

TIÊU LUẬN MÔN HỌC PHÁT TRIỂN HỆ THỐNG THÔNG MINH (VERSION 1)

Họ và tên: Nguyễn Việt Quang

Mã sinh viên: B22DCCN650

PHẦN 1: 20 CÂU HỎI LÝ THUYẾT TỔNG HỢP

Câu 1. Deep learning là gì? So sánh deep learning với machine learning truyền thống.

1. Deep Learning là gì?

Deep Learning (Học sâu) là một nhánh của Machine Learning sử dụng các mạng nơ-ron nhân tạo nhiều lớp (Deep Neural Networks – DNN) để học và biểu diễn dữ liệu ở nhiều mức trừu tượng khác nhau.

2. Ý tưởng cốt lõi của Deep Learning

Deep Learning mô phỏng cách não người xử lý thông tin:

- Neuron nhân tạo → mô phỏng tế bào thần kinh sinh học.
- Các tầng (layers) → mô phỏng cách não chia nhỏ và trừu tượng hóa thông tin.
- Trọng số (weights) → mô tả mức độ quan trọng của thông tin.
- Hàm kích hoạt (activation function) → giúp mô hình học được quan hệ phi tuyến.

Một mạng sâu có thể gồm hàng triệu trọng số và hàng chục đến hàng nghìn tầng.

3. Cách Deep Learning học

Khi bạn đưa dữ liệu đầu vào (ảnh, âm thanh, văn bản...), mỗi tầng của mạng sẽ tự động học:

- Tầng thấp: học đặc trưng đơn giản (cạnh, đường thẳng, màu sắc...).
- Tầng giữa: học đặc trưng phức tạp hơn (hình dạng, mô hình cấu trúc...).

- Tầng cao: học đặc trưng cấp cao (khuôn mặt người, chữ cái, đối tượng hoàn chỉnh...).

Nhờ khả năng học đặc trưng tự động, Deep Learning vượt trội hơn hầu hết các thuật toán truyền thống.

4. So sánh Deep Learning và Machine Learning truyền thống

Tiêu chí	Machine Learning truyền thống	Deep Learning
Khái niệm	Học từ dữ liệu bằng các thuật toán như SVM, Random Forest, KNN, Logistic Regression...	Học bằng mạng nơ-ron nhiều lớp với khả năng tự trích xuất đặc trưng.
Xử lý đặc trưng (Feature Engineering)	Cần con người tạo ra và tối ưu đặc trưng.	Tự động học đặc trưng từ dữ liệu.
Dữ liệu yêu cầu	Hoạt động tốt với dữ liệu nhỏ – vừa.	Cần dữ liệu lớn và nhiều tài nguyên.
Hiệu năng	Tốt cho bài toán đơn giản hoặc cấu trúc rõ ràng.	Vượt trội trong bài toán phức tạp: ảnh, âm thanh, ngôn ngữ.
Thời gian huấn luyện	Nhanh hơn.	Chậm hơn do nhiều tham số.
Tài nguyên tính toán	CPU thường là đủ.	Thường cần GPU/TPU mạnh.
Khả năng diễn giải	Để giải thích hơn.	"Hộp đen", khó giải thích quyết định.

Câu 2. Giải thích vai trò của tensor trong deep learning. Tensor khác gì so với mảng NumPy?

1. Tensor là gì?

Tensor là **khối dữ liệu nhiều chiều (multi-dimensional array)**, nền tảng dữ liệu trong deep learning:

- **Scalar**: tensor bậc 0 — số đơn.
- **Vector**: tensor bậc 1 — mảng 1 chiều.
- **Matrix**: tensor bậc 2 — bảng 2 chiều.
- **Tensor bậc N**: ví dụ ảnh ($H \times W \times C$) hoặc batch ảnh ($\text{Batch} \times H \times W \times C$).

2. Vai trò của Tensor trong Deep Learning

2.1. Biểu diễn dữ liệu đầu vào

Mọi dữ liệu đều chuyển về tensor:

- Ảnh → tensor 3D/4D
- Văn bản → tensor embedding
- Audio → tensor phô tần số
- Bảng số liệu → tensor 2D

2.2. Hỗ trợ tính toán nhanh trên CPU/GPU/TPU

Tensor cho phép framework như TensorFlow, PyTorch chạy tính toán hiệu quả trên nhiều thiết bị.

2.3. Là thành phần của computational graph

- Phép toán trên tensor được ghi lại thành đồ thị tính toán.
- Mô hình có thể tự tính đạo hàm (**autograd**) và backpropagation.

2.4. Tối ưu cho ma trận và đại số tuyến tính

- Nhân ma trận, tích chập, broadcasting.
- Thiết kế để thực hiện các phép toán này **tối ưu**.

3. So sánh Tensor với NumPy Array

Tiêu chí	Tensor (TF/PyTorch)	NumPy Array
Chạy trên GPU/TPU	Có	Không
Tự động tính đạo hàm (autograd)	Có	Không
Tạo đồ thị tính toán	Có	Không
Dùng trong deep learning	Chuẩn	Không dùng trực tiếp
Hiệu năng toán ma trận lớn	Rất tối ưu (BLAS, cuDNN)	Nhanh nhưng không tối ưu bằng
Khả năng broadcast	Tốt	Tốt
Tích hợp với model training	Hoàn toàn	Không

Câu 3. Mô tả lifecycle của một mô hình deep learning trong Keras (chuẩn bị dữ liệu → xây dựng mô hình → compile → train → evaluate → inference).

1. Chuẩn bị dữ liệu (Data Preparation) – Giai đoạn quan trọng nhất

Keras nhận dữ liệu dưới dạng tensor, vì vậy toàn bộ dữ liệu thô cần được biến đổi thành dạng tensor và batch trước khi đưa vào mô hình.

1.1 Thu thập và tải dữ liệu

- Có thể đọc từ CSV, ảnh, video, text, audio.
- Keras hỗ trợ nhiều loader:
`tf.keras.datasets, tf.keras.utils.image_dataset_from_directory, tf.data.TFRecordDataset...`

1.2. Làm sạch (Cleaning)

- Loại bỏ dòng lỗi, dữ liệu thiếu.
- Xử lý class imbalance (oversampling, undersampling).

1.3. Tiền xử lý (Preprocessing)

Tùy loại dữ liệu:

Ảnh: resize, normalize, augmentation.

Text: tokenization, padding, embedding.

Tabular: normalization, standardization, categorical encoding.

Trong Keras có các lớp preprocessing tích hợp:

- `layers.Normalization()`
- `layers.TextVectorization()`
- `layers.Rescaling()`
- `layers.RandomFlip(), layers.RandomRotation()...`

1.4 Tách dữ liệu

Chia dataset thành:

- train set – để học
- validation set – để tuning
- test set – để đánh giá cuối

1.5 Tạo pipeline hiệu quả bằng tf.data

tf.data.Dataset giúp Keras chạy nhanh hơn vì:

- x2 tốc độ khi training
- prefetch song song CPU – GPU
- shuffle, batch, map, cache

Ví dụ:

```
# Nguyễn Việt Quang B22DCCN650
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)) \
    .shuffle(10000) \
    .batch(32) \
    .prefetch(tf.data.AUTOTUNE)
```

2. Xây dựng mô hình (Model Building)

Có 3 cách chính:

2.1 Sequential API

Dùng cho mô hình đơn giản, tuy nhiên tính lớp này → lớp kia.

```
# Nguyễn Việt Quang B22DCCN650
model = tf.keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

2.2 Functional API

Dùng cho kiến trúc phức tạp:

- multi-input
- multi-output
- skip connection (ResNet)
- branching

```
# Nguyễn Việt Quang B22DCCN650
inputs = layers.Input(shape=(28, 28))
x = layers.Flatten()(inputs)
x = layers.Dense(128, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)
model = tf.keras.Model(inputs, outputs)
```

2.3 Subclassing API

Tối đa linh hoạt, phù hợp custom training loop.

```
# Nguyễn Việt Quang B22DCCN650
class MyModel(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.flatten = layers.Flatten()
        self.fc1 = layers.Dense(128, activation='relu')
        self.fc2 = layers.Dense(10)

    def call(self, x):
        x = self.flatten(x)
        x = self.fc1(x)
        return self.fc2(x)
```

3. Compile mô hình (Compile) – Cấu hình quá trình học

Khi gọi model.compile(), Keras thực hiện:

3.1 Chọn optimizer

Các thuật toán tối ưu:

- SGD, Momentum
- Adam
- RMSprop

Optimizer quyết định:

- tốc độ hội tụ

- cách cập nhật weight
- độ ổn định khi training

3.2 Chọn loss function (Hàm mất mát)

Phụ thuộc bài toán:

- Classification: crossentropy
- Regression: MSE, MAE
- Binary: binary crossentropy

Loss là mục tiêu để optimizer tối thiểu hóa.

3.3 Chọn metrics

Dùng để hiển thị chất lượng mô hình:

- accuracy
- precision
- recall
- MAE...

Ví dụ Compile:

```
# Nguyễn Việt Quang B22DCCN650
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Keras lúc này mới build computational graph – đồ thị tính toán dùng cho autograd.

4. Train mô hình (Training)

Khi gọi model.fit(), nội bộ Keras thực hiện vòng lặp huấn luyện:

4.1 Forward pass

- Input → từng layer → output
- Keras tạo computational graph để ghi lại phép toán

4.2 Tính loss

So sánh output với ground truth.

4.3 Backpropagation

TensorFlow tự:

- tính đạo hàm loss theo từng weight
- cập nhật weight bằng optimizer

4.4 Batch training

Keras sẽ:

- lấy từng batch từ Dataset
- shuffle để tránh overfitting

4.5 Validation

Sau mỗi epoch, mô hình kiểm tra trên validation set để:

- phát hiện overfitting
- theo dõi meta cho early stopping

4.6 Callbacks

Keras hỗ trợ các callback:

- ModelCheckpoint – lưu mô hình tốt nhất
- EarlyStopping – dừng sớm khi overfit
- TensorBoard – plot loss/accuracy

Ví dụ:

```
# Nguyễn Việt Quang B22DCCN650
history = model.fit(
    train_ds,
    epochs=10,
    validation_data=val_ds,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=3)
    ]
)
```

5. Evaluate mô hình (Đánh giá chất lượng)

Dùng bộ test để kiểm chứng khách quan.

model.evaluate()

- chạy forward pass trên toàn test set
- không cập nhật weight
- trả về loss & metrics

Ví dụ:

```
# Nguyễn Việt Quang B22DCCN650
test_loss, test_acc = model.evaluate(test_ds)
print("Accuracy:", test_acc)
```

Keras sẽ:

- tắt dropout (training=False)
- lấy trung bình loss & metrics trên toàn dataset

6. Inference (Dự đoán) – Triển khai mô hình

Sau khi huấn luyện xong, mô hình được dùng để dự đoán.

Dự đoán bằng model.predict()

preds = model.predict(new_data)

Dự đoán thủ công

Dùng dạng eager:

```
outputs = model(x, training=False)
```

Triển khai lên môi trường thực

- TensorFlow Serving
- TF Lite (mobile)
- TensorFlow.js (web)
- ONNX (đa nền tảng)

Câu 4. Gradient descent hoạt động như thế nào? Batch GD, mini-batch GD, stochastic GD khác nhau ra sao?

1. Gradient Descent Là Gì?

Gradient Descent là thuật toán dùng để tìm cực tiểu của một hàm số, thường là hàm mất mát (loss function) trong quá trình huấn luyện mô hình. Nói cách khác, đây là công cụ giúp chúng ta đi tìm điểm tối ưu trên bề mặt của hàm mất mát, nơi mà sai số dự đoán được giảm thiểu tối đa.

1.1 Công Thức Cập Nhật:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \alpha \cdot \nabla L(\mathbf{w})$$

Trong đó:

- w là vector trọng số.
- α (learning rate) là tốc độ học, xác định bước nhảy của chúng ta.
- $\nabla L(w)$ là gradient (đạo hàm) của hàm mất mát theo trọng số.

1.2 Ví dụ:

Tưởng tượng bạn là người leo núi đang cố xuống thung lũng trong sương mù. Gradient Descent giống như la bàn chỉ hướng dốc nhất để bạn bước từng bước an toàn!

2. Tại Sao Gradient Descent Quan Trọng?

Khi huấn luyện một mạng neural, mục tiêu của chúng ta là giảm thiểu sai số dự đoán (loss).

Gradient Descent giúp ta:

- Điều chỉnh trọng số: Mỗi bước cập nhật dựa trên gradient, ta “đi” theo hướng giảm nhanh nhất của hàm mất mát.
- Học từ sai sót: Qua mỗi vòng lặp (epoch), mô hình dần dần cải thiện khả năng dự đoán, nhờ vào các điều chỉnh nhỏ và liên tục.
- Cải thiện hiệu suất: Đảm bảo rằng mô hình không “bế tắc” ở một điểm không tối ưu và luôn tiến về hướng có hiệu suất tốt hơn.

3. Cách Gradient Descent Hoạt Động: 4 Bước Đơn Giản

Bước 1: Khởi Tạo

Khởi tạo trọng số của mô hình với các giá trị ngẫu nhiên.

Bước 2: Tính Gradient

Tính gradient của hàm mất mát $L(\mathbf{w})$ theo từng trọng số. Gradient cho biết hướng và độ lớn của sự thay đổi cần thiết.

Bước 3: Cập Nhật Trọng Số

Sử dụng công thức cập nhật:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \alpha \cdot \nabla L(\mathbf{w})$$

Di chuyển “lui” theo hướng ngược gradient – vì ta muốn giảm giá trị hàm mất mát.

Lưu ý: Nếu α quá lớn, bạn sẽ “nhảy qua” điểm tối ưu; quá nhỏ thì tốn thời gian!

Bước 4: Lặp Lại Cho Đến Khi Hội Tụ

Lặp lại các bước 2 và 3 cho đến khi sự thay đổi của hàm mất mát trở nên rất nhỏ (hội tụ) hoặc đạt đến số vòng lặp tối đa.

4. Batch GD, mini-batch GD, stochastic GD khác nhau ra sao

Loại	Đặc Điểm	Ưu/Nhược
Batch GD	Dùng toàn bộ dataset để tính gradient	Chính xác nhưng chậm
Stochastic GD	Dùng 1 sample mỗi lần	Nhanh nhưng nhiễu
Mini-batch GD	Dùng một nhóm samples (ví dụ: 32-256)	Cân bằng tốt → Phổ biến nhất trong DL

Câu 5. Mục đích của hàm kích hoạt (activation function). So sánh ReLU, sigmoid, tanh

Mục đích của hàm kích hoạt

Hàm kích hoạt (activation function) được dùng trong mạng neural để giới thiệu tính phi tuyến (non-linearity) vào mô hình.

Nếu không có hàm kích hoạt:

- Một mạng nhiều lớp chỉ tương đương một phép biến đổi tuyến tính → không thể học các quan hệ phức tạp.
- Không thể mô hình hóa dữ liệu thực tế như ảnh, âm thanh, ngôn ngữ.

Vai trò chính:

1. Tạo phi tuyến tính → giúp mạng học các hàm phức tạp.
2. Giúp gradient tồn tại và lan truyền trong quá trình backpropagation.
3. Chuẩn hóa hoặc "nén" giá trị đầu ra của mỗi neuron (vd: sigmoid nén vào [0,1]).

So sánh các hàm kích hoạt phổ biến: ReLU, Sigmoid, Tanh

Sigmoid

- Công thức:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Miền giá trị: (0, 1)
- Đặc điểm:
 - Tốt cho các bài toán binary classification (tầng output).
 - Có ý nghĩa xác suất.
- Nhược điểm:
 - Vanishing gradient khi $|x|$ lớn.
 - Giá trị không đổi xung quanh 0 → ảnh hưởng tốc độ học.
- Ứng dụng:

- Output lớp cuối bài toán phân loại nhị phân.
- Không còn được khuyến khích dùng trong hidden layer.

Tanh (Hyperbolic Tangent)

Công thức:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Miền giá trị: (-1, 1)

Đặc điểm:

- Có dạng giống sigmoid nhưng đối xứng qua 0 → giúp mô hình hội tụ nhanh hơn.

Nhược điểm:

- Vẫn bị vanishing gradient khi đầu vào quá lớn.

Ứng dụng:

- Hidden layer trong các mô hình cũ.
- Một số mạng RNN truyền thống.

ReLU (Rectified Linear Unit)

Công thức:

$$\text{ReLU}(x) = \max(0, x)$$

Miền giá trị: [0, +∞)

Đặc điểm:

- Tính toán nhanh.
- Giảm nguy cơ vanishing gradient (khi $x > 0$).
- Học nhanh, là chuẩn hiện nay cho CNN, DNN.

Nhược điểm:

- Có thể bị "dead ReLU": nếu gradient = 0 với $x < 0$ → neuron chết.

Ứng dụng:

- Mạng CNN, mạng sâu nhiều lớp DNN.

- Hầu như tất cả mô hình deep learning hiện đại.

Câu 6. Vì sao normalization và standardization dữ liệu lại quan trọng đối với deep learning?

Trong deep learning, dữ liệu đầu vào thường là các vector feature có scale khác nhau và phân phối không đồng đều. Nếu đưa dữ liệu thô vào mạng neuron:

- Mạng học chậm hơn
- Gradient dao động hoặc bùng nổ
- Mô hình có thể không học đúng trọng số quan trọng

Normalization và standardization là kỹ thuật tiền xử lý dữ liệu bắt buộc để giải quyết các vấn đề này.

2. Nguyên nhân quan trọng

2.1. Giúp mạng hội tụ nhanh hơn

Gradient descent là phương pháp tối ưu phổ biến trong deep learning. Nếu các feature có scale khác nhau:

- Feature lớn → gradient lớn → bước đi quá dài → zig-zag
- Feature nhỏ → gradient nhỏ → bước đi quá ngắn → học chậm

Ví dụ minh họa:

Feature	Scale	Gradient magnitude
---------	-------	--------------------

Diện tích nhà 30–200 m² 10² → gradient lớn

Số phòng 1–6 1 → gradient nhỏ

Kết quả: Gradient descent di chuyển không đều, mất nhiều epoch hơn.
→ Chuẩn hóa dữ liệu giúp cân bằng gradient → hội tụ nhanh, ổn định.

2.2. Tránh hiện tượng gradient vanish / explode

Trong mạng sâu, đầu vào quá lớn hoặc quá nhỏ:

- Exploding gradient: giá trị gradient quá lớn → số liệu NaN, model không học được
- Vanishing gradient: giá trị gradient quá nhỏ → trọng số gần như không cập nhật

Normalization / standardization giữ dữ liệu trong phạm vi hợp lý → giảm nguy cơ này.

2.4. Giúp các feature có tầm quan trọng tương đương

Nếu scale giữa các feature quá khác nhau, mạng có thể cho rằng feature lớn là quan trọng, mặc dù thực tế không phải vậy.

Ví dụ:

- Giá đất: 1 triệu → 200 triệu
- Số phòng: 1 → 6

Nếu không chuẩn hóa, mô hình “nghĩ” giá đất quan trọng hơn số phòng, dẫn đến prediction lệch lạc.

2.5. Ôn định hóa trọng số và weight initialization

Deep learning thường sử dụng các kỹ thuật weight initialization (Xavier, He).

Các phương pháp này giả định:

- Input có mean ≈ 0
- Variance ≈ 1

Nếu dữ liệu chưa chuẩn hóa → initialization không còn hợp lý → gradient dao động → mạng học chậm hoặc thất bại.

2.6. Tương thích với BatchNorm / LayerNorm

- BatchNorm / LayerNorm chuẩn hóa trong mỗi layer để ổn định gradient.
- Tuy nhiên, chuẩn hóa đầu vào vẫn cần để đảm bảo mạng học hiệu quả ngay từ lớp đầu tiên.

3. Minh họa trực quan

Giả sử mạng MLP dự đoán giá nhà dựa trên:

Feature Giá trị gốc

Diện tích 30–200 m²

Số phòng 1–6

Giá đất 1 triệu – 200 triệu

Trường hợp không chuẩn hóa

- Loss giảm rất chậm
- Gradient dao động mạnh
- Mô hình “nghiêng” về feature giá đắt

Trường hợp chuẩn hóa

- Loss giảm nhanh, mượt
- Gradient ổn định
- Mô hình học đúng mối quan hệ giữa diện tích, số phòng và giá đắt

Câu 7. Overfitting là gì? Liệt kê các kỹ thuật chống overfitting trong sách (dropout, giảm số chiều, L1/L2 regularization, data augmentation, early stopping...).

Overfitting là gì?

Overfitting (tạm dịch: quá khớp) xảy ra khi một mô hình học quá kỹ các chi tiết và nhiễu (noise) trong dữ liệu huấn luyện, dẫn đến:

- Hiệu suất trên dữ liệu huấn luyện rất tốt, nhưng
- Hiệu suất trên dữ liệu mới (test/validation) kém, vì mô hình không tổng quát hóa được.

Nói đơn giản: mô hình "học thuộc lòng" dữ liệu thay vì học các quy luật thực sự.

Ví dụ minh họa:

- Dữ liệu huấn luyện: hình ảnh mèo và chó.
- Mô hình nhớ tất cả chi tiết trong ảnh mèo huấn luyện.
- Khi gặp mèo mới, hình khác hoàn toàn, mô hình không nhận ra mèo → overfitting.

1. Weight Regularization (L1 / L2)

- Thêm penalty vào hàm loss để hạn chế trọng số quá lớn.
- L1 → khuyến khích trọng số nhỏ bằng 0 (sparse).
- L2 → hạn chế trọng số quá lớn.
- Áp dụng trong Keras: kernel_regularizer=regularizers.l2(0.001)

2. Dropout

- Ngẫu nhiên “tắt” một số neuron trong quá trình huấn luyện.

- Giúp mô hình không phụ thuộc vào một số feature nhất định.
- Áp dụng trong Keras: Dropout(0.5)

3. Giảm độ phức tạp mô hình (Model capacity)

- Giảm số layer hoặc số units trong layer.
- Mục tiêu: đủ khả năng học quy luật, không quá lớn để “ghi nhớ” dữ liệu huấn luyện.

4. Data Augmentation

- Tăng dữ liệu huấn luyện bằng biến đổi dữ liệu gốc: xoay, lật, zoom, shift...
- Giúp mô hình học được nhiều dạng dữ liệu hơn → generalization tốt hơn.
- Keras: ImageDataGenerator(rotation_range=40, width_shift_range=0.2, ...)

5. Early Stopping

- Dừng huấn luyện khi validation loss không còn giảm.
- Tránh mô hình học noise trên tập huấn luyện.
- Keras: EarlyStopping(monitor='val_loss', patience=5)

6. Thu thập thêm dữ liệu (More training data)

- Càng nhiều dữ liệu, mô hình càng ít khả năng overfit.
- Nếu dữ liệu thực ít, kết hợp với data augmentation để nhân rộng dữ liệu.

7. Chuẩn hóa dữ liệu (Input Normalization)

- Chuẩn hóa giá trị đầu vào để ổn định gradient và tránh mô hình tập trung vào feature bất thường.
- Ví dụ: chia giá trị ảnh cho 255, chuẩn hóa về [-1, 1] hoặc z-score.

Câu 8. Ý nghĩa của hàm loss và optimizer trong mô hình. Nêu ví dụ loss phù hợp cho classification và regression.

1) Loss function là gì — vai trò

Loss function (hàm mất mát) là một hàm số đo “độ tệ” giữa dự đoán của mô hình và giá trị đúng (ground truth). Trong quá trình huấn luyện, mục tiêu là tối thiểu hóa loss trên tập huấn luyện để mô hình dự đoán càng chính xác càng tốt.

2) Optimizer là gì — vai trò

Optimizer là thuật toán dùng để cập nhật các trọng số của mô hình dựa trên gradient của loss. Nói ngắn: loss cho biết “cần sửa bao nhiêu”, optimizer quyết định cách sửa (kích thước bước, momentum, learning-rate adaptative, v.v.). Ví dụ: SGD, SGD+momentum, RMSprop, Adam đều là optimizer khác nhau với lợi ích/hạn chế riêng

3) Loss thường dùng cho classification (phân loại)

a) Binary classification (2 lớp)

- Binary cross-entropy (log loss) — dùng khi output là xác suất cho 2 lớp (sigmoid).

Thích hợp cho bài toán nhị phân hoặc multi-label (nhãn độc lập).

b) Multiclass single-label (mỗi mẫu thuộc 1 lớp)

- Categorical cross-entropy — khi nhãn one-hot và đầu ra softmax.
- Sparse categorical cross-entropy — khi nhãn là integer (0..K-1) — về bản chất giống categorical nhưng hiệu năng tốt hơn về bộ nhớ. Chú ý tham số from_logits nếu mạng trả logits thay vì xác suất.

