# CHƯƠNG 1. CÂU 1

Tìm hiểu, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy

## 1.1 Optimizer

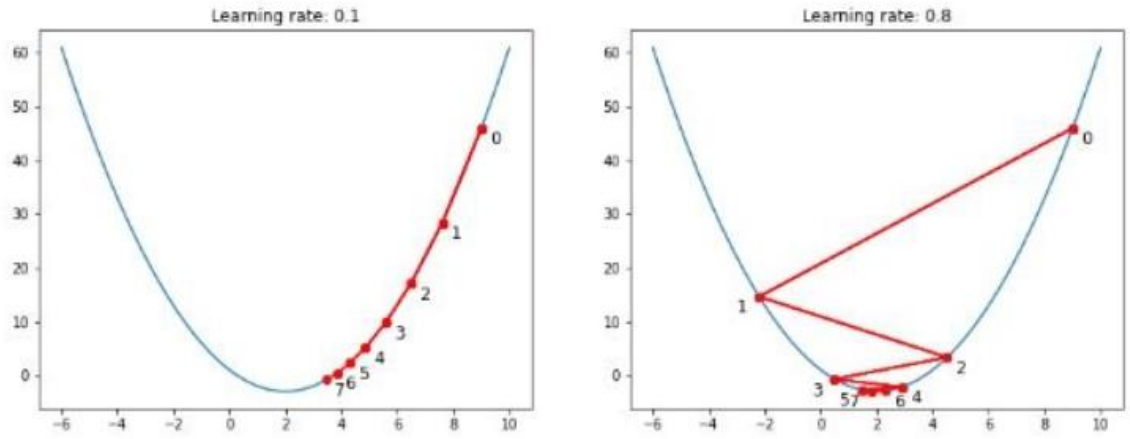
Trong quá trình huấn luyện mô hình học máy, optimizer là một phần quan trọng của quá trình tối ưu hóa. Nó là một thuật toán được sử dụng để điều chỉnh các tham số của mô hình (như trọng số và bias) dựa trên dữ liệu huấn luyện để mục tiêu làm giảm thiểu hàm mất mát (loss function).

Khi một mô hình học máy được huấn luyện, nó cố gắng điều chỉnh các tham số của mình để dự đoán đầu ra chính xác hơn. Optimizer là công cụ giúp mô hình thực hiện việc này bằng cách sử dụng gradient của hàm mất mát đối với các tham số để điều chỉnh chúng theo hướng giảm thiểu mất mát.

Các loại optimizer khác nhau như Gradient Descent, Stochastic Gradient Descent (SGD), Adam, RMSprop, và nhiều thuật toán tối ưu hóa khác có cách tiếp cận khác nhau để cập nhật các tham số của mô hình. Mục tiêu chung của tất cả các optimizer là học mô hình sao cho nó có thể dự đoán chính xác hơn trên dữ liệu mới mà nó chưa từng thấy.

## 1.2 Gradient descent (GB)

Đây là một trong những phương pháp tối ưu hoá cơ bản và phổ biến trong học máy. Được sử dụng để cập nhật các tham số của mô hình dựa trên gradient của hàm mất mát theo các tham số đó. Mục tiêu là di chuyển từ điểm khởi đầu trên không gian tham số đến điểm tối ưu, thường là điểm tối thiểu của hàm mất mát




Hình 1.1: Gradient Descent Optimizer

### *1.2.1 Hoạt động*

Tính toán Gradient: gradient của hàm mất mát được tính bằng cách lấy đạo hàm của hàm mất mát theo từng tham số trong mô hình

$$\nabla J(\theta_i) = \frac{\partial J}{\partial \theta_i}$$

Trong đó:

- $\nabla J(\theta_i)$ là gradient của hàm mất mát J theo trọng số $\theta_i$
- $\frac{\partial J}{\partial \theta_i}$ là đạo hàm riêng của hàm mất mát J theo trọng số $\theta_i$

