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

**FINAL REPORT**

# INTRODUCTION TO MACHINE LEARNING

*Người hướng dẫn*: **TS LƯƠNG QUỐC ĐẠI**

*Người thực hiện*: **NGUYỄN HOÀNG ÂN - 52200183**

**NGUYỄN NHẬT TRƯỜNG - 52200192**

**NGUYỄN DUY HÀ VỸ - 52200180**

Lớp **: 22050301**

Khoá **: 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**

**FINAL REPORT**

# INTRODUCTION TO MACHINE LEARNING

Người hướng dẫn: **TS TRẦN LƯƠNG QUỐC ĐẠI**
Người thực hiện: **NGUYỄN HOÀNG ÂN - 52200183**
**NGUYỄN NHẬT TRƯỜNG - 52200192**
**NGUYỄN DUY HÀ VỸ - 52200180**
Lớp **: 22050301**
Khoá **: 26**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

# LỜI CẢM ƠN

Chúng tôi xin gửi lời cảm ơn chân thành đến **TS. Trần Lương Quốc Đại** vì đã giúp đỡ chúng tôi trong quá trình làm bài báo cáo vừa qua. Nhờ sự hướng dẫn và giải đáp của thầy, chúng tôi đã có thể hoàn thành bài báo cáo của mình một cách tốt nhất có thể.

Thầy đã dành thời gian và tâm huyết để hướng dẫn chúng tôi từng bước, giải thích những khái niệm khó hiểu, và chỉ ra những lỗi sai mà tôi mắc phải trong quá trình viết bài. Những lời khuyên và sự hỗ trợ của thầy đã giúp chúng tôi tự tin hơn trong việc hoàn thiện bài báo cáo của mình.

Chúng tôi rất biết ơn sự giúp đỡ của thầy và mong rằng sẽ có nhiều học trò khác được thầy hướng dẫn và giúp đỡ như chúng tôi. Chúng tôi hy vọng sẽ có cơ hội được học hỏi thêm từ thầy trong tương lai.

# CÔNG TRÌNH ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Chúng tôi xin cam đoan đây là công trình nghiên cứu của riêng chúng tôi và được sự hướng dẫn khoa học của TS Trần Lương Quốc Đại. Các nội dung nghiên cứu, kết quả trong đề tài 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 luận văn 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 tôi xin hoàn toàn chịu trách nhiệm về nội dung luận văn của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày  tháng  năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Hoàng Ân*

*Nguyễn Nhật Trường*

*Nguyễn Duy Hà Vỹ*

# TÓM TẮT

Bài báo cáo gồm 3 chương:

Chương 1 : THEORY OF OPTIMIZATION METHODS

Chương 2: STOCK PRICE PREDICTION PROBLEM

Chương 3: STUDY AND PRESENT A DEEP MACHINE LEARNING

# MỤC LỤC

# DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

## DANH MỤC HÌNH

# CHƯƠNG 1 – THEORY OF OPTIMIZATION METHODS

## 1. OPTIMIZATION METHODS

### *1.1 Gradient Descent (GD)*

Gradient Descent là một thuật toán tối ưu hóa nhằm tìm giá trị cực tiểu của hàm mất mát bằng cách cập nhật các tham số dựa trên gradient của hàm mất mát. Công thức cập nhật:

$$\theta = \theta - \eta \nabla \theta J(\theta)$$

Trong đó:

θ: Tham số mô hình (weights và bias).

η: Learning rate (tốc độ học).

∇J(θ): Gradient của hàm mất mát J(θ).

**Các biến thể của Gradient Descent:**

- **Batch Gradient Descent:** Sử dụng toàn bộ dữ liệu để tính gradient. Cập nhật tham số dựa trên toàn bộ tập dữ liệu:

$$\theta = \theta - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla J(\theta, xi, yi)$$

  - Ưu điểm: Tính gradient chính xác.
  - Nhược điểm: Tốn nhiều tài nguyên khi dữ liệu lớn.
- **Stochastic Gradient Descent (SGD):** Sử dụng một mẫu ngẫu nhiên (sample) để tính gradient:

$$\theta = \theta - \eta \nabla J(\theta, xi, yi)$$

  - Ưu điểm: Cập nhật nhanh, phù hợp với dữ liệu lớn.
  - Nhược điểm: Gradient dao động mạnh, có thể dẫn đến hội tụ chậm.
- **Mini-Batch Gradient Descent:** Kết hợp giữa Batch và SGD, sử dụng một lô dữ liệu nhỏ (mini-batch) để tính gradient:

$$\theta = \theta - \eta \frac{1}{b} \sum_{i=1}^{b} \nabla J(\theta, xi, yi)$$

- Ưu điểm: Cân bằng giữa tốc độ và độ chính xác.

### *1.2 Momentum*

Momentum là một cải tiến của Gradient Descent nhằm tăng tốc hội tụ bằng cách sử dụng động lượng (momentum). Công thức:

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla J(\theta)$$

$$\theta = \theta - \eta v_t$$

**Trong đó:**

$v_t$: Động lượng (momentum) tại bước t.

β: Hệ số momentum (thường chọn β=0.9).

**Ưu điểm:**

- Giảm dao động gradient.
- Tăng tốc hội tụ trong các vùng phẳng của hàm mất mát.

### *1.3 Adagrad (Adaptive Gradient Algorithm)*

Adagrad tự động điều chỉnh learning rate cho từng tham số dựa trên gradient trước đó. Công thức:

$$s_t = s_{t-1} + \nabla J(\theta)^2$$

$$\theta = \theta - \frac{n}{\sqrt{s_t} + \epsilon} \nabla J(\theta)$$

Trong đó:

- $s_t$: Tổng bình phương gradient của tham số.
- ϵ: Số nhỏ để tránh chia cho 0 (thường là 10-8)

**Ưu điểm:**

- Hiệu quả với các tham số hiếm gặp (sparse features).
- Giảm tốc độ học với các gradient lớn.

**Nhược điểm:**

- Learning rate giảm quá nhanh, dẫn đến hội tụ chậm.

### *1.4 RMSProp (Root Mean Square Propagation)*

RMSProp cải thiện Adagrad bằng cách sử dụng giá trị trung bình động của bình phương gradient. Công thức:

$$s_t = \beta s_{t-1} + (1-\beta)\nabla J(\theta)^2$$

$$\theta = \theta - \frac{n}{\sqrt{s_t} + \epsilon}\nabla J(\theta)$$

**Trong đó:**

- $s_t$: Trung bình động của bình phương gradient.
- $\beta$: Hệ số điều chỉnh (thường chọn $\beta$=0.9).

**Ưu điểm:**

- Phù hợp với các bài toán có gradient dao động mạnh.
- Giữ learning rate ổn định hơn Adagrad.

### *1.5 Adam (Adaptive Moment Estimation)*

Adam kết hợp cả Momentum và RMSProp, sử dụng trung bình động của gradient và bình phương gradient. Công thức:

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)\nabla J(\theta)$$

$$s_t = \beta_2 s_{t-1} + (1-\beta_2)\nabla J(\theta)^2$$

$${v_t}^{\text{corrected}} = \frac{v_t}{1-{\beta_1}^t}$$

$${s_t}^{\text{corrected}} = \frac{s_t}{1-{\beta_2}^t}$$

$$\theta = \theta - \frac{n}{\sqrt{{s_t}^{\text{corrected}}} + \epsilon}{v_t}^{\text{corrected}}$$

**Trong đó:**

- $v_t$: Trung bình động của gradient (momentum).
- $s_t$: Trung bình động của bình phương gradient.
- $\beta 1, \beta 2$ : Hệ số điều chỉnh ($\beta 1$=0.9,$\beta 2$=0.999).

**Ưu điểm:**

- Hiệu quả với gradient không ổn định.
- Kết hợp ưu điểm của Momentum và RMSProp.

## 2. Thực nghiệm

### *1. Phân tích Dataset*

1.1. Tải dữ liệu Titanic

Dữ liệu Titanic được tải về sử dụng thư viện seaborn:

```
import seaborn as sns
# Load Titanic Dataset
titanic = sns.load_dataset('titanic')
```

Hình 1. 1 Thư viện Seaborn

1.2. Tiền xử lý dữ liệu

**1.2.1 Xóa các hàng có giá trị khuyết:**

- Các hàng có giá trị khuyết trong các cột age và embarked được loại bỏ.

**1.2.2 Chuyển đổi các các cột dữ liệu thành dạng số:**

- Cột sex (để hiển thị giới tính) được chuyển thành 0 (“male”) và 1 (“female”).
- Cột embarked (để hiển thị nơi lên tàu) được chuyển thành 0 (C), 1 (Q), và 2 (S).

**1.2.3 Chọn các thuộc tính quan trọng:**

- Các thuộc tính được chọn bao gồm:
- pclass: hạng ghế
- sex: giới tính
- age: tuổi
- sibsp: số anh/chị/em ruột đi cùng
- parch: số cha/mẹ con đi cùng

- fare: giá vé
- embarked: nơi lên tàu

#### 1.2.4 Chuẩn hóa dữ liệu:

- Sử dụng StandardScaler để chuẩn hóa các thuộc tính về trung bình 0 và độ lệch chuẩn 1.

#### 1.2.5 Chia tập dữ liệu:

- Tập dữ liệu được chia thành hai phần:
- Tập huấn: 80%
- Tập kiểm tra: 20%

## *2. Huấn luyện mô hình Logistic Regression*

### 2.1. Hàm sigmoid

Hàm sigmoid được sử dụng để chuyển đổi giá trị dự đoán thành xác suất trong khoảng [0, 1].

```
# Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Hình 1. 2 Hàm sigmoid

### 2.2. Hàm loss

Hàm loss được tính bằng cross-entropy để đánh giá độ lệch giữa giá trị dự đoán và giá trị thực tế.

```
# Loss Function
def compute_loss(y, y_pred):
    return -np.mean(y * np.log(y_pred + 1e-8) + (1 - y) * np.log(1 - y_pred + 1e-8))
```

Hình 1. 3 Hàm Loss

### 2.3. Thuật toán tối ưu hóa

Lớp Optimizer được xây dựng để hỗ trợ nhiều thuật toán tối ưu hóa:

```python
class Optimizer:
    def __init__(self, method, lr=0.01, beta=0.9, epsilon=1e-8, batch_size=None):
        self.method = method
        self.lr = lr
        self.beta = beta  # Momentum beta
        self.epsilon = epsilon
        self.v = 0  # Momentum term
        self.s = 0  # RMSProp term
        self.t = 0  # Time step for Adam
        self.batch_size = batch_size  # For Mini-Batch GD

    def update(self, dw, dw_prev=None):
        if self.method == "sgd":
            return -self.lr * dw
        elif self.method == "momentum":
            self.v = self.beta * self.v + (1 - self.beta) * dw
            return -self.lr * self.v
        elif self.method == "adagrad":
            self.s += dw ** 2
            return -self.lr * dw / (np.sqrt(self.s) + self.epsilon)
        elif self.method == "rmsprop":
            self.s = self.beta * self.s + (1 - self.beta) * (dw ** 2)
            return -self.lr * dw / (np.sqrt(self.s) + self.epsilon)
        elif self.method == "adam":
            self.t += 1
            self.v = self.beta * self.v + (1 - self.beta) * dw
            self.s = self.beta * self.s + (1 - self.beta) * (dw ** 2)
            v_corrected = self.v / (1 - self.beta ** self.t)
            s_corrected = self.s / (1 - self.beta ** self.t)
            return -self.lr * v_corrected / (np.sqrt(s_corrected) + self.epsilon)
        elif self.method == "batch_gd":
            return -self.lr * dw
        elif self.method == "mini_batch_gd":
            return -self.lr * dw
```

Hình 1. 4 Thuật toán tối ưu hóa

## 2.4. Huấn luyện Logistic Regression

Mô hình Logistic Regression được huấn luyện bằng nhiều phương pháp tối ưu hóa khác nhau.

```python
def logistic_regression(X_train, y_train, X_test, y_test, optimizer_name, epochs=100, lr=0.01, batch_size=None):
    m, n = X_train.shape
    weights = np.zeros(n)
    bias = 0
    optimizer = Optimizer(optimizer_name, lr, batch_size=batch_size)
    losses = []
    epoch_accuracies = []  # List to store accuracy after each epoch

    for epoch in range(epochs):
        if optimizer_name == "mini_batch_gd":
            # Mini-Batch Gradient Descent
            indices = np.random.choice(m, batch_size, replace=False)
            X_batch = X_train[indices]
            y_batch = y_train[indices]
        else:
            X_batch = X_train
            y_batch = y_train

        y_pred = sigmoid(np.dot(X_batch, weights) + bias)  # y_pred has shape (batch_size,)
        dw = np.dot(X_batch.T, (y_pred - y_batch)) / len(y_batch)  # dw has shape (n,)
        db = np.sum(y_pred - y_batch) / len(y_batch)  # db is scalar

        # Update weights using optimizer
        weights += optimizer.update(dw)

        # Update bias directly (no optimizer needed for bias)
        bias -= optimizer.lr * db

        loss = compute_loss(y_batch, y_pred)
        losses.append(loss)

        # Predict on test set
        y_pred_test = sigmoid(np.dot(X_test, weights) + bias) >= 0.5
        acc = accuracy_score(y_test, y_pred_test)
        epoch_accuracies.append(acc)  # Store accuracy for this epoch

    return losses, epoch_accuracies
```

Hình 1. 5 Huấn luyện logistic regression

## *3. Phân tích từng thuật toán*

### 3.1. Gradient Descent (GD)

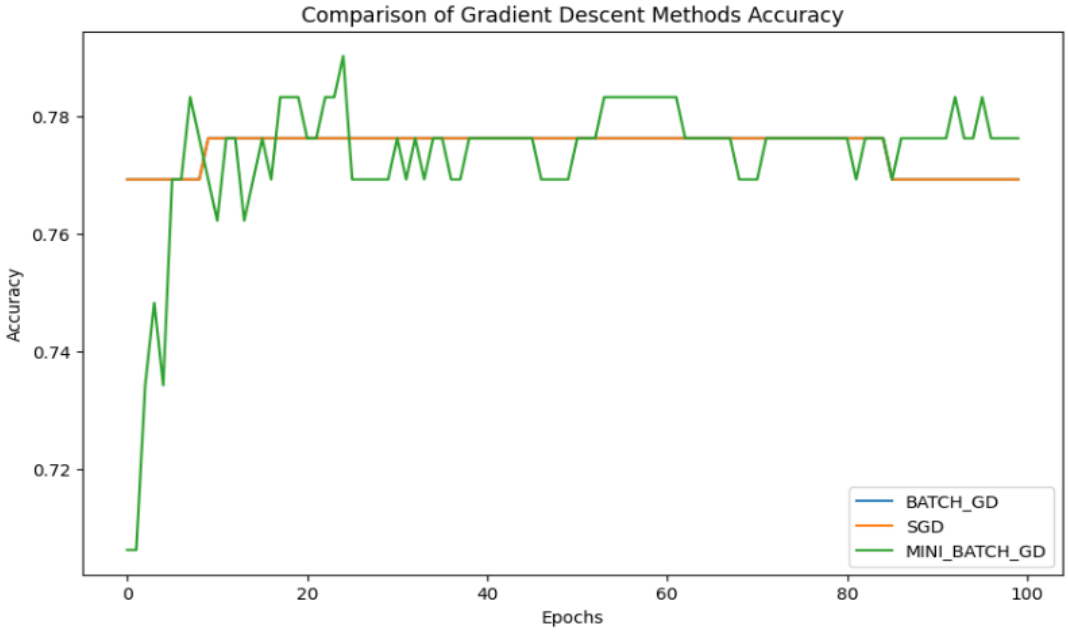
#### 3.1.1 Code

