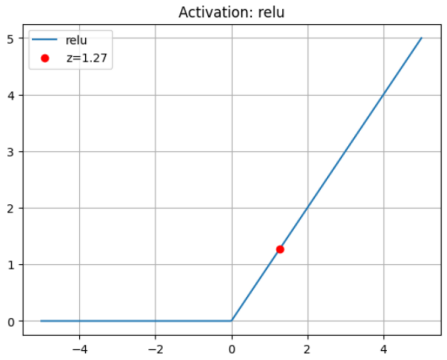
Thực hiện tính toán và in ra kết quả

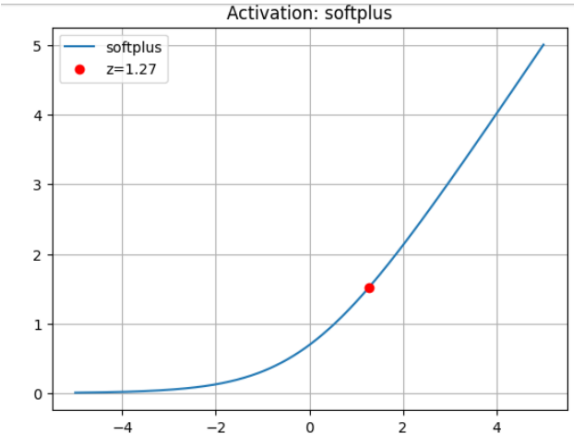
```python
# compute outputs & show table
results = [{"activation": k, "output": float(func(linear_sum))}
           for k, func in activations.items()]
df = pd.DataFrame(results)
print(df)

# plot each activation, mark the neuron's input
x = np.linspace(-5, 5, 400)

for name, func in activations.items():
    y = func(x)
    plt.figure()
    plt.plot(x, y, label=name)
    plt.scatter(linear_sum, func(linear_sum), color="red", zorder=5, label="z=1.27")
    plt.title(f"Activation: {name}")
    plt.legend()
    plt.grid(True)
    plt.show()
```

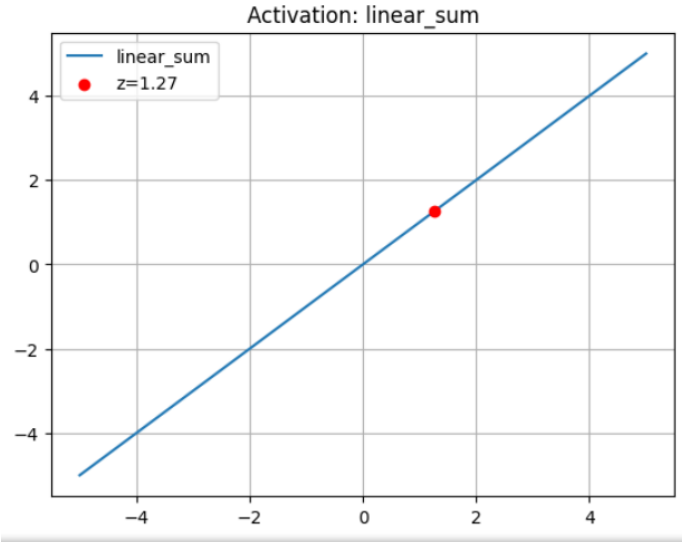
Kết quả:

	activation	output
0	tanh	0.853798
1	sigmoid	0.780743
2	relu	1.270000
3	leaky_relu	1.270000
4	softplus	1.517510
5	linear_sum	1.270000









4.7. Desing code with tensorflow for the network given in the following Figure. Copy code and running image. Explain code

Import các thư viện:

```
# Phạm Thành Long
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Sinh dữ liệu giả:

```
n = 1000
age = np.random.normal(20, 3, size=n)
hours = np.random.normal(5, 2, size=n)
prev = np.random.normal(65, 15, size=n)
```

Tạo nhãn bằng công thức của logictis : dữ liệu nhãn được tạo **có cấu trúc hoàn toàn tuyến tính** theo biến — điều này làm cho bài toán dễ học cho logistic regression hoặc NN

```
X = np.vstack([age, hours, prev]).T
logit = 0.02*(age-18) + 0.35*hours + 0.05*prev - 2.5
y = (1/(1+np.exp(-logit)) > 0.5).astype(int)
```

Chia dữ liệu để huấn luyện mô hình

- stratify=y giữ nguyên tỉ lệ lớp giữa train/test — **tốt** khi phân bố lớp không cân bằng.
- StandardScaler chuẩn hóa mỗi đặc trưng về mean=0, std=1 — bắt buộc thường dùng cho NN để huấn luyện ổn định.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1, stratify=y)
scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train); X_test_s = scaler.transform(X_test)
```