Cập nhật tham số: thực hiện cập nhất các tham số của mô hình bằng cách di chuyển ngược chiều của gradient với một tỉ lệ học (learning rate). Bằng cách này, các tham số được điều chỉnh để giảm độ lớn của gradient, mô hình tiến gần đến điểm tối ưu

$$\theta_{t+1} = \theta_t - \eta \nabla \mathrm{J}(\theta_t)$$

Trong đó:

- $\theta_{t+1}$ là trọng số mới sau bước thời gian t + 1
- $\theta_t$ là trọng số tại bước thời gian t
- $\eta$ là learning rate, quyết định tốc độ học của thuật toán

### *1.2.2 Ưu điểm*

Đơn giản, dễ triển khai

Hiệu suất tốt

Phổi biến và linh hoạt

### *1.2.3 Nhược điểm:*

Dễ rơi vào điểm cực tiểu cục bộ

Khả năng hội tụ chậm

Phụ thuộc vào learning rate

## 1.3 Stochastic Gradient descent (SGB)

Đây là thuật toán tối ưu hoá cơ bản theo họ gradient, là 1 biến thể của Gradient Descent. SGD tính toán gradient và cập nhật tham số dựa trên từng điểm dữ liệu trong tập huấn luyện. Điều này làm giảm thiểu độ phức tạp tính toán, nhưng có thể dẫn đến các bước di chuyển không ổn định.

### *1.3.1 Hoạt động*

Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần. SGD sẽ làm giảm đi tốc độ của 1 epoch. Tuy nhiên nhìn theo 1 hướng khác, SGD sẽ hội tụ rất nhanh chỉ sau vài epoch. Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu.

Chọn mẫu ngẫu nhiên: Đầu tiên, một điểm dữ liệu hoặc một mini-batch nhỏ được chọn ngẫu nhiên từ tập dữ liệu huấn luyện.

Tính gradient: Gradient của hàm mất mát được tính dựa trên điểm dữ liệu hoặc mini-batch vừa chọn.

Cập nhật trọng số: Trọng số của mô hình được cập nhật bằng cách di chuyển theo hướng ngược với gradient với một bước theo learning rate (tốc độ học).

$$\theta_{t+1} = \theta_t - \eta \nabla \mathrm{J}(\theta_{t,i})$$

Trong đó:

- $\theta_{t+1}$ là trọng số mới sau bước thời gian t + 1

- $\theta_t$ là trọng số tại bước thời gian t
- $\eta$ là learning rate, quyết định tốc độ học của thuật toán
- $\nabla J(\theta_{t,i})$ là gradient của hàm mất mát J theo trọng số $\theta_{t,i}$ dựa trên một mẫu ngẫu nhiên tại thời điểm t

Lặp lại quá trình: Quá trình trên được lặp lại qua nhiều vòng lặp (epochs) hoặc qua toàn bộ tập dữ liệu huấn luyện.

### *1.3.2 Ưu điểm*

Tính linh hoạt: Đặc điểm ngẫu nhiên giúp tránh khỏi các điểm cực tiểu cục bộ và tạo sự đa dạng trong quá trình tối ưu hóa.

Nhanh chóng và hiệu quả với dữ liệu lớn: Do chỉ cần tính toán gradient trên một điểm hoặc một mini-batch nhỏ.

Yêu cầu ít bộ nhớ

### *1.3.3 Nhược điểm:*

Phương sai cao, dễ dẫn đến overfitting

Yêu cầu mô hình thay đổi liên tục (online learning)

## 1.4 Adagrad

AdaGrad là một phương pháp tối ưu hóa trong học máy, đặc biệt được sử dụng trong quá trình huấn luyện mô hình để cập nhật trọng số theo gradient của hàm mất mát. Phương pháp này có khả năng điều chỉnh tỷ lệ học (learning rate) cho từng tham số của mô hình dựa trên lịch sử của gradient đã tính toán cho tham số đó. Không giống như các thuật toán trước đó thì learning rate hầu như giống nhau trong quá trình training (learning rate là hằng số), Adagrad coi learning rate là 1 tham số. Tức là Adagrad sẽ cho learning rate biến thiên sau mỗi thời điểm t.