```python
# Gradient Descent (GD) Methods
methods_gd = ["batch_gd", "sgd", "mini_batch_gd"]
results_loss_gd = {}
results_acc_gd = {}

for method in methods_gd:
    batch_size = 32 if method == "mini_batch_gd" else None
    losses, acc = logistic_regression(X_train, y_train, X_test, y_test, method, epochs=100, lr=0.01, batch_size=batch_size)
    results_loss_gd[method] = losses
    results_acc_gd[method] = acc
    print(f"Method {method.upper()}: Accuracy = {acc[-1]:.4f}")
```

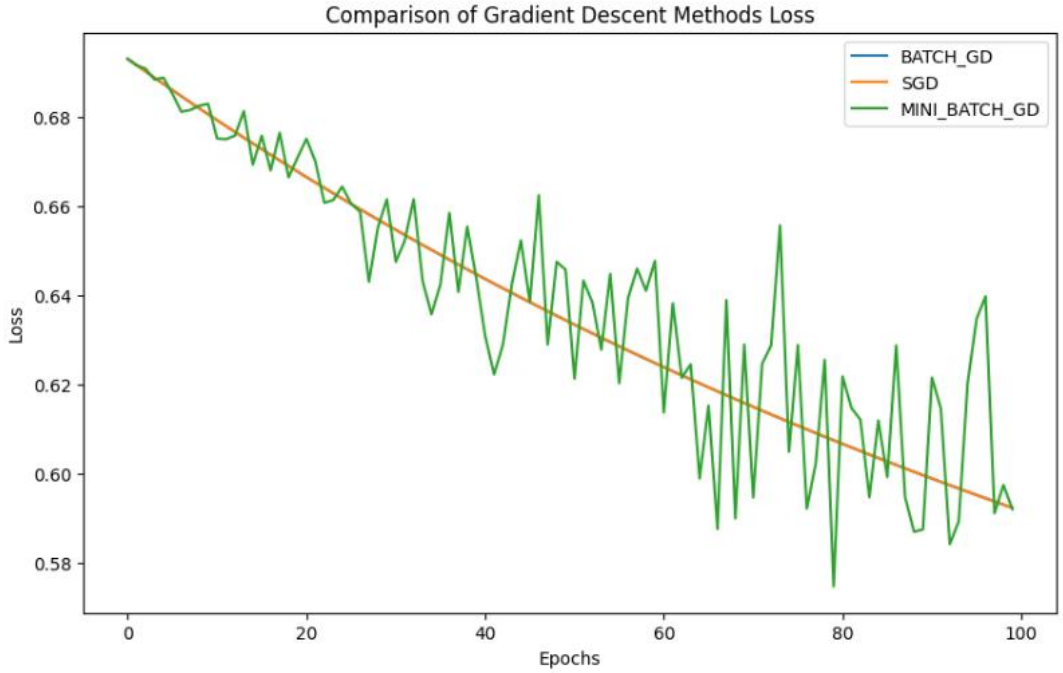
Hình 1. 6 Phương pháp Gradient Descent (GD)

### 3.1.2. Kết quả

```
Method BATCH_GD: Accuracy = 0.7692
Method SGD: Accuracy = 0.7692
Method MINI_BATCH_GD: Accuracy = 0.7762
```




Hình 1. 7 Kết quả Accuracy Gradient Descent (GD)




Hình 1. 8 Kết quả Loss Gradient Descent (GD)

### 3.1.3. Đánh giá

#### a. Phân tích các phương pháp Gradient Descent (GD)

Gradient Descent (GD) là một phương pháp tối ưu hóa để cập nhật các tham số của mô hình nhằm giảm thiểu hàm mất mát. Trong bài này, có 3 biến thể của GD được sử dụng:

- **Batch Gradient Descent (BATCH_GD):**
  - Sử dụng toàn bộ dữ liệu huấn luyện để tính toán gradient trong mỗi epoch.
  - Ưu điểm: Đảm bảo gradient chính xác, ổn định.
  - Nhược điểm: Chậm với các tập dữ liệu lớn do cần tính toán trên toàn bộ dữ liệu mỗi lần cập nhật.
- **Stochastic Gradient Descent (SGD):**
  - Sử dụng một mẫu dữ liệu ngẫu nhiên để tính gradient tại mỗi lần cập nhật.
  - Ưu điểm: Nhanh hơn BATCH_GD, phù hợp với dữ liệu lớn.
  - Nhược điểm: Gradient không ổn định, dễ dao động xung quanh điểm hội tụ.
- **Mini-Batch Gradient Descent (MINI_BATCH_GD):**
  - Sử dụng một nhóm nhỏ dữ liệu (mini-batch) để tính gradient tại mỗi lần cập nhật.
  - Ưu điểm: Kết hợp giữa BATCH_GD và SGD, vừa nhanh vừa ổn định hơn SGD.
  - Nhược điểm: Hiệu suất phụ thuộc vào kích thước mini-batch.

#### b. Kết quả Accuracy

- ***BATCH_GD***: Accuracy = 0.7692
- ***SGD: Accuracy*** = 0.7692
- ***MINI_BATCH_GD***: Accuracy = 0.7762

- ***Nhận xét:***
    - Cả 3 phương pháp đều đạt được độ chính xác tương tự, với MINI_BATCH_GD cao hơn một chút (0.7762 so với 0.7692).
    - Điều này cho thấy các phương pháp hội tụ tốt trong bài toán Logistic Regression trên dữ liệu Titanic.
    - MINI_BATCH_GD có thể khai thác sự cân bằng giữa tính ổn định của BATCH_GD và tốc độ của SGD, dẫn đến kết quả tốt hơn.

**c. Phân tích Loss**

- Loss giảm dần theo các epoch đối với cả 3 phương pháp, chứng minh rằng các thuật toán hội tụ đúng hướng.
- MINI_BATCH_GD có xu hướng hội tụ nhanh hơn và ổn định hơn so với SGD, do giảm được sự dao động khi cập nhật gradient.

**d. Ưu và nhược điểm từng phương pháp trong bài toán**

| Phương pháp | Ưu điểm | Nhược điểm |
|---|---|---|
| **BATCH_GD** | Kết quả ổn định, hội tụ đều đặn. | Chậm khi dữ liệu lớn, không phù hợp với ứng dụng thời gian thực. |
| **SGD** | Nhanh, phù hợp với dữ liệu lớn. | Dao động mạnh, không ổn định, có thể mất nhiều thời gian để hội tụ. |
| **MINI_BATCH_GD** | Kết hợp ưu điểm của cả hai phương pháp trên, ổn định và nhanh hơn. | Cần chọn kích thước batch hợp lý, có thể ảnh hưởng đến kết quả. |

**e. Kết luận**

- **MINI_BATCH_GD** là phương pháp phù hợp nhất trong bài toán này, vì nó đạt được độ chính xác cao nhất (0.7762) và có sự cân bằng giữa tốc độ và độ ổn định.
- **SGD** có thể cải thiện nếu sử dụng các phương pháp như giảm tốc độ học (learning rate decay) để giảm dao động trong quá trình hội tụ.
- **BATCH_GD** không tối ưu về mặt thời gian, nhưng có thể phù hợp với các bài toán nhỏ hoặc yêu cầu tính toán chính xác cao.

Nếu dữ liệu lớn hơn hoặc bài toán phức tạp hơn, **MINI_BATCH_GD** hoặc các biến thể như Adam Optimizer sẽ là lựa chọn tốt hơn.

## 3.2. Momentum

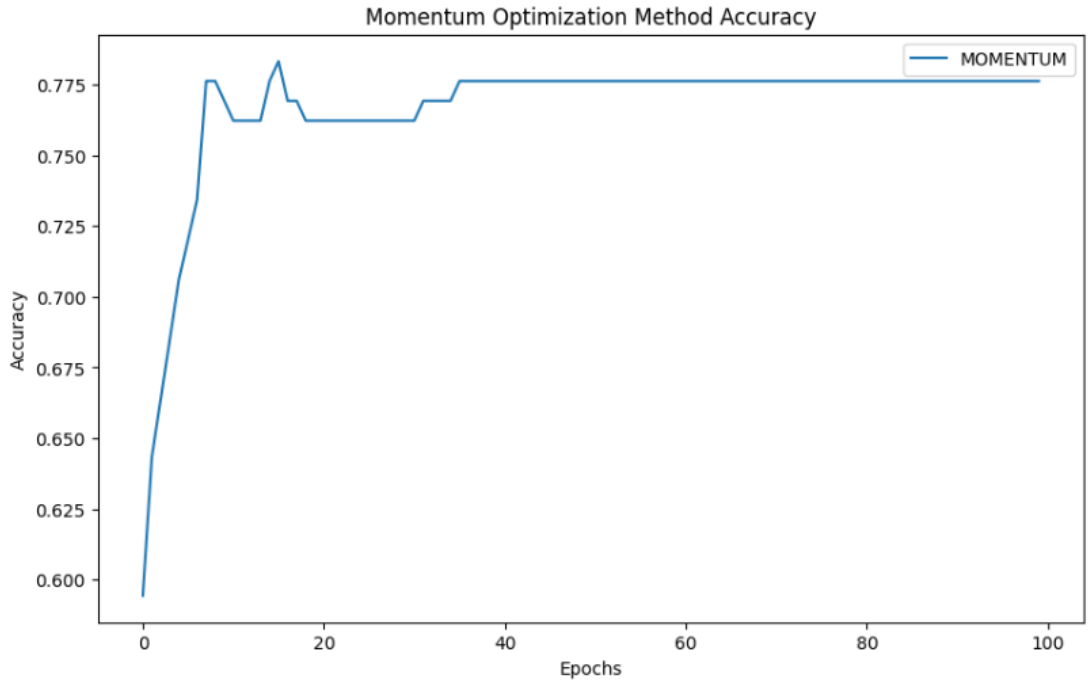
### 3.2.1 Code

```
# Momentum Method
method = "momentum"
losses, acc = logistic_regression(X_train, y_train, X_test, y_test, method, epochs=100, lr=0.01)
print(f"Method {method.upper()}: Accuracy = {acc[-1]:.4f}")
```

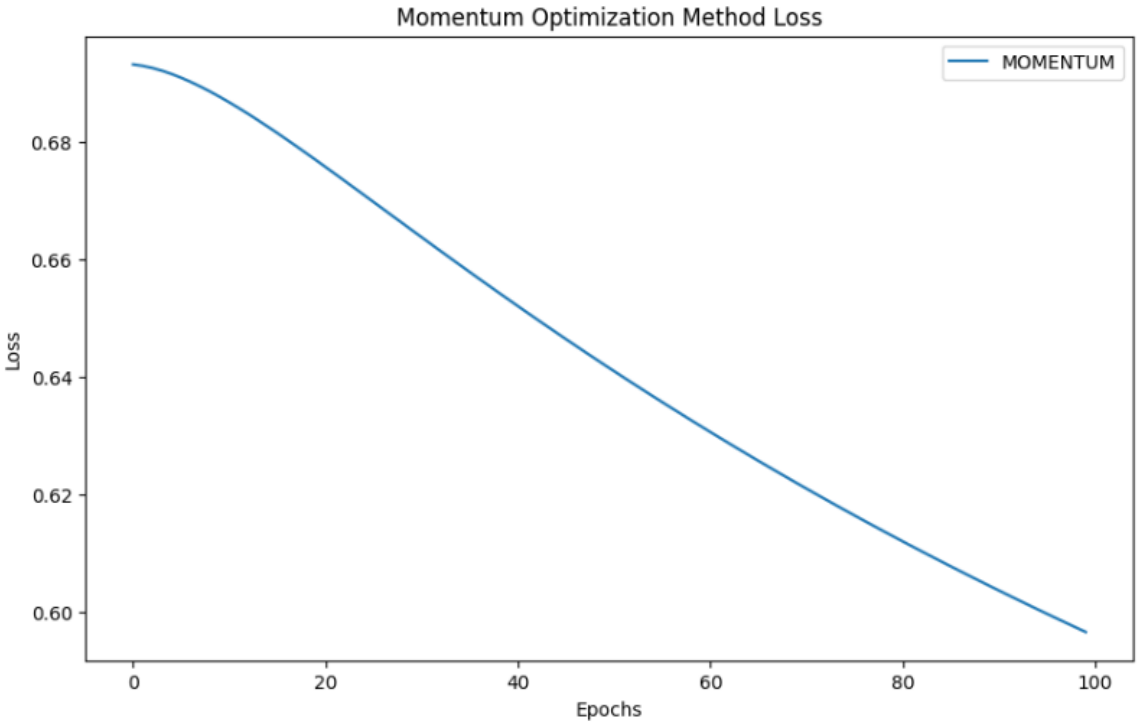
Hình 1. 9 Phương pháp Momentum

### 3.2.2 Kết quả

```
Method MOMENTUM: Accuracy = 0.7762
```




Hình 1. 10 Kết quả Accuracy Momentum




Hình 1. 11 Kết quả Loss Momentum

### 3.2.3 Đánh giá

**a. Phân tích phương pháp Momentum**

Momentum là một kỹ thuật tối ưu hóa được sử dụng để tăng tốc độ hội tụ của Gradient Descent, đặc biệt trong các bài toán có bề mặt hàm mất mát phức tạp.

- **Nguyên lý hoạt động:**
  - Momentum sử dụng một thành phần "động lượng" (velocity) để duy trì và tăng cường hướng đi trong quá trình tối ưu hóa.
  - Điều này giúp giảm dao động khi gradient thay đổi nhanh chóng, đặc biệt trong các thung lũng dốc.
- **Ưu điểm:**
  - Tăng tốc độ hội tụ, đặc biệt khi gradient có hướng không ổn định.
  - Giảm thiểu dao động, giúp tối ưu hóa hiệu quả hơn.
- **Nhược điểm:**
  - Phụ thuộc vào tham số động lượng $\gamma$.

**b. Kết quả Accuracy**

- **Momentum Accuracy**: 0.7762
  - Kết quả tương đương với MINI_BATCH_GD trong bài trước.
  - Điều này cho thấy Momentum hoạt động hiệu quả, hội tụ nhanh và đạt được độ chính xác cao.

**c. Phân tích Loss**

- Loss giảm đều qua các epoch, chứng tỏ phương pháp hội tụ ổn định.
- Sự giảm của Loss có thể nhanh hơn so với SGD hoặc MINI_BATCH_GD nhờ thành phần động lượng giúp giảm dao động.

**d. Kết luận**

- Momentum là một cải tiến đáng kể so với các phương pháp Gradient Descent cơ bản (SGD, Mini-Batch GD).

- Với độ chính xác đạt 0.7762, Momentum cho thấy khả năng hội tụ nhanh và ổn định hơn, đặc biệt phù hợp với các bài toán phức tạp.
- Trong thực tế, Momentum thường được kết hợp với các kỹ thuật tối ưu khác như Nesterov Accelerated Gradient (NAG) hoặc Adam Optimizer để đạt hiệu quả tối ưu.

## 3.3. Adagrad (Adaptive Gradient Algorithm)

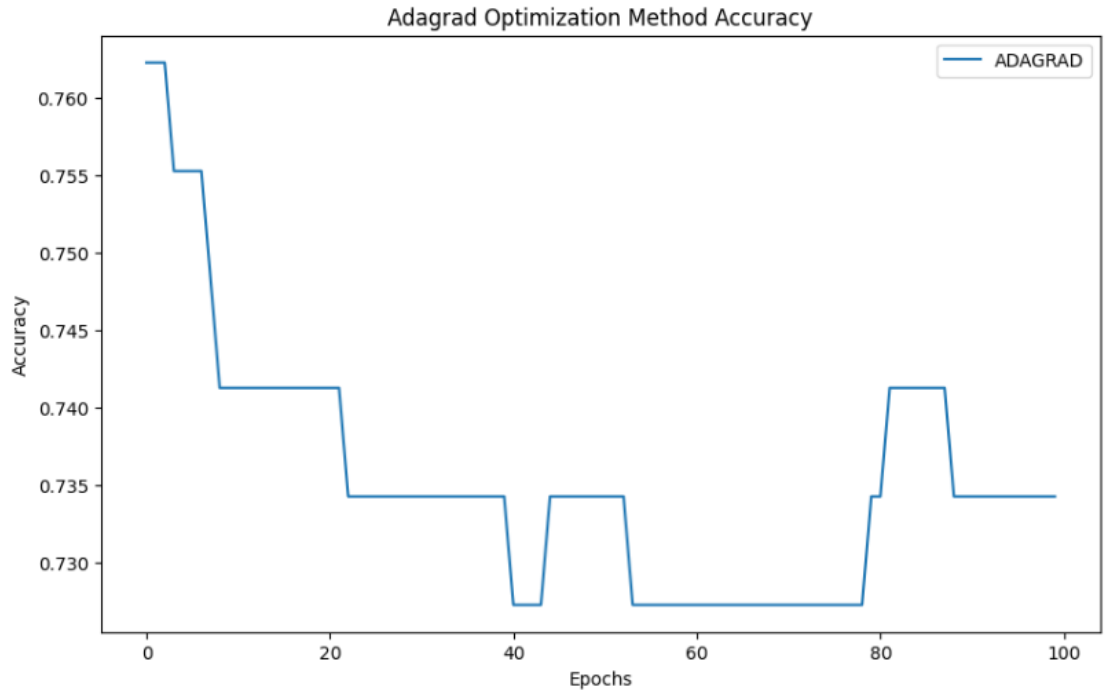
### 3.3.1. Code