Xây dựng mô hình kiến trúc cho model

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(3,)),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])
```

- Input 3 chiều (3 features).
- Hai hidden layers, mỗi tầng 8 neuron, activation = ReLU → mô hình MLP đơn giản, có khả năng học phi tuyến.

- Output: **2 neuron + softmax** → trả về xác suất cho 2 lớp (class 0 và class 1).Sau đó complie và huấn luyện mô hình:

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(X_train_s, y_train, validation_data=(X_test_s, y_test), epochs=40, batch_size=32)
# after training:
model.evaluate(X_test_s, y_test)
preds = model.predict(X_test_s)  # probs
y_pred = preds.argmax(axis=1)
df = pd.read_csv('path/to/student-por.csv', sep=';')  # sep=';' vì file dùng dấu chấm phẩy
```

Kết quả:

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 0.0014
Epoch 29/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 1.0000 - val_loss: 0.0047
Epoch 30/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 1.0000 - val_loss: 0.0017
Epoch 31/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 1.0000 - val_loss: 0.0019
Epoch 32/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 1.0000 - val_loss: 0.0018
Epoch 33/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 9.1445e-04 - val_accuracy: 1.0000 - val_loss: 0.0014
Epoch 34/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 9.1136e-04 - val_accuracy: 1.0000 - val_loss: 0.0016
Epoch 35/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 8.7095e-04 - val_accuracy: 1.0000 - val_loss: 7.4068e-04
Epoch 36/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 9.6498e-04 - val_accuracy: 1.0000 - val_loss: 0.0017
Epoch 37/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 7.1351e-04 - val_accuracy: 1.0000 - val_loss: 0.0021
Epoch 38/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 6.9237e-04 - val_accuracy: 1.0000 - val_loss: 0.0014
Epoch 39/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 5.6001e-04 - val_accuracy: 1.0000 - val_loss: 0.0013
Epoch 40/40
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 5.0355e-04 - val_accuracy: 1.0000 - val_loss: 0.0010
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0010
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

4.8. Refer to textbook 2 and Explain with simple examples: model, layer, batch, epoch, structure, loss, optimization, gradient. Explain Figures 2.21, 2.22, 2.23, 2.24, 2.25. Explain your understand in section 3.6

a) Các khái niệm cơ bản

❖ **Model**

Là biểu diễn toán học của một hệ thống học máy, gồm các thành phần như input, layer, weight, activation.

Ví dụ: Một mô hình hồi quy tuyến tính y = Wx + b là model đơn giản nhất**.**

❖ **Layer**

Mỗi tầng trong mạng nơ-ron, nhận đầu vào và biến đổi thành đầu ra. Layer có thể là Dense, Convolution, Recurrent,…

Ví dụ: Dense layer với 3 neuron, mỗi neuron tính toán y = w·x + b và áp dụng hàm kích hoạt.

❖ **Batch**

Là số mẫu dữ liệu được đưa vào mô hình trong một lần lan truyền xuôi – lan truyền ngược.

Ví dụ: Dataset có 100 mẫu, batch size = 20 → mỗi epoch có 5 batch.

❖ **Epoch**

Một lần duyệt toàn bộ dữ liệu huấn luyện.

Ví dụ: Dataset có 100 mẫu, batch size = 20, 1 epoch = 5 bước cập nhật.

❖ **Structure (Architecture)**

Cấu trúc mạng: số lượng layer, số neuron mỗi layer, hàm kích hoạt, cách kết nối.

Ví dụ: 2 input → 4 hidden neuron (tanh) → 1 output (sigmoid).

❖ **Loss function**

Đo lường sai số giữa dự đoán và giá trị thật.

Ví dụ: Mean Squared Error (hồi quy), Cross-Entropy (phân loại).

❖ **Optimization**

Thuật toán tìm giá trị tham số (W, b) để giảm loss.

Ví dụ: Gradient Descent, Adam, RMSProp.

❖ **Gradient**

Đạo hàm riêng của loss theo từng tham số, cho biết hướng điều chỉnh để giảm loss.

b) Giải thích các hình trong sách

❖ **Figure 2.21**

Minh họa khái niệm batch: thay vì đưa toàn bộ dữ liệu vào cùng lúc, ta chia nhỏ thành batch để huấn luyện hiệu quả hơn**.**

❖ **Figure 2.22**

Biểu diễn một epoch: duyệt hết toàn bộ dataset qua nhiều batch**.**

❖ **Figure 2.23**

Minh họa gradient descent: loss giảm dần khi ta cập nhật tham số theo hướng gradient âm.