### *1.4.1 Hoạt động*

Tính toán gradient: AdaGrad tính toán gradient của hàm mất mát đối với từng tham số của mô hình.

Điều chỉnh learning rate: AdaGrad sử dụng một learning rate tự điều chỉnh cho mỗi tham số. Learning rate này phụ thuộc vào lịch sử của gradient đã tính toán cho tham số đó. Cách tính toán learning rate trong AdaGrad là bình phương của tổng bình phương của các gradient đã tính trước đó.

Công thức cập nhật trọng số:

$$\theta_{t+1,i} = \theta_{t,i} - \left(\frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}}\right) . g_{t,i}$$

Trong đó:

- $\theta_{t+1,i}$ là giá trị mới của tham số thứ i sau bước cập nhật.
- $\theta_{t,i}$ là giá trị hiện tại của tham số thứ i.
- $\eta$ là learning rate ban đầu.
- $g_{t,i}$ là gradient của tham số thứ i tại thời điểm t.
- $G_{t,ii}$ là tổng bình phương của các gradient đã tính trước đó cho tham số thứ i.
- $\varepsilon$ là một giá trị nhỏ (như 10^-8) được thêm vào để tránh việc chia cho 0.

### *1.4.2 Ưu điểm:*

Không cần điều chỉnh thủ công tốc độ học tập.

Hội tụ nhanh hơn

Đáng tin cậy hơn

### *1.4.3 Nhược điểm:*

Tỷ lệ học (learning rate) trở nên nhỏ khi mạng neural có độ sâu tăng lên.

Dễ dẫn đến dead neuron

## 1.5 RMSProp

RMSprop là một phương pháp tối ưu hóa thường được sử dụng trong quá trình huấn luyện mạng neural, đặc biệt là trong các mô hình sử dụng hàm kích hoạt phi tuyến tính như ReLU.

### *1.5.1 Hoạt động*

Exponential Moving Average (EMA) của bình phương gradient: RMSprop theo dõi EMA của bình phương gradient cho mỗi tham số của mạng neural.

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

Trong đó:

- $E[g^2]_t$ là EMA của bình phương gradient ở thời điểm t
- $g_t$ là gradient tại thời điểm t
- $\beta$ là trọng số giữa các bước gradient trước đó và bước gradient hiện tại (thường trong khoản 0.8 – 0.999)

Thay đổi learning rate: RMSprop chia learning rate cho căn bậc hai của EMA bình phương gradient tương ứng. Điều này giúp làm giảm learning rate cho các tham số mà gradient lớn và tăng learning rate cho các tham số mà gradient nhỏ.

$$learning\_rate_t = \frac{learning\_rate_0}{\sqrt{E[g^2]_t + \epsilon}}$$

Trong đó:

- $learning\_rate_t$ là learning rate ở thời điểm t
- $learning\_rate_0$ là learning rate ở thời điểm ban đầu
- $E[g^2]_t$ là EMA của bình phương gradient ở thời điểm t
- $\epsilon$ là một giá trị nhỏ (như 10^-8) được thêm vào để tránh việc chia cho 0.

Tính toán và cập nhật trọng số: RMSprop sử dụng learning rate đã điều chỉnh để cập nhật trọng số của mạng neural theo hướng giảm độ lớn của gradient.

Điều chỉnh adaptive learning rate: Phương pháp này giúp ổn định quá trình tối ưu hóa và giảm vấn đề về learning rate quá lớn hoặc quá nhỏ.

### *1.5.2 Ưu điểm*

Giải quyết được vấn đề tốc độ học giảm dần của Adagrad ( vấn đề tốc độ học giảm dần theo thời gian sẽ khiến việc training chậm dần, có thể dẫn tới bị đóng băng)

Hiệu suất trong việc học từ dữ liệu phi tuyến tính: Đặc biệt hiệu quả với các mô hình sử dụng hàm kích hoạt phi tuyến tính như ReLU.

Khả năng tối ưu hóa tốt hơn: RMSprop thường hoạt động tốt trong việc tối ưu hóa mô hình mạng neural và giảm thiểu vấn đề về learning rate không ổn định.

