

model 1

```
# Xây dựng mô hình ANN cơ bản
class ANN1(nn.Module):
    def __init__(self):
        super(ANN1, self).__init__()
        self.layer1 = nn.Linear(2,8) # Đầu vào 2, ẩn 4
        self.relu = nn.ReLU() # Công tắc ReLU
        self.layer2 = nn.Linear(8,1) # Ẩn 4, đầu ra 1
        self.sigmoid = nn.Sigmoid() # Xác suất 0-1

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        x = self.sigmoid(x)
        return x

# Khởi tạo mô hình
modell = ANN1()

# Định nghĩa mất mát và tối ưu hóa
criterion = nn.BCELoss()
optimizer = optim.Adam(modell.parameters(), lr=0.01)

# Huấn luyện
epochs = 100
for epoch in range(epochs):
    modell.train()
    # Xóa gradient cũ
    optimizer.zero_grad()
    # Dự đoán
    y_pred = modell(X_train)
    # Tính sai lầm
    loss = criterion(y_pred, y_train)
    # Lan truyền ngược
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 20 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], loss: {loss.item():.4f}")
```

```
Epoch [20/100], loss: 0.5349
Epoch [40/100], loss: 0.4027
Epoch [60/100], loss: 0.2889
Epoch [80/100], loss: 0.2052
Epoch [100/100], loss: 0.1452
```

```
# Kiểm tra
modell.eval()
with torch.no_grad():
    y_pred = modell(X_test)
    y_pred = (y_pred >= 0.5).float()# Chuyển thành 0 hoặc 1
    accuracy = (y_pred == y_test).float().mean()
    print(f"Độ chính xác: {accuracy*100:.2f}%")
```

Pyth

... Độ chính xác: 98.33%

model 2

```
# Xây dựng mô hình ANN cơ bản
class ANN12(nn.Module):
    def __init__(self):
        super(ANN12, self).__init__()
        self.layer1 = nn.Linear(2,8) # Đầu vào 2, ẩn 4
        self.relu = nn.ReLU() # Công tắc ReLU
        self.layer2 = nn.Linear(8,6) # Ẩn 4, đầu ra 1
        self.relu = nn.ReLU() # Công tắc ReLU
        self.layer3 = nn.Linear(6,1) # Ẩn 4, đầu ra 1
        self.sigmoid = nn.Sigmoid() # Xuất 0-1

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        x = self.relu(x)
        x = self.layer3(x)
        x = self.sigmoid(x)
        return x

# Khởi tạo mô hình
model12 = ANN12()

# Định nghĩa mất mát và tối ưu hóa
criterion = nn.BCELoss()
optimizer = optim.Adam(model12.parameters(), lr=0.01)

# Huấn luyện
epochs = 100
for epoch in range(epochs):
    model12.train()
    # Xóa gradient cũ
    optimizer.zero_grad()
    # Dự đoán
    y_pred = model12(X_train)
    # Tính sai lầm
    loss = criterion(y_pred, y_train)
    # Lan truyền ngược
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 20 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], loss: {loss.item():.4f}")
```

```
Epoch [20/100], loss: 0.5997
Epoch [40/100], loss: 0.4960
Epoch [60/100], loss: 0.4460
Epoch [80/100], loss: 0.3944
Epoch [100/100], loss: 0.3126
```

```
# Kiểm tra
model12.eval()
with torch.no_grad():
    y_pred = model12(X_test)
    y_pred = (y_pred >= 0.5).float() # Chuyển thành 0 hoặc 1
    accuracy = (y_pred == y_test).float().mean()
    print(f"Độ chính xác: {accuracy*100:.2f}%")
```

Độ chính xác: 86.67%

Nhận xét:

Loss càng giảm dần

- Độ chính xác tăng dần từ 66% => 100%

Độ chính xác:

- Mô hình 4 và 8 nút đều có cùng độ chính xác (98,33%).
- Mô hình 8+6 nút có độ chính xác thấp hơn có thể do overfitting.

loss càng giảm dần độ chính xác tăng dần từ 66% => 100%

Phần 2: Thử nghiệm với hàm mất mát và tối ưu hóa

Yêu cầu

1. Dùng BCEWithLogitsLoss thay cho BCELoss:

- Thay `nn.BCELoss()` bằng `nn.BCEWithLogitsLoss()`.
- Xóa hàm Sigmoid khỏi lớp đầu ra của mô hình (vì `BCEWithLogitsLoss` tự xử lý).
- Huấn luyện lại mô hình với cấu trúc ban đầu (2-4-1, 100 epochs).
- Ghi lại mất mát cuối cùng và độ chính xác.