```
# Adagrad Method
method = "adagrad"
losses, acc = logistic_regression(X_train, y_train, X_test, y_test, method, epochs=100, lr=0.01)
print(f"Method {method.upper()}: Accuracy = {acc[-1]:.4f}")
```

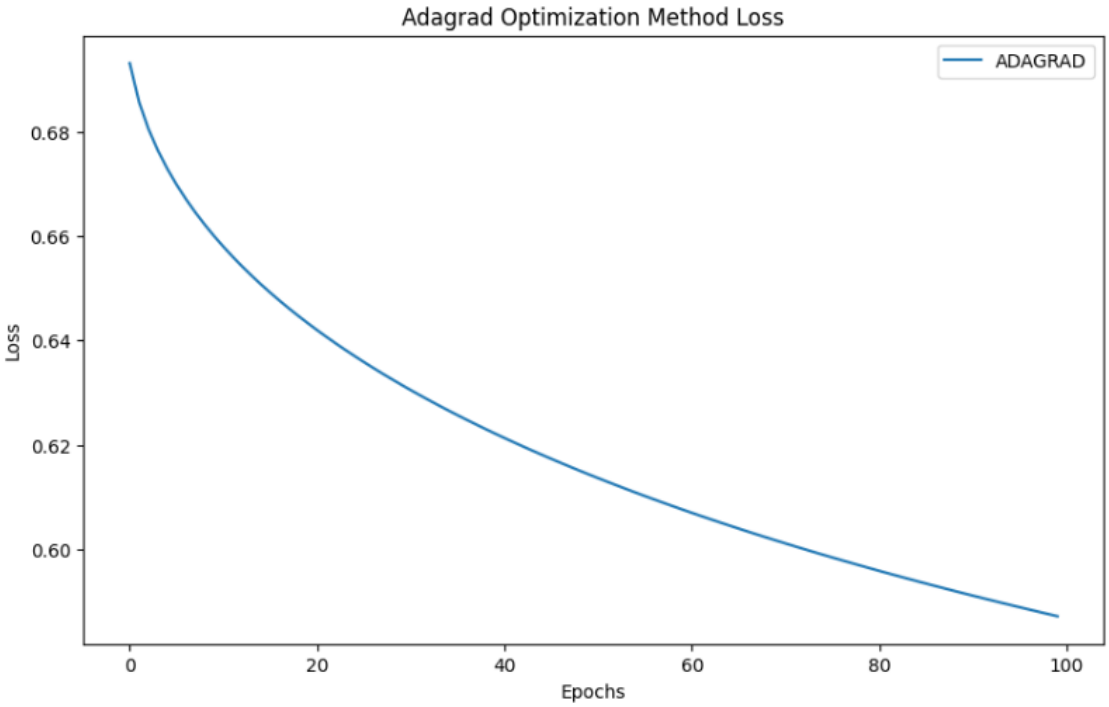
Hình 1. 12 Phương pháp Adagrad (Adaptive Gradient Algorithm)

### 3.3.2. Kết quả

```
Method ADAGRAD: Accuracy = 0.7343
```




Hình 1. 13 Kết quả Accuracy Adagrad (Adaptive Gradient Algorithm)




Hình 1. 14 Kết quả Loss Adagrad (Adaptive Gradient Algorithm)

### 3.3.3. Đánh giá

**a. Phân tích phương pháp Adagrad**

Adagrad là một thuật toán tối ưu hóa tự điều chỉnh tốc độ học (learning rate) cho từng tham số trong quá trình huấn luyện.

Nguyên lý hoạt động:

- Adagrad cập nhật tốc độ học dựa trên tổng bình phương của gradient từ các epoch trước đó.
- Tốc độ học giảm dần theo thời gian, đặc biệt với các tham số có gradient lớn.

Ưu điểm : Tự động điều chỉnh learning rate.

Nhược điểm: Tốc độ học giảm quá nhanh, khiến hội tụ sớm và kém.

**b. Kết quả Accuracy**

- Accuracy cuối cùng: 0.7343
- Biểu đồ cho thấy độ chính xác giảm dần qua các epoch và không có sự cải thiện rõ rệt.

- Kết quả thấp hơn so với Momentum (0.7762) và các phương pháp GD (0.7692 - 0.7762).

**c. Phân tích Loss**

Tốc độ học giảm dần khiến gradient cập nhật nhỏ đi.

**d. Kết luận**

- Adagrad có khả năng tự điều chỉnh tốc độ học nhưng thường giảm quá nhanh, dẫn đến hội tụ sớm ở kết quả không tối ưu.
- Với Accuracy chỉ đạt 0.7343, Adagrad kém hiệu quả hơn so với Momentum và các phương pháp GD khác.

Để khắc phục, có thể thử các thuật toán cải tiến như RMSProp hoặc Adam Optimizer, vì chúng giải quyết vấn đề tốc độ học giảm quá nhanh của Adagrad.

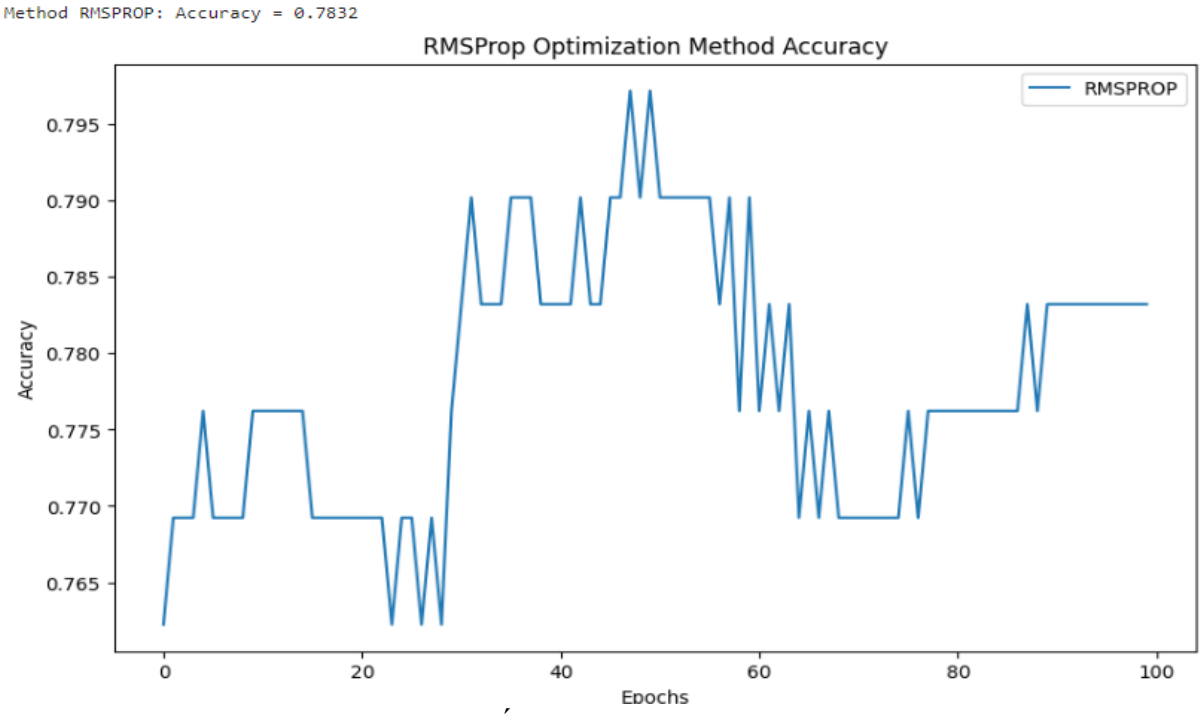
## 3.4. RMSProp (Root Mean Square Propagation)

### 3.4.1. Code

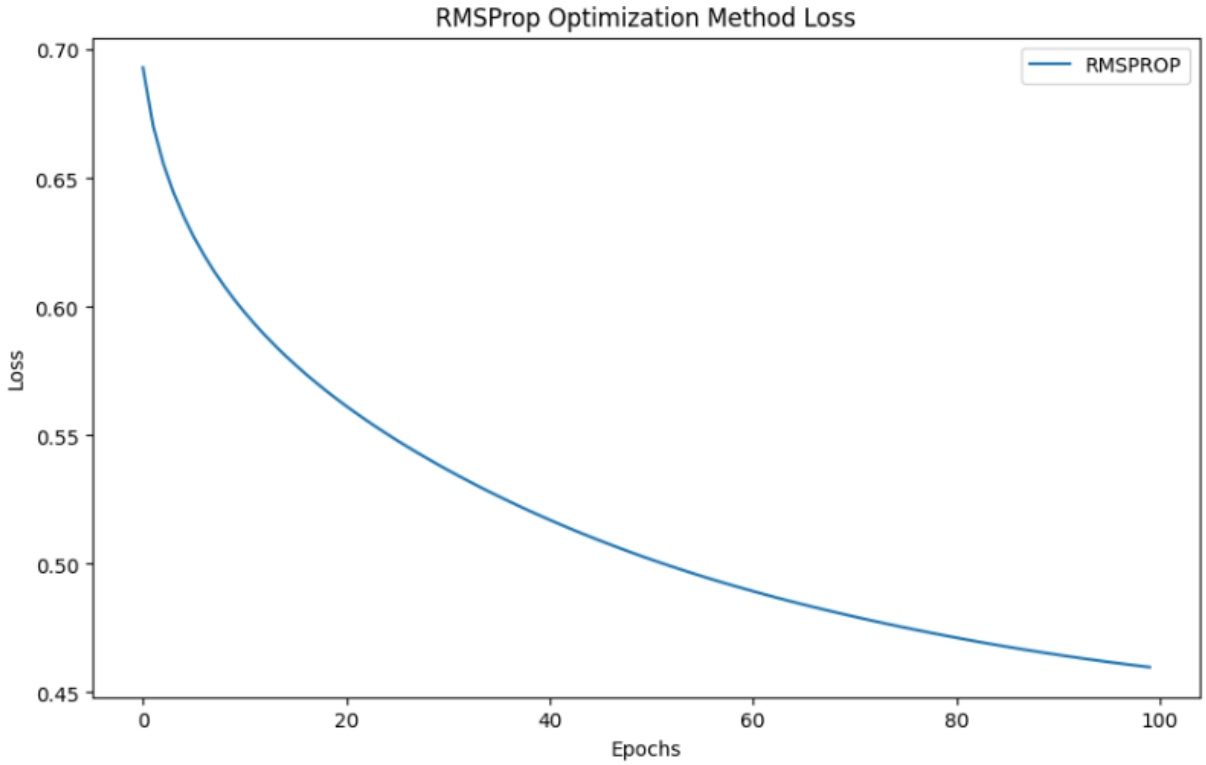
```
# RMSProp Method
method = "rmsprop"
losses, acc = logistic_regression(X_train, y_train, X_test, y_test, method, epochs=100, lr=0.01)
print(f"Method {method.upper()}: Accuracy = {acc[-1]:.4f}")
```

Hình 1. 15 Phương pháp RMSProp (Root Mean Square Propagation)

### 3.4.2. Kết quả




Hình 1. 16 Kết quả Accuracy RMSProp




Hình 1. 17 Kết quả Loss RMSProp

### 3.4.3. Đánh giá

#### a. Phân tích phương pháp RMSProp

- RMSProp (Root Mean Square Propagation) là một thuật toán tối ưu hóa được sử dụng khi làm việc với các mô hình có nhiều tham số.
- Thuật toán điều chỉnh tốc độ học cho từng tham số dựa trên bình phương trung bình của các gradient trước đó.
- Sử dụng một yếu tố beta để điều chỉnh trọng số của các gradient trong quá khứ, giúp ổn định quá trình học và giảm thiểu sự dao động trong các cập nhật tham số.

#### b. Kết quả Accuracy

- Kết quả accuracy của thuật toán RMSProp đạt 78.32% cho mô hình Logistic Regression trên bộ dữ liệu Titanic.
- Mô hình có khả năng phân loại khá tốt giữa các hành khách sống sót và không sống sót dựa trên các đặc trưng như độ tuổi, giới tính, hạng vé, số người đi cùng, và nơi lên tàu.
- Mô hình có thể áp dụng hiệu quả trong các tình huống phân loại cơ bản, đặc biệt khi dữ liệu không quá phức tạp.

#### c. Phân tích Loss

Loss giảm dần qua các epoch cho thấy quá trình huấn luyện của mô hình ổn định và thuật toán tối ưu đang học tốt từ dữ liệu. Thuật toán RMSProp điều chỉnh tham số của mô hình hiệu quả, giúp cải thiện độ chính xác của dự đoán qua từng vòng lặp.

#### d. Kết luận

Thuật toán RMSProp điều chỉnh tốc độ học cho từng tham số, giúp giảm sự dao động trong huấn luyện, hoạt động tốt khi dữ liệu có sự thay đổi lớn giữa các đặc trưng, cải thiện độ ổn định và tốc độ học so với SGD. Tuy nhiên, có thể không đạt được hiệu suất tối đa như các thuật toán khác (ví dụ: Adam), không yêu cầu quá nhiều điều chỉnh, nhưng vẫn cần tìm các siêu tham số phù hợp.

## 3.5. Adam (Adaptive Moment Estimation)

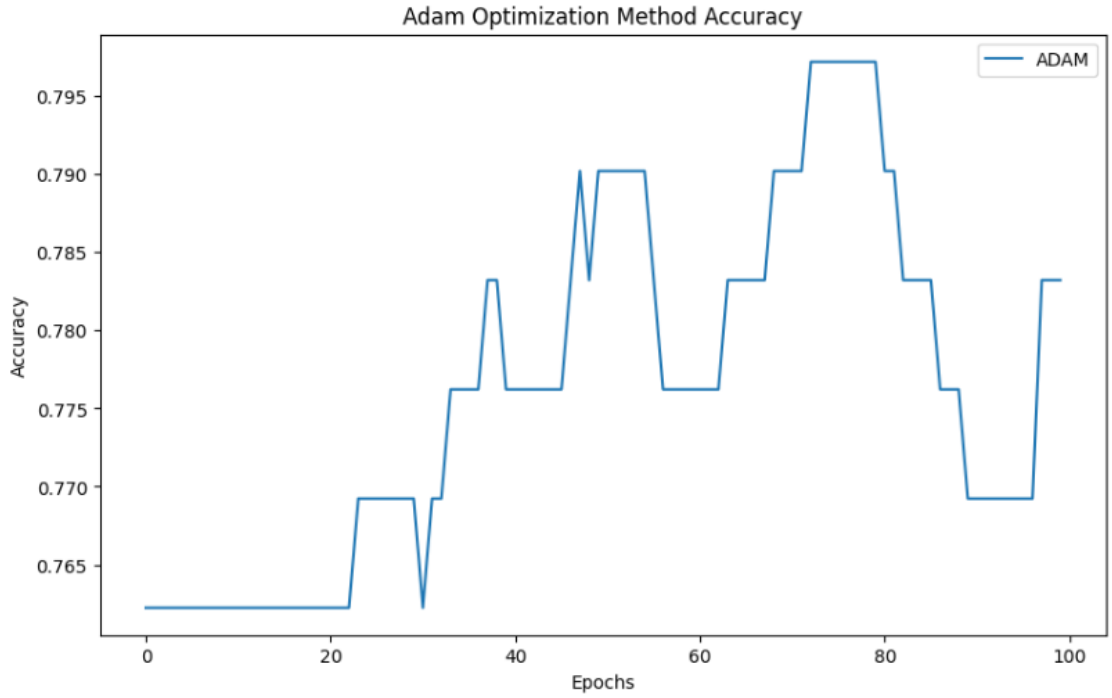
### 3.5.1. Code

