Họ và tên : Hoàng Đạo Thông

MSSV : 2174802010149

# FINAL EXAMINATION

- Các bạn có thời gian làm bài là 2.5h (Từ 9h35 - 12h00)
- Chỉ được sử dụng tài liệu có sẵn và không được sử dụng những tài liệu tham khảo là AI(ChatGPT, Gemini,...)
- Cố gắng làm hết khả năng của mình nha :)

## Câu 1: 3 Điểm

Tạo một tập dữ liệu với các ví dụ $x^i \in R^2$ and $y^i \in \{0,1\}$.

Lớp 0 nên đến từ một phân phối Gaussian 2D với trung bình $\begin{bmatrix} 10 \\ 5 \end{bmatrix}$ và hiệp phương sai $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$.

Lớp 1 nên đến từ một phân phối Gaussian 2D với trung bình $\begin{bmatrix} 5 \\ 10 \end{bmatrix}$ và hiệp phương sai $\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$.

```
#Khởi tạo data
import numpy as np
X0 = np.random.multivariate_normal([10, 5], [[2, 0], [0, 2]], 100)
X1 = np.random.multivariate_normal([5, 10], [[2, 0], [0, 2]], 100)
X = np.concatenate((X0, X1), 0)
y = np.concatenate((np.zeros((100)), np.ones((100))))
```

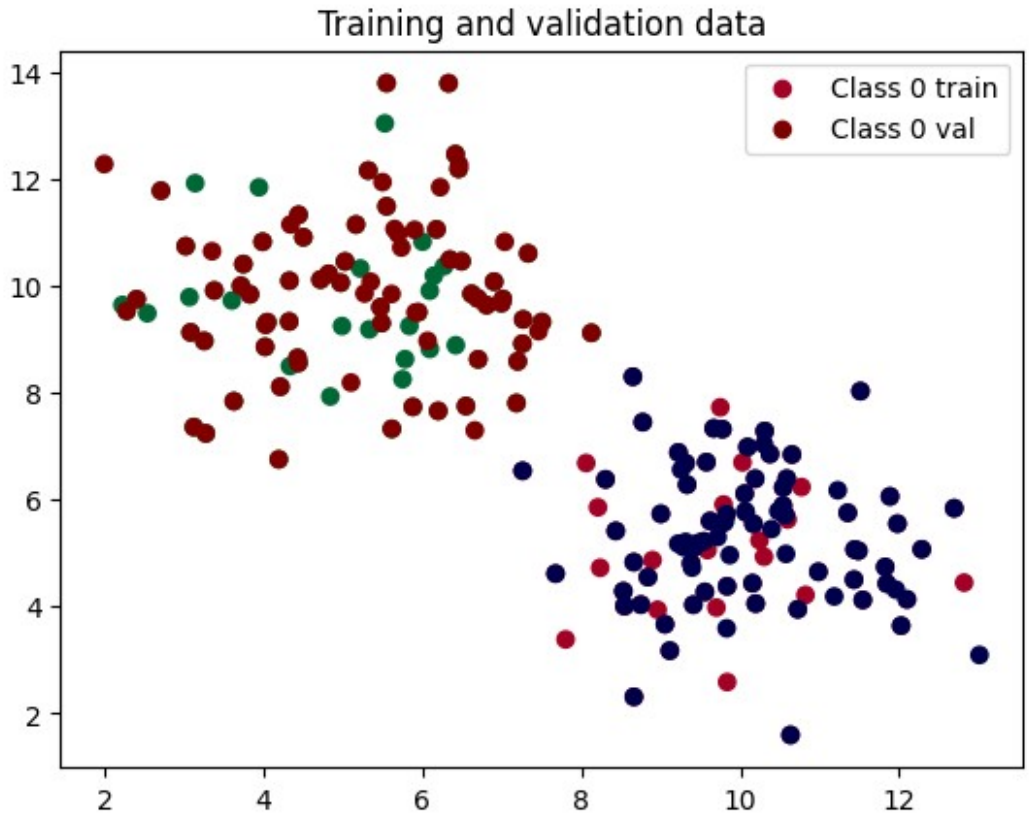
Chia tập dữ liệu với tập train (80%) và test (20%).

```
index = np.random.permutation(200)
train_index = index[:160]
val_index = index[40:]
X_train = X[train_index]
X_val = X[val_index]
y_train = y[train_index]
y_val = y[val_index]
```