2. Thay Adam bằng SGD:

- Dùng lại cấu trúc ban đầu (2-4-1) với `nn.BCELoss()`.
- Thay `optim.Adam` bằng `optim.SGD` với `lr=0.01`.
- Huấn luyện lại (100 epochs).
- Ghi lại mất mát cuối cùng và độ chính xác.

3. Câu hỏi:

- So sánh kết quả:
 - `BCEWithLogitsLoss`: có khác gì so với `BCELoss` về mất mát và độ chính xác? Tại sao?
 - `SGD` so với `Adam`: Mất mát giảm nhanh hơn hay chậm hơn? Độ chính xác thay đổi ra sao?
- Viết câu trả lời trong ô Markdown.

###1

###1

```
# Xây dựng mô hình ANN cơ bản
class ANN21(nn.Module):
    def __init__(self):
        super(ANN21, self).__init__()
        self.layer1 = nn.Linear(2,4) # Đầu vào 2, ẩn 4
        self.relu = nn.ReLU() # Công tắc ReLU
        self.layer2 = nn.Linear(4,1) # Ẩn 4, đầu ra 1

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x

# Khởi tạo mô hình
model21 = ANN21()

# Định nghĩa mất mát và tối ưu hóa
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model21.parameters(), lr=0.01)

# Huấn luyện
epochs = 100
for epoch in range(epochs):
    model21.train()
    # Xóa gradient cũ
    optimizer.zero_grad()
    # Dự đoán
    y_pred = model21(X_train)
    # Tính sai lầm
    loss = criterion(y_pred, y_train)
    # Lan truyền ngược
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 20 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], loss: {loss.item():.4f}")
```

```
... Epoch [20/100], loss: 0.6166
Epoch [40/100], loss: 0.5268
Epoch [60/100], loss: 0.4080
Epoch [80/100], loss: 0.3019
Epoch [100/100], loss: 0.2231
```

```

# Kiểm tra
model21.eval()
with torch.no_grad():
    y_pred = model21(X_test)
    y_pred = (y_pred >= 0.5).float()# Chuyển thành 0 hoặc 1
    accuracy = (y_pred == y_test).float().mean()
    print(f"Độ chính xác: {accuracy*100:.2f}%")

```

Độ chính xác: 95.00%

###2

```

# Xây dựng mô hình ANN cơ bản
class ANN22(nn.Module):
    def __init__(self):
        super(ANN22, self).__init__()
        self.layer1 = nn.Linear(2,4) # Đầu vào 2, ẩn 4
        self.relu = nn.ReLU() # Công tắc ReLU
        self.layer2 = nn.Linear(4,1) # Ẩn 4, đầu ra 1
        self.sigmoid = nn.Sigmoid() # Xác suất 0-1

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        x = self.sigmoid(x)
        return x

# Khởi tạo mô hình
model22 = ANN22()

# Định nghĩa mất mát và tối ưu hóa
criterion = nn.BCELoss()
optimizer = optim.SGD(model22.parameters(), lr=0.01)

# Huấn luyện
epochs = 100
for epoch in range(epochs):
    model22.train()
    # Xóa gradient cũ
    optimizer.zero_grad()
    # Dự đoán
    y_pred = model22(X_train)
    # Tính sai lầm
    loss = criterion(y_pred, y_train)
    # Lan truyền ngược
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 20 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], loss: {loss.item():.4f}")