```
# Adam Method
method = "adam"
losses, acc = logistic_regression(X_train, y_train, X_test, y_test, method, epochs=100, lr=0.01)
print(f"Method {method.upper()}: Accuracy = {acc[-1]:.4f}")
```

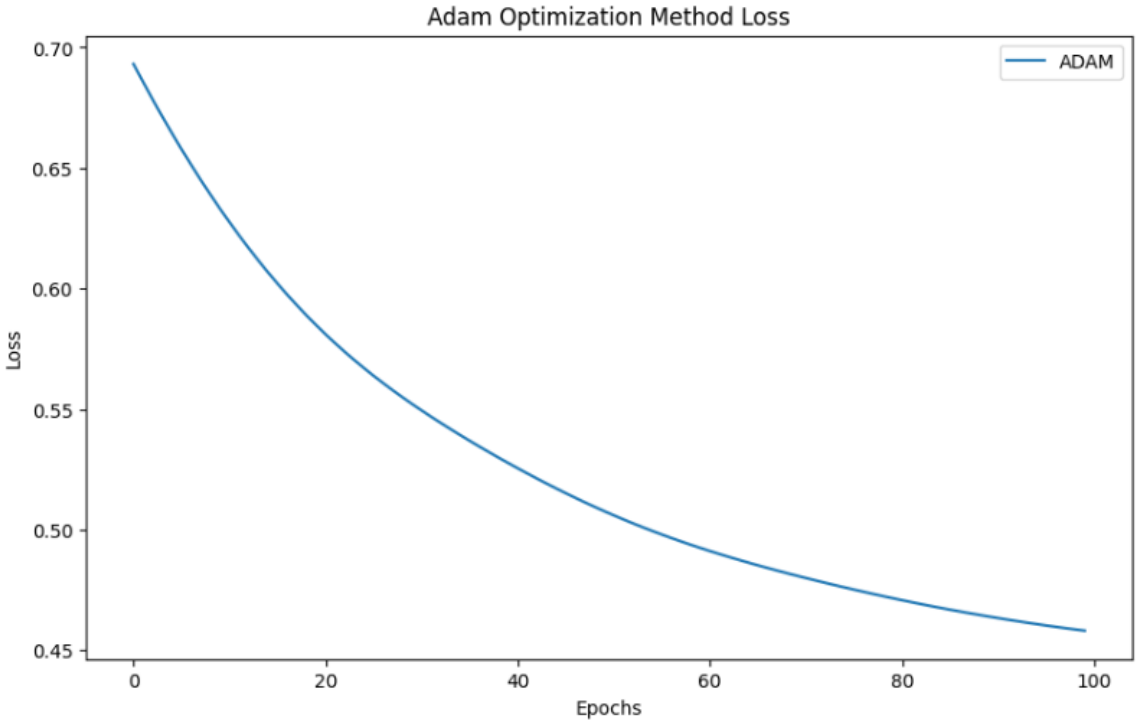
Hình 1. 18 Phương pháp Adam (Adaptive Moment Estimation)

### 3.5.2. Kết quả

```
Method ADAM: Accuracy = 0.7832
```




Hình 1. 19 Kết quả Accuracy Adam




Hình 1. 20 Kết quả Loss Adam

### 3.5.3. Đánh giá

**a. Phân tích phương pháp Adam**

Adam là thuật toán tối ưu hóa kết hợp ưu điểm của Momentum và RMSProp. Nó sử dụng thông tin về gradient và bình phương gradient để điều chỉnh tốc độ học cho từng tham số. Adam tự động điều chỉnh tốc độ học cho mỗi tham số, giúp tăng tốc độ hội tụ và giảm dao động trong quá trình huấn luyện.

**b. Kết quả Accuracy**

Accuracy = 0.7832 cho thấy mô hình Logistic Regression tối ưu hóa bằng Adam đạt độ chính xác khá cao trong việc phân loại dữ liệu Titanic. Điều này chứng tỏ Adam đã giúp mô hình học tốt và đạt được kết quả phân loại chính xác hơn.

**c. Phân tích Loss**

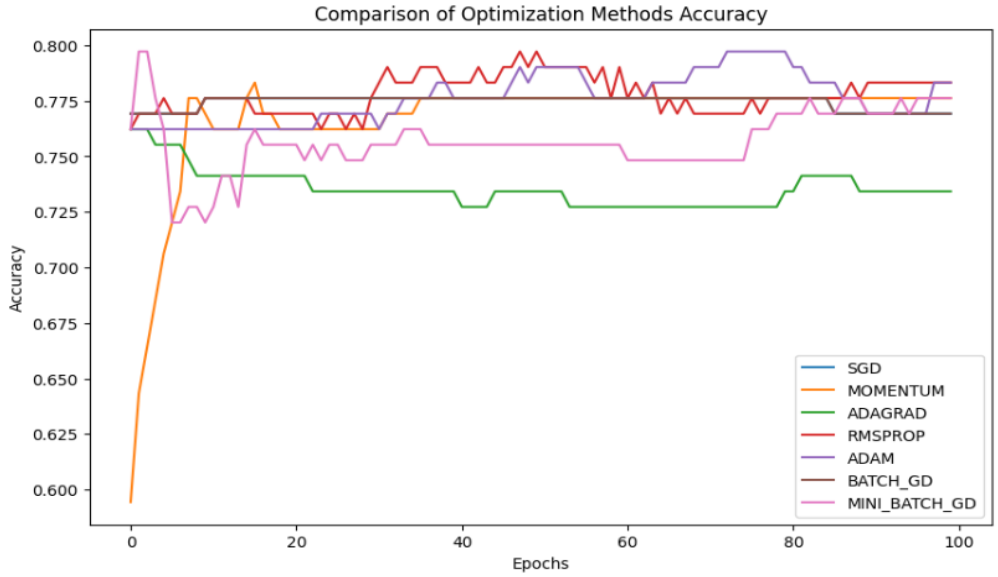
Loss giảm dần trong quá trình huấn luyện, cho thấy Adam đang giúp mô hình học tốt hơn và giảm sai số qua từng epoch. Điều này chứng tỏ thuật toán đang tối ưu hóa hiệu quả và giúp mô hình hội tụ nhanh chóng.

**d. Kết luận**

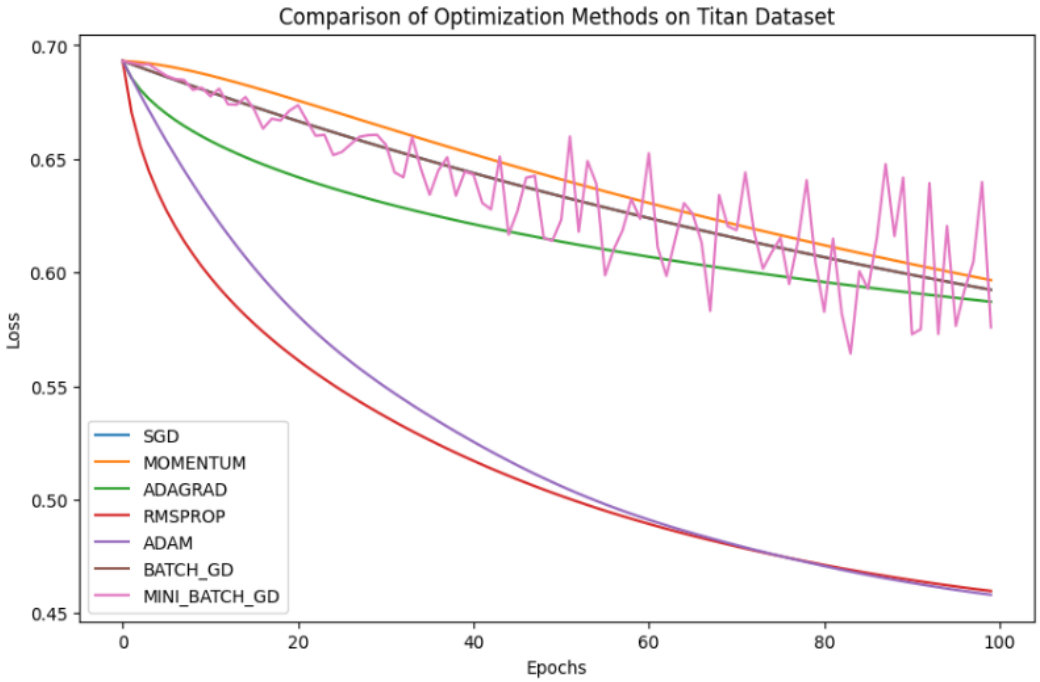
Thuật toán Adam đã hoạt động hiệu quả trong việc tối ưu hóa Logistic Regression, đạt độ chính xác cao và loss giảm dần. Adam kết hợp Momentum và RMSProp, giúp tối ưu hóa nhanh và ổn định, không cần điều chỉnh thủ công tốc độ học, thích hợp cho các bài toán phức tạp.

## 4. So sánh