4) Loss thường dùng cho regression (hồi quy)

- MSE (Mean Squared Error): $\frac{1}{n} \sum (y - \hat{y})^2$. Ưu: dễ tối ưu (gradient mượt). Nhược: nhạy cảm với outliers.
- MAE (Mean Absolute Error): $\frac{1}{n} \sum |y - \hat{y}|$. Ít nhạy với outliers hơn MSE nhưng gradient không liên tục tại 0.
- Huber loss: trung gian giữa MSE và MAE — robust với outliers, mượt hóa gần 0. Khi có nhiều/outliers, Huber là lựa chọn tốt

Câu 9. Mục đích của validation set. Phân biệt training-validation-test.

1. Mục đích của validation set

Validation set (tập xác thực) là một phần dữ liệu không dùng để huấn luyện nhưng dùng để đánh giá mô hình trong quá trình huấn luyện.

Mục đích chính:

1. Điều chỉnh siêu tham số (hyperparameters):

- Ví dụ: learning rate, số layer, số neuron, regularization strength, dropout rate...
- Mục tiêu: chọn cấu hình cho mô hình sao cho hiệu suất tốt trên dữ liệu chưa thấy.

2. Phát hiện overfitting sớm:

- So sánh loss/accuracy trên training set và validation set.
- Nếu training loss giảm nhưng validation loss tăng → mô hình đang overfit.

3. Early stopping:

- Validation set được dùng làm tín hiệu để dừng huấn luyện khi hiệu suất trên validation không còn cải thiện.

2. Phân biệt Training – Validation – Test

Tên tập	Mục đích	Khi nào dùng	Ví dụ
Training set	Huấn luyện mô hình, cập nhật weights	Trong quá trình training	Các hình ảnh mèo và chó dùng để backpropagation
Validation set	Điều chỉnh hyperparameters, kiểm tra overfitting	Trong quá trình training (nhưng không dùng để cập nhật weights)	Dùng để chọn learning rate, số layer
Test set	Đánh giá cuối cùng hiệu suất mô hình	Sau khi huấn luyện xong, mô hình đã có định	Kiểm tra accuracy trên ảnh mèo/chó mới chưa xuất hiện

Câu 10. Mô tả kiến trúc mạng fully connected (Dense). Ưu và nhược điểm.

1. Khái niệm mạng Fully Connected (Dense)

Mạng Fully Connected (FC) hay Dense layer là một loại mạng neural network trong đó:

- Mỗi neuron ở một layer được kết nối với tất cả các neuron ở layer trước.
- Kết nối này bao gồm trọng số (weights) và bias.
- Mỗi neuron tính toán tổng có trọng số của tất cả đầu vào và áp dụng hàm kích hoạt (activation function):

$$y = f(\sum_i w_i x_i + b)$$

Trong đó:

- x_i là đầu vào từ layer trước

- w_i là trọng số kết nối
- b là bias
- f là hàm kích hoạt (ReLU, Sigmoid, Tanh, ...)
- y là output của neuron

Ví dụ:

- Input layer: 3 neuron (x_1, x_2, x_3)
- Hidden layer Dense: 4 neuron → mỗi neuron nhận 3 đầu vào, có 3 trọng số + 1 bias → tổng $4*4=16$ parameters

2. Kiến trúc của mạng Fully Connected

Một mạng Dense cơ bản gồm:

1. Input layer

- Nhận dữ liệu đầu vào.
- Số neuron = số feature của dữ liệu.

2. Hidden layers (có thể nhiều layer)

- Mỗi neuron kết nối đầy đủ với layer trước.
- Có thể dùng nhiều layer để học các biểu diễn phức tạp.
- Mỗi neuron dùng hàm kích hoạt phi tuyến (ReLU, Sigmoid, Tanh...)

3. Output layer

- Cho kết quả cuối cùng.
- Số neuron = số class (classification) hoặc 1 (regression)
- Hàm kích hoạt: softmax (classification nhiều class), sigmoid (binary classification), linear (regression)

3. Nguyên lý hoạt động

1. Feedforward (truyền xuôi):

- Input → từng neuron layer → tổng trọng số + bias → hàm kích hoạt → output layer

2. Backpropagation (lan truyền ngược):

- Tính loss (MSE, cross-entropy...)

- Tính gradient theo weights và bias
- Cập nhật weights bằng optimizer (SGD, Adam...)

4. Ưu điểm của mạng Fully Connected

1. Đơn giản và dễ hiểu
 - Cấu trúc rõ ràng, mỗi neuron đều có trọng số và bias
 - Dễ triển khai trong các framework (Keras, PyTorch...)
2. Có khả năng học các biểu diễn phi tuyến phức tạp
 - Với ít nhất 1 hidden layer + activation phi tuyến, mạng Dense có khả năng xấp xỉ bất kỳ hàm liên tục nào (theorem Universal Approximation).
3. Ứng dụng rộng rãi
 - Classification, regression, feature extraction, embedding, autoencoder.

5. Nhược điểm của mạng Fully Connected

1. Số lượng tham số lớn
 - Với input lớn (vd: hình ảnh $224 \times 224 \times 3 = 150.528$ feature), mỗi layer Dense tạo ra hàng trăm ngàn đến hàng triệu parameters → memory và computational cost cao.
2. Dễ overfitting với dữ liệu nhỏ
 - Vì mỗi neuron kết nối tất cả input → mô hình có thể “ghi nhớ” dữ liệu huấn luyện.
3. Không tận dụng cấu trúc dữ liệu
 - Với hình ảnh, Dense không nhận biết mối quan hệ không gian (spatial correlation) giữa các pixel → CNN hiệu quả hơn.
 - Với chuỗi thời gian, Dense không nhận biết thứ tự → RNN hoặc Transformer hiệu quả hơn.
4. Không hiệu quả cho input kích thước lớn
 - Input nhiều chiều → số parameters quá lớn, training chậm và tốn tài nguyên.

Câu 11. Pooling layer (max/avg) và vai trò của nó.

1. Giới thiệu chung về Pooling Layer

Trong kiến trúc mạng nơ-ron tích chập (Convolutional Neural Network – CNN), Pooling layer là một thành phần quan trọng đứng sau các lớp Convolution. Nhiệm vụ chính của lớp này là giảm kích thước không gian của dữ liệu đầu vào ($height \times width$) trong khi vẫn giữ lại các đặc trưng

quan trọng nhất. Pooling được xem như một bước “tóm tắt thông tin”: thay vì giữ mọi giá trị trong một vùng ảnh, mạng chỉ chọn ra giá trị đại diện.

Lý do Pooling trở thành chuẩn trong CNN là bởi vì dữ liệu hình ảnh thường rất lớn, và nếu không giảm kích thước dần qua mỗi tầng, mô hình sẽ bắt buộc phải học với số lượng tham số khổng lồ—dẫn tới chi phí tính toán lớn và nguy cơ overfitting nghiêm trọng. Do đó, Pooling xuất hiện như một giải pháp cân bằng giữa giữ lại thông tin và giảm số lượng dữ liệu phải xử lý.

2. Các loại Pooling thông dụng

Có nhiều phương pháp pooling, nhưng hai phương pháp phổ biến nhất là:

2.1 Max Pooling

Max pooling lấy **giá trị lớn nhất** trong mỗi vùng cửa sổ (pooling window).

Ví dụ với cửa sổ 2×2 :

[3, 6]

[2, 1] → Max = 6

Ý nghĩa: vùng ảnh thường có nhiều pixel khác nhau, nhưng giá trị lớn nhất thường biểu thị đặc trưng mạnh nhất—ví dụ như cạnh, góc, hoặc vùng có độ tương phản mạnh. Nhờ vậy, max pooling giúp mô hình giữ lại các đặc điểm nổi bật nhất của ảnh.

2.2 Average Pooling

Average pooling lấy **giá trị trung bình** của vùng cửa sổ.

[3, 6]

[2, 1] → Average = $(3+6+2+1)/4 = 3$

Phương pháp này “mềm mại” hơn max pooling, giữ lại khuynh hướng tổng quan của giá trị pixel thay vì chỉ chọn điểm mạnh nhất. Average pooling được sử dụng nhiều trong giai đoạn trước đây của CNN hoặc trong các ứng dụng cần độ tròn trịa cao hơn, ví dụ như xử lý tín hiệu.

2.3 Global Average Pooling (GAP)

GAP tính trung bình **khắp toàn bộ feature map**, biến mỗi map thành một giá trị duy nhất.

Ví dụ một feature map $7 \times 7 \rightarrow 1$ con số.

Lớp này được dùng phổ biến trong:

- Mạng phân loại hình ảnh hiện đại (ResNet, Inception,)
- Thay thế cho Fully Connected layer để giảm tham số

- Giảm overfitting và tăng khả năng tổng quát

3. Vai trò chính của Pooling Layer trong mạng CNN

3.1 Giảm kích thước dữ liệu (Dimensionality Reduction)

Mục đích chính của pooling là giảm chiều không gian (spatial dimensions).

Ví dụ ban đầu có ảnh kích thước:

$$64 \times 64 \times 32 \rightarrow \text{Pooling } 2 \times 2 \text{ stride } 2 \rightarrow 32 \times 32 \times 32$$

Việc giảm kích thước giúp:

- Giảm số tham số phải tính
- Giảm thời gian huấn luyện
- Giảm nhu cầu bộ nhớ
- Tăng tốc độ suy luận

Nếu không có pooling, kích thước feature map sẽ giữ nguyên hoặc tăng dần khi thêm các convolution layer, khiến mô hình càng kềnh và khó huấn luyện hơn nhiều.

3.2 Ngăn overfitting

Overfitting là hiện tượng mô hình học quá sát dữ liệu huấn luyện, dẫn đến việc kém hiệu quả trên dữ liệu mới. Pooling giúp hạn chế overfitting qua việc:

- Giảm số lượng đặc trưng cần học
- Làm mịn thông tin
- Loại bỏ nhiễu cục bộ

Max pooling đặc biệt hữu ích vì nó tập trung vào vùng có tín hiệu mạnh, trong khi bỏ qua những biến động nhỏ không quan trọng.

Trong một số mô hình hiện đại, người ta kết hợp pooling + dropout để tăng khả năng chống overfitting.

3.3 Tạo tính bất biến đối với dịch chuyển (Translation Invariance)

Một trong những ưu điểm lớn nhất của CNN là khả năng nhận diện đối tượng ngay cả khi nó bị dịch chuyển nhẹ trong hình ảnh.

Pooling góp phần vào khả năng này:

- Dù đổi tượng dịch chuyển 1–2 pixel, giá trị lớn nhất trong vùng pooling thường không thay đổi đáng kể.
- Nhờ vậy, feature map sau pooling ổn định hơn và ít nhạy với vị trí chính xác của đặc trưng.

Ví dụ:

Một cạnh sáng trong ảnh có thể hơi thay đổi vị trí, nhưng max pooling vẫn chọn được đặc trưng đó ở trong vùng 2×2 . Điều này làm mô hình “ít quan tâm đến tọa độ tuyệt đối”, và tăng khả năng tổng quát.

3.4 Giữ lại đặc trưng quan trọng nhất (Feature Selection)

Max pooling đóng vai trò như một “bộ lọc chọn lọc đặc trưng”.

Trong một vùng pixel:

- Pixel quan trọng → giá trị lớn
- Pixel không quan trọng → giá trị nhỏ

Do đó:

Max pooling = giữ lại tín hiệu mạnh nhất

Từ đó mô hình dễ dàng phát hiện đặc trưng như:

- Cạnh
- Hướng gradient mạnh
- Góc tạo hình
- Texture nổi bật

Average pooling thì giữ lại thông tin tổng quan về kết cấu, giúp mô hình duy trì mức độ ổn định tổng thể của dữ liệu.

3.5 Giảm nhiễu (Noise Reduction)

Pooling, nhất là average pooling, có khả năng loại bỏ nhiễu bằng cách làm mờ các biến động nhỏ. Ví dụ: nhiễu ngẫu nhiên thường không ảnh hưởng nhiều đến giá trị trung bình.

Max pooling cũng giảm nhiễu vì:

- Nhiều hiếm khi tạo ra giá trị pixel lớn hơn tín hiệu thật
- Nếu nhiễu nhỏ → không ảnh hưởng đến giá trị được chọn

Nhờ vậy, feature map sau pooling “sạch” hơn và phù hợp hơn cho các tầng CNN tiếp theo.

4. So sánh Max Pooling và Average Pooling

Tiêu chí	Max Pooling	Average Pooling
Giữ đặc trưng	Giữ tín hiệu mạnh nhất	Giữ thông tin tổng quan
Khả năng chống nhiễu	Tốt	Trung bình
Tính sắc nét	Cao, làm nổi bật cạnh	Mềm mại hơn
Thường dùng trong	CNN hiện đại	Các mô hình cũ hoặc xử lý tín hiệu
Ảnh hưởng đến học	Giúp tập trung vào đặc trưng	Làm mượt thông tin

Trong thị giác máy tính hiện đại, **max pooling gần như là tiêu chuẩn**, vì nó giúp mạng học sâu phát hiện đặc trưng tốt hơn.

5. Những hạn chế của Pooling Layer

Dù hữu ích, pooling cũng có nhược điểm:

5.1 Mất mát thông tin

Giảm kích thước luôn đi kèm với mất thông tin.

Max pooling bỏ toàn bộ các giá trị khác ngoài phần tử lớn nhất.

5.2 Mất vị trí chính xác của đặc trưng

Pooling giúp bắt biến dịch chuyển, nhưng đồng thời làm mô hình khó nắm được vị trí chính xác.

Ví dụ trong bài toán segmentation (phân vùng ảnh), việc dùng pooling quá nhiều gây giảm độ phân giải và giảm độ chính xác vùng biên.

5.3 Không học được (Non-learnable)

Pooling hoạt động cố định và không học tham số, khác với convolution layer.

Một số mô hình hiện đại thay pooling bằng **strided convolution**, vì nó có tham số học được.

6. Ứng dụng của Pooling Layer

Pooling được sử dụng rộng rãi trong các mô hình:

- Phân loại hình ảnh (Image Classification – CNN truyền thống, VGG, AlexNet,...)
- Nhận diện đối tượng (Object Detection)

- Nhận diện chữ viết (OCR)
- Nhận diện khuôn mặt
- Xử lý tín hiệu 1D (âm thanh)
- Mạng xử lý video 3D (3D pooling)

Global Average Pooling còn được dùng trong:

- ResNet
- MobileNet
- EfficientNet
- Inception

Trong các mô hình này, GAP thay thế tầng Fully Connected cuối để giảm hơn 90% số tham số.

Câu 12. Convolution hoạt động như thế nào? Vì sao CNN hiệu quả cho bài toán ảnh?

1. Convolution hoạt động như thế nào?

1.1 Khái niệm cơ bản

Trong mạng CNN, convolution (tích chập) là phép toán dùng để trích xuất đặc trưng từ ảnh. Thay vì học trọng số theo từng pixel độc lập như mạng Fully Connected (FC), convolution học một tập các kernel (filter) nhỏ, thường kích thước:

- 3×3
- 5×5
- 7×7

Mỗi kernel “quét” toàn bộ ảnh và tạo ra feature map, cho biết kernel phát hiện đặc trưng gì tại mỗi vị trí.

1.2 Cách convolution hoạt động

Giả sử có:

- Input: Ma trận ảnh 6×6

- Kernel: 3×3
- Stride = 1
- Padding = 0

Ta thực hiện giống như “nhân ma trận trượt”:

1. Đặt kernel lên góc trái của ảnh
2. Nhân từng phần tử rồi cộng lại
3. Ghi kết quả vào feature map
4. Trượt kernel sang phải 1 pixel
5. Lặp lại cho đến hết ảnh

Ví dụ một kernel:

[1, 0, -1]

[1, 0, -1]

[1, 0, -1]

Đây là bộ lọc phát hiện cạnh dọc.

Khi quét qua ảnh, giá trị lớn sẽ xuất hiện ở nơi có biên dọc, còn vùng trống sẽ cho ra giá trị nhỏ hoặc gần 0.

→ CNN “tự học” các bộ lọc này trong quá trình training.

1.3 Nhiều filter = nhiều loại đặc trưng

Một layer có thể có:

- 32 filters → tạo 32 feature maps
- 64 filters
- 128 filters
- 512 filters (VGG16)

Mỗi filter học một loại đặc trưng khác nhau:

- Filter 1 → phát hiện cạnh ngang

- Filter 2 → phát hiện cạnh dọc
- Filter 3 → phát hiện góc
- Filter 4 → phát hiện texture
- Filter 5 → phát hiện đường cong

Nhờ có hàng trăm filters, CNN học được LỚP ĐẶC TRƯNG rất phong phú.

1.4 Convolution là phép toán có chia sẻ trọng số (Weight Sharing)

Điểm khác biệt lớn nhất so với Fully Connected:

Trong FC:

- Mỗi pixel có *trọng số riêng*.
- Ảnh $256 \times 256 \times 3 \rightarrow 196,608$ pixel → FC layer cần hàng triệu tham số.

Trong Convolution:

- Một kernel nhỏ (3×3) *dùng chung cùng một bộ trọng số* cho toàn bộ ảnh
→ Chỉ có 9 trọng số (hoặc $9 \times$ số kênh màu)

→ Số tham số giảm cực mạnh.

Ví dụ:

Ảnh $256 \times 256 \times 3$, 64 filter 3×3 :

Tham số = $3 \times 3 \times 3 \times 64 = 1,728$ tham số

So với FC cần >10 triệu tham số

Weight sharing = ít tham số = huấn luyện nhanh hơn + giảm overfitting.

1.5 Tính cục bộ (Local Connectivity)

Convolution chỉ nhìn một vùng nhỏ của ảnh (local receptive field) như:

- 3×3
- 5×5

Mạng không cố gắng đọc toàn bộ ảnh cùng lúc mà chỉ tập trung vào các mảng cục bộ → giống cách não người nhận diện:

- Nhìn cạnh

- Nhìn góc
- Nhìn đường cong
- Nhìn hoa văn
→ Rồi ghép lại thành hình lớn hơn

1.6 Xây dựng đặc trưng theo tầng (Hierarchy of Features)

Các tầng CNN học đặc trưng theo cấp độ:

Tầng đầu:

- Cạnh
- Góc
- Gradient
- Texture

Tầng giữa:

- Hình dạng đơn giản (mắt, miệng, bánh xe, cửa sổ)

Tầng sâu:

- Bộ phận phức tạp (mặt người, con mèo, chiếc ô tô)

Tầng cuối:

- Khái niệm cấp cao
→ phân loại đối tượng

Đây là ưu điểm vượt trội so với các phương pháp truyền thống như SIFT, HOG, SURF (đặc trưng thủ công không phân cấp).

CNN *tự học* đặc trưng mà không cần con người thiết kế.

2. Vì sao CNN hiệu quả cho bài toán ảnh?

2.1 Do cấu trúc ảnh có tính cục bộ (Locality)

Pixel gần nhau thường liên quan đến nhau:

- Một đường thẳng tạo thành từ nhiều pixel liên tiếp
- Một cạnh là sự thay đổi đột ngột của vùng lân cận

- Một hình tròn là họa tiết liên tục theo vùng

CNN rất hợp với quét vùng local để học đặc trưng → phù hợp tự nhiên với cấu trúc ảnh.

2.2 Weight Sharing → Giảm Số Tham Số Cực Lớn

CNN dựa trên trọng số chia sẻ (cả ảnh dùng chung 1 filter):

- Ít tham số → tránh overfitting
- Ít tham số → dễ huấn luyện hơn
- Ít tham số → nhanh hơn, dùng GPU tốt hơn
- Ít tham số → yêu cầu ít dữ liệu hơn

Các mô hình ảnh như VGG, ResNet, EfficientNet... đạt hiệu suất cao vì nhờ cơ chế weight sharing.

2.3 Tính bất biến dịch chuyển (Translation Invariance)

Nếu một đổi tượng trong ảnh dịch chuyển vị trí 5–10 pixel, CNN vẫn nhận ra được.

Nhờ:

- Convolution hoạt động trên toàn ảnh
- Pooling làm giảm nhạy với vị trí tuyệt đối

Ví dụ: con mèo hơi dịch chuyển trong ảnh → CNN vẫn detect ra mèo.

Đây là điều mà MLP (Fully Connected) *không làm được*.

2.4 Học đặc trưng tự động (Feature Learning)

Trước CNN, người ta phải tự thiết kế đặc trưng:

- SIFT
- HOG
- GLCM
- Edge detectors

Vừa tốn thời gian, vừa không tối ưu.

CNN giải quyết hoàn toàn:

- Tự học đặc trưng tốt nhất
- Học từ dữ liệu lớn
- Không cần kỹ sư thiết kế thủ công

Feature learning tự động = yếu tố làm CNN bùng nổ.

2.5 Xây dựng đặc trưng phân cấp (Hierarchical Features)

CNN học từ đặc trưng đơn giản → đặc trưng phức tạp → ý nghĩa cấp cao.

Nhờ stacking nhiều convolution layer:

- Tầng đầu: cạnh, góc
- Tầng giữa: hình dạng
- Tầng cuối: đối tượng

Điều này giống cách thị giác sinh học của con người hoạt động.

2.6 Sử dụng tốt GPU

Convolution hoạt động bằng:

- Nhân
- Cộng
- Ma trận
- Phép trượt có tính lặp lại cao

GPU rất tối ưu cho loại tính toán này → tốc độ tăng hàng trăm lần → đào tạo CNN trở nên khả thi.

2.7 Khả năng tổng quát hóa cao (Generalization)

Nhờ:

- Weight sharing
- Pooling
- Regularization tự nhiên
- Kiến trúc phân tầng

CNN generalize tốt hơn MLP rất nhiều trong các bài toán:

- Phân loại ảnh
- Nhận diện khuôn mặt
- Nhận diện chữ viết
- Phát hiện đối tượng
- Segmentation

2.8 Dễ mở rộng cho nhiều loại dữ liệu

CNN không chỉ dùng cho ảnh:

- Âm thanh 1D → Conv1D
- Video → Conv3D
- Văn bản → Text-CNN
- Giải bài toán chuỗi thời gian → Conv1D

CNN linh hoạt và mạnh mẽ cho mọi dữ liệu có cấu trúc lân cận (local).

Câu 13. Kiến trúc RNN cơ bản và hạn chế của RNN truyền thống.

1. Giới thiệu Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) là một loại mạng nơ-ron đặc biệt được thiết kế để xử lý **dữ liệu tuần tự (sequential data)**, nơi giá trị hiện tại phụ thuộc vào các giá trị trước đó.

Ví dụ: chuỗi văn bản, chuỗi thời gian, tín hiệu âm thanh, video.

Điểm khác biệt cơ bản so với mạng nơ-ron truyền thông (Fully Connected hay CNN):

- **Trạng thái ẩn (hidden state)** được duy trì qua các bước thời gian.
- Mỗi đầu vào tại thời điểm t không chỉ ảnh hưởng đến đầu ra tại thời điểm đó mà còn cập nhật trạng thái ẩn để truyền thông tin sang bước tiếp theo.

2. Kiến trúc cơ bản của RNN

2.1 Thành phần chính

Một RNN cơ bản gồm các thành phần:

1. Input Layer (x_t)

- Nhận đầu vào tại thời điểm t.
- Kích thước: ($batch_size, input_dim$)

2. Hidden Layer (h_t)

- Trạng thái ẩn chứa thông tin từ các bước trước.
- Cập nhật theo công thức:

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

Trong đó:

- W_{xh} : trọng số từ input → hidden
- W_{hh} : trọng số từ hidden trước → hidden hiện tại
- b_h : bias
- σ : hàm kích hoạt (tanh hoặc ReLU)

3. Output Layer (y_t)

- Dựa trên trạng thái ẩn, dự đoán đầu ra tại thời điểm t:

$$y_t = \phi(W_{hy}h_t + b_y)$$

Trong đó ϕ là hàm kích hoạt thích hợp, ví dụ softmax cho phân loại.

2.2 Cách RNN hoạt động (Forward Pass)

1. Bước đầu tiên:

$$\begin{aligned} h_0 &= 0 \text{ hoặc giá trị khởi tạo ngẫu nhiên} \\ y_0 &= f(x_0, h_0) \end{aligned}$$

2. Bước tiếp theo:

$$h_1 = f(x_1, h_0), y_1 = g(h_1)$$

3. Lặp lại cho toàn bộ chuỗi ($t = 0 \rightarrow T$).

Điểm đặc trưng: RNN có **mối liên hệ thời gian** (**temporal dependency**) thông qua hidden state h_t .

2.3 Biểu diễn đồ họa (từ unfolded RNN)

Nếu ta “mở” RNN theo thời gian:

$$x_0 \rightarrow [\text{RNN Cell}] \rightarrow h_0 \rightarrow y_0$$

$$x_1 \rightarrow [\text{RNN Cell}] \rightarrow h_1 \rightarrow y_1$$

$$x_2 \rightarrow [\text{RNN Cell}] \rightarrow h_2 \rightarrow y_2$$

...

- Mỗi RNN cell nhận input tại thời điểm t và trạng thái ẩn từ bước trước.
- Trạng thái ẩn giúp mạng “ghi nhớ” thông tin trước đó.

2.4 Truyền thông tin ngược (Backpropagation Through Time – BPTT)

RNN huấn luyện bằng phương pháp **BPTT**, mở rộng quá trình backprop qua từng bước thời gian:

- Gradient được lan truyền từ bước cuối về bước đầu.
- Giúp cập nhật trọng số W_{xh}, W_{hh}, W_{hy} để mạng học được phụ thuộc theo thời gian.

3. Hạn chế của RNN truyền thống

Mặc dù RNN cơ bản mạnh mẽ trong lý thuyết, **RNN truyền thống gặp nhiều vấn đề thực tế**:

3.1 Vanishing Gradient (Gradient biến mất)

- Khi chuỗi dài, gradient truyền về các bước đầu bị **giảm cực nhỏ** do nhiều lần nhân với ma trận trọng số nhỏ (<1).
- Hậu quả: mạng khó học các phụ thuộc dài hạn (long-term dependency).
- Ví dụ: trong chuỗi 100 bước, thông tin bước đầu gần như không ảnh hưởng đến bước cuối.

3.2 Exploding Gradient (Gradient bùng nổ)

- Ngược lại, nếu trọng số lớn (>1), gradient tăng theo cấp số nhân qua thời gian.
- Hậu quả: trọng số trở nên rất lớn \rightarrow mất ổn định khi huấn luyện.
- Giải pháp: gradient clipping (cắt gradient) nhưng không giải quyết vấn đề dài hạn.

3.3 Khó học các phụ thuộc dài hạn

- RNN truyền thống chỉ học tốt **ngắn hạn (short-term dependency)**.
- Ví dụ: dự đoán từ cuối chuỗi phụ thuộc nhiều vào từ đầu chuỗi → RNN truyền thông khó nhận ra.

3.4 Tốn thời gian tính toán (Sequential Computation)

- Các bước thời gian phải tính theo thứ tự: $h_1 \rightarrow h_2 \rightarrow h_3 \dots$
- Không thể tính song song như CNN.
- Hạn chế tốc độ huấn luyện, đặc biệt với chuỗi dài.

3.5 Hiệu quả thấp với chuỗi dài và dữ liệu phức tạp

- Khi chuỗi dài ($>50-100$ bước), RNN truyền thông hầu như không học được phụ thuộc quan trọng.
- Các bài toán: dịch máy, nhận diện giọng nói, dự báo chuỗi thời gian dài → RNN truyền thông kém hiệu quả.

4. Giải pháp khắc phục

Để giải quyết hạn chế của RNN truyền thông:

1. **LSTM (Long Short-Term Memory)**
 - Thêm **cell state** và **các cổng điều khiển (input, forget, output)**
 - Giữ thông tin dài hạn hiệu quả, giảm vanishing gradient
2. **GRU (Gated Recurrent Unit)**
 - Tương tự LSTM nhưng ít tham số hơn, vẫn duy trì thông tin dài hạn
3. **Bidirectional RNN**
 - Học cả phụ thuộc trước và sau trong chuỗi
4. **Attention / Transformer**
 - Thay RNN, học trực tiếp mối liên hệ dài hạn mà không cần tính toán tuần tự

5. Kết luận

- **RNN cơ bản** là mạng tuần tự, duy trì hidden state để truyền thông tin qua thời gian, thích hợp cho dữ liệu chuỗi.

- **Hạn chế chính:** vanishing/exploding gradient, khó học phụ thuộc dài hạn, tốn thời gian tính toán.
- Đây là lý do LSTM, GRU và Attention ra đời, giúp giải quyết các vấn đề của RNN truyền thống, đặc biệt trong các ứng dụng: dịch máy, nhận diện giọng nói, dự báo chuỗi thời gian, phân loại văn bản dài.

Câu 14. Vanishing gradient là gì? LSTM và GRU khắc phục vấn đề vanishing gradient như thế nào?

1. Vanishing Gradient là gì?

1.1 Khái niệm

Trong mạng nơ-ron sâu (Deep Neural Network – DNN) và mạng tuần tự (RNN), **gradient** là đạo hàm của hàm lỗi theo trọng số, dùng để cập nhật trọng số trong quá trình huấn luyện (backpropagation).

Vanishing gradient (gradient biến mất) xảy ra khi:

$$\frac{\partial L}{\partial W} \rightarrow 0 \text{ khi lan truyền ngược qua nhiều tầng hoặc bước thời gian}$$

- L = loss function
- W = trọng số

Hậu quả: trọng số gần như không được cập nhật \rightarrow mạng **không học được các phụ thuộc dài hạn** (long-term dependencies).

1.2 Nguyên nhân

1. Hàm kích hoạt bão hòa (Saturating Activation)

- Hàm tanh hoặc sigmoid: đầu ra bị giới hạn trong $[-1,1]$ hoặc $[0,1]$
- Khi giá trị input quá lớn/nhỏ, đạo hàm gần bằng 0
- Khi lan truyền gradient: đạo hàm nhân nhiều lần \rightarrow gradient giảm về 0

2. Ma trận trọng số nhỏ (<1)

- Gradient nhân nhiều lần với trọng số <1 \rightarrow suy giảm nhanh theo cấp số nhân

3. RNN truyền thông

- Gradient lan truyền qua nhiều bước thời gian \rightarrow vanishing gradient xảy ra rất nhanh

- Không thể học phụ thuộc dài hạn, ví dụ dự đoán từ bước đầu chuỗi đến bước cuối chuỗi dài 50–100 bước.

1.3 Ví dụ minh họa

Giả sử:

$$h_t = \tanh(W h_{t-1} + U x_t)$$

- $W = 0.5$
- Chuỗi dài 10 bước

Gradient tại bước đầu:

$$\frac{\partial L}{\partial h_0} = \prod_{t=1}^{10} \tanh'(h_t) \cdot W$$

- $\tanh' < 1$, $W = 0.5$
- Gradient $\approx (0.5 \times 1)^{10} = 0.000976 \rightarrow$ gần 0

→ RNN gần như **không học được thông tin từ bước đầu chuỗi**.

2. LSTM khắc phục vanishing gradient như thế nào?

2.1 Kiến trúc LSTM cơ bản

LSTM (Long Short-Term Memory) giới thiệu **Cell State (C_t)** và **các cổng điều khiển**:

- Forget gate (f_t)** – quyết định thông tin nào cần giữ
- Input gate (i_t)** – quyết định thông tin mới thêm vào
- Output gate (o_t)** – quyết định thông tin nào xuất ra hidden state
- Cell state (C_t)** – đường truyền thẳng qua thời gian, giảm nguy cơ mất gradient

Công thức chính:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

- \odot = nhân Hadamard (element-wise)
- \tilde{C}_t = candidate cell state mới

2.2 Nguyên lý khắc phục vanishing gradient

1. Cell state = đường truyền thẳng (Linear Path)

- Gradient có thể lan truyền qua nhiều bước thời gian mà gần như không bị nhân nhiều lần với trọng số nhỏ
- Giúp giữ thông tin dài hạn

2. Gates điều chỉnh thông tin

- Forget gate: loại bỏ thông tin không cần thiết
- Input gate: thêm thông tin quan trọng
- Output gate: quyết định giá trị đầu ra
- Nhờ các gate này, mạng **chọn lọc thông tin**, giảm bão hòa gradient

3. Tích chập gradient qua cổng = 1 hoặc gần 1

- Cell state cho phép gradient **truyền hầu như không bị giảm**
- Gradient biến mất gần như được khắc phục

2.3 Hình ảnh trực quan

- RNN truyền thống: gradient → nhỏ dần → mất thông tin
- LSTM: gradient → truyền dọc cell state → giữ được thông tin quan trọng

→ LSTM cho phép học các phụ thuộc dài hạn >100 bước.

3. GRU khắc phục vanishing gradient

3.1 Kiến trúc GRU (Gated Recurrent Unit)

GRU là biến thể đơn giản của LSTM:

- **Update gate (z_t)** – quyết định bao nhiêu thông tin cũ giữ lại
- **Reset gate (r_t)** – quyết định bỏ thông tin cũ khi kết hợp với input mới
- Không có cell state riêng, hidden state vừa là trạng thái vừa là output

Công thức chính:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

$$\bullet \quad \tilde{h}_t = \tanh(Wx_t + U(r_t \odot h_{t-1}))$$

3.2 Nguyên lý khắc phục vanishing gradient

- Update gate cho phép gradient **truyền trực tiếp từ $h_{\{t-1\}}$ đến h_t**

- Reset gate quyết định thông tin nào bỏ đi
- Vì gradient có đường dẫn trực tiếp → giảm vanishing gradient
- GRU ít tham số hơn LSTM, vẫn giữ khả năng học phụ thuộc dài hạn hiệu quả

4. So sánh RNN, LSTM, GRU về vanishing gradient

Mạng	Vanishing gradient	Khả năng học long-term
RNN truyền thống	Cao (dễ mất gradient)	Kém
LSTM	Thấp (công + cell state)	Tốt
GRU	Thấp (công + update gate)	Tốt

Câu 15. Word embedding là gì? Phân biệt one-hot vector và embedding vector

1. Word Embedding là gì?

1.1 Khái niệm cơ bản

Word embedding là phương pháp biểu diễn từ (word) dưới dạng **vector số thực (real-valued vector)** trong không gian liên tục.

Mục tiêu:

- Chuyển từ từ rời rạc (discrete word) sang vector liên tục (continuous vector)
- Giúp máy tính hiểu được **ngữ nghĩa (semantic meaning)** và **mối quan hệ ngữ cảnh (contextual relationship)** giữa các từ

Kích thước vector embedding thường nhỏ hơn số lượng từ trong từ điển:

- Ví dụ: từ điển 10,000 từ → embedding vector dimension = 100, 200 hoặc 300

1.2 Tại sao cần word embedding?

1. Giảm kích thước dữ liệu

- So với one-hot vector (một chiều = vocab size), embedding vector dimension nhỏ hơn nhiều → giảm bộ nhớ và chi phí tính toán

2. Giữ mối quan hệ ngữ nghĩa

- Các từ đồng nghĩa được biểu diễn bằng các vector gần nhau trong không gian embedding
- Ví dụ: “king – man + woman ≈ queen” (word2vec)

3. Dễ dàng đưa vào mô hình deep learning

- RNN, LSTM, GRU, Transformer đều dùng embedding layer trước khi đưa từ vào mạng

1.3 Cách tạo word embedding

1. Train embedding trực tiếp:

- Trong mô hình neural network, embedding layer học vector từ dữ liệu training
- Ví dụ: TensorFlow/PyTorch nn.Embedding(num_words, embedding_dim)

2. Pre-trained embeddings:

- Word2Vec, GloVe, FastText
- Trained trên corpus lớn
- Cho vector có ý nghĩa ngữ nghĩa, có thể dùng lại cho nhiều bài toán NLP

2. One-hot vector

2.1 Khái niệm

One-hot vector là biểu diễn từ dưới dạng **vector nhị phân** với:

- Kích thước = vocab size (số từ trong từ điển)
- Tất cả phần tử = 0, **ngoại trừ vị trí của từ đó = 1**

Ví dụ: vocab = [“cat”, “dog”, “fish”]

- “dog” → [0, 1, 0]

2.2 Ưu điểm

- Đơn giản, dễ cài đặt
- Không cần training, biểu diễn trực tiếp từ index trong từ điển

2.3 Nhược điểm

1. Kích thước lớn

- Vocab size lớn → vector rất dài, sparse → tốn bộ nhớ, khó tính toán

2. Không phản ánh ngữ nghĩa

- “cat” và “dog” → [1,0,0] và [0,1,0] → khoảng cách Euclidean = $\sqrt{2}$
- Máy tính không biết “cat” và “dog” gần nghĩa nhau

3. Sparse representation

- Hầu hết phần tử = 0 → không tối ưu cho deep learning

3. Embedding vector

3.1 Khái niệm

- Embedding vector là **vector liên tục, đặc (dense)**
- Dimension nhỏ hơn vocab size
- Giá trị là số thực, có thể âm hoặc dương

Ví dụ: 100-dim embedding vector cho “dog” →
[0.21, -0.34, 0.56, ..., 0.12]

3.2 Ưu điểm

1. Giảm chiều dữ liệu

- One-hot = vocab size (10,000+)
- Embedding = 50–300 → giảm đáng kể

2. Giữ ngữ nghĩa

- Các từ gần nghĩa → vector gần nhau
- Ví dụ: cosine_similarity(“cat”, “dog”) > cosine_similarity(“cat”, “car”)

3. Tối ưu cho deep learning

- Dense vector dễ đưa vào RNN, LSTM, Transformer
- Học end-to-end từ dữ liệu hoặc dùng pre-trained embedding

3.3 Cách học embedding trong mạng

- Embedding layer là ma trận $W \in (\text{vocab_size} \times \text{embedding_dim})$
- Mỗi row → vector biểu diễn một từ
- Gradient descent cập nhật W trong quá trình huấn luyện
- Vector học được mối quan hệ ngữ nghĩa qua loss function

4. So sánh One-hot vector và Embedding vector

Tiêu chí	One-hot vector	Embedding vector
----------	----------------	------------------

Kích thước	Bằng vocab size (rất lớn)	Nhỏ hơn vocab size (50–300)
------------	---------------------------	-----------------------------

Tiêu chí	One-hot vector	Embedding vector
Biểu diễn	Sparse (đa phần 0)	Dense (mọi phần tử $\neq 0$)
Ngữ nghĩa	Không phản ánh	Giữ mối quan hệ ngữ nghĩa
Khả năng học	Không học	Có thể học trong mạng NN
Hiệu quả cho NN	Thấp	Cao, tối ưu cho RNN, LSTM, Transformer
Tính toán similarity	Không hợp lý	Có thể dùng cosine similarity, dot product

PHẦN 2 – 10 CASE STUDY ỨNG DỤNG

Case study 1: Ảnh MNIST – Thiết kế 2 mô hình CNN có và không có Keras

- Mục tiêu: phân loại chữ số viết tay
- Tập data: MNIST + 100 ảnh sinh viên tự tạo ra. Thể hiện phân bố data
- Sinh viên mô tả: kiến trúc CNN + dropout + kết quả dự đoán + đánh giá 2 mô hình

Tải dataset MNIST

```
# Nguyễn Việt Quang B22DCCN650
from tensorflow.keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt

# Tải dữ liệu
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Tạo thêm 100 data và thêm vào tập train

```
# Nguyễn Việt Quang B22DCCN650
# tạo thêm 100 data
from PIL import Image, ImageDraw, ImageFont
import numpy as np
import random
new_images = []
new_labels = []
for _ in range(100):
    img = Image.new('L', (28, 28), color=0)
    draw = ImageDraw.Draw(img)
    digit = random.randint(0, 9) # Lưu dưới dạng số
    draw.text((5, 2), str(digit), fill=255)
    new_images.append(np.array(img))
    new_labels.append(digit)
new_images = np.array(new_images)
new_labels = np.array(new_labels)
plt.figure(figsize=(8,8))
for i in range(100):
    plt.subplot(10, 10, i+1)
    plt.imshow(new_images[i], cmap='gray')
    plt.axis("off")
plt.show()
x_train = np.concatenate([x_train, new_images], axis=0)
y_train = np.concatenate([y_train, new_labels], axis=0)
```

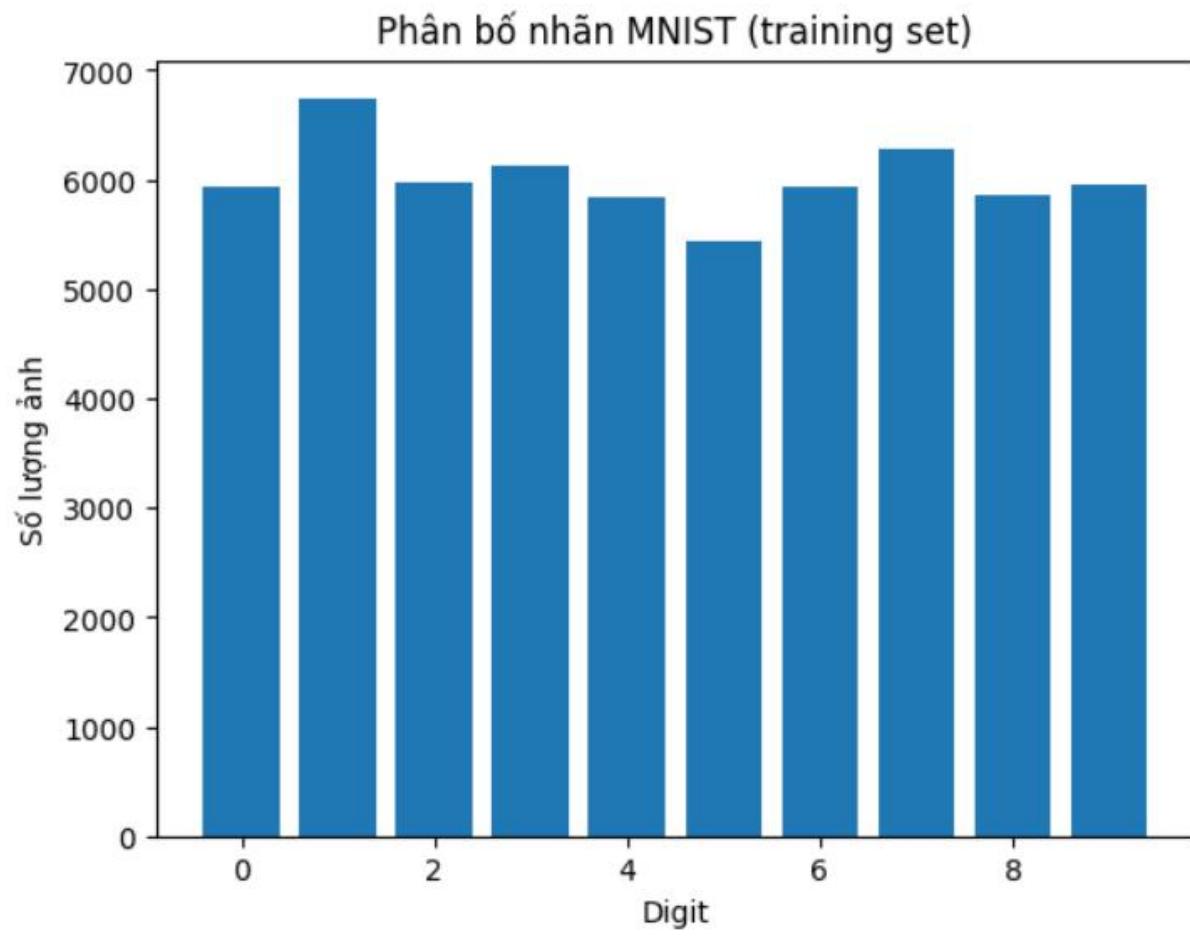
7	8	0	5	2	8	6	2	9	1
7	0	7	9	8	6	5	9	5	5
7	0	5	2	4	8	7	2	6	9
3	7	2	9	2	7	2	9	0	8
9	0	1	9	8	4	9	5	6	8
9	7	4	2	4	0	9	9	7	6
5	5	2	8	2	8	8	2	8	5
9	7	0	1	7	0	8	6	2	1
5	7	0	4	8	4	4	0	8	5
8	6	6	9	0	0	5	7	6	2

Trực quan hóa phân bố nhãn (label distribution)

```
# Nguyễn Việt Quang B22DCCN650
import collections

counter = collections.Counter(y_train)

plt.bar(counter.keys(), counter.values())
plt.title("Phân bố nhãn MNIST (training set)")
plt.xlabel("Digit")
plt.ylabel("Số lượng ảnh")
plt.show()
```



Chuẩn hóa dữ liệu về [0,1]

```
# Nguyễn Việt Quang
# Chuẩn hóa dữ liệu về [0,1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
```

Thêm kênh màu (channel) cho dữ liệu

```
# Nguyễn Việt Quang B22DCCN650
# Thêm kênh màu (channel) cho dữ liệu
x_train = np.expand_dims(x_train, -1) # Kích thước mới: (num_samples, 28, 28, 1)
x_test = np.expand_dims(x_test, -1) # Kích thước mới: (num_samples, 28, 28, 1)
```

One-hot labels

```
# Nguyễn Việt Quang B22DCCN650
# One-hot labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

Xây dựng mô hình CNN sử dụng keras

```
# Xây dựng mô hình CNN
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 225,034 (879.04 KB)

Trainable params: 225,034 (879.04 KB)

Non-trainable params: 0 (0.00 B)

Mô tả kiến trúc:

STT	Tầng	Cấu hình	Vai trò
1	Conv2D	32 filter, kernel 3x3, activation ReLU, input_shape=(28,28,1)	Trích xuất đặc trưng từ ảnh đầu vào, ReLU giúp phi tuyến
2	MaxPooling2D	pool size 2x2	Giảm kích thước không gian, giữ đặc trưng quan trọng
3	Conv2D	64 filter, kernel 3x3, activation ReLU	Trích xuất đặc trưng phức tạp hơn
4	MaxPooling2D	pool size 2x2	Giảm kích thước, tăng tính trừu tượng
5	Flatten	—	Chuyển ma trận đặc trưng 2D thành vector 1D để đưa vào Dense
6	Dense	128 neurons, activation ReLU	Học mối quan hệ phi tuyến giữa

STT	Tầng	Cấu hình	Vai trò
			các đặc trưng
7	Dropout	rate=0.5	Ngăn overfitting bằng cách tạm thời “tắt” 50% neuron trong huấn luyện
8	Dense	10 neurons, activation softmax	Lớp đầu ra, dự đoán xác suất 10 lớp (digits 0-9)

3. Compile mô hình

```
# Nguyễn Việt Quang B22DCCN650
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

4. Huấn luyện mô hình

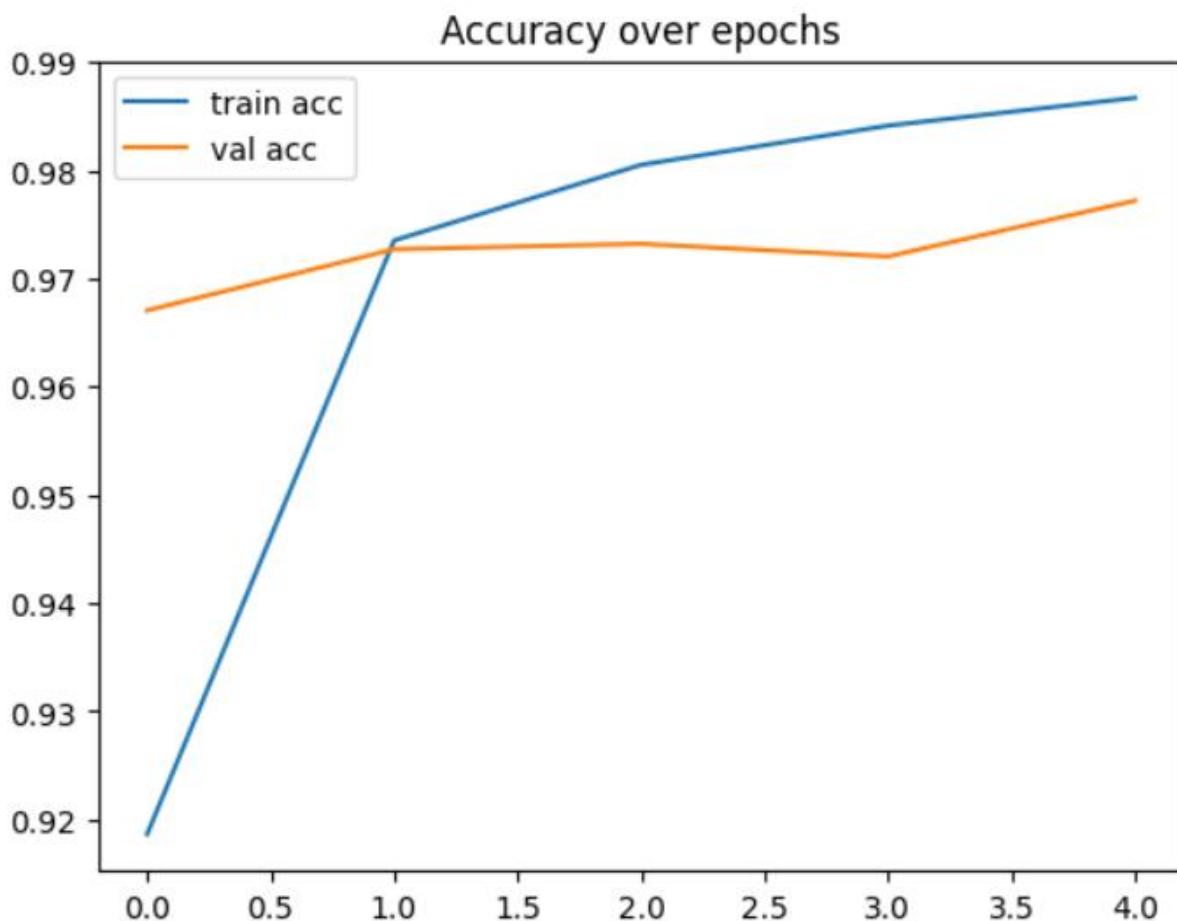
```
# Nguyễn Việt Quang B22DCCN650
history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=64,
    validation_split=0.1
)
```

Đánh giá mô hình

```
# Nguyễn Việt Quang B22DCCN650
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

Vẽ lịch sử train/validation accuracy

```
# Nguyễn Việt Quang B22DCCN650
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='train acc')
plt.plot(history.history['val_accuracy'], label='val acc')
plt.legend()
plt.title("Accuracy over epochs")
plt.show()
```



Xây dựng mô hình CNN không sử dụng keras

Phần load dataset và và chuẩn hóa dữ liệu vẫn giống như ở trên

Tách validation set (10%)

```
# =====
# 2. Tách validation set (10%)
# Nguyễn Việt Quang B22DCCN650
# =====
val_size = int(0.1 * x_train.shape[0])

x_val = x_train[:val_size]
y_val = y_train[:val_size]

x_train = x_train[val_size:]
y_train = y_train[val_size:]

num_samples = x_train.shape[0]
```

Build CNN bằng TensorFlow thuần (Cấu trúc mạng CNN vẫn giống mạng CNN khi build bằng keras)

```
# =====
# 3. Xây CNN bằng TensorFlow thuần
# Nguyễn Việt Quang B22DCCN650
# =====
class CNN(tf.Module):
    def __init__(self):
        super().__init__()

        # Conv 1
        self.conv1 = tf.Variable(tf.random.normal([3, 3, 1, 32], stddev=0.1))

        # Conv 2
        self.conv2 = tf.Variable(tf.random.normal([3, 3, 32, 64], stddev=0.1))

        # Dense 1
        self.w1 = tf.Variable(tf.random.normal([64*5*5, 128], stddev=0.1))
        self.b1 = tf.Variable(tf.zeros([128]))

        # Dense 2
        self.w2 = tf.Variable(tf.random.normal([128, 10], stddev=0.1))
        self.b2 = tf.Variable(tf.zeros([10]))
```

```

# Nguyễn Việt Quang B22DCCN650
def __call__(self, x, training=True):
    # Conv1
    x = tf.nn.conv2d(x, self.conv1, strides=1, padding="VALID")
    x = tf.nn.relu(x)
    x = tf.nn.max_pool2d(x, ksize=2, strides=2, padding="VALID")

    # Conv2
    x = tf.nn.conv2d(x, self.conv2, strides=1, padding="VALID")
    x = tf.nn.relu(x)
    x = tf.nn.max_pool2d(x, ksize=2, strides=2, padding="VALID")

    # Flatten
    x = tf.reshape(x, [-1, 64*5*5])

    # Dense1
    x = tf.nn.relu(tf.matmul(x, self.w1) + self.b1)

    if training:
        x = tf.nn.dropout(x, rate=0.5)

    return tf.matmul(x, self.w2) + self.b2

```

```

model = CNN()
optimizer = tf.optimizers.Adam(0.001)
loss_fn = tf.nn.softmax_cross_entropy_with_logits

```

Training loop

Đánh giá trên tập test

```
# =====
# 4. Training Loop
# Nguyễn Việt Quang B22DCCN650
# =====
batch_size = 64
epochs = 5
steps = num_samples // batch_size

train_acc_history = []
val_acc_history = []
```

```

# Nguyễn Việt Quang B22DCCN650
for epoch in range(epochs):
    # Shuffle train data
    idx = np.random.permutation(num_samples)
    x_train = x_train[idx]
    y_train = y_train[idx]
    correct = 0
    total = 0
    # ---- Training ----
    for step in range(steps):
        x_batch = x_train[step*batch_size:(step+1)*batch_size]
        y_batch = y_train[step*batch_size:(step+1)*batch_size]

        with tf.GradientTape() as tape:
            logits = model(x_batch, training=True)
            loss = tf.reduce_mean(loss_fn(y_batch, logits))

        grads = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))

        preds = tf.argmax(logits, axis=1)
        labels = tf.argmax(y_batch, axis=1)
        correct += tf.reduce_sum(tf.cast(preds == labels, tf.int32))
        total += x_batch.shape[0]

```

```

train_acc = correct / total
train_acc_history.append(train_acc)

# ---- Validation ----
logits_val = model(x_val, training=False)
preds_val = tf.argmax(logits_val, axis=1)
labels_val = tf.argmax(y_val, axis=1)
val_acc = tf.reduce_mean(tf.cast(preds_val == labels_val, tf.float32))
val_acc_history.append(val_acc)

print(f"Epoch {epoch+1}: Train Acc = {train_acc:.4f} | Val Acc = {val_acc:.4f}")

```

Dánh giá trên tập Test

```

# =====
# 5. Evaluate Test Accuracy
# Nguyễn Việt Quang B22DCCN650
# =====

logits_test = model(x_test, training=False)
preds_test = tf.argmax(logits_test, axis=1)
labels_test = tf.argmax(y_test, axis=1)
test_acc = tf.reduce_mean(tf.cast(preds_test == labels_test, tf.float32))

print("\n=====")
print("Test accuracy:", float(test_acc))
print("=====\\n")

```

Vẽ biểu đồ Train vs Val Accuracy

```

# =====
# 6. Vẽ biểu đồ Train vs Val Accuracy
# Nguyễn Việt Quang B22DCCN650
# =====

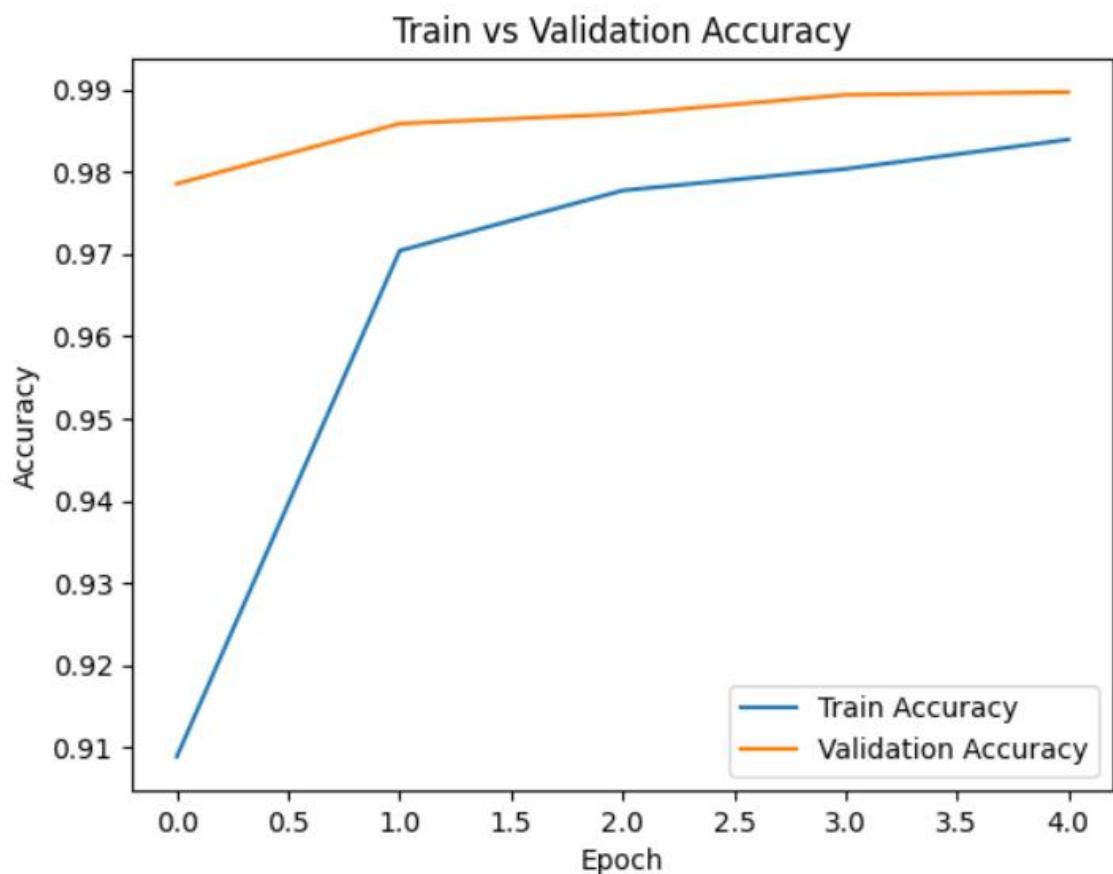
plt.plot(train_acc_history, label="Train Accuracy")
plt.plot(val_acc_history, label="Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Train vs Validation Accuracy")
plt.legend()
plt.show()

```

Output

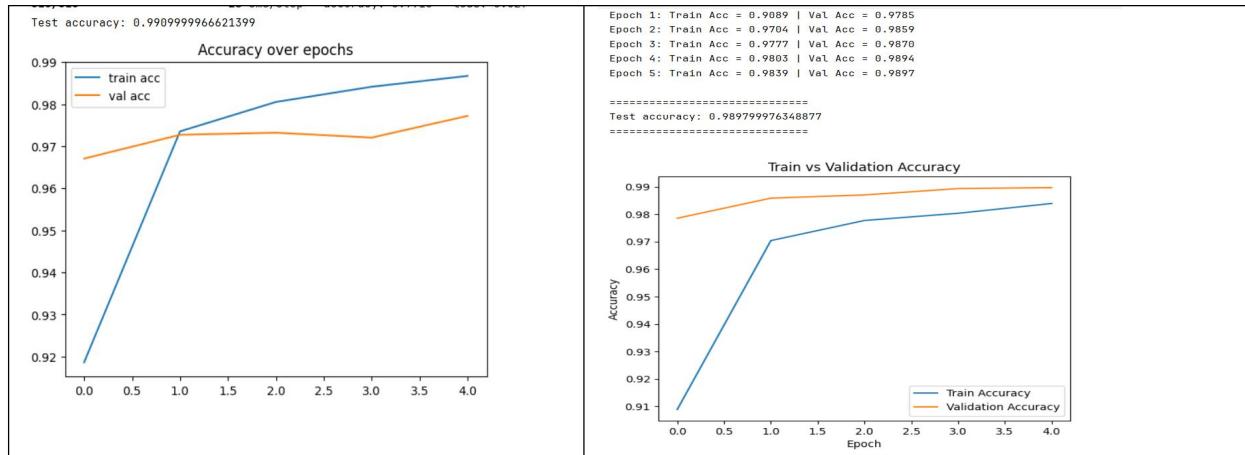
```
Epoch 1: Train Acc = 0.9089 | Val Acc = 0.9785
Epoch 2: Train Acc = 0.9704 | Val Acc = 0.9859
Epoch 3: Train Acc = 0.9777 | Val Acc = 0.9870
Epoch 4: Train Acc = 0.9803 | Val Acc = 0.9894
Epoch 5: Train Acc = 0.9839 | Val Acc = 0.9897

=====
Test accuracy: 0.989799976348877
=====
```



Đánh giá 2 mô hình

Mô hình sử dụng keras	Mô hình không sử dụng keras
-----------------------	-----------------------------



Tốc độ huấn luyện của mô hình sử dụng keras nhanh hơn mô hình không sử dụng keras

Code bằng keras ngắn gọn hơn rất nhiều so với không sử dụng keras

Mô hình	Độ chính xác MNIST (ước lượng, sau 5 epoch)
Keras Sequential CNN	0.9910 trên training, 0.990 trên test
TensorFlow thuần CNN (same architecture)	0.9839 trên training, 0.989 trên test

Sự khác biệt thường **rất nhỏ (<0.1%)** và do **random initialization, shuffling, dropout**

Case study 2: Phân tích âm xúc văn bản (sentiment analysis)

1. Tải dữ liệu

Ở bước đầu tiên, ta tải file 'IMDB-Dataset.csv' chứa hai cột:

- review: nội dung đánh giá phim dạng text
- sentiment: nhãn positive/negative

Sau đó ta chuyển sentiment thành dạng số để mô hình học dễ hơn.

```
# 1. Load dataset
# Nguyễn Việt Quang B22DCCN650
df = pd.read_csv("DATA/IMDB-Dataset.csv")

reviews = df["review"].astype(str).values
labels = df["sentiment"].map({"positive": 1, "negative": 0}).values
```

2. Chia dữ liệu Train/Test

Ta chia dữ liệu thành 80% train và 20% test để đánh giá mô hình khách quan.

```

7 # 2. Train/test split
8 # Nguyễn Việt Quang B22DCCN650
9 X_train, X_test, y_train, y_test = train_test_split(
10     reviews, labels, test_size=0.2, random_state=42
11 )

```

3. Tokenization & Chuyển đổi văn bản thành số

Tokenizer sẽ học danh sách từ vựng từ tập train, sau đó mỗi từ được chuyển thành số.

Ta giới hạn 20.000 từ phổ biến nhất để tránh nhiễu và giảm độ phức tạp.

```

3 # 3. Tokenizer + Text to sequences
4 # Nguyễn Việt Quang B22DCCN650
5 vocab_size = 20000    # số lượng từ tối đa
6 tokenizer = Tokenizer(num_words=vocab_size, oov_token "<OOV>")
7 tokenizer.fit_on_texts(X_train)
8
9 X_train_seq = tokenizer.texts_to_sequences(X_train)
10 X_test_seq = tokenizer.texts_to_sequences(X_test)

```

4. Padding

LSTM/GRU yêu cầu các chuỗi phải có độ dài bằng nhau. Ta chọn max_len = 200 từ.

Padding sẽ thêm số 0 vào những câu ngắn

```

# 4. Padding
# Nguyễn Việt Quang B22DCCN650
max_len = 200
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding="post")
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len, padding="post")

```

hon.

5. Xây dựng mô hình Embedding + LSTM/GRU

Mô hình gồm các lớp chính:

- Embedding: biến số nguyên thành vector đặc trưng
- LSTM/GRU: học ngữ cảnh trong câu
- Dense: phân loại cảm xúc

Có thể chọn LSTM hoặc GRU.

```

8 # 5. Build model - chọn LSTM hoặc GRU
9 # Nguyễn Việt Quang B22DCCN650
0 model = Sequential([
1     Embedding(input_dim=vocab_size, output_dim=128, input_length=max_len),
2
3     # Chọn 1 trong 2:
4     LSTM(128, return_sequences=False),
5     # GRU(128, return_sequences=False),
6
7     Dropout(0.3),
8     Dense(64, activation="relu"),
9     Dropout(0.3),
0     Dense(1, activation="sigmoid")
1 ])
2
3 model.compile(
4     optimizer=Adam(1e-3),
5     loss="binary_crossentropy",
6     metrics=["accuracy"]
7 )
8
9 model.summary()

```

6.

Huấn luyện mô hình

Mô hình được train trong 5 epoch cùng validation_split = 0.2.

```

# 6. Train model
# Nguyễn Việt Quang B22DCCN650
history = model.fit(
    X_train_pad, y_train,
    validation_split=0.2,
    epochs=5,
    batch_size=64
)

```

7. Đánh giá mô hình

Sau khi huấn luyện, ta kiểm tra accuracy trên tập

```
'0 # 7. Evaluate
'1 # Nguyễn Việt Quang B22DCCN650
'2 loss, acc = model.evaluate(X_test_pad, y_test)
'3 print("Test Accuracy:", acc)
test.'4
```

8. Vẽ biểu đồ Accuracy & Loss

Biểu đồ giúp quan sát mô hình có bị overfit hay không.

```
' # Nguyễn Việt Quang B22DCCN650
| # Lấy lịch sử huấn luyện
| acc = history.history['accuracy']
| val_acc = history.history['val_accuracy']
| loss = history.history['loss']
| val_loss = history.history['val_loss']
| epochs = range(1, len(acc) + 1)

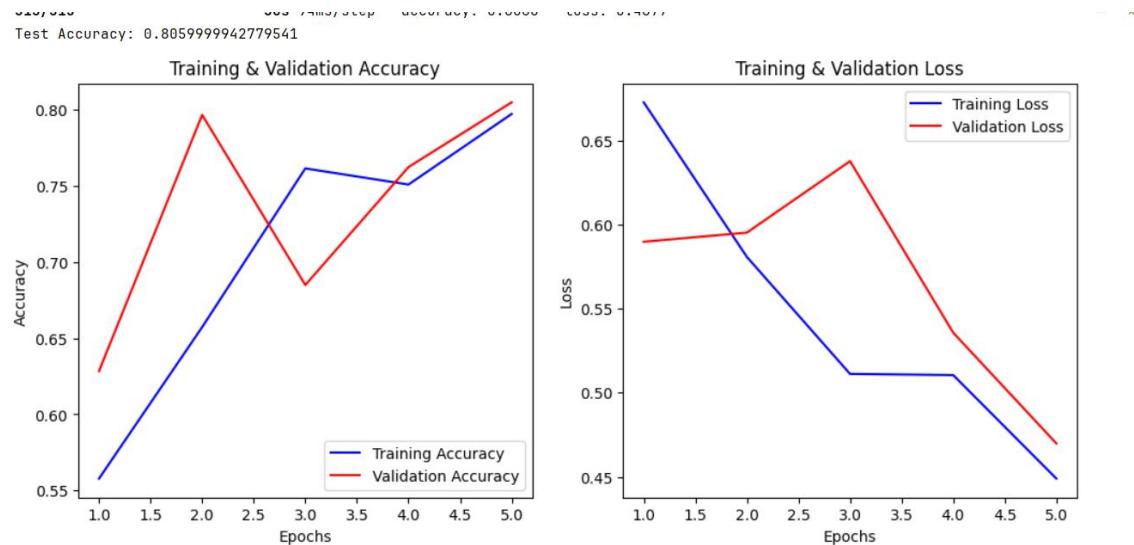
| # Vẽ accuracy
| plt.figure(figsize=(12,5))

| plt.subplot(1,2,1)
| plt.plot(epochs, acc, 'b', label='Training Accuracy')
| plt.plot(epochs, val_acc, 'r', label='Validation Accuracy')
| plt.title('Training & Validation Accuracy')
| plt.xlabel('Epochs')
| plt.ylabel('Accuracy')
| plt.legend()

| # Vẽ loss
| plt.subplot(1,2,2)
| plt.plot(epochs, loss, 'b', label='Training Loss')
| plt.plot(epochs, val_loss, 'r', label='Validation Loss')
| plt.title('Training & Validation Loss')
| plt.xlabel('Epochs')
| plt.ylabel('Loss')
| plt.legend()

| plt.show()
```

9 Kết quả output



Phân tích Độ chính xác (Accuracy)

- **Độ chính xác huấn luyện (Training Accuracy - đường màu xanh):** Bắt đầu ở mức khoảng 55.5% và có xu hướng tăng đều đặn qua các epoch, kết thúc ở mức cao nhất là khoảng 80% (epoch 5).
- **Độ chính xác kiểm tra/xác thực (Validation Accuracy - đường màu đỏ):**
 - Nó cho thấy sự cải thiện nhanh chóng từ epoch 1 lên đến khoảng 79% ở epoch 2.
 - Tuy nhiên, từ epoch 2 đến epoch 3, độ chính xác kiểm tra **giảm mạnh** (trong khi độ chính xác huấn luyện vẫn tăng).
 - Sau đó, nó tăng trở lại và kết thúc ở mức khoảng 80.5% ở epoch 5 (con số này được xác nhận bởi dòng chữ **Test Accuracy: 0.805...** phía trên biểu đồ).
- **Nhận xét:**
 - Độ chính xác cuối cùng là tốt (**trên 80%**).
 - Sự biến động lớn của đường Validation Accuracy, đặc biệt là sự sụt giảm ở epoch 3, cho thấy mô hình có thể đã **khớp quá mức (overfit) tạm thời** với dữ liệu huấn luyện ở epoch 2 và phải "học lại" ở epoch 3.

Phân tích Hàm mất mát (Loss)

- **Hàm mất mát huấn luyện (Training Loss - đường màu xanh):** Có xu hướng **giảm đều** qua 5 epoch, bắt đầu từ khoảng 0.67 và kết thúc ở mức thấp nhất là khoảng 0.45. Đây là dấu hiệu tốt, cho thấy mô hình đang học hiệu quả trên dữ liệu huấn luyện.

- **Hàm mất mát kiểm tra/xác thực (Validation Loss - đường màu đỏ):**
 - Bắt đầu ở mức khoảng 0.59.
 - Nó **tăng nhẹ** từ epoch 1 đến epoch 2 (mặc dù Validation Accuracy tăng), và sau đó **tăng mạnh** lên mức cao nhất là khoảng **0.64** tại epoch 3.
 - Sau epoch 3, nó **giảm mạnh** và kết thúc ở mức khoảng **0.47** tại epoch 5.
- **Nhận xét:**
 - Khoảng thời gian từ **epoch 2 đến epoch 3** là giai đoạn đáng chú ý nhất:
 - Training Loss tiếp tục giảm.
 - Validation Loss **tăng mạnh** (cùng với việc Validation Accuracy giảm mạnh).
 - Đây là dấu hiệu rõ ràng của **hiện tượng khớp quá mức (overfitting)** xảy ra tại/sau epoch 2. Mô hình trở nên quá tốt với dữ liệu huấn luyện (loss giảm) nhưng lại kém hơn với dữ liệu mới (loss tăng).

Đề xuất cải tiến cho lần huấn luyện sau:

- **Sử dụng Kỹ thuật Điều chỉnh (Regularization):** Để giảm hiện tượng overfitting, hãy thử thêm **Dropout layers** hoặc áp dụng **L2 Regularization (Weight Decay)**.
- **Giảm Tốc độ Học (Learning Rate):** Tốc độ học có thể hơi cao, gây ra sự biến động lớn (dao động) trong Validation Accuracy và Loss. Thử sử dụng Learning Rate nhỏ hơn hoặc áp dụng **Learning Rate Scheduler** để giảm tốc độ học sau một số epoch.
- **Huấn luyện thêm:** Vì cả Training Loss và Validation Loss đều đang tiếp tục giảm mạnh ở epoch 5, mô hình có thể vẫn chưa hội tụ hoàn toàn. Thử huấn luyện thêm 5-10 epoch nữa để xem kết quả có tiếp tục cải thiện không (tuy nhiên, hãy quan sát sát sao Validation Loss).

10 Pipeline xử lý text trong bài toán phân loại cảm xúc IMDB

Pipeline xử lý text gồm các bước từ dữ liệu văn bản thô cho đến đầu vào đã được số hóa sẵn sàng cho mô hình học sâu.

10.1. Chuẩn hóa dữ liệu đầu vào

- Đọc cột review dạng text.
- Chuyển toàn bộ về kiểu chuỗi để tránh lỗi dữ liệu.

(Bước này đảm bảo dữ liệu đồng nhất trước khi xử lý.)

10.2. Tokenization – tách văn bản thành các token

Dùng Tokenizer của Keras để:

- Học từ vựng từ tập train.
- Tách câu thành từng từ.
- Loại bỏ các ký tự không cần thiết.
- Mỗi từ xuất hiện trong văn bản được đưa vào một **word index**.

Ví dụ:

"This movie is great"

→ ["This", "movie", "is", "great"]

10.3. Chuyển token thành chỉ số (Text to Sequences)

Mỗi từ được ánh xạ sang một số nguyên duy nhất dựa trên từ điển mà Tokenizer đã học.

Ví dụ:

["This", "movie", "is", "great"]

→ [12, 85, 9, 432]

Tại sao cần bước này?

Vì mô hình machine learning chỉ xử lý được dữ liệu dạng số, không hiểu ký tự.

10.4. Giới hạn từ vựng (Vocabulary Limiting)

Ta chọn:

vocab_size = 20000

- Chỉ giữ lại 20.000 từ phổ biến nhất.
- Từ hiếm, lỗi chính tả → gán thành <OOV> (out-of-vocabulary).

Lợi ích:

- Tránh nhiễu
- Giảm kích thước embedding

- Huấn luyện nhanh hơn

10.5. Padding – đưa mọi chuỗi về cùng độ dài

Các câu review dài ngắn khác nhau. LSTM/GRU yêu cầu input có độ dài cố định.

Ta chọn:

`max_len = 200`

Thực hiện:

- Chuỗi ngắn → thêm 0 vào cuối
- Chuỗi dài → cắt bớt từ cuối

Ví dụ:

[12, 85, 9]

→ [12, 85, 9, 0, 0, 0, ... đến 200]

Mục đích:

- Tạo ma trận đầu vào có kích thước đều nhau
- Giúp mô hình xử lý batch hiệu quả

10.6. Encoding cuối cùng (Chuẩn bị cho Embedding)

Sau khi đi qua 3 bước chính:

1. Tokenization
2. Sequences
3. Padding

Ta nhận được một tensor có dạng:

(num_samples, max_len)

Ví dụ:

[[12, 85, 9, 432, 0, 0, ...],
[51, 883, 120, 12, 0, 0, ...],

Đây chính là dữ liệu sạch – chuẩn – sẵn sàng để đưa vào Embedding và mô hình học sâu.

Case study 3: Dự báo bệnh tiểu đường

1. Chuẩn bị dữ liệu (Data Preparation)

1.1 Thu thập và tải dữ liệu

Sử dụng tập dữ liệu **Pima Indians Diabetes Database**

Code tải dữ liệu (CSV)

```
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Nguyễn Việt Quang B22DCCN650
# Load CSV
df = pd.read_csv("diabetes.csv")
print(df.head())
```

Kết quả

```
Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \
0          6      148           72        35       0  33.6
1          1       85           66        29       0  26.6
2          8      183           64        0       0  23.3
3          1       89           66        23      94  28.1
4          0      137           40        35     168  43.1

DiabetesPedigreeFunction Age Outcome
0            0.627    50      1
1            0.351    31      0 |
2            0.672    32      1
3            0.167    21      0
```

1.2 Làm sạch dữ liệu (Cleaning)

1.2.1 Kiểm tra dữ liệu thiếu

Thống kê giá trị NaN

```
# Thống kê số giá trị NaN
# Nguyen Việt Quang B22DCCN650
nan_counts = df.isna().sum()
print("Số giá trị NaN trong từng cột:")
print(nan_counts)
```

Kết quả: Không có giá trị nào bị NaN

```
Số giá trị NaN trong từng cột:
Pregnancies          0
Glucose              0
BloodPressure        0
SkinThickness        0
Insulin              0
BMI                  0
DiabetesPedigreeFunction 0
Age                  0
Outcome              0
dtype: int64
```

Thống kê số giá trị bằng 0

```
# Thống kê số giá trị bằng 0
# Nguyen Việt Quang B22DCCN650
zero_counts = (df == 0).sum()
print("\nSố giá trị bằng 0 trong từng cột:")
print(zero_counts)
```

Kết quả:

Số giá trị bằng 0 trong từng cột:

Pregnancies	111
Glucose	5
BloodPressure	35
SkinThickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0
Outcome	500

dtype: int64

Dataset này có giá trị 0 bất thường:

- Glucose = 0
- BloodPressure = 0
- SkinThickness = 0
- Insulin = 0
- BMI = 0

→ Đây là lỗi đo, cần thay bằng **median** hoặc **mean**.

Code xử lý

```
# Thay thế giá trị 0 bằng giá trị trung vị của cột tương ứng
# Nguyen Việt Quang B22DCCN650
cols_with_zero = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]
for c in cols_with_zero:
    df[c] = df[c].replace(0, df[c].median())
```

Thông kê lại số giá trị bằng 0 sau khi thay thế

```
# Thống kê lại số giá trị bằng 0 sau khi thay thế
# Nguyen Việt Quang B22DCCN650
zero_counts_after = (df == 0).sum()
print("\nSố giá trị bằng 0 trong từng cột sau khi thay thế:")
print(zero_counts_after)
```

Kết quả các giá trị 0 bất thường đã được xử lý:

Số giá trị bằng 0 trong từng cột sau khi thay thế:	
Pregnancies	111
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	500

1.2.2 Xử lý class imbalance

Thống kê số lượng và tỷ lệ phần trăm từng class trong cột 'Outcome'

```
# Thống kê số lượng và tỷ lệ phần trăm từng class trong cột 'Outcome'
# Nguyen Việt Quang B22DCCN650
class_counts = df['Outcome'].value_counts()
class_percent = df['Outcome'].value_counts(normalize=True) * 100 # chuyển sang %
print("Số lượng từng class:")
print(class_counts)
print("\nTỷ lệ phần trăm từng class:")
print(class_percent)
```

Số lượng từng class:

Outcome

0 500

1 268

Name: count, dtype: int64

Tỷ lệ phần trăm từng class:

Outcome

0 65.104167

1 34.895833

Name: proportion, dtype: float64

Mức chênh lệch không quá nghiêm trọng -> không cần xử lý mất cân bằng

1.3. Tiền xử lý (Preprocessing)

1.3.1 Tách feature – label

```
# Tách features và target  
# Nguyen Việt Quang B22DCCN650  
X = df.drop("Outcome", axis=1)  
y = df["Outcome"]
```

1.3.2 Chuẩn hoá dữ liệu (Standardization)

```
# Chuẩn hóa dữ liệu  
# Nguyen Việt Quang B22DCCN650  
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)
```

1.4 Tách dữ liệu Train / Validation / Test

Tỉ lệ:

- train: 70%
- val: 15%

- test: 15%

```
# Chia dữ liệu thành tập huấn luyện, tập validation và tập test
# Nguyen Việt Quang B22DCCN650
X_train, X_temp, y_train, y_temp = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=42
)
```

1.5 Tạo Pipeline hiệu quả bằng tf.data

Chuyển X, y thành tensor dataset

```
# Chuyển đổi dữ liệu thành tf.data.Dataset
# Nguyen Việt Quang B22DCCN650
train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train))
val_ds   = tf.data.Dataset.from_tensor_slices((X_val, y_val))
test_ds  = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```

Thêm batch, shuffle, prefetch

```
# Thiết lập batch size và prefetching
# Nguyen Việt Quang B22DCCN650
batch_size = 32

train_ds = train_ds.shuffle(500).batch(batch_size).prefetch(tf.data.AUTOTUNE)
val_ds   = val_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
test_ds  = test_ds.batch(batch_size).prefetch(tf.data.AUTOTUNE)
```

2. Xây dựng mô hình (Model Building)

Xây dựng 2 mô hình

```

# ----- Build Model A -----
def build_model_A(input_shape):
    model = keras.Sequential([
        layers.Input(shape=input_shape),
        layers.Dense(128, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(64, activation='relu'),
        layers.Dense(32, activation='relu'),
        layers.Dense(16, activation='relu'),
        layers.Dense(8, activation='relu'),
        layers.Dense(1, activation='sigmoid') # output probability
    ], name="MLP_7layers")
    return model

def build_model_B(input_shape):
    model = keras.Sequential([
        layers.Input(shape=input_shape),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(32, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(16, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ], name="MLP_dropout")

    return model

```

```

input_shape = (X_train.shape[1],) # e.g. (8,)

model_A = build_model_A(input_shape)
model_A.summary()
model_B = build_model_B(input_shape)
model_B.summary()

```

kiến trúc mô hình A:

Model: "MLP_7layers"

Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 128)	1,152
dense_28 (Dense)	(None, 64)	8,256
dense_29 (Dense)	(None, 64)	4,160
dense_30 (Dense)	(None, 32)	2,080
dense_31 (Dense)	(None, 16)	528
dense_32 (Dense)	(None, 8)	136
dense_33 (Dense)	(None, 1)	9

Total params: 16,321 (63.75 KB)

Trainable params: 16,321 (63.75 KB)

Non-trainable params: 0 (0.00 B)

Kiến trúc mô hình B

Model: "MLP_dropout"

Layer (type)	Output Shape	Param #
dense_34 (Dense)	(None, 64)	576
dropout_2 (Dropout)	(None, 64)	0
dense_35 (Dense)	(None, 32)	2,080
dropout_3 (Dropout)	(None, 32)	0
dense_36 (Dense)	(None, 16)	528
dense_37 (Dense)	(None, 1)	17

Total params: 3,201 (12.50 KB)

Trainable params: 3,201 (12.50 KB)

Non-trainable params: 0 (0.00 B)

3. Compile mô hình (Compile) – Cấu hình quá trình học

```
# ----- Compile -----|
# model A
optimizer = keras.optimizers.Adam(learning_rate=1e-3)
loss = 'binary_crossentropy'
metrics = ['accuracy', keras.metrics.Precision(name='precision'), keras.metrics.Recall(name='recall')]
model_A.compile(optimizer=optimizer, loss=loss, metrics=metrics)
# model B
optimizer_B = keras.optimizers.Adam(learning_rate=1e-3)
model_B.compile(optimizer=optimizer_B, loss=loss, metrics=metrics)
```

4. Train mô hình (Training)

Xây dựng callback

```
# ----- Callbacks -----
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

callbacks = [
    EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True, verbose=1),
    ModelCheckpoint("./best_model_A.h5", monitor='val_loss', save_best_only=True, verbose=1)
]
callbacks_B = [
    EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True, verbose=1),
    ModelCheckpoint("./best_model_B.h5", monitor='val_loss', save_best_only=True, verbose=1)
]
```

Train mô hình

```
# ----- Training -----
EPOCHS = 30
batch_size = 32

history = model_A.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=EPOCHS,
    batch_size=batch_size,
    callbacks=callbacks,
    verbose=1
)
history_B = model_B.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=EPOCHS,
    batch_size=batch_size,
    callbacks=callbacks_B,
    verbose=1
)
```

Lưu mô hình ,đánh giá trên tập train và tập test

```

# ----- Load best weights -----
model_A.load_weights("./best_model_A.h5")
model_B.load_weights("./best_model_B.h5")
# ----- Predictions -----
val_probs = model_A.predict(X_val).ravel()
test_probs = model_A.predict(X_test).ravel()

val_preds = (val_probs >= 0.5).astype(int)
test_preds = (test_probs >= 0.5).astype(int)

val_probs_B = model_B.predict(X_val).ravel()
test_probs_B = model_B.predict(X_test).ravel()

val_preds_B = (val_probs_B >= 0.5).astype(int)
test_preds_B = (test_probs_B >= 0.5).astype(int)

```

Lưu lại kết quả

```

results = {
    'val_acc': accuracy_score(y_val, val_preds),
    'test_acc': accuracy_score(y_test, test_preds),
    'val_precision': precision_score(y_val, val_preds, zero_division=0),
    'test_precision': precision_score(y_test, test_preds, zero_division=0),
    'val_recall': recall_score(y_val, val_preds, zero_division=0),
    'test_recall': recall_score(y_test, test_preds, zero_division=0),
    'val_mae': mean_absolute_error(y_val, val_probs),
    'test_mae': mean_absolute_error(y_test, test_probs),
    'val_mse': mean_squared_error(y_val, val_probs),
    'test_mse': mean_squared_error(y_test, test_probs),
    'val_rmse': np.sqrt(mean_squared_error(y_val, val_probs)),
    'test_rmse': np.sqrt(mean_squared_error(y_test, test_probs))
}
print("==== Results for Model A ====")
print(results)

```

```
results_B = {  
    'val_acc': accuracy_score(y_val, val_preds_B),  
    'test_acc': accuracy_score(y_test, test_preds_B),  
    'val_precision': precision_score(y_val, val_preds_B, zero_division=0),  
    'test_precision': precision_score(y_test, test_preds_B, zero_division=0),  
    'val_recall': recall_score(y_val, val_preds_B, zero_division=0),  
    'test_recall': recall_score(y_test, test_preds_B, zero_division=0),  
    'val_mae': mean_absolute_error(y_val, val_probs_B),  
    'test_mae': mean_absolute_error(y_test, test_probs_B),  
    'val_mse': mean_squared_error(y_val, val_probs_B),  
    'test_mse': mean_squared_error(y_test, test_probs_B),  
    'val_rmse': np.sqrt(mean_squared_error(y_val, val_probs_B)),  
    'test_rmse': np.sqrt(mean_squared_error(y_test, test_probs_B))  
}  
  
print("\n===== Results for Model B =====")  
print(results_B)
```

Visualize

```
# Nguyễn Việt Quang B22DCCN650
# visualize
import matplotlib.pyplot as plt
# --- Plot Training History ---
plt.figure(figsize=(16, 10))
# Loss
plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Train Loss A')
plt.plot(history.history['val_loss'], label='Val Loss A')
plt.plot(history_B.history['loss'], label='Train Loss B')
plt.plot(history_B.history['val_loss'], label='Val Loss B')
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
# Accuracy
plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='Train Acc A')
plt.plot(history.history['val_accuracy'], label='Val Acc A')
plt.plot(history_B.history['accuracy'], label='Train Acc B')
plt.plot(history_B.history['val_accuracy'], label='Val Acc B')
plt.title("Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
```

```

# Nguyễn Việt Quang B22DCCN650
plt.subplot(2, 2, 3)
plt.plot(history.history['precision'], label='Train Precision A')
plt.plot(history.history['val_precision'], label='Val Precision A')
plt.plot(history_B.history['precision'], label='Train Precision B')
plt.plot(history_B.history['val_precision'], label='Val Precision B')
plt.title("Precision over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Precision")
plt.legend()

# Recall
plt.subplot(2, 2, 4)
plt.plot(history.history['recall'], label='Train Recall A')
plt.plot(history.history['val_recall'], label='Val Recall A')
plt.plot(history_B.history['recall'], label='Train Recall B')
plt.plot(history_B.history['val_recall'], label='Val Recall B')
plt.title("Recall over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Recall")
plt.legend()

```

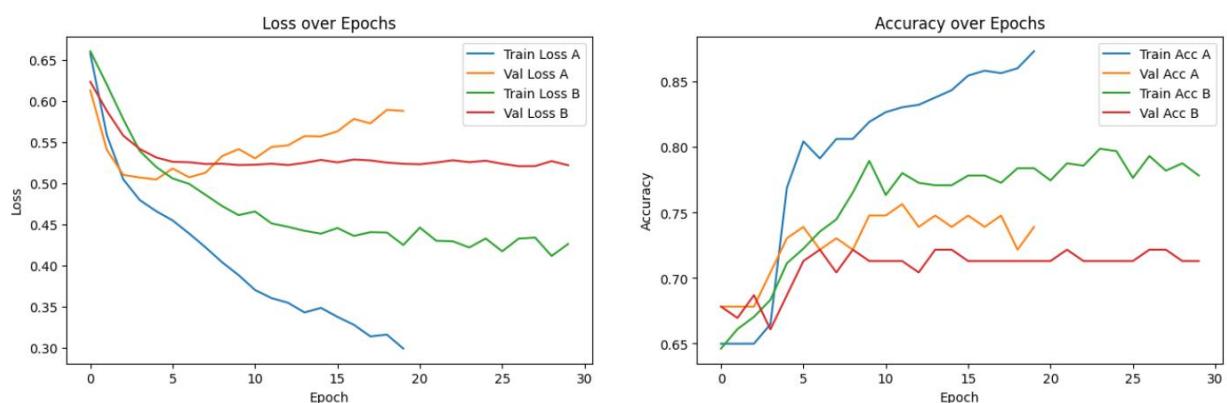
Kết quả

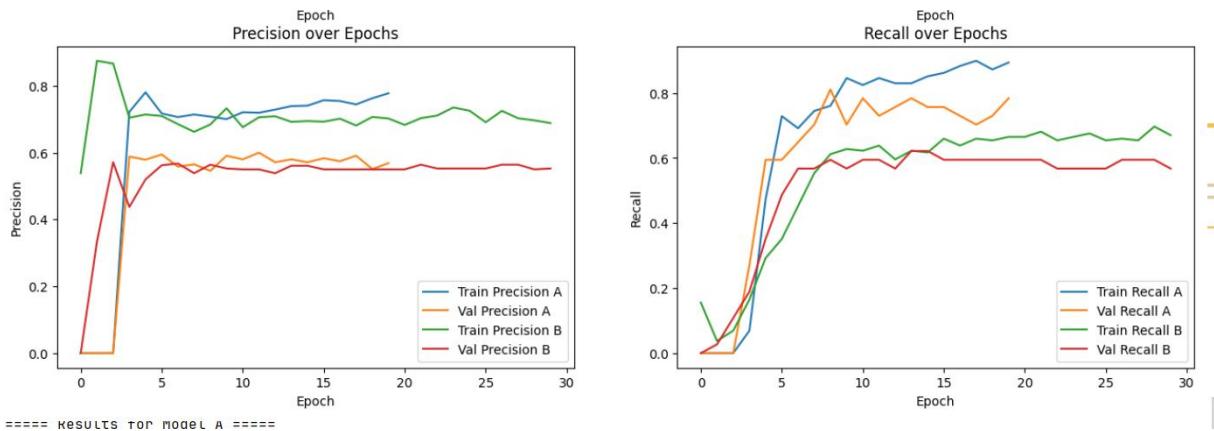
```

4/4 - 0s 22ms/step
===== Results for Model A =====
{'val_acc': 0.7304347826086957, 'test_acc': 0.75, 'val_precision': 0.5789473684210527, 'test_precision': 0.6666666666666666,
 'val_recall': 0.5945945945945946, 'test_recall': 0.6511627906976745, 'val_mae': 0.3397722542285919, 'test_mae': 0.35973021388053894,
 'val_mse': 0.17035235464572906, 'test_mse': 0.18579180538654327, 'val_rmse': np.float64(0.4127376341524105), 'test_rmse': np.float64(0.4310357356258797)}


===== Results for Model B =====
{'val_acc': 0.7217391304347827, 'test_acc': 0.7586206896551724, 'val_precision': 0.5641025641025641, 'test_precision': 0
 .6666666666666666, 'val_recall': 0.5945945945945946, 'test_recall': 0.6976744186046512, 'val_mae': 0.31930479407310486, 'test_mae':
 0.31466421484947205, 'val_mse': 0.17652031779289246, 'test_mse': 0.17011313140392303, 'val_rmse': np.float64(0.4201432110517704),
 'test_rmse': np.float64(0.4124477317235761)}
<matplotlib.legend.Legend at 0x2473677ef90>

```





Đánh giá Mô hình A

Ưu điểm

- Train Loss giảm đều và mạnh, từ ~ 0.55 xuống ~ 0.30 → mô hình học tốt trên tập huấn luyện.
- Train Accuracy tăng liên tục, đạt ~ 0.86 → mô hình có khả năng fitting dữ liệu train rất tốt.
- Val Loss giảm rồi tăng mạnh \rightarrow mô hình bị overfit nặng
- Val Accuracy chỉ được ~ 0.75 thấp hơn tập train

Nhược điểm

- Độ chênh giữa Train Acc (~ 0.86) và Val Acc (~ 0.75) tuy không quá lớn nhưng vẫn cho thấy mức độ overfitting nhẹ.

Đánh giá Mô hình B

- Train Loss giảm tương đối ổn định đến khoảng 0.45.
- Train Accuracy tăng đến khoảng 0.78–0.79 → mức chấp nhận được.
- Val Accuracy chỉ khoảng 0.71–0.72 và dao động, thấp hơn mô hình A.
- Mô hình bị overfit nhẹ

Kết luận: mô hình A tốt hơn mô hình B nhưng cả 2 accuracy của 2 model đều khá thấp chỉ khoảng 0.75–0.76

Kết quả triển khai



Dự đoán Bệnh Tiểu đường (MLP Model)

Vui lòng nhập 8 chỉ số của bệnh nhân để dự đoán xác suất mắc bệnh:

1. Số lần mang thai (Pregnancies):

1

2. Glucose (mg/dL):

100

3. Huyết áp (BloodPressure) (mmHg):

70

4. Độ dày da (SkinThickness) (mm):

20

5. Insulin (mu U/ml):

100

6. BMI:

30

7. DPF (Hàm phâ hệ):

0.3

8. Tuổi (Age):

30

Dự đoán

Kết quả dự đoán: Không (Âm tính) (Xác suất: 0.1927)

Case study 4: Dự báo thời tiết theo thời gian

Chuẩn bị dữ liệu (3 tập dữ liệu từ kaggle)

Weathei.csv : <https://www.kaggle.com/datasets/vanviethieuanh/vietnam-weather-data>

Weather2.csv : <https://www.kaggle.com/datasets/l duy hi/vietnam-real-time-weather-data?select=weather-vn-1.csv>

Weather3.csv : <https://www.kaggle.com/datasets/hoantainson/dataset-weather-vit-nam-trong-1-nm-li>

Mô tả dữ liệu:

Weather1.csv

Column Names	
0	province
1	max
2	min
3	wind
4	wind_d
5	rain
6	humidi
7	cloud
8	pressure
9	date

#	province	max	min	wind	wind_d	rain	humidi	cloud	pressure	date
0	Bac Lieu	27	22	17	NNE	6.9	90	71	1010	2009-01-01
1	Bac Lieu	31	25	20	ENE	0	64	24	1010	2010-01-01
2	Bac Lieu	29	24	14	E	0	75	45	1008	2011-01-01
3	Bac Lieu	30	24	30	E	0	79	52	1012	2012-01-01
4	Bac Lieu	31	25	20	ENE	0	70	24	1010	2013-01-01
5	Bac Lieu	28	23	14	ENE	0	75	55	1012	2014-01-01
6	Bac Lieu	29	23	10	ENE	0.4	75	42	1012	2015-01-01
7	Bac Lieu	32	24	22	ENE	0	63	9	1015	2016-01-01
8	Bac Lieu	30	24	20	ENE	0.5	76	35	1011	2017-01-01
9	Bac Lieu	29	23	16	E	0	70	33	1010	2018-01-01

Weather2.csv

	Column Names
0	time
1	province
2	city
3	temperature
4	temp_min
5	temp_max
6	humidity
7	feels_like
8	visibility
9	precipitation
10	cloudcover
11	wind_speed
12	wind_gust
13	wind_direction
14	pressure
15	is_day
16	weather_code
17	weather_main
18	weather_description
19	weather_icon

	time	province	city	temperature	temp_min	temp_max	humidity	feels_like	visibility
0	2025-06-29 23:29:22	An Giang-Chau Doc Chau Doc		25.63	25.63	25.63	79	26.31	
1	2025-06-29 23:30:30	An Giang-Chau Doc Chau Doc		25.63	25.63	25.63	79	26.31	
2	2025-06-29 23:32:58	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
3	2025-06-29 23:35:01	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
4	2025-06-29 23:37:03	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
5	2025-06-29 23:39:09	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
6	2025-06-29 23:41:11	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
7	2025-06-29 23:43:21	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
8	2025-06-29 23:45:25	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	
9	2025-06-29 23:47:27	An Giang-Chau Doc Chau Doc		25.48	25.48	25.48	78	26.12	

	visibility	precipitation	cloudcover	wind_speed	wind_gust	wind_direction	pressure	is_day	weather_main
1	10000	0	100	2.36	4.58	277	1008	0	
1	10000	0	100	2.36	4.58	277	1008	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	
2	10000	0	100	2.24	4.24	262	1007	0	

	weather_code	weather_main	weather_description	weather_icon
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n
0	804	Clouds	overcast clouds	04n

Weather3.csv

Column Names

```
0 location.name
1 location.region
2 location.terrain
3 location.country
4 location.lat
5 location.lon
6 date
7 date_epoch
8 day.maxtemp_c
9 day.maxtemp_f
10 day.mintemp_c
11 day.mintemp_f
12 day.avgtemp_c
13 day.avgtemp_f
14 day.maxwind_mph
15 day.maxwind_kph
16 day.totalprecip_mm
17 day.totalprecip_in
18 day.totalsnow_cm
19 day.avgvis_km
```

```
18 day.totalsnow_cm  
19 day.avgvis_km  
20 day.avgvis_miles  
21 day.avghumidity  
22 day.daily_will_it_rain  
23 day.daily_chance_of_rain  
24 day.condition.text  
25 day.condition.icon  
26 day.condition.code  
27 day.uv  
28 astro.sunrise  
29 astro.sunset  
30 astro.moonrise  
31 astro.moonset  
32 astro.moon_phase  
33 astro.moon_illumination
```

Gộp 3 bộ dataset lại

Chuẩn hóa 3 file lại có các cột chung sau

- province
- date
- temp_max
- temp_min
- wind_speed
- precipitation
- humidity

code xử lý:

```

import pandas as pd
# Nguyễn Việt Quang B22DCCN650
# chuẩn hóa dữ liệu từ weather1.csv
df1 = pd.read_csv("DATA/weather1.csv")

df1 = df1.rename(columns={
    "province": "province",
    "date": "date",
    "max": "temp_max",
    "min": "temp_min",
    "wind": "wind_speed",
    "rain": "precipitation",
    "humidi": "humidity"
})

df1 = df1[["province", "date", "temp_max", "temp_min", "wind_speed", "precipitation", "humidity"]]

```

```

# Nguyễn Việt Quang B22DCCN650
# chuẩn hóa dữ liệu từ weather2.csv
df2 = pd.read_csv("DATA/weather2.csv")

df2 = df2.rename(columns={
    "province": "province",
    "time": "date",           # bạn yêu cầu date là time
    "temp_max": "temp_max",
    "temp_min": "temp_min",
    "wind_speed": "wind_speed",
    "precipitation": "precipitation",
    "humidity": "humidity"
})

df2 = df2[["province", "date", "temp_max", "temp_min", "wind_speed", "precipitation", "humidity"]]

```

```

# Nguyễn Việt Quang B22DCCN650
# chuẩn hóa dữ liệu từ weather3.csv
df3 = pd.read_csv("DATA/weather3.csv")

df3 = df3.rename(columns={
    "location.name": "province",
    "date": "date",
    "day.maxtemp_c": "temp_max",
    "day.mintemp_c": "temp_min",
    "day.maxwind_kph": "wind_speed",
    "day.totalprecip_mm": "precipitation",
    "day.avghumidity": "humidity"
})

df3 = df3[["province", "date", "temp_max", "temp_min", "wind_speed", "precipitation", "humidity"]]

```

Chuẩn hóa dữ liệu ở cột date về cùng 1 dạng

```

# Nguyễn Việt Quang B22DCCN650
# Chuyển toàn bộ về datetime
df1["date"] = pd.to_datetime(df1["date"], errors="coerce")
df2["date"] = pd.to_datetime(df2["date"], errors="coerce")
df3["date"] = pd.to_datetime(df3["date"], errors="coerce")

# Chuẩn hóa lại định dạng yyyy-mm-dd
df1["date"] = df1["date"].dt.strftime("%Y-%m-%d")
df2["date"] = df2["date"].dt.strftime("%Y-%m-%d")
df3["date"] = df3["date"].dt.strftime("%Y-%m-%d")

```

gộp 3 file lại

```

# Nguyễn Việt Quang B22DCCN650
# gộp 3 file lại
df = pd.concat([df1, df2, df3], ignore_index=True)

```

Vấn đề ở cột province:

- Một số là **tên tỉnh**, một số là **tên thành phố trong tỉnh**
- Một số là **tỉnh-huyện-thành**, ví dụ:
AN GIANG-LONG XUYEN, KHANH HOA-NHA TRANG, THANH HOA-THANH HOA
- Một số còn bị **viết khác nhau**:
HA NOI, HANOI, TP HO CHI MINH, HO CHI MINH CITY, TP. HO CHI MINH
- Một số vẫn còn **chữ Đ**, BINH ĐINH, ĐAK NONG, v.v.
- Một số có **dấu cách + dấu gạch lộn xộn**

Code xử lý chuẩn hóa code province:

```
# chuẩn hóa cột province
# Nguồn Việt Quang B22DCCN650
import unicodedata

SPECIAL_MAP = {
    "TP HO CHI MINH": "HO CHI MINH",
    "TP. HO CHI MINH": "HO CHI MINH",
    "HO CHI MINH CITY": "HO CHI MINH",
    "HANOI": "HA NOI",
    "DA NANG": "DA NANG",
    "ĐÀ NẴNG": "DA NANG"
}

def remove_accents(text):
    text = unicodedata.normalize("NFD", text)
    return ''.join(c for c in text if unicodedata.category(c) != 'Mn')

def normalize_province(value):
    if not isinstance(value, str):
        return value
```

```

def normalize_province(value):
    if not isinstance(value, str):
        return value

    # 1. Bỏ dấu + uppercase
    v = remove_accents(value).upper().strip()

    # 2. Nếu có dạng "TINH-THANH PHO", lấy phần TỈNH
    v = v.split("-")[0].strip()

    # 3. Chuẩn hóa trường hợp đặc biệt
    if v in SPECIAL_MAP:
        v = SPECIAL_MAP[v]

    return v

df["province"] = df["province"].apply(normalize_province)

```

Huấn luyện trên 3 khu vực Hồ Chí Minh, Hà Nội, Đà Nẵng

```

# chỉ để lại khu vực HO CHI MINH, HA NOI, DA NANG
df = df[df["province"].isin(["HO CHI MINH", "HA NOI", "DA NANG"])].reset_index(drop=True)

# lưu file đã chuẩn hóa
df.to_csv("DATA/weather_cleaned.csv", index=False)

```

I. Cấu hình

Tập lệnh này thực hiện huấn luyện và so sánh hai mô hình **Deep SimpleRNN** và **Deep LSTM** để dự đoán loại hình thời tiết (weather_type) vào **ngày tiếp theo** dựa trên chuỗi dữ liệu thời tiết của 14 ngày trước đó.

1. Cấu hình Quan trọng

Tham số	Giá trị	Mô tả
---------	---------	-------

Tham số	Giá trị	Mô tả
DATA_PATH	"DATA/weather_cleaned.csv"	Tệp dữ liệu đầu vào đã được chuẩn hóa.
SEQUENCE_LENGTH	14	Độ dài chuỗi lịch sử đầu vào (14 ngày).
PRED_STEPS	1	Số bước dự báo (dự báo ngày tiếp theo).
EPOCHS	18	Số epoch huấn luyện tối đa.
TEST_SIZE	0.2	Tỷ lệ dữ liệu dành cho tập kiểm tra (20%).

```

# -----
# 1. Params (điều chỉnh nếu cần)
# # Nguyễn Việt Quang B22DCCN650
DATA_PATH = "DATA/weather_cleaned.csv"
RESULT_DIR = "results_weather_models"
os.makedirs(RESULT_DIR, exist_ok=True)

SEQUENCE_LENGTH = 14      # dùng 14 ngày để dự đoán ngày tiếp theo
PRED_STEPS = 1            # dự báo 1 bước (next day). mở rộng nếu cần.
TEST_SIZE = 0.2
RANDOM_STATE = 42
BATCH_SIZE = 64
EPOCHS = 18               # sửa nếu muốn nhiều hơn
LEARNING_RATE = 1e-3

```

2. Định Nghĩa Nhãn Loại Thời Tiết (make_weather_type)

Hàm này được dùng để phân loại lượng mưa (precipitation) thành 4 loại hình thời tiết:

Giá trị	precipitation (mm)	Mô tả
0	0	Không mưa (No rain)
1	< 2	Mưa nhẹ (Light rain)
2	2 <= precip < 10	Mưa vừa (Moderate rain)
3	>= 10	Mưa lớn (Heavy rain)

```

i2 # -----
i3 # 2. Utility: create weather_type label from precipitation
i4 # Nguyễn Việt Quang B22DCCN650
i5 def make_weather_type(precip_mm):
i6     # 0: no rain, 1: light (<2mm), 2: moderate (2-10), 3: heavy (>=10)
i7     if pd.isna(precip_mm):
i8         return 0
i9     if precip_mm == 0:
i0         return 0
i1     if precip_mm < 2:
i2         return 1
i3     if precip_mm < 10:
i4         return 2
i5     return 3
,
```

II. Tiền Xử Lý và Kỹ Thuật Đặc Trung

1. Tái và Làm Sạch Dữ Liệu Cơ Bản

```

# 3. Load & preprocess
# Nguyễn Việt Quang B22DCCN650
df = pd.read_csv(DATA_PATH, parse_dates=["date"])
# ensure columns exist
required_cols = ["province", "date", "temp_max", "temp_min", "wind_speed", "precipitation", "humidity"]
for c in required_cols:
    if c not in df.columns:
        raise ValueError(f"Thiếu cột {c} trong {DATA_PATH}")

# Drop rows with missing province or date
df = df.dropna(subset=["province", "date"]).copy()

# Convert date to datetime (if not)
df['date'] = pd.to_datetime(df['date'], errors='coerce')
df = df.dropna(subset=['date'])

# Sort
df = df.sort_values(["province", "date"]).reset_index(drop=True)

# Make weather_type label
df['weather_type'] = df['precipitation'].apply(make_weather_type).astype(int)

# Optionally: fill missing numeric features with forward fill per province then global mean
num_cols = ["temp_max", "temp_min", "wind_speed", "precipitation", "humidity"]
df[num_cols] = df.groupby("province")[num_cols].transform(lambda x: x.ffill().bfill())
df[num_cols] = df[num_cols].fillna(df[num_cols].mean())

```

2. Chuẩn Hóa Thời Gian (Reindex Time Series)

Mã này đảm bảo rằng mỗi tỉnh thành có chuỗi dữ liệu liên tục theo ngày (freq='D') bằng cách thêm các ngày bị thiếu và sử dụng kỹ thuật **Forward-fill** (diễn tiến) và **Backward-fill** (diễn lùi) để xử lý các giá trị thiếu trong chuỗi thời gian.

```
# -----
# 4. Reindex time series per province to ensure continuous dates (daily)
# Nguyễn Việt Quang B22DCCN650
# This helps when some provinces miss dates; we will forward-fill features (or keep NaN handling policy).
provinces = df['province'].unique().tolist()
rows = []
for p in provinces:
    g = df[df['province']==p].set_index('date').sort_index()
    if g.empty:
        continue
    # create full date index from min to max
    full_idx = pd.date_range(g.index.min(), g.index.max(), freq='D')
    g = g.reindex(full_idx)
    # fill province column
    g['province'] = p
    # forward fill numeric then backfill
    g[num_cols] = g[num_cols].ffill().bfill()
    # weather_type: if still NaN use 0 (no rain) as conservative default
    g['weather_type'] = g['weather_type'].fillna(0).astype(int)
    g = g.reset_index().rename(columns={'index':'date'})
    rows.append(g[['province','date'] + num_cols + ['weather_type']])

df_full = pd.concat(rows, ignore_index=True)
df = df_full.copy()
```

3. Kỹ Thuật Đặc Trưng (Feature Engineering)

Thêm các đặc trưng phụ trợ như nhiệt độ trung bình (temp_avg), ngày trong năm (day_of_year), và tháng (month) để cung cấp thêm thông tin theo mùa cho mô hình.

```
# -----
# 5. Feature engineering: create additional features if desired
# Nguyễn Việt Quang B22DCCN650
#
# Example: temp_avg, day_of_year, month, lag features can be created in sequence stage
df['temp_avg'] = (df['temp_max'] + df['temp_min']) / 2
df['day_of_year'] = df['date'].dt.dayofyear
df['month'] = df['date'].dt.month

feature_cols = ["temp_max","temp_min","temp_avg","wind_speed","precipitation","humidity","day_of_year","month"]
```

III. Xây Dựng Chuỗi và Chia Dữ Liệu

1. Tạo Chuỗi Dữ Liệu (Sequences Creation)

Hàm `create_sequences_for_province` sử dụng cửa sổ trượt (**Sliding Window**) để tạo các cặp X,y trong đó X là chuỗi 14 ngày và y là loại hình thời tiết vào ngày thứ 15.

```

"""
# 7. Create sequences (X, y) per province using sliding window
# Nguyễn Việt Quang B22DCCN650
def create_sequences_for_province(gdf, seq_len=SEQUENCE_LENGTH, pred_steps=PRED_STEPS, feature_cols=feature_cols,
    target_col='weather_type'):
    """
    gdf: df for single province sorted by date
    returns list of (X_seq, y_target)
    X_seq shape: (seq_len, n_features)
    y_target: integer class of next day (or next pred_steps aggregated)
    """

    Xs, ys = [], []
    arr_feat = gdf[feature_cols].values
    arr_target = gdf[target_col].values
    n = len(gdf)
    for i in range(n - seq_len - pred_steps + 1):
        X_seq = arr_feat[i:i+seq_len]
        # target is weather_type at step i+seq_len (next day)
        y = arr_target[i+seq_len]  # single-step
        Xs.append(X_seq)
        ys.append(y)
    return np.array(Xs), np.array(ys)

```

2. Chia Tập Huấn Luyện/Kiểm Tra (Train-Test Split)

Sử dụng **Stratified Shuffle Split** để đảm bảo phân phối các lớp mục tiêu (weather_type) được cân bằng giữa tập huấn luyện và tập kiểm tra.

```

# Nguyễn Việt Quang B22DCCN650
# do stratified split by class
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size=TEST_SIZE, random_state=RANDOM_STATE)
train_idx, test_idx = next(sss.split(X, y))
X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y[train_idx], y[test_idx]
prov_train, prov_test = province_index[train_idx], province_index[test_idx]

print("Train:", X_train.shape, y_train.shape, "Test:", X_test.shape, y_test.shape)

```

3. Chuẩn Hóa và Mã Hóa (Scaling and Encoding)

- **StandardScaler:** Được fitted (khớp) trên tập huấn luyện để chuẩn hóa các đặc trưng, giúp mô hình hội tụ nhanh hơn.
- **LabelEncoder:** Mã hóa nhãn \$y\\$ (0, 1, 2, 3) thành định dạng One-Hot Encoding (y_{train_cat}) cần thiết cho bài toán phân loại đa lớp.

```

# -----
# 9. Scale features: fit scaler on flattened training features then apply per time-step
# Nguyễn Việt Quang B22DCCN650
n_features = X_train.shape[2]
scaler = StandardScaler()
# fit on 2D data: (n_samples*seq_len, n_features)
scaler.fit(X_train.reshape(-1, n_features))
# transform
def scale_X(X_in):
    s = scaler.transform(X_in.reshape(-1, n_features)).reshape(X_in.shape)
    return s

X_train_scaled = scale_X(X_train)
X_test_scaled = scale_X(X_test)

# save scaler
joblib.dump(scaler, os.path.join(RESULT_DIR, "scaler.save"))

```

```

# 10. Encode labels
# Nguyễn Việt Quang B22DCCN650
le = LabelEncoder()
y_train_enc = le.fit_transform(y_train)
y_test_enc = le.transform(y_test)
num_classes = len(le.classes_)
print("Classes:", le.classes_)

# save label encoder
joblib.dump(le, os.path.join(RESULT_DIR, "label_encoder.save"))

# convert to categorical for training
y_train_cat = utils.to_categorical(y_train_enc, num_classes=num_classes)
y_test_cat = utils.to_categorical(y_test_enc, num_classes=num_classes)

```

IV. Xây Dựng và Huấn Luyện Mô Hình Sâu

1. Kiến Trúc Mô Hình (Deep Sequence Architecture)

Hàm build_rnn_model tạo ra các mô hình RNN/LSTM **Sâu** với **7 tầng hồi quy** được xếp chồng lên nhau (return_sequences=True cho 6 tầng đầu, return_sequences=False cho tầng cuối) và kết thúc bằng một lớp Dense với activation softmax cho phân loại.

```

# Nguyễn Việt Quang B22DCCN650
def build_rnn_model(input_shape, num_classes, recurrent_type='simple', n_recur_layers=7, units=64, dropout=0.2):
    inp = layers.Input(shape=input_shape, name='input')
    x = inp
    for i in range(n_recur_layers):
        # For stacking, all but last must return_sequences=True
        is_last = (i == n_recur_layers - 1)
        if recurrent_type == 'simple':
            x = layers.SimpleRNN(units, return_sequences=not is_last, dropout=dropout, name=f"rnn_{i+1}")(x)
        elif recurrent_type == 'gru':
            x = layers.GRU(units, return_sequences=not is_last, dropout=dropout, name=f"gru_{i+1}")(x)
        elif recurrent_type == 'lstm':
            x = layers.LSTM(units, return_sequences=not is_last, dropout=dropout, name=f"lstm_{i+1}")(x)
        else:
            raise ValueError("Unknown recurrent_type")
        # optionally batchnorm
        if not is_last:
            x = layers.BatchNormalization()(x)
    # Dense head
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    out = layers.Dense(num_classes, activation='softmax', name='out')(x)
    model = models.Model(inp, out)
    return model

```

```

# Nguyễn Việt Quang B22DCCN650
# build rnn (SimpleRNN)
input_shape = (SEQUENCE_LENGTH, n_features)
rnn_model = build_rnn_model(input_shape, num_classes, recurrent_type='simple', n_recur_layers=7, units=128)
rnn_model.compile(optimizer=tf.keras.optimizers.Adam(LEARNING_RATE),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
rnn_model.summary()

# build lstm
lstm_model = build_rnn_model(input_shape, num_classes, recurrent_type='lstm', n_recur_layers=7, units=128)
lstm_model.compile(optimizer=tf.keras.optimizers.Adam(LEARNING_RATE),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
lstm_model.summary()

```

2. Huấn Luyện và Cân Bằng Lớp (Class Weighting)

Sử dụng **Class Weighting** để giải quyết vấn đề mất cân bằng lớp (ví dụ: ngày không mưa thường nhiều hơn ngày mưa lớn) và **Early Stopping** cùng với **Model Checkpoint** để tìm mô hình tốt nhất.

```

# -----
# 12. Training callbacks
# Nguyễn Việt Quang B22DCCN650
checkpoint_rnn = callbacks.ModelCheckpoint(os.path.join(RESULT_DIR, "rnn_best.h5"),
                                           monitor='val_accuracy', save_best_only=True, mode='max')
checkpoint_lstm = callbacks.ModelCheckpoint(os.path.join(RESULT_DIR, "lstm_best.h5"),
                                            monitor='val_accuracy', save_best_only=True, mode='max')
es = callbacks.EarlyStopping(monitor='val_loss', patience=8, restore_best_weights=True)

# class weights (help if classes imbalanced)
from sklearn.utils.class_weight import compute_class_weight
class_weights_vals = compute_class_weight('balanced', classes=np.unique(y_train_enc), y=y_train_enc)
class_weight = {i: w for i, w in enumerate(class_weights_vals)}
print("class_weight:", class_weight)

# -----
# 13. Fit models (you can train one at a time if GPU limited)
# -----

history_rnn = rnn_model.fit(
    X_train_scaled, y_train_cat,
    validation_split=0.1,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[checkpoint_rnn, es],
    class_weight=class_weight,
    verbose=2
)

# Nguyễn Việt Quang B22DCCN650
history_lstm = lstm_model.fit(
    X_train_scaled, y_train_cat,
    validation_split=0.1,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[checkpoint_lstm, es],
    class_weight=class_weight,
    verbose=2
)

```

```

7     # Save final models
8     rnn_model.save(os.path.join(RESULT_DIR, "rnn_final.h5"))
9     lstm_model.save(os.path.join(RESULT_DIR, "lstm_final.h5"))
10
11    # Save training history
12    pd.DataFrame(history_rnn.history).to_csv(os.path.join(RESULT_DIR, "history_rnn.csv"), index=False)
13    pd.DataFrame(history_lstm.history).to_csv(os.path.join(RESULT_DIR, "history_lstm.csv"), index=False)

```

V. Đánh Giá và Dự Báo

1. Dánh Giá trên Tập Kiểm Tra (Test Set Evaluation)

Các mô hình được đánh giá bằng **Accuracy** và **Classification Report** trên tập kiểm tra (X_test_scaled) để so sánh hiệu suất.

```

8 def eval_and_report(model, X_test, y_test_enc, le, title="model"):
9     preds = model.predict(X_test, batch_size=128)
10    y_pred = np.argmax(preds, axis=1)
11    from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
12    acc = accuracy_score(y_test_enc, y_pred)
13    print(f"{title} Test Accuracy: {acc:.4f}")
14    print("Classification report:")
15    print(classification_report(y_test_enc, y_pred, target_names=[str(x) for x in le.classes_]))
16    cm = confusion_matrix(y_test_enc, y_pred)
17    return acc, cm, y_pred
18
19 acc_rnn, cm_rnn, y_pred_rnn = eval_and_report(rnn_model, X_test_scaled, y_test_enc, le, title="RNN")
20 acc_lstm, cm_lstm, y_pred_lstm = eval_and_report(lstm_model, X_test_scaled, y_test_enc, le, title="LSTM")

```

2. Biểu Đồ

Tập lệnh tạo ra các biểu đồ **Accuracy/Loss** để trực quan hóa quá trình huấn luyện và kết quả.

```

# vẽ biểu đồ accuracy và loss
# Nguyễn Việt Quang B22DCCN650
def plot_history(history, title, result_dir):
    plt.figure(figsize=(12,5))
    # accuracy
    plt.subplot(1,2,1)
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(f'{title} Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    # loss
    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.title(f'{title} Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(result_dir, f"{title}_training_history.png"))
    plt.show()
plot_history(history_rnn, "RNN", RESULT_DIR)
plot_history(history_lstm, "LSTM", RESULT_DIR)

```

3. Dự Báo Ngày Tiếp Theo (Next-Day Prediction Example)

Sử dụng mô hình đã huấn luyện, tập lệnh trích xuất 14 ngày cuối cùng của mỗi tỉnh thành, chuẩn hóa, và dự đoán loại hình thời tiết vào ngày tiếp theo, sau đó lưu kết quả vào file JSON.

```

# Nguyễn Việt Quang B22DCCN650
# We'll take the last SEQUENCE_LENGTH days for each province and predict next day weather_type
def predict_next_day_all_provinces(model, df, feature_cols, scaler, seq_len=SEQUENCE_LENGTH):
    preds = {}
    for p in df['province'].unique():
        g = df[df['province']==p].sort_values('date').reset_index(drop=True)
        if len(g) < seq_len:
            continue
        last_seq = g[feature_cols].values[-seq_len:]
        last_seq_scaled = scaler.transform(last_seq).reshape(1, seq_len, -1)
        prob = model.predict(last_seq_scaled)[0]
        pred_class = np.argmax(prob)
        preds[p] = {"pred_class": int(pred_class), "prob": prob.tolist()}

    return preds

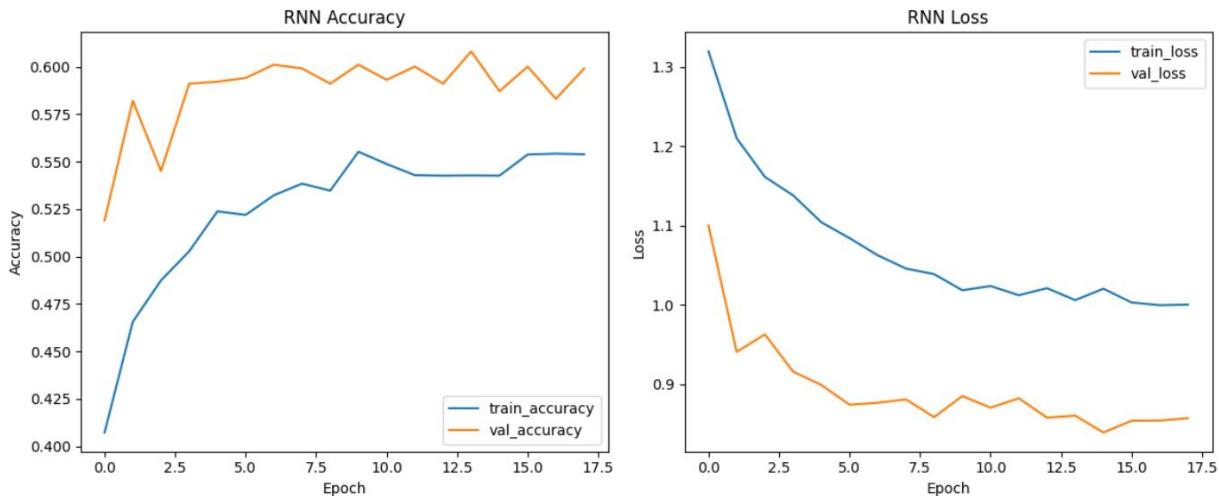
preds_rnn = predict_next_day_all_provinces(rnn_model, df, feature_cols, scaler)
preds_lstm = predict_next_day_all_provinces(lstm_model, df, feature_cols, scaler)

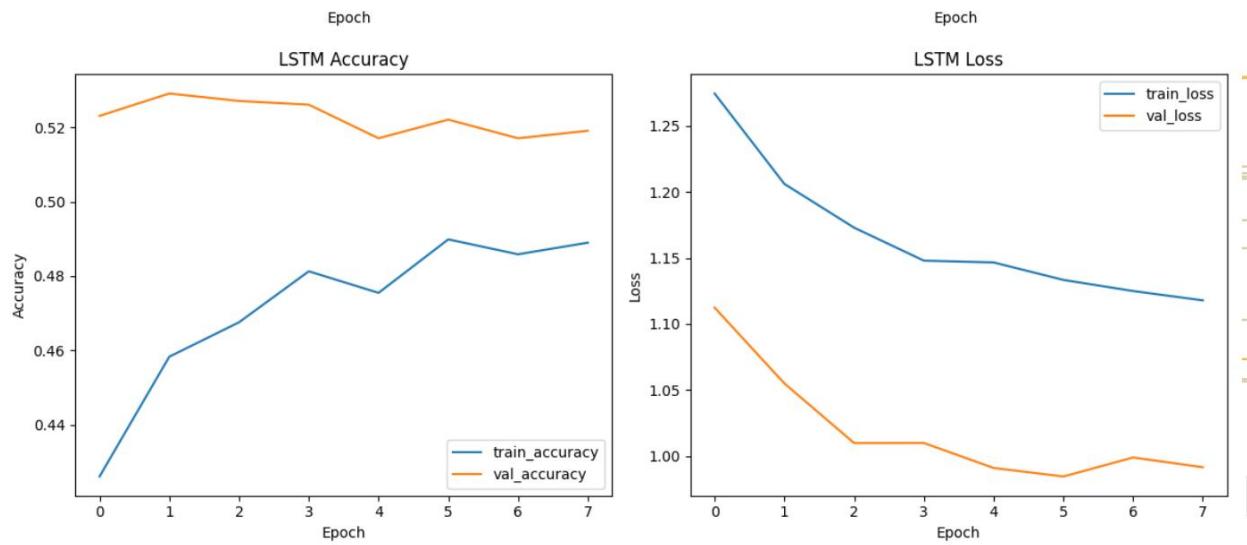
# Save predictions
import json
with open(os.path.join(RESULT_DIR, "next_day_preds_rnn.json"), "w") as f:
    json.dump(preds_rnn, f, indent=2, ensure_ascii=False)
with open(os.path.join(RESULT_DIR, "next_day_preds_lstm.json"), "w") as f:
    json.dump(preds_lstm, f, indent=2, ensure_ascii=False)

print("Next-day predictions saved to", RESULT_DIR)

```

Kết quả huấn luyện:





Dự báo các loại thời tiết vùng miền cả nước theo thời gian

Loại thời tiết (weather_type) từ cột precipitation (0: no rain, 1: light, 2: moderate, 3: heavy)

Mô hình rnn

```
{  
    "DA NANG": {  
        "pred_class": 3,  
        "prob": [  
            0.02332298830151558,  
            0.14850027859210968,  
            0.3842354714870453,  
            0.44394123554229736  
        ]  
    },  
    "HA NOI": {  
        "pred_class": 2,  
        "prob": [  
            0.0482010655105114,  
            0.2302381992340088,  
            0.4043623208999634,  
            0.3171984553337097  
        ]  
    },  
    "HO CHI MINH": {  
        "pred_class": 2,  
        "prob": [  
            0.06661935150623322,  
            0.24887262284755707,  
            0.4143671691417694,  
            0.2701408267021179  
        ]  
    }  
}
```

Mô hình LSTM

```
{
  "DA NANG": {
    "pred_class": 3,
    "prob": [
      0.07027098536491394,
      0.19099247455596924,
      0.32501131296157837,
      0.4137251675128937
    ]
  },
  "HA NOI": {
    "pred_class": 3,
    "prob": [
      0.05208412557840347,
      0.15400606393814087,
      0.3143957853317261,
      0.47951409220695496
    ]
  },
  "HO CHI MINH": {
    "pred_class": 0,
    "prob": [
      0.5212450623512268,
      0.20821411907672882,
      0.14031507074832916,
      0.1302257627248764
    ]
  }
}
```

Đánh giá

Cả hai mô hình RNN và LSTM đều đạt **accuracy khá thấp**, cho thấy đây là một bài toán phân loại khó:

- Dữ liệu thời tiết nhiều vùng miền **trùng lặp về đặc điểm chung**
- Dữ liệu từ nhiều nguồn cần **làm sạch sâu hơn**
- Một số feature quan trọng đối với phân loại miền (ví dụ: vị trí địa lý, độ cao, vĩ độ/kinh độ...) **không có trong dataset**
- Chuỗi thời gian 14 ngày có thể **không đủ dài**

Do đó, mặc dù mô hình không bị overfit, nhưng **hiệu suất tổng thể còn hạn chế** và cần các hướng cải thiện như:

- thêm feature mạnh hơn,
- tăng độ dài chuỗi,
- tuning siêu tham số,
- thử các mô hình mạnh hơn như GRU, CNN-LSTM, Transformer-based models.

Case study 5: Dự báo cổ phiếu (Phát hiện overfitting)

Mô tả dữ liệu

	date	open	high	low	close	volume	Name
0	2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
1	2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2	2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
3	2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
4	2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

Tập dữ liệu gồm dữ liệu về giá cổ phiếu của 3 công ty có tên sau: ALL, ADP, AJG

Danh sách các cột:

- date
- open
- high
- low
- close
- volume
- Name

I Cấu hình

1. Cấu hình Chính

Tham số	Giá trị	Mô tả
CSV_PATH	"DATA/stock.csv"	Đường dẫn đến tệp dữ liệu đầu vào.
OUTPUT_DIR	"models_and_results"	Thư mục lưu kết quả, mô hình và biểu đồ.
SEQ_LEN	60	Độ dài chuỗi lịch sử đầu vào (60 ngày).
HORIZON	7	Số bước dự báo lặp lại (7 ngày tiếp theo).
EPOCHS	15	Số lần lặp huấn luyện tối đa.
VALID_SPLIT	0.1	Tỷ lệ dữ liệu dùng cho tập kiểm tra/xác thực (10%).

2. Khai Báo Thư Viện và Cấu Hình

Đoạn code này nhập các thư viện cần thiết như tensorflow/keras cho mô hình, pandas cho xử lý dữ liệu, sklearn cho tiền xử lý và đánh giá, và matplotlib để vẽ biểu đồ.

```

# Nguyễn Việt Quang B22DCCN650
import os
import json
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
import matplotlib.pyplot as plt

# ----- CONFIG -----
# Nguyễn Việt Quang B22DCCN650
CSV_PATH = "DATA/stock.csv"
OUTPUT_DIR = "models_and_results"
SEQ_LEN = 60          # số ngày lịch sử dùng làm input
HORIZON = 7           # dự báo 7 ngày tiếp theo
BATCH_SIZE = 64
EPOCHS = 15
VALID_SPLIT = 0.1
RANDOM_SEED = 42
# -----

```

```

# ----- CONFIG -----
# Nguyễn Việt Quang B22DCCN650
CSV_PATH = "DATA/stock.csv"
OUTPUT_DIR = "models_and_results"
SEQ_LEN = 60          # số ngày lịch sử dùng làm input
HORIZON = 7           # dự báo 7 ngày tiếp theo
BATCH_SIZE = 64
EPOCHS = 15
VALID_SPLIT = 0.1
RANDOM_SEED = 42
# -----
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
os.makedirs(OUTPUT_DIR, exist_ok=True)

```

II. Các Hàm Tiện Ích (Utility Functions)

Phần này định nghĩa các hàm cơ bản để tải dữ liệu, tiền xử lý (chia tỷ lệ), tạo chuỗi đầu vào/đầu ra, và vẽ lịch sử huấn luyện.

1. Tải và Tiền Xử Dữ Liệu (load_data, scale_array)

```
# ----- util functions -----
# Nguyễn Việt Quang B22DCCN650
def load_data(path):
    df = pd.read_csv(path, parse_dates=[ "date"])
    df.sort_values(["Name", "date"], inplace=True)
    return df

def scale_array(arr):
    scaler = MinMaxScaler(feature_range=(0,1))
    scaled = scaler.fit_transform(arr.reshape(-1,1)).flatten()
    return scaled, scaler
```

2. Tạo Chuỗi Dữ Liệu (create_sequences)

Hàm này chuẩn bị dữ liệu cho mô hình học sâu, tạo ra các cặp X (chuỗi lịch sử dài SEQ_LEN) và y (chuỗi mục tiêu). Lưu ý: Trong hàm run_pipeline, y được cố định là dự báo 1 bước (horizon=1) để sử dụng cho chiến lược dự báo lặp.

```
def create_sequences(values, seq_len, horizon=1):
    """Tạo sequences; mặc định tạo label 1-step (dùng iterative để dự báo multi-step)."""
    X, y = [], []
    n = len(values)
    for i in range(n - seq_len - horizon + 1):
        X.append(values[i:i+seq_len])
        y.append(values[i+seq_len:i+seq_len+horizon])
    X = np.array(X).reshape(-1, seq_len, 1)
    y = np.array(y)
    return X, y
```

3. Vẽ Lịch Sử Huấn Luyện (plot_history)

Hàm này vẽ biểu đồ Loss (MSE) và MAE cho cả tập huấn luyện (train) và tập xác thực (val) qua các epoch, lưu vào thư mục OUTPUT_DIR.

```

# Nguyễn Việt Quang B22DCCN650
def plot_history(history, prefix, outdir):
    # Loss
    plt.figure(figsize=(8,4))
    plt.plot(history.history["loss"], label="train_loss")
    plt.plot(history.history["val_loss"], label="val_loss")
    plt.title(f"{prefix} - Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, f"{prefix}_loss.png"))
    plt.close()

    # MAE
    plt.figure(figsize=(8,4))
    if "mae" in history.history:
        plt.plot(history.history["mae"], label="train_mae")
    if "val_mae" in history.history:
        plt.plot(history.history["val_mae"], label="val_mae")
    plt.title(f"{prefix} - MAE")
    plt.xlabel("Epoch")
    plt.ylabel("MAE")
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, f"{prefix}_mae.png"))
    plt.close()

```

III. Xây Dựng Mô Hình Sâu (Deep Model Builders)

Cả hai mô hình SimpleRNN và LSTM đều được xây dựng với 7 tầng hồi quy (return_sequences=True cho 6 tầng đầu, return_sequences=False cho tầng cuối), cùng với các kỹ thuật điều chỉnh: BatchNormalization và Dropout (0.15).

1. Deep SimpleRNN (*build_deep_rnn*)

```
# Nguyễn Việt Quang B22DCCN650
def build_deep_lstm(input_shape, horizon):
    units = [128, 128, 64, 64, 32, 32, 16]
    model = Sequential()
    for i, u in enumerate(units):
        if i == 0:
            model.add(LSTM(u, return_sequences=True, input_shape=input_shape))
        elif i < len(units)-1:
            model.add(LSTM(u, return_sequences=True))
        else:
            model.add(LSTM(u, return_sequences=False))
        model.add(BatchNormalization())
        model.add(Dropout(0.15))
    model.add(Dense(horizon))
    model.compile(optimizer="adam", loss="mse", metrics=["mae"])
    return model
```

2. Deep LSTM (*build_deep_lstm*)

```
# Nguyễn Việt Quang B22DCCN650
def build_deep_lstm(input_shape, horizon):
    units = [128, 128, 64, 64, 32, 32, 16]
    model = Sequential()
    for i, u in enumerate(units):
        if i == 0:
            model.add(LSTM(u, return_sequences=True, input_shape=input_shape))
        elif i < len(units)-1:
            model.add(LSTM(u, return_sequences=True))
        else:
            model.add(LSTM(u, return_sequences=False))
        model.add(BatchNormalization())
        model.add(Dropout(0.15))
    model.add(Dense(horizon))
    model.compile(optimizer="adam", loss="mse", metrics=["mae"])
    return model
```

IV. Dự Báo Lặp và Huấn Luyện

1. Hàm Dự Báo Lặp (*iterative_forecast*)

Chức năng cốt lõi để dự báo chuỗi thời gian nhiều bước (steps=7). Nó sử dụng mô hình 1-step, sau đó lấy dự báo đó làm input cho bước tiếp theo (giống như chiến lược Recursive)

Forecasting).

```
# Nguyễn Việt Quang B22DCCN650
# hàm này có chức năng là dự báo nhiều bước (multi-step) bằng cách sử dụng dự báo từng bước (1-step) lặp đi lặp lại
def iterative_forecast(model, last_sequence, steps, scaler):
    """
    last_sequence: 1D array scaled length SEQ_LEN
    steps: number of iterative predicted steps
    returns: unscaled predictions (1D array length steps)
    """

    seq = list(last_sequence)
    preds_scaled = []
    for _ in range(steps):
        x = np.array(seq[-SEQ_LEN:]).reshape(1, SEQ_LEN, 1)
        p = model.predict(x, verbose=0)[0][0] # assume model trained 1-step output
        preds_scaled.append(p)
        seq.append(p)
    preds_scaled = np.array(preds_scaled).reshape(-1,1)
    preds = scaler.inverse_transform(preds_scaled).flatten()
    return preds
```

2. Hàm Huấn Luyện và Lưu Kết Quả (train_and_save)

Hàm này thực hiện huấn luyện, sử dụng EarlyStopping (chờ 10 epoch) và ModelCheckpoint để lưu mô hình có val_loss tốt nhất.

```
# ----- training + evaluation -----
# Nguyễn Việt Quang B22DCCN650
def train_and_save(X, y, builder, prefix):
    n = X.shape[0]
    val_n = max(1, int(n * VALID_SPLIT))
    train_X, val_X = X[:-val_n], X[-val_n:]
    train_y, val_y = y[:-val_n], y[-val_n:]

    model = builder((SEQ_LEN,1), train_y.shape[1])
    ckpt_path = os.path.join(OUTPUT_DIR, f"{prefix}_best.h5")
    callbacks = [
        EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True, verbose=1),
        ModelCheckpoint(ckpt_path, monitor="val_loss", save_best_only=True, verbose=1)
    ]
    history = model.fit(
        train_X, train_y,
        validation_data=(val_X, val_y),
        epochs=EPOCHS,
        batch_size=BATCH_SIZE,
        callbacks=callbacks,
        verbose=2
    )
    # Save history plots
    plot_history(history, prefix, OUTPUT_DIR)
```

```

# Nguyễn Việt Quang B22DCCN650
# Evaluate on validation (use first step since y shape is (n,1) here)
val_pred = model.predict(val_X)
mse = mean_squared_error(val_y[:,0], val_pred[:,0])
mae = mean_absolute_error(val_y[:,0], val_pred[:,0])
metrics = {"mse": float(mse), "mae": float(mae), "val_samples": len(val_X)}
# return model and metrics
return model, history, metrics

```

V. Luồng Xử Lý Chính (run_pipeline)

Hàm này điều phối toàn bộ quy trình:

1. Tải và xử lý từng mã cổ phiếu (sym) riêng biệt.
2. Chia tỷ lệ (scale_array) và tạo chuỗi dữ liệu 1-step (create_sequences).
3. Huấn luyện và đánh giá Deep SimpleRNN và Deep LSTM.
4. So sánh kết quả trên tập xác thực (val_mse) và chọn mô hình tốt nhất.
5. Sử dụng mô hình tốt nhất để thực hiện dự báo 7 ngày lặp đi lặp lại (iterative_forecast).
6. Lưu kết quả dự báo và các chỉ số vào file JSON và vẽ biểu đồ kết quả (.png).

```

# ----- main pipeline -----
# Nguyễn Việt Quang B22DCCN650
def run_pipeline():
    df = load_data(CSV_PATH)
    symbols = df["Name"].unique()
    print("Found symbols:", symbols)
    summary = {}

    for sym in symbols:
        print("\n====")
        print("Processing:", sym)
        print("====")
        sub = df[df["Name"] == sym].sort_values("date")
        prices = sub["close"].values.astype(float)
        dates = sub["date"].values

        # scale
        scaled, scaler = scale_array(prices)

        # create sequences for 1-step training (we use iterative forecasting for HORIZON steps)
        X, y = create_sequences(scaled, SEQ_LEN, horizon=1) # y shape (n,1)

```

```

# Nguyễn Việt Quang B22DCCN650
# sanity check: enough samples
if X.shape[0] < 10:
    print(f"Not enough samples for {sym} (need >=10 sequences). Skipping.")
    continue

# Train RNN
rnn_prefix = f"{sym}_RNN"
rnn_model, rnn_hist, rnn_metrics = train_and_save(X, y, build_deep_rnn, rnn_prefix)
print(f"RNN metrics: {rnn_metrics}")

# Train LSTM
lstm_prefix = f"{sym}_LSTM"
lstm_model, lstm_hist, lstm_metrics = train_and_save(X, y, build_deep_lstm, lstm_prefix)
print(f"LSTM metrics: {lstm_metrics}")

# choose best by val MSE
chosen = "RNN" if rnn_metrics["mse"] <= lstm_metrics["mse"] else "LSTM"
chosen_model = rnn_model if chosen == "RNN" else lstm_model
chosen_metrics = rnn_metrics if chosen == "RNN" else lstm_metrics
print(f"Chosen model for {sym}: {chosen} (val_mse={chosen_metrics['mse']:.6f})")

# Forecast HORIZON days from last sequence
last_seq = scaled[-SEQ_LEN:]
preds = iterative_forecast(chosen_model, last_seq, HORIZON, scaler)

```

```

# Nguyễn Việt Quang B22DCCN650
# Save results
out = {
    "symbol": sym,
    "last_date": str(dates[-1]),
    "horizon_days": HORIZON,
    "chosen_model": chosen,
    "chosen_metrics": chosen_metrics,
    "predictions": preds.tolist()
}
out_path = os.path.join(OUTPUT_DIR, f"{sym}_forecast_7day.json")
with open(out_path, "w") as f:
    json.dump(out, f, indent=2)
summary[sym] = out

# Optional: save simple plot of recent actual + 7-day forecast
try:
    recent_n = min(200, len(prices))
    recent_prices = prices[-recent_n:]
    recent_dates = pd.to_datetime(dates[-recent_n:])
    plt.figure(figsize=(10,4))
    plt.plot(recent_dates, recent_prices, label="actual_close")

```

```

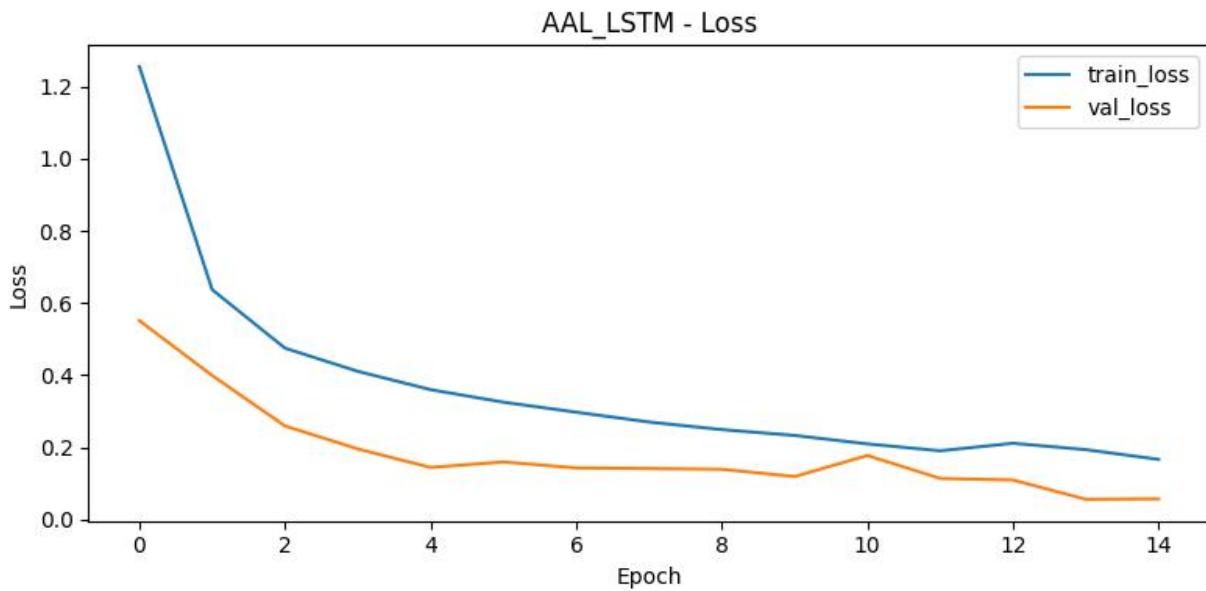
# Nguyen Viet Quang B22DCCN650
# forecast dates: next calendar days (not trading-day aware)
last_date = pd.to_datetime(dates[-1])
fut_dates = [last_date + np.timedelta64(i+1, 'D') for i in range(HORIZON)]
plt.plot(fut_dates, preds, marker='o', linestyle='--', label=f"{HORIZON}-day forecast")
plt.title(f"{sym} recent close + {HORIZON}-day forecast")
plt.xlabel("date")
plt.ylabel("close")
plt.legend()
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, f"{sym}_recent_plus_7day.png"))
plt.close()
except Exception as e:
    print("Plot save skipped:", e)

# write summary
with open(os.path.join(OUTPUT_DIR, "summary_7day.json"), "w") as f:
    json.dump(summary, f, indent=2)
print("All done. Results saved to", OUTPUT_DIR)

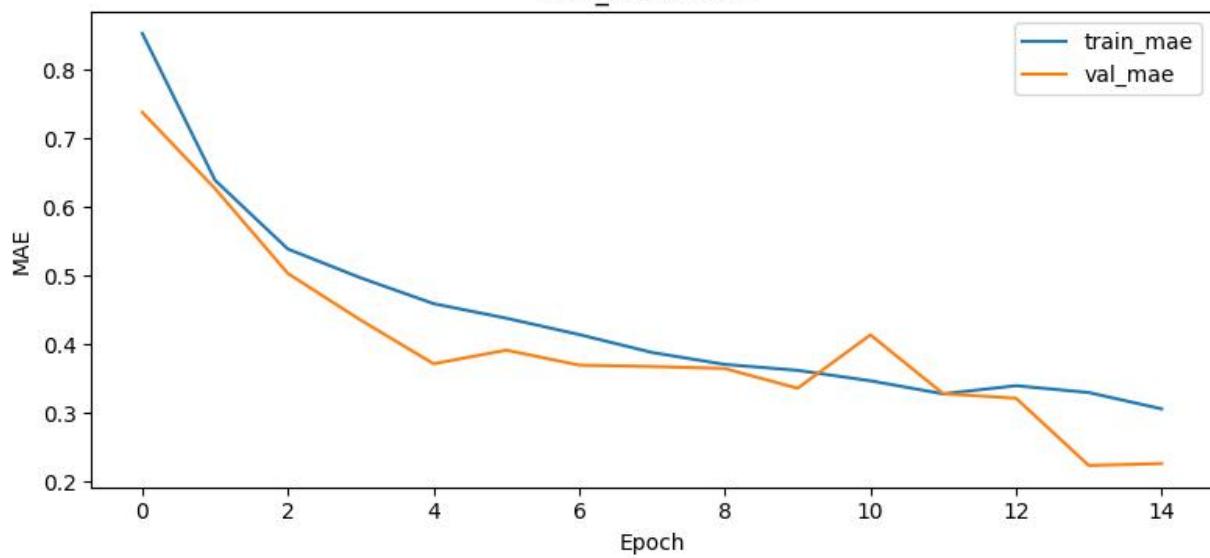
if __name__ == "__main__":
    run_pipeline()

```

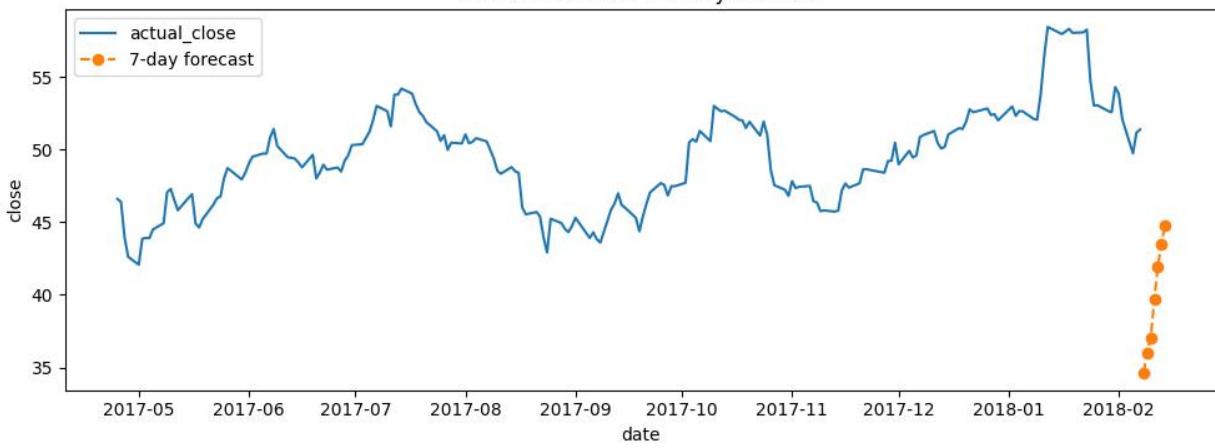
VI. Kết quả



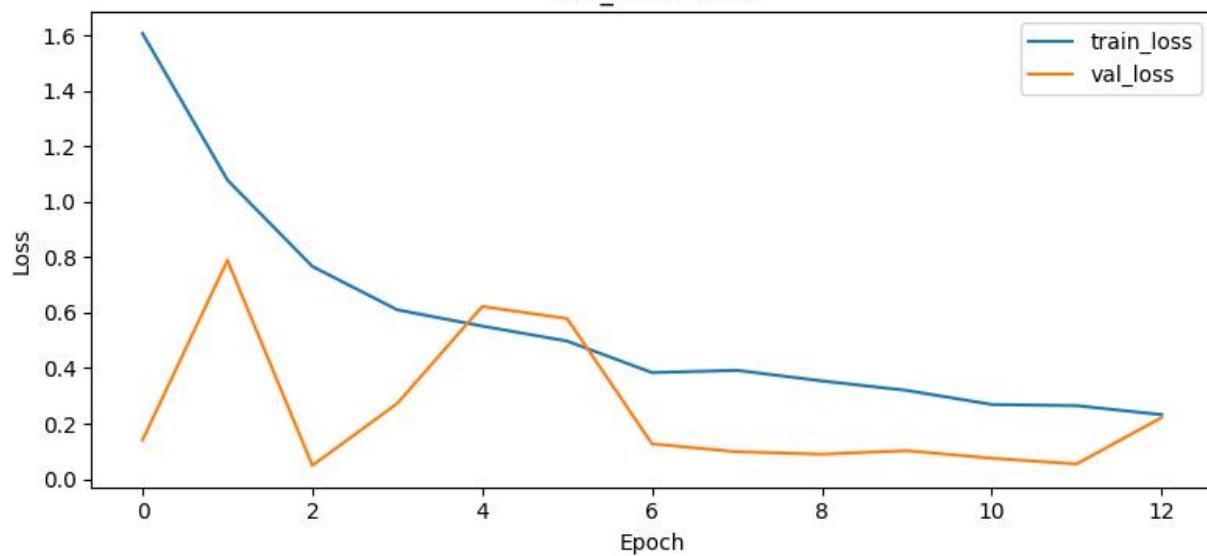
AAL_LSTM - MAE



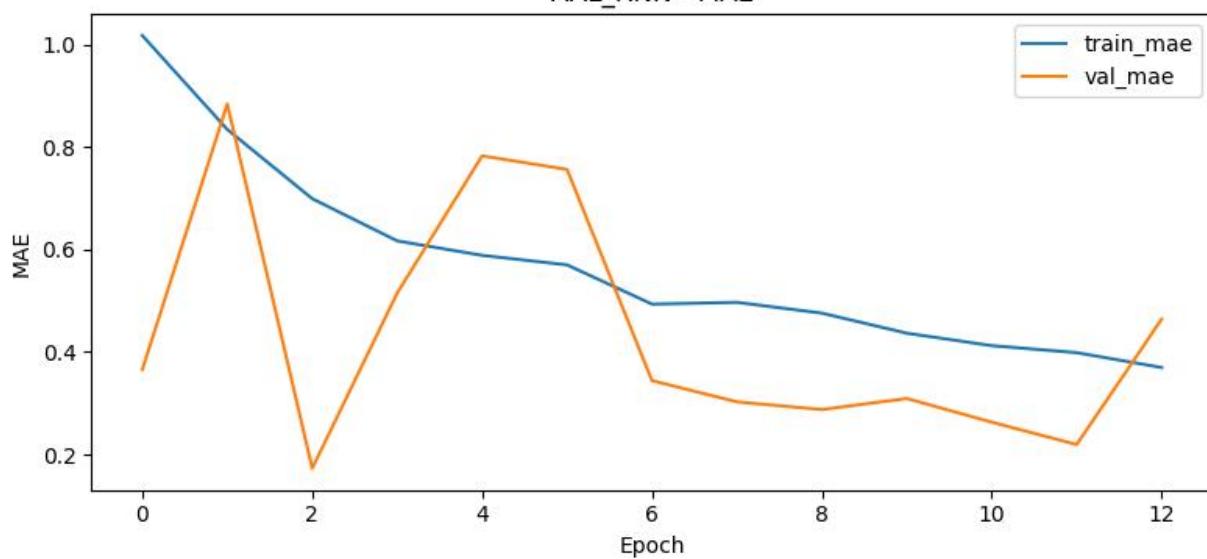
AAL recent close + 7-day forecast



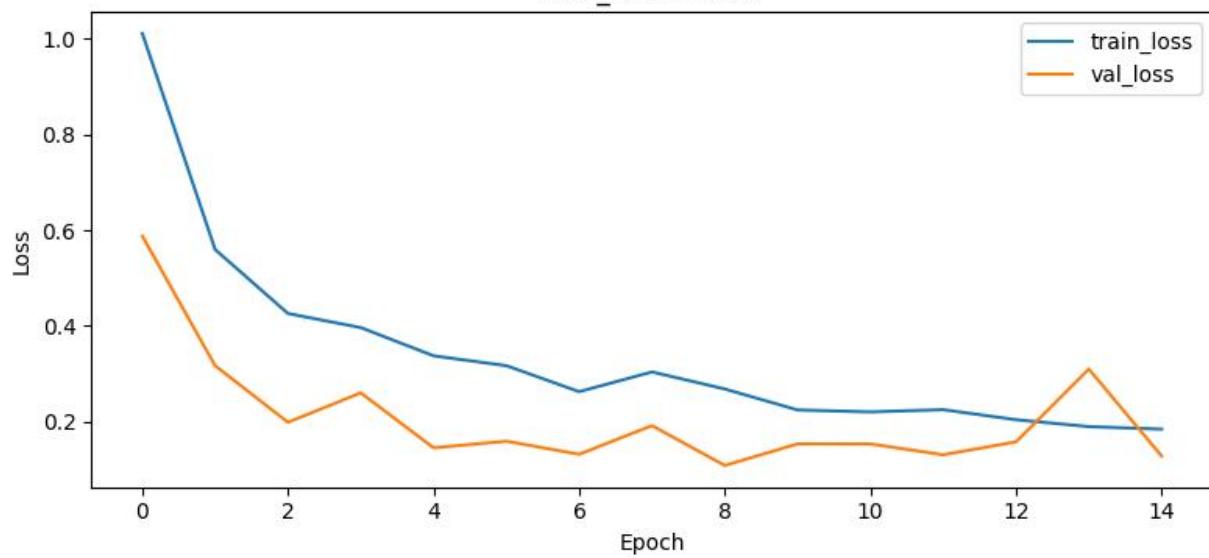
AAL_RNN - Loss



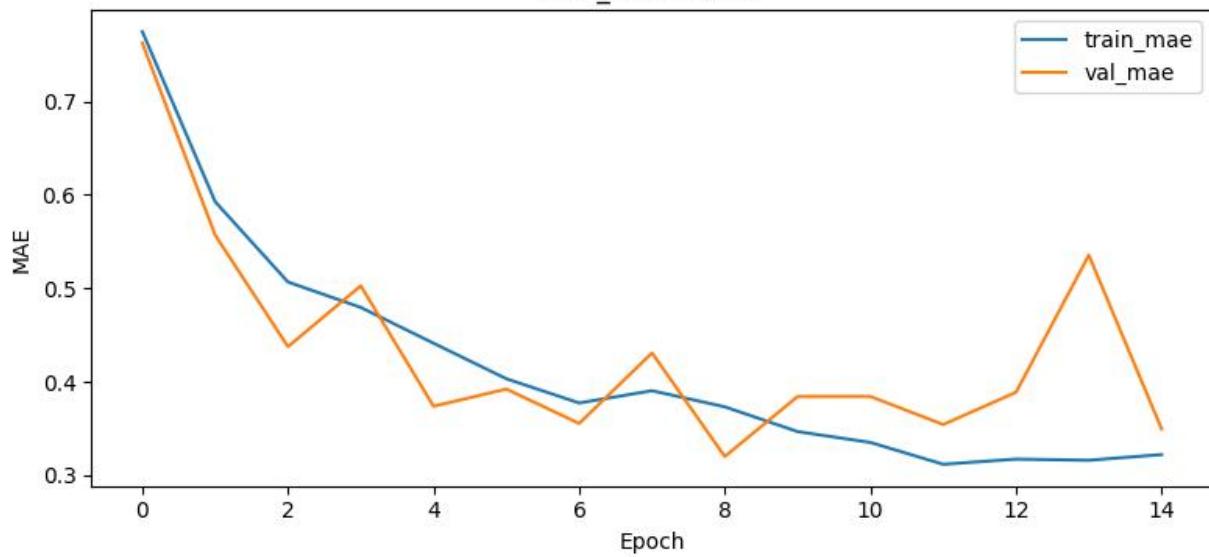
AAL_RNN - MAE

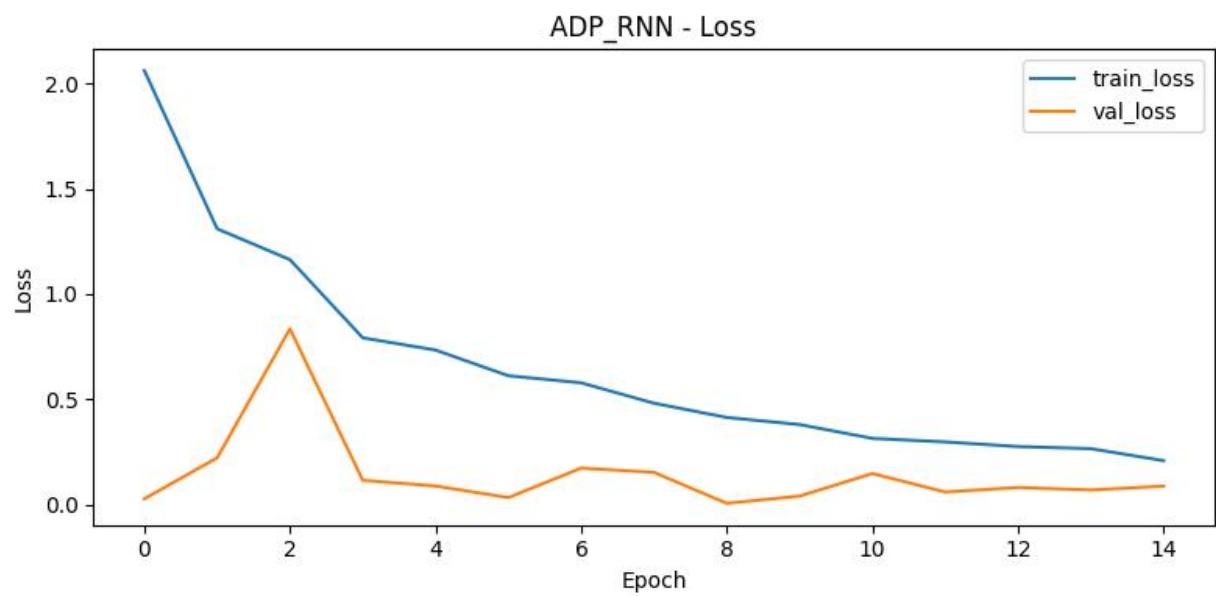
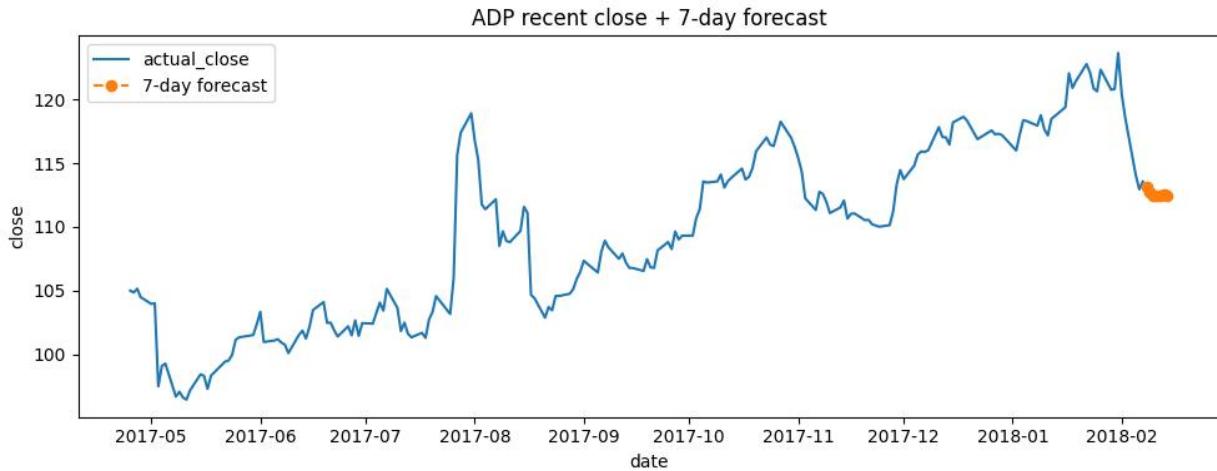


ADP_LSTM - Loss

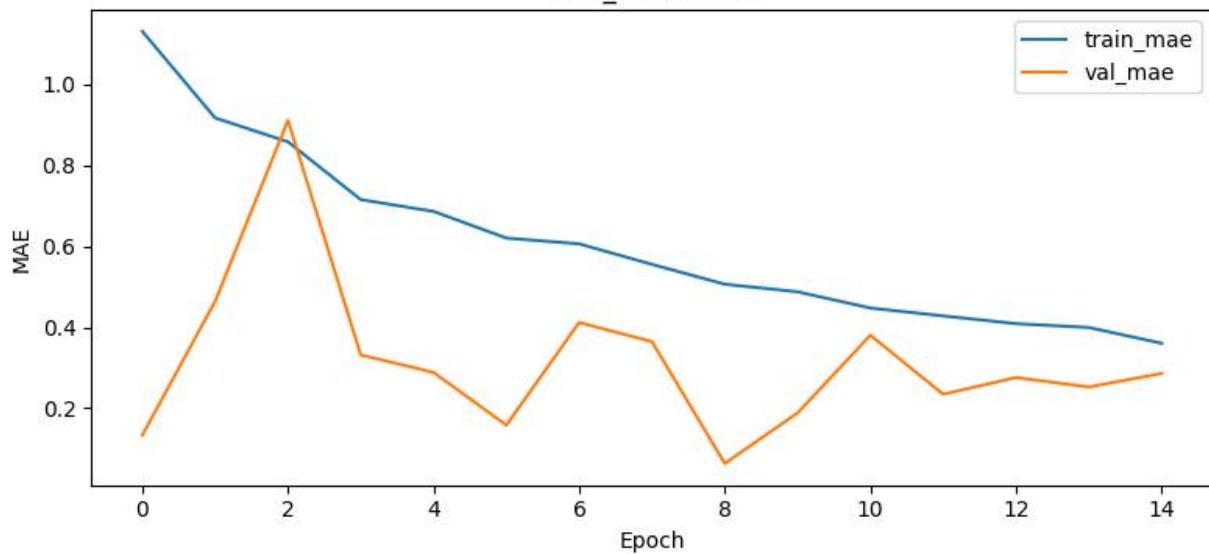


ADP_LSTM - MAE

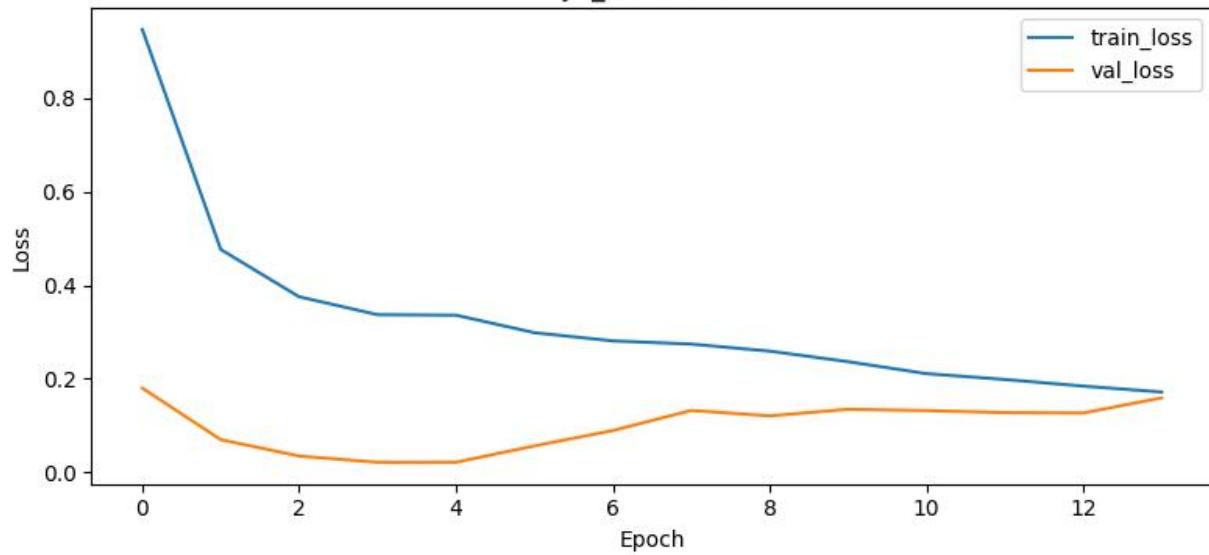




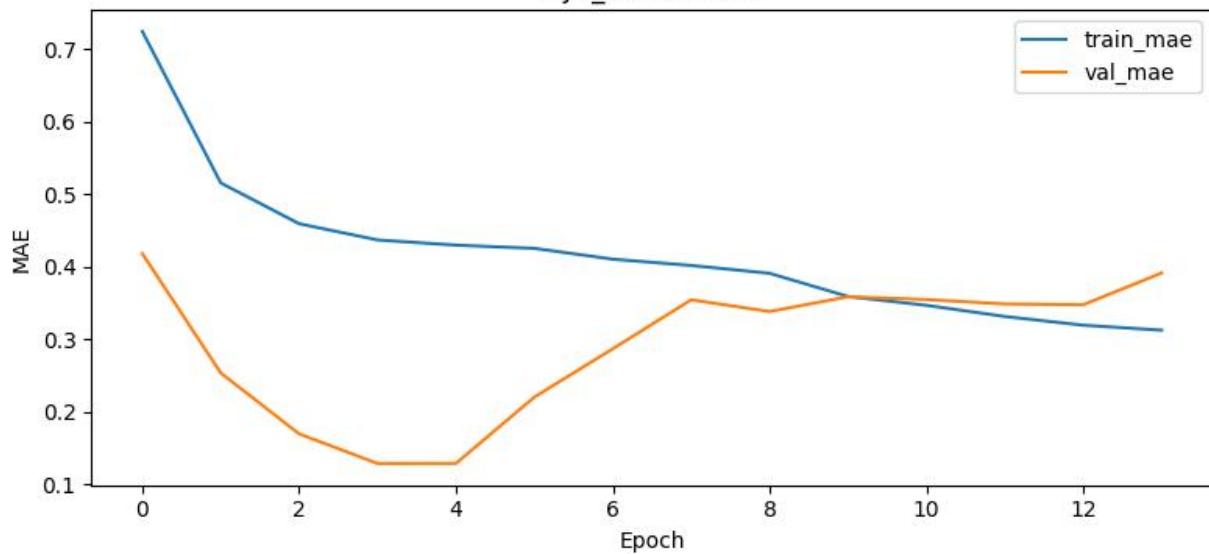
ADP_RNN - MAE



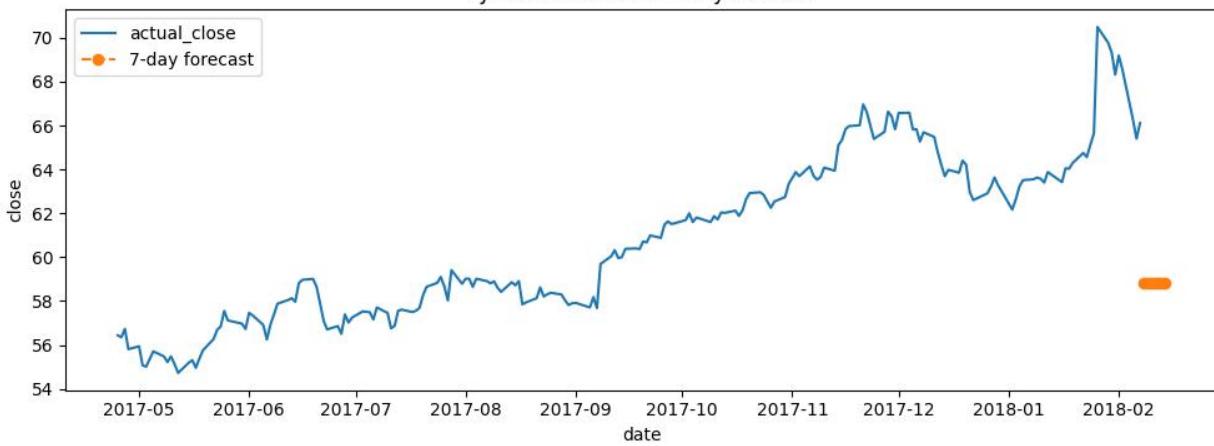
AJG_LSTM - Loss

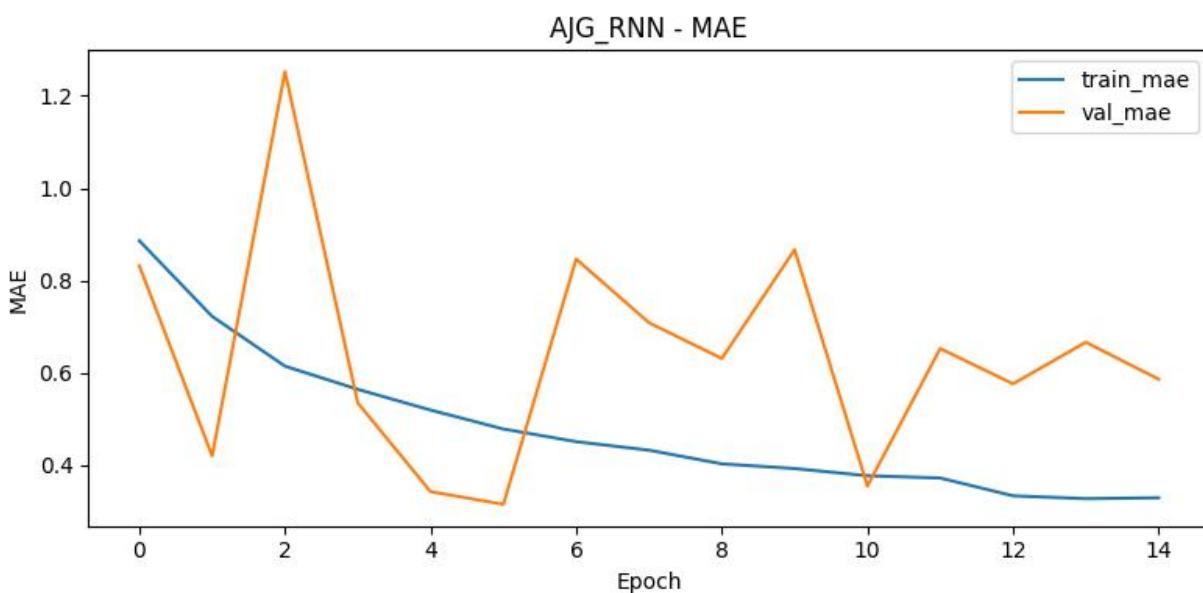
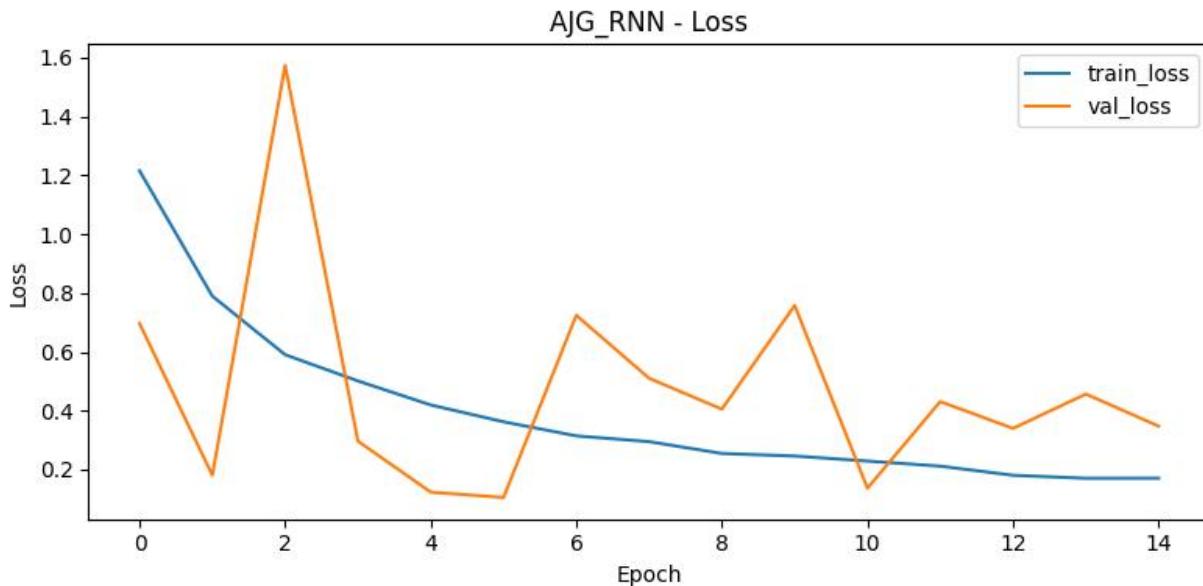


AJG_LSTM - MAE



AJG recent close + 7-day forecast





Đánh giá mô hình

Trong quá trình huấn luyện hai mô hình RNN và LSTM trên tập dữ liệu của ba công ty AAL, ADP và AJG, kết quả cho thấy phần lớn các mô hình đều hội tụ ổn định và đạt độ lỗi (loss) giảm dần theo từng epoch. Tuy nhiên, riêng mô hình **RNN huấn luyện trên dữ liệu của công ty AJG xuất hiện hiện tượng overfitting rõ rệt**.

Cụ thể, đường *training loss* của RNN giảm mạnh và duy trì ở mức thấp, trong khi *validation loss* có xu hướng dao động và thậm chí tăng trở lại sau một số epoch. Điều này chứng tỏ mô hình đã học quá kỹ các mẫu trong tập huấn luyện nhưng không tổng quát tốt khi đánh giá trên dữ liệu chưa thấy. Nguyên nhân có thể đến từ các yếu tố:

- **Độ nhiễu trong dữ liệu giá của AJG** cao hơn so với hai mã còn lại, khiến mô hình RNN dễ bị “học thuộc” các biến động bất thường.
- **Cấu trúc 7 tầng recurrent của RNN** (SimpleRNN) có xu hướng kém ổn định hơn LSTM khi xử lý chuỗi dài, dẫn tới giảm khả năng khai quát.
- **Lượng dữ liệu thực tế của AJG có thể ít hơn** hoặc phân bố khác biệt, khiến mô hình khó học quy luật chung.

Trong khi đó, mô hình **LSTM không gặp vấn đề này**, thể hiện validation loss ổn định hơn và độ sai lệch giữa train-val nhỏ hơn, cho thấy LSTM phù hợp hơn với đặc thù biến động của mã AJG cũng như có khả năng mô hình hóa tốt hơn các mối quan hệ dài hạn trong chuỗi thời gian.

Các giải pháp ngắn gọn để khắc phục overfitting:

- **Tăng Dropout** ở các tầng RNN/LSTM.
- **Giảm số tầng hoặc số units** để mô hình bớt phức tạp.
- **Early stopping chặt hơn** (patience nhỏ hơn).
- **Tăng dữ liệu huấn luyện** hoặc dùng kỹ thuật sinh thêm dữ liệu.
- **Regularization** như L2 để hạn chế mô hình học quá sâu.
- **Dùng LSTM/GRU thay SimpleRNN**, ổn định hơn khi chuỗi dài.