### *1.5.3 Nhược điểm:*

Có thể cho kết quả nghiệm chỉ là cực tiểu cục bộ chứ không đạt được cực tiểu toàn cục

Không cập nhật learning rate theo từng tham số một cách riêng biệt, có thể dẫn đến hiệu suất huấn luyện không tối ưu đối với các mô hình có độ phức tạp cao.

## 1.6 Momentum

Tối ưu hóa theo Momentum là một phương pháp tối ưu hóa trong quá trình huấn luyện mạng neural, nhằm cải thiện quá trình hội tụ và tăng tốc độ của việc tối ưu hóa.

### *1.6.1 Hoạt động*

Momentum: Đà đại diện cho độ lớn và hướng của việc cập nhật trọng số. Nó giữ thông tin về hướng di chuyển từ các bước cập nhật trước đó và giúp giảm thiểu độ dao động khi di chuyển đến điểm tối ưu.

Các Bước Tính Toán:

- Tại mỗi bước thời gian trong quá trình huấn luyện:
  + Tính gradient của hàm mất mát theo trọng số hiện tại
  + Cập nhật đà dựa trên gradient hiện tại và đà từ bước trước theo công thức

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

Trong đó:

- $v_t$ là đà tại thời điểm t
- $\beta$ là hệ số momentum, thường nằm trong khoảng 0.8 – 0.999

- $\eta$ là learning rate
- $\nabla J(\theta_t)$ là gradient của hàm mất mát J theo trọng số $\theta$ tại thời điểm t

Cập nhật trọng số: sau khi tính toán đà, cập nhật trọng số mới bằng công thức

$$\theta_{t+1} = \theta_t - v_t$$

Trong đó:

- $\theta_{t+1}$ là trọng số mới sau bước thời gian t+1
- $\theta_t$ là trọng số tại bước thời gian t
- $v_t$ là đà tính toán tại thời điểm t

### *1.6.2 Ưu Điểm:*

Hội tụ Nhanh Hơn: Giúp mô hình nhanh chóng hội tụ tới điểm tối ưu hóa.

Ổn Định Hơn: Giúp giảm đi sự dao động không cần thiết trong quá trình huấn luyện.

### *1.6.3 Nhược Điểm:*

Tùy Chỉnh Hệ Số Momentum ($\beta$): Sử dụng sai hệ số momentum có thể làm ảnh hưởng đến tốc độ hội tụ và ổn định của quá trình tối ưu hóa.

## 1.7 Adam

Adam (Adaptive Moment Estimation) là một phương pháp tối ưu hóa thông dụng trong huấn luyện mạng neural. Nó kết hợp cả Momentum và RMSprop, cung cấp sự linh hoạt và hiệu quả trong việc tối ưu hóa hàm mất mát.

### *1.7.1 Hoạt động*

Tính toán Momentum: Adam sử dụng đà (momentum) để tính toán thông tin về hướng di chuyển từ các gradient trước đó, tương tự như Momentum Optimization.

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1)(\nabla J(\theta_t))^2$$

Trong đó:

- $v_t$ là đà tại thời điểm t.

- $\beta_1$ là hệ số momentum (thường là 0.9).
- $\nabla J(\theta_t)$ là gradient của hàm mất mát J theo trọng số $\theta$ tại thời điểm t

Tính toán RMSprop: Adam cũng sử dụng RMSprop để điều chỉnh learning rate theo từng tham số riêng lẻ, giúp cân bằng giữa việc cập nhật các trọng số lớn và nhỏ.

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

Trong đó:

- $s_t$ là squared gradient tại thời điểm t
- $\beta_2$ là hệ số giảm giá trị của squared gradient (thường là 0.999)

Kết hợp Momentum và RMSprop: Adam kết hợp cả hai thông tin từ đà và RMSprop để điều chỉnh cả hướng di chuyển và learning rate, giúp quá trình hội tụ nhanh chóng hơn.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{s_t + \epsilon}} v_t$$

Trong đó:

- $\theta_{t+1}$ là trọng số mới sau bước thời gian t+1
- $\eta$ là learning rate
- $\epsilon$ là một giá trị nhỏ (như 10^-8) được thêm vào để tránh việc chia cho 0.