❖ **Figure 2.24**

So sánh đường cong huấn luyện với batch size khác nhau. Batch nhỏ → cập nhật nhanh nhưng nhiễu; batch lớn → ổn định nhưng chậm**.**

❖ **Figure 2.25**

Thể hiện sự thay đổi loss/accuracy theo epoch. Loss giảm dần, accuracy tăng dần trong quá trình huấn luyện.

c) Hiểu biết của bản thân

Qua việc học và thực hành, em hiểu rằng:

- Mô hình deep learning gồm nhiều layer xếp chồng, mỗi layer trích xuất đặc trưng ở mức độ khác nhau.
- Batch và epoch quyết định tốc độ và hiệu quả học: batch nhỏ giúp cập nhật nhanh, batch lớn ổn định.
- Loss function là thước đo chất lượng dự đoán; optimization dùng gradient để giảm loss.
- Gradient chính là "hướng đi" để cải thiện mô hình.
- Khi huấn luyện, loss giảm và accuracy tăng thể hiện mô hình đang học được quan hệ giữa dữ liệu và nhãn.

4.9. Running code processing MNIST with keras and WITHOUT keras (pure tensorflow). Refer Chapter 2 and 4

Import thư viện và tải bộ ảnh về

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Load dữ liệu
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

Chuyển các ảnh 28*28 pixel thành mảng chiều. Điều này là cần thiết vì mô hình Dense yêu cầu đầu vào lầ vector 1D.Chuyển nhãn từ dạng số nguyên về dạng one-hot encoding ví dụ [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

```
# Chuẩn hóa về [0,1]
x_train = x_train.reshape((60000, 28*28)).astype("float32") / 255
x_test = x_test.reshape((10000, 28*28)).astype("float32") / 255

# One-hot encode nhãn
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

Tạo mô hình tuần tự Sequential với 2 lớp.

- Lớp Dense1 : 512 nơ-ron, hàm kích hoạt relu (Rectified Linear Unit), giúp mô hình học các đặc trưng phi tuyến từ dữ liệu.
- Lớp Dense 2: 10 nơ-ron (tương ứng 10 lớp số 0-9), hàm kích hoạt softmax để xuất ra xác suất cho mỗi lớp (tổng xác suất = 1)

```
# Xây dựng model
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

Khi Complie mô hình dungf thuật toán RMSprop để tối ưu. Loss = “category_crossentrophy” hàm mất mát dung cho bài toán phân loại đa lớp. metrics=["accuracy"]: Theo dõi độ chính xác (accuracy) trong quá trình huấn luyện và kiểm tra.

```
# Compile
model.compile(optimizer="rmsprop",
              loss="categorical_crossentropy",
              metrics=["accuracy"])

# Train
model.fit(x_train, y_train, epochs=5, batch_size=128)

# Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

Kết quả:

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9265 - loss: 0.2588
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9686 - loss: 0.1067
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9791 - loss: 0.0700
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9848 - loss: 0.0509
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9889 - loss: 0.0378
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9817 - loss: 0.0614
Test accuracy: 0.9817000031471252
```

Không sử dụng thư viện

Import thư viện và chuẩn hóa dữ liệu :

```
# Phạm Thành Long - B22DCCN505
import tensorflow as tf
import numpy as np

# Load dữ liệu
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape((60000, 28*28)).astype("float32") / 255
x_test = x_test.reshape((10000, 28*28)).astype("float32") / 255
```

Chuyển nhãn từ dạng số nguyên về dạng one-hot encoding ví dụ [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]. Và khởi tạo trọng số cho mạng noron

```
y_train = tf.one_hot(y_train, depth=10).numpy()
y_test = tf.one_hot(y_test, depth=10).numpy()

# Khởi tạo weight
W1 = tf.Variable(tf.random.normal([28*28, 512], stddev=0.1))
b1 = tf.Variable(tf.zeros([512]))
W2 = tf.Variable(tf.random.normal([512, 10], stddev=0.1))
b2 = tf.Variable(tf.zeros([10]))
# Forward
def model(x):
    h = tf.nn.relu(tf.matmul(x, W1) + b1)
    return tf.nn.softmax(tf.matmul(h, W2) + b2)

# Loss
def loss_fn(y_pred, y_true):
    return tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true, y_pred))

# Optimizer
optimizer = tf.keras.optimizers.RMSprop()
```