```
Method SGD: Accuracy = 0.7692
Method MOMENTUM: Accuracy = 0.7762
Method ADAGRAD: Accuracy = 0.7343
Method RMSPROP: Accuracy = 0.7832
Method ADAM: Accuracy = 0.7832
Method BATCH_GD: Accuracy = 0.7692
Method MINI_BATCH_GD: Accuracy = 0.7762
```




Hình 1. 21 So sánh Accuracy các phương pháp tối ưu




Hình 1. 22 So sánh Loss các phương pháp tối ưu

**4.1. Loss qua các epoch**

- **RMSPROP** và **ADAM** có tốc độ giảm loss nhanh nhất và ổn định nhất. Cả hai phương pháp đều đạt loss thấp hơn so với các phương pháp khác khi số epoch tăng lên.
- **SGD** và **Momentum** có xu hướng giảm loss chậm hơn và ổn định, nhưng không đạt được mức loss thấp như **RMSPROP** và **ADAM**.
- **Mini Batch Gradient Descent** thể hiện sự biến động khá lớn, loss dao động mạnh và kém ổn định hơn so với các phương pháp còn lại.
- **ADAGRAD** giảm loss chậm và ổn định nhưng không đạt mức loss thấp nhất.

**4.2. Accuracy qua các epoch**

- RMSPROP và ADAM đạt độ chính xác (accuracy) cao nhất, khoảng 0.7832 và 0.7802, và ổn định sau khi đạt ngưỡng.
- Momentum và Mini Batch Gradient Descent cũng thể hiện độ chính xác tương đối tốt, khoảng 0.7762, nhưng không cao bằng RMSPROP và ADAM.

- SGD và Batch Gradient Descent có độ chính xác tương tự nhau (0.7692) và ổn định.
- ADAGRAD có độ chính xác thấp nhất, khoảng 0.7343, và không cải thiện nhiều trong suốt quá trình học.

### 4.3 Đánh giá chung

| Phương pháp | Tốc độ hội tụ | Ổn định | Phù hợp với dữ liệu lớn |
|---|---|---|---|
| **Batch GD** | Chậm | Cao | Không |
| **SGD** | Nhanh | Thấp | Có |
| **Mini-Batch** | Trung bình | Trung bình | Có |
| **Momentum** | Nhanh hơn GD | Cao | Có |
| **Adagrad** | Trung bình | Trung bình | Có |
| **RMSProp** | Nhanh | Cao | Có |
| **Adam** | Nhanh nhất | Cao | Có |

### 4.4 Tổng kết:

- RMSPROP và ADAM là hai phương pháp tối ưu hóa tốt nhất trong trường hợp này, với khả năng giảm loss nhanh và đạt độ chính xác cao nhất.
- Mini Batch Gradient Descent có lợi thế tốc độ cập nhật nhưng kém ổn định.
- SGD, Momentum và Batch Gradient Descent hoạt động ổn định nhưng hiệu suất không tốt bằng ADAM và RMSPROP.
- ADAGRAD kém hiệu quả nhất trong cả loss và accuracy.

# CHƯƠNG 2 – STOCK PRICE PREDICTION PROBLEM

## 1.1 Thế nào là Stock Price Prediction Problem (Opening Price)

Stock Price Prediction Problem (Opening Price) là một bài toán trong lĩnh vực dự đoán tài chính, tập trung vào việc dự đoán giá mở cửa của một cổ phiếu vào một ngày cụ thể trong tương lai. Giá mở cửa là mức giá mà cổ phiếu được giao dịch đầu tiên khi thị trường chứng khoán mở cửa vào đầu phiên giao dịch.

## 1.2 Mục tiêu của bài toán

Dự đoán giá mở cửa của cổ phiếu trong tương lai dựa trên các yếu tố liên quan, chẳng hạn như:

- Dữ liệu lịch sử giá: Giá mở cửa, giá đóng cửa, giá cao nhất, giá thấp nhất, khối lượng giao dịch.
- Dữ liệu bên ngoài: Tin tức tài chính, sự kiện kinh tế, chính sách của chính phủ, tỷ giá hối đoái, giá vàng, giá dầu, chỉ số thị trường (S&P 500, VN-Index, v.v.).
- Dữ liệu kỹ thuật: Các chỉ báo như RSI, MACD, EMA, Bollinger Bands..

## 1.3 Tính chất của bài toán

- Bài toán hồi quy (Regression Problem): Vì giá cổ phiếu là một giá trị số liên tục, bài toán này thường được xây dựng dưới dạng một bài toán hồi quy.
- Tính ngẫu nhiên cao: Giá cổ phiếu phụ thuộc vào nhiều yếu tố không thể kiểm soát hoặc dự đoán chính xác, chẳng hạn như tâm lý nhà đầu tư, các sự kiện bất ngờ.
- Dữ liệu thời gian (Time Series): Giá cổ phiếu là một chuỗi thời gian, do đó các mô hình cần xử lý tốt dữ liệu theo thứ tự thời gian.

## 1.4 Chuẩn bị dữ liệu:

### 1.4.1 Tải và chuẩn bị dữ liệu cổ phiếu:

- Nguồn dữ liệu: Sử dụng gói yfinance để tải dữ liệu giá cổ phiếu của chỉ số S&P 500 (^GSPC) từ ngày 1/1/2010 đến 1/1/2024.
- Các tính năng được thêm mới:

- Day_of_Week: Ngày giao dịch trong tuần
- Month: Tháng giao dịch
- Year: Năm giao dịch.
- Volume_Change: Tỷ lệ thay đổi khối lượng giao dịch so với ngày trước đó.
- Tomorrow: Giá đóng cửa của ngày tiếp theo (được sử dụng làm biến mục tiêu).

- Tạo các đặc trưng lag (Lag Features): Tính giá đóng cửa của 5 ngày trước đó (Close_Lag_1 đến Close_Lag_5) để cung cấp thêm thông tin về xu hướng giá.
- Xử lý dữ liệu thiếu: Loại bỏ các giá trị NaN do tạo lag.

**1.4.2 Xử lý dữ liệu GDP của Hoa Kỳ:**

- Nguồn dữ liệu: Dữ liệu GDP được tải từ tệp CSV (API_NY.GDP.MKTP.CD_DS2_en_csv_v2_2.csv).
- Giữ lại các cột quan trọng: Chỉ giữ lại dữ liệu GDP của Hoa Kỳ từ năm 2010 đến năm 2023.
- Định dạng dữ liệu:
  - Chuyển đổi dữ liệu từ dạng wide sang long (cột "Year" chứa các năm, cột "GDP" chứa giá trị GDP).
  - Đảm bảo cột "Year" được chuyển đổi thành kiểu số nguyên.
  - Loại bỏ các giá trị bị thiếu (NaN).

**1.4.3 Kết hợp dữ liệu cổ phiếu và GDP:**

- Mục tiêu: Thêm dữ liệu GDP theo từng năm vào dữ liệu cổ phiếu để cung cấp thêm thông tin kinh tế vĩ mô.
- Thực hiện:
  - Gộp hai tập dữ liệu (stock_data và gdp_data) theo cột "Year".
  - Loại bỏ các hàng có giá trị NaN sau khi gộp.

**1.4.4 Chuẩn bị dữ liệu cho mô hình:**

- Predictors (Biến độc lập): Gồm 12 đặc trưng:
    - Các đặc trưng tài chính: Close, Volume, Open, High, Low, các lag của giá đóng cửa (Close_Lag_1 đến Close_Lag_5).
    - Đặc trưng vĩ mô: GDP.
    - Đặc trưng thời gian: Day_of_Week, Month.
- Target : Tomorrow (Giá đóng cửa của ngày tiếp theo).
- Tách dữ liệu:
    - Tập huấn luyện (Train): Tất cả dữ liệu trừ 100 hàng cuối cùng.
    - Tập kiểm tra (Test): 100 hàng cuối cùng của dữ liệu.
- Chuẩn hóa dữ liệu:
    - Sử dụng MinMaxScaler để đưa các biến độc lập vào khoảng [0, 1].
    - Thực hiện chuẩn hóa trên cả tập huấn luyện và kiểm tra.

## 1.5 Phương pháp tiếp cận

Phương pháp thống kê, phân tích, học máy,..

Ở đây chúng ta sẽ tiếp cận theo hướng học máy.

Các phương pháp học máy:

- Feedforward Neural Network
- Recurrent Neural Network (RNN)
- Các phương pháp khác: Linear Regression, SVM, Decision Tree, Random Forest.

### *1.5.1 Feedforward Neural Network*

Mô hình là một trong những kiến trúc cơ bản và phổ biến nhất trong lĩnh vực học máy. Đây là loại mạng nơ-ron nhân tạo không có kết nối vòng lặp, dữ liệu chỉ lan truyền theo một chiều từ đầu vào qua các lớp ẩn và đến đầu ra.

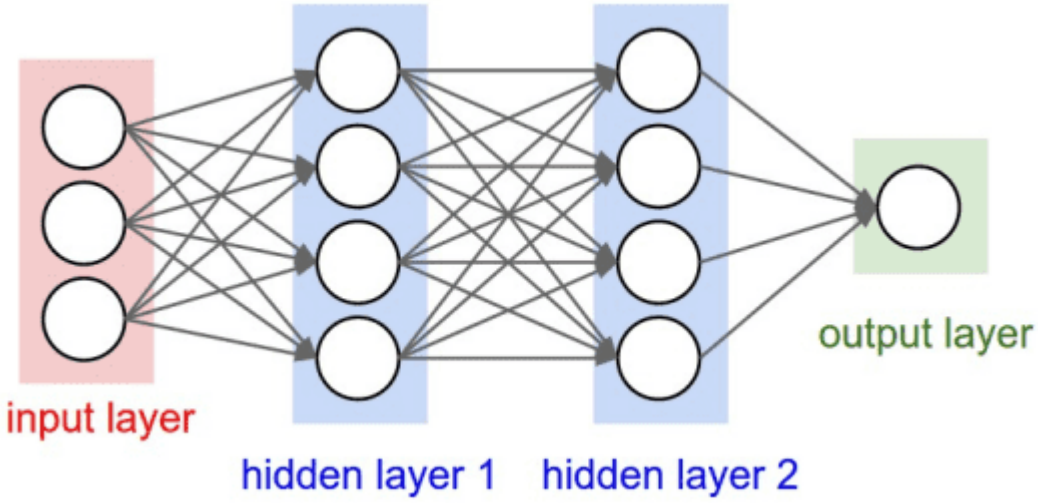
#### 1.5.1.1 Cấu trúc của Feedforward Neural Network

**Feedforward Neural Network được chia thành 3 thành phần chính:**

**Lớp đầu vào (input layer):**Lớp này sẽ đảm nhiệm vai trò nhận dữ liệu đầu vào từ tập dữ liệu, với mỗi neuron trong lớp này sẽ đại diện cho một đặc trưng của tập dữ liệu. Số lượng neuron trong lớp đầu vào bằng số chiều của dữ liệu.

**Lớp ẩn (hidden layers):** Lớp này chứa các neuron sẽ thực hiện các phép toán phi tuyến để trích xuất đặc trưng từ tập dữ liệu. Ở lớp này có thể xuất hiện nhiều lớp ẩn mà số lượng lớp và số neuron trong mỗi lớp thường được điều chỉnh tùy theo tình huống cụ thể.

**Lớp đầu ra (output layer):** Lớp này sẽ chịu trách nhiệm đưa ra dự đoán của mạng, số lượng neuron phụ thuộc vào bài toán.




Hình 2. 1 Cấu trúc Feedforward Neural Network

Nguồn: https://mukulrathi.com/demystifying-deep-learning/feed-forward-neural-network/

#### 1.5.1.2 Hoạt động của Feedforward Neural Network

**1. Lan truyền tiến (Forward Propagation)**

Quá trình lan truyền tiến mô tả cách dữ liệu đầu vào được truyền qua mạng để tạo ra dự đoán. Mỗi neuron thực hiện các bước sau:

- **Tính tổng có trọng số:**

$$z = \sum_{i=1}^{n} w_i \, x_i + b$$

Trong đó:

  - $w_i$: trọng số của neuron.
  - $x_i$: giá trị đầu vào.
  - $b$: bias (độ lệch).

- **Áp dụng hàm kích hoạt (Activation Function):**

$$a = \sigma(z)$$

Hàm kích hoạt giúp tạo ra tính phi tuyến trong mạng.

**2. Lan truyền ngược (Backpropagation)**

- Sau khi dự đoán, mạng so sánh đầu ra dự đoán với giá trị thực tế và tính toán lỗi bằng hàm mất mát (Loss Function).

- Thông qua chuỗi đạo hàm (Chain Rule), gradient của hàm mất mát được tính toán với từng trọng số và bias.
- Các trọng số và bias được cập nhật bằng thuật toán tối ưu hóa như Gradient Descent:

$$w = w - \mu \frac{\partial Loss}{\partial w}$$

μ: tốc độ học.

#### 1.5.1.3 Các hàm thực thi

**1. Hàm kích hoạt (Activation Function)**

Hàm kích hoạt quyết định cách một neuron xử lý đầu vào và tạo ra đầu ra. Một số hàm phổ biến:

- Sigmoid:
  $\sigma(x) = \frac{1}{1+e^{-x}}$
  Giá trị đầu ra từ 0 đến 1.
- **ReLU (Rectified Linear Unit):**

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

- **Tanh:**

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

  Giá trị đầu ra từ -1 đến 1.

**2. Hàm mất mát (Loss Function)**

Đo lường độ sai lệch giữa dự đoán và thực tế: sử dụng Mean Squared Error (MSE) cho bài toán hồi quy và Cross-Entropy Loss cho những bài toán phân loại.

### *1.5.2 Recurrent Neural Networks (RNN)*

- Mạng Nơ-ron Hồi tiếp (Recurrent Neural Networks - RNN) là một lớp mạng nơ-ron được thiết kế đặc biệt để xử lý và phân tích dữ liệu chuỗi thời gian (sequential data), nơi các giá trị đầu vào có mối quan hệ phụ thuộc vào nhau qua thời gian, chẳng hạn như văn bản, chuỗi số, âm thanh, hoặc video.

- RNN khác biệt với các mạng nơ-ron truyền thống như feedforward neural networks ở chỗ chúng có một kết nối hồi tiếp, cho phép RNN có thể ghi nhớ thông tin từ các bước trước và sử dụng chúng để đưa ra dự đoán trong các bước tiếp theo.

- Cấu trúc của RNN:Trong một mạng RNN cơ bản, mỗi bước trong chuỗi thời gian có một "trạng thái ẩn" (hidden state) được cập nhật sau mỗi bước, giúp mạng duy trì thông tin qua thời gian.

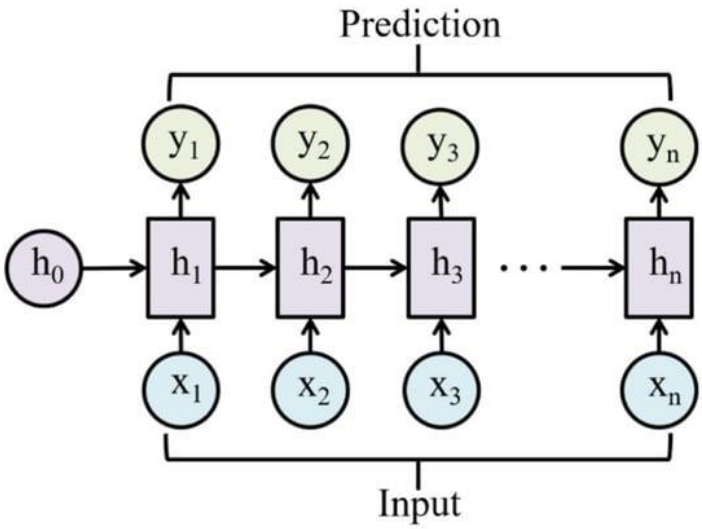
**Công thức tính toán trong RNN:**

- Ở mỗi thời điểm t, trạng thái ẩn $h_t$ được tính toán như sau:
- $h_t = \text{activation}(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$
  - $W_{xh}$: trọng số giữa đầu vào xtx_txt và trạng thái ẩn.
  - $W_{hh}$: trọng số giữa trạng thái ẩn trước đó ht−1h_{t-1}ht−1 và trạng thái ẩn hiện tại.
  - $b_h$ : bias của trạng thái ẩn.
  - activation: hàm sigmoid, tanh, hoặc ReLU, tùy thuộc vào ứng dụng cụ thể.

Trạng thái ẩn này sau đó được sử dụng để tính toán đầu ra $y_t$ của mạng:

$$y_t = W_{hy}h_t + b_y$$

$W_{hy}$ :trọng số giữa trạng thái ẩn $h_t$ và đầu ra.

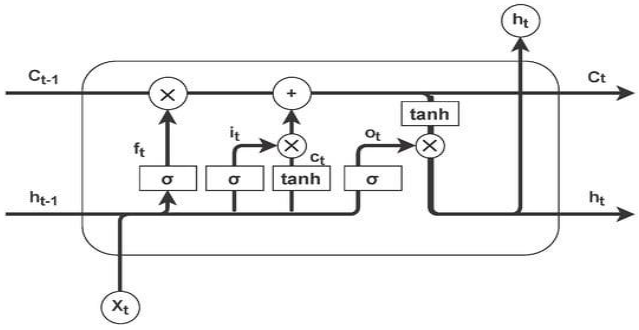



Hình 2. 2 Sơ đồ mô phỏng RNN

Nguồn: https://www.mdpi.com/2078-2489/15/9/517

**RNN** khá ưu việt nhưng nếu chuỗi đầu vào lớn hay rất lớn, mô hình sẽ gặp phải một vấn đề gọi là **vanishing gradient** (biến mất đạo hàm) .Từ đó để giải quyết vấn đề này, Long short term memory (LSTM) ra đời được phát triển từ RNN trước đó.

**Long Short-Term Memory (LSTM)** : Đây là một loại biến thể của RNN được thiết kế để giải quyết vấn đề **vanishing gradient** bằng cách sử dụng các cổng (gates) để kiểm soát thông tin mà mạng sẽ lưu trữ, quên hoặc đưa vào. LSTM giúp mạng có thể ghi nhớ thông tin trong một thời gian dài mà không bị mất mát hoặc quá tải.




Hình 2. 3 Sơ đồ mô phỏng LSTM

Nguồn: https://www.mdpi.com/2078-2489/15/9/517

- **Cấu trúc của LSTM:**

  LSTM có ba cổng chính:

  1. **Cổng quên (Forget Gate $f_t$):** Quyết định thông tin nào trong trạng thái ẩn trước đó sẽ bị "quên".
  2. **Cổng nhập (Input Gate $i_t$):** Quyết định thông tin nào sẽ được lưu trữ trong trạng thái ẩn.
  3. **Cổng xuất (Output Gate $o_t$):** Quyết định thông tin nào sẽ được sử dụng để tính toán đầu ra.

### *1.5.3 Các phương pháp học máy khác:*

#### 1.5.3.1 Linear Regression

Linear Regression là một phương pháp học máy thuộc nhóm Supervised Learning (học có giám sát), được sử dụng để giải quyết bài toán hồi quy. Mục tiêu chính của Linear Regression là dự đoán một giá trị đầu ra liên tục dựa trên một hoặc nhiều biến đầu vào.

**1. Ý tưởng chính**

Linear Regression dựa trên giả định rằng có một mối quan hệ tuyến tính giữa biến đầu vào (independent variables, thường ký hiệu là X) và biến đầu ra (dependent variable, ký hiệu là y).

Phương trình tuyến tính tổng quát có dạng:

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

- y: Biến mục tiêu (đầu ra cần dự đoán).
- $x_1, x_2, \dots, x_n$: Các biến đầu vào (feature).
- $w_1, w_2, \dots, w_n$: Hệ số (weights) của các biến đầu vào, thể hiện mức độ ảnh hưởng của từng biến đầu vào đến đầu ra.
- b: Hằng số tự do (bias hoặc intercept), giá trị đầu ra khi tất cả $x_i = 0$.

Trong trường hợp chỉ có một biến đầu vào (x), mô hình sẽ là:

$$y = w_x + b$$

**2. Cách huấn luyện mô hình**

Mục tiêu của mô hình Linear Regression là tìm các tham số $w_i$ và b sao cho sai số giữa giá trị dự đoán ($\hat{y}$) và giá trị thực tế (y) là nhỏ nhất.

a) Hàm dự đoán:

$$\hat{y} = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

b) Hàm mất mát (Loss function):

Hàm mất mát thường được sử dụng là **Mean Squared Error (MSE)**:

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

- $y_i$: Giá trị thực tế.
- $\hat{y}_i$: Giá trị dự đoán.
- m: Số lượng mẫu.

MSE đo lường sự khác biệt giữa giá trị thực tế và giá trị dự đoán. Mục tiêu là tối thiểu hóa MSE.

**1.5.3.2 SVM**

Support Vector Machine (SVM) là một thuật toán học máy mạnh mẽ được sử dụng trong các bài toán phân loại (classification) và hồi quy (regression). SVM hoạt động dựa trên ý tưởng tìm một siêu phẳng (hyperplane) tối ưu để phân tách dữ liệu hoặc dự đoán giá trị đầu ra.

**1. Ý tưởng chính**

- SVM tìm kiếm một siêu phẳng trong không gian đặc trưng để phân chia dữ liệu thuộc các lớp khác nhau. Mục tiêu là tối đa hóa khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất từ mỗi lớp, gọi là khoảng cách lề.
- Với bài toán phân loại, SVM tìm một siêu phẳng tối ưu để phân chia các lớp.
- Với bài toán hồi quy (SVM Regression), SVM tìm một siêu phẳng hoặc đường hồi quy trong không gian đặc trưng để dự đoán giá trị liên tục, với sai số nằm trong một khoảng cho trước ($\epsilon$).

**2. Siêu phẳng**

**Siêu phẳng (Hyperplane):**

Siêu phẳng là một đường hoặc mặt phẳng trong không gian nnn-chiều, được biểu diễn bởi phương trình:

$$W^T x + b = 0$$

- w: Vector trọng số, xác định hướng của siêu phẳng.
- b: Tham số bù (bias), xác định vị trí của siêu phẳng.

**3. Cách hoạt động**

**a) Bài toán tối ưu hóa (Optimization Problem):**

SVM giải bài toán tối ưu hóa sau:

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

với điều kiện ràng buộc:

$$y_i(w^T x_i + b) \geq 1, \forall i$$

- $y_i$: Nhãn của dữ liệu (+1 hoặc −1).
- $x_i$: Dữ liệu đầu vào.

**b) Soft Margin (Lề mềm):**

Trong thực tế, dữ liệu có thể không hoàn toàn phân tách tuyến tính. Soft Margin cho phép một số điểm dữ liệu nằm trong lề hoặc phân loại sai. Hàm mất mát trở thành:

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \epsilon_i$$

- $\epsilon_i$: Sai số của từng điểm.
- C: Tham số điều chỉnh giữa việc tối ưu hóa lề và giảm sai số.

**4. Huấn luyện mô hình**

Xây dựng bài toán tối ưu:

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \epsilon_i$$

Điều kiện ràng buộc:

$$y_i(w^T x_i + b) \geq 1 - \epsilon_i, \epsilon_i \geq 0$$

Mục tiêu là tìm w và b sao cho:

1. Tối đa hóa khoảng cách lề.
2. Giảm thiểu lỗi phân loại sai hoặc sai số trong hồi quy.

**Quy trình tối ưu:**

- Các điểm dữ liệu nằm gần siêu phẳng nhất được gọi là support vectors. Chỉ những điểm này ảnh hưởng đến siêu phẳng, giúp tăng tốc độ tính toán.
- Kernel trick được áp dụng khi dữ liệu không tuyến tính, ánh xạ dữ liệu sang không gian đặc trưng cao hơn để giải bài toán tuyến tính trong không gian đó.

#### 1.5.3.3 Decision Tree

Decision Tree là một thuật toán học máy thuộc nhóm Supervised Learning (học có giám sát), được sử dụng trong cả bài toán phân loại (classification) và hồi quy (regression). Thuật toán hoạt động bằng cách chia dữ liệu thành các tập con nhỏ hơn dựa trên các điều kiện được biểu diễn dưới dạng nút và nhánh trong một cấu trúc cây.

**1. Ý tưởng chính**

Decision Tree xây dựng mô hình dựa trên việc phân tách dữ liệu thành các nhóm dựa trên đặc trưng (features). Mỗi phân tách được thực hiện bằng cách chọn đặc trưng và ngưỡng giá trị sao cho độ không thuần (impurity) hoặc sai số giảm nhiều nhất.

- **Nút gốc (Root Node):** Điểm bắt đầu của cây, chứa toàn bộ dữ liệu.
- **Nút trung gian (Internal Node):** Biểu diễn điều kiện phân tách dựa trên giá trị của một đặc trưng.
- **Nhánh (Branch):** Biểu diễn kết quả của một điều kiện phân tách.
- **Nút lá (Leaf Node):** Biểu diễn kết quả cuối cùng, có thể là nhãn (cho phân loại) hoặc giá trị (cho hồi quy).

**2. Quy trình xây dựng cây quyết định**

- Chọn đặc trưng để phân tách:

Tìm đặc trưng và ngưỡng giá trị sao cho dữ liệu được chia thành các nhóm con với độ không thuần thấp nhất.

- Lặp lại phân tách:

Tiếp tục chia nhỏ dữ liệu ở các nút con đến khi đạt tiêu chí dừng (ví dụ: số lượng điểm dữ liệu nhỏ hơn ngưỡng hoặc đạt độ không thuần tối đa).

- Gán nhãn hoặc giá trị:

Tại các nút lá, gán nhãn (phân loại) hoặc giá trị trung bình (hồi quy) cho dữ liệu thuộc nút đó.

**3. Đo lường độ không thuần (Impurity Metrics)**

a) Phân loại:

1. *Gini Impurity (Gini Index):*

Đo lường xác suất một điểm dữ liệu được phân loại sai khi được chọn ngẫu nhiên.

$$\text{Gini} = 1 - \sum_{i=1}^{n} p_i^2$$

$p_i$: Xác suất một điểm thuộc lớp iii.

2. *Entropy:*

Đo lường độ hỗn loạn của dữ liệu.

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Giá trị entropy thấp cho thấy dữ liệu có tính đồng nhất cao.

3. *Information Gain:*

Đo lường mức độ giảm entropy sau khi phân tách dữ liệu.

$$\text{IG} = \text{Entropy}(\text{parent}) - \sum_{k} \frac{N^k}{N} \text{Entropy}(\text{Child}_k)$$

b) Hồi quy:

**1. Mean Squared Error (MSE):**

Đo lường sai số trung bình bình phương giữa giá trị thực và giá trị dự đoán.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

2. **Reduction in Variance:**

Đo lường mức độ giảm phương sai sau khi phân tách.

## 1.6 Thực nghiệm, đánh giá và so sánh:

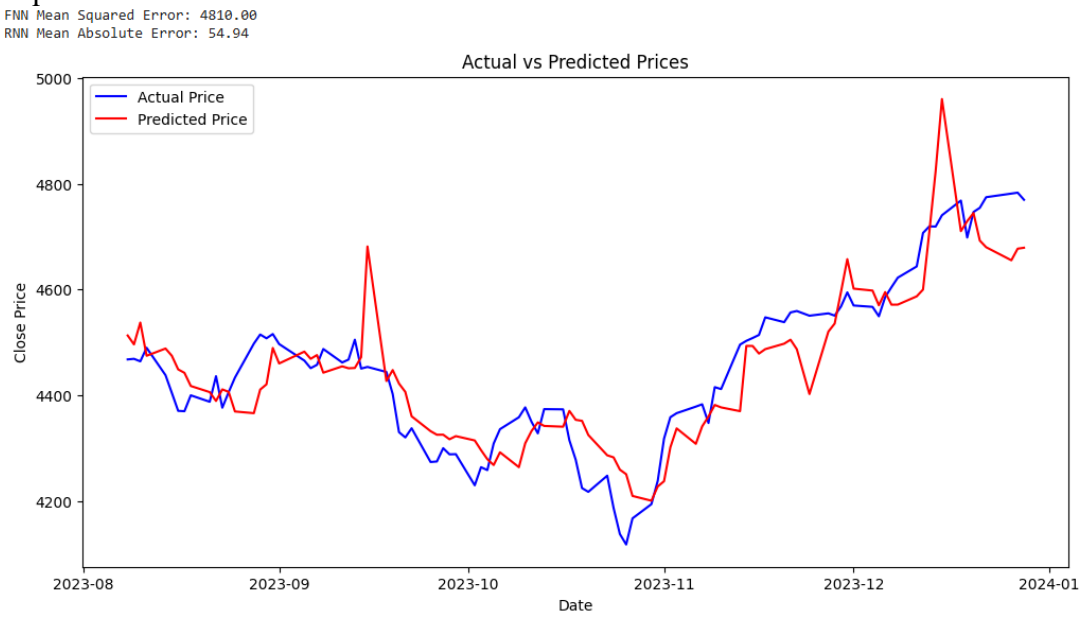
### *1.6.1 Feedforward Neural Network*

Code:

```
# Build the Feedforward Neural Network
print("Training Feedforward Neural Network...")

fnn_model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1)
])

fnn_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
fnn_history = fnn_model.fit(
    X_train, y_train,
    epochs=50,
    validation_data=(X_test, y_test),
    batch_size=32,
    verbose=1
)
```

Hình 2. 4 Phương pháp Feedforward Neural Network

Kết quả:




Hình 2. 5 Kết quả phương pháp Feedforward Neural Network

Đánh giá phương pháp:

**MSE (4810.00):** Giá trị này khá cao, cho thấy sai số bình phương trung bình lớn, đặc biệt nếu giá trị thực của giá cổ phiếu nằm trong khoảng nhỏ.

**MAE (54.94):** Trung bình sai số tuyệt đối là khoảng 54.94 đơn vị. Nếu giá cổ phiếu dao động trong khoảng lớn, sai số này có thể chấp nhận được. Tuy nhiên, nếu dao động nhỏ, kết quả chưa thực sự tốt.

**Ưu điểm:** Mô hình bám sát tốt xu hướng chung của giá cổ phiếu trong các giai đoạn ổn định hoặc biến động nhẹ.

**Hạn chế:** Mô hình gặp khó khăn trong việc phản ứng với các biến động mạnh, đặc biệt tại các đỉnh và đáy, và thường có độ trễ trong dự đoán.

**Hướng cải thiện:** Sử dụng các mô hình phù hợp hơn với dữ liệu chuỗi thời gian (LSTM, GRU) hoặc bổ sung các đặc trưng dữ liệu để nâng cao độ chính xác.

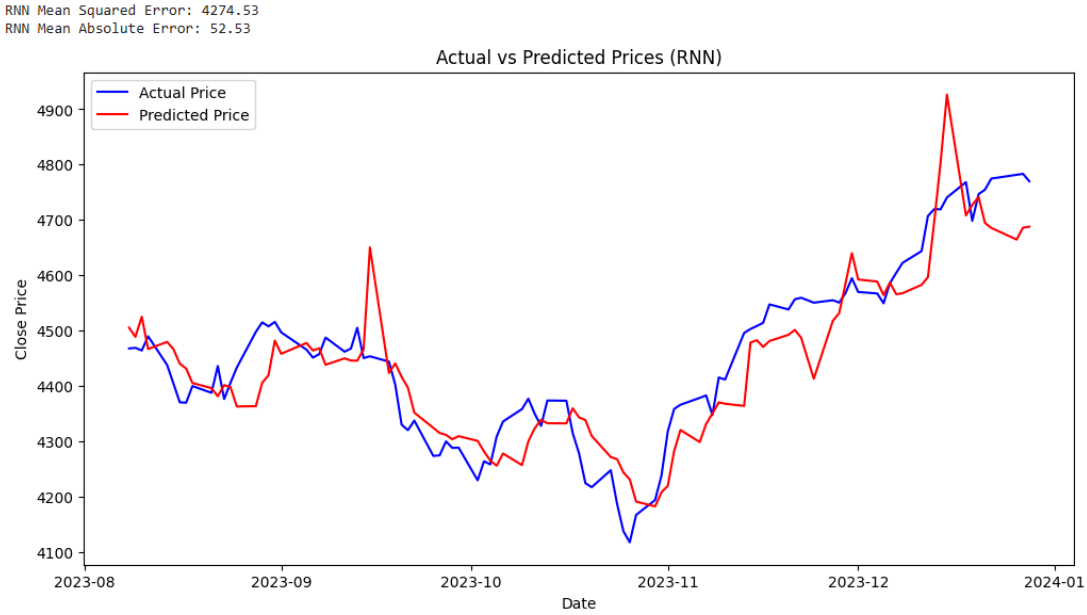
### *1.6.2 Recurrent Neural Network (RNN)*

Code:

```
# Build the RNN Model
print("Training RNN...")

rnn_model = Sequential([
    SimpleRNN(64, activation='relu', input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    Dense(32, activation='relu'),
    Dense(1)  # Linear activation for regression
])

rnn_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the Model
rnn_history = rnn_model.fit(
    X_train_rnn, y_train_rnn,
    epochs=50,
    validation_data=(X_test_rnn, y_test_rnn),
    batch_size=32,
    verbose=1
)
```

Hình 2. 6 Phương pháp Recurrent Neural Network (RNN)

Kết quả:




Hình 2. 7 Kết quả phương pháp Recurrent Neural Network (RNN)

Đánh giá phương pháp:

Mô hình RNN đã bắt kịp được xu hướng chung của giá cổ phiếu, đặc biệt là các giai đoạn tăng hoặc giảm mạnh. Điều này cho thấy khả năng của mô hình trong việc học các mẫu tuần hoàn hoặc chuỗi thời gian.

- **MSE (Mean Squared Error) là 4274.53:**

Đây là giá trị lỗi bình phương trung bình. Với bài toán chuỗi thời gian dự đoán giá, phụ thuộc vào quy mô của giá trị thực tế. Nếu giá đóng cửa thường dao động trong khoảng hàng trăm hoặc hàng nghìn, thì mức MSE này là khá lớn, nghĩa là mô hình đang dự đoán không đủ chính xác.

- **MAE (Mean Absolute Error) là 52.53:**

MAE đo lường độ lệch trung bình tuyệt đối giữa giá trị thực tế và dự đoán. Con số 52.53 cho thấy rằng, trung bình, dự đoán của mô hình lệch 52.53 đơn vị so với giá trị thực tế. Nếu giá trị thực tế dao động từ 500 đến 1000, mức lỗi này khá đáng kể.

### *1.6.3 Linear Regression, SVM, Decision Tree, Random Forest*

Code:

```
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

# SVM
svm_model = SVR(kernel='linear')
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Decision Tree
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

# Random Forest
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
```

Hình 2. 8 Phương pháp Linear Regression, SVM, Decision Tree, Random Forest

Kết quả:

```
Linear Regression Mean Squared Error: 1164.19
Linear Regression Mean Absolute Error: 26.59
SVM Mean Squared Error: 3348.75
SVM Mean Absolute Error: 48.13
Decision Tree Mean Squared Error: 2707.08
Decision Tree Mean Absolute Error: 39.68
Random Forest Mean Squared Error: 1428.86
Random Forest Mean Absolute Error: 29.93
```

Hình 2. 9 Kết quả Linear Regression, SVM, Decision Tree, Random Forest

**Đánh giá các phương pháp:**

**Linear Regression (LR)**

- MSE: 1164.19
- MAE: 26.59

- Linear Regression có hiệu suất tốt nhất trong các mô hình được đánh giá, với cả MSE và MAE đều thấp nhất. Điều này cho thấy LR phù hợp với dữ liệu và có thể là một mô hình đơn giản nhưng hiệu quả trong trường hợp này.

**Support Vector Machine (SVM)**

- MSE: 3348.75
- MAE: 48.13
- Hiệu suất của SVM thấp hơn nhiều so với các mô hình khác. Điều này có thể do SVM không phù hợp với dữ liệu này hoặc cần tối ưu hóa thêm các siêu tham số như kernel, C, hoặc epsilon.

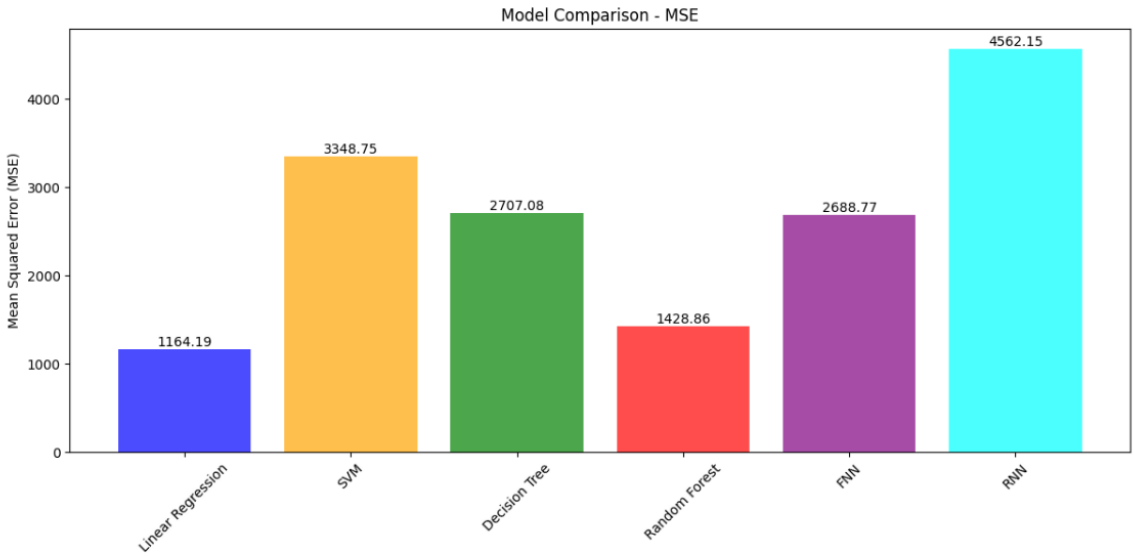
**Decision Tree (DT)**

- MSE: 2707.08
- MAE: 39.68
- Decision Tree có hiệu suất tốt hơn SVM nhưng vẫn kém hơn Linear Regression và Random Forest. Có khả năng mô hình bị overfitting hoặc không thể khái quát hóa tốt trên tập kiểm tra.

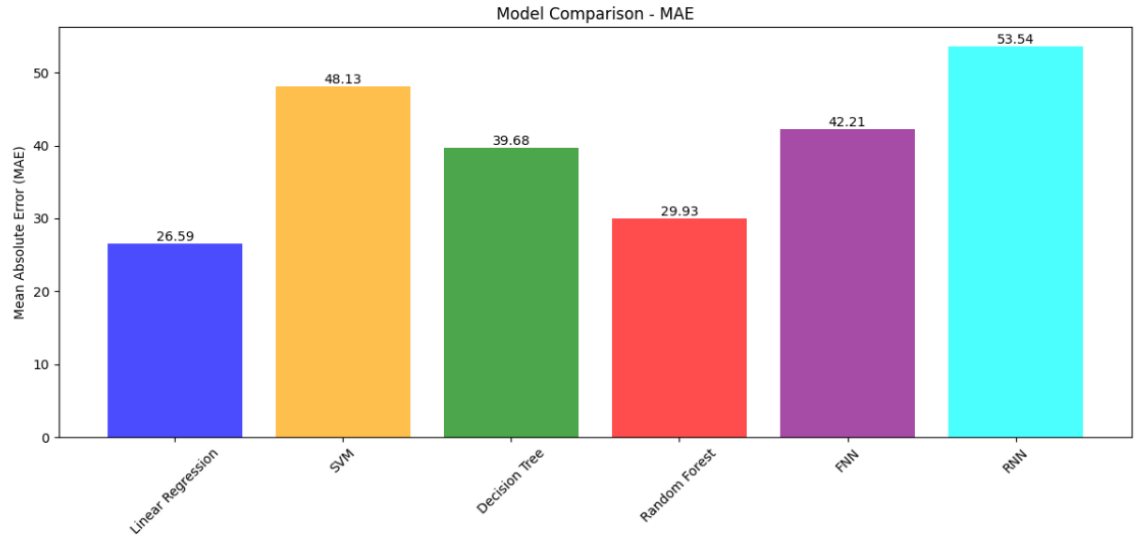
**Random Forest (RF)**

- MSE: 1428.86
- MAE: 29.93
- Random Forest hoạt động khá tốt, chỉ kém hơn Linear Regression một chút. Đây là một mô hình mạnh mẽ, đặc biệt trong các bài toán phức tạp hơn.

**Biểu đồ so sánh các phương pháp sử dụng**




Hình 2. 10 Biểu đồ so sánh  MSE các phương pháp




Hình 2. 11 Biểu đồ so sánh MAE các phương pháp

Nhận xét:

· Linear Regression và Random Forest là hai mô hình có hiệu suất tốt nhất.

· Các mô hình SVM, Decision Tree, PNN và RNN đều thể hiện hiệu suất kém hơn, có thể do dữ liệu không phù hợp hoặc các siêu tham số chưa được tối ưu.

· Để chọn mô hình cuối cùng, nên ưu tiên Linear Regression vì đơn giản và hiệu quả, hoặc Random Forest nếu cần tính linh hoạt và khả năng xử lý dữ liệu phức tạp hơn.

## 1.7 Áp dụng các phương pháp giải quyết overfitting:

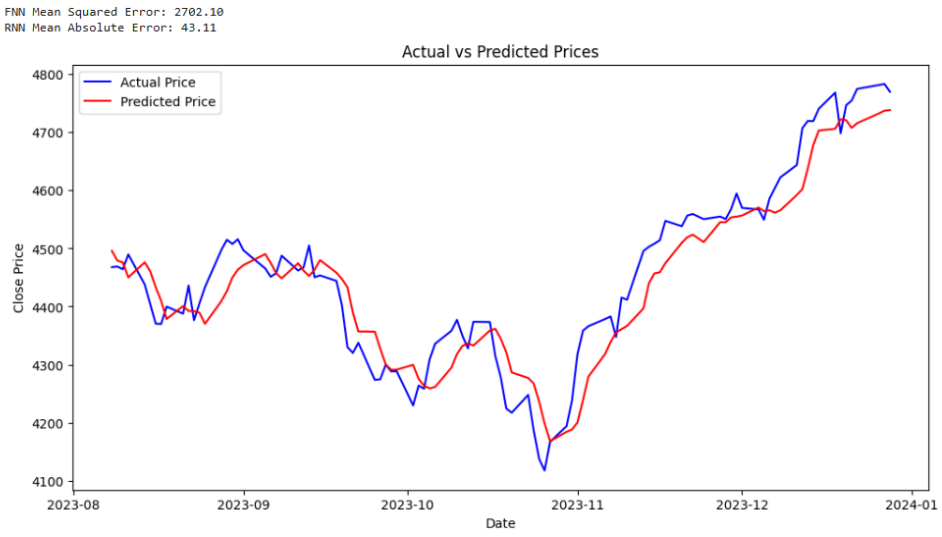
### *1.7.1 Feedforward Neural Network*

Code:

```
fnn_model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1)
])

fnn_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
early_stopping = EarlyStopping(patience=10, restore_best_weights=True)
fnn_history = fnn_model.fit(
    X_train, y_train,
    epochs=50,
    validation_data=(X_test, y_test),
    batch_size=32,
    verbose=1,
    callbacks=[early_stopping]
)
```

Hình 2. 12 Áp dụng Overfitting Feedforward Neural Network

Kết quả:




Hình 2. 13 Kết quả áp dụng giải quyết Overfitting Feedforward Neural Network

**Phương pháp giải quyết overfitting:**

- Dropout ngẫu nhiên(deactivate) 30% số lượng nơ-ron trong mỗi lần huấn luyện, làm giảm khả năng phụ thuộc vào một số nơ-ron cụ thể, từ đó cải thiện khả năng tổng quát hóa của mô hình.
- Nếu hiệu suất trên tập dữ liệu kiểm tra (validation set) không được cải thiện trong 10 epoch liên tiếp, việc huấn luyện sẽ dừng sớm.
- Sử dụng batch size nhỏ hơn giúp cập nhật trọng số thường xuyên hơn, làm giảm nguy cơ mô hình ghi nhớ quá mức dữ liệu huấn luyện.

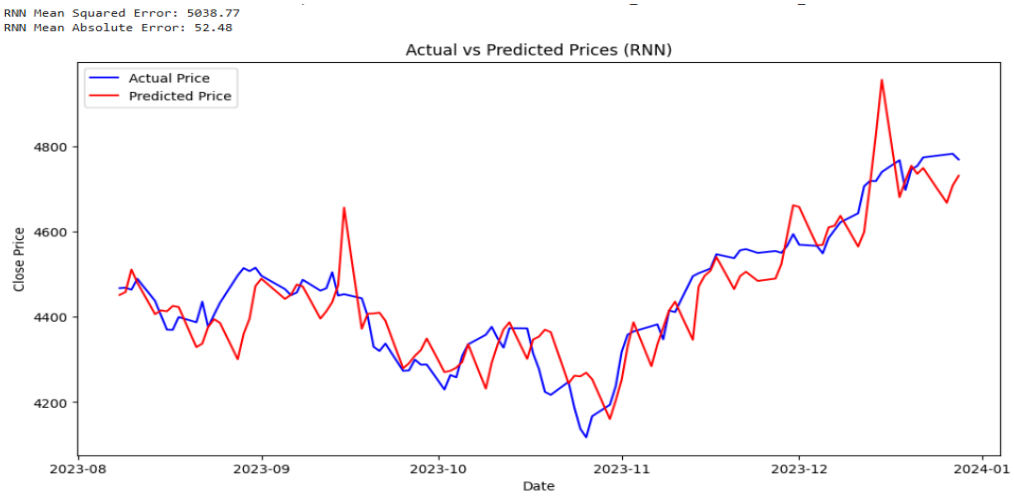
### *1.7.2 Recurrent Neural Network (RNN)*

Code:

```
# Build the RNN Model
print("Training RNN...")

rnn_model = Sequential([
    SimpleRNN(128, activation='relu', kernel_regularizer=l2(0.001), input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    Dropout(0.3),
    Dense(32, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(1)
])

rnn_model.compile(optimizer='adam', loss='mse', metrics=['mae'])

#Train the Model
early_stopping = EarlyStopping(patience=5, restore_best_weights=True)
rnn_history = rnn_model.fit(
    X_train_rnn, y_train_rnn,
    epochs=50,
    validation_data=(X_test_rnn, y_test_rnn),
    batch_size=32,
    verbose=1,
    callbacks=[early_stopping]
)
```

Hình 2. 14 Áp dụng Overfitting Recurrent Neural Network (RNN)

Kết quả:




Hình 2. 15 Kết quả áp dụng Overfitting Recurrent Neural Network (RNN)

**Phương pháp giải quyết overfitting:**

- L2 regularization được sử dụng trong các lớp
- Điều này thêm một hình phạt vào hàm mất mát dựa trên độ lớn của trọng số, khuyến khích các trọng số nhỏ hơn và giảm nguy cơ overfitting.
- Dropout được thêm vào sau các lớp chính.
- Early stopping ngăn chặn mô hình huấn luyện quá lâu, dẫn đến khả năng ghi nhớ (memorization) thay vì học (generalization)
- Dữ liệu đầu vào được chuẩn hóa với MinMaxScaler

Các phương pháp này phối hợp để giảm overfitting trong mô hình RNN bằng cách điều chỉnh trọng số, giảm phụ thuộc vào các nơ-ron cụ thể, dừng huấn luyện khi cần thiết, và đảm bảo dữ liệu đầu vào được chuẩn hóa.

### *1.7.3 Linear Regression, SVM, Decision Tree, Random Forest*

Code:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Ridge Regression
lr_model = Ridge(alpha=1.0)
lr_model.fit(X_train_scaled, y_train)
lr_pred = lr_model.predict(X_test_scaled)

# SVM
svm_model = SVR(kernel='linear', C=0.1)
svm_model.fit(X_train_scaled, y_train)
svm_pred = svm_model.predict(X_test_scaled)

# Decision Tree
dt_model = DecisionTreeRegressor(max_depth=5, min_samples_split=10, min_samples_leaf=5)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

# Random Forest
rf_model = RandomForestRegressor(max_depth=10, n_estimators=50, max_features='sqrt', random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
```

Hình 2. 16 Áp dụng Overfitting Linear Regression, SVM, Decision Tree, Random Forest

**Phương pháp giải quyết overfitting:**

- **Ridge Regression** sử dụng L2 regularization, thêm một hình phạt tỷ lệ thuận với bình phương độ lớn của các hệ số (weights). Điều này ngăn các hệ số trở nên quá lớn, giảm nguy cơ overfitting.
- **Support Vector Machine (SVM)**: Tham số C kiểm soát mức độ cho phép các lỗi (slack variables) và phạt các điểm dữ liệu nằm ngoài biên. Giá trị nhỏ của C khuyến khích biên rộng hơn, giúp giảm overfitting.
- **DecisionTree:**

Depth Limitation: Giới hạn độ sâu của cây quyết định, ngăn cây phân tách quá sâu, điều này làm giảm khả năng mô hình ghi nhớ dữ liệu huấn luyện.

Minimum Samples for Split and Leaf Nodes:

· min_samples_split=10: Yêu cầu ít nhất 10 mẫu trước khi phân chia một nút.

· min_samples_leaf=5: Mỗi nút lá phải chứa ít nhất 5 mẫu. Điều này đảm bảo cây không quá phức tạp.

- **Random Forest:**

· Giới hạn độ sâu (max_depth=10): Ngăn cây trong rừng phát triển quá sâu, giảm khả năng ghi nhớ.

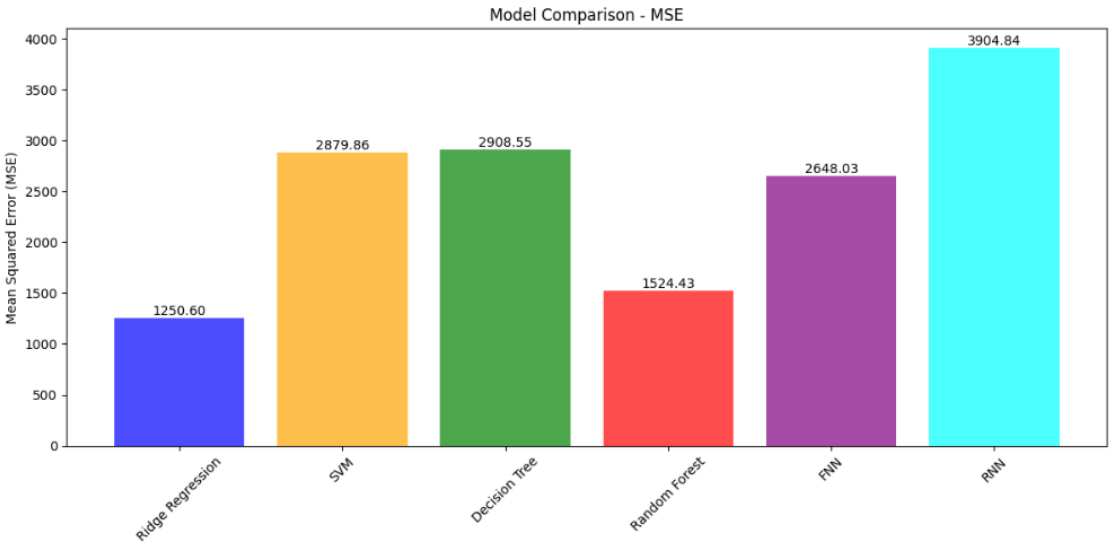
· Số lượng cây (n_estimators=50): Kết hợp dự đoán từ 50 cây quyết định, giảm overfitting bằng cách trung bình hóa kết quả.

· Giới hạn số đặc trưng (max_features='sqrt'): Mỗi cây chỉ sử dụng căn bậc hai số đặc trưng, tạo đa dạng giữa các cây và giảm phụ thuộc vào dữ liệu huấn luyện.

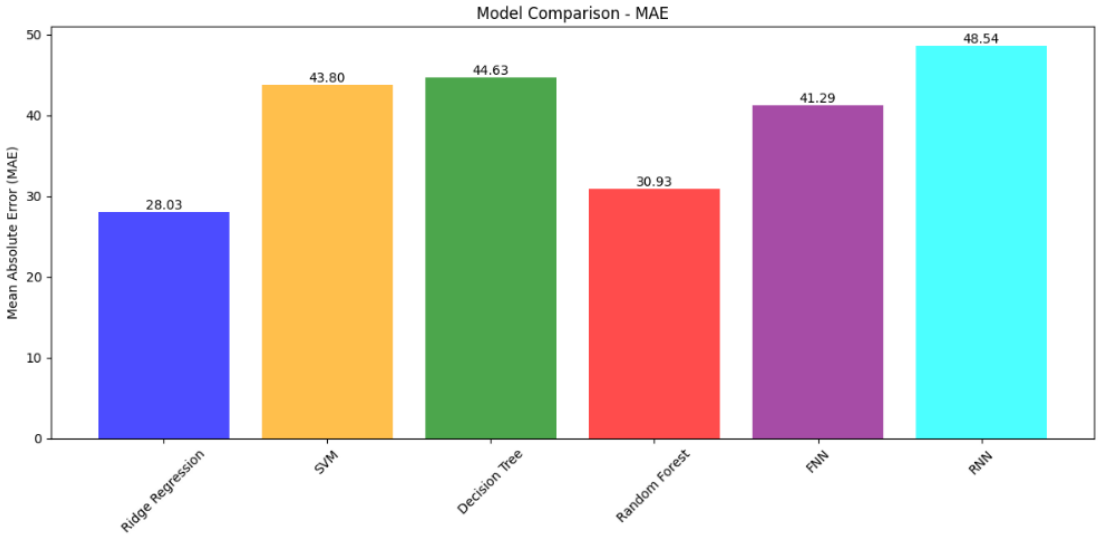
**Kết luận**

- Các mô hình như Ridge và SVM dựa vào regularization.
- Các thuật toán cây (Decision Tree, Random Forest) sử dụng giới hạn độ sâu, số lượng mẫu tối thiểu, và tính đa dạng của rừng cây để tránh overfitting.
- Chuẩn hóa dữ liệu đầu vào là phương pháp chung giúp cải thiện khả năng tổng quát hóa của tất cả các mô hình.