## 1.8 So sánh

| **Optimizer** | **Hoạt Động** | **Ưu Điểm** | **Nhược Điểm** |
|---|---|---|---|
| Gradient Descent | Tính Gradient, Cập Nhật Tham Số | Đơn giản, Hiệu Suất Tốt, Phổ Biến | Dễ Rơi vào Cực Tiểu Cục Bộ, Hội Tụ Chậm, Phụ Thuộc vào Learning Rate |

| | | | |
|---|---|---|---|
| SGD | Chọn Mẫu Ngẫu Nhiên, Tính Gradient, Cập Nhật | Tính Linh Hoạt, Nhanh Chóng với Dữ Liệu Lớn | Phương Sai Cao, Yêu Cầu Mô Hình Thay Đổi Liên Tục |
| Adagrad | Tính Gradient, Điều Chỉnh Learning Rate | Không Cần Điều Chỉnh Learning Rate, Hội Tụ Nhanh | Tỷ Lệ Học Giảm Dần, Dễ Dẫn Đến Dead Neuron |
| RMSprop | Exponential Moving Average, Thay Đổi Learning Rate | Giải Quyết Tốt Vấn Đề Tốc Độ Học Giảm Dần, Hiệu Suất Với Dữ Liệu Phi Tuyến | Có Thể Đạt Cực Tiểu Cục Bộ, Không Cập Nhật Learning Rate Riêng Biệt |
| Momentum | Đà, Tính Toán Gradient, Cập Nhật Trọng Số | Hội Tụ Nhanh Hơn, Ổn Định Hơn | Tùy Chỉnh Hệ Số Momentum, Không Ổn Định Nếu Hệ Số Sai |
| Adam | Tính Toán Momentum và RMSprop, Cập Nhật | Linh Hoạt, Hội Tụ Nhanh, Ổn Định Hơn | Có Thể Đạt Cực Tiểu Cục Bộ, Phức Tạp Trong Điều Chỉnh Tham Số |

Bảng 1.1: So sánh các Optimizer

Như vậy, mỗi Optimizer có các ưu điểm và nhược điểm riêng, và việc lựa chọn Optimizer thường phụ thuộc vào loại dữ liệu, cấu trúc mô hình, và yêu cầu cụ thể của bài toán.

# CHƯƠNG 2. CÂU 2

Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

## 2.1 Continual Learning

### *2.1.1 Định nghĩa*

Continual Learning (còn được gọi là Incremental Learning hoặc Life-long Learning) là một mô hình học máy tập trung vào việc liên tục cập nhật kiến thức mới và thích ứng với sự thay đổi của dữ liệu theo thời gian.

Trái với học máy truyền thống, các mô hình thường được đào tạo trên tập dữ liệu cố định và giả định rằng phân phối của dữ liệu không thay đổi, học liên tục được thiết kế để xử lý các phân phối dữ liệu đang thay đổi và liên tục học từ dữ liệu mới mà vẫn giữ được kiến thức từ những trải nghiệm trước đó. Điều này đặc biệt quan trọng khi dữ liệu không ổn định và thay đổi theo thời gian.

Ngoài ra, Continual Learning cũng có thể được xem như một thuật toán thích nghi, có khả năng học từ một luồng thông tin liên tục, với thông tin đó dần trở nên có sẵn theo thời gian và không có sẵn số lượng nhiệm vụ cần học trước. Quan trọng là việc tiếp nhận thông tin mới phải được thực hiện mà không làm mất thông tin cũ hoặc gây can thiệp quá mức vào quá trình học.

### *2.1.2 Thách thức*

#### 2.1.2.1 Quên Thảm Hại (Catastrophic Forgetting)

Catastrophic Forgetting là hiện tượng trong học máy, đặc biệt trong bối cảnh của học liên tục (Continual Learning), khi một mô hình học từ dữ liệu mới, nó có thể mất hoặc ghi đè hoàn toàn lên kiến thức đã học trước đó mà không giữ được hiệu suất của các nhiệm vụ hoặc dữ liệu cũ.

