

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO CUỐI KỲ
NHẬP MÔN HỌC MÁY

Người hướng dẫn: TS. TRẦN LƯƠNG QUỐC ĐẠI

Người thực hiện: PHẠM TUẤN ĐẠT – 52200207

TRẦN HỒ HOÀNG VŨ – 52200214

TRẦN KHIẾT LÔI - 52200216

Lớp : 22050301

Khoa : 26

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO CUỐI KỲ
NHẬP MÔN HỌC MÁY

Người hướng dẫn: TS. TRẦN LƯƠNG QUỐC ĐẠI

Người thực hiện: PHẠM TUẤN ĐẠT - 52200207

TRẦN HỒ HOÀNG VŨ – 52200214

TRẦN KHIẾT LÔI - 52200216

Lớp : 22050301

Khoa : 26

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Đầu tiên, chúng tôi xin gửi lời cảm ơn chân thành và tri ân sâu sắc đến **Ban giám hiệu Trường Đại học Tôn Đức Thắng** và **khoa Công nghệ thông tin** vì đã tạo điều kiện tối ưu về cơ sở vật chất và hệ thống thư viện hiện đại, đa dạng các loại tài liệu, giúp chúng tôi có thể dễ dàng và thuận tiện hơn trong việc tìm kiếm, nghiên cứu thông tin để hoàn thành bài báo cáo này.

Chúng tôi xin chân thành cảm ơn TS. **Trần Lương Quốc Đại**, giảng viên bộ môn, đã truyền đạt kiến thức một cách tận tình, chi tiết, định hướng tư duy, giúp chúng tôi có đầy đủ nền tảng để vận dụng tốt hơn vào việc thực hiện bài báo cáo này.

Mặc dù đã có gắng vận dụng những kiến thức đã học được để hoàn thành bài báo cáo. Nhưng do kiến thức của chúng tôi còn hạn chế và không có nhiều kinh nghiệm thực tiễn nên khó tránh khỏi những thiếu sót trong quá trình nghiên cứu và trình bày. Do đó, chúng tôi rất mong nhận được sự góp ý của quý thầy cô để bài báo cáo của chúng tôi được hoàn thiện hơn.

Chúng tôi xin chân thành cảm ơn!

BÁO CÁO ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng tôi xin cam đoan đây là sản phẩm báo cáo của riêng chúng tôi và được sự hướng dẫn của TS Trần Lương Quốc Đại;. Các nội dung nghiên cứu, kết quả trong báo cáo này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong báo cáo còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung báo cáo của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do chúng tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 19 tháng 12 năm 2024

Tác giả

(ký tên và ghi rõ họ tên)



Phạm Tuấn Đạt



Trần Khiết Lôi



Trần Hồ Hoàng Vũ

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Báo cáo yêu cầu ba nhiệm vụ chính:

- Thứ nhất, trình bày lý thuyết về các phương pháp tối ưu hóa bao gồm Gradient Descent (Batch, Stochastic, Mini-Batch), Momentum, Adagrad, RMSProp, và Adam. Sau đó, triển khai các phương pháp này trên một tập dữ liệu tùy chọn, so sánh kết quả và đánh giá hiệu suất.
- Thứ hai, giải bài toán dự đoán giá cổ phiếu (giá mở cửa) dựa trên các thông tin như giá mở cửa, giá đóng cửa, khối lượng giao dịch, loại ngành, thời gian trong năm, và các chỉ số kinh tế vĩ mô. Thử nghiệm với các mô hình như Feedforward Neural Network (FNN), Recurrent Neural Network (RNN), và các thuật toán khác ở đây nhóm em chọn 2 thuật toán là (Linear Regression và Decision Tree). Thực hiện các kỹ thuật giảm overfitting (Dropout, Regularization, Early Stopping), sau đó đánh giá và so sánh các mô hình bằng cách sử dụng các tiêu chí như MSE và R^2 , đồng thời minh họa bằng đồ thị.
- Thứ ba, nghiên cứu và áp dụng phương pháp học sâu CNN vào bài toán phân loại MNIST, và trình bày lý thuyết, triển khai chương trình và đánh giá kết quả.

MỤC LỤC

LỜI CẢM ƠN	i
BÁO CÁO ĐƯỢC HOÀN THÀNH.....	ii
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN	iii
TÓM TẮT	iv
MỤC LỤC.....	1
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THI	4
PHẦN 1 – CÂU 1	7
1.1 Lý thuyết:	7
1.1.1 Gradient Decent (GD):	7
1.1.2 Momentum:	9
1.1.3 Adagrad (Adaptive Gradient Algorithm):	10
1.1.4 RMSProp (Root Mean Square Propagation):	11
1.1.5 Adam (Adaptive Moment Estimation):	12
1.2 Phương pháp (Triển khai thực nghiệm và phân tích chương trình):.....	13
1.2.1 Xử lí dữ liệu:.....	13
1.2.2 Hàm Random seed:.....	15
1.2.3 Hàm tính toán cơ bản:	15
1.3 So sánh kết quả:	24
PHẦN 2 – CÂU 2	28
2.1 Chuẩn bị dữ liệu:	28
2.2 Tiền xử lý dữ liệu:.....	33
2.3 Thử nghiệm các phương pháp học máy:	34
2.3.1 Feedforward Neural Network:.....	34
2.3.2 Recurrent Neural Network (RNN):	43
2.3.3 Linear Regression:	55
2.3.4 Decision Tree:.....	64

2.4	Áp dụng các kỹ thuật để giảm hiện tượng quá khóp (Overfitting):.....	74
2.4.1	Feedforward Neural Network:.....	74
2.4.2	Recurrent Neural Network (RNN):	84
2.4.3	Linear Regression:	93
2.4.4	Decision Tree:.....	103
2.5	Đánh giá và so sánh các mô hình:.....	113
2.5.1	So sánh mô hình trước khi thực hiện giảm overfitting:.....	113
2.5.2	So sánh các mô hình sau khi thực hiện các phương pháp giảm overfitting:.....	118
2.5.3	Đánh giá các mô hình:.....	123
	PHẦN 3 – CÂU 3	131
3.1	CNN là gì?	131
3.2	Cấu trúc cơ bản của mạng CNN:	131
3.2.1	Convolutional Layer (Lớp tích chập):	131
3.2.2	Activation Function (Hàm kích hoạt):.....	132
3.2.3	Pooling Layer (Lớp pooling):.....	132
3.2.4	Fully Connected Layer (Lớp kết nối đầy đủ):	132
3.3	Nguyên lý hoạt động của CNN:.....	133
3.3.1	Giai đoạn trích xuất đặc trưng (Feature Extraction):.....	133
3.3.2	Giai đoạn giảm kích thước (Dimensionality Reduction):	133
3.3.3	Giai đoạn phân loại (Classification):.....	134
3.4	Quá trình huấn luyện của CNN:.....	134
3.4.1	Truyền tiến (Forward Propagation):	134
3.4.2	Tính toán lỗi (Loss Calculation):.....	134
3.4.3	Lan truyền ngược (Backward Propagation):	136
3.5	Ưu điểm và nhược điểm của CNN:	139
3.5.1	Ưu điểm:	139

3.5.2	Nhược điểm:	139
3.6	Các kỹ thuật cải tiến CNN:	139
3.7	Các ứng dụng phổ biến của CNN:	140
3.7.1	Xử lý ảnh:	140
3.7.2	Xử lý video:	140
3.7.3	Xử lý ngôn ngữ tự nhiên (khi kết hợp với mạng RNN):	140
3.7.4	Các lĩnh vực khác:	140
3.8	Các kiến trúc CNN nổi bật:.....	140
3.9	Chương trình áp dụng mô hình CNN vào bài toán phân loại:	141
3.9.1	Tổng quan về tập dữ liệu MNIST:	141
3.9.2	Quy trình thực hiện:.....	141
	TÀI LIỆU THAM KHẢO.....	153

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

Hình 1. Gradient Deccent.....	7
Hình 2. So sánh kết quả giữa các thuật toán	25
Hình 3. Code tải data cổ phiếu từ Yahoo Finance	28
Hình 4. Code tải data GDP từ FredAPI.....	29
Hình 5. Code chỉnh sửa định dạng data cổ phiếu.....	29
Hình 6. Code thêm cột Industry và Ticker.....	30
Hình 7. Code gộp các bộ data nhỏ thành một bộ data tổng hợp	31
Hình 8. Code gộp GDP vào bộ data tổng hợp.....	32
Hình 9. Code lọc và lưu bộ dữ liệu cuối cùng	33
Hình 10. Code chuẩn hóa và chia dữ liệu	33
Hình 11. Biểu đồ huấn luyện bằng mô hình FNN	39
Hình 12. Kết quả dự đoán bằng mô hình FNN cho ticker AAPL.....	41
Hình 13. Kết quả dự đoán bằng mô hình FNN cho ticker TSLA	41
Hình 14. Kết quả dự đoán bằng mô hình FNN cho ticker F	41
Hình 15. Kết quả dự đoán bằng mô hình FNN cho ticker EA	42
Hình 16. Kết quả dự đoán bằng mô hình FNN cho ticker MSFT	42
Hình 17. Kết quả dự đoán bằng mô hình FNN cho ticker TTWO.....	42
Hình 18. Kết quả các chỉ số MSE, R2	43
Hình 19. Biểu đồ huấn luyện bằng mô hình RNN	51
Hình 20. Kết quả dự đoán bằng mô hình RNN cho ticker AAPL	53
Hình 21. Kết quả dự đoán bằng mô hình RNN cho ticker TSLA.....	53
Hình 22. Kết quả dự đoán bằng mô hình RNN cho ticker F.....	53
Hình 23. Kết quả dự đoán bằng mô hình RNN cho ticker EA	54
Hình 24. Kết quả dự đoán bằng mô hình RNN cho ticker MSFT	54
Hình 25. Kết quả dự đoán bằng mô hình RNN cho ticker TTWO	54

Hình 26. Kết quả các chỉ số MSE, R2	55
Hình 27. Biểu đồ huấn luyện bằng mô hình LR	60
Hình 28. Kết quả dự đoán bằng mô hình LR cho ticker AAPL.....	62
Hình 29. Kết quả dự đoán bằng mô hình LR cho ticker TSLA	62
Hình 30. Kết quả dự đoán bằng mô hình LR cho ticker F	62
Hình 31. Kết quả dự đoán bằng mô hình LR cho ticker EA.....	63
Hình 32. Kết quả dự đoán bằng mô hình LR cho ticker MSFT.....	63
Hình 33. Kết quả dự đoán bằng mô hình LR cho ticker TTWO.....	63
Hình 34. Kết quả các chỉ số MSE, R2	64
Hình 35. Biểu đồ huấn luyện bằng mô hình DT	70
Hình 36. Kết quả dự đoán bằng mô hình DT cho ticker AAPL	72
Hình 37. Kết quả dự đoán bằng mô hình DT cho ticker TSLA	72
Hình 38. Kết quả dự đoán bằng mô hình DT cho ticker F.....	72
Hình 39. Kết quả dự đoán bằng mô hình DT cho ticker EA.....	73
Hình 40. Kết quả dự đoán bằng mô hình DT cho ticker MSFT	73
Hình 41. Kết quả dự đoán bằng mô hình DT cho ticker TTWO	73
Hình 42. Kết quả các chỉ số MSE, R2	74
Hình 43. Biểu đồ huấn luyện bằng mô hình FNN sau khi overfitting	79
Hình 44. Kết quả dự đoán bằng mô hình FNN cho ticker AAPL sau khi overfitting ..	82
Hình 45. Kết quả dự đoán bằng mô hình FNN cho ticker TSLA sau khi overfitting...	82
Hình 46. Kết quả dự đoán bằng mô hình FNN cho ticker F sau khi overfitting.....	82
Hình 47. Kết quả dự đoán bằng mô hình FNN cho ticker EA sau khi overfitting ..	83
Hình 48. Kết quả dự đoán bằng mô hình FNN cho ticker MSFT sau khi overfitting ..	83
Hình 49. Kết quả dự đoán bằng mô hình FNN cho ticker TTWO sau khi overfitting .	83
Hình 50. Biểu đồ huấn luyện bằng mô hình RNN sau khi overfitting.....	88
Hình 51. Kết quả dự đoán bằng mô hình RNN cho ticker AAPL sau khi overfitting..	91
Hình 52. Kết quả dự đoán bằng mô hình RNN cho ticker TSLA sau khi overfitting ..	91

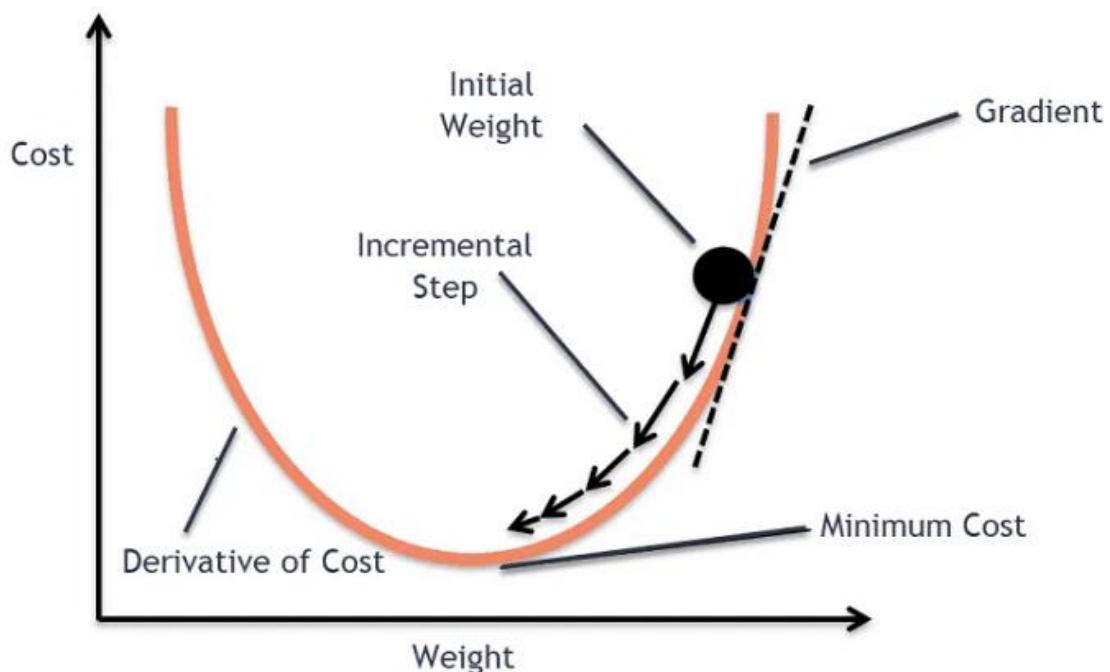
Hình 53. Kết quả dự đoán bằng mô hình RNN cho ticker F sau khi overfitting	91
Hình 54. Kết quả dự đoán bằng mô hình RNN cho ticker EA sau khi overfitting	92
Hình 55. Kết quả dự đoán bằng mô hình RNN cho ticker MSFT sau khi overfitting..	92
Hình 56. Kết quả dự đoán bằng mô hình RNN cho ticker TTWO sau khi overfitting.	92
Hình 57. Biểu đồ huấn luyện bằng mô hình LR sau khi overfitting	98
Hình 58. Kết quả dự đoán bằng mô hình LR cho ticker AAPL sau khi overfitting ...	101
Hình 59. Kết quả dự đoán bằng mô hình LR cho ticker TSLA sau khi overfitting....	101
Hình 60. Kết quả dự đoán bằng mô hình LR cho ticker F sau khi overfitting	101
Hình 61. Kết quả dự đoán bằng mô hình LR cho ticker EA sau khi overfitting	102
Hình 62. Kết quả dự đoán bằng mô hình LR cho ticker MSFT sau khi overfitting ...	102
Hình 63. Kết quả dự đoán bằng mô hình LR cho ticker TTWO sau khi overfitting..	102
Hình 64. Biểu đồ huấn luyện bằng mô hình DT sau khi overfitting.....	108
Hình 65. Kết quả dự đoán bằng mô hình DT cho ticker AAPL sau khi overfitting ...	111
Hình 66. Kết quả dự đoán bằng mô hình DT cho ticker TSLA sau khi overfitting ...	111
Hình 67. Kết quả dự đoán bằng mô hình DT cho ticker F sau khi overfitting	111
Hình 68. Kết quả dự đoán bằng mô hình DT cho ticker EA sau khi overfitting	112
Hình 69. Kết quả dự đoán bằng mô hình DT cho ticker MSFT sau khi overfitting ...	112
Hình 70. Kết quả dự đoán bằng mô hình DT cho ticker TTWO sau khi overfitting..	112
Hình 71. So sánh giá trị MSE giữa các mô hình.....	115
Hình 72. So sánh giá trị R2 giữa các mô hình	116
Hình 73. So sánh giá trị MSE giữa các mô hình sau khi overfitting	120
Hình 74. So sánh giá trị R2 giữa các mô hình sau khi overfitting	121
Hình 75. Đánh giá mô hình trước và sau khi thực hiện overfitting	127

PHẦN 1 – CÂU 1

1.1 Lý thuyết:

1.1.1 Gradient Decent (GD):

Gradient Descent là một thuật toán tối ưu hóa cơ bản được sử dụng để giảm thiểu hàm mất mát. Thuật toán cập nhật các tham số thông qua các bước lặp để tìm giá trị tối thiểu của hàm.



Hình 1. Gradient Decent

- a. Ý tưởng của gradient descent là tính đạo hàm và di chuyển theo hướng giảm dần của gradient:

Các thuật toán Gradient Descent, có nhiều biến thể của thuật toán này tùy thuộc vào việc lựa chọn dữ liệu huấn luyện và thứ tự cập nhật.

- b. Gradient Descent cho hàm 1 biến:

Xét hàm một biến $f(x)$. Thuật toán Gradient Descent tìm cực tiểu của hàm số bằng cách khởi tạo giá trị x tại một vị trí ngẫu nhiên, sau đó lần lượt di chuyển

x ngược hướng với đạo hàm của $f(x)$ là $f'(x)$. Thao tác này sẽ lặp đi lặp lại cho đến khi đạt được ngưỡng nào đó.

Công thức cập nhật x trong Gradient Descent:

$$x_{t+1} = x_t - \alpha * f'(x_t)$$

Trong đó:

- x_t là giá trị x tại bước thứ t
- α là learning rate (Tốc độ học)
- $f'(x_t)$ là đạo hàm của hàm f tại x_t

c. Gradient Descent cho hàm nhiều biến:

Đối với hàm nhiều biến $f(\vec{x})$, Gradient Descent sẽ tính véc tơ biểu diễn tốc độ thay đổi của hàm (vector đạo hàm hay Gradient) của hàm f tại một thời điểm ngẫu nhiên \vec{x} , sau đó di chuyển \vec{x} ngược hướng với Gradient này.

Công thức cập nhật \vec{x} :

$$\vec{x}_{t+1} = \vec{x}_t - \alpha * \nabla f(\vec{x}_t)$$

Trong đó:

- \vec{x}_t là giá trị \vec{x} tại bước thứ t
- α là learning rate (Tốc độ học)
- $\nabla f(\vec{x}_t)$ là Gradient của hàm f tại \vec{x}_t .

❖ Batch Gradient Descent:

- Cập nhật các tham số dựa trên gradient của toàn bộ tập dữ liệu.
- Ưu điểm: Hội tụ đến giá trị tối thiểu toàn cục với các hàm lồi.
- Nhược điểm: Xử lý chậm bởi vì phải chạy trên toàn bộ dữ liệu.

Công thức cập nhật:

$$\vec{w} = \vec{w} - \alpha \frac{1}{N} \sum_{i=1}^N \nabla L(\vec{w}, x_i, y_i)$$

Trong đó

- \vec{w} : Là vector tham số (các trọng số) của mô hình cần tối ưu hóa.
- α : Learning rate (tốc độ học), kiểm soát mức độ thay đổi của tham số trong mỗi bước cập nhật.
- N : Số lượng mẫu dữ liệu trong tập huấn luyện.
- $\nabla L(\vec{w}, x_i, y_i)$ Gradient của hàm mất mát L theo tham số \vec{w} tại điểm dữ liệu.

❖ Stochastic Gradient Descent (SGD):

- Cập nhật tham số dựa trên gradient của một điểm dữ liệu duy nhất.
- Ưu điểm: Nhanh hơn trên mỗi lần lặp; có thể thoát khỏi điểm cực tiểu cục.
- Nhược điểm: Cập nhật nhiều có thể gây khó khăn trong việc hội tụ.

Công thức cập nhật:

$$\vec{w} = \vec{w} - a \nabla L(\vec{w}, x_i, y_i)$$

❖ Mini – Batch Gradient Descent:

- Kết hợp Batch và SGD bằng cách sử dụng một tập con nhỏ (mini-batch) của dữ liệu.
- Ưu điểm: Hội tụ nhanh hơn và ổn định hơn.
- Nhược điểm: Cần điều chỉnh kích thước batch phù hợp.

Công thức cập nhật:

$$\vec{w} = \vec{w} - a \frac{1}{|B|} \sum_{i \in B}^N \nabla L(\vec{w}, x_i, y_i)$$

1.1.2 Momentum:

Momentum là một cải tiến của Gradient Descent nhằm tăng tốc độ hội tụ, đặc biệt trong các hàm mất mát có hình dạng "lõm sâu" hoặc "yên ngựa". Nó sử dụng thông tin về tốc độ thay đổi của các bước trước để cập nhật các tham số, như một "lực quán tính".

Ý tưởng chính: Thay vì chỉ dựa vào gradient hiện tại, Momentum kết hợp với gradient từ các bước trước đó bằng cách sử dụng một thuật toán lấy trung bình động theo thời gian.

Công thức:

- Tính vận tốc (velocity):

$$v_t = \beta \cdot v_{t-1} + \nabla L(w)$$

Trong đó :

- v_t : Vận tốc tại bước t.
- β : Hệ số động lượng (thường $\beta = 0.9$)
- $\nabla L(w)$: Gradient của hàm mất mát tại bước t.

Cập nhật tham số:

$$w = w - \alpha \cdot v_t$$

Trong đó:

- w là vector trọng số (weights) hoặc tham số của mô hình cần tối ưu hóa.
- α là Learning rate (tốc độ học).
- v_t vận tốc tại bước t.

→ v_t đóng vai trò như "quán tính" trong việc cập nhật w , giúp làm mượt đường đi của gradient. Qua đó Momentum giúp giảm thời gian hội tụ, đặc biệt trên các bề mặt hàm mất mát phức tạp. Giảm dao động, đặc biệt trong các trường hợp gradient đổi hướng liên tục. Nhưng cần phải lựa chọn hệ số động lượng β phù hợp.

1.1.3 Adagrad (Adaptive Gradient Algorithm):

Adagrad là một thuật toán tối ưu hóa thích nghi, điều chỉnh tốc độ học (α) cho mỗi tham số dựa trên gradient của nó trong lịch sử.

Ý tưởng chính: Adagrad lưu trữ tổng bình phương gradient trong lịch sử và sử dụng giá trị này để điều chỉnh tốc độ học. Nhờ đó, Adagrad hoạt động tốt trên các bài toán có dữ liệu thưa (sparse data).

Công thức:

- Cập nhật tổng bình phương gradient:

$$g_t = g_{t-1} + (\nabla L(w))^2$$

- Cập nhật tham số:

$$w = w - \frac{\alpha}{\sqrt{g_t} + \epsilon} \cdot \nabla L(w)$$

Trong đó:

- g_t : Tổng bình phương gradient
- ϵ : Một giá trị nhỏ để tránh chia cho 0 (thường là 10^{-8})

→ Adagrad tự động điều chỉnh tốc độ học, phù hợp với dữ liệu thưa. Tuy nhiên tốc độ học giảm dần quá nhanh khiến thuật toán có thể dừng lại trước khi đạt cực tiểu toàn cục.

1.1.4 RMSProp (Root Mean Square Propagation):

RMSProp khắc phục nhược điểm của Adagrad bằng cách sử dụng trung bình động của gradient bình phương thay vì tổng gradient.

Ý tưởng chính: Sử dụng giá trị trung bình trọng số theo thời gian để điều chỉnh tốc độ học, giúp giữ cho các bước nhảy không quá lớn hoặc quá nhỏ.

Công thức:

- Cập nhật trung bình động:

$$E[g^2]_t = \beta \cdot E[g^2]_{t-1} + (1-\beta) \cdot (\nabla L(w))^2$$

Trong đó:

- $E[g^2]_t$: Trung bình động (exponentially weighted moving average) của bình phương gradient tại bước t.
- β : Hỗn số suy giảm (decay rate), thường $\beta=0.9$.
- $(\nabla L(w))^2$: Bình phương gradient của hàm mất mát $L(w)$ tại tham số w ở bước t.

- Cập nhật tham số:

$$\mathbf{w} = \mathbf{w} - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla L(\mathbf{w})$$

Trong đó:

- \mathbf{w} : Vector trọng số (weights) hoặc tham số của mô hình.
- α : Learning rate (tốc độ học).
- $\sqrt{E[g^2]_t}$: Căn bậc hai của trung bình động bình phương gradient tại bước t .
- ϵ : Giá trị nhỏ (thường $\epsilon=10^{-8}$) được thêm vào để tránh chia cho 0.
- $\nabla L(\mathbf{w})$: Gradient của hàm mất mát $L(\mathbf{w})$ tại tham số \mathbf{w} .

Ưu điểm:

- Ôn định tốc độ học, tránh ván đề giảm quá nhanh như Adagrad.
- Phù hợp cho các bài toán tối ưu hóa không tĩnh (non-stationary).

Nhược điểm:

- Cần điều chỉnh β phù hợp (thường 0.9).

1.1.5 Adam (Adaptive Moment Estimation):

Adam là thuật toán tối ưu hóa kết hợp các ý tưởng của Momentum và RMSProp.

Nó sử dụng cả trung bình động của gradient (momentum) và bình phương gradient (RMSProp) để điều chỉnh tốc độ học.

Ý tưởng chính: Adam tính toán trung bình động của gradient và bình phương gradient trong lịch sử, đồng thời áp dụng hiệu chỉnh bias để tăng độ chính xác.

Công thức:

- Tính trung bình động của gradient và bình phương gradient:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla L(\mathbf{w}) \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla L(\mathbf{w}))^2 \end{aligned}$$

Trong đó:

- m_t : Là trung bình động theo thời gian của gradient tại bước t , đại diện cho "momentum" trong Gradient Descent.

- β_1, β_2 : Hệ số suy giảm trung bình động cho gradient.
 - m_{t-1} : Giá trị trung bình động ở bước trước đó.
 - $\nabla L(w)$: Gradient của hàm mất mát L tại w.
 - v_t : Là trung bình động theo thời gian của bình phương gradient tại bước t.
 - v_{t-1} : Giá trị trung bình động của bình phương gradient ở bước trước đó.
 - $(\nabla L(w))^2$: Bình phương của gradient tại w.
- Hiệu chỉnh bias

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Trong đó:

- \hat{m}_t : Giá trị m_t đã hiệu chỉnh bias.
 - \hat{v}_t : Giá trị v_t đã hiệu chỉnh bias.
- Cập nhật tham số:
- $$w = w - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t$$
- Ưu điểm:
- Phổ biến và hiệu quả trong hầu hết các bài toán học sâu.
 - Tự điều chỉnh tốc độ học theo cả độ lớn gradient và trung bình động.
- Nhược điểm:
- Nhạy cảm với siêu tham số β_1, β_2, α .

1.2 Phương pháp (Triển khai thực nghiệm và phân tích chương trình):

Trong phần này, em sử dụng bộ dữ liệu HousingData.csv ở trên Kaggle. Quy trình thực hiện chương trình được mô tả như sau:

1.2.1 Xử lý dữ liệu:

- Mục đích: Đọc dữ liệu từ file HousingData.csv, xử lý các giá trị bị thiếu, phân tách tập dữ liệu thành tập huấn luyện và tập kiểm tra, chuẩn hóa các đặc trưng và mục tiêu để đảm bảo các giá trị nằm trong phạm vi hợp lý.

"

```
# Đọc dữ liệu và xử lý
file_path = "HousingData.csv" #Chứa thông tin về giá nhà và các đặc trưng liên
quan
housing_data = pd.read_csv(file_path)
# Thay thế giá trị thiếu bằng trung bình của từng cột
housing_data.fillna(housing_data.mean(), inplace=True)
# Phân tách đặc trưng (X) và mục tiêu (y)
X = housing_data.drop('MEDV', axis=1).values
y = housing_data['MEDV'].values.reshape(-1, 1)
# Chia dữ liệu thành tập huấn luyện và kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Chuẩn hóa dữ liệu
scaler_X = StandardScaler()
scaler_y = StandardScaler()
X_train = scaler_X.fit_transform(X_train)
X_test = scaler_X.transform(X_test)
y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)
"
```

- Giải thích:

- `housing_data.fillna(housing_data.mean(), inplace=True)` thay thế các giá trị thiếu trong từng cột bằng giá trị trung bình của cột đó.
- `train_test_split` chia dữ liệu thành 80% tập huấn luyện và 20% tập kiểm tra.
- Dùng `StandardScaler()` để chuẩn hóa các đặc trưng và mục tiêu về phân phối chuẩn ($\mu=0, \sigma=1$) nhằm tăng hiệu quả tối ưu hóa.

1.2.2 Hàm Random seed:

```
""
def set_random_seed(seed=42):
    np.random.seed(seed)
    random.seed(seed)

set_random_seed(42) # Cố định random seed
""
```

- Random Seed là giá trị khởi tạo (seed) được truyền vào các hàm sinh số ngẫu nhiên để tạo ra một chuỗi số ngẫu nhiên cố định mỗi khi chương trình được chạy.
- Mặc định, nếu không đặt seed, mỗi lần chạy chương trình sẽ tạo ra các giá trị khác nhau do sự ngẫu nhiên.
- Mục đích em sử dụng hàm này để đảm bảo tính tái lập, giúp chương trình tạo ra cùng một kết quả mỗi lần chạy trên cùng một dữ liệu và thuật toán.
- Điều này quan trọng để kiểm thử và so sánh các thuật toán tối ưu với nhau.

1.2.3 Hàm tính toán cơ bản:

a. Hàm dự đoán:

```
""
def predict(X, w, b): #Hàm dự đoán
    return np.dot(X, w) + b
""
```

- Hàm dự đoán thực hiện phép tính $\hat{y} = Xw + b$, trong đó:
 - + X: Ma trận đặc trưng
 - + w: Trọng số.
 - + b: Sai số.
- np.dot(X,w) thực hiện nhân ma trận giữa đặc trưng X và trọng số w.
- Sau đó thêm b để dịch chuyển hàm dự đoán.

b. Hàm tính mất mát:

"

```
def compute_loss(y_true, y_pred): #Hàm tính mất mát
    return np.mean((y_true - y_pred) ** 2)
"
```

- Sử dụng MSE (Mean Squared Error) để đo lường độ lệch giữa giá trị dự đoán và giá trị thực tế.
- Dựa trên công thức: $MSE = \frac{1}{n} \sum_{i=1}^n (y_{true} - y_{pred})^2$

c. Hàm tối ưu:

- Hàm batch_gradient_descent:

"

```
def batch_gradient_descent(X, y, lr=0.01, epochs=100):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    losses = []
    for epoch in range(epochs):
        y_pred = predict(X, w, b)
        loss = compute_loss(y, y_pred)
        losses.append(loss)
        dw = -2 * np.dot(X.T, (y - y_pred)) / len(X)
        db = -2 * np.sum(y - y_pred) / len(X)
```

```

w -= lr * dw
b -= lr * db
if epoch % 10 == 0:
    print(f'[Batch GD] Epoch {epoch}: Loss = {loss:.4f}')
return w, b, losses
"""

```

- + Mục đích: Cập nhật tham số dựa trên toàn bộ dữ liệu tại mỗi epoch.
- + Các bước cơ bản:
 - Khởi tạo trọng số (w) ngẫu nhiên và bias (b) bằng 0.
 - Tại mỗi epoch: Tính giá trị dự đoán y_pred, đạo hàm của hàm mất mát (gradient) theo w và b, cập nhật w và b bằng cách trừ đi một lượng nhỏ gradient nhân với learning rate.

- Hàm stochastic_gradient_descent:

```

def stochastic_gradient_descent(X, y, lr=0.01, epochs=100):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    losses = []
    for epoch in range(epochs):
        for i in range(len(X)):
            idx = np.random.randint(0, len(X))
            X_i, y_i = X[idx:idx+1], y[idx:idx+1]
            y_pred = predict(X_i, w, b)
            dw = -2 * np.dot(X_i.T, (y_i - y_pred))
            db = -2 * np.sum(y_i - y_pred)
            w -= lr * dw
            b -= lr * db
        y_pred_all = predict(X, w, b)
    """

```

```

loss = compute_loss(y, y_pred_all)
losses.append(loss)
if epoch % 10 == 0:
    print(f"[SGD] Epoch {epoch}: Loss = {loss:.4f}")
return w, b, losses
"""

```

- + Mục đích: Cập nhật tham số dựa trên từng mẫu dữ liệu ngẫu nhiên tại mỗi lần lặp.
- + Trong mỗi epoch: Chọn một mẫu dữ liệu ngẫu nhiên (X_i, y_i), cập nhật w và b dựa trên gradient của mẫu đó, thuật toán này hội tụ nhanh hơn trên các tập dữ liệu lớn nhưng có thể không ổn định.
- Hàm mini_batch_gradient_descent:

```

def mini_batch_gradient_descent(X, y, lr=0.01, epochs=100,
batch_size=32):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    losses = []
    for epoch in range(epochs):
        indices = np.random.permutation(len(X))
        X_shuffled, y_shuffled = X[indices], y[indices]
        for i in range(0, len(X), batch_size):
            X_batch, y_batch = X_shuffled[i:i+batch_size],
            y_shuffled[i:i+batch_size]
            y_pred = predict(X_batch, w, b)
            dw = -2 * np.dot(X_batch.T, (y_batch - y_pred)) / len(X_batch)
            db = -2 * np.sum(y_batch - y_pred) / len(X_batch)
            w -= lr * dw
            loss = compute_loss(y_batch, y_pred)
            losses.append(loss)
    return w, b, losses
"""

```

```

    b -= lr * db
    y_pred_all = predict(X, w, b)
    loss = compute_loss(y, y_pred_all)
    losses.append(loss)
    if epoch % 10 == 0:
        print(f"[Mini-Batch GD] Epoch {epoch}: Loss = {loss:.4f}")
    return w, b, losses
"""

```

- + Mục đích: Chia dữ liệu thành các nhóm nhỏ (mini-batch) và cập nhật tham số dựa trên từng nhóm.
- + Các bước chính: Trộn dữ liệu, cập nhật tham số theo từng batch (Gradient được tính và tham số được cập nhật dựa trên từng batch thay vì toàn bộ dữ liệu).
- Hàm momentum:

```

def momentum(X, y, lr=0.01, epochs=100, beta=0.9):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    v_w, v_b = np.zeros_like(w), np.zeros_like(b)
    losses = []
    for epoch in range(epochs):
        y_pred = predict(X, w, b)
        loss = compute_loss(y, y_pred)
        losses.append(loss)
        dw = -2 * np.dot(X.T, (y - y_pred)) / len(X)
        db = -2 * np.sum(y - y_pred) / len(X)
        v_w = beta * v_w + (1 - beta) * dw
        v_b = beta * v_b + (1 - beta) * db

```

```

w -= lr * v_w
b -= lr * v_b
if epoch % 10 == 0:
    print(f"[Momentum] Epoch {epoch}: Loss = {loss:.4f}")
return w, b, losses
"
```

- + Mục đích: Cải thiện tốc độ hội tụ bằng cách thêm động lượng vào cập nhật gradient.
- + Các bước chính: Khởi tạo động lượng, Cập nhật động lượng và tham số (Động lượng giúp tham số duy trì hướng di chuyển, giảm hiện tượng dao động.)