**Biểu đồ so sánh sau khi áp dụng giải quyết Overfitting:**




Hình 2. 17 Biểu đồ MSE áp dụng Overfitting

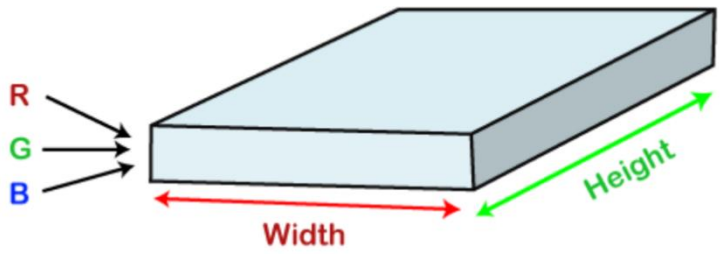



Hình 2. 18 Biểu đồ MAE áp dụng Overfitting

# CHƯƠNG 3 – STUDY AND PRESENT A DEEP MACHINE LEARNING

## 1.1 Giới thiệu về CNN

- Convolutional Neural Network (CNN) là một kiến trúc mạng nơ-ron sâu được thiết kế để xử lý dữ liệu dạng lưới (như hình ảnh hoặc dữ liệu chuỗi), đây là 1 trong những mô hình nổi tiếng dùng để nhận dạng và phân loại hình ảnh.
- CNN nổi bật nhờ khả năng tự động trích xuất và học các đặc trưng từ dữ liệu đầu vào, đặc biệt phù hợp cho các tác vụ liên quan đến thị giác máy tính (computer vision).
- Dữ liệu đầu vào thường là các ma trận dạng lưới, chẳng hạn như ảnh h × w × c (chiều cao h, chiều rộng w, số kênh màu c).

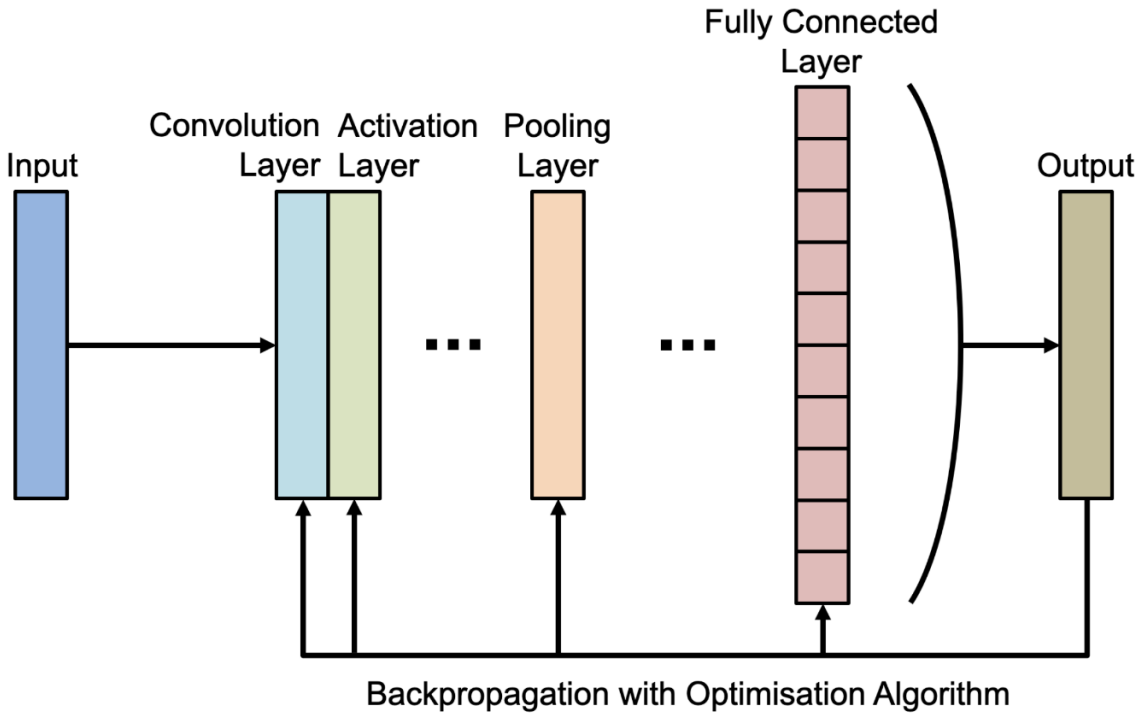



Hình 3. 1 Mô phỏng dữ liệu đầu vào

Nguồn: https://logicmojo.com/convolutional-neural-network

## 1.2 Cấu trúc CNN

Mô hình CNN được xây dựng dựa theo cấu trúc gồm 3 lớp cơ bản, bao gồm:




Hình 3. 2 Mô hình CNN

Nguồn: https://www.mdpi.com/2076-3417/11/2/744

**Convolutional Layers (Lớp tích chập):**

- Đây là lớp nền tảng của CNN, giúp trích xuất các đặc trưng quan trọng, cơ bản như cạnh, góc, và kết cấu.
- Bộ lọc (filter) được sử dụng để quét qua hình ảnh, giúp học các mẫu cục bộ.
- Đầu ra (feature map): Kết quả từ tích chập là một ma trận mới thể hiện đặc trưng cục bộ, như cạnh hoặc đường viền.
- Kernel được dịch chuyển qua toàn bộ ảnh và thực hiện phép nhân tích chập (convolution) giữa kernel và vùng dữ liệu tương ứng, sau khi dịch chuyển, áp dụng công thức như sau:

$$O(i,j) = \sum_{m}\sum_{n}(I[i+m][j+n].K[m][n])$$

Với I: ma trận đầu vào

K:kernel

m,n: Chỉ số dịch chuyển trong kernel.

O(i,j): Giá trị tại vị trí (i, j) trên feature map.

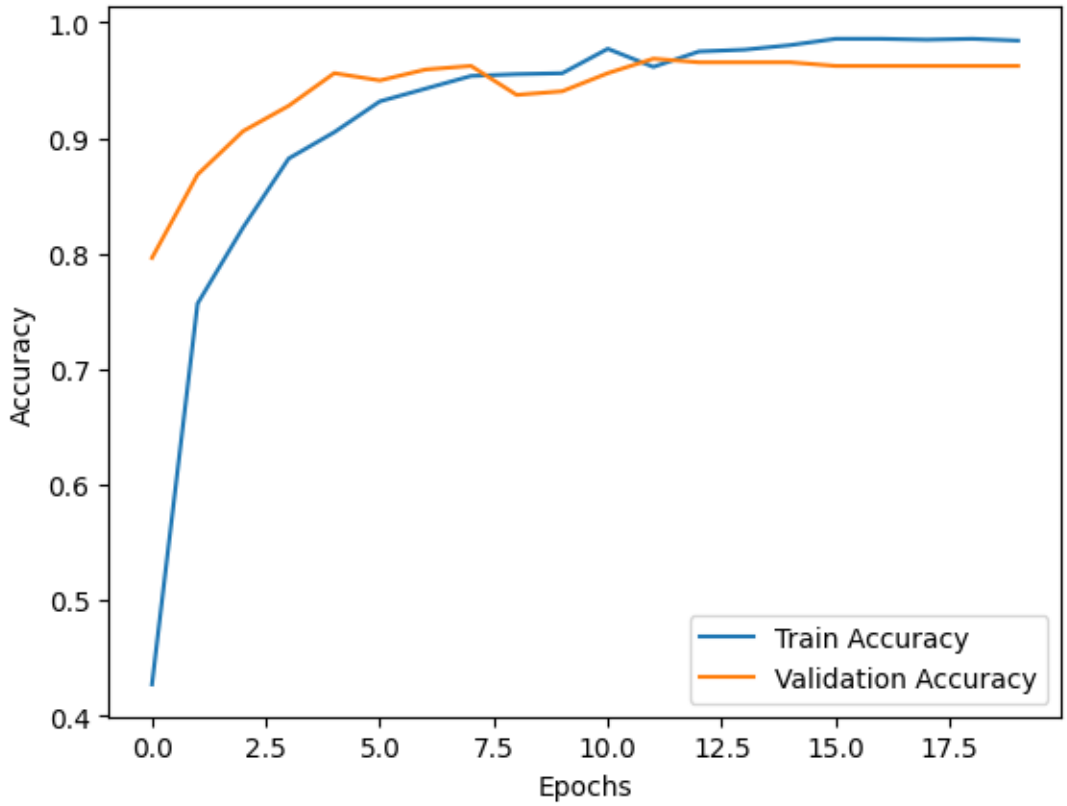
**Pooling Layers (Lớp gộp):**

có nhiệm vụ giảm kích thước của các đặc trưng đầu ra từ lớp tích chập, giúp giảm độ phức tạp tính toán và hạn chế overfitting, đồng thời giữ lại các thông tin quan trọng.Có ba loại gộp chính thường được sử dụng trong CNN:

1. **Max Pooling (Gộp cực đại):** Lấy giá trị lớn nhất trong mỗi vùng nhỏ của feature map. Phương pháp này giúp giữ lại các đặc trưng nổi bật như cạnh, góc hoặc đường viền, từ đó làm nổi bật các thông tin quan trọng. Loại gộp này được sử dụng nhiều nhất nhờ khả năng bảo toàn các đặc trưng nổi bật của dữ liệu.
2. **Average Pooling (Gộp trung bình):** Tính giá trị trung bình của tất cả các phần tử trong vùng nhỏ. Phương pháp này giúp giảm nhiễu và duy trì thông tin tổng quát của các đặc trưng.
3. **Sum Pooling (Gộp tổng):** Tính tổng các phần tử trong vùng nhỏ. Tuy nhiên, phương pháp này ít được sử dụng trong thực tế vì khó bảo toàn các đặc trưng quan trọng so với Max Pooling và Average Pooling.

**Fully Connected Layers (Lớp kết nối đầy đủ):** Sau khi dữ liệu đã được giảm kích thước và trích xuất đặc trưng, lớp kết nối đầy đủ sẽ đưa ra kết quả phân loại hoặc nhận diện dựa trên đặc trưng đã học.

**Kết quả thực nghiệm**




Hình 3. 3 Kết quả thực nghiệm

Test Accuracy: 96.24%

**Độ chính xác**

- Train Accuracy (độ chính xác trên tập huấn luyện): **~99%**.
- Validation Accuracy (độ chính xác trên tập validation): **~95-96%**.
- Test Accuracy (độ chính xác trên tập kiểm tra): **96.24%.**

**Đồ thị huấn luyện**

Nhìn vào đồ thị, ta có thể thấy được sự thay đổi của Train Accuracy và Validation Accuracy theo số epoch và có thể nhận xét như sau:

- Train Accuracy tăng khá nhanh từ epoch 0-5 và đạt hội tụ ở khoảng epoch 10-15.

- Validation Accuracy thì ổn định và bám sát đường huấn luyện, không có hiện tượng chênh lệch quá lớn.

**Phân tích kết quả thu được**

1. **Hội tụ của mô hình:**
   - Độ chính xác trên cả tập huấn luyện và tập kiểm tra đều đạt giá trị cao (~96%).
   - Mô hình hội tụ ổn định sau khoảng 10 epochs.
2. **Độ khái quát hóa của mô hình:**
   - Validation Accuracy và Test Accuracy khá cao, cho thấy mô hình hoạt động tốt trên dữ liệu chưa thấy.
   - Khoảng cách nhỏ giữa Train Accuracy và Validation Accuracy chứng tỏ mô hình không bị overfitting.
3. **Khả năng học đặc trưng:**

CNN đã trích xuất thành công các đặc trưng quan trọng từ dữ liệu và thực hiện phân loại chính xác.

**Nhận xét và đánh giá**

- **Ưu điểm:**
  - Mô hình CNN đạt hiệu suất cao (Test Accuracy 96.24%).
  - Quá trình huấn luyện ổn định và nhanh chóng.
  - Mô hình khái quát tốt và không xảy ra overfitting.
- **Hạn chế:**

Với các bộ dữ liệu lớn hơn và phức tạp hơn, mô hình có thể cần thêm các lớp hoặc kỹ thuật điều chỉnh.

**Kết luận**

Mô hình CNN đã thể hiện khả năng xử lý và phân loại dữ liệu một cách hiệu quả. Với độ chính xác trên tập kiểm tra đạt 96.24% trên thực nghiệm, cho thấy mô hình đã học và trích xuất được các đặc trưng quan trọng từ dữ liệu đầu vào.

Kết quả này cũng phản ánh tính ổn định của mô hình trong suốt quá trình huấn luyện, khi không có hiện tượng quá khớp (overfitting) và độ chính xác trên tập validation cũng bám sát độ chính xác huấn luyện.

Nhờ kiến trúc CNN, mô hình có thể tự động học các đặc trưng cục bộ trong dữ liệu và tổng hợp chúng để đưa ra kết quả phân loại chính xác, minh chứng rõ ràng cho hiệu suất và khả năng khái quát hóa của CNN.

# TÀI LIỆU THAM KHẢO

## Tiếng Việt


[1] T. Vu, "Bài 7: Gradient Descent (phần 1/2)," *Tiep Vu's Blog*, Jan. 12, 2017. https://machinelearningcoban.com/2017/01/12/gradientdescent/

[2] T. Vu, "Bài 8: Gradient Descent (phần 2/2)," *Tiep Vu's Blog*, Jan. 16, 2017. https://machinelearningcoban.com/2017/01/16/gradientdescent2/

[3] T. T. Trực, "Optimizer- Hiểu sâu về các thuật toán tối ưu ( GD,SGD,Adam,..)," *Viblo*, Dec. 18, 2024. https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8

[4] T. Q. T. B, "Một số hàm kích hoạt trong các mô hình Deep learning, tại sao chúng lại quan trọng đến vậy ? - Part 1: Hàm ...," *Viblo*, Dec. 18, 2024. https://viblo.asia/p/mot-so-ham-kich-hoat-trong-cac-mo-hinh-deep-learning-tai-sao-chung-lai-quan-trong-den-vay-part-1-ham-sigmoid-bWrZn4Rv5xw

[5] "Hàm mất mát (loss function) | Machine Learning: mì, súp và công thức nấu." https://khanh-personal.gitbook.io/ml-book-vn/chapter1/ham-mat-mat

[6] P. Ngoc, "Thuật toán tối ưu adam," *Viblo*, Dec. 18, 2024. https://viblo.asia/p/thuat-toan-toi-uu-adam-aWj53k8Q56m


Tiếng Anh


[7] N. Malingan, "Adaptive Methods of Gradient Descent in Deep Learning - Scaler topics," *Scaler Topics*, Mar. 23, 2023. https://www.scaler.com/topics/deep-learning/adagrad/

[8] A. C. H. Chen, "Exploring the optimized value of each hyperparameter in various gradient descent algorithms," *arXiv (Cornell University)*, Jan. 2022, doi: 10.48550/arxiv.2212.12279.

[9] GeeksforGeeks, "What is Adam Optimizer?," *GeeksforGeeks*, Mar. 20, 2024. https://www.geeksforgeeks.org/adam-optimizer/

[10] Recurrent Neural Networks,"Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications",MDIP, 21 July 2024 https://www.mdpi.com/2078-2489/15/9/517

**[11]** FeedForward Neural Networks," Mukul Rathi", 2024 https://mukulrathi.com/demystifying-deep-learning/feed-forward-neural-network/