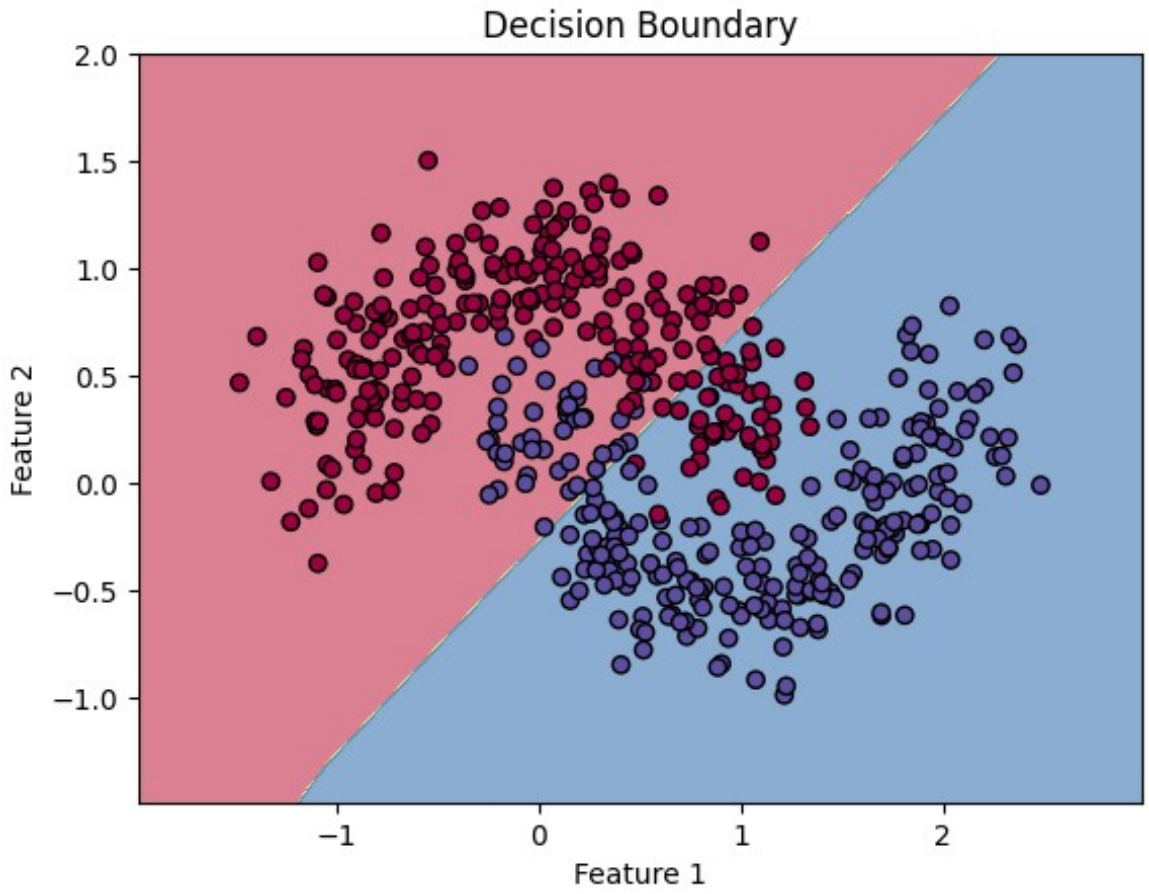
Hiển thị biểu đồ phân tán với lớp 0 và lớp 1 được hiển thị bằng các màu khác nhau

```
#code here
import matplotlib.pyplot as plt

plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='RdYlGn')
plt.scatter(X_val[:, 0], X_val[:, 1], c=y_val, cmap='seismic')
plt.plot()
```

```
plt.title('Training and validation data')
plt.legend(['Class 0 train','Class 0 val','Class 1 train','Class 1
val'])
plt.show()
```




Kết quả sẽ ra thế này:




# Câu 2: 2 Điểm

```python
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

X = np.array([[1, 2], [2, 3], [3, 3], [6, 5], [7, 8], [8, 9]])
y = np.array([0, 0, 0, 1, 1, 1])
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.8, random_state=42)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
X_train = np.fit_transform(x_train)
X_test = np.transform(x_test)

# Tạo mô hình
model = RandomForestClassifier(n_estimators=100, max_depth=3,
min_samples_split=2, random_state=42)
```

```
# Huấn luyện mô hình
model.fit(X_test, y)

# Dự đoán
y_pred = model.predict(y_test)

# Đánh giá mô hình
accuracy = accuracy_score(X_test, y_pred)
print(f"Độ chính xác của mô hình: {accuracy * 100:.2f}%")
```

Đoạn code trên đúng hay sai hay bất thường? Nếu sai hoặc bất thường thì sửa lại như thế nào cho đúng ?

```
# code here
# code here
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

X = np.array([[1, 2], [2, 3], [3, 3], [6, 5], [7, 8], [8, 9]])
y = np.array([0, 0, 0, 1, 1, 1])
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Chuẩn hóa dữ liệu
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Tạo mô hình
model = RandomForestClassifier(n_estimators=100, max_depth=3,
min_samples_split=2, random_state=42)

# Huấn luyện mô hình
model.fit(X_train, y_train)

# Dự đoán
y_pred = model.predict(X_test)

# Đánh giá mô hình
accuracy = accuracy_score(y_test, y_pred)
print(f"Độ chính xác của mô hình: {accuracy * 100:.2f}%")

Độ chính xác của mô hình: 100.00%
```