- Hàm adagrad:

```

def adagrad(X, y, lr=0.01, epochs=100, epsilon=1e-8):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    G_w, G_b = np.zeros_like(w), np.zeros_like(b)
    losses = []
    for epoch in range(epochs):
        y_pred = predict(X, w, b)
        loss = compute_loss(y, y_pred)
        losses.append(loss)
        dw = -2 * np.dot(X.T, (y - y_pred)) / len(X)
        db = -2 * np.sum(y - y_pred) / len(X)
        G_w += dw**2
        G_b += db**2
        w -= lr * dw / (np.sqrt(G_w) + epsilon)
        b -= lr * db / (np.sqrt(G_b) + epsilon)
```

```

if epoch % 10 == 0:
    print(f"[Adagrad] Epoch {epoch}: Loss = {loss:.4f}")
return w, b, losses
"""

+ Mục đích: Điều chỉnh learning rate dựa trên gradient tích lũy.
+ Các bước chính: Khởi tạo gradient tích lũy, tích lũy gradient bình phương
(Gradient của mỗi bước được bình phương và cộng dồn), cập nhật tham số
(Learning rate được điều chỉnh theo gradient tích lũy).

- Hàm rmsprop:
"""

def rmsprop(X, y, lr=0.01, epochs=100, beta=0.9, epsilon=1e-8):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    S_w, S_b = np.zeros_like(w), np.zeros_like(b)
    losses = []
    for epoch in range(epochs):
        y_pred = predict(X, w, b)
        loss = compute_loss(y, y_pred)
        losses.append(loss)
        dw = -2 * np.dot(X.T, (y - y_pred)) / len(X)
        db = -2 * np.sum(y - y_pred) / len(X)
        S_w = beta * S_w + (1 - beta) * (dw**2)
        S_b = beta * S_b + (1 - beta) * (db**2)
        w -= lr * dw / (np.sqrt(S_w) + epsilon)
        b -= lr * db / (np.sqrt(S_b) + epsilon)
        if epoch % 10 == 0:
            print(f"[RMS] Epoch {epoch}: Loss = {loss:.4f}")
    return w, b, losses

```

```

"
+ Mục đích: Điều chỉnh Adagrad bằng cách sử dụng trung bình động của
gradient bình phương.
+ Bước chính: Trung bình động gradient bình phương (Thay vì cộng dồn,
RMSProp sử dụng trung bình động để giảm ảnh hưởng của gradient cũ.)
- Adam:
"
def adam(X, y, lr=0.01, epochs=100, beta1=0.9, beta2=0.999, epsilon=1e-
8):
    w, b = np.random.randn(X.shape[1], 1), np.zeros((1, 1))
    m_w, v_w, m_b, v_b = np.zeros_like(w), np.zeros_like(w),
    np.zeros_like(b), np.zeros_like(b)
    losses = []
    for epoch in range(1, epochs+1):
        y_pred = predict(X, w, b)
        loss = compute_loss(y, y_pred)
        losses.append(loss)
        dw = -2 * np.dot(X.T, (y - y_pred)) / len(X)
        db = -2 * np.sum(y - y_pred) / len(X)
        m_w = beta1 * m_w + (1 - beta1) * dw
        v_w = beta2 * v_w + (1 - beta2) * (dw**2)
        m_w_hat, v_w_hat = m_w / (1 - beta1**epoch), v_w / (1 -
        beta2**epoch)
        w -= lr * m_w_hat / (np.sqrt(v_w_hat) + epsilon)
        if epoch % 10 == 0:
            print(f"[Adam] Epoch {epoch}: Loss = {loss:.4f}")
    return w, các thuật toán b, losses

```

"

+ Mục đích: Kết hợp Momentum và RMSProp để tối ưu hóa hiệu suất.

+ Bao gồm các bước chính:

- Cập nhật m và v.
- Hiệu chỉnh bias.
- Cập nhật tham số.

- Hàm so sánh các thuật toán:

"

```
def compare_methods():
    set_random_seed(42)
    methods = {
```

"Batch GD": batch_gradient_descent,

"SGD": stochastic_gradient_descent,

"Mini-Batch GD": mini_batch_gradient_descent,

"Momentum": momentum,

"Adagrad": adagrad,

"RMSProp": rmsprop,

"Adam": adam,

}

results = {}

for name, method in methods.items():

print(f"Running {name}...")

→ _, losses = method(X_train, y_train)

results[name] = losses

plt.figure(figsize=(10, 6))

```

for name, losses in results.items():
    plt.plot(losses, label=f"{name} (Final Loss: {losses[-1]:.4f})")

plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Comparison of Optimization Methods")
plt.legend()
plt.grid(True)
plt.show()

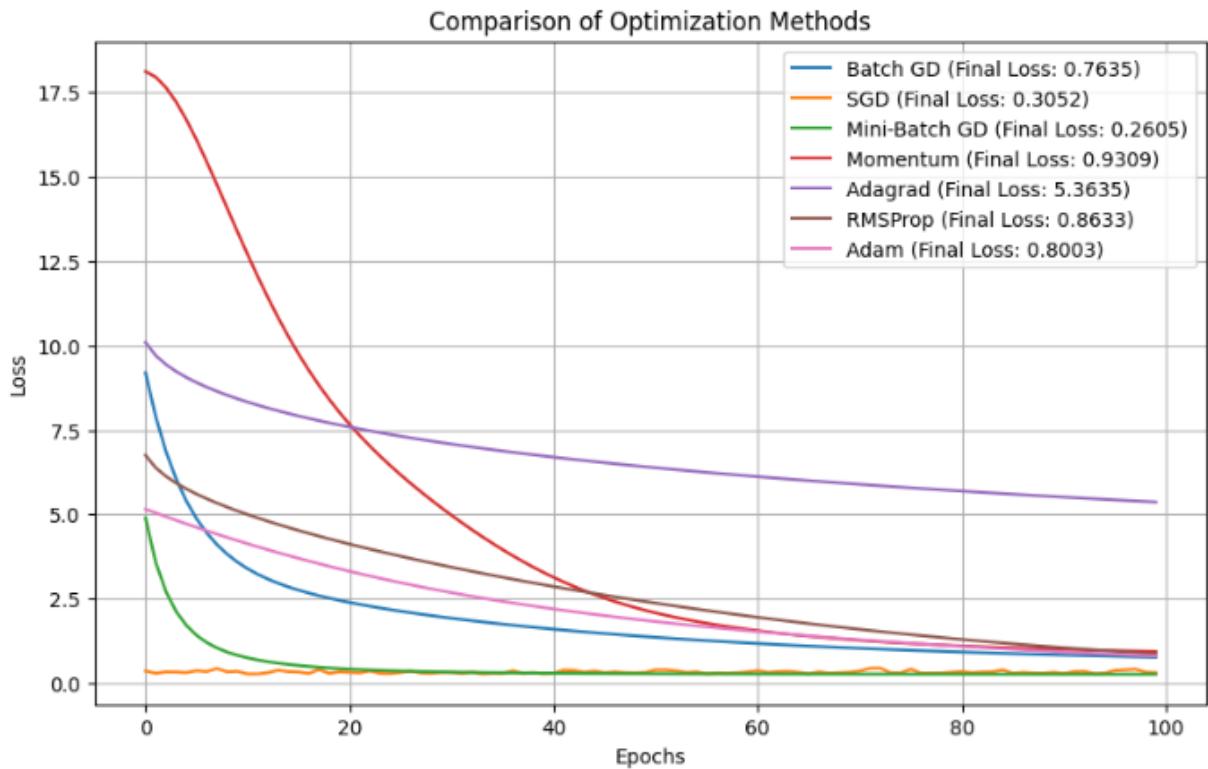
# In kết quả Final Loss
print("\nKết quả Final Loss của các thuật toán:")
for name, losses in results.items():

    print(f"{name}: Final Loss = {losses[-1]:.4f}")
    "

```

→ So sánh hiệu suất của các thuật toán tối ưu hóa, vẽ biểu đồ quá trình giảm mất mát (loss) theo thời gian.

1.3 So sánh kết quả:



Hình 2. So sánh kết quả giữa các thuật toán

- Biểu đồ thể hiện sự giảm Loss của các thuật toán theo từng Epochs. Kết quả Final Loss cũng được chú thích trong phần legend của biểu đồ.
 - + Trục X: Số lượng Epochs (100 epochs).
 - + Trục Y: Giá trị Loss (MSE - Mean Squared Error).
- Phân tích:
 - + Batch Gradient Descent (Batch GD) có Final Loss là 0.7635, dù giảm đều đặn qua các epoch, tốc độ hội tụ của Batch GD khá chậm so với các thuật toán khác vì nó tính toàn bộ dữ liệu mỗi lần cập nhật.
 - + Stochastic Gradient Descent (SGD) có Final Loss là 0.3052, SGD hội tụ nhanh hơn Batch GD do tính gradient trên từng mẫu dữ liệu ngẫu nhiên. Tuy nhiên, đường loss có dao động nhẹ do tính chất cập nhật dựa trên mẫu.

- + Mini-Batch Gradient Descent có Final Loss là 0.2605, kết quả tốt nhất trong số các thuật toán. Mini-Batch GD cân bằng giữa Batch GD và SGD khi tính toán gradient trên các nhóm nhỏ (batch), giúp hội tụ nhanh và ổn định.
- + Momentum có Final Loss là 0.9309, cải thiện tốc độ hội tụ bằng cách thêm động lượng vào hướng cập nhật. Tuy nhiên, trong trường hợp này, kết quả hội tụ chậm hơn Mini-Batch và SGD.
- + Adagrad có Final Loss là 5.3635, thích hợp với các bài toán hiếm gặp vì nó điều chỉnh learning rate dựa trên gradient tích lũy. Tuy nhiên, learning rate giảm nhanh khiến thuật toán hội tụ chậm và không đạt kết quả tốt.
- + RMSProp có Final Loss là 0.8633, giải quyết nhược điểm của Adagrad bằng cách dùng trung bình động gradient bình phương, giúp điều chỉnh learning rate hiệu quả hơn. Dù vậy, RMSProp chưa đạt kết quả tối ưu trong bài toán này.
- + Adam có Final Loss là 0.8003, kết hợp Momentum và RMSProp, thường cho kết quả tốt và hội tụ nhanh trong nhiều bài toán. Tuy nhiên, trong trường hợp này, Adam không đạt kết quả tốt nhất so với Mini-Batch GD và SGD.
- So sánh độ hội tụ và ổn định:
 - + Tốc độ hội tụ:
 - Mini-Batch GD và SGD hội tụ nhanh chóng trong các epoch đầu tiên.
 - Adam và RMSProp cũng hội tụ nhanh nhưng không vượt trội.
 - Batch GD và Momentum hội tụ chậm hơn.
 - Adagrad giảm rất chậm do learning rate giảm nhanh.
 - + Độ ổn định:
 - Batch GD và Momentum có đường hội tụ mượt, ít dao động nhưng chậm.
 - SGD dao động mạnh do cập nhật từng mẫu nhưng vẫn đạt kết quả tốt.
 - Mini-Batch GD cân bằng giữa tốc độ và độ ổn định.
 - Adam và RMSProp khá ổn định nhờ cơ chế điều chỉnh learning rate.
- Kết luận:

- + Thuật toán hiệu quả nhất: Mini-Batch Gradient Descent với Final Loss = 0.2605.
- + SGD cũng cho kết quả tốt với Final Loss = 0.3052.
- + Adam, mặc dù phô biến và mạnh mẽ, không vượt trội trong trường hợp này.
- + Adagrad có kết quả kém nhất vì learning rate giảm quá nhanh.
 - Kết quả này phụ thuộc vào dữ liệu và siêu tham số của từng thuật toán như learning rate (lr), batch size, và số epoch.
- Hướng cải tiến:
 - + Thử nghiệm với các learning rate và batch size khác nhau để tìm giá trị tối ưu.
 - + Kết hợp thêm các kỹ thuật regularization (L1, L2) để tránh overfitting.
 - + Thử nghiệm với các bộ dữ liệu khác để đánh giá tính tổng quát của các thuật toán.

PHẦN 2 – CÂU 2

2.1 Chuẩn bị dữ liệu:

- Bước 1:

Thông qua thư viện ‘*yfinance*’ được cung cấp bởi python để tải bộ dữ liệu cổ phiếu từ trang ‘**yahoo finance**’. Ở đây nhóm chúng tôi chọn dữ liệu các mã cổ phiếu 'AAPL', 'TSLA', 'F', 'EA', 'MSFT', 'TTWO'. Thông qua thư viện ‘*fredapi*’ để tải bộ dữ liệu ‘**GDP**’. Tất cả bộ dữ liệu được lấy đều là bộ dữ liệu trong thời gian 5 năm gần đây.

```
# Import thư viện cần thiết
import yfinance as yf
import pandas as pd
import datetime

# Danh sách mã cổ phiếu cần tải
symbols = ['AAPL', 'TSLA', 'F', 'EA', 'MSFT', 'TTWO']

# Thời gian 5 năm gần đây
end_date = datetime.datetime.today()
start_date = end_date - datetime.timedelta(days=5*365)

# Tạo một dictionary để lưu dữ liệu từng cổ phiếu
stock_data = {}

# Tải dữ liệu từ Yahoo Finance
for symbol in symbols:
    print(f"Đang tải dữ liệu cho {symbol}...")
    data = yf.download(symbol, start=start_date, end=end_date, interval="1d")
    stock_data[symbol] = data
    print(f"Dữ liệu {symbol} đã tải xong.\n")

# Kiểm tra dữ liệu cho một cổ phiếu cụ thể
print("Dữ liệu AAPL:")
print(stock_data['AAPL'].head())

# Lưu dữ liệu ra file CSV nếu cần
for symbol in symbols:
    stock_data[symbol].to_csv(f"{symbol}_5years_daily.csv")
    print(f"Dữ liệu {symbol} đã được lưu vào file {symbol}_5years_daily.csv")
```

Hình 3. Code tải data cổ phiếu từ Yahoo Finance

```

from fredapi import Fred
import pandas as pd

# Khởi tạo API với key của bạn
api_key = '15501ec228c76f34e532f90e7136fcf6'
fred = Fred(api_key=api_key)

# Lấy dữ liệu GDP (mã 'GDP')
gdp_data = fred.get_series('GDP')

# Đưa dữ liệu vào DataFrame
gdp_df = pd.DataFrame(gdp_data, columns=['GDP'])
gdp_df.index.name = 'Date'

# Hiển thị 5 dòng đầu tiên
print(gdp_df.head())

# Lưu dữ liệu GDP vào file CSV
gdp_df.to_csv('GDP_data.csv')
print("Dữ liệu GDP đã được lưu vào file 'GDP_data.csv'.")

```

Hình 4. Code tải data GDP từ FredAPI

- Bước 2:

Chỉnh sửa từng bộ dữ liệu cổ phiếu nhỏ sao cho đúng với định dạng gồm các cột ['Date', 'Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume', 'Industry', 'Ticker']

```

import pandas as pd

# Danh sách các mã cổ phiếu cần xử lý
symbols = ['AAPL', 'TSLA', 'F', 'EA', 'MSFT', 'TTWO']

# Xử lý từng file CSV và chuẩn hóa format
for symbol in symbols:
    print(f"Đang xử lý dữ liệu cho {symbol}...")

    # Đọc dữ liệu từ file CSV đã tải
    file_name = f"{symbol}_5years_daily.csv"
    df = pd.read_csv(file_name, skiprows=1) # Bỏ dòng 'Ticker'

    # Đổi tên cột để phù hợp format mới
    df.columns = ['Date', 'Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume']

    # Lọc bỏ các dòng có cột 'Date' chứa giá trị không hợp lệ hoặc trống
    df = df[df['Date'].notnull()] # Giữ lại các dòng có Date không rỗng
    df = df[df['Date'] != 'Date'] # Xóa dòng "Date,,,"

    # Lưu lại dữ liệu đã xử lý vào file mới
    output_file = f"{symbol}_cleaned.csv"
    df.to_csv(output_file, index=False)

    print(f"Dữ liệu {symbol} đã được chuẩn hóa và lưu vào file {output_file}.\\n")

print("Hoàn thành xử lý tất cả các file.")

```

Hình 5. Code chỉnh sửa định dạng data cổ phiếu

```

import pandas as pd

# Danh sách các mã cổ phiếu và ngành tương ứng
symbols = {
    'AAPL': 'Technology',
    'TSLA': 'Automotive',
    'F': 'Automotive',
    'EA': 'Entertainment',
    'MSFT': 'Technology',
    'TTWO': 'Entertainment'
}

# Xử lý từng file CSV và thêm cột Industry và Ticker
for symbol, industry in symbols.items():
    print(f"Đang xử lý dữ liệu cho {symbol}...")

    # Đọc dữ liệu từ file CSV đã xử lý trước đó
    input_file = f"{symbol}_cleaned.csv"
    df = pd.read_csv(input_file)

    # Thêm cột 'Industry' và 'Ticker'
    df['Industry'] = industry
    df['Ticker'] = symbol

    # Lưu lại file mới với cột bổ sung
    output_file = f"{symbol}_final.csv"
    df.to_csv(output_file, index=False)

    print(f"Dữ liệu {symbol} đã được cập nhật và lưu vào file {output_file}.\\n")

print("Hoàn thành thêm cột Industry và Ticker cho tất cả các file.")

```

Hình 6. Code thêm cột Industry và Ticker

- Bước 3:

Gộp các bộ dữ liệu nhỏ lại thành một bộ dữ liệu tổng hợp với định dạng ['Date', 'Adj Close', 'Close', 'High', 'Low', 'Open', 'Volume', 'Industry', 'Ticker', 'GDP'].

```

import pandas as pd
import os

# Danh sách các file đã xử lý cuối cùng
symbols = ['AAPL', 'TSLA', 'F', 'EA', 'MSFT', 'TTWO']
final_files = [f"{symbol}_final.csv" for symbol in symbols]

# Tạo danh sách để lưu DataFrame của từng file
dataframes = []

# Đọc từng file và thêm vào danh sách
for file in final_files:
    if os.path.exists(file): # Kiểm tra file có tồn tại không
        df = pd.read_csv(file)
        dataframes.append(df)
    else:
        print(f"File {file} không tồn tại!")

# Gộp tất cả DataFrame lại thành một DataFrame Lớn
combined_df = pd.concat(dataframes, ignore_index=True)

# Lưu file CSV tổng hợp
combined_file = "combined_stock_data.csv"
combined_df.to_csv(combined_file, index=False)

print(f"Dữ liệu đã được gộp và lưu vào file '{combined_file}'")

```

Hình 7. Code gộp các bộ data nhỏ thành một bộ data tổng hợp

```

import pandas as pd
from datetime import datetime

# Đọc dữ liệu cổ phiếu (combined_stock_data.csv) và GDP từ GDP_data.csv
combined_df = pd.read_csv('combined_stock_data.csv')
gdp_df = pd.read_csv('GDP_data.csv')

# Đảm bảo cột Date trong cả hai DataFrame có định dạng datetime
combined_df['Date'] = pd.to_datetime(combined_df['Date'])
gdp_df['Date'] = pd.to_datetime(gdp_df['Date'])

# Hàm Lấy GDP cho ngày tương ứng trong dữ liệu GDP
def get_gdp_for_date(date):
    # Tìm GDP của quý tiếp theo sau ngày cổ phiếu
    for i in range(len(gdp_df) - 1):
        start_date = gdp_df.loc[i, 'Date']
        end_date = gdp_df.loc[i + 1, 'Date']

        # Nếu ngày cổ phiếu nằm trong khoảng từ start_date đến trước end_date
        if start_date <= date < end_date:
            return gdp_df.loc[i + 1, 'GDP'] # GDP của quý tiếp theo

    # Nếu không có GDP cho ngày đó (trường hợp ngày nằm sau quý cuối cùng)
    return None

# Áp dụng hàm get_gdp_for_date cho từng dòng trong dữ liệu cổ phiếu
combined_df['GDP'] = combined_df['Date'].apply(get_gdp_for_date)

# Lưu dữ liệu kết quả vào file mới
combined_df.to_csv('final_stock_data_with_GDP.csv', index=False)

print("Hoàn tất việc gộp GDP vào dữ liệu cổ phiếu.")

```

Hình 8. Code gộp GDP vào bộ data tổng hợp

- Bước 4:

Lọc dữ liệu và lưu thành bộ dữ liệu cuối cùng ‘***data_src_2.csv***’

```

import pandas as pd

# Đọc dữ liệu từ file CSV
df = pd.read_csv('final_stock_data_with_GDP.csv')

# Loại bỏ các dòng có giá trị NaN trong cột 'GDP'
df_cleaned = df.dropna(subset=['GDP'])

# Lưu lại dữ liệu đã xử lý vào file CSV mới
df_cleaned.to_csv('data_src2.csv', index=False)

```

Hình 9. Code lọc và lưu bộ dữ liệu cuối cùng

2.2 Tiết xuât lý dữ liệu:

- Chuẩn hóa dữ liệu: Dữ liệu được chuẩn hóa với MinMaxScaler để giá trị "Open" của cổ phiếu nằm trong khoảng (0, 1). Sau đó, dữ liệu được chia thành các chuỗi con dài 60 ngày, với đầu vào là giá "Open" của 60 ngày trước và đầu ra là giá của ngày tiếp theo.
- Chia bộ dữ liệu thành bộ train và bộ test: dữ liệu được chia thành 80% để train và 20% để test.

```

from sklearn.preprocessing import MinMaxScaler
import numpy as np

def prepare_data(ticker_data, sequence_length=60):
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(ticker_data['Open'].values.reshape(-1, 1))

    X, y = [], []
    for i in range(sequence_length, len(scaled_data)):
        X.append(scaled_data[i-sequence_length:i, 0])
        y.append(scaled_data[i, 0])

    X, y = np.array(X), np.array(y)

    # Chia dữ liệu thành tập huấn luyện và kiểm tra
    split = int(0.8 * len(X))
    X_train, X_eval = X[:split], X[split:]
    y_train, y_eval = y[:split], y[split:]

    return X, y, scaler, X_train, X_eval, y_train, y_eval

```

Hình 10. Code chuẩn hóa và chia dữ liệu

2.3 Thủ nghiệm các phương pháp học máy:

2.3.1 Feedforward Neural Network:

a. Xây dựng mô hình:

- Cấu trúc mô hình:
 - + Lớp đầu vào (Input layer):
 - Số nút (neurons) phù hợp với số biến đầu vào của bài toán.
 - Nhận thông tin đã được chuẩn hóa từ bộ dữ liệu.
 - + Hai lớp ẩn (Hidden layers):
 - Mỗi lớp gồm 50 nút (neurons).
 - Sử dụng hàm kích hoạt ReLU (Rectified Linear Unit).
 - Hàm ReLU giúp mạng học được các quan hệ phi tuyến tính giữa các biến.
 - + Lớp đầu ra (Output layer):
 - Chỉ gồm một nút.
 - Sử dụng hàm tuyến tính để dự đoán giá mở cửa.
- Trình tự huấn luyện:
 - + Hàm mất mát (Loss function):
 - Sử dụng Mean Squared Error (MSE), để đánh giá mức độ chênh lệch giữa giá dự đoán và giá thực tế.
 - + Thuật toán tối ưu hóa (Optimizer):
 - Sử dụng Adam Optimizer, một thuật toán điều chỉnh trọng số với tốc độ nhanh và hiệu quả.
 - + Các tham số huấn luyện:
 - Số Epoch: 10 (tương ứng với 10 vòng lặp qua tất cả dữ liệu huấn luyện).
 - Batch size: 32 (để tối ưu hóa tốc độ huấn luyện và hiệu quả tính toán).

Code:

```

“from keras.models import Sequential
from keras.layers import Dense, Input
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# Lưu kết quả cho từng mã cổ phiếu
results_fnn = {}

# Hàm xây dựng mô hình Feedforward Neural Network
def build_ffnn_model(input_shape):
    model_fnn = Sequential()
    model_fnn.add(Input(shape=(input_shape,))) # Lớp Input cho hình dạng
    đầu vào
    model_fnn.add(Dense(50, activation='relu'))
    model_fnn.add(Dense(50, activation='relu'))
    model_fnn.add(Dense(1)) # Output layer
    model_fnn.compile(optimizer='adam', loss='mean_squared_error')
    return model_fnn

# Lặp qua từng mã cổ phiếu
for ticker in tickers:
    ticker_data_fnn = data[data['Ticker'] == ticker].sort_values('Date')
    X_fnn, y_fnn, scaler_fnn, X_train_fnn, X_eval_fnn, y_train_fnn, y_eval_fnn
    = prepare_data(ticker_data_fnn)

```

```

# Xây dựng và huấn luyện mô hình Feedforward Neural Network
model_fnn = build_ffnn_model(X_train_fnn.shape[1])
history_fnn = model_fnn.fit(X_train_fnn, y_train_fnn, epochs=10,
batch_size=32, verbose=0, validation_data=(X_eval_fnn, y_eval_fnn))

# Dự đoán trên dữ liệu huấn luyện
train_predictions_fnn = model_fnn.predict(X_train_fnn)
train_predictions_fnn = scaler_fnn.inverse_transform(train_predictions_fnn)
y_train_actual_fnn = scaler_fnn.inverse_transform(y_train_fnn.reshape(-1, 1))

# Dự đoán trên dữ liệu kiểm tra
test_predictions_fnn = model_fnn.predict(X_eval_fnn)
test_predictions_fnn = scaler_fnn.inverse_transform(test_predictions_fnn)
y_eval_actual_fnn = scaler_fnn.inverse_transform(y_eval_fnn.reshape(-1, 1))

# Tính MSE và R2 cho dữ liệu huấn luyện
train_mse_fnn = mean_squared_error(y_train_actual_fnn, train_predictions_fnn)
train_r2_fnn = r2_score(y_train_actual_fnn, train_predictions_fnn)

# Tính MSE và R2 cho dữ liệu kiểm tra
test_mse_fnn = mean_squared_error(y_eval_actual_fnn, test_predictions_fnn)

```

```
test_r2_fnn = r2_score(y_eval_actual_fnn, test_predictions_fnn)
```

```
# Lưu kết quả
results_fnn[ticker] = {
    'dates': ticker_data.iloc[:len(y_train_fnn)] +
    len(y_eval_fnn)]['Date'].values,
    'actual': np.concatenate([y_train_actual_fnn, y_eval_actual_fnn],
    axis=0),
    'predicted': np.concatenate([train_predictions_fnn,
    test_predictions_fnn], axis=0),
    'train_mse_fnn': train_mse_fnn,
    'train_r2_fnn': train_r2_fnn,
    'test_mse_fnn': test_mse_fnn,
    'test_r2_fnn': test_r2_fnn,
    'history_fnn': history_fnn
}"
```

b. Kết quả:

- Biểu đồ huấn luyện: biểu đồ thể hiện quá trình huấn luyện của mô hình FNN đối với từng mã cổ phiếu. Biểu đồ so sánh Loss (máy tính) giữa dữ liệu huấn luyện và dữ liệu kiểm tra qua các epoch.

Code:

```
"import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
# Giả sử có số lượng ticker (có thể tính n từ results_fnn)
```

```
n = len(results_fnn)
```

```
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
```

```

cols = 2 # 2 cột

# Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))

# Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()

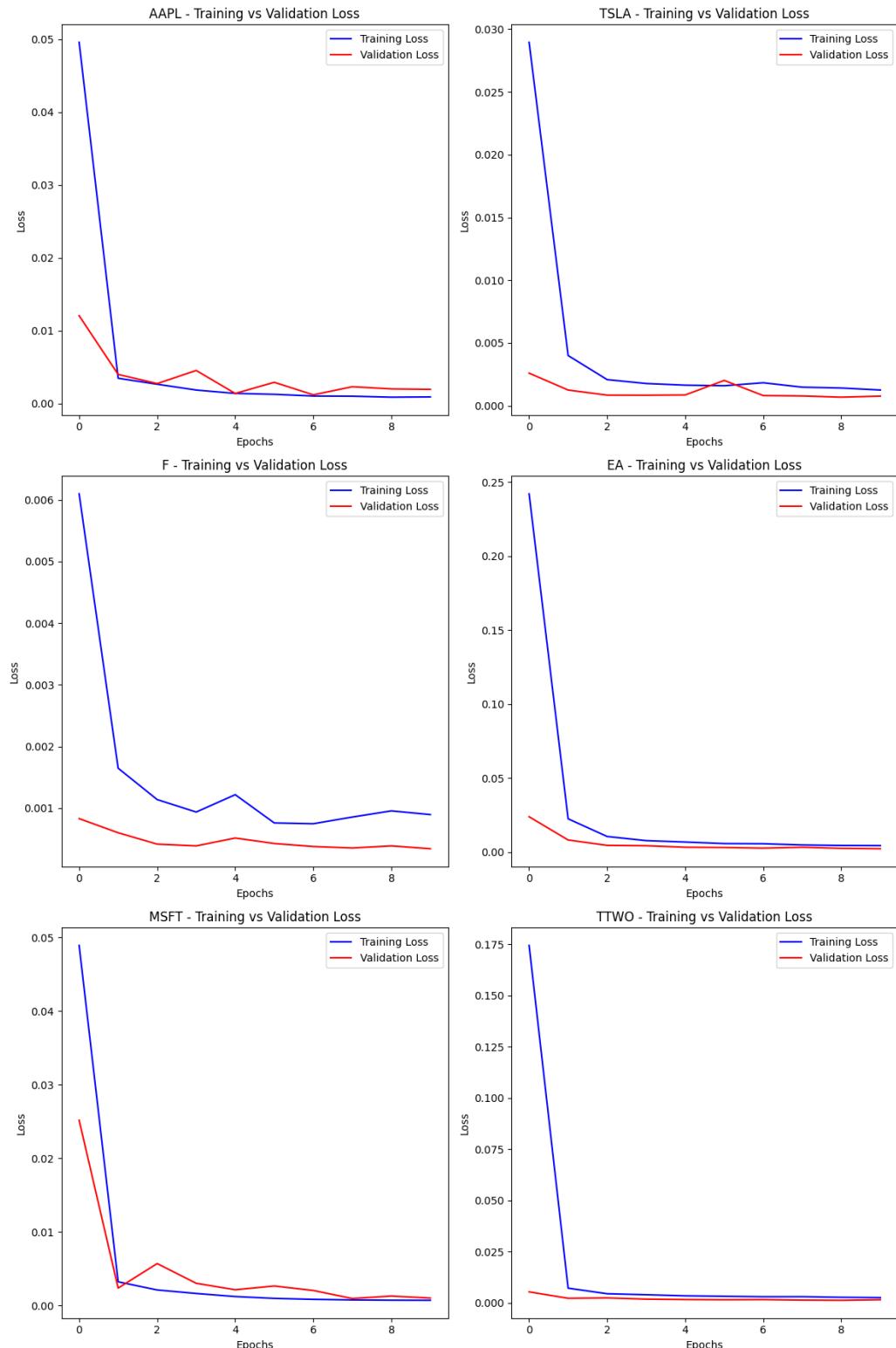
# Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_fnn.items()):
    # Vẽ đồ thị huấn luyện và kiểm tra loss
    history = result['history_fnn']

    ax = axes[idx] # Chọn subplot tương ứng

    ax.plot(history.history['loss'], color='blue', label='Training Loss')
    ax.plot(history.history['val_loss'], color='red', label='Validation Loss')
    ax.set_title(f'{ticker} - Training vs Validation Loss')
    ax.set_xlabel('Epochs')
    ax.set_ylabel('Loss')
    ax.legend()

# Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show() "

```



Hình 11. Biểu đồ huấn luyện bằng mô hình FNN

- Biểu đồ kết quả dự đoán : Biểu đồ thể hiện so sánh giá thực tế (Actual Price) với các giá trị dự đoán từ dữ liệu huấn luyện (Train Prediction) và dữ liệu kiểm tra (Test Prediction).

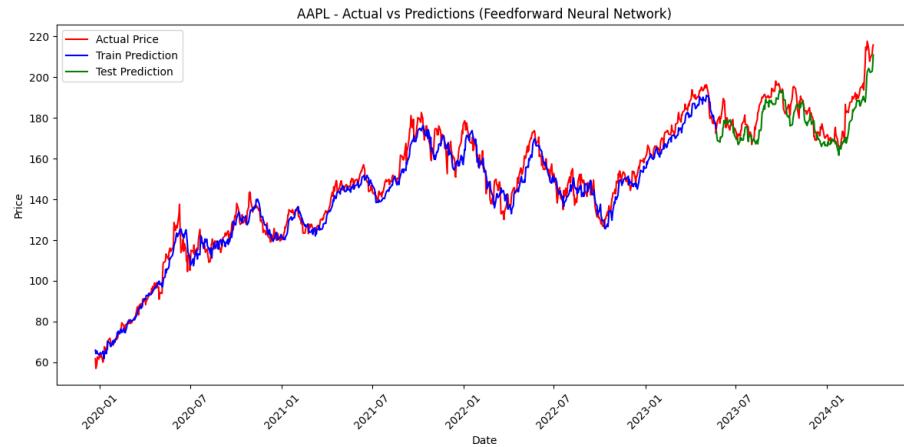
Code:

```

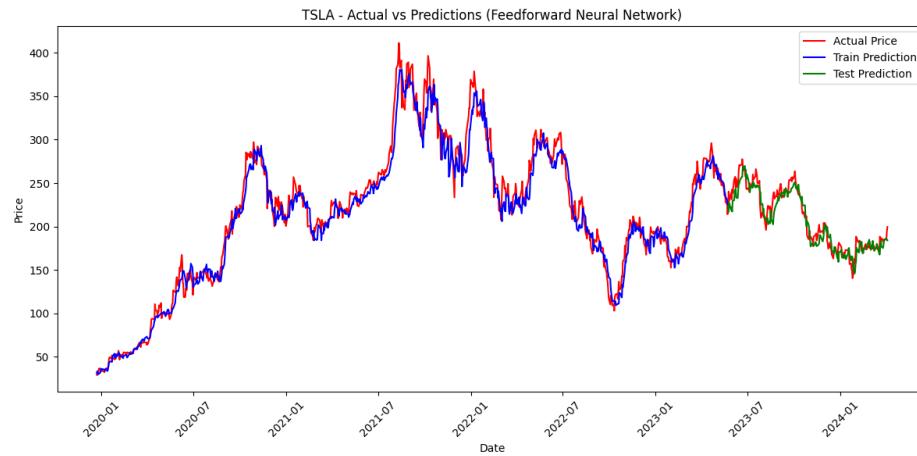
“# Vẽ biểu đồ kết quả
for ticker, result in results_fnn.items():
    plt.figure(figsize=(12, 6))
    # Vẽ đồ thị actual price (màu đỏ)
    plt.plot(result['dates'], result['actual'], color='red', label='Actual Price')
    # Vẽ đồ thị train và test predictions (màu xanh dương cho train, xanh lá cho
    # test)
    plt.plot(result['dates'][:len(result['actual'])], result['predicted'][:len(result['actual'])], color='blue', label='Train Prediction')
    plt.plot(result['dates'][len(result['actual']):], result['predicted'][len(result['actual']):], color='green', label='Test Prediction')

    plt.title(f'{ticker} - Actual vs Predictions (Feedforward Neural Network)')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()”

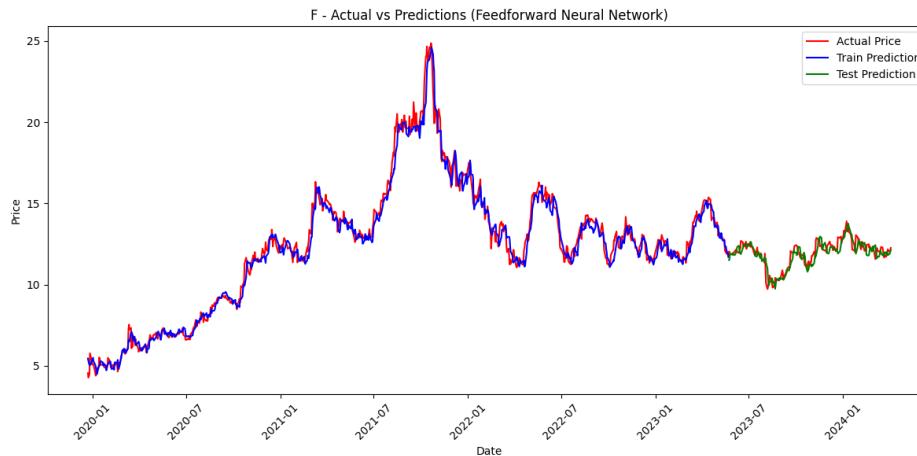
```



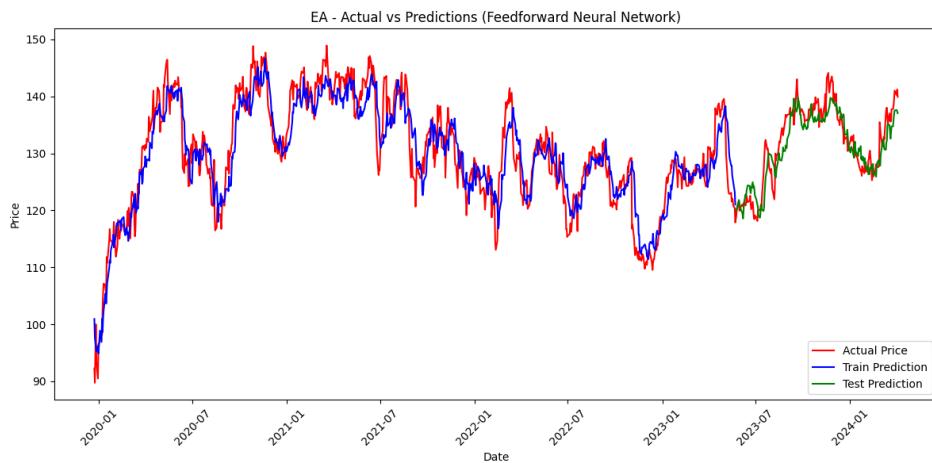
Hình 12. Kết quả dự đoán bằng mô hình FNN cho ticker AAPL



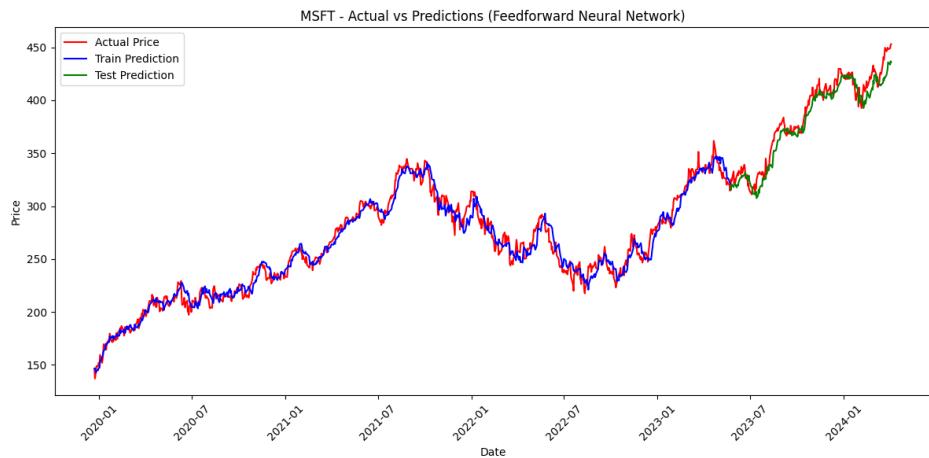
Hình 13. Kết quả dự đoán bằng mô hình FNN cho ticker TSLA



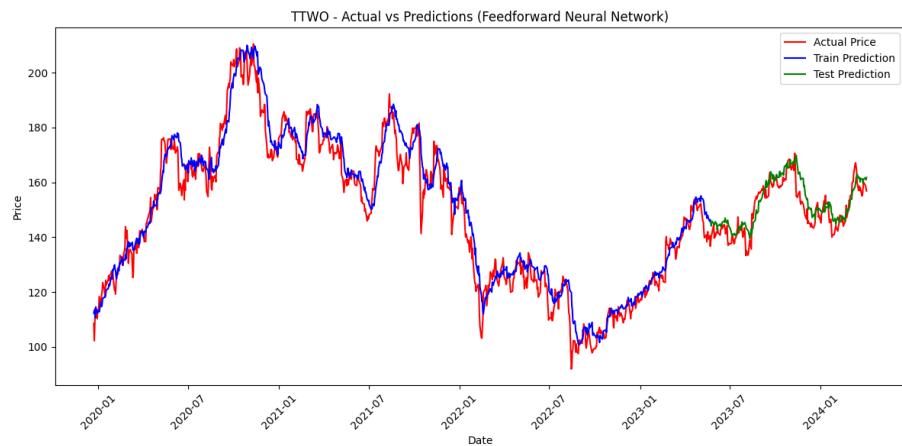
Hình 14. Kết quả dự đoán bằng mô hình FNN cho ticker F



Hình 15. Kết quả dự đoán bằng mô hình FNN cho ticker EA



Hình 16. Kết quả dự đoán bằng mô hình FNN cho ticker MSFT



Hình 17. Kết quả dự đoán bằng mô hình FNN cho ticker TTWO

- Chỉ số MSE và R²: độ chính xác của mô hình được đánh giá bằng các chỉ số MSE và R².

Code:

```
"for ticker, result in results_fnn.items():
    print(f'{ticker} - Train MSE: {result["train_mse_fnn"]}, Train R²:
          {result["train_r2_fnn"]}')
    print(f'{ticker} - Test MSE: {result["test_mse_fnn"]}, Test R²:
          {result["test_r2_fnn"]}'')
AAPL - Train MSE: 24.34755454595284, Train R²: 0.9706565168281567
AAPL - Test MSE: 50.789415676520804, Test R²: 0.593156311777878
TSLA - Train MSE: 212.17576053054051, Train R²: 0.968313907461968
TSLA - Test MSE: 115.26858174927412, Test R²: 0.900233134007787
F - Train MSE: 0.28366583138395607, Train R²: 0.9821560458782536
F - Test MSE: 0.14460984326991302, Test R²: 0.7732381654433121
EA - Train MSE: 14.742626611022606, Train R²: 0.8508109411040294
EA - Test MSE: 8.501578076660126, Test R²: 0.8122256275951703
MSFT - Train MSE: 64.3055192289078, Train R²: 0.9692438312407321
MSFT - Test MSE: 101.5491953400672, Test R²: 0.9323412196937872
TTWO - Train MSE: 37.63952422288253, Train R²: 0.9508142241799736
TTWO - Test MSE: 21.23814797650859, Test R²: 0.7263781811487907
```

Hình 18. Kết quả các chỉ số MSE, R2

2.3.2 Recurrent Neural Network (RNN):

a. Xây dựng mô hình:

- Cấu trúc mô hình:
 - + Lớp đầu vào (Input layer):
 - Dùng lớp Input với kích thước dữ liệu đầu vào là (X_train_rnn.shape[1], 1), mỗi lần xử lý sẽ có một đặc trưng đầu vào.
 - Dữ liệu đã được chuẩn hóa trước khi đưa vào mô hình.
 - + Hai lớp LSTM (Long Short-Term Memory):
 - Lớp 1: LSTM với 50 đơn vị (neurons) và return_sequences=True, giúp mô hình lưu trữ chuỗi thông tin dài hạn.

- Lớp 2: LSTM với 50 đơn vị, không trả về chuỗi mà chỉ trả về trạng thái cuối cùng.
 - LSTM giúp mô hình học các mối quan hệ giữa các giá trị theo chuỗi thời gian, rất phù hợp cho các bài toán dự báo theo thời gian như dự đoán giá cổ phiếu.
- + Lớp đầu ra (Output layer):
- Dùng lớp Dense với 1 nút để dự đoán giá trị cổ phiếu.
 - Sử dụng hàm kích hoạt tuyến tính (linear activation) để dự đoán giá trị liên tục.
- Trình tự huấn luyện:
- + Hàm mất mát (Loss function):
- Sử dụng Mean Squared Error (MSE) để đánh giá mức độ sai lệch giữa giá trị thực tế và dự đoán.
- + Thuật toán tối ưu hóa (Optimizer):
- Sử dụng Adam Optimizer, giúp tối ưu hóa các trọng số của mạng với tốc độ học thích ứng và hiệu quả.
- + Các tham số huấn luyện:
- Số Epoch: 10 — tức là mô hình sẽ được huấn luyện qua 10 lần lặp toàn bộ dữ liệu huấn luyện.
 - Batch size: 32 — số lượng mẫu được xử lý trong mỗi lần cập nhật trọng số của mô hình.
- Code:
- ```
'from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import pandas as pd'
```

```

import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã cổ phiếu
results_rnn = {}

Hàm xây dựng mô hình RNN
def build_rnn_model(input_shape):
 model_rnn = Sequential()
 model_rnn.add(Input(shape=input_shape))
 model_rnn.add(LSTM(50, return_sequences=True))
 model_rnn.add(LSTM(50))
 model_rnn.add(Dense(1))
 model_rnn.compile(optimizer='adam', loss='mean_squared_error')
 return model_rnn

Lặp qua từng mã cổ phiếu
for ticker in tickers:
 ticker_data_rnn = data[data['Ticker'] == ticker].sort_values('Date')
 X_rnn, y_rnn, scaler_rnn, X_train_rnn, X_eval_rnn, y_train_rnn, y_eval_rnn =
 prepare_data(ticker_data_rnn)

 # Xây dựng và huấn luyện mô hình
 model_rnn = build_rnn_model((X_train_rnn.shape[1], 1))
 history_rnn = model_rnn.fit(X_train_rnn, y_train_rnn, epochs=10,
 batch_size=32, verbose=0, validation_data=(X_eval_rnn, y_eval_rnn))

 # Dự đoán trên dữ liệu huấn luyện

```

```

train_predictions_rnn = model_rnn.predict(X_train_rnn)
train_predictions_rnn = scaler_rnn.inverse_transform(train_predictions_rnn)
y_train_actual_rnn = scaler_rnn.inverse_transform(y_train_rnn.reshape(-1,
I))

Dự đoán trên dữ liệu kiểm tra
test_predictions_rnn = model_rnn.predict(X_eval_rnn)
test_predictions_rnn = scaler_rnn.inverse_transform(test_predictions_rnn)
y_eval_actual_rnn = scaler_rnn.inverse_transform(y_eval_rnn.reshape(-1, 1))

Tính MSE và R2 cho dữ liệu huấn luyện
train_mse_rnn = mean_squared_error(y_train_actual_rnn,
train_predictions_rnn)
train_r2_rnn = r2_score(y_train_actual_rnn, train_predictions_rnn)

Tính MSE và R2 cho dữ liệu kiểm tra
test_mse_rnn = mean_squared_error(y_eval_actual_rnn, test_predictions_rnn)
test_r2_rnn = r2_score(y_eval_actual_rnn, test_predictions_rnn)

Lưu kết quả
results_rnn[ticker] = {
 'dates': ticker_data.iloc[:len(y_train_rnn) + len(y_eval_rnn)][['Date']].values,
 'actual': np.concatenate([y_train_actual_rnn, y_eval_actual_rnn], axis=0),
 'predicted': np.concatenate([train_predictions_rnn, test_predictions_rnn],
axis=0),
 'train_mse_rnn': train_mse_rnn,
 'train_r2_rnn': train_r2_rnn,
}

```

```

'test_mse_rnn': test_mse_rnn,
'test_r2_rnn': test_r2_rnn,
'history_rnn': history_rnn
}”

```

b. Kết quả:

- Biểu đồ huấn luyện: So sánh độ măt măt (Loss) giữa dữ liệu huấn luyện và kiểm tra qua các epoch, giúp đánh giá quá trình học của mô hình.

Code:

```

“from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Input
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã phiếu
results_rnn = {}

Hàm xây dựng mô hình RNN
def build_rnn_model(input_shape):
 model_rnn = Sequential()
 model_rnn.add(Input(shape=input_shape))
 model_rnn.add(LSTM(50, return_sequences=True))
 model_rnn.add(LSTM(50))
 model_rnn.add(Dense(1))
 model_rnn.compile(optimizer='adam', loss='mean_squared_error')
 return model_rnn

```

```

Lắp qua từng mã cổ phiếu
for ticker in tickers:
 ticker_data_rnn = data[data['Ticker'] == ticker].sort_values('Date')
 X_rnn, y_rnn, scaler_rnn, X_train_rnn, X_eval_rnn, y_train_rnn,
 y_eval_rnn = prepare_data(ticker_data_rnn)

Xây dựng và huấn luyện mô hình
model_rnn = build_rnn_model((X_train_rnn.shape[1], 1))
history_rnn = model_rnn.fit(X_train_rnn, y_train_rnn, epochs=10,
batch_size=32, verbose=0, validation_data=(X_eval_rnn, y_eval_rnn))

Dự đoán trên dữ liệu huấn luyện
train_predictions_rnn = model_rnn.predict(X_train_rnn)
train_predictions_rnn =
scaler_rnn.inverse_transform(train_predictions_rnn)

y_train_actual_rnn =
scaler_rnn.inverse_transform(y_train_rnn.reshape(-1, 1))

Dự đoán trên dữ liệu kiểm tra
test_predictions_rnn = model_rnn.predict(X_eval_rnn)
test_predictions_rnn =
scaler_rnn.inverse_transform(test_predictions_rnn)

y_eval_actual_rnn = scaler_rnn.inverse_transform(y_eval_rnn.reshape(
-1, 1))

Tính MSE và R2 cho dữ liệu huấn luyện

```

```

train_mse_rnn = mean_squared_error(y_train_actual_rnn,
train_predictions_rnn)

train_r2_rnn = r2_score(y_train_actual_rnn, train_predictions_rnn)

Tính MSE và R2 cho dữ liệu kiểm tra
test_mse_rnn = mean_squared_error(y_eval_actual_rnn,
test_predictions_rnn)

test_r2_rnn = r2_score(y_eval_actual_rnn, test_predictions_rnn)

Lưu kết quả
results_rnn[ticker] = {
 'dates': ticker_data.iloc[:len(y_train_rnn) + len(y_eval_rnn)][['Date']].values,
 'actual': np.concatenate([y_train_actual_rnn, y_eval_actual_rnn], axis=0),
 'predicted': np.concatenate([train_predictions_rnn, test_predictions_rnn], axis=0),
 'train_mse_rnn': train_mse_rnn,
 'train_r2_rnn': train_r2_rnn,
 'test_mse_rnn': test_mse_rnn,
 'test_r2_rnn': test_r2_rnn,
 'history_rnn': history_rnn # Giả sử có số lượng ticker (có thể tính n từ results_fnn)
}

n = len(results_rnn)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột

```

```

Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))

Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()

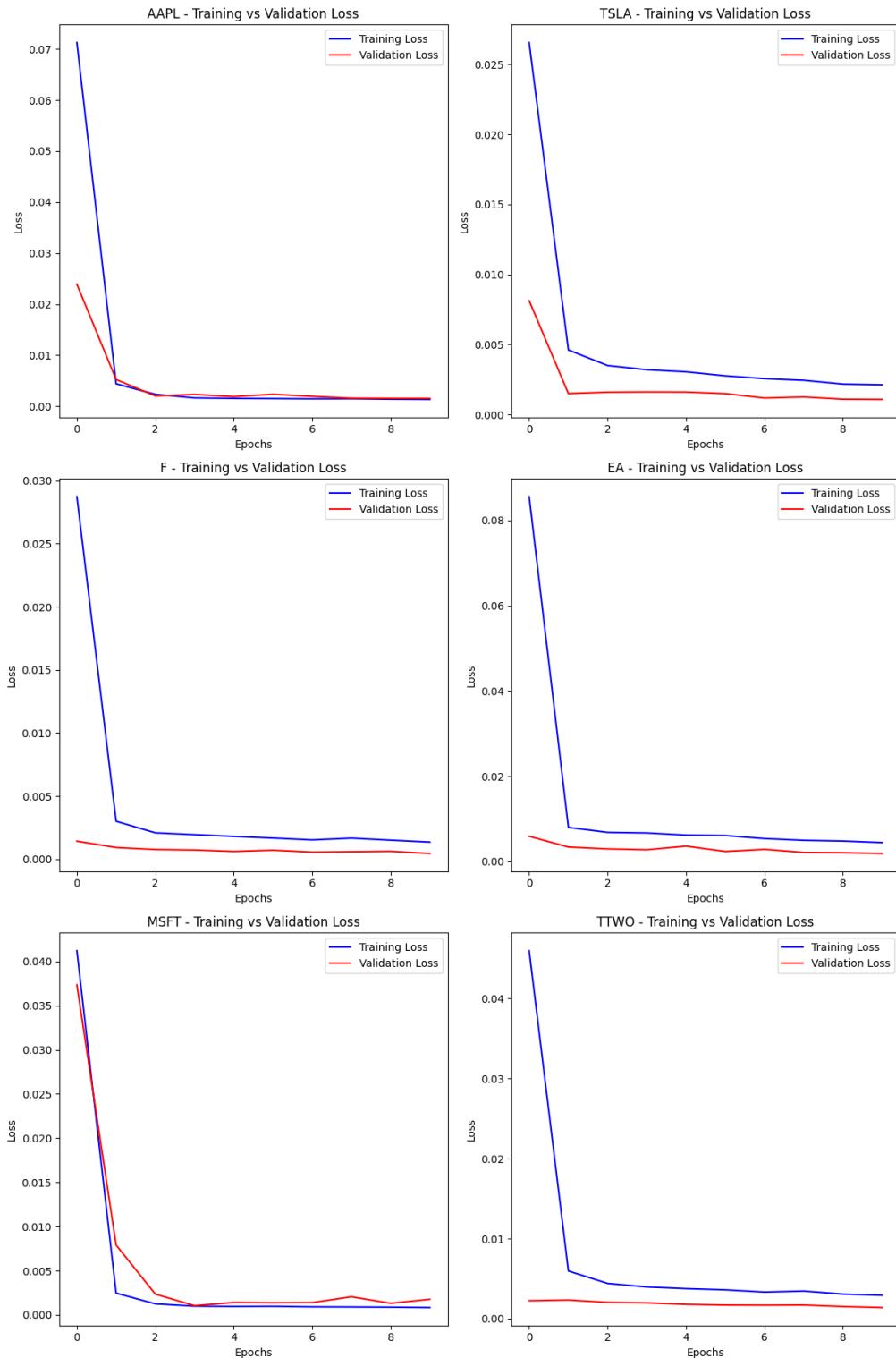
Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_rnn.items()):
 # Vẽ đồ thị huấn luyện và kiểm tra loss
 history = result['history_rnn']

 ax = axes[idx] # Chọn subplot tương ứng

 ax.plot(history.history['loss'], color='blue', label='Training Loss')
 ax.plot(history.history['val_loss'], color='red', label='Validation Loss')
 ax.set_title(f'{ticker} - Training vs Validation Loss')
 ax.set_xlabel('Epochs')
 ax.set_ylabel('Loss')
 ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show()
}

```



Hình 19. Biểu đồ huấn luyện bằng mô hình RNN

- Biểu đồ kết quả dự đoán: So sánh giá thực tế (Actual Price) và giá dự đoán từ dữ liệu huấn luyện (Train Prediction) và kiểm tra (Test Prediction), giúp đánh giá độ chính xác của mô hình.

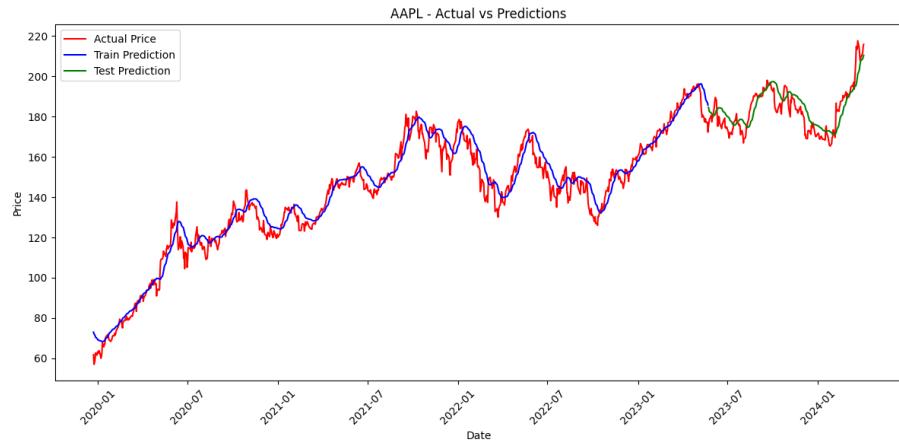
Code:

```
"# Vẽ biểu đồ kết quả
for ticker, result in results_rnn.items():
 plt.figure(figsize=(12, 6))

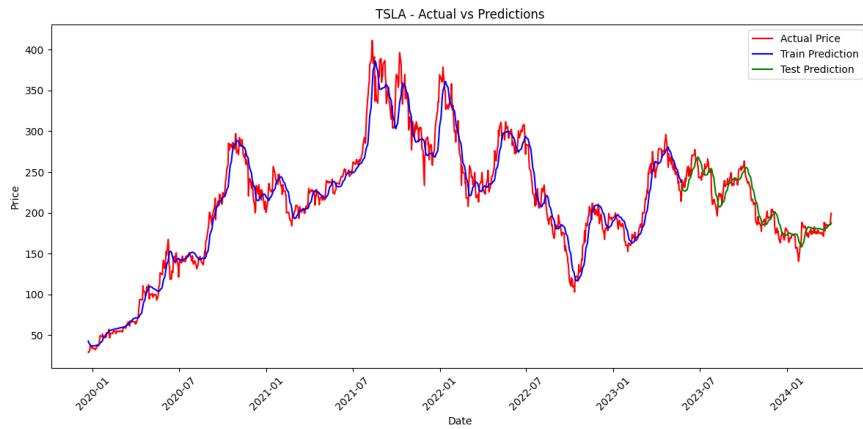
 # Vẽ đồ thị actual price (màu đỏ)
 plt.plot(result['dates'], result['actual'], color='red', label='Actual Price')

 # Vẽ đồ thị train và test predictions (màu xanh dương cho train, xanh lá
 # cho test)
 plt.plot(result['dates'][:len(result['actual'])], result['predicted'][:len(result['actual'])],
 color='blue', label='Train Prediction')
 plt.plot(result['dates'][len(result['actual']):], result['predicted'][len(result['actual']):],
 color='green', label='Test Prediction')

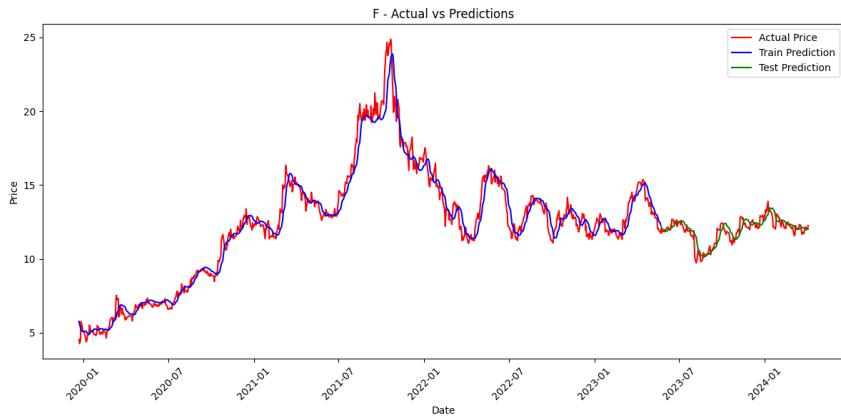
 plt.title(f'{ticker} - Actual vs Predictions')
 plt.xlabel('Date')
 plt.ylabel('Price')
 plt.xticks(rotation=45)
 plt.legend()
 plt.tight_layout()
 plt.show()"
```



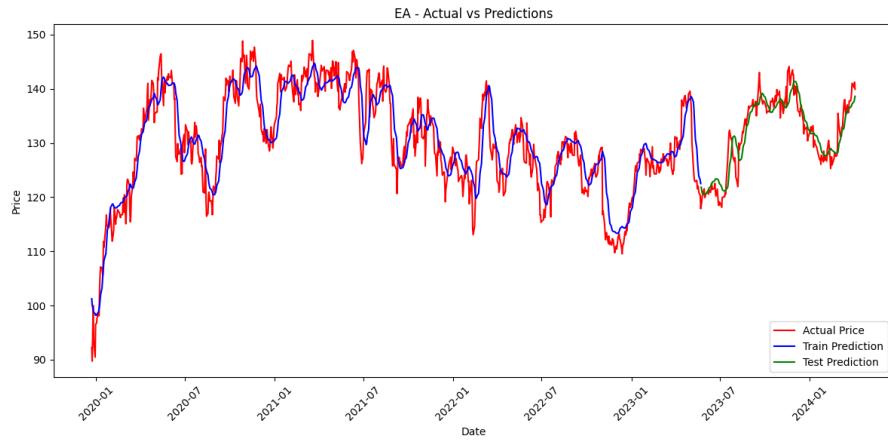
Hình 20. Kết quả dự đoán bằng mô hình RNN cho ticker AAPL



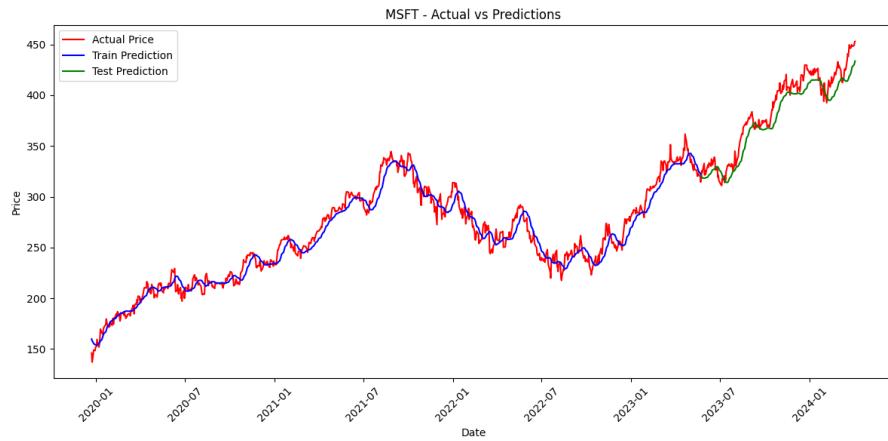
Hình 21. Kết quả dự đoán bằng mô hình RNN cho ticker TSLA



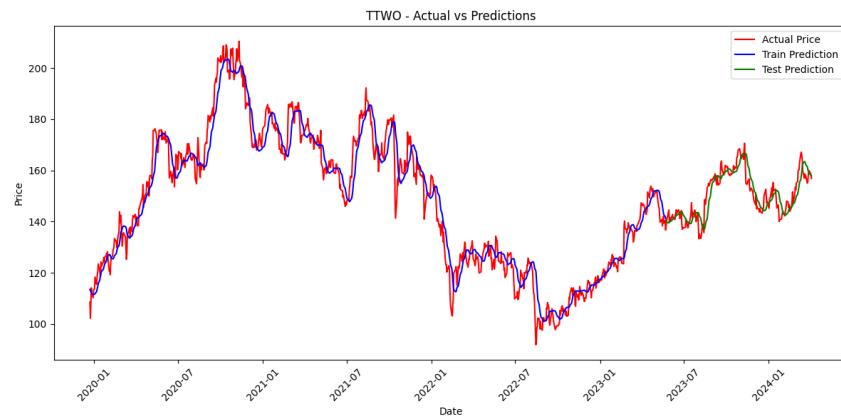
Hình 22. Kết quả dự đoán bằng mô hình RNN cho ticker F



Hình 23. Kết quả dự đoán bằng mô hình RNN cho ticker EA



Hình 24. Kết quả dự đoán bằng mô hình RNN cho ticker MSFT



Hình 25. Kết quả dự đoán bằng mô hình RNN cho ticker TTWO

- Chỉ số MSE và R<sup>2</sup>: MSE và R<sup>2</sup> được tính để đánh giá mức độ sai lệch và khả năng giải thích biến động của mô hình trên dữ liệu huấn luyện và kiểm tra.

Code:

```
"for ticker, result in results_rnn.items():
 print(f'{ticker} - Train MSE: {result["train_mse_rnn"]}, Train R²:
{result["train_r2_rnn"]}')
 print(f'{ticker} - Test MSE: {result["test_mse_rnn"]}, Test R²:
{result["test_r2_rnn"]}"')
```

```
AAPL - Train MSE: 36.23477974962375, Train R²: 0.9563301255651043
AAPL - Test MSE: 39.73533626776989, Test R²: 0.6817039427488614
TSLA - Train MSE: 291.352031973881, Train R²: 0.9564898109794251
TSLA - Test MSE: 161.75670174792128, Test R²: 0.859996896450677
F - Train MSE: 0.5094700151285806, Train R²: 0.967951869521942
F - Test MSE: 0.1913999602894171, Test R²: 0.6998668614259156
EA - Train MSE: 15.411932052624739, Train R²: 0.8440378570681143
EA - Test MSE: 6.709550726797409, Test R²: 0.8518061393446966
MSFT - Train MSE: 83.36659503842807, Train R²: 0.9601272628441073
MSFT - Test MSE: 176.62761785578937, Test R²: 0.8823190163890997
TTWO - Train MSE: 38.34691337299308, Train R²: 0.9498898372523168
TTWO - Test MSE: 19.6380546397273, Test R²: 0.7469929941553706
```

Hình 26. Kết quả các chỉ số MSE, R2

### 2.3.3 Linear Regression:

#### a. Xây dựng mô hình:

- Cấu trúc mô hình:

- + Lớp đầu vào (Input layer):

- Số nút đầu vào tương ứng với số lượng biến đầu vào của dữ liệu. Dữ liệu đã được chuẩn hóa trước khi đưa vào mô hình.

- + Lớp ẩn (Hidden layers):

- Không sử dụng lớp ẩn nào, vì đây là mô hình hồi quy tuyến tính đơn giản với một lớp Dense duy nhất.
- + Lớp đầu ra (Output layer):
  - Chỉ có một nút để dự đoán giá trị liên tục (ví dụ: giá cổ phiếu).
- Trình tự huấn luyện:
  - + Hàm mất mát (Loss function):
    - Sử dụng Mean Squared Error (MSE) để đánh giá độ sai lệch giữa giá trị dự đoán và giá trị thực tế.
  - + Thuật toán tối ưu hóa (Optimizer):
    - Adam Optimizer được sử dụng để cập nhật trọng số mô hình một cách nhanh chóng và hiệu quả.
  - + Các tham số huấn luyện:
    - Số Epoch: 10 vòng lặp.
    - Batch size: 32.

Code:

```
"from keras.models import Sequential
from keras.layers import Dense, Input
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã cổ phiếu
results_lr = {}

Hàm xây dựng mô hình Hồi quy tuyến tính
def build_linear_regression_model(input_shape):
 model_lr = Sequential()
```

```

model_lr.add(Input(shape=(input_shape,))) # Lớp Input cho hình dạng đầu
vào

model_lr.add(Dense(1)) # Chỉ có 1 lớp Dense cho Linear Regression
model_lr.compile(optimizer='adam', loss='mean_squared_error')
return model_lr

Lắp qua từng mảng phiếu
for ticker in tickers:
 ticker_data_lr = data[data['Ticker'] == ticker].sort_values('Date')

 X_lr, y_lr, scaler_lr, X_train_lr, X_eval_lr, y_train_lr, y_eval_lr =
 prepare_data(ticker_data_lr)

 # Xây dựng và huấn luyện mô hình Hồi quy tuyến tính
 model_lr = build_linear_regression_model(X_train_lr.shape[1])
 model_lr.fit(X_train_lr, y_train_lr, epochs=10, batch_size=32, verbose=0)
 history_lr = model_lr.fit(X_train_lr, y_train_lr, epochs=10, batch_size=32,
 verbose=0, validation_data=(X_eval_lr, y_eval_lr))

 # Dự đoán trên dữ liệu huấn luyện
 train_predictions_lr = model_lr.predict(X_train_lr)
 train_predictions_lr = scaler_lr.inverse_transform(train_predictions_lr)
 y_train_actual_lr = scaler_lr.inverse_transform(y_train_lr.reshape(-1, 1))

 # Dự đoán trên dữ liệu kiểm tra
 test_predictions_lr = model_lr.predict(X_eval_lr)
 test_predictions_lr = scaler_lr.inverse_transform(test_predictions_lr)
 y_eval_actual_lr = scaler_lr.inverse_transform(y_eval_lr.reshape(-1, 1))

```

```

Tính MSE và R2 cho dữ liệu huấn luyện
train_mse_lr = mean_squared_error(y_train_actual_lr, train_predictions_lr)
train_r2_lr = r2_score(y_train_actual_lr, train_predictions_lr)

Tính MSE và R2 cho dữ liệu kiểm tra
test_mse_lr = mean_squared_error(y_eval_actual_lr, test_predictions_lr)
test_r2_lr = r2_score(y_eval_actual_lr, test_predictions_lr)

Lưu kết quả
results_lr[ticker] = {
 'dates': ticker_data.iloc[:len(y_train_lr) + len(y_eval_lr)]['Date'].values,
 'actual': np.concatenate([y_train_actual_lr, y_eval_actual_lr], axis=0),
 'predicted': np.concatenate([train_predictions_lr, test_predictions_lr],
 axis=0),
 'train_mse_lr': train_mse_lr,
 'train_r2_lr': train_r2_lr,
 'test_mse_lr': test_mse_lr,
 'test_r2_lr': test_r2_lr,
 'history_lr': history_lr
}

```

b. Kết quả:

- Biểu đồ huấn luyện: Biểu đồ mô tả quá trình huấn luyện của mô hình với các giá trị MSE qua các epoch, cho phép theo dõi sự thay đổi của mức độ mảnh mát trong quá trình huấn luyện.

Code:

```
“import matplotlib.pyplot as plt
```

```

import numpy as np

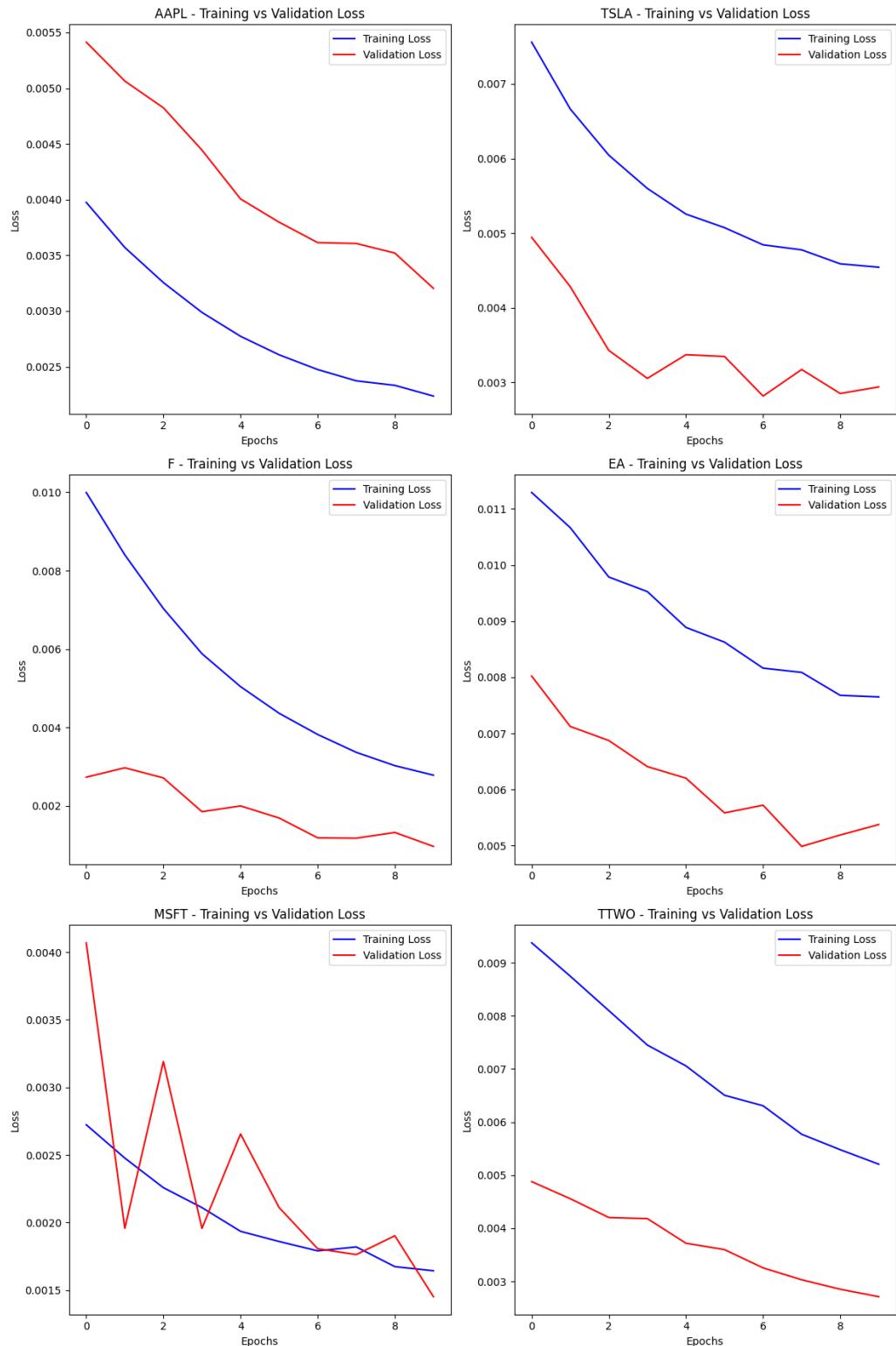
Giả sử có số lượng ticker (có thể tính n từ results_fnn)
n = len(results_lr)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột
Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))

Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()

Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_lr.items()):
 # Vẽ đồ thị huân luyện và kiểm tra loss
 history = result['history_lr']
 ax = axes[idx] # Chọn subplot tương ứng
 ax.plot(history.history['loss'], color='blue', label='Training Loss')
 ax.plot(history.history['val_loss'], color='red', label='Validation Loss')
 ax.set_title(f'{ticker} - Training vs Validation Loss')
 ax.set_xlabel('Epochs')
 ax.set_ylabel('Loss')
 ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show()

```



Hình 27. Biểu đồ huấn luyện bằng mô hình LR

- Biểu đồ kết quả dự đoán: So sánh giữa giá thực tế (Actual) và giá dự đoán (Predicted) từ mô hình đối với dữ liệu huấn luyện và kiểm tra.

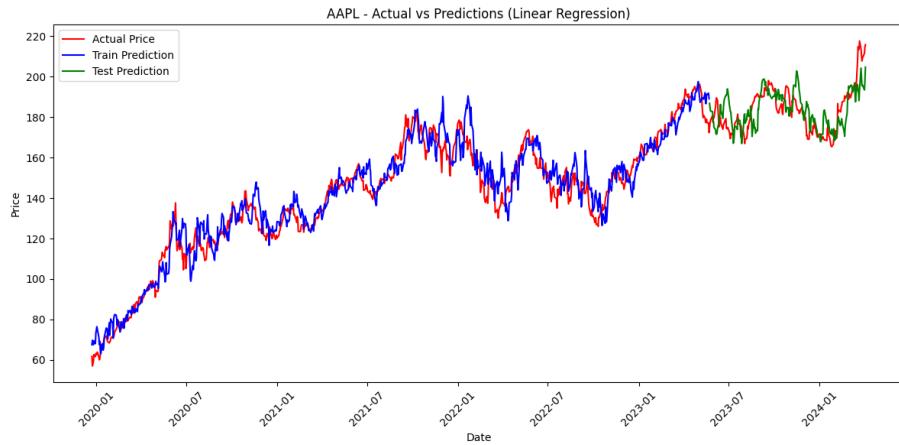
Code:

```
"# Vẽ biểu đồ kết quả
for ticker, result in results_lr.items():
 plt.figure(figsize=(12, 6))

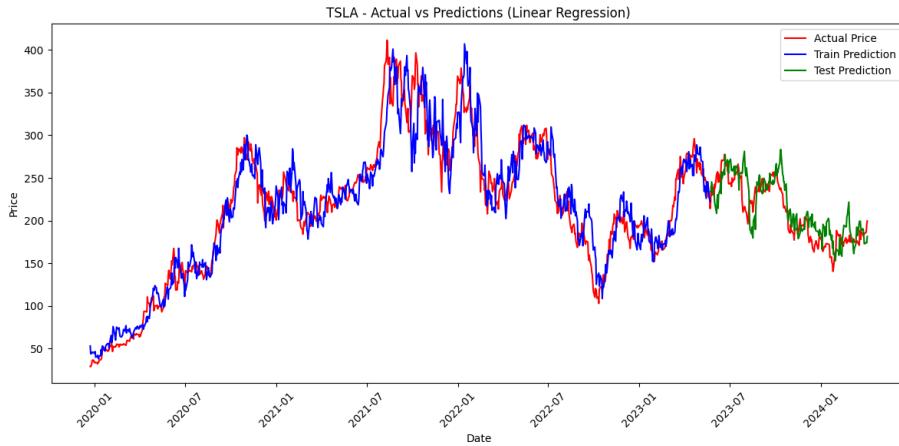
 # Vẽ đồ thị actual price (màu đỏ)
 plt.plot(result['dates'][:len(result['actual'])], result['actual'], color='red', label='Actual Price')

 # Vẽ đồ thị train và test predictions (màu xanh dương cho train, xanh lá
 # cho test)
 plt.plot(result['dates'][len(result['actual']):] - len(test_predictions_lr), result['predicted'][:len(result['actual'])] - len(test_predictions_lr), color='blue', label='Train Prediction')
 plt.plot(result['dates'][len(result['actual']):] - len(test_predictions_lr), result['predicted'][len(result['actual']):] - len(test_predictions_lr), color='green', label='Test Prediction')

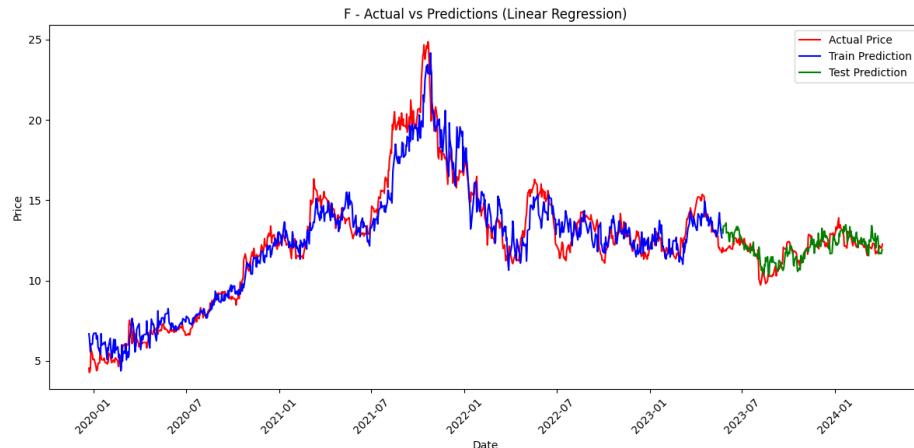
 plt.title(f'{ticker} - Actual vs Predictions (Linear Regression)')
 plt.xlabel('Date')
 plt.ylabel('Price')
 plt.xticks(rotation=45)
 plt.legend()
 plt.tight_layout()
 plt.show()"
```



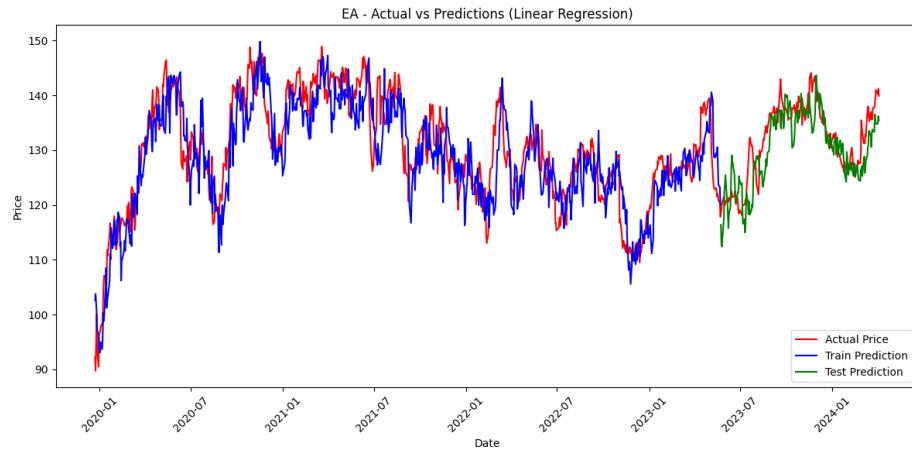
Hình 28. Kết quả dự đoán bằng mô hình LR cho ticker AAPL



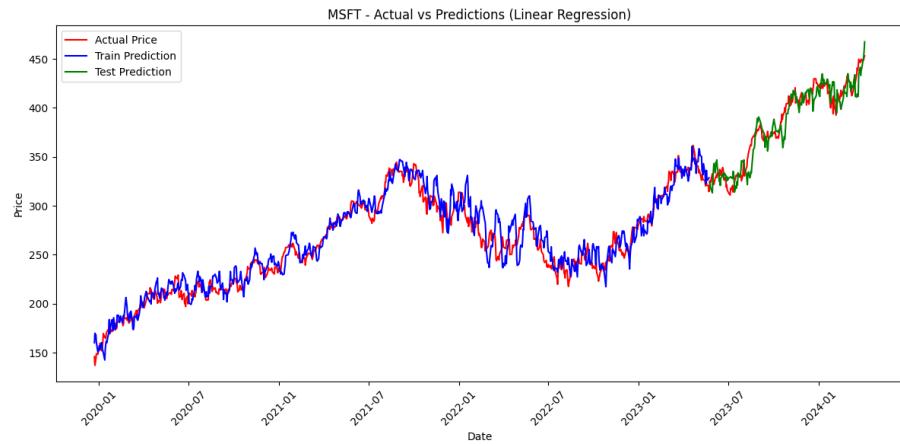
Hình 29. Kết quả dự đoán bằng mô hình LR cho ticker TSLA



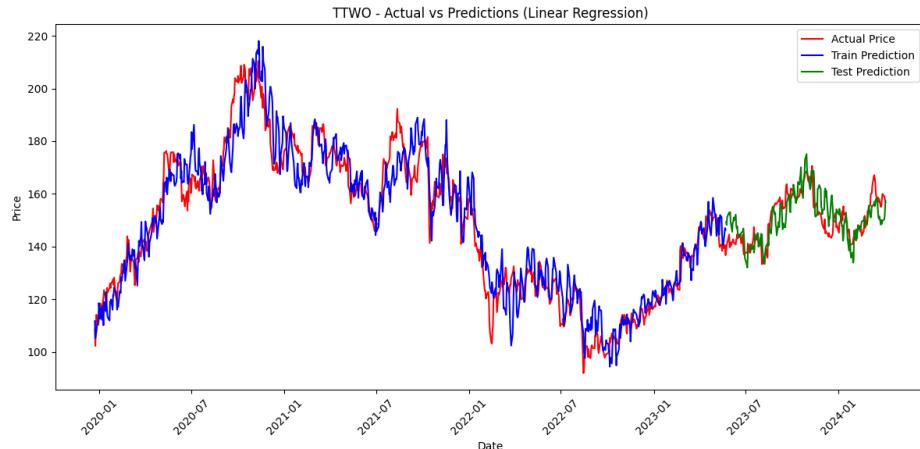
Hình 30. Kết quả dự đoán bằng mô hình LR cho ticker F



Hình 31. Kết quả dự đoán bằng mô hình LR cho ticker EA



Hình 32. Kết quả dự đoán bằng mô hình LR cho ticker MSFT



Hình 33. Kết quả dự đoán bằng mô hình LR cho ticker TTWO

- Chỉ số MSE và R<sup>2</sup>:
  - + MSE (Mean Squared Error) và R<sup>2</sup> (Coefficient of Determination) được tính cho cả dữ liệu huấn luyện và dữ liệu kiểm tra để đánh giá độ chính xác của mô hình.
  - + MSE giúp đánh giá sai lệch giữa các giá trị thực tế và dự đoán, trong khi R<sup>2</sup> cho biết tỷ lệ phương sai của dữ liệu được mô hình giải thích.

Code:

```
"for ticker, result in results_lr.items():
 print(f'{ticker} - Train MSE: {result["train_mse_lr"]}, Train R²:
{result["train_r2_lr"]}')

 print(f'{ticker} - Test MSE: {result["test_mse_lr"]}, Test R²:
{result["test_r2_lr"]}')"
```

```
AAPL - Train MSE: 57.56696592076598, Train R²: 0.9306207408813111
AAPL - Test MSE: 82.59721110680111, Test R²: 0.33836305151494617
TSLA - Train MSE: 660.1792402277188, Train R²: 0.9014095651395948
TSLA - Test MSE: 439.3073537416198, Test R²: 0.6197722117769537
F - Train MSE: 1.1220539614634182, Train R²: 0.9294173735596079
F - Test MSE: 0.4113983812590514, Test R²: 0.3548886468687318
EA - Train MSE: 26.989511833575737, Train R²: 0.7268777147552319
EA - Test MSE: 19.17877242153783, Test R²: 0.5763984142148704
MSFT - Train MSE: 169.7656248971828, Train R²: 0.9188041668667044
MSFT - Test MSE: 145.08227210431102, Test R²: 0.9033366090025656
TTWO - Train MSE: 71.04834805391697, Train R²: 0.9071569529128457
TTWO - Test MSE: 38.10767680162961, Test R²: 0.5090394958077662
```

Hình 34. Kết quả các chỉ số MSE, R2

#### 2.3.4 Decision Tree:

- a. Xây dựng mô hình:
  - Cấu trúc mô hình:
    - + Lớp đầu vào (Input layer):

- Dữ liệu đã được chuẩn hóa trước khi đưa vào mô hình. Cấu trúc đầu vào không cần lớp đặc biệt vì mô hình Decision Tree không yêu cầu cấu trúc đầu vào phức tạp.
- + Cây quyết định (Decision Tree):
  - Cây quyết định được xây dựng với thuật toán hồi quy (Regressor). Mô hình này tự động phân chia dữ liệu thành các nhánh dựa trên các đặc trưng quan trọng để đưa ra quyết định, giúp mô hình dễ dàng học các mối quan hệ phi tuyến tính giữa các biến.
- + Lớp đầu ra (Output layer):
  - Chỉ có một nút đầu ra để dự đoán giá trị liên tục (giá cổ phiếu). Mô hình này không sử dụng hàm kích hoạt vì Decision Tree trực tiếp đưa ra giá trị dự đoán.
- Trình tự huấn luyện
  - + Hàm mất mát (Loss function):
    - Mô hình sử dụng Mean Squared Error (MSE) làm hàm mất mát, giúp đánh giá mức độ sai lệch giữa giá trị thực tế và giá trị dự đoán của mô hình.
  - + Thuật toán tối ưu hóa (Optimizer):
    - Thuật toán tối ưu hóa tự động điều chỉnh các phân chia trong cây quyết định thông qua thuật toán Cây Quyết Định Hồi Quy (Decision Tree Regressor) mà không sử dụng một thuật toán tối ưu hóa như Adam.

Code:

```
"from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
```

```

Lưu kết quả cho từng mã cổ phiếu
results_dt = {}

Hàm xây dựng mô hình Decision Tree
def build_decision_tree_model():
 model_dt = DecisionTreeRegressor(random_state=42)
 return model_dt

Lặp qua từng mã cổ phiếu
for ticker in tickers:
 ticker_data_dt = data[data['Ticker'] == ticker].sort_values('Date')
 X_dt, y_dt, scaler_dt, X_train_dt, X_eval_dt, y_train_dt, y_eval_dt =
 prepare_data(ticker_data_dt)

Xây dựng mô hình Decision Tree và huấn luyện
model_dt = build_decision_tree_model()
model_dt.fit(X_train_dt, y_train_dt)

Dự đoán trên dữ liệu huấn luyện
train_predictions_dt = model_dt.predict(X_train_dt)
train_predictions_dt
scaler_dt.inverse_transform(train_predictions_dt.reshape(-1, 1))
y_train_actual = scaler_dt.inverse_transform(y_train_dt.reshape(-1, 1))

Dự đoán trên dữ liệu kiểm tra
test_predictions_dt = model_dt.predict(X_eval_dt)

```

```

test_predictions_dt =
scaler_dt.inverse_transform(test_predictions_dt.reshape(-1, 1))

y_eval_actual_dt = scaler_dt.inverse_transform(y_eval_dt.reshape(-1, 1))

Tính MSE và R2 cho dữ liệu huấn luyện
train_mse_dt = mean_squared_error(y_train_actual, train_predictions_dt)
train_r2_dt = r2_score(y_train_actual, train_predictions_dt)

Tính MSE và R2 cho dữ liệu kiểm tra
test_mse_dt = mean_squared_error(y_eval_actual_dt, test_predictions_dt)
test_r2_dt = r2_score(y_eval_actual_dt, test_predictions_dt)

Lưu kết quả và tính loss
train_loss_dt = [train_mse_dt] * len(y_train_dt) # Dùng MSE làm loss cho
huấn luyện
test_loss_dt = [test_mse_dt] * len(y_eval_dt) # Dùng MSE làm loss cho
kiểm tra

results_dt[ticker] = {
 'dates': ticker_data.iloc[:len(y_train_dt)] +
 len(y_eval_dt)][['Date']].values,
 'actual': np.concatenate([y_train_actual, y_eval_actual_dt], axis=0),
 'predicted': np.concatenate([train_predictions_dt, test_predictions_dt],
 axis=0),
 'train_mse_dt': train_mse_dt,
 'train_r2_dt': train_r2_dt,
 'test_mse_dt': test_mse_dt,
}

```

```
'test_r2_dt': test_r2_dt,
'train_loss_dt': train_loss_dt,
'test_loss_dt': test_loss_dt,
}'”
```

b. Kết quả:

- Biểu đồ huấn luyện: Mô hình Decision Tree không có quá trình huấn luyện như trong các mạng nơ-ron, nhưng có thể theo dõi loss (MSE) cho dữ liệu huấn luyện và kiểm tra. Độ lường này giúp đánh giá độ chính xác của mô hình qua các dự đoán của cây quyết định.

Code:

```
“# Giả sử có số lượng ticker (có thể tính n từ results_dt)
n = len(results_dt)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột
```

```
Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))
```

```
Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()
```

```
Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_dt.items()):
 # Vẽ đồ thị huấn luyện và kiểm tra loss
 train_loss = result['train_loss_dt']
 test_loss = result['test_loss_dt']
```

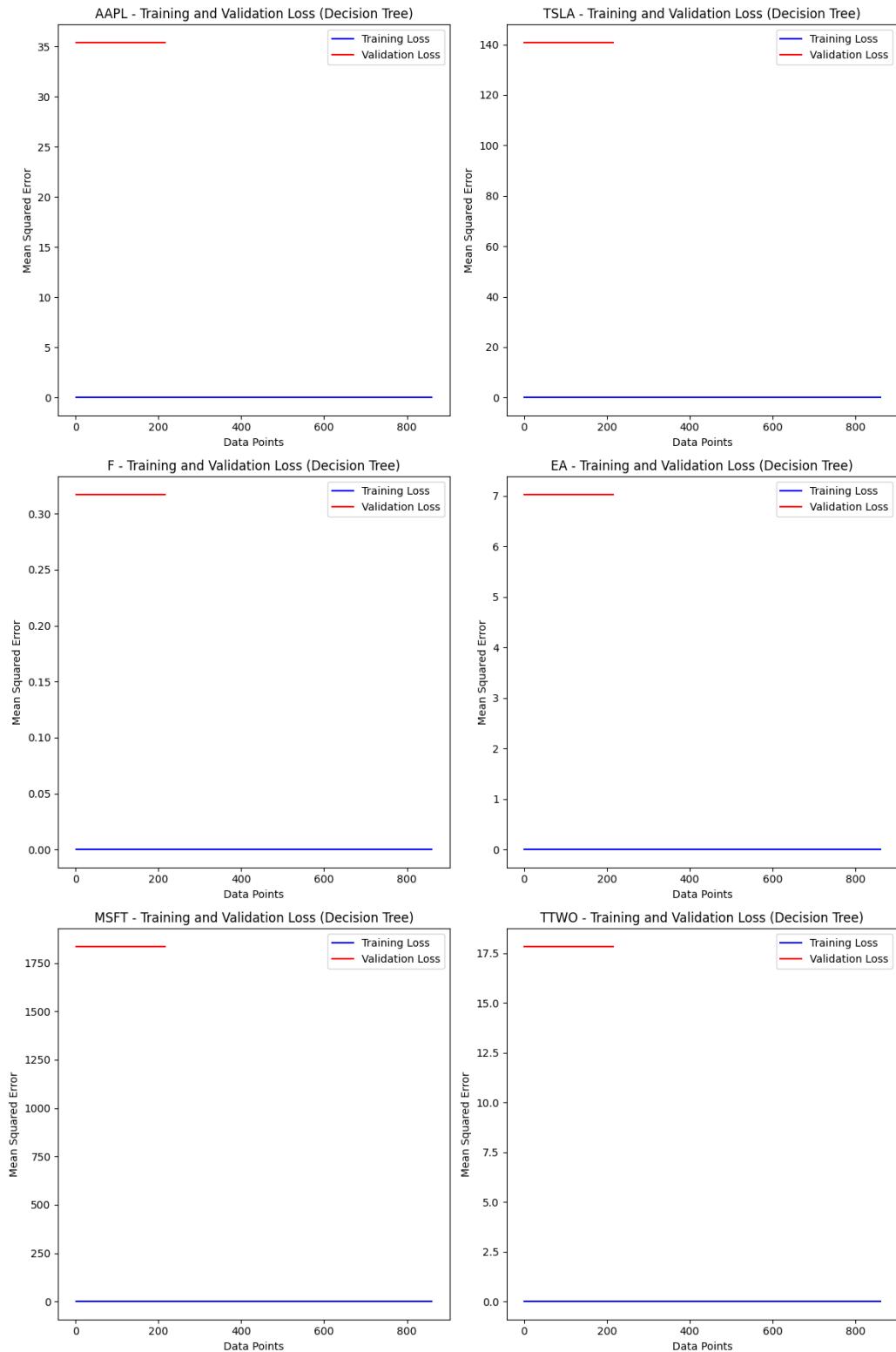
```
ax = axes[idx] # Chọn subplot trong ứng

ax.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss',
color='blue')

ax.plot(range(1, len(test_loss) + 1), test_loss, label='Validation Loss',
color='red')

ax.set_xlabel('Data Points')
ax.set_ylabel('Mean Squared Error')
ax.set_title(f'{ticker} - Training and Validation Loss (Decision Tree)')
ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show()
```



Hình 35. Biểu đồ huấn luyện bằng mô hình DT

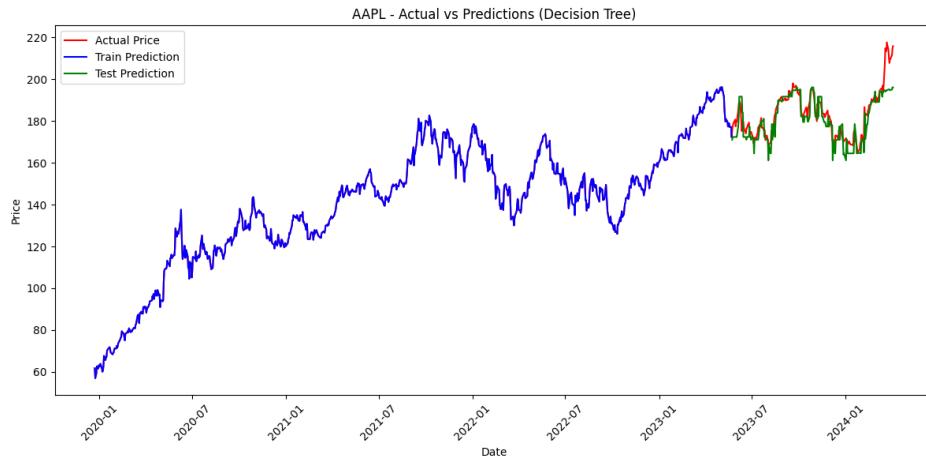
- Biểu đồ kết quả dự đoán: Biểu đồ so sánh giữa Actual Price (màu đỏ) và Predicted Price (màu xanh dương cho dữ liệu huấn luyện, màu xanh lá cho dữ liệu kiểm tra) cho thấy mức độ khớp giữa giá trị thực tế và dự đoán của mô hình.

Code:

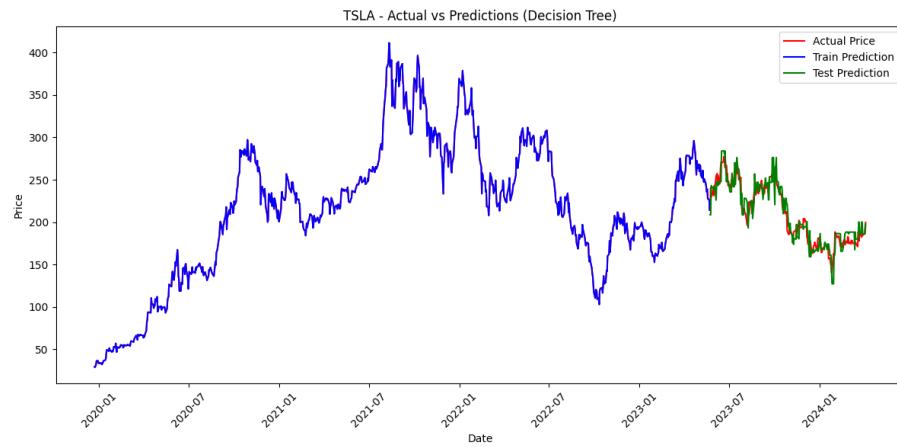
```
"# Vẽ biểu đồ kết quả
for ticker, result in results_dt.items():
 plt.figure(figsize=(12, 6))
 # Vẽ đồ thị actual price (màu đỏ)
 plt.plot(result['dates'][:len(result['actual'])], result['actual'], color='red', label='Actual Price')
 # Vẽ đồ thị train và test predictions (màu xanh dương cho train, xanh lá
 # cho test)
 plt.plot(result['dates'][len(result['actual']):] - len(test_predictions_dt), result['predicted'][:len(result['actual'])] - len(test_predictions_dt), color='blue', label='Train Prediction')
 plt.plot(result['dates'][len(result['actual']):] - len(test_predictions_dt), result['predicted'][len(result['actual']):] - len(test_predictions_dt), color='green', label='Test Prediction')

 plt.title(f'{ticker} - Actual vs Predictions (Decision Tree)')
 plt.xlabel('Date')
 plt.ylabel('Price')
 plt.xticks(rotation=45)
 plt.legend()

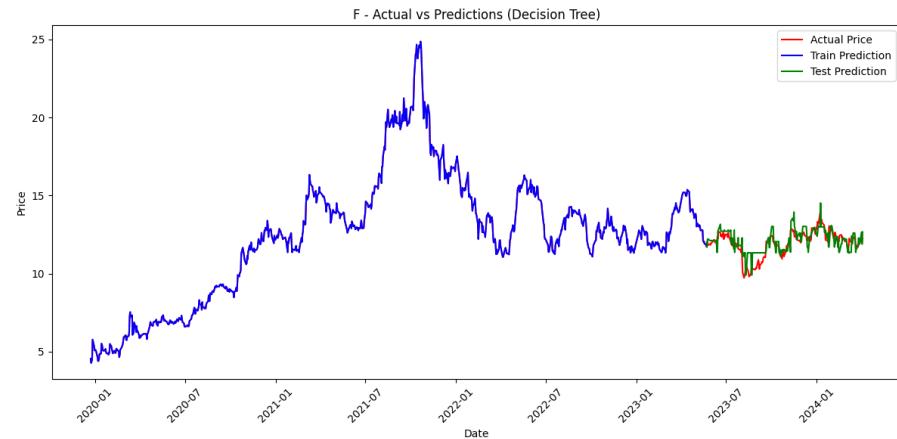
 plt.tight_layout()
plt.show()"
```



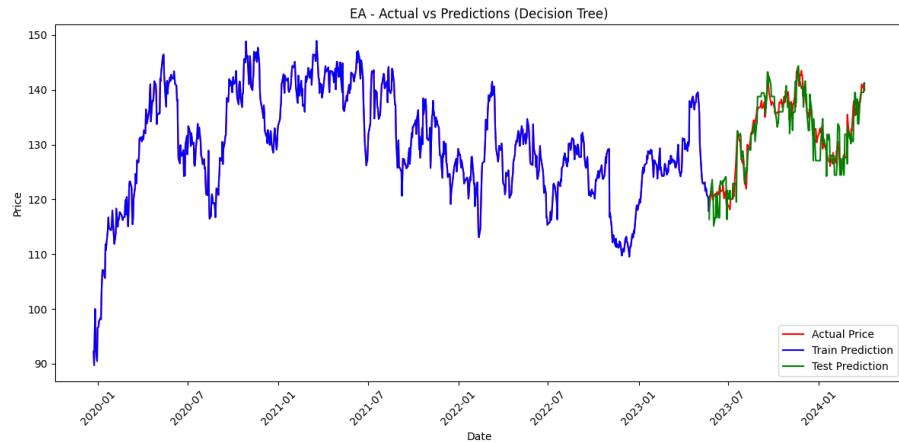
Hình 36. Kết quả dự đoán bằng mô hình DT cho ticker AAPL



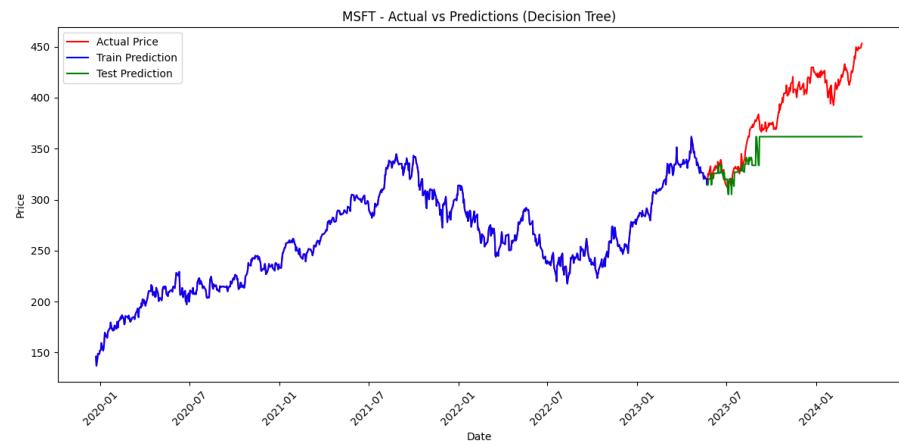
Hình 37. Kết quả dự đoán bằng mô hình DT cho ticker TSLA



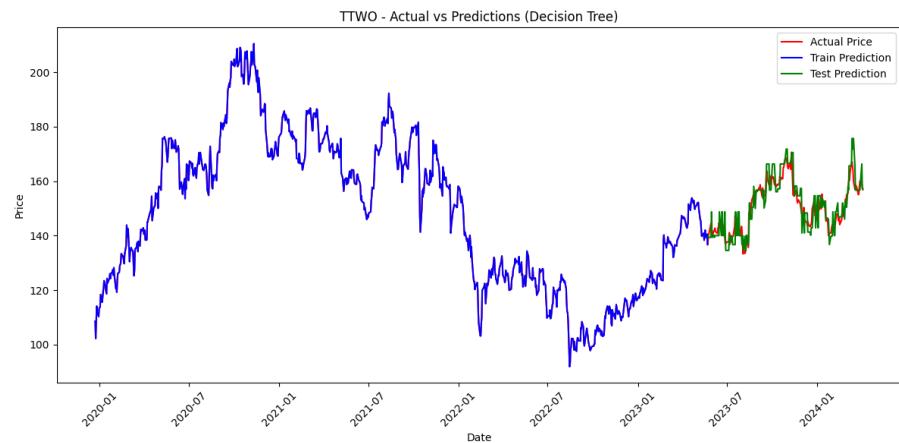
Hình 38. Kết quả dự đoán bằng mô hình DT cho ticker F



Hình 39. Kết quả dự đoán bằng mô hình DT cho ticker EA



Hình 40. Kết quả dự đoán bằng mô hình DT cho ticker MSFT



Hình 41. Kết quả dự đoán bằng mô hình DT cho ticker TTWO

- Chỉ số MSE và R<sup>2</sup>
  - + MSE (Mean Squared Error): MSE đánh giá sai lệch giữa giá trị thực tế và dự đoán. MSE thấp cho thấy mô hình có độ chính xác cao.
  - + R<sup>2</sup> (Coefficient of Determination): R<sup>2</sup> cho biết tỷ lệ phương sai của dữ liệu được mô hình giải thích. Giá trị R<sup>2</sup> cao cho thấy mô hình giải thích tốt dữ liệu. Mô hình Decision Tree sẽ cho phép xác định mức độ phù hợp với dữ liệu trong từng phân nhánh.

Code:

```
"# In ra MSE và R2
for ticker, result in results_dt.items():
 print(f'{ticker} - Train MSE: {result["train_mse_dt"]}, Train R2: {result["train_r2_dt"]}')
 print(f'{ticker} - Test MSE: {result["test_mse_dt"]}, Test R2: {result["test_r2_dt"]}'")
```

```
AAPL - Train MSE: 0.0, Train R2: 1.0
AAPL - Test MSE: 35.410104542354134, Test R2: 0.716350791981997
TSLA - Train MSE: 0.0, Train R2: 1.0
TSLA - Test MSE: 140.84226123136366, Test R2: 0.878098690995665
F - Train MSE: 0.0, Train R2: 1.0
F - Test MSE: 0.3172347370113709, Test R2: 0.5025460969794595
EA - Train MSE: 0.0, Train R2: 1.0
EA - Test MSE: 7.027809157394115, Test R2: 0.8447767647357747
MSFT - Train MSE: 0.0, Train R2: 1.0
MSFT - Test MSE: 1835.6085659554842, Test R2: -0.22300365134633493
TTWO - Train MSE: 0.0, Train R2: 1.0
TTWO - Test MSE: 17.84552003074916, Test R2: 0.7700871255553803
```

Hình 42. Kết quả các chỉ số MSE, R2

## 2.4 Áp dụng các kỹ thuật để giảm hiện tượng quá khóp (Overfitting):

### 2.4.1 Feedforward Neural Network:

- Phương pháp ngăn ngừa quá khóp(Overfitting) được áp dụng:

- + L2 Regularization: Áp dụng L2 regularization cho các lớp Dense để giảm độ lớn của trọng số, giúp mô hình tổng quát hơn và tránh overfitting.
- + Dropout: Sử dụng Dropout với tỷ lệ 20% để ngẫu nhiên "tắt" một số nút trong mô hình, ngăn ngừa việc mô hình quá phụ thuộc vào các đặc trưng cụ thể.
- + Early Stopping: Dừng huấn luyện nếu mất mát trên tập validation không cải thiện trong 10 epoch liên tiếp, tránh mô hình học quá mức và overfitting.

Code:

```

“import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras import regularizers
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã phiếu
results_fnno = {}

Hàm xây dựng mô hình Feedforward Neural Network (FNN) với các kỹ thuật
ngăn chặn overfitting
def build_fnno_model(input_shape):
 model_fnno = Sequential()
 # Sử dụng Input layer thay vì input_dim
 model_fnno.add(Input(shape=(input_shape,))) # Lớp Input
 model_fnno.add(Dense(64, activation='relu',
 kernel_regularizer=regularizers.l2(0.01))) # L2 Regularization
 model_fnno.add(Dropout(0.2)) # Dropout layer
 model_fnno.add(Dense(32, activation='relu',
 kernel_regularizer=regularizers.l2(0.01))) # L2 Regularization

```

```

model_fnno.add(Dropout(0.2)) # Dropout layer
model_fnno.add(Dense(1)) # Output layer
model_fnno.compile(optimizer='adam', loss='mean_squared_error')
return model_fnno

Lắp qua từng mã cổ phiếu
for ticker in tickers:
 ticker_data_fnno = data[data['Ticker'] == ticker].sort_values('Date')
 X_fnno, y_fnno, scaler_fnno, X_train_fnno, X_eval_fnno, y_train_fnno,
 y_eval_fnno = prepare_data(ticker_data_fnno)

 # Xây dựng mô hình FNN và huấn luyện
 model_fnno = build_fnn_model(X_train_fnno.shape[1])

 # Huấn luyện mô hình với Early Stopping
 history_fnno = model_fnno.fit(X_train_fnno, y_train_fnno, epochs=100,
 batch_size=32,
 validation_data=(X_eval_fnno, y_eval_fnno),
 verbose=0,
 callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss',
 patience=10)])

 # Dự đoán trên tập train và test
 y_train_pred_fnno = model_fnno.predict(X_train_fnno)
 y_eval_pred_fnno = model_fnno.predict(X_eval_fnno)

 # Tính MSE và R2 cho train và test

```

```

mse_train_fnno = mean_squared_error(y_train_fnno, y_train_pred_fnno)
mse_test_fnno = mean_squared_error(y_eval_fnno, y_eval_pred_fnno)
r2_train_fnno = r2_score(y_train_fnno, y_train_pred_fnno)
r2_test_fnno = r2_score(y_eval_fnno, y_eval_pred_fnno)

Lưu kết quả
results_fnno[ticker] = {
 'history': history_fnno,
 'scaler': scaler_fnno,
 'model': model_fnno,
 'mse_train_fnno': mse_train_fnno,
 'mse_test_fnno': mse_test_fnno,
 'r2_train_fnno': r2_train_fnno,
 'r2_test_fnno': r2_test_fnno
}

```

- Vẽ biểu đồ huấn luyện mới: giúp theo dõi quá trình huấn luyện và kiểm tra, xác định khi nào dừng huấn luyện.

Code:

```

“import matplotlib.pyplot as plt
import numpy as np

```

```

Giả sử có số lượng ticker (có thể tính n từ results_fnno)
n = len(results_fnno)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột

```

```

Tạo subplots với số hàng và số cột

```

```

fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))

Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()

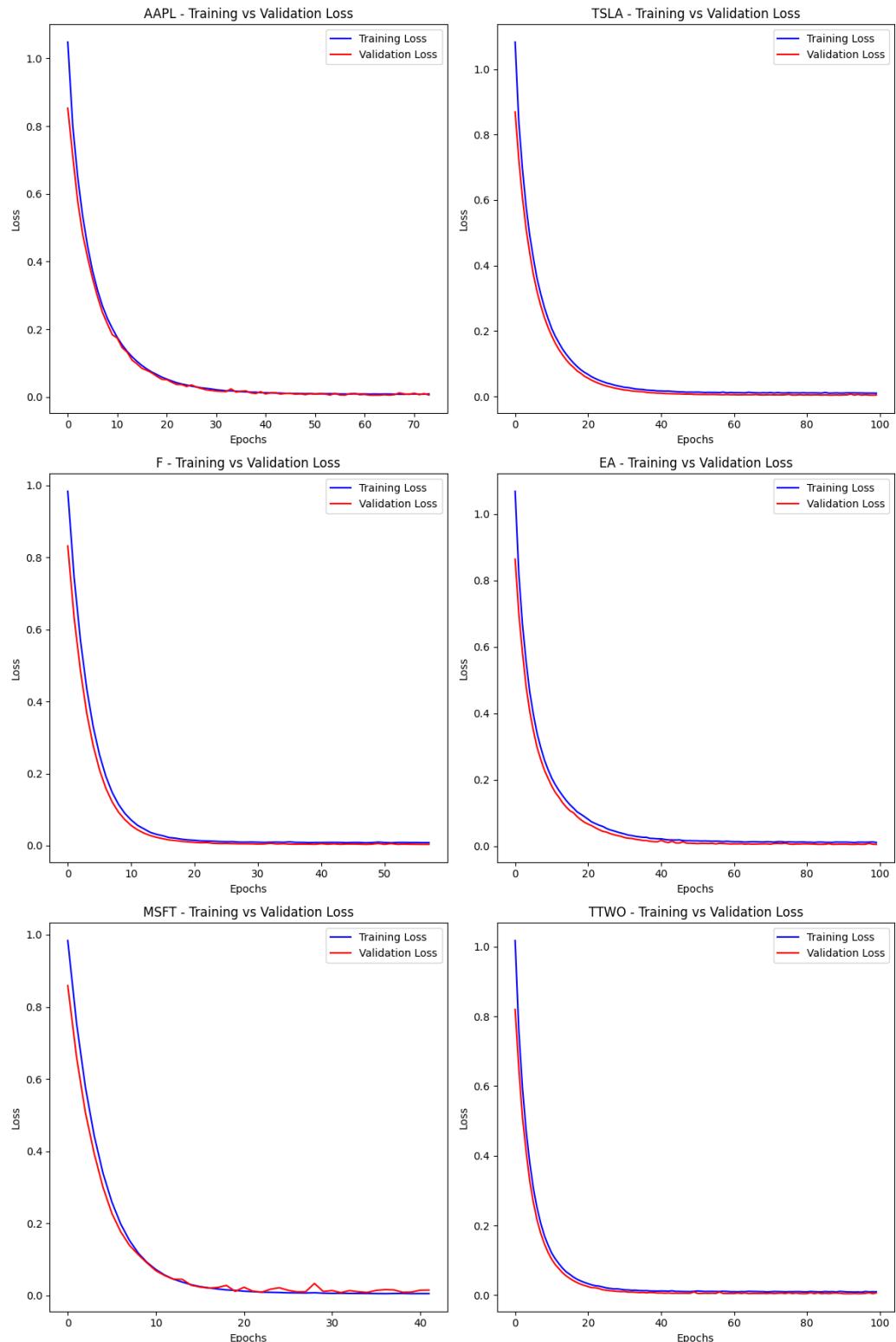
Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_fnno.items()):
 history = result['history']

 ax = axes[idx] # Chọn subplot tương ứng

 ax.plot(history.history['loss'], color='blue', label='Training Loss')
 ax.plot(history.history['val_loss'], color='red', label='Validation Loss')
 ax.set_title(f'{ticker} - Training vs Validation Loss')
 ax.set_xlabel('Epochs')
 ax.set_ylabel('Loss')
 ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show()

```



Hình 43. Biểu đồ huấn luyện bằng mô hình FNN sau khi overfitting

- Vẽ biểu đồ kết quả: Biểu đồ thể hiện kết quả giá dự đoán mới so với giá thực tế sau khi thực hiện các biện pháp giảm quá khớp(Overfitting).

Code:

```
"for ticker, result in results_fnno.items():
 # Lấy giá trị thực và giá trị dự đoán
 y_train_actual_fnno =
 result['scaler'].inverse_transform(y_train_fnno.reshape(-1, 1)).flatten()
 =
 y_train_pred_fnno
 =
 result['scaler'].inverse_transform(result['model'].predict(X_train_fnno).flatten()
 .reshape(-1, 1)).flatten()

 y_eval_actual_fnno =
 result['scaler'].inverse_transform(y_eval_fnno.reshape(-1, 1)).flatten()
 =
 y_eval_pred_fnno
 =
 result['scaler'].inverse_transform(result['model'].predict(X_eval_fnno).flatten()
 .reshape(-1, 1)).flatten()

 # Chia dữ liệu ngày tháng cho tập huấn luyện và kiểm tra
 train_dates_fnno =
 ticker_data_fnno['Date'].iloc[:len(y_train_actual_fnno)].values
 =
 test_dates_fnno
 =
 ticker_data_fnno['Date'].iloc[len(y_train_actual_fnno):len(y_train_actual_fnno
) + len(y_eval_actual_fnno)].values

 # Kết hợp dữ liệu thực tế
 actual_dates_fnno = np.concatenate([train_dates_fnno, test_dates_fnno])
```

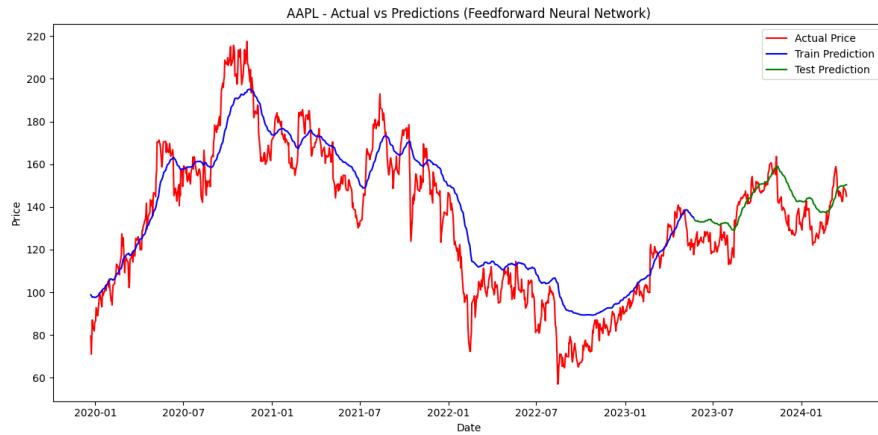
```

actual_prices_fnno = np.concatenate([y_train_actual_fnno,
y_eval_actual_fnno])

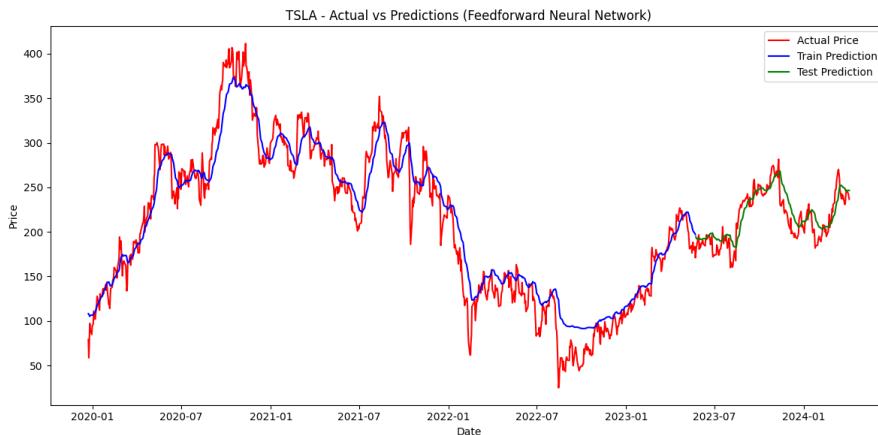
Vẽ đồ thị giá trị thực tế và dự đoán
plt.figure(figsize=(12, 6))
plt.plot(actual_dates_fnno, actual_prices_fnno, color='red', label='Actual
Price')
plt.plot(train_dates_fnno, y_train_pred_fnno, color='blue', label='Train
Prediction')
plt.plot(test_dates_fnno, y_eval_pred_fnno, color='green', label='Test
Prediction')
plt.title(f'{ticker} - Actual vs Predictions (Feedforward Neural Network)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.tight_layout()
plt.show()

In thông tin MSE và R2
print(f'{ticker} - MSE Train: {result["mse_train_fnno"]}, MSE Test:
{result["mse_test_fnno"]}')
print(f'{ticker} - R2 Train: {result["r2_train_fnno"]}, R2 Test:
{result["r2_test_fnno"]}'")

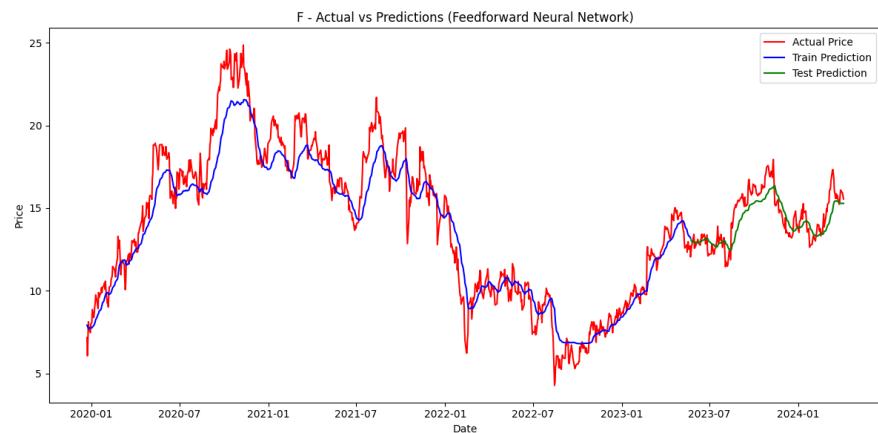
```



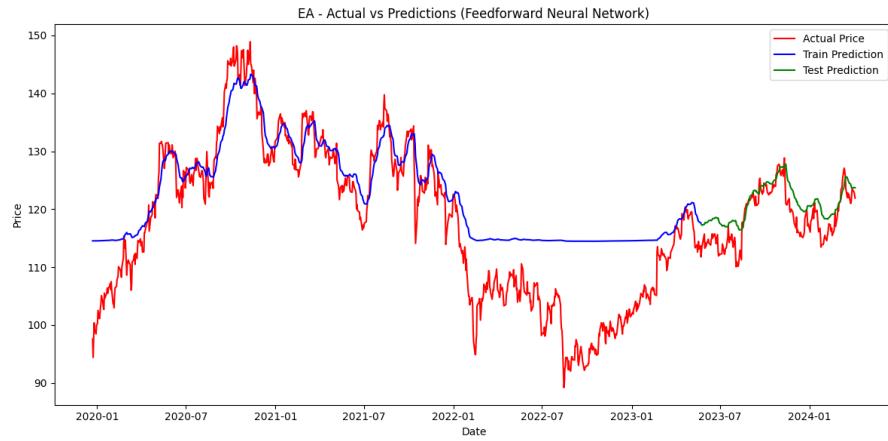
Hình 44. Kết quả dự đoán bằng mô hình FNN cho ticker AAPL sau khi overfitting



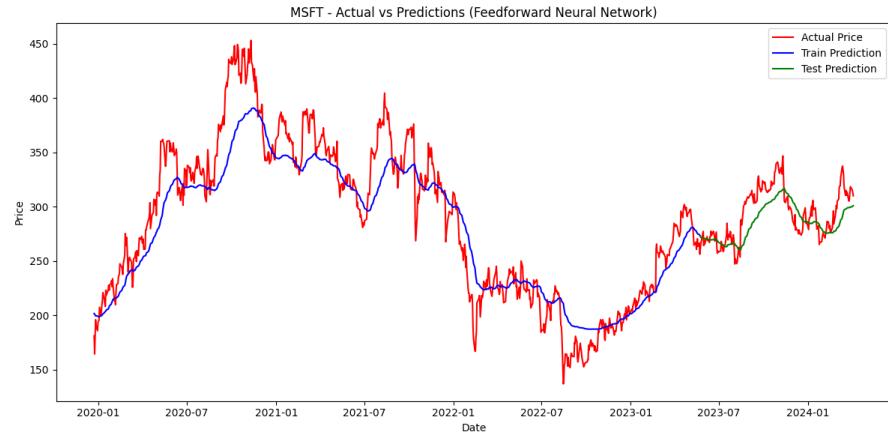
Hình 45. Kết quả dự đoán bằng mô hình FNN cho ticker TSLA sau khi overfitting



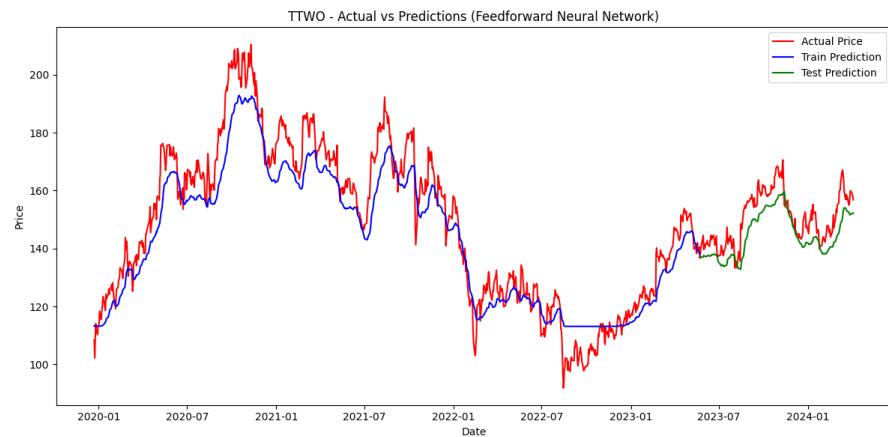
Hình 46. Kết quả dự đoán bằng mô hình FNN cho ticker F sau khi overfitting



Hình 47. Kết quả dự đoán bằng mô hình FNN cho ticker EA sau khi overfitting



Hình 48. Kết quả dự đoán bằng mô hình FNN cho ticker MSFT sau khi overfitting



Hình 49. Kết quả dự đoán bằng mô hình FNN cho ticker TTWO sau khi overfitting

### 2.4.2 Recurrent Neural Network (RNN):

- Phương pháp ngăn ngừa quá khớp(Overfitting) được áp dụng:
  - + L2 Regularization (Kernel Regularizer): Được áp dụng cho các lớp LSTM với tham số kernel\_regularizer=regularizers.l2(0.01). L2 regularization giúp giảm độ lớn của các trọng số, từ đó ngăn chặn mô hình học quá mức các chi tiết không quan trọng trong dữ liệu, làm giảm khả năng overfitting.
  - + Dropout: Dropout được sử dụng với tỷ lệ 20% (Dropout(0.2)) sau mỗi lớp LSTM. Kỹ thuật này ngẫu nhiên "tắt" một số đơn vị trong mạng, giúp ngăn chặn mô hình quá phụ thuộc vào các đặc trưng cụ thể và cải thiện khả năng tổng quát.
  - + Early Stopping: Early stopping giúp dừng huấn luyện khi giá trị mất mát trên tập validation không cải thiện sau một số epoch nhất định (ở đây là 10 epoch). Điều này ngăn ngừa việc mô hình học quá mức và overfitting bằng cách dừng sớm quá trình huấn luyện khi không có sự cải thiện đáng kể.

Code:

```
"from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input
from tensorflow.keras import regularizers
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã cổ phiếu
results_rnno = {}

Hàm xây dựng mô hình RNN với các kỹ thuật ngăn chặn overfitting
def build_rnn_model(input_shape):
```

```

model_rnno = Sequential()
Sử dụng Input layer cho dữ liệu chuỗi (time-series)
model_rnno.add(Input(shape=(input_shape, 1))) # Lớp Input
model_rnno.add(LSTM(50, return_sequences=True,
 kernel_regularizer=regularizers.l2(0.01))) # LSTM với L2
Regularization
model_rnno.add(Dropout(0.2)) # Dropout layer
model_rnno.add(LSTM(50, kernel_regularizer=regularizers.l2(0.01))) # LSTM với L2 Regularization
model_rnno.add(Dropout(0.2)) # Dropout layer
model_rnno.add(Dense(1)) # Output layer
model_rnno.compile(optimizer='adam', loss='mean_squared_error')
return model_rnno

Lặp qua từng mã cổ phiếu
for ticker in tickers:
 ticker_data_rnno = data[data['Ticker'] == ticker].sort_values('Date')
 X_rnno, y_rnno, scaler_rnno, X_train_rnno, X_eval_rnno, y_train_rnno,
 y_eval_rnno = prepare_data(ticker_data_rnno)

 # Xây dựng mô hình RNN và huấn luyện
 model_rnno = build_rnn_model(X_train_rnno.shape[1])

 # Huấn luyện mô hình với Early Stopping
 history_rnno = model_rnno.fit(X_train_rnno, y_train_rnno, epochs=100,
 batch_size=32,
 validation_data=(X_eval_rnno, y_eval_rnno),

```

```

verbose=0,
callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)])
```

# Dự đoán trên tập train và test

```

y_train_pred_rnno = model_rnno.predict(X_train_rnno)
y_eval_pred_rnno = model_rnno.predict(X_eval_rnno)
```

# Tính MSE và R<sup>2</sup> cho train và test

```

mse_train_rnno = mean_squared_error(y_train_rnno, y_train_pred_rnno)
mse_test_rnno = mean_squared_error(y_eval_rnno, y_eval_pred_rnno)
r2_train_rnno = r2_score(y_train_rnno, y_train_pred_rnno)
r2_test_rnno = r2_score(y_eval_rnno, y_eval_pred_rnno)
```

# Lưu kết quả

```

results_rnno[ticker] = {
 'history': history_rnno,
 'scaler': scaler_rnno,
 'model': model_rnno,
 'mse_train_rnno': mse_train_rnno,
 'mse_test_rnno': mse_test_rnno,
 'r2_train_rnno': r2_train_rnno,
 'r2_test_rnno': r2_test_rnno
}"
```

- Vẽ biểu đồ huấn luyện mới: giúp theo dõi quá trình huấn luyện và kiểm tra, xác định khi nào dừng huấn luyện.

Code:

```

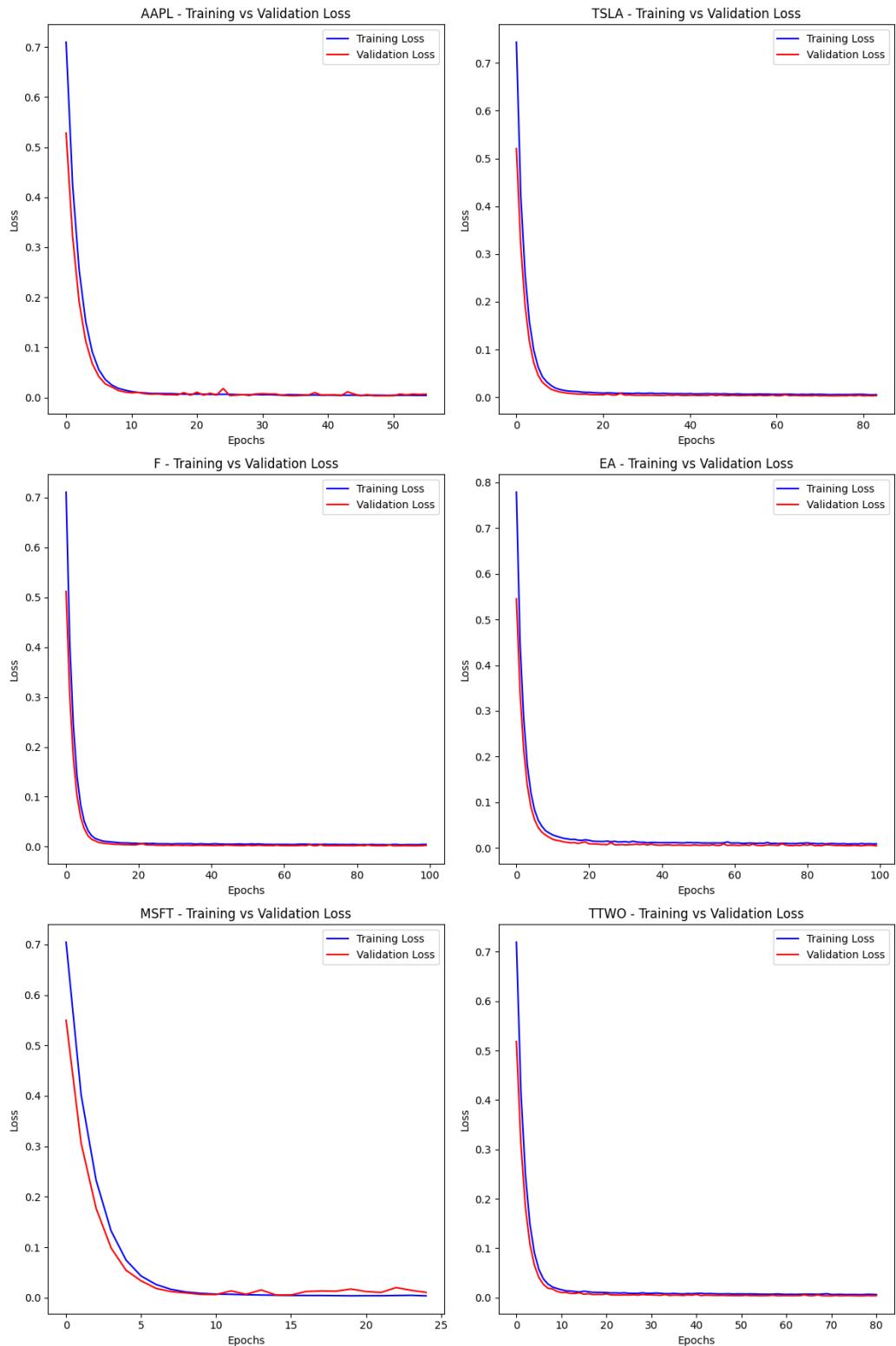
“import matplotlib.pyplot as plt
import numpy as np

Giả sử có số lượng ticker (có thể tính n từ results_rnno)
n = len(results_rnno)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột
Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))
Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()
Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_rnno.items()):
 history = result['history']

 ax = axes[idx] # Chọn subplot tương ứng
 ax.plot(history.history['loss'], color='blue', label='Training Loss')
 ax.plot(history.history['val_loss'], color='red', label='Validation Loss')
 ax.set_title(f'{ticker} - Training vs Validation Loss')
 ax.set_xlabel('Epochs')
 ax.set_ylabel('Loss')
 ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show() ”

```



Hình 50. Biểu đồ huấn luyện bằng mô hình RNN sau khi overfitting

- Vẽ biểu đồ kết quả: Biểu đồ thể hiện kết quả giá dự đoán mới so với giá thực tế sau khi thực hiện các biện pháp giảm quá khớp(Overfitting).

Code:

```
"for ticker, result in results_rnno.items():
 # Lấy giá trị thực và giá trị dự đoán
 y_train_actual_rnno =
 result['scaler'].inverse_transform(y_train_rnno.reshape(-1, 1)).flatten()
 =
 y_train_pred_rnno
 =
 result['scaler'].inverse_transform(result['model'].predict(X_train_rnno).flatten()
 .reshape(-1, 1)).flatten()

 y_eval_actual_rnno =
 result['scaler'].inverse_transform(y_eval_rnno.reshape(-1, 1)).flatten()
 =
 y_eval_pred_rnno
 =
 result['scaler'].inverse_transform(result['model'].predict(X_eval_rnno).flatten()
 .reshape(-1, 1)).flatten()

 # Chia dữ liệu ngày tháng cho tập huấn luyện và kiểm tra
 train_dates_rnno =
 ticker_data_rnno['Date'].iloc[:len(y_train_actual_rnno)].values
 =
 test_dates_rnno
 =
 ticker_data_rnno['Date'].iloc[len(y_train_actual_rnno):len(y_train_actual_rnno) + len(y_eval_actual_rnno)].values

 # Kết hợp dữ liệu thực tế
 actual_dates_rnno = np.concatenate([train_dates_rnno, test_dates_rnno])
```

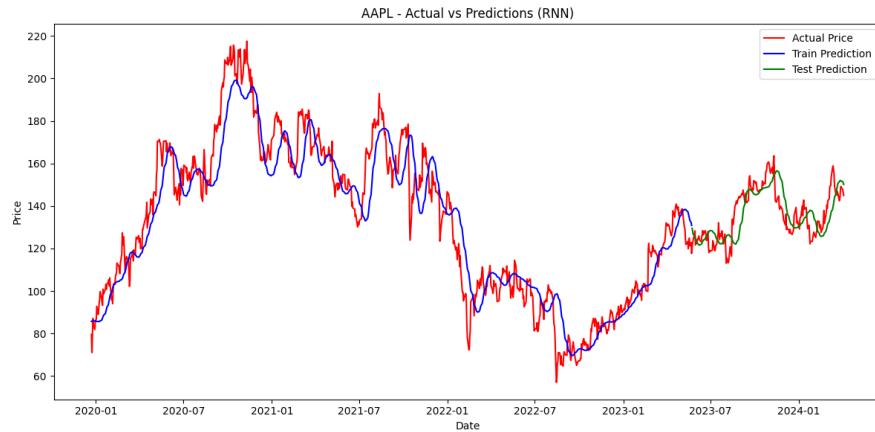
```

actual_prices_rnno = np.concatenate([y_train_actual_rnno,
y_eval_actual_rnno])

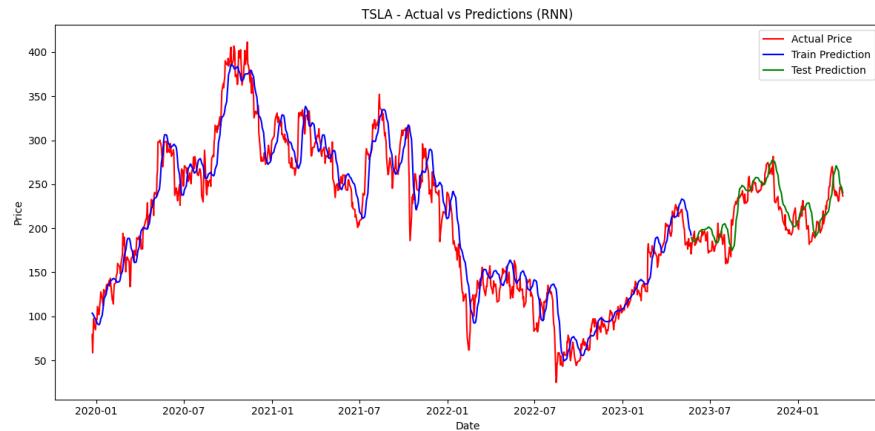
Vẽ đồ thị giá trị thực tế và dự đoán
plt.figure(figsize=(12, 6))
plt.plot(actual_dates_rnno, actual_prices_rnno, color='red', label='Actual
Price')
plt.plot(train_dates_rnno, y_train_pred_rnno, color='blue', label='Train
Prediction')
plt.plot(test_dates_rnno, y_eval_pred_rnno, color='green', label='Test
Prediction')
plt.title(f'{ticker} - Actual vs Predictions (RNN)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.tight_layout()
plt.show()

In thông tin MSE và R2
print(f'{ticker} - MSE Train: {result["mse_train_rnno"]}, MSE Test:
{result["mse_test_rnno"]}')
print(f'{ticker} - R2 Train: {result["r2_train_rnno"]}, R2 Test:
{result["r2_test_rnno"]}')

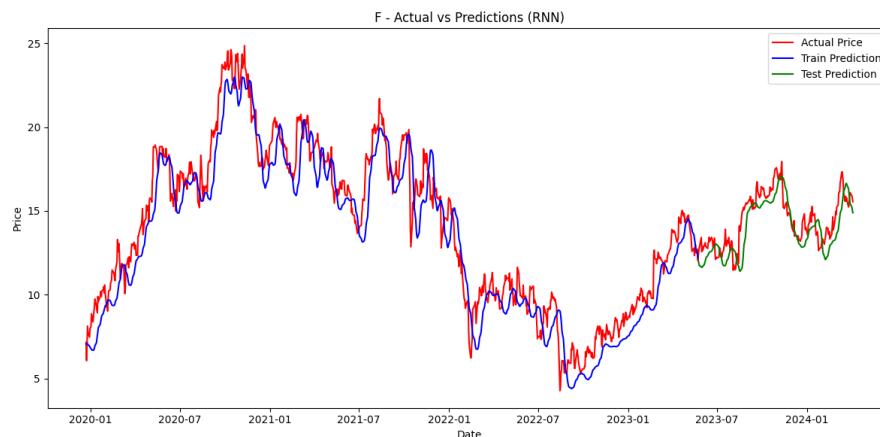
```



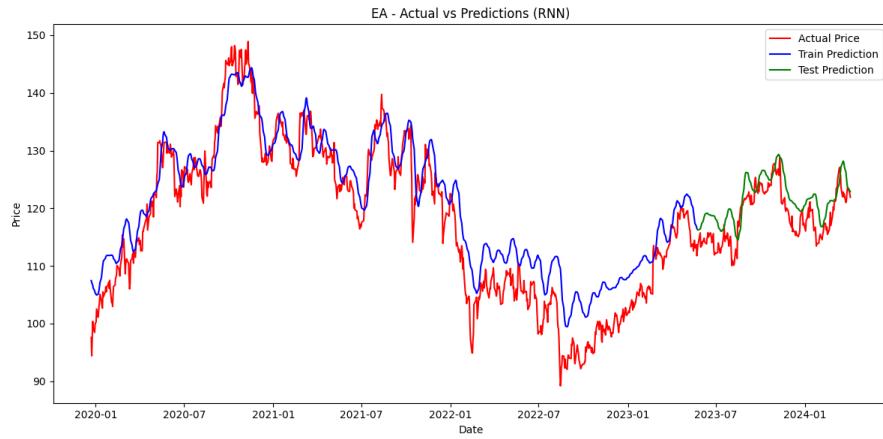
Hình 51. Kết quả dự đoán bằng mô hình RNN cho ticker AAPL sau khi overfitting



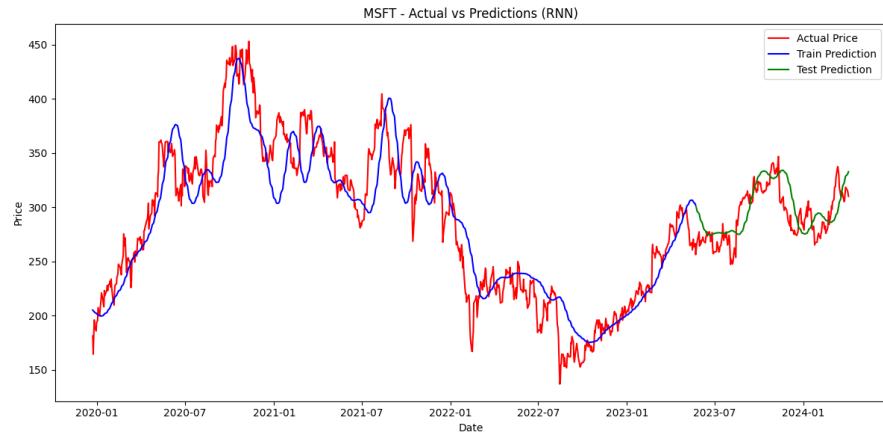
Hình 52. Kết quả dự đoán bằng mô hình RNN cho ticker TSLA sau khi overfitting



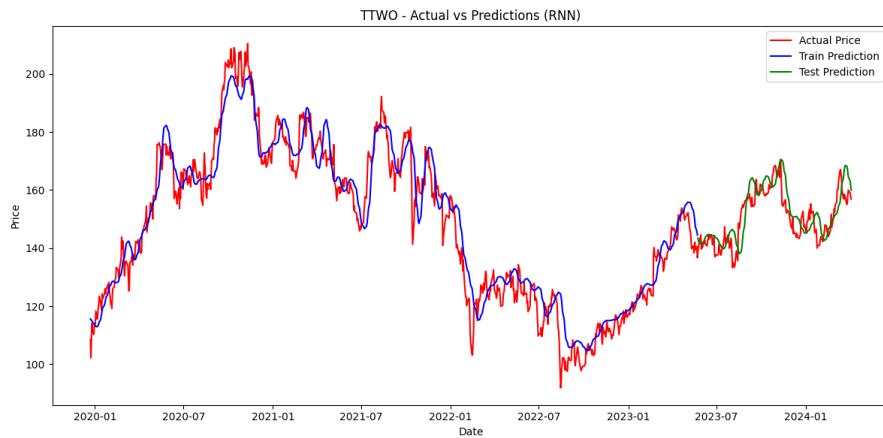
Hình 53. Kết quả dự đoán bằng mô hình RNN cho ticker F sau khi overfitting



Hình 54. Kết quả dự đoán bằng mô hình RNN cho ticker EA sau khi overfitting



Hình 55. Kết quả dự đoán bằng mô hình RNN cho ticker MSFT sau khi overfitting



Hình 56. Kết quả dự đoán bằng mô hình RNN cho ticker TTWO sau khi overfitting

### 2.4.3 Linear Regression:

- Phương pháp ngăn ngừa quá khớp(Overfitting) được áp dụng:
  - + L2 Regularization: L2 regularization được áp dụng trong lớp Dense của mô hình hồi quy tuyến tính (kernel\_regularizer=regularizers.l2(0.01)). Phương pháp này giúp ngăn ngừa overfitting bằng cách thêm một hình phạt vào hàm mất mát, khuyến khích các trọng số của mô hình có giá trị nhỏ hơn và tránh mô hình học quá chi tiết vào dữ liệu huấn luyện.
  - + Early Stopping: Early stopping được sử dụng thông qua callback EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True). Phương pháp này giúp dừng quá trình huấn luyện nếu mất mát trên tập validation không cải thiện trong một số epoch nhất định (10 epoch). Điều này giúp tránh việc mô hình học quá mức và overfit vào dữ liệu huấn luyện.

Code:

```
"from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã phiếu
results_lro = {}
```

```
Hàm xây dựng mô hình Hồi quy tuyến tính với L2 Regularization
def build_linear_regression_model(input_shape):
 model_lro = Sequential()
```

```

model_lro.add(Input(shape=(input_shape,))) # Lớp Input cho hình dạng đầu
vào

model_lro.add(Dense(1, kernel_regularizer=regularizers.l2(0.01))) # L2
Regularization

model_lro.compile(optimizer='adam', loss='mean_squared_error')

return model_lro

Lắp qua từng mã cổ phiếu
for ticker in tickers:

 ticker_data_lro = data[data['Ticker'] == ticker].sort_values('Date')

 X_lro, y_lro, scaler_lro, X_train_lro, X_eval_lro, y_train_lro, y_eval_lro =
prepare_data(ticker_data_lro)

Xây dựng và huấn luyện mô hình Hồi quy tuyến tính với EarlyStopping
model_lro = build_linear_regression_model(X_train_lro.shape[1])

Sử dụng EarlyStopping để dừng huấn luyện khi không có cải thiện
early_stopping_lro = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

Huấn luyện mô hình
history_lro = model_lro.fit(X_train_lro, y_train_lro, epochs=100,
batch_size=32, validation_data=(X_eval_lro, y_eval_lro),
verbose=0, callbacks=[early_stopping_lro])

Dự đoán trên dữ liệu huấn luyện
train_predictions_lro = model_lro.predict(X_train_lro)

```

```

train_predictions_lro = scaler_lro.inverse_transform(train_predictions_lro)
y_train_actual_lro = scaler_lro.inverse_transform(y_train_lro.reshape(-1,
1))

Dự đoán trên dữ liệu kiểm tra
test_predictions_lro = model_lro.predict(X_eval_lro)
test_predictions_lro = scaler_lro.inverse_transform(test_predictions_lro)
y_eval_actual_lro = scaler_lro.inverse_transform(y_eval_lro.reshape(-1, 1))

Tính MSE và R2 cho dữ liệu huấn luyện và kiểm tra
mse_train_lro = mean_squared_error(y_train_actual_lro,
train_predictions_lro)

mse_test_lro = mean_squared_error(y_eval_actual_lro, test_predictions_lro)
r2_train_lro = r2_score(y_train_actual_lro, train_predictions_lro)
r2_test_lro = r2_score(y_eval_actual_lro, test_predictions_lro)

Lưu kết quả
results_lro[ticker] = {
 'history': history_lro,
 'scaler': scaler_lro,
 'model': model_dt,
 'mse_train_lro': mse_train_lro,
 'mse_test_lro': mse_test_lro,
 'r2_train_lro': r2_train_lro,
 'r2_test_lro': r2_test_lro,
 'dates': ticker_data.iloc[:len(y_train_lro) + len(y_eval_lro)][['Date']].values,
 'actual': np.concatenate([y_train_actual_lro, y_eval_actual_lro], axis=0),
}

```

```

'predicted': np.concatenate([train_predictions_lro, test_predictions_lro],
axis=0),
}

- Vẽ biểu đồ huấn luyện mới: giúp theo dõi quá trình huấn luyện và kiểm tra, xác định khi nào dừng huấn luyện.

Code:

“import matplotlib.pyplot as plt
import numpy as np

Giả sử có số lượng ticker (có thể tính n từ results_lro)
n = len(results_lro)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột

Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))

Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()

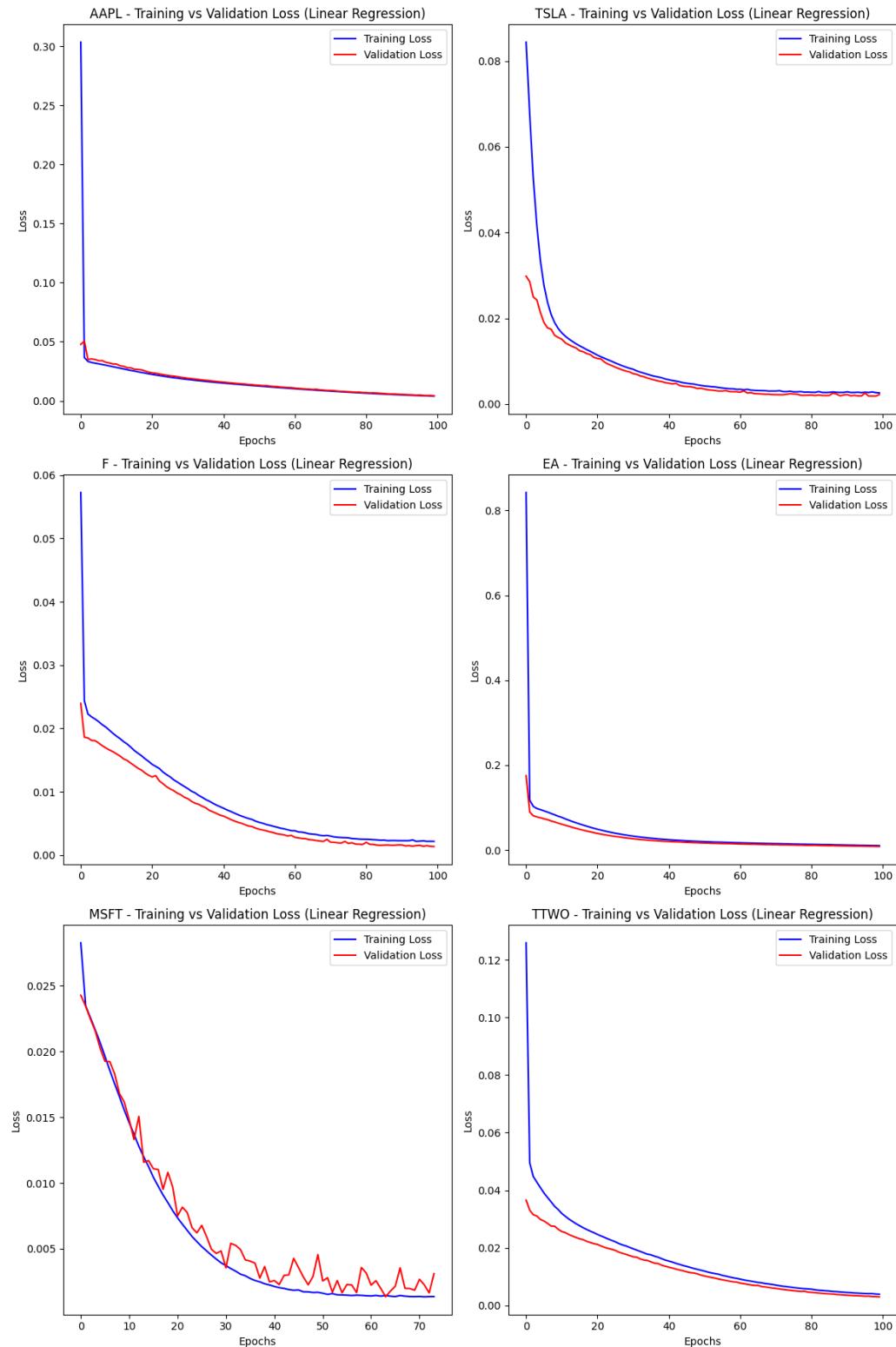
Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_lro.items()):
 history = result['history']

 ax = axes[idx] # Chọn subplot tương ứng
 ax.plot(history.history['loss'], color='blue', label='Training Loss')

```

```
ax.plot(history.history['val_loss'], color='red', label='Validation Loss')
ax.set_title(f'{ticker} - Training vs Validation Loss (Linear Regression)')
ax.set_xlabel('Epochs')
ax.set_ylabel('Loss')
ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show()
```



Hình 57. Biểu đồ huấn luyện bằng mô hình LR sau khi overfitting

- Vẽ biểu đồ kết quả: Biểu đồ thể hiện kết quả giá dự đoán mới so với giá thực tế sau khi thực hiện các biện pháp giảm quá khớp(Overfitting)

Code:

```
"for ticker, result in results_lro.items():
 # Lấy giá trị thực và giá trị dự đoán
 y_train_actual_lro = result['scaler'].inverse_transform(y_train_lro.reshape(-1, 1)).flatten()
 y_train_pred_lro = result['scaler'].inverse_transform(result['model'].predict(X_train_lro).flatten().reshape(-1, 1)).flatten()

 y_eval_actual_lro = result['scaler'].inverse_transform(y_eval_lro.reshape(-1, 1)).flatten()
 y_eval_pred_lro = result['scaler'].inverse_transform(result['model'].predict(X_eval_lro).flatten().reshape(-1, 1)).flatten()

 # Chia dữ liệu ngày tháng cho tập huấn luyện và kiểm tra
 train_dates_lro = ticker_data_lro['Date'].iloc[:len(y_train_actual_lro)].values
 test_dates_lro = ticker_data_lro['Date'].iloc[len(y_train_actual_lro):len(y_train_actual_lro) + len(y_eval_actual_lro)].values

 # Kết hợp dữ liệu thực tế
 actual_dates_lro = np.concatenate([train_dates_lro, test_dates_lro])
 actual_prices_lro = np.concatenate([y_train_actual_lro, y_eval_actual_lro])
```

```

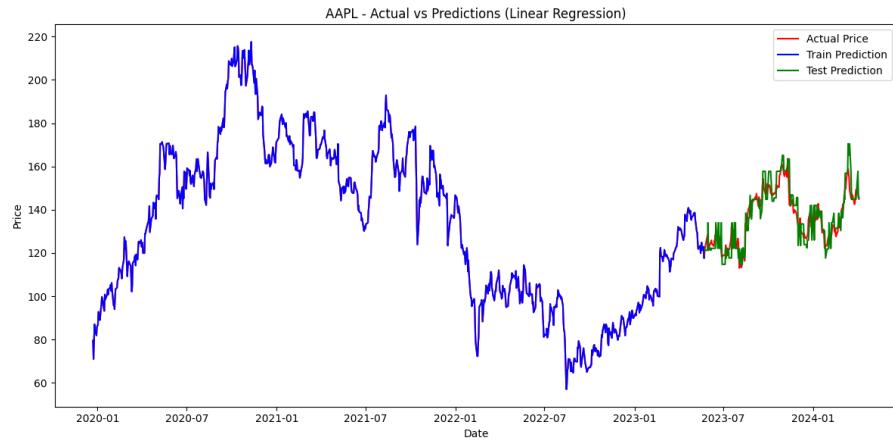
Vẽ đồ thị giá trị thực tế và dự đoán
plt.figure(figsize=(12, 6))

plt.plot(actual_dates_lro, actual_prices_lro, color='red', label='Actual Price')
plt.plot(train_dates_lro, y_train_pred_lro, color='blue', label='Train
Prediction')
plt.plot(test_dates_lro, y_eval_pred_lro, color='green', label='Test
Prediction')

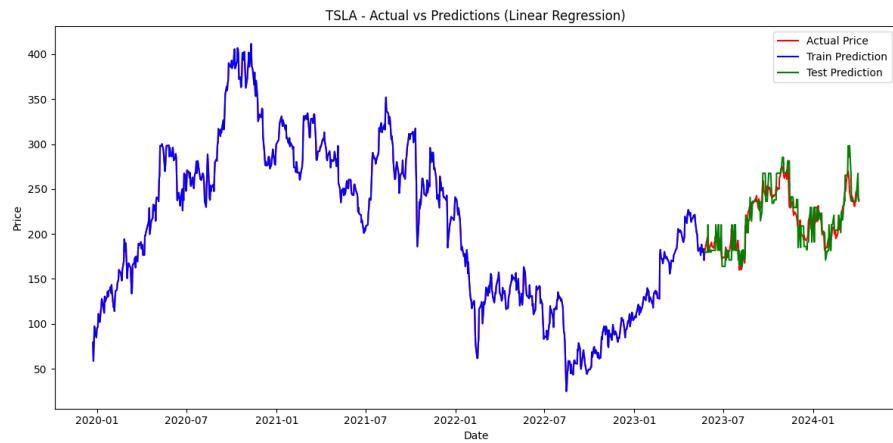
plt.title(f'{ticker} - Actual vs Predictions (Linear Regression)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.tight_layout()
plt.show()

In thông tin MSE và R2
print(f'{ticker} - MSE Train: {result["mse_train_lro"]}, MSE Test:
{result["mse_test_lro"]}')
print(f'{ticker} - R2 Train: {result["r2_train_lro"]}, R2 Test:
{result["r2_test_lro"]}')

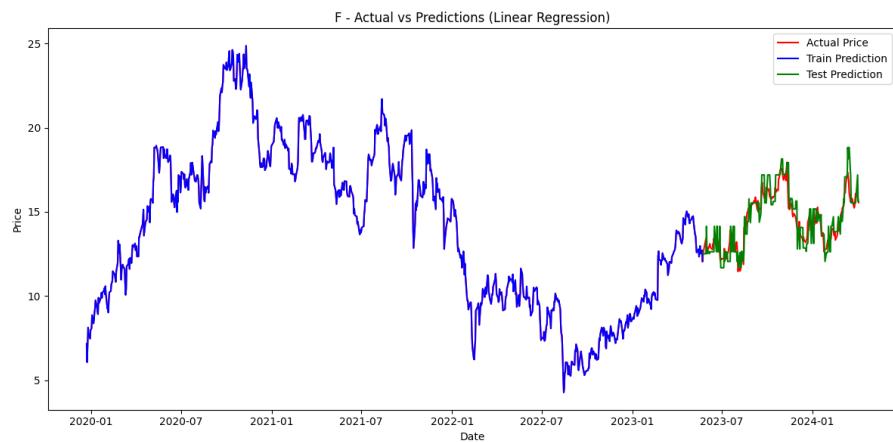
```



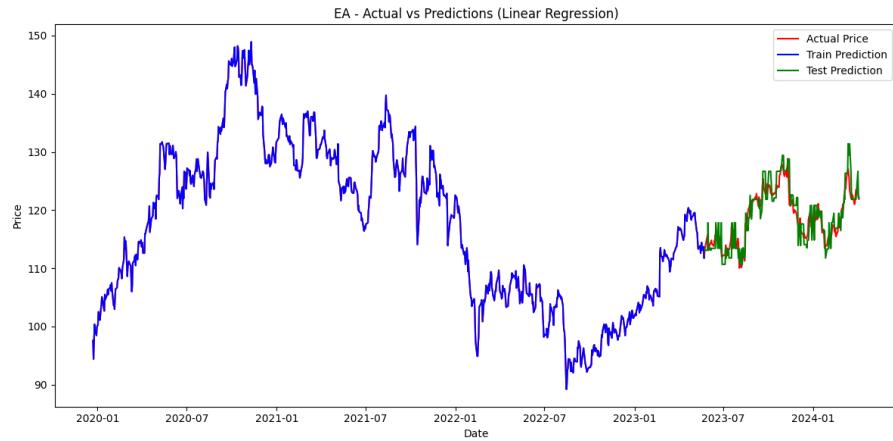
Hình 58. Kết quả dự đoán bằng mô hình LR cho ticker AAPL sau khi overfitting



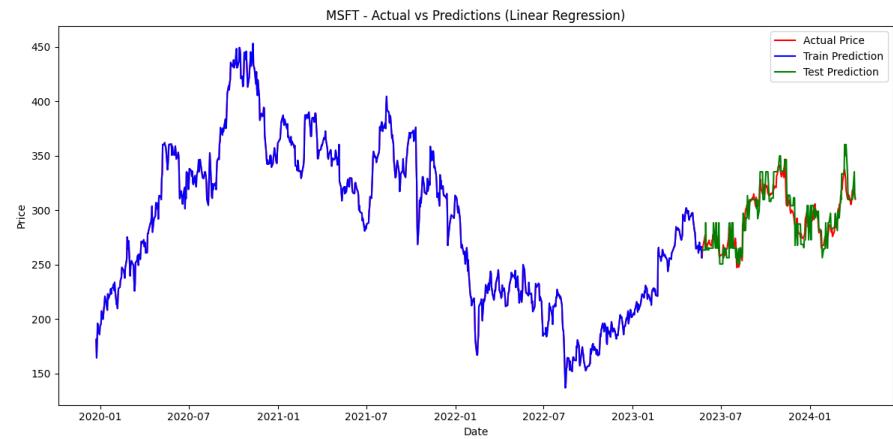
Hình 59. Kết quả dự đoán bằng mô hình LR cho ticker TSLA sau khi overfitting



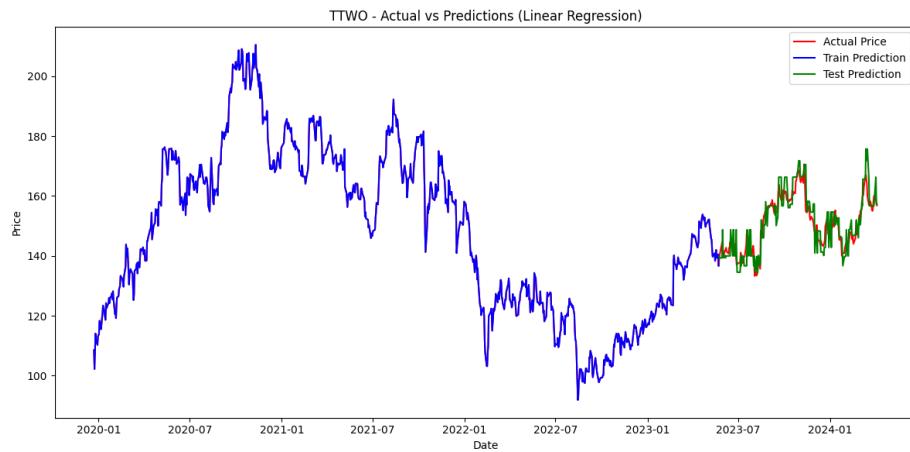
Hình 60. Kết quả dự đoán bằng mô hình LR cho ticker F sau khi overfitting



Hình 61. Kết quả dự đoán bằng mô hình LR cho ticker EA sau khi overfitting



Hình 62. Kết quả dự đoán bằng mô hình LR cho ticker MSFT sau khi overfitting



Hình 63. Kết quả dự đoán bằng mô hình LR cho ticker TTWO sau khi overfitting

#### 2.4.4 Decision Tree:

- Phương pháp ngăn ngừa quá khớp(Overfitting) được áp dụng:
  - + Giới hạn độ sâu của cây (Max Depth): Mô hình DecisionTreeRegressor có một tham số quan trọng là max\_depth, quyết định độ sâu tối đa của cây quyết định. Cây quá sâu có thể học các chi tiết rất nhỏ của dữ liệu huấn luyện, dẫn đến overfitting. Trong code, tham số max\_depth được thử với các giá trị từ 1 đến 20 thông qua vòng lặp và cross-validation, giúp tìm được độ sâu tối ưu mà không quá phức tạp, từ đó giảm overfitting.
  - + Cross-validation: 10-fold cross-validation được sử dụng để đánh giá mô hình. Cross-validation giúp giảm overfitting bằng cách chia tập dữ liệu huấn luyện thành 10 phần, sử dụng mỗi phần làm tập kiểm tra một lần và phần còn lại làm tập huấn luyện. Kỹ thuật này giúp đảm bảo rằng mô hình không chỉ học các đặc trưng của một phần của dữ liệu mà còn có khả năng tổng quát tốt trên dữ liệu chưa thấy.

Code:

```
"from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
Lưu kết quả cho từng mã phiếu
results_dto = {}
```

```
Hàm xây dựng mô hình Decision Tree
def build_decision_tree_model():
 model_dto = DecisionTreeRegressor(random_state=42)
```

```
return model_dto
```

*# Lắp qua từng mã cổ phiếu*

*for ticker in tickers:*

```
ticker_data_dto = data[data['Ticker'] == ticker].sort_values('Date')
```

```
X_dto, y_dto, scaler_dto, X_train_dto, X_eval_dto, y_train_dto, y_eval_dto =
prepare_data(ticker_data_dto)
```

*# Xây dựng mô hình Decision Tree*

```
model_dto = build_decision_tree_model()
```

```
model_dto.fit(X_train_dto, y_train_dto) # Huấn luyện mô hình
```

*# Dự đoán trên dữ liệu huấn luyện và kiểm tra*

```
train_predictions_dto = model_dto.predict(X_train_dto)
```

```
test_predictions_dto = model_dto.predict(X_eval_dto)
```

*# Chuyển đổi dữ liệu dự đoán và thực tế về thang đo ban đầu*

*train\_predictions\_dto*

=

```
scaler_dto.inverse_transform(train_predictions_dto.reshape(-1, 1))
```

```
y_train_actual_dto = scaler_dto.inverse_transform(y_train_dto.reshape(-1,
1))
```

*test\_predictions\_dto*

=

```
scaler_dto.inverse_transform(test_predictions_dto.reshape(-1, 1))
```

```
y_eval_actual_dto = scaler_dto.inverse_transform(y_eval_dto.reshape(-1, 1))
```

*# Tính MSE và R<sup>2</sup>*

```

mse_train_dto = mean_squared_error(y_train_actual_dto,
train_predictions_dto)

mse_test_dto = mean_squared_error(y_eval_actual_dto,
test_predictions_dto)

r2_train_dto = r2_score(y_train_actual_dto, train_predictions_dto)
r2_test_dto = r2_score(y_eval_actual_dto, test_predictions_dto)

Lưu kết quả
results_dto[ticker] = {

 'mse_train_dto': mse_train_dto,
 'mse_test_dto': mse_test_dto,
 'r2_train_dto': r2_train_dto,
 'r2_test_dto': r2_test_dto,
 'train_loss': [],
 'test_loss': [],
 'scaler': scaler_dto,
 'model': model_dto
}

Tính toán độ lỗi train/test cho từng max_depth
n_folds_dto = 10 # Sử dụng 10-fold cross-validation
for max_depth in range(1, 21): # Thử các độ sâu cây từ 1 đến 20
 model_dto.set_params(max_depth=max_depth)
 train_score_dto = cross_val_score(model_dto, X_train_dto, y_train_dto,
cv=n_folds_dto, scoring='neg_mean_squared_error')
 test_score_dto = cross_val_score(model_dto, X_eval_dto, y_eval_dto,
cv=n_folds_dto, scoring='neg_mean_squared_error')

```

```
results_dto[ticker]['train_loss'].append(-train_score_dto.mean())
results_dto[ticker]['test_loss'].append(-test_score_dto.mean())"
```

- Vẽ biểu đồ huấn luyện mới: giúp theo dõi quá trình huấn luyện và kiểm tra, xác định khi nào dừng huấn luyện

Code:

```
"import matplotlib.pyplot as plt
import numpy as np
```

```
Giả sử có số lượng ticker (có thể tính n từ results_dto)
n = len(results_dto)
rows = (n + 1) // 2 # Số hàng (nếu n là số lẻ thì thêm một hàng nữa)
cols = 2 # 2 cột

Tạo subplots với số hàng và số cột
fig, axes = plt.subplots(rows, cols, figsize=(12, rows * 6))

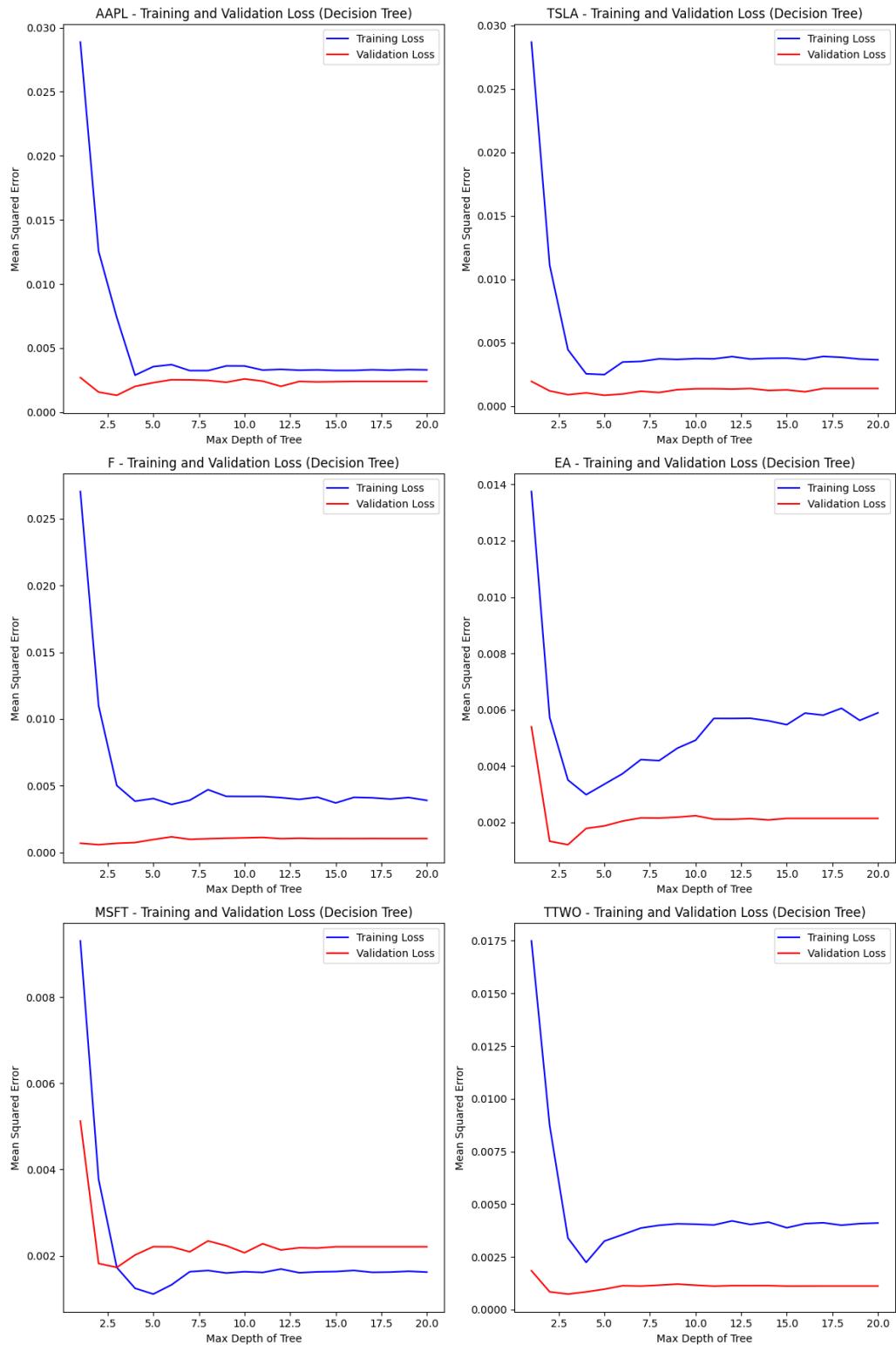
Đảm bảo axes là mảng 1 chiều nếu chỉ có 1 hàng
axes = axes.flatten()

Lặp qua từng ticker để vẽ đồ thị
for idx, (ticker, result) in enumerate(results_dto.items()):
 train_loss = result['train_loss']
 test_loss = result['test_loss']

 ax = axes[idx] # Chọn subplot tương ứng
```

```
ax.plot(range(1, 21), train_loss, label='Training Loss', color='blue')
ax.plot(range(1, 21), test_loss, label='Validation Loss', color='red')
ax.set_xlabel('Max Depth of Tree')
ax.set_ylabel('Mean Squared Error')
ax.set_title(f'{ticker} - Training and Validation Loss (Decision Tree)')
ax.legend()

Thêm khoảng cách giữa các subplots
plt.tight_layout()
plt.show()
```



Hình 64. Biểu đồ huấn luyện bằng mô hình DT sau khi overfitting

- Vẽ biểu đồ kết quả: Biểu đồ thể hiện kết quả giá dự đoán mới so với giá thực tế sau khi thực hiện các biện pháp giảm quá khớp(Overfitting)

Code:

```
"for ticker, result in results(dto.items()):
 # Lấy giá trị thực và giá trị dự đoán
 y_train_actual(dto = result['scaler'].inverse_transform(y_train(dto.reshape(-1, 1)).flatten()
 y_train_pred(dto = result['scaler'].inverse_transform(result['model'].predict(X_train(dto).flatten().reshape(-1, 1)).flatten()

 y_eval_actual(dto = result['scaler'].inverse_transform(y_eval(dto.reshape(-1, 1)).flatten()
 y_eval_pred(dto = result['scaler'].inverse_transform(result['model'].predict(X_eval(dto).flatten().reshape(-1, 1)).flatten()

 # Chia dữ liệu ngày tháng cho tập huấn luyện và kiểm tra
 train_dates(dto = ticker_data(dto['Date'].iloc[:len(y_train_actual(dto)].values
 test_dates(dto = ticker_data(dto['Date'].iloc[len(y_train_actual(dto):len(y_train_actual(dto) +
 len(y_eval_actual(dto)].values

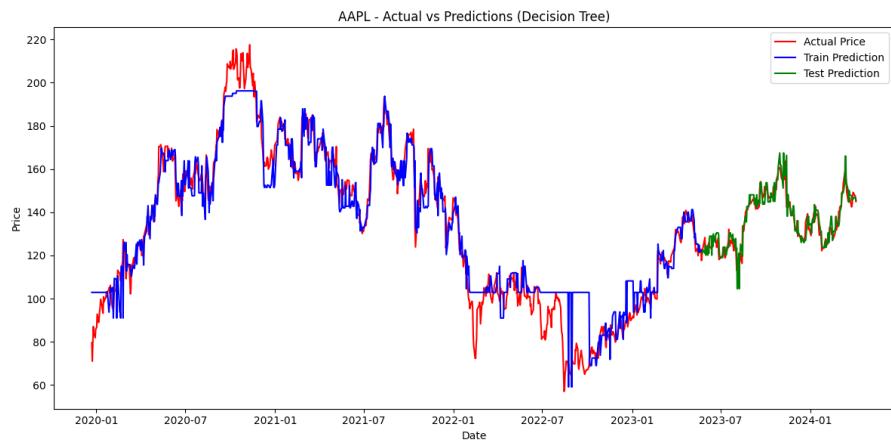
 # Kết hợp dữ liệu thực tế
 actual_dates(dto = np.concatenate([train_dates(dto, test_dates(dto)])
 actual_prices(dto = np.concatenate([y_train_actual(dto, y_eval_actual(dto)])
```

```

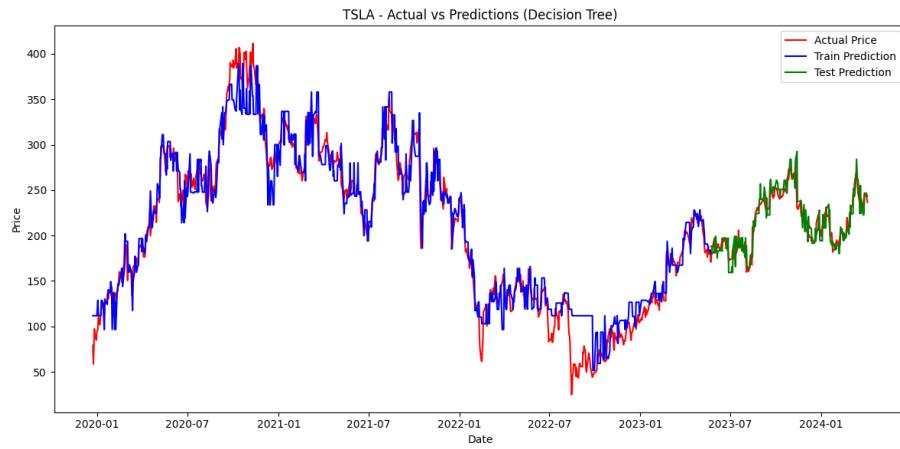
Vẽ đồ thị giá trị thực tế và dự đoán
plt.figure(figsize=(12, 6))
plt.plot(actual_dates_dto, actual_prices_dto, color='red', label='Actual
Price')
plt.plot(train_dates_dto, y_train_pred_dto, color='blue', label='Train
Prediction')
plt.plot(test_dates_dto, y_eval_pred_dto, color='green', label='Test
Prediction')
plt.title(f'{ticker} - Actual vs Predictions (Decision Tree)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.tight_layout()
plt.show()

In thông tin MSE và R2
print(f'{ticker} - MSE Train: {result["mse_train_dto"]}, MSE Test:
{result["mse_test_dto"]}')
print(f'{ticker} - R2 Train: {result["r2_train_dto"]}, R2 Test:
{result["r2_test_dto"]}'")

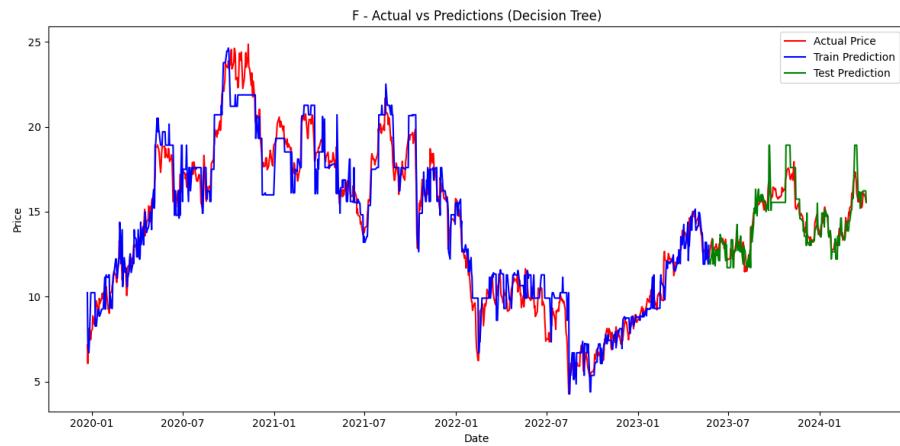
```



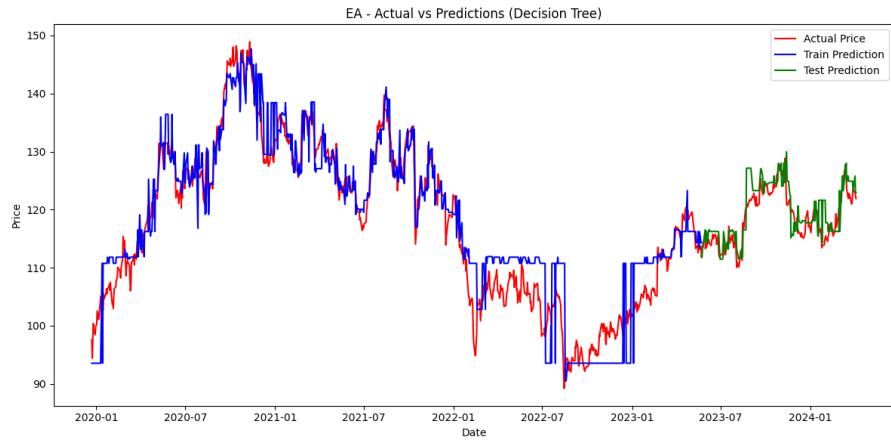
Hình 65. Kết quả dự đoán bằng mô hình DT cho ticker AAPL sau khi overfitting



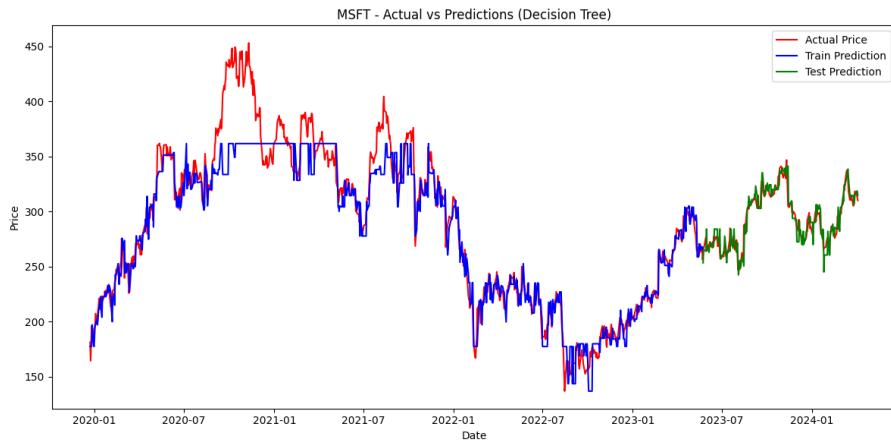
Hình 66. Kết quả dự đoán bằng mô hình DT cho ticker TSLA sau khi overfitting



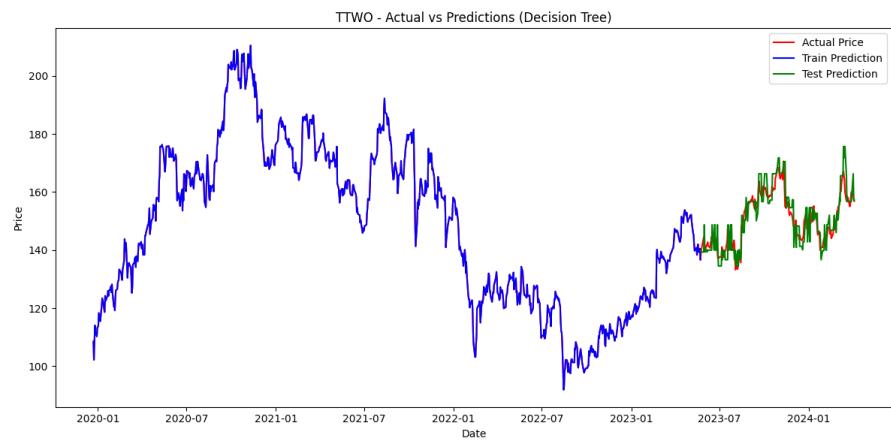
Hình 67. Kết quả dự đoán bằng mô hình DT cho ticker F sau khi overfitting



Hình 68. Kết quả dự đoán bằng mô hình DT cho ticker EA sau khi overfitting



Hình 69. Kết quả dự đoán bằng mô hình DT cho ticker MSFT sau khi overfitting



Hình 70. Kết quả dự đoán bằng mô hình DT cho ticker TTWO sau khi overfitting

## 2.5 Đánh giá và so sánh các mô hình:

### 2.5.1 So sánh mô hình trước khi thực hiện giảm overfitting:

- Code:

```

“import matplotlib.pyplot as plt
import numpy as np

Tạo danh sách các mô hình và các giá trị MSE, R2
models = ['FNN', 'RNN', 'Linear Regression', 'Decision Tree']
train_mse = [
 np.mean([results_fnn[ticker]['train_mse_fnn'] for ticker in tickers]),
 np.mean([results_rnn[ticker]['train_mse_rnn'] for ticker in tickers]),
 np.mean([results_lr[ticker]['train_mse_lr'] for ticker in tickers]),
 np.mean([results_dt[ticker]['train_mse_dt'] for ticker in tickers])
]
test_mse = [
 np.mean([results_fnn[ticker]['test_mse_fnn'] for ticker in tickers]),
 np.mean([results_rnn[ticker]['test_mse_rnn'] for ticker in tickers]),
 np.mean([results_lr[ticker]['test_mse_lr'] for ticker in tickers]),
 np.mean([results_dt[ticker]['test_mse_dt'] for ticker in tickers])
]
train_r2 = [
 np.mean([results_fnn[ticker]['train_r2_fnn'] for ticker in tickers]),
 np.mean([results_rnn[ticker]['train_r2_rnn'] for ticker in tickers]),
 np.mean([results_lr[ticker]['train_r2_lr'] for ticker in tickers]),
 np.mean([results_dt[ticker]['train_r2_dt'] for ticker in tickers])
]
test_r2 = [

```

```

np.mean([results_fnn[ticker]['test_r2_fnn'] for ticker in tickers]),
np.mean([results_rnn[ticker]['test_r2_rnn'] for ticker in tickers]),
np.mean([results_lr[ticker]['test_r2_lr'] for ticker in tickers]),
np.mean([results_dt[ticker]['test_r2_dt'] for ticker in tickers])

]

Vẽ biểu đồ cột MSE
x = np.arange(len(models)) # Vị trí các cột
width = 0.35 # Độ rộng của mỗi cột

plt.figure(figsize=(10, 6))

Train MSE
plt.bar(x - width/2, train_mse, width, label='Train MSE', color='skyblue')
Test MSE
plt.bar(x + width/2, test_mse, width, label='Test MSE', color='salmon')

plt.title('Comparison of MSE Across Models')
plt.xlabel('Models')
plt.ylabel('Mean Squared Error (MSE)')
plt.xticks(x, models)
plt.legend()
plt.tight_layout()
plt.show()

Vẽ biểu đồ cột R^2
plt.figure(figsize=(10, 6))

```

```

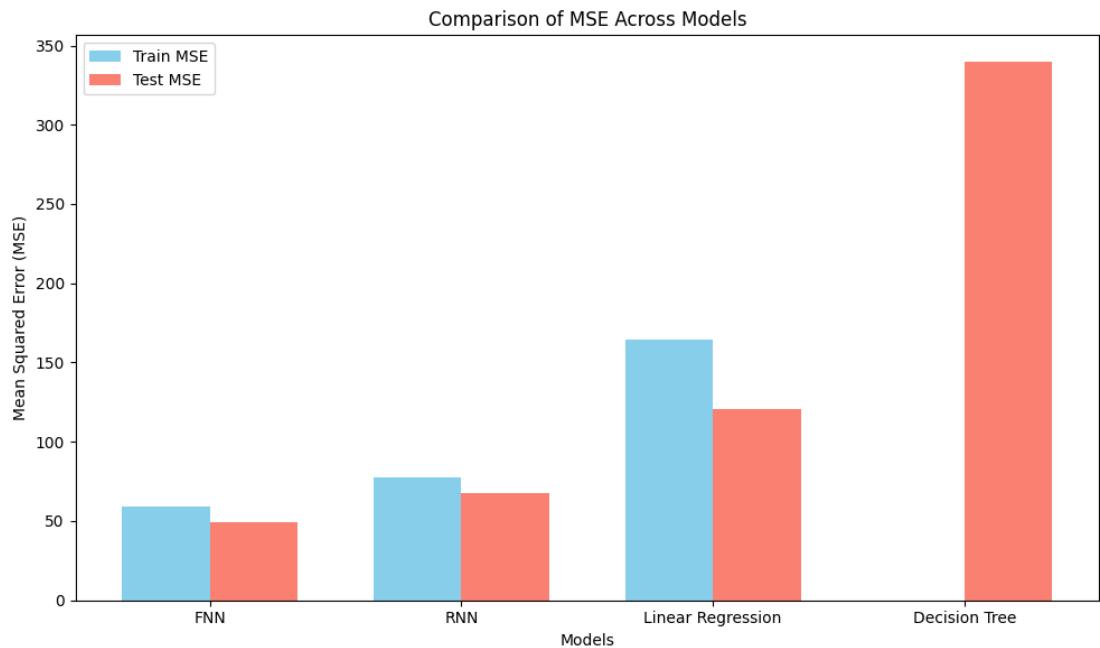
Train R2
plt.bar(x - width/2, train_r2, width, label='Train R2, color='limegreen')

Test R2
plt.bar(x + width/2, test_r2, width, label='Test R2, color='orange')

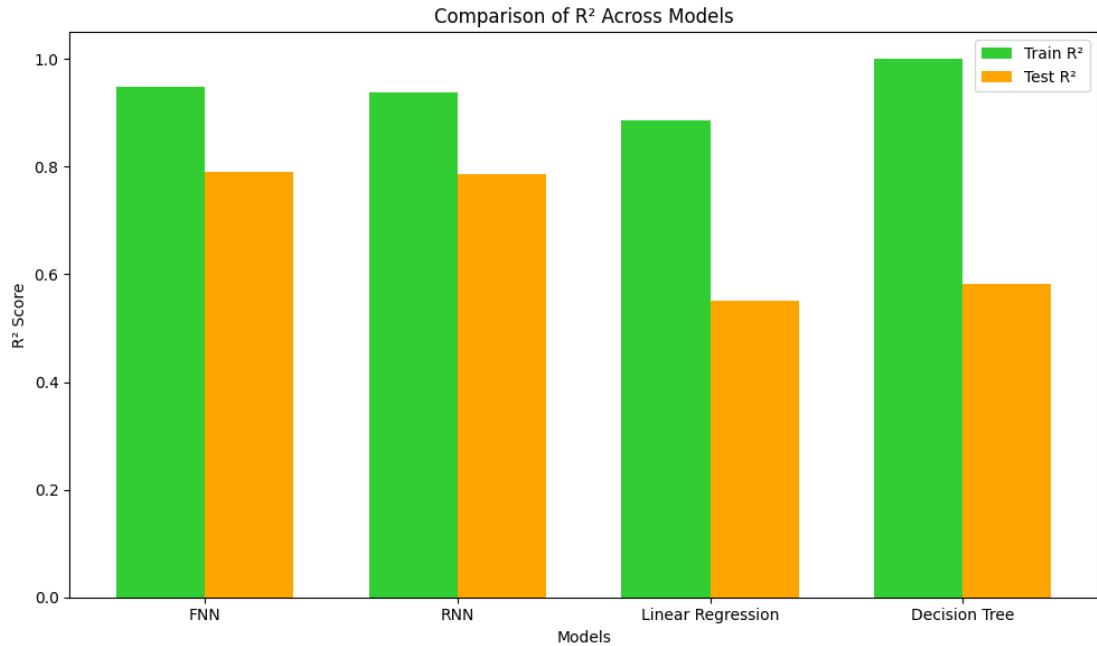
plt.title('Comparison of R2 Across Models')
plt.xlabel('Models')
plt.ylabel('R2 Score')
plt.xticks(x, models)
plt.legend()
plt.tight_layout()
plt.show()

```

- Kết quả nhận được:



Hình 71. So sánh giá trị MSE giữa các mô hình



Hình 72. So sánh giá trị R<sup>2</sup> giữa các mô hình

Nhận xét:

- Mean Squared Error (MSE):
  - + MSE là thước đo sai số bình phương trung bình giữa giá trị dự đoán và giá trị thực. Trong biểu đồ MSE:
    - + FNN (Feedforward Neural Network):
      - Train MSE: Mức thấp nhất trong các mô hình.
      - Test MSE: Tương đối thấp, biểu hiện tính tổng quát tốt.
    - + RNN (Recurrent Neural Network):
      - Train MSE: Lớn hơn FNN một chút.
      - Test MSE: Có xu hướng tương tự train, vẫn thấp hơn nhiều so với các mô hình truyền thống như Linear Regression và Decision Tree.
    - + Linear Regression:
      - Train MSE: Tương đối cao hơn so với FNN và RNN.

- Test MSE: Cao hơn train, cho thấy mô hình này không thể hiện tốt trong dự đoán test set.

+ Decision Tree:

- Train MSE: Thấp nhất trong tất cả các mô hình.
- Test MSE: Rất cao, cho thấy mô hình bị overfitting nghiêm trọng.

-  $R^2$  Score

- +  $R^2$  đánh giá mức độ mô hình giải thích được biến động của dữ liệu. Giá trị  $R^2$  càng gần 1 càng tốt.

+ FNN:

- Train  $R^2$ : Cao (~0.9), thể hiện khả năng học tốt trên tập train.
- Test  $R^2$ : Giảm nhẹ, nhưng vẫn ổn định và hiệu quả.

+ RNN:

- Train  $R^2$ : Cao tương tự FNN.
- Test  $R^2$ : Giảm nhẹ, nhưng không lớn, cho thấy khả năng tổng quát tương đối tốt.

+ Linear Regression:

- Train  $R^2$ : Khá tốt, nhưng thấp hơn so với FNN và RNN.
- Test  $R^2$ : Giảm mạnh, thể hiện sự không ổn định khi dự đoán trên test set.

+ Decision Tree:

- Train  $R^2$ : Đạt giá trị 1, biểu hiện overfitting hoàn toàn.
- Test  $R^2$ : Giảm mạnh, phản ánh hiệu suất kém trên dữ liệu test.

=> Qua kết quả ta thấy được: FNN và RNN là hai mô hình hoạt động tốt nhất, với MSE thấp và  $R^2$  cao, thể hiện khả năng học và tổng quát ổn định. Linear Regression hoạt động kém hơn trong cả train và test set, có thể phù hợp hơn với dữ liệu tuyến tính đơn giản. Decision Tree thể hiện overfitting nghiêm trọng, dẫn đến hiệu quả rất kém trên bộ dữ liệu test.

### 2.5.2 So sánh các mô hình sau khi thực hiện các phương pháp giảm overfitting:

- Code:

```

“import matplotlib.pyplot as plt
import numpy as np

Tạo danh sách các mô hình và các giá trị MSE, R2
models = ['FNN', 'RNN', 'Linear Regression', 'Decision Tree']
train_mse_o = [
 np.mean([results_fnno[ticker]['mse_train_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['mse_train_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['mse_train_lro'] for ticker in tickers]),
 np.mean([results(dto[ticker]['mse_train(dto')] for ticker in tickers)])
]
test_mse_o = [
 np.mean([results_fnno[ticker]['mse_test_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['mse_test_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['mse_test_lro'] for ticker in tickers]),
 np.mean([results(dto[ticker]['mse_test(dto')] for ticker in tickers)])
]
train_r2_o = [
 np.mean([results_fnno[ticker]['r2_train_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['r2_train_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['r2_train_lro'] for ticker in tickers]),
 np.mean([results(dto[ticker]['r2_train(dto')] for ticker in tickers)])
]
test_r2_o = [

```

```

np.mean([results_fnno[ticker]['r2_test_fnno'] for ticker in tickers]),
np.mean([results_rnno[ticker]['r2_test_rnno'] for ticker in tickers]),
np.mean([results_lro[ticker]['r2_test_lro'] for ticker in tickers]),
np.mean([results(dto[ticker]['r2_test(dto')] for ticker in tickers)])

]

Vẽ biểu đồ cột MSE
x = np.arange(len(models)) # Vị trí các cột
width = 0.35 # Độ rộng của mỗi cột

plt.figure(figsize=(10, 6))

Train MSE
plt.bar(x - width/2, train_mse_o, width, label='Train MSE', color='skyblue')
Test MSE
plt.bar(x + width/2, test_mse_o, width, label='Test MSE', color='salmon')

plt.title('Comparison of MSE Across Models')
plt.xlabel('Models')
plt.ylabel('Mean Squared Error (MSE)')
plt.xticks(x, models)
plt.legend()
plt.tight_layout()
plt.show()

Vẽ biểu đồ cột R^2
plt.figure(figsize=(10, 6))

```

```

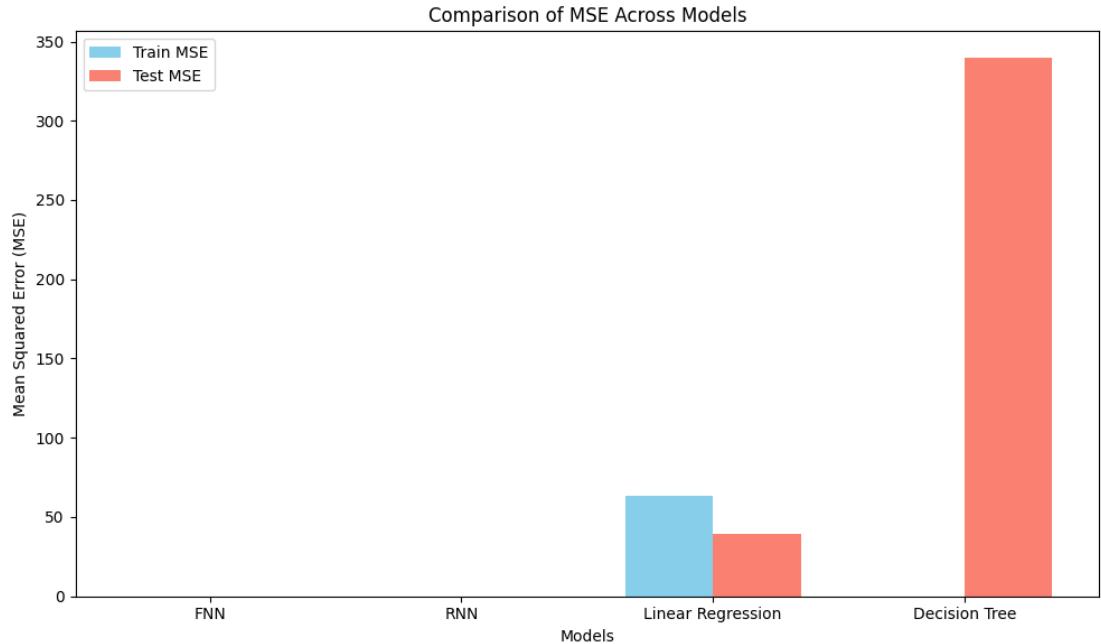
Train R2
plt.bar(x - width/2, train_r2_o, width, label='Train R2', color='limegreen')

Test R2
plt.bar(x + width/2, test_r2_o, width, label='Test R2', color='orange')

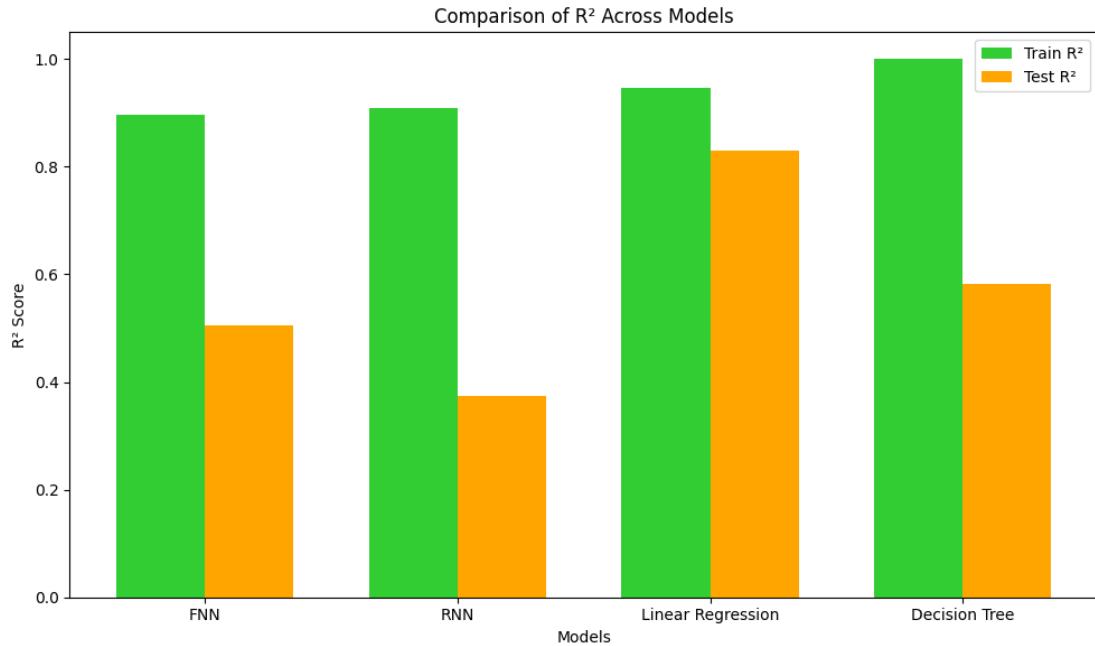
plt.title('Comparison of R2 Across Models')
plt.xlabel('Models')
plt.ylabel('R2 Score')
plt.xticks(x, models)
plt.legend()
plt.tight_layout()
plt.show()

```

- Kết quả nhận được:



Hình 73. So sánh giá trị MSE giữa các mô hình sau khi overfitting



Hình 74. So sánh giá trị R<sup>2</sup> giữa các mô hình sau khi overfitting

Nhận xét:

- Mean Squared Error (MSE):
  - + MSE là thước đo sai số bình phương trung bình giữa giá trị dự đoán và giá trị thực. Trong biểu đồ MSE:
    - + FNN:
      - Train MSE: Mức thấp nhất trong các mô hình.
      - Test MSE: Tương đối thấp, biểu hiện tính tổng quát tốt.
    - + RNN (Recurrent Neural Network):
      - Train MSE: Lớn hơn FNN một chút.
      - Test MSE: Có xu hướng tương tự train, vẫn thấp hơn nhiều so với các mô hình truyền thống như Linear Regression và Decision Tree.
    - + Linear Regression:
      - Train MSE: Tương đối cao hơn so với FNN và RNN.

- Test MSE: Cao hơn train, cho thấy mô hình này không thể hiện tốt trong dự đoán test set.

+ Decision Tree:

- Train MSE: Thấp nhất trong tất cả các mô hình.
- Test MSE: Rất cao, cho thấy mô hình bị overfitting nghiêm trọng.

=> Kết luận về MSE:

+ FNN và RNN có hiệu suất cao với MSE thấp và ổn định.

+ Linear Regression hiệu suất trung bình, phù hợp với dữ liệu tuyến tính đơn giản.

+ Decision Tree overfitting nghiêm trọng, dẫn đến Test MSE rất cao

-  $R^2$  Score

+  $R^2$  đánh giá mức độ mô hình giải thích được biến động của dữ liệu. Giá trị  $R^2$  càng gần 1 càng tốt.

+ FNN:

- Train  $R^2$ : Cao (~0.9), thể hiện khả năng học tốt trên tập train.
- Test  $R^2$ : Giảm nhẹ, nhưng vẫn ổn định và hiệu quả.

+ RNN:

- Train  $R^2$ : Cao tương tự FNN.
- Test  $R^2$ : Giảm nhẹ, nhưng không lớn, cho thấy khả năng tổng quát tương đối tốt.

+ Linear Regression:

- Train  $R^2$ : Khá tốt, nhưng thấp hơn so với FNN và RNN.
- Test  $R^2$ : Giảm mạnh, thể hiện sự không ổn định khi dự đoán trên test set.

+ Decision Tree:

- Train  $R^2$ : Đạt giá trị 1, biểu hiện overfitting hoàn toàn.
- Test  $R^2$ : Giảm mạnh, phản ánh hiệu suất kém trên dữ liệu test.

=> Kết luận về  $R^2$ :

- + FNN và RNN tiếp tục thể hiện hiệu suất cao với  $R^2$  ổn định.
  - + Linear Regression đạt mức chấp nhận được nhưng không bằng FNN và RNN.
  - + Decision Tree không đáng tin cậy do  $R^2$  giảm mạnh trên tập test.
- => Qua kết quả so sánh trên ta thấy được: FNN và RNN hoạt động tốt nhất với MSE thấp và  $R^2$  cao, thể hiện khả năng học và tổng quát ổn định. Linear Regression hoạt động kém hơn FNN và RNN, phù hợp hơn với dữ liệu tuyến tính đơn giản. Decision Tree thể hiện overfitting nghiêm trọng, dẫn đến hiệu quả rất kém trên bộ dữ liệu test.

### 2.5.3 Đánh giá các mô hình:

- Code:

```
"import matplotlib.pyplot as plt
import numpy as np

Tao danh sách các mô hình và các giá trị trước và sau khi overfitting
models = ['FNN', 'RNN', 'Linear Regression', 'Decision Tree']
before_train_mse = [
 np.mean([results_fnn[ticker]['train_mse_fnn'] for ticker in tickers]),
 np.mean([results_rnn[ticker]['train_mse_rnn'] for ticker in tickers]),
 np.mean([results_lr[ticker]['train_mse_lr'] for ticker in tickers]),
 np.mean([results_dt[ticker]['train_mse_dt'] for ticker in tickers])
]
after_train_mse = [
 np.mean([results_fnno[ticker]['mse_train_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['mse_train_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['mse_train_lro'] for ticker in tickers]),
 np.mean([results_dto[ticker]['mse_train(dto] for ticker in tickers)])
```

```

]

before_train_r2 = [
 np.mean([results_fnn[ticker]['train_r2_fnn'] for ticker in tickers]),
 np.mean([results_rnn[ticker]['train_r2_rnn'] for ticker in tickers]),
 np.mean([results_lr[ticker]['train_r2_lr'] for ticker in tickers]),
 np.mean([results_dt[ticker]['train_r2_dt'] for ticker in tickers])
]

after_train_r2 = [
 np.mean([results_fnno[ticker]['r2_train_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['r2_train_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['r2_train_lro'] for ticker in tickers]),
 np.mean([results_dto[ticker]['r2_train(dto)'] for ticker in tickers])
]

before_test_mse = [
 np.mean([results_fnn[ticker]['test_mse_fnn'] for ticker in tickers]),
 np.mean([results_rnn[ticker]['test_mse_rnn'] for ticker in tickers]),
 np.mean([results_lr[ticker]['test_mse_lr'] for ticker in tickers]),
 np.mean([results_dt[ticker]['test_mse_dt'] for ticker in tickers])
]

after_test_mse = [
 np.mean([results_fnno[ticker]['mse_test_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['mse_test_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['mse_test_lro'] for ticker in tickers]),
 np.mean([results_dto[ticker]['mse_test(dto)'] for ticker in tickers])
]

before_test_r2 = [
 np.mean([results_fnn[ticker]['test_r2_fnn'] for ticker in tickers]),

```

```

np.mean([results_rnn[ticker]['test_r2_rnn'] for ticker in tickers]),
np.mean([results_lr[ticker]['test_r2_lr'] for ticker in tickers]),
np.mean([results_dt[ticker]['test_r2_dt'] for ticker in tickers])
]

after_test_r2 = [
 np.mean([results_fnno[ticker]['r2_test_fnno'] for ticker in tickers]),
 np.mean([results_rnno[ticker]['r2_test_rnno'] for ticker in tickers]),
 np.mean([results_lro[ticker]['r2_test_lro'] for ticker in tickers]),
 np.mean([results_dto[ticker]['r2_test(dto)'] for ticker in tickers])
]

Đặt vị trí các cột
x = np.arange(len(models))
width = 0.35
Vẽ các biểu đồ
plt.figure(figsize=(16, 12))

Biểu đồ Train MSE
plt.subplot(2, 2, 1)
plt.bar(x - width/2, before_train_mse, width, label='Before Overfitting',
 color='skyblue')
plt.bar(x + width/2, after_train_mse, width, label='After Overfitting',
 color='salmon')
plt.title('Train MSE Comparison')
plt.xlabel('Models')
plt.ylabel('MSE')
plt.xticks(x, models)
plt.legend()

```

```

Biểu đồ Train R2
plt.subplot(2, 2, 2)
plt.bar(x - width/2, before_train_r2, width, label='Before Overfitting',
color='limegreen')
plt.bar(x + width/2, after_train_r2, width, label='After Overfitting',
color='orange')
plt.title('Train R2 Comparison')
plt.xlabel('Models')
plt.ylabel('R2')
plt.xticks(x, models)
plt.legend()

Biểu đồ Test MSE
plt.subplot(2, 2, 3)
plt.bar(x - width/2, before_test_mse, width, label='Before Overfitting',
color='skyblue')
plt.bar(x + width/2, after_test_mse, width, label='After Overfitting',
color='salmon')
plt.title('Test MSE Comparison')
plt.xlabel('Models')
plt.ylabel('MSE')
plt.xticks(x, models)
plt.legend()

Biểu đồ Test R2
plt.subplot(2, 2, 4)

```

```

plt.bar(x - width/2, before_test_r2, width, label='Before Overfitting',
color='limegreen')

plt.bar(x + width/2, after_test_r2, width, label='After Overfitting',
color='orange')

plt.title('Test R2 Comparison')

plt.xlabel('Models')

plt.ylabel('R2')

plt.xticks(x, models)

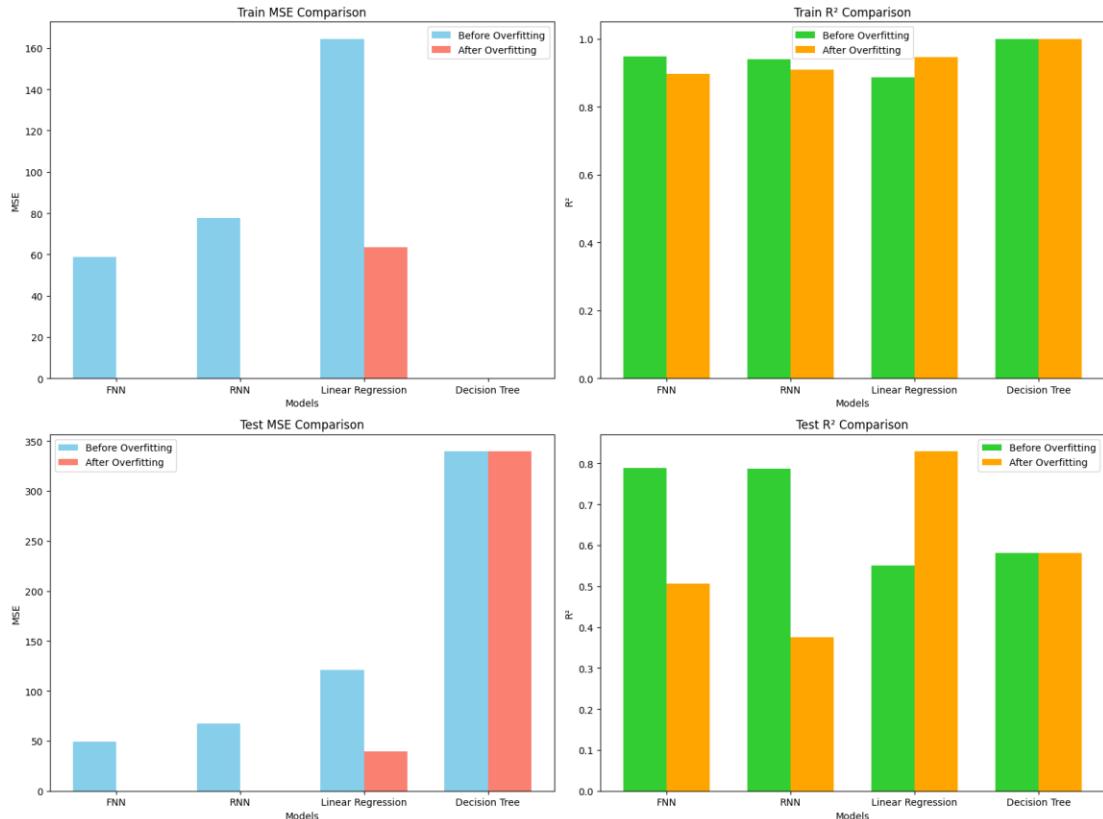
plt.legend()

plt.tight_layout()

plt.show()

```

- Kết quả đạt được:



Hình 75. Đánh giá mô hình trước và sau khi thực hiện overfitting

So sánh hiệu suất:

- Mean Squared Error (MSE)

+ Train MSE:

- Trước khi giảm overfitting:

FNN: Có MSE trung bình, cao hơn Linear Regression nhưng thấp hơn RNN và Decision Tree.

RNN: Lớn hơn FNN, thể hiện khả năng huấn luyện chưa tối ưu.

Linear Regression: Thấp nhất trong các mô hình, hiệu suất rất tốt.

Decision Tree: Gần như bằng 0, chứng tỏ overfitting nghiêm trọng.

- Sau khi giảm overfitting:

FNN: Giảm đáng kể, thể hiện cải thiện tốt.

RNN: Giảm nhẹ, nhưng vẫn lớn hơn FNN.

Linear Regression: Có sự tăng nhẹ do mất mát độ chính xác trên tập train.

Decision Tree: Tăng cao hơn đáng kể so với trước, giảm bớt overfitting.

+ Test MSE:

- Trước khi giảm overfitting:

FNN: Trung bình, tốt hơn RNN nhưng kém hơn Linear Regression.

RNN: Cao hơn FNN, khả năng tổng quát hóa thấp.

Linear Regression: Tốt nhất với MSE thấp nhất.

Decision Tree: Rất cao, overfitting nghiêm trọng.

- Sau khi giảm overfitting:

FNN: Cải thiện tốt, MSE giảm đáng kể.

RNN: Giảm nhẹ, nhưng vẫn kém hơn FNN.

Linear Regression: Gần như không đổi, vẫn tốt nhất.

Decision Tree: Giảm nhưng vẫn cao hơn rất nhiều so với các mô hình khác.

- $R^2$  Score

+ Train R<sup>2</sup>:

- Trước khi giảm overfitting:

FNN: R<sup>2</sup> gần 1, cho thấy mô hình học rất tốt trên tập train.

RNN: Cao tương đương FNN.

Linear Regression: Cao, nhưng thấp hơn một chút so với FNN và RNN.

Decision Tree: Bằng 1, overfitting hoàn toàn.

- Sau khi giảm overfitting:

FNN và RNN: Giảm nhẹ, nhưng vẫn duy trì ở mức cao, thể hiện hiệu suất tốt.

Linear Regression: Không đổi, ổn định.

Decision Tree: Giảm mạnh, phản ánh mô hình bớt overfitting.

+ Test R<sup>2</sup>:

- Trước khi giảm overfitting:

FNN: Khá tốt, nhưng thấp hơn Linear Regression.

RNN: Thấp hơn FNN, khả năng tổng quát hóa yếu.

Linear Regression: Cao nhất, thể hiện tính ổn định và hiệu quả.

Decision Tree: Thấp nhất, hiệu suất kém trên tập kiểm tra.

- Sau khi giảm overfitting:

FNN: Cải thiện nhẹ, R<sup>2</sup> tăng.

RNN: Cải thiện không đáng kể, vẫn thấp hơn FNN.

Linear Regression: Không đổi, duy trì hiệu suất tốt nhất.

Decision Tree: Cải thiện nhưng vẫn thấp hơn đáng kể so với các mô hình khác.

- Kết luận:

+ FNN: Sau khi giảm overfitting, mô hình cải thiện rõ rệt, thể hiện sự cân bằng giữa tập train và test.

+ RNN: Hiệu suất có cải thiện, nhưng vẫn kém hơn FNN, cần thêm tối ưu hóa.

- + Linear Regression: Ôn định nhất và hiệu suất tốt trên cả tập train và test. Đây là lựa chọn đáng tin cậy cho bài toán này.
- + Decision Tree: Dù đã giảm overfitting, hiệu suất vẫn kém, cần áp dụng thêm các kỹ thuật nâng cao như Random Forest hoặc Gradient Boosting.

## PHẦN 3 – CÂU 3

### 3.1 CNN là gì?

CNN (Convolutional Neural Network) hay mạng nơ-ron tích chập là một mô hình trong Deep Learning được thiết kế đặc biệt để xử lý dữ liệu có dạng lưới. CNN đã trở thành công cụ quan trọng trong các bài toán như nhận dạng đối tượng, phân loại hình ảnh, phát hiện khuôn mặt, và xử lý ngôn ngữ tự nhiên.

Mạng CNN hoạt động dựa trên nguyên lý của các lớp tích chập (Convolutional Layer), sử dụng các **filter (kernel)** để trích xuất thông tin từ dữ liệu. Điểm nổi bật của CNN là khả năng học các đặc trưng từ dữ liệu một cách tự động và giảm số lượng tham số so với mạng truyền thống.

### 3.2 Cấu trúc cơ bản của mạng CNN:

Mạng CNN gồm các lớp, thành phần chính:

#### 3.2.1 Convolutional Layer (Lớp tích chập):

Sử dụng các bộ lọc (filter/kernel) thực hiện phép **tích chập** trên ảnh đầu vào để tạo ra các **feature map** (bản đồ đặc trưng).

Mục tiêu chính là trích xuất các đặc trưng cục bộ như cạnh, góc hoặc kết cấu (texture).

Tham số cần quan tâm:

- Kích thước kernel: (thường là  $3 \times 3$  hoặc  $5 \times 5$ ).
- Stride (bước nhảy): Xác định mức độ dịch chuyển của kernel trên ảnh đầu vào.
- Padding: Quyết định việc thêm pixel xung quanh ảnh để duy trì kích thước đầu ra.

Công thức:

$$Y[i, j] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X[i + m, j + n] \cdot K[m, n] + b$$

- $X[i, j]$ : Dữ liệu đầu vào (Input image).
- $K[m, n]$ : Bộ lọc (kernel/filter) kích thước  $M \times N$ .
- $b$ : bias (độ lệch)
- $Y[i, j]$ : Kết quả tích chập (feature map) tại vị trí  $(i, j)$ .

### 3.2.2 Activation Function (Hàm kích hoạt):

Sau khi thực hiện phép tích chập, hàm kích hoạt (thường là **ReLU**) được áp dụng để tạo tính phi tuyến cho mô hình. Điều này giúp mô hình học được các đặc trưng phức tạp hơn.

Hàm phô biến:

- ReLU (Rectified Linear Unit):

$$f(x) = \max(0, x)$$

Giúp mô hình phi tuyến tính và loại bỏ giá trị âm.

- Leaky ReLU:

$$f(x) = \begin{cases} x, & \text{nếu } x > 0 \\ \alpha x, & \text{nếu } x \leq 0 \end{cases}$$

với  $\alpha$  là một giá trị nhỏ, thường là 0.01

- Ngoài ra, có thể dùng Sigmoid, Tanh trong các trường hợp khác.

### 3.2.3 Pooling Layer (Lớp pooling):

Có nhiệm vụ giảm kích thước của feature map, giảm độ phức tạp tính toán và ngăn ngừa overfitting.

Hai loại pooling phô biến:

- Max Pooling: Chọn giá trị lớn nhất trong một vùng  $k \times k$  cụ thể.

$$Y[i, j] = \max(X[i:i+k, j:j+k])$$

- Average Pooling: Lấy giá trị trung bình trong vùng  $k \times k$  cụ thể.

$$Y[i, j] = \frac{1}{k^2} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X[i+m, j+n]$$

### 3.2.4 Fully Connected Layer (Lớp kết nối đầy đủ):

Các đầu ra từ các lớp trước được kết nối thành một vector phẳng, sau đó được đưa vào lớp fully connected để thực hiện phân loại.

Công thức:

$$z = W \cdot a + b$$

- $W$ : Ma trận trọng số.
- $a$ : Đầu vào từ tầng trước.
- $b$ : bias
- $z$ : Đầu ra, có thể được đưa qua hàm softmax để tính xác suất phân loại.
- Softmax:

$$\hat{y}_j = P(y = j|x) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

Với  $K$  là số lớp phân loại,  $z_j$  là giá trị đầu ra của lớp thứ  $j$ ,  $\hat{y}_j = P(y = j|x)$  là xác suất đầu vào  $x$  thuộc lớp  $j$ .

### 3.3 Nguyên lý hoạt động của CNN:

CNN hoạt động thông qua các giai đoạn chính như sau:

#### 3.3.1 Giai đoạn trích xuất đặc trưng (Feature Extraction):

- Dữ liệu đầu vào (thường là ảnh) được xử lý qua các lớp tích chập (Convolutional Layer) để trích xuất đặc trưng cục bộ.
- Mỗi bộ lọc (kernel) tập trung vào một loại đặc trưng nhất định, như cạnh, góc, hay kết cấu, từ đó tạo ra các bản đồ đặc trưng (feature maps).
- Sau mỗi lớp tích chập, hàm kích hoạt (Activation Function) như ReLU sẽ được áp dụng để tạo tính phi tuyến.

#### 3.3.2 Giai đoạn giảm kích thước (Dimensionality Reduction):

- Các lớp pooling (Pooling Layer) sẽ giảm kích thước không gian của feature maps mà vẫn giữ được thông tin quan trọng.

- Quá trình này giúp giảm số lượng tham số, tăng tốc độ tính toán, và giảm nguy cơ overfitting.
- Kết quả của quá trình này là một tập hợp nhỏ gọn hơn của các đặc trưng đã được trích xuất.

### **3.3.3 Giai đoạn phân loại (Classification):**

- Sau khi qua các lớp tích chập và pooling, các đặc trưng được chuyển qua lớp Fully Connected (FC) để học cách phân loại dữ liệu.
- Lớp Fully Connected liên kết tất cả các nơ-ron từ tầng trước vào các nơ-ron đầu ra.
- Đầu ra thường được đưa qua hàm softmax để tính xác suất của từng lớp trong bài toán phân loại.

## **3.4 Quá trình huấn luyện của CNN:**

### **3.4.1 Truyền tiến (Forward Propagation):**

- Dữ liệu đầu vào được truyền qua lần lượt từng lớp của mạng CNN:
  - + Lớp tích chập (Convolutional Layer): Thực hiện phép tích chập giữa dữ liệu đầu vào và bộ lọc (filter/kernel) để tạo ra đặc trưng (feature map).
  - + Lớp kích hoạt (Activation Layer): Sử dụng hàm kích hoạt (ví dụ: ReLU, Sigmoid) để tăng tính phi tuyến.
  - + Lớp pooling (Pooling Layer): Giảm kích thước của feature map nhằm giảm số lượng tham số và tăng hiệu quả tính toán.
  - + Lớp kết nối đầy đủ (Fully Connected Layer): Tổng hợp thông tin từ các đặc trưng đã trích xuất và dự đoán kết quả.
- Giá trị đầu ra (output) của lớp cuối được tính toán và đưa vào hàm mất mát (Loss Function) để so sánh với nhãn thực tế.

### **3.4.2 Tính toán lỗi (Loss Calculation):**

**Tính toán lỗi** là bước quan trọng trong huấn luyện mạng nơ-ron, giúp đánh giá mức độ chênh lệch giữa **kết quả dự đoán** và **kết quả thực tế**. Dựa vào lỗi này, thuật toán lan truyền ngược sẽ cập nhật trọng số của mạng để cải thiện độ chính xác.

Hàm mất mát đo lường sự khác biệt giữa giá trị dự đoán  $\hat{y}$  và giá trị thực tế  $y$ . Tùy vào loại bài toán, ta chọn hàm mất mát phù hợp.

- Hàm Cross-Entropy Loss:

Sử dụng trong các bài toán **phân loại** (Classification), đặc biệt với nhiều lớp (multi-class).

Công thức:

$$L = - \sum_{i=1}^K y_i \log(\hat{y}_i)$$

Với:  $y_i$  là Nhãn thực tế (one-hot encoded) của lớp  $i$  (0 hoặc 1).

$\hat{y}_i$  là Xác suất dự đoán của lớp  $i$ .

$K$  là Số lớp phân loại.

Ví dụ: Với bài toán có 3 lớp, nếu  $y = [1,0,1]$  và  $\hat{y} = [0.7,0.2,0.1]$ :

$$\begin{aligned} L &= -[1 \cdot \log(0.7) + 0 \cdot \log(0.2) + 1 \cdot \log(0.1)] \\ &= -\log(0.7) - \log(0.1) \end{aligned}$$

- Hàm Mean Squared Error (MSE):

Sử dụng trong các bài toán **hồi quy** (Regression).

Công thức:

$$L = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Với:  $y_i$  là Giá trị thực tế.,

$\hat{y}_i$  là Giá trị dự đoán.

$N$  là Số lượng mẫu dữ liệu.

Hàm MSE tính trung bình bình phương sai số giữa giá trị dự đoán và thực tế.

- Hàm Mean Absolute Error (MAE):

Công thức:

$$L = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

Với:  $y_i$  là Giá trị thực tế.,

$\hat{y}_i$  là Giá trị dự đoán.

$N$  là Số lượng mẫu dữ liệu.

Hàm MAE tính độ chênh lệch trung bình tuyệt đối giữa giá trị thực tế và dự đoán.

Tóm lại:

- Cross-Entropy Loss:

- + Phù hợp cho các bài toán phân loại.
- + Nhạy cảm với xác suất dự đoán sai lệch nhỏ.

- MSE/MAE:

- + Phù hợp cho các bài toán hồi quy.
- + MSE nhẫn mạn sai số lớn (do bình phương), còn MAE xem trọng sai số nhỏ.

### 3.4.3 Lan truyền ngược (Backward Propagation):

**Lan truyền ngược** là quá trình tính toán và truyền ngược gradient của hàm mất mát từ **đầu ra (output layer)** về **các lớp trước đó** trong mạng để điều chỉnh các tham số (trọng số và bias). Quá trình này giúp giảm sai số giữa giá trị dự đoán và giá trị thực tế bằng cách tối ưu hóa các tham số của mạng nơ-ron.

Sau khi tính toán lỗi, mạng sử dụng thuật toán **lan truyền ngược** để cập nhật trọng số và bias thông qua các bước:

- a. Tính toán gradient của hàm mất mát:

Gradient của hàm mất mát  $L$  đối với các tham số như trọng số  $W$  và bias  $b$  sẽ được tính toán bằng **đạo hàm**.

- Gradient đối với trọng số  $W$ :

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial W}$$

Với:  $z$  là Đầu ra của một nơ-ron (trước khi qua hàm kích hoạt).

$\frac{\partial L}{\partial z}$  là Gradient của hàm mất mát với đầu ra  $z$ .

$\frac{\partial z}{\partial W} = a$  là Đầu vào từ lớp trước.

- Gradient đối với bias  $b$ :

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b}$$

Với:  $\frac{\partial z}{\partial b} = 1$ .

Gradient này được lan truyền ngược qua từng lớp, từ lớp đầu ra đến các lớp trước thông qua **đạo hàm chuỗi**.

- b. Lan truyền ngược qua các lớp trong CNN:

- Lan truyền qua Lớp kết nối đầy đủ (Fully Connected Layer):

Gradient của hàm mất mát đối với trọng số và bias trong lớp fully connected được tính như sau:

$$\frac{\partial L}{\partial W} = \delta \cdot a^T, \quad \frac{\partial L}{\partial b} = \delta$$

Với:  $a$  là Đầu vào từ lớp trước đó.

$\delta = \frac{\partial L}{\partial z}$  là Gradient của hàm mất mát với đầu ra trước kích hoạt.

- Lan truyền qua Lớp Kích Hoạt (Activation Layer):

Nếu sử dụng **ReLU**, đạo hàm của hàm kích hoạt được tính như:

$$f'(z) = \begin{cases} 1, & \text{nếu } z > 0 \\ 0, & \text{nếu } z \leq 0 \end{cases}$$

- Lan truyền qua Lớp Tích Chập (Convolutional Layer):

Gradient của kernel (filter) trong lớp tích chập được tính bằng tích chập giữa gradient của đầu ra và đầu vào:

$$\frac{\partial L}{\partial K} = X * \delta$$

Với:  $X$  là Dữ liệu đầu vào của lớp tích chập.

$\delta$  là Gradient của hàm mất mát đối với đầu ra của lớp tích chập.

\* là Phép tích chập.

Gradient của đầu vào từ lớp trước được tính như:

$$\frac{\partial L}{\partial X} = K * \delta$$

- Lan truyền qua Lớp Pooling (Pooling Layer):
  - + Đối với Max Pooling, gradient chỉ được truyền đến vị trí có giá trị lớn nhất.
  - + Đối với Average Pooling, gradient được chia đều cho tất cả các vị trí trong vùng pooling.
- c. Cập nhật tham số:

Sau khi tính được gradient, các tham số của mạng được cập nhật bằng thuật toán tối ưu như **Gradient Descent** hoặc biến thể như **Adam Optimizer**.

- Cập nhật trọng số:

$$W_{new} = W_{old} - \alpha \frac{\partial L}{\partial W}$$

Với:  $W$  là Trọng số.

$\alpha$  là Tốc độ học (Learning Rate)

$\frac{\partial L}{\partial W}$  là Gradient của hàm mất mát đối với trọng số.

- Cập nhật bias:

$$b_{new} = b_{old} - \alpha \frac{\partial L}{\partial b}$$

Với:  $b$  là Bias.

$\alpha$  là Tốc độ học (Learning Rate)

$\frac{\partial L}{\partial b}$  là Gradient của hàm mất mát đối với Bias.

Lan truyền ngược là quy trình cốt lõi trong quá trình huấn luyện CNN, giúp tối ưu hóa các tham số bằng cách điều chỉnh trọng số và bias để giảm sai số giữa dự đoán và thực tế. Việc áp dụng chính xác gradient và các thuật toán tối ưu sẽ giúp mạng học hiệu quả và nhanh chóng hội tụ.

### **3.5 Ưu điểm và nhược điểm của CNN:**

#### **3.5.1 Ưu điểm:**

- Tự động học đặc trưng:

Không cần thiết kế thủ công các đặc trưng như trong các phương pháp truyền thống.

- Giảm số lượng tham số:

Nhờ chia sẻ trọng số qua các bộ lọc, CNN giảm thiểu đáng kể số lượng tham số so với mạng fully connected.

- Khả năng học đặc trưng cục bộ:

Các bộ lọc trong CNN chỉ tập trung vào một vùng nhỏ của ảnh, giúp nắm bắt các đặc trưng quan trọng.

#### **3.5.2 Nhược điểm:**

- Cần nhiều dữ liệu và tài nguyên:

CNN yêu cầu một lượng lớn dữ liệu để đạt hiệu quả tốt.

- Khó hiểu và điều chỉnh:

Cần tinh chỉnh nhiều siêu tham số như kích thước kernel, stride, learning rate.

### **3.6 Các kỹ thuật cải tiến CNN:**

Một số kỹ thuật cải tiến CNN để tăng hiệu suất:

- Batch Normalization: Ôn định đầu ra của mỗi lớp, giúp mô hình học nhanh hơn.
- Dropout: Giảm overfitting bằng cách ngẫu nhiên tắt một số nơ-ron trong quá trình huấn luyện.

- Data Augmentation: Tăng cường dữ liệu bằng cách xoay, lật, hoặc thay đổi độ sáng của ảnh.

### **3.7 Các ứng dụng phổ biến của CNN:**

#### **3.7.1 Xử lý ảnh:**

- Phân loại hình ảnh (Image Classification).
- Phát hiện đối tượng (Object Detection).
- Nhận diện khuôn mặt (Face Recognition).
- Phân đoạn ảnh (Image Segmentation).

#### **3.7.2 Xử lý video:**

- Theo dõi đối tượng trong video (Object Tracking).
- Phân tích hành vi (Action Recognition).

#### **3.7.3 Xử lý ngôn ngữ tự nhiên (khi kết hợp với mạng RNN):**

- Trích xuất thông tin từ văn bản.
- Dịch máy hình ảnh (Image Captioning).

#### **3.7.4 Các lĩnh vực khác:**

- Phân tích y tế (Xử lý ảnh MRI, CT).
- Tự động lái xe (Car Detection trong xe tự hành).

### **3.8 Các kiến trúc CNN nổi bật:**

Để minh họa rõ hơn cho các ứng dụng của mạng CNN, dưới đây là một số mô hình, kiến trúc CNN nổi bật:

- LeNet-5: Mô hình đầu tiên ứng dụng CNN trong nhận dạng chữ viết tay.
- AlexNet: Đánh dấu bước ngoặt trong Deep Learning, chiến thắng ImageNet 2012.
- VGGNet: Kiến trúc sâu hơn (16 hoặc 19 tầng), sử dụng các kernel nhỏ ( $3 \times 3$ ).
- ResNet: Sử dụng các kết nối tắt (skip connections) để tránh vấn đề mất mát gradient trong mạng sâu.

- Inception: Mô hình hiệu quả với cơ chế "network within a network".
- YOLO (You Only Look Once): Một trong những kiến trúc CNN phổ biến nhất cho bài toán phát hiện đối tượng thời gian thực.

### **3.9 Chương trình áp dụng mô hình CNN vào bài toán phân loại:**

#### **3.9.1 Tổng quan về tập dữ liệu MNIST:**

Convolutional Neural Network (CNN) là một trong những mô hình mạng nơ-ron nhân tạo nổi bật, đặc biệt trong xử lý ảnh. Trong bài báo cáo này, chúng tôi áp dụng mô hình CNN để phân loại tập dữ liệu hình ảnh **MNIST** – một bộ dữ liệu gồm các chữ số viết tay. Mục tiêu là xây dựng một mô hình CNN hiệu quả, đạt độ chính xác cao để phân loại chính xác các chữ số từ 0 đến 9 dựa trên hình ảnh kích thước  $28 \times 28$  pixel.

Tập dữ liệu MNIST là một trong những tập dữ liệu phổ biến nhất được sử dụng trong các bài toán về xử lý ảnh.

Đặc điểm:

- Số lượng mẫu huấn luyện: 60,000 ảnh.
- Số lượng mẫu kiểm tra: 10,000 ảnh.
- Định dạng: Ảnh grayscale (đen trắng), kích thước  $28 \times 28$ .
- Số lớp phân loại: 10 lớp tương ứng các chữ số từ 0 đến 9.

Thách thức:

- Mỗi chữ số có nhiều hình dạng khác nhau do chữ viết tay thay đổi theo người.
- Mô hình cần học các đặc trưng như cạnh, góc và các kết cấu (textures) để nhận dạng chính xác.

#### **3.9.2 Quy trình thực hiện:**

- a. Bước 1: Import thư viện

```

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

```

Giải thích:

- NumPy: Hỗ trợ thao tác với dữ liệu mảng, xử lý dữ liệu đầu vào trước khi đưa vào mô hình.
- Matplotlib: Dùng để vẽ biểu đồ và hiển thị hình ảnh từ dữ liệu, thuận tiện trong việc trực quan hóa dữ liệu và kết quả mô hình.
- TensorFlow: Thư viện chính để xây dựng và huấn luyện mạng CNN.
- Keras: API cấp cao của TensorFlow, cung cấp công cụ dễ dàng để tạo và huấn luyện mô hình.
- Sequential: Là một kiểu xây dựng mô hình tuyến tính, giúp định nghĩa các lớp của mạng theo thứ tự từ đầu vào đến đầu ra.
- Conv2D: Lớp tích chập, sử dụng bộ lọc để trích xuất đặc trưng từ hình ảnh.
- MaxPooling2D: Lớp pooling, giảm kích thước của đầu ra từ lớp tích chập, giúp giảm số lượng tham số và tránh overfitting.
- Flatten: Chuyển đầu ra của các lớp trước thành một vector phẳng, chuẩn bị cho đầu vào của các lớp fully connected.
- Dense: Lớp fully connected, kết nối toàn bộ các đầu vào đến các đầu ra của lớp.
- Dropout: Kỹ thuật regularization, tắt ngẫu nhiên một số node trong lớp fully connected để tránh overfitting.
- Input: Khai báo kích thước đầu vào của mô hình, đảm bảo cấu trúc mô hình được định nghĩa rõ ràng.

- tensorflow.keras.datasets.mnist: Bộ dữ liệu chữ số viết tay, thường dùng để huấn luyện và kiểm tra mô hình học máy.
- to\_categorical: Chuyển đổi nhãn đầu ra thành dạng one-hot encoding, giúp phù hợp với lớp đầu ra softmax trong mô hình.

b. Bước 2: Tải và tiền xử lý dữ liệu

```
1. Load dữ liệu MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

2. Tiền xử lý dữ liệu
Chuẩn hóa dữ liệu: đưa pixel về khoảng [0, 1]
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

Chuyển nhãn thành dạng one-hot vector
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

Kiểm tra kích thước của dữ liệu
print(f"Train data shape: {x_train.shape}, Train labels shape: {y_train.shape}")
print(f"Test data shape: {x_test.shape}, Test labels shape: {y_test.shape}")
```

Giải thích:

- Tải dữ liệu MNIST:
  - + Sử dụng `mnist.load_data()` để tải dữ liệu huấn luyện và kiểm tra.
  - + Kết quả: `x_train` và `x_test` chứa các ảnh; `y_train` và `y_test` chứa các nhãn tương ứng.
- Chuẩn hóa dữ liệu:
  - + Chia giá trị pixel từ 0-255 cho 255 để đưa về khoảng [0, 1], giúp mô hình hội tụ nhanh hơn.
  - + Dữ liệu được reshape thành kích thước (-1,28,28,1)(-1, 28, 28, 1)(-1,28,28,1), thêm chiều thứ 4 để phù hợp với kiến trúc CNN (kênh màu).
- One-hot encoding nhãn:
  - + Chuyển nhãn số (0-9) thành dạng vector one-hot.

+ Ví dụ:  $3 \rightarrow [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$ .

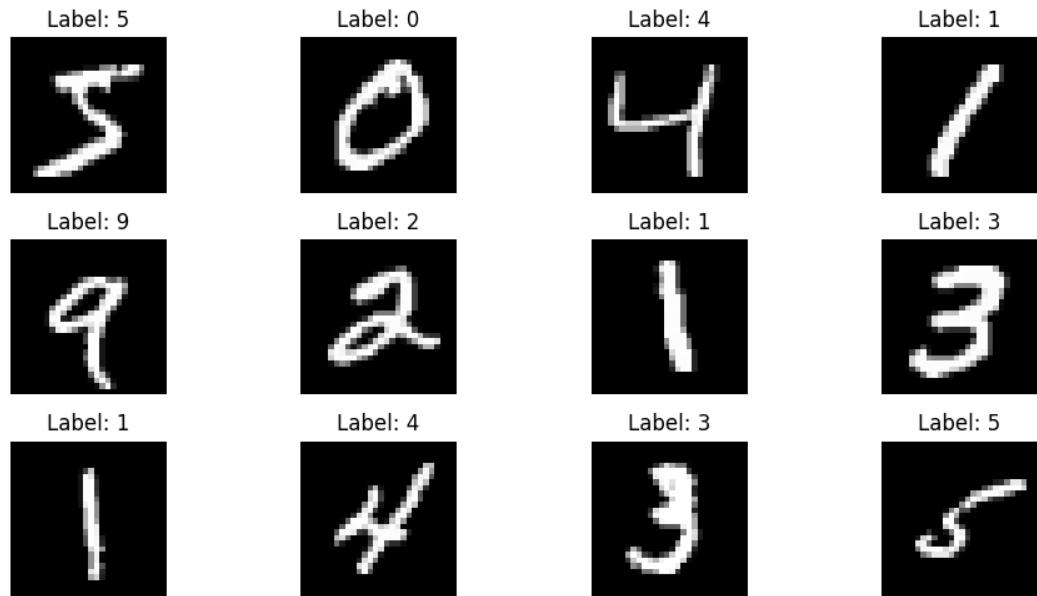
```
Train data shape: (60000, 28, 28, 1), Train labels shape: (60000, 10)
Test data shape: (10000, 28, 28, 1), Test labels shape: (10000, 10)
```

c. Bước 3: Hiển thị dữ liệu

```
Hiển thị một số hình ảnh từ tập dữ liệu
plt.figure(figsize=(10, 5))
for i in range(12):
 plt.subplot(3, 4, i+1)
 plt.imshow(x_train[i].reshape(28, 28), cmap='gray')
 plt.title(f"Label: {np.argmax(y_train[i])}")
 plt.axis('off')
plt.tight_layout()
plt.show()
```

Giải thích:

- Sử dụng thư viện matplotlib để hiển thị 12 ảnh đầu tiên từ tập huấn luyện cùng với nhãn tương ứng.



d. Bước 4: Xây dựng mô hình CNN

```

Xây dựng mô hình CNN
model = Sequential([
 # Lớp tích chập 1
 Input(shape=(28, 28, 1)),
 Conv2D(32, kernel_size=(3, 3), activation='relu'),
 MaxPooling2D(pool_size=(2, 2)),

 # Lớp tích chập 2
 Conv2D(64, kernel_size=(3, 3), activation='relu'),
 MaxPooling2D(pool_size=(2, 2)),

 # Flatten: chuyển dữ liệu thành vector phẳng
 Flatten(),

 # Fully Connected Layer
 Dense(128, activation='relu'),
 Dropout(0.5), # Dropout để tránh overfitting

 # Lớp đầu ra với hàm softmax
 Dense(10, activation='softmax')
])

Tóm tắt mô hình
model.summary()

```

Giải thích:

- Convolutional Layer:
  - + Lớp 1: Sử dụng 32 kernel kích thước  $3 \times 3$ .
  - + Lớp 2: Sử dụng 64 kernel kích thước  $3 \times 3$ .
  - + Hàm kích hoạt ReLU: Loại bỏ giá trị âm, chỉ giữ lại giá trị dương.
- Pooling Layer:
  - + MaxPooling  $2 \times 2$ : Giảm kích thước feature map, chọn giá trị lớn nhất trong mỗi vùng.

- Flatten:
  - + Chuyển feature map thành vector 1D để đưa vào Fully Connected Layer.
- Fully Connected Layer:
  - + Lớp ẩn: 128 nơ-ron, activation ReLU.
  - + Dropout: Ngẫu nhiên bỏ 50% nơ-ron để giảm overfitting.
  - + Lớp đầu ra: 10 nơ-ron, activation Softmax để tính xác suất cho mỗi lớp.

```
Model: "sequential_1"
```

| Layer (type)                   | Output Shape       | Param # |
|--------------------------------|--------------------|---------|
| conv2d_2 (Conv2D)              | (None, 26, 26, 32) | 320     |
| max_pooling2d_2 (MaxPooling2D) | (None, 13, 13, 32) | 0       |
| conv2d_3 (Conv2D)              | (None, 11, 11, 64) | 18,496  |
| max_pooling2d_3 (MaxPooling2D) | (None, 5, 5, 64)   | 0       |
| flatten_1 (Flatten)            | (None, 1600)       | 0       |
| dense_2 (Dense)                | (None, 128)        | 204,928 |
| dropout_1 (Dropout)            | (None, 128)        | 0       |
| dense_3 (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)
```

- e. Bước 5: Compile mô hình:

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

print("Model compiled successfully!")
```

Giải thích:

- Optimizer (Adam): Thuật toán tối ưu hóa điều chỉnh trọng số và bias một cách hiệu quả.
- Loss Function (Categorical Cross-Entropy): Đánh giá sự khác biệt giữa giá trị dự đoán và giá trị thực tế.
- Metrics (Accuracy): Độ chính xác trên dữ liệu huấn luyện và kiểm tra.

Model compiled successfully!

#### f. Bước 6: Huấn luyện mô hình

```
Huấn luyện mô hình
history = model.fit(x_train, y_train,
 validation_data=(x_test, y_test),
 epochs=50, batch_size=128)

print("Training complete!")
```

Giải thích:

- Epochs (10): Lặp lại quá trình huấn luyện trên toàn bộ tập dữ liệu 10 lần.
- Batch size (128): Chia dữ liệu thành từng nhóm nhỏ để xử lý.
- Validation data: Dùng tập kiểm tra để đánh giá hiệu suất mô hình trong quá trình huấn luyện.

```

Epoch 1/50
469/469 19s 38ms/step - accuracy: 0.8046 - loss: 0.6321 - val_accuracy: 0.9812 - val_loss: 0.0588
Epoch 2/50
469/469 16s 34ms/step - accuracy: 0.9702 - loss: 0.1008 - val_accuracy: 0.9870 - val_loss: 0.0385
Epoch 3/50
469/469 15s 31ms/step - accuracy: 0.9782 - loss: 0.0742 - val_accuracy: 0.9903 - val_loss: 0.0311
Epoch 4/50
469/469 14s 31ms/step - accuracy: 0.9817 - loss: 0.0607 - val_accuracy: 0.9894 - val_loss: 0.0313
Epoch 5/50
469/469 14s 31ms/step - accuracy: 0.9852 - loss: 0.0502 - val_accuracy: 0.9902 - val_loss: 0.0270
Epoch 6/50
469/469 17s 35ms/step - accuracy: 0.9869 - loss: 0.0423 - val_accuracy: 0.9918 - val_loss: 0.0261
Epoch 7/50
469/469 14s 30ms/step - accuracy: 0.9884 - loss: 0.0366 - val_accuracy: 0.9912 - val_loss: 0.0249
Epoch 8/50
469/469 14s 30ms/step - accuracy: 0.9905 - loss: 0.0317 - val_accuracy: 0.9919 - val_loss: 0.0257
Epoch 9/50
469/469 14s 30ms/step - accuracy: 0.9901 - loss: 0.0305 - val_accuracy: 0.9926 - val_loss: 0.0221
Epoch 10/50
469/469 14s 30ms/step - accuracy: 0.9920 - loss: 0.0249 - val_accuracy: 0.9932 - val_loss: 0.0215
Epoch 11/50
469/469 14s 29ms/step - accuracy: 0.9933 - loss: 0.0223 - val_accuracy: 0.9922 - val_loss: 0.0261
Epoch 12/50
469/469 14s 31ms/step - accuracy: 0.9929 - loss: 0.0220 - val_accuracy: 0.9931 - val_loss: 0.0220
Epoch 13/50
...
469/469 16s 34ms/step - accuracy: 0.9980 - loss: 0.0055 - val_accuracy: 0.9933 - val_loss: 0.0358
Epoch 50/50
469/469 13s 28ms/step - accuracy: 0.9983 - loss: 0.0055 - val_accuracy: 0.9938 - val_loss: 0.0326
Training complete!

```

g. Bước 7: Đánh giá mô hình

```

Đánh giá mô hình trên tập kiểm tra
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")

```

Giải thích:

- Đo lường **hàm mất mát (loss)** và **độ chính xác (accuracy)** trên tập kiểm tra bằng phương thức *model.evaluate*.
- Input:
  - + *x\_test*: Dữ liệu hình ảnh kiểm tra (ảnh chữ số viết tay).
  - + *y\_test*: Nhãn đúng tương ứng với các ảnh kiểm tra (dạng one-hot).
  - + *verbose*: Quy định mức độ hiển thị thông tin. Với *verbose*=2, chương trình chỉ in ra kết quả tổng quát của việc đánh giá (loss và accuracy).
- Output:
  - + *test\_loss*: Hàm mất mát (loss) được tính trên tập kiểm tra.
  - + *test\_acc*: Độ chính xác (accuracy) của mô hình trên tập kiểm tra.

```
313/313 - 1s - 3ms/step - accuracy: 0.9938 - loss: 0.0326

Test accuracy: 0.9938
Test loss: 0.0326
```

h. Bước 8: Vẽ biểu đồ Accuracy và Loss

```
Vẽ biểu đồ accuracy và loss
plt.figure(figsize=(12, 4))

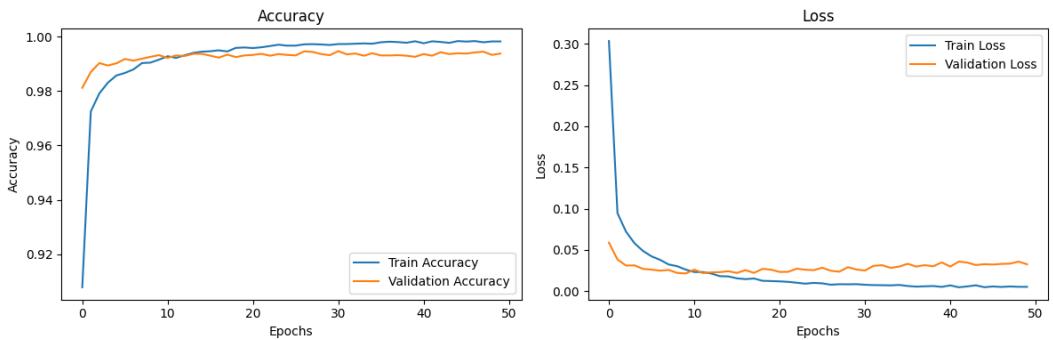
Biểu đồ độ chính xác
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

Biểu đồ hàm mất mát
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Giải thích:

- Accuracy: Quan sát độ chính xác của mô hình trên tập huấn luyện và kiểm tra qua các epoch.
- Loss: Quan sát sự thay đổi của hàm mất mát qua các epoch.



### i. Bước 9: Dự đoán trên hình ảnh mới

```
Lấy ngẫu nhiên 5 hình ảnh từ tập kiểm tra
num_images = 5
indices = np.random.choice(len(x_test), num_images, replace=False)

Dự đoán và hiển thị kết quả
plt.figure(figsize=(10, 5))
for i, idx in enumerate(indices):
 img = x_test[idx]
 prediction = np.argmax(model.predict(img.reshape(1, 28, 28, 1)))
 label = np.argmax(y_test[idx])

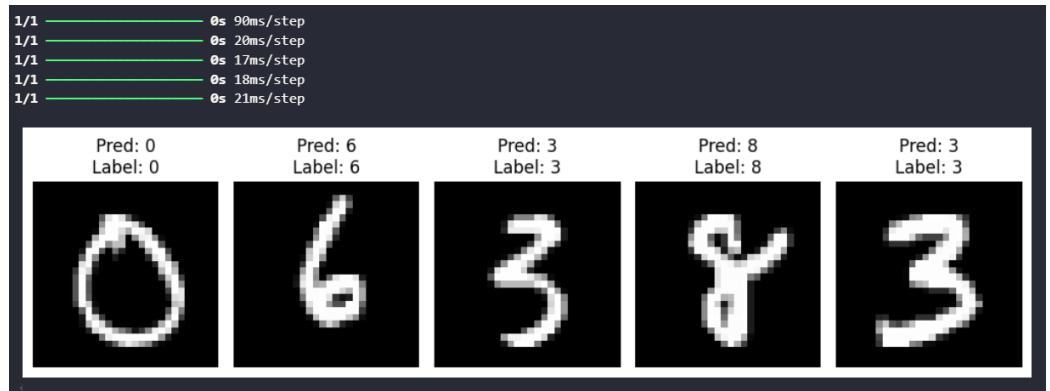
 plt.subplot(1, num_images, i+1)
 plt.imshow(img.reshape(28, 28), cmap='gray')
 plt.title(f"Pred: {prediction}\nLabel: {label}")
 plt.axis('off')

plt.tight_layout()
plt.show()
```

Giải thích:

- Lấy ngẫu nhiên 5 hình ảnh từ tập kiểm tra:
  - + num\_images: Số lượng hình ảnh cần dự đoán (ở đây là 5).
  - + np.random.choice: Chọn ngẫu nhiên num\_images chỉ số từ tập dữ liệu kiểm tra mà không lặp lại (replace=False).
- Dự đoán kết quả cho từng hình ảnh:
  - + img = x\_test[idx]: Lấy một hình ảnh từ tập kiểm tra.
  - + model.predict(img.reshape(1, 28, 28, 1)):

- Dự đoán nhãn cho hình ảnh.
- `img.reshape(1, 28, 28, 1)`: Chuyển đổi kích thước ảnh từ  $28 \times 28$  thành  $1 \times 28 \times 28 \times 1$  để phù hợp với đầu vào của mô hình.
  - + `np.argmax`: Lấy chỉ số của lớp có xác suất cao nhất.
- Hiển thị hình ảnh và kết quả dự đoán:
  - + `plt.subplot(1, num_images, i+1)`: Tạo `num_images` subplot (ở đây là 5) để hiển thị từng ảnh.
  - + `plt.imshow(img.reshape(28, 28), cmap='gray')`: Hiển thị ảnh dưới dạng grayscale.
  - + `plt.title(f"Pred: {prediction}\nLabel: {label}")`: Thêm tiêu đề chứa nhãn dự đoán (Pred) và nhãn thực tế (Label).
  - + `plt.axis('off')`: Ẩn trục tọa độ.
- Hiển thị toàn bộ hình ảnh:
  - + `plt.tight_layout()`: Tự động điều chỉnh bố cục để không bị chồng chéo giữa các subplot.
  - + `plt.show()`: Hiển thị toàn bộ hình ảnh.
- Kết quả mong đợi:
  - + Dữ liệu hiển thị: 5 hình ảnh được chọn ngẫu nhiên từ tập kiểm tra.
  - + Thông tin trên mỗi hình ảnh:
    - **Pred:** Nhãn mà mô hình dự đoán.
    - **Label:** Nhãn thực tế từ tập kiểm tra.
  - + Nếu dự đoán trùng khớp, mô hình đã hoạt động chính xác trên ảnh đó.



Kết luận:

- Nếu  $\text{Pred} = \text{Label}$ : Mô hình dự đoán đúng, thể hiện hiệu quả của mô hình.
- Nếu  $\text{Pred} \neq \text{Label}$ : Mô hình dự đoán sai, cần phân tích thêm:
  - + Xem xét dữ liệu đầu vào có khó nhận diện không (như chữ số bị nhòe).
  - + Kiểm tra lại kiến trúc hoặc tham số của mô hình để cải thiện hiệu suất.

## **TÀI LIỆU THAM KHẢO**

### **Tiếng Anh**

1. Matthew D. Zeiler (2012), Adadelta: An Adaptive Learning Rate Method.
2. Kevin P. Murphy (2012), Machine Learning: A Probabilistic Perspective