```

Epoch [20/100], loss: 0.6763
 Epoch [40/100], loss: 0.6654
 Epoch [60/100], loss: 0.6558
 Epoch [80/100], loss: 0.6473
 Epoch [100/100], loss: 0.6397

```

# Kiểm tra
model22.eval()
with torch.no_grad():
    y_pred = model22(X_test)
    y_pred = (y_pred >= 0.5).float()# Chuyển thành 0 hoặc 1
    accuracy = (y_pred == y_test).float().mean()
    print(f"Độ chính xác: {accuracy*100:.2f}%")

```

Độ chính xác: 63.33%

Yêu cầu 3:

Nhận xét:

1. BCEWithLogitsLoss có khác gì so với BCELoss về mất mát và độ chính xác? Tại sao?
 - BCEWithLogitsLoss: Đã được tích hợp Sigmoid bên trong, ổn định do có tính toán số mũ trực tiếp.
 - BCELoss: Cần Sigmoid trước khi tính mất mát, nếu quên Sigmoid thì dễ bị lỗi giá trị số học.
-> Khi dùng BCEWithLogitsLoss mô hình có thể học ổn định hơn còn dùng BCELoss nếu quên Sigmoid thì mất mát có thể cao hơn.
2. SGD so với Adam: Mất mát giảm nhanh hơn hay chậm hơn? Độ chính xác thay đổi ra sao?
 - Adam giảm mất mát nhanh hơn và cho độ chính xác cao hơn so với SGD nhờ cơ chế tự điều chỉnh learning rate.
 - SGD cập nhật trọng số đơn giản hơn, có thể giảm chậm và dao động nhiều quanh cực tiểu.

Phần 3: Phân tích kết quả

Yêu cầu

1. Vẽ đồ thị mất mát:

- Sửa code huấn luyện để lưu giá trị mất mát (loss) sau mỗi epoch vào một danh sách.
- Vẽ đồ thị mất mát theo epoch cho 3 trường hợp:
 - Cấu trúc ban đầu (2-4-1, Adam, BCELoss).
 - Cấu trúc 2-8-1 (Adam, BCELoss).
 - Cấu trúc 2-4-1 (SGD, BCELoss).
- Dùng `matplotlib` để vẽ 3 đường trên cùng một đồ thị, thêm chú thích (`legend`).