# Câu 3: 5 Điểm

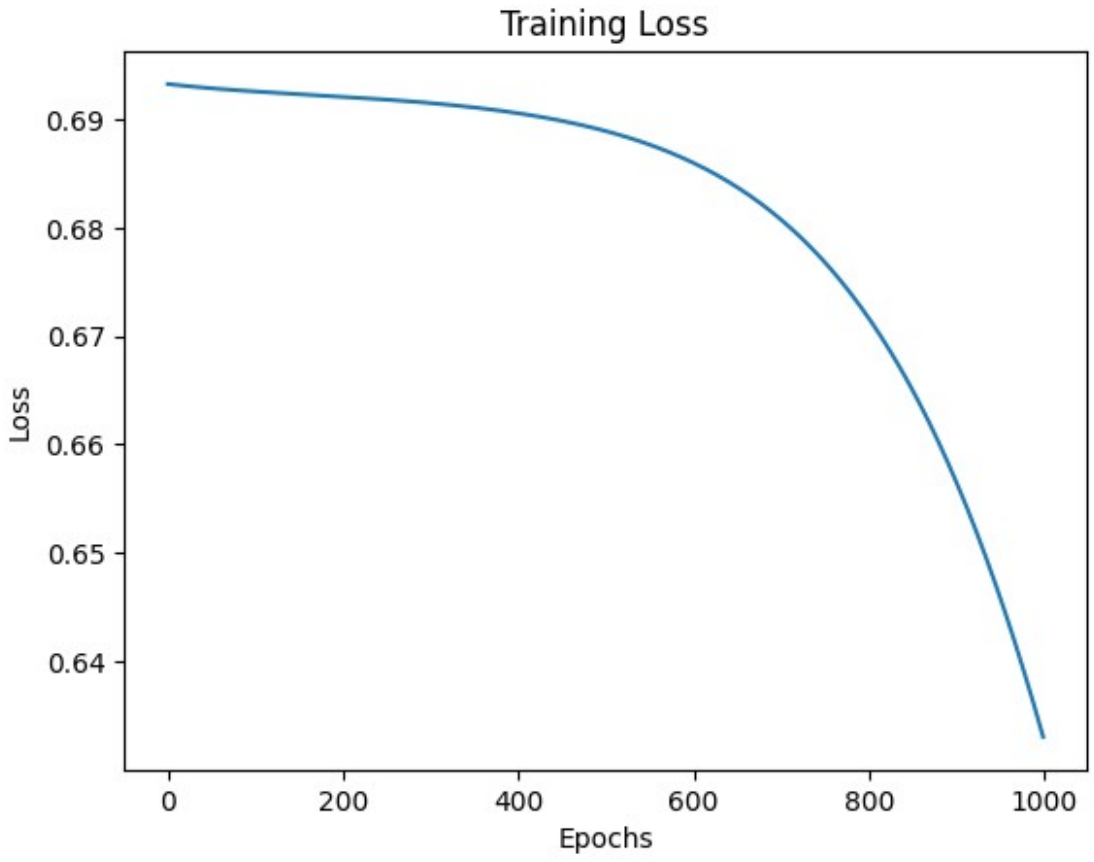
Điền vào chỗ trống để mô hình MLP bên dưới có thể run chính xác

```
import torch
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder

# Tạo dữ liệu mẫu
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
y = y.reshape(-1, 1) #code here # Chuyển y thành vector cột

# One-hot encode nhãn
encoder = OneHotEncoder(sparse_output=False)
y_onehot = encoder.fit_transform(y) #code here

# Chia dữ liệu thành tập train/test với train =80%
X_train, X_test, y_train, y_test = train_test_split(X, y_onehot,
test_size=0.2, random_state=42) #code here

# Chuyển dữ liệu sang tensor (Float32)
X_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train).float()
X_test = torch.from_numpy(X_test).float()
y_test = torch.from_numpy(y_test).float()

#hyperparameters
input_size = X_train.shape[1] #code here
hidden_size = 16
output_size = y_train.shape[1] #code here
learning_rate = 0.01 #code here
epochs = 1000

# Khởi tạo trọng số và bias
W1 = torch.randn(input_size, hidden_size, dtype=torch.float32) * 0.01
b1 = torch.zeros(hidden_size, dtype=torch.float32)
W2 = torch.randn(hidden_size, output_size, dtype=torch.float32) * 0.01
b2 = torch.zeros(output_size, dtype=torch.float32)

# Hàm kích hoạt
def relu(x):
    return torch.maximum(torch.tensor(0), x)

def relu_derivative(x):
    return (x > 0).float()

def softmax(x):
```

```
    exp_x = torch.exp(x)
    return exp_x / exp_x.sum(dim=1, keepdim=True)

# Hàm mâ´t mát (Cross-Entropy)
def cross_entropy_loss(y_pred, y_true):
    return -torch.sum(y_true * torch.log(y_pred)) / y_pred.shape[0]

# Huâ´n luyện
losses = []
for epoch in range(epochs):
    # Forward pass
    z1 = X_train @ W1 + b1
    a1 = relu(z1)
    z2 = a1 @ W2 + b2
    y_pred = softmax(z2)

    # Tính loss
    loss = cross_entropy_loss(y_pred, y_train)
    losses.append(loss.item())

    # Backward pass
    dz2 = y_pred - y_train
    dW2 = a1.T @ dz2 / X_train.shape[0]
    db2 = dz2.mean(dim=0)

    da1 = dz2 @ W2.T
    dz1 = da1 * relu_derivative(z1)
    dW1 = X_train.T @ dz1 / X_train.shape[0]
    db1 = dz1.mean(dim=0)

    # Cập nhật trọng sô´
    W1 -= learning_rate * dW1 #code here
    b1 -= learning_rate * db1 #code here
    W2 -= learning_rate * dW2 #code here
    b2 -= learning_rate * db2 #code here

    # In loss mô˜i 100 epochs
    if epoch % 100 == 0:
        #code here
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")

# Đánh giá trên tập test
with torch.no_grad():
    z1 = X_test @ W1 + b1
    a1 = relu(z1)
    z2 = a1 @ W2 + b2
    y_test_pred = softmax(z2)
    y_test_pred_class = torch.argmax(y_test_pred, axis=1)
    y_test_true_class = torch.argmax(y_test, axis=1)
    accuracy = (y_test_pred_class == y_test_true_class).float().mean()
```

```
    print(f"Accuracy on test set: {accuracy.item():.4f}")

# Vẽ biểu đồ loss
#code here
plt.plot(losses)
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```