Trong quá trình học từ dữ liệu mới, mô hình có thể điều chỉnh các trọng số hoặc tham số của nó để phù hợp với dữ liệu mới này. Tuy nhiên, việc điều chỉnh này có thể dẫn đến việc mất đi hoặc làm mờ thông tin quan trọng đã học từ trước, dẫn đến

việc mô hình không thể hiệu quả trong việc nhớ lại hoặc sử dụng kiến thức từ các nhiệm vụ hoặc miền dữ liệu trước.

#### 2.1.2.2 Số liệu đánh giá (Evaluation Metrics)

Số liệu đánh giá (Evaluation Metrics) trong Continual Learning là các tiêu chí hoặc đơn vị đo lường được sử dụng để đánh giá hiệu suất và khả năng học liên tục của một mô hình máy học khi nó phải đối mặt và học từ dữ liệu mới một cách liên tục và ngày càng đa dạng. Các metrics này giúp đo lường khả năng của mô hình duy trì kiến thức từ các nhiệm vụ hoặc dữ liệu trước đó khi học từ dữ liệu mới, đồng thời đánh giá khả năng chuyển giao kiến thức và tránh hiện tượng Catastrophic Forgetting.

Trong Continual Learning, Evaluation Metrics thường bao gồm các chỉ số như Retention (duy trì thông tin), Adaptation (thích ứng), Interference (can thiệp), Forgetting Rate (tốc độ quên) và Transfer Efficiency (hiệu quả chuyển giao). Các số liệu đánh giá này tập trung vào việc đo lường sự linh hoạt, khả năng học từ dữ liệu mới mà không ảnh hưởng quá mức đến kiến thức đã học từ trải nghiệm trước, cũng như khả năng sử dụng lại kiến thức từ trước để học từ dữ liệu mới. Điều này giúp mô hình học liên tục duy trì và tích lũy kiến thức từ trải nghiệm trước đó trong quá trình học tiếp diễn.

#### 2.1.2.3 Hướng khắc phục

Bộ đệm phát lại (Replay Buffers): Bộ đệm phát lại lưu trữ một số mẫu dữ liệu quan trọng từ quá khứ và sử dụng chúng trong quá trình huấn luyện hiện tại. Khi mô hình học từ dữ liệu mới, nó cũng được huấn luyện với các mẫu dữ liệu cũ từ bộ đệm phát lại. Điều này giúp mô hình duy trì kiến thức từ các nhiệm vụ hoặc miền dữ liệu trước đó, ngăn chặn quên thông tin quan trọng.

Chính quy hóa (Regularization): Chính quy hóa trong Continual Learning bao gồm các kỹ thuật như Elastic Weight Consolidation (EWC) hoặc Synaptic Intelligence (SI). Các kỹ thuật này sửa đổi hàm mất mát (loss function) bằng cách thêm các thành phần chính quy cho các trọng số quan trọng. Việc này giúp mô hình

tránh sự thay đổi quá mức các trọng số quan trọng từ các nhiệm vụ trước, giữ cho thông tin quan trọng không bị mất.

Phương pháp tiếp cận kiến trúc (Architectural Approaches): Các phương pháp này thay đổi kiến trúc của mạng neural để tạo điều kiện cho việc học liên tục. Ví dụ, Progressive Neural Networks (PNNs) tăng dần mạng neural khi học được các nhiệm vụ mới, trong khi các mô hình mạng khác có thể mở rộng hoặc linh hoạt thay đổi kiến trúc tùy thuộc vào nhiệm vụ.

Học chuyển giao (Transfer Learning): Trong Continual Learning, học chuyển giao có thể được sử dụng để chuyển đổi kiến thức từ các tập dữ liệu đa dạng đã được đào tạo trước đó vào các nhiệm vụ mới. Mô hình được huấn luyện trước trên dữ liệu đa dạng có thể khái quát hóa tốt hơn khi học từ các nhiệm vụ mới, giúp mô hình tận dụng hiệu quả thông tin đã học.