2. Câu hỏi:

- Quan sát đồ thị:
 - Mất mát giảm nhanh nhất ở trường hợp nào? Chậm nhất ở đâu?
 - Có trường hợp nào mất mát không giảm đều không (dao động)? Giải thích tại sao.
- Viết câu trả lời trong ô Markdown.

```

import matplotlib.pyplot as plt

# Danh sách để lưu loss
losses ANN = []
losses ANN1 = []
losses ANN22 = []

# Huấn luyện mô hình ANN
epochs = 100
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
    y_pred = model(X_train)
    loss = criterion(y_pred, y_train)
    loss.backward()
    optimizer.step()
    losses ANN.append(loss.item())

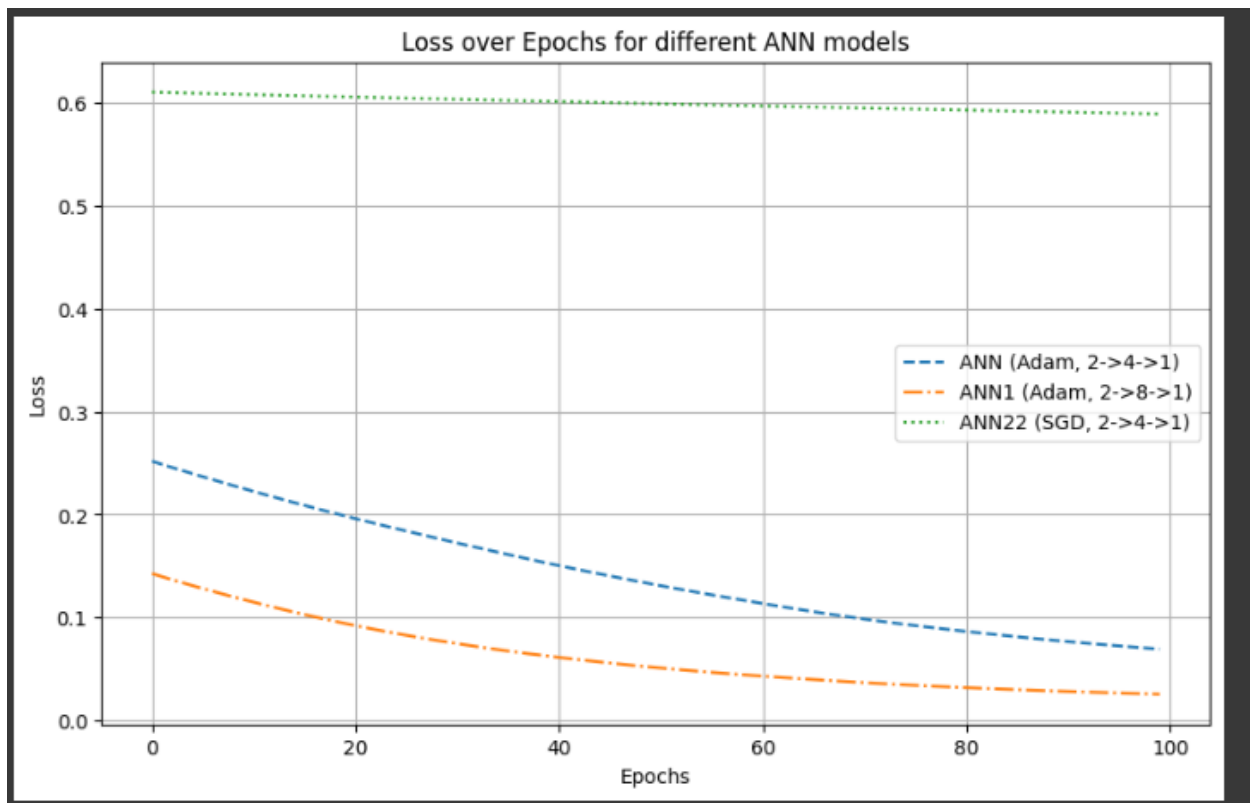
# Huấn luyện mô hình ANN1
for epoch in range(epochs):
    model1.train()
    optimizer.zero_grad()
    y_pred = model1(X_train)
    loss = criterion(y_pred, y_train)
    loss.backward()
    optimizer.step()
    losses ANN1.append(loss.item())

# Huấn luyện mô hình ANN22
for epoch in range(epochs):
    model22.train()
    optimizer.zero_grad()
    y_pred = model22(X_train)
    loss = criterion(y_pred, y_train)
    loss.backward()
    optimizer.step()
    losses ANN22.append(loss.item())

# Vẽ biểu đồ loss
plt.figure(figsize=(10, 6))
plt.plot(losses ANN, label="ANN (Adam, 2->4->1)", linestyle="--")
plt.plot(losses ANN1, label="ANN1 (Adam, 2->8->1)", linestyle="--")
plt.plot(losses ANN22, label="ANN22 (SGD, 2->4->1)", linestyle=":")

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Loss over Epochs for different ANN models")
plt.legend()
plt.grid()
plt.show()

```



Yêu cầu 2:

Nhận xét:

- Mất mát giảm nhanh nhất ở trường hợp nào? Chậm nhất ở đâu?
 - Mô hình ANN (2-4-1, Adam) và ANN1 (2-8-1, Adam) có xu hướng giảm mất mát nhanh hơn.
 - Tốc độ giảm mất mát của mô hình ANN (2-4-1, SGD) chậm hơn so với các mô hình sử dụng Adam.
- Có trường hợp nào mất mát không giảm đều không (dao động)? Giải thích tại sao.
 - Mô hình sử dụng SGD (ANN22) có dao động nhiều hơn.
 - Vì SGD cập nhật theo patch nhỏ nên khi cập nhật tăng hoặc giảm mất mát sẽ không đều gây ra dao động.