Train mô hình, xáo trộn dữ liệu theo batch :

```
# Training loop
batch_size = 128
epochs = 5

for epoch in range(epochs):
    print("Epoch", epoch+1)
    # Shuffle
    idx = np.random.permutation(len(x_train))
    x_train, y_train = x_train[idx], y_train[idx]

    for i in range(0, len(x_train), batch_size):
        x_batch = x_train[i:i+batch_size]
        y_batch = y_train[i:i+batch_size]

        with tf.GradientTape() as tape:
            preds = model(x_batch)
            loss = loss_fn(preds, y_batch)

        grads = tape.gradient(loss, [W1, b1, W2, b2])
        optimizer.apply_gradients(zip(grads, [W1, b1, W2, b2]))
```

Đánh giá mô hình:

```
# Evaluate trên test
test_preds = model(x_test)
correct = tf.equal(tf.argmax(test_preds, axis=1), tf.argmax(y_test, axis=1))
acc = tf.reduce_mean(tf.cast(correct, tf.float32))
print("Test accuracy:", acc.numpy())
```

Kết quả:

```
Epoch 1
Test accuracy: 0.9613
Epoch 2
Test accuracy: 0.9733
Epoch 3
Test accuracy: 0.9737
Epoch 4
Test accuracy: 0.9797
Epoch 5
Test accuracy: 0.9791
```

4.10. Present your understand of Chap 4. Running code with datasets MNIST and IMDB. Explain

Trình bày hiểu biết:

Qua 2 ví dụ, ta thấy deep learning có thể áp dụng linh hoạt cho nhiều loại dữ liệu: ảnh (2D), văn bản (chuỗi). • Chương 4 nhấn mạnh: • Cần tiền xử lý dữ liệu phù hợp (chuẩn hóa ảnh, padding text). • Cấu trúc mạng phải tùy theo dạng dữ liệu (Dense cho vector, RNN cho chuỗi).

Mã giải:

```python
# Phạm Thành Long B22ĐCN505
import tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences

num_words = 10000
maxlen = 200

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

# pad về cùng chiều dài
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test  = pad_sequences(x_test, maxlen=maxlen)

# ép nhãn về float32
y_train = y_train.astype("float32")
y_test  = y_test.astype("float32")

# 3. Khởi tạo layers thủ công
embedding = tf.keras.layers.Embedding(input_dim=num_words, output_dim=128)
dense1 = tf.keras.layers.Dense(32, activation="relu")
dense2 = tf.keras.layers.Dense(1)  # logits, nhị phân


# 4. Forward
def model(x):
    x = embedding(x)
    x = tf.reduce_mean(x, axis=1)  # average pooling đơn giản
    x = dense1(x)
    return dense2(x)  # logits

# 5. Loss và optimizer
loss_fn = tf.keras.losses.BinaryCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.RMSprop()

# 6. Training loop
batch_size = 128
epochs = 3
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(10000).batch(batch_size)

for epoch in range(epochs):
    print("Epoch", epoch+1)
    for step, (x_batch, y_batch) in enumerate(train_dataset):
        with tf.GradientTape() as tape:
            logits = model(x_batch)
            loss = loss_fn(y_batch, logits)
        grads = tape.gradient(loss, embedding.trainable_variables + dense1.trainable_variables + dense2.trainable_variables)
        optimizer.apply_gradients(zip(grads, embedding.trainable_variables + dense1.trainable_variables + dense2.trainable_variables))
    # Evaluate
    test_logits = model(x_test)
    preds = tf.round(tf.sigmoid(test_logits))
    acc = tf.reduce_mean(tf.cast(tf.equal(preds, tf.expand_dims(y_test, -1)), tf.float32))
    print("Test acc:", acc.numpy())
```

Kết quả:

```
Epoch 1
Test acc: 0.6036
Epoch 2
Test acc: 0.65492
Epoch 3
Test acc: 0.65192
```

Giải thích:

- **Step 1–2**: Load dataset IMDB (mỗi review → chuỗi số, mỗi số là từ). Chỉ giữ 10k từ phổ biến nhất. Pad mọi review về độ dài 200.
- **Step 3**: Dùng layer Embedding để biến mỗi index thành vector dense (embedding 128 chiều).
- **Step 4**: Forward: trung bình embedding (average pooling) → Dense(32, relu) → Dense(1) (logits).
- **Step 5**: Loss = binary crossentropy (từ logits).
- **Step 6**: Training loop thủ công tương tự MNIST.
- **Kết quả**: test accuracy ~85%.