Siêu học tập (Meta-Learning): Siêu học tập đào tạo mô hình cách học, giúp chúng nhanh chóng thích ứng và học từ các nhiệm vụ mới. Các thuật toán siêu học tập hướng tới việc tối ưu hóa quá trình học của mô hình, giúp chúng tiếp thu kiến thức mới một cách hiệu quả và nhanh chóng.

### *2.1.3 Các thuật toán*

#### 2.1.3.1 Progressive Neural Networks (PNNs)

Mạng thần kinh tiến bộ (PNN) được xây dựng để tiếp tục học và duy trì kiến thức từ các nhiệm vụ trước đó. PNN mở rộng mô hình khi có nhiệm vụ mới bằng cách sử dụng nhiều mạng nơ-ron cho mỗi nhiệm vụ cụ thể. Mỗi mạng được dành riêng cho một nhiệm vụ và kết quả của tất cả các mạng được kết hợp để đưa ra dự đoán. PNN ngăn chặn việc quên thông tin quan trọng của các nhiệm vụ trước bằng cách phân tách kiến thức vào các mạng riêng biệt. Tuy nhiên, việc quản lý một tập hợp lớn các mạng có thể tăng độ phức tạp tính toán.

#### 2.1.3.2 Learning without Forgetting (LwF)

Học mà không quên (LwF) là phương pháp sử dụng kiến thức từ mô hình đã được đào tạo trước đó (mô hình giáo viên) để huấn luyện mô hình mới (mô hình học

sinh). Khi học nhiệm vụ mới, mô hình học sinh được huấn luyện để sao chép dự đoán của mô hình giáo viên về cả dữ liệu cũ và mới. Quá trình này giúp mô hình học sinh duy trì kiến thức từ các nhiệm vụ trước.

#### 2.1.3.3 iCaRL (Incremental Classifier and Representation Learning)

iCaRL là thuật toán được thiết kế cho việc học liên tục trong phân loại. Nó kết hợp chiến lược học biểu diễn đặc trưng và lưu trữ mẫu đại diện từng lớp. Mô hình duy trì một tập các mẫu đại diện từ mỗi lớp đã học trước đó. Khi có lớp mới, iCaRL sử dụng các mẫu này để duy trì kiến thức về các lớp cũ và lớp mới.

#### 2.1.3.4 Meta-Learning Approaches

Siêu học tập (Meta-Learning) bao gồm việc đào tạo mô hình để học hiệu quả và cũng được áp dụng cho Continual Learning. Trong siêu học tập cho học liên tục, mô hình được đào tạo với nhiều nhiệm vụ để học cách thức khởi đầu hoặc học tập tốt để thích ứng nhanh chóng với các nhiệm vụ mới. Các kỹ thuật siêu học đã chứng minh tiềm năng trong việc giảm thiểu hiện tượng quên lãng nghiêm trọng bằng cách trang bị mô hình với kiến thức ban đầu mạnh mẽ để học từ các nhiệm vụ mới.

## 2.2 Test Production

### *2.2.1 Định nghĩa*

Test Production trong việc xây dựng giải pháp học máy là một bước quan trọng để đảm bảo rằng mô hình học máy đã được triển khai hoạt động chính xác và hiệu quả trong môi trường thực tế. Đây là quá trình kiểm tra mô hình trước khi nó được triển khai để đảm bảo rằng nó hoạt động như mong đợi và có khả năng chịu đựng với dữ liệu thực tế. Bước này giúp mô hình hoạt động đáng tin cậy và hiệu quả, đồng thời kiểm tra khả năng chịu đựng và đối phó với dữ liệu mới.

### *2.2.2 Hoạt động*

#### 2.2.2.1 Đánh giá mô hình (Model Evaluation)

Giai đoạn ban đầu của quá trình kiểm thử mô hình (gọi là giai đoạn 0) tập trung vào việc đánh giá chức năng của mô hình. Các số liệu đánh giá khác nhau tùy thuộc vào loại mô hình và tính chất của tập dữ liệu. Ví dụ, trong các tác vụ phân loại với các tập dữ liệu mất cân bằng, F1 score và AUC score được coi