```
Epoch 0, Loss: 0.6932
Epoch 100, Loss: 0.6926
Epoch 200, Loss: 0.6921
Epoch 300, Loss: 0.6915
Epoch 400, Loss: 0.6905
Epoch 500, Loss: 0.6889
Epoch 600, Loss: 0.6860
Epoch 700, Loss: 0.6807
Epoch 800, Loss: 0.6716
Epoch 900, Loss: 0.6565
Accuracy on test set: 0.7500
```




Kết quả tham khảo:

Epoch 0, Loss: 0.6930

Epoch 100, Loss: 0.6924

Epoch 200, Loss: 0.6919

Epoch 300, Loss: 0.6911

Epoch 400, Loss: 0.6897

Epoch 500, Loss: 0.6870

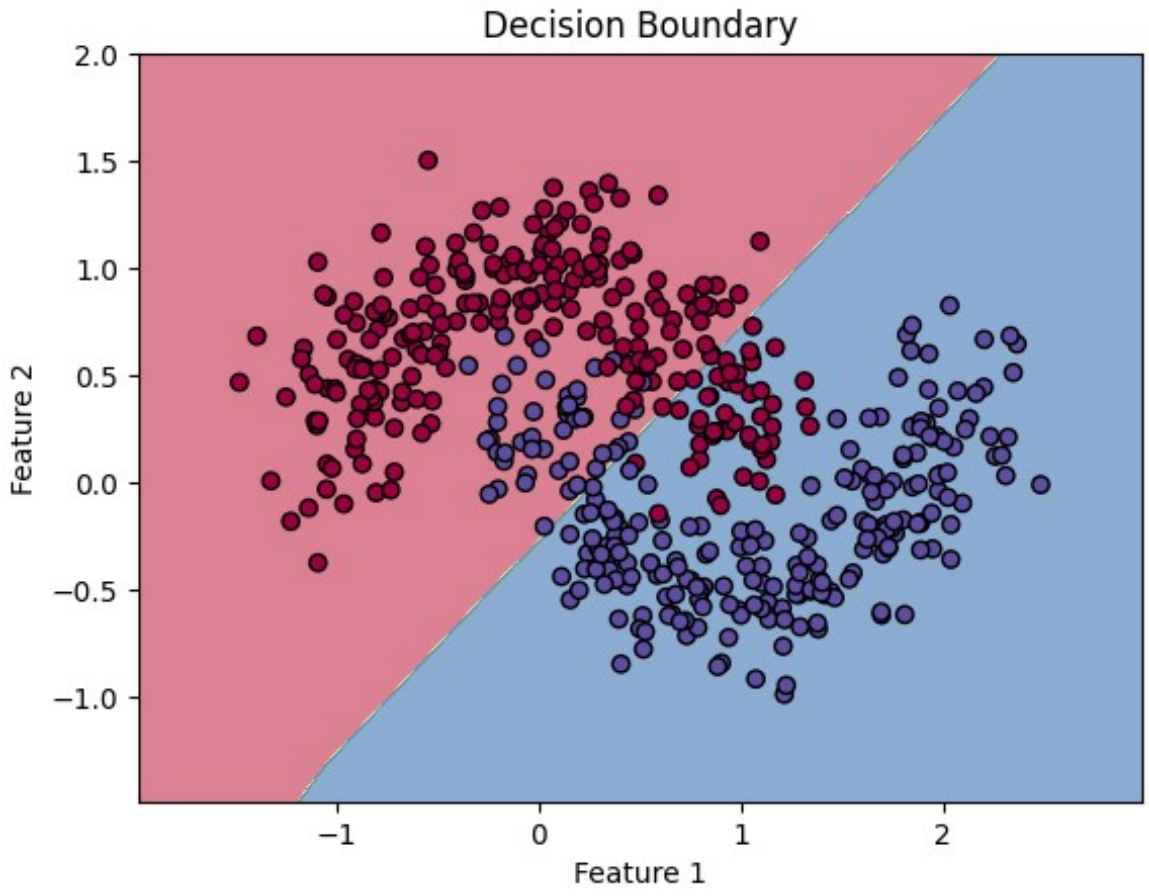
Epoch 600, Loss: 0.6821

Epoch 700, Loss: 0.6731

Epoch 800, Loss: 0.6573

Epoch 900, Loss: 0.6314

Accuracy on test set: 0.8400

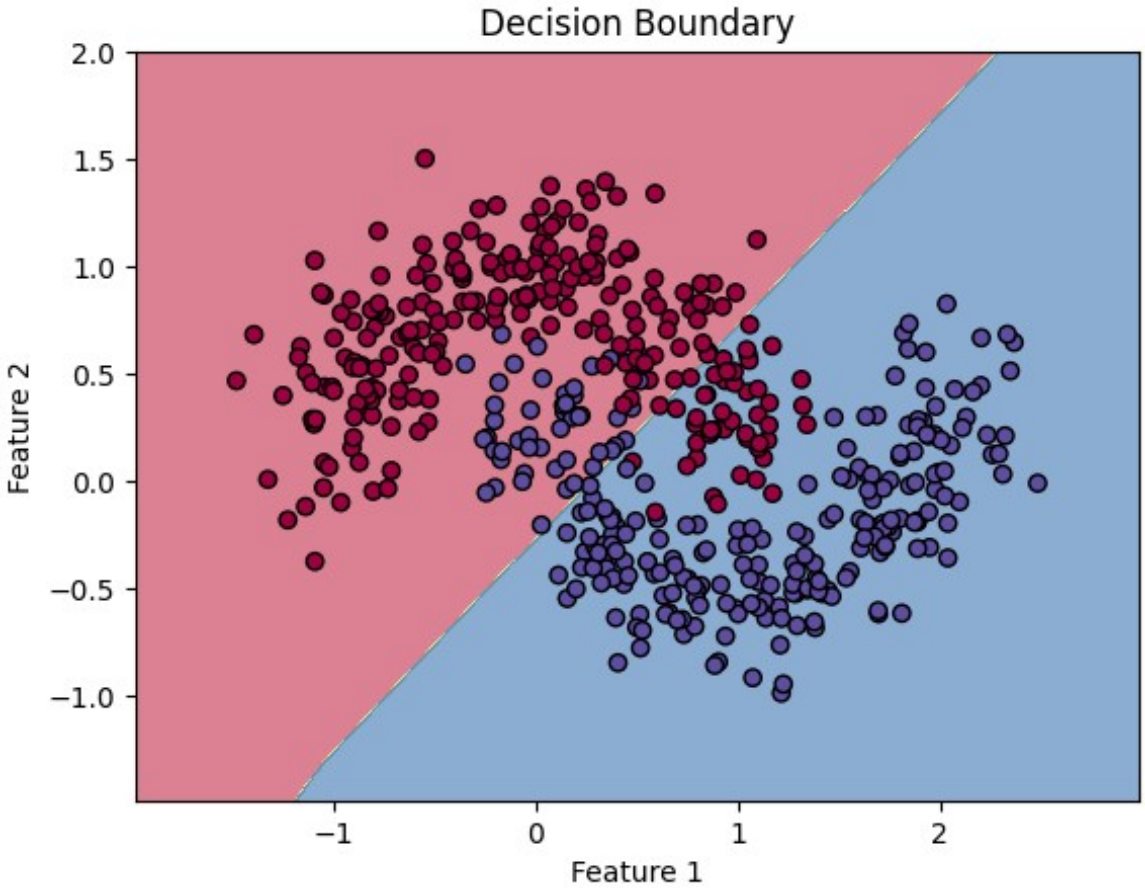



```
##--------------Phần này không cần code, chỉ run cho giống đáp án dưới
def plot_decision_boundary(X, y, model):
```

```
    # Đảm bảo X và y ở dạng NumPy array
    X = X.numpy() if isinstance(X, torch.Tensor) else X
    y = y.numpy().flatten() if isinstance(y, torch.Tensor) else
y.flatten()
    # Tạo lưới điểm để vẽ
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(
        np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01)
    )
    grid = np.c_[xx.ravel(), yy.ravel()]
    grid_tensor = torch.tensor(grid, dtype=torch.float32)
    with torch.no_grad():
        z1 = grid_tensor @ model["W1"] + model["b1"]
        a1 = relu(z1)
        z2 = a1 @ model["W2"] + model["b2"]
        preds = torch.argmax(softmax(z2), axis=1).numpy()

    # Chuyển về định dạng lưới
    preds = preds.reshape(xx.shape)
    # Vẽ ranh giới
    plt.contourf(xx, yy, preds, alpha=0.6, cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor="k",
cmap=plt.cm.Spectral)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("Decision Boundary")
    plt.show()
plot_decision_boundary(X, y, model)
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[2], line 29
     27     plt.title("Decision Boundary")
     28     plt.show()
---> 29 plot_decision_boundary(X, y, model)

NameError: name 'model' is not defined
```

Tham khảo kết quả


Decision Boundary
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
Feature 2
−1
0
1
2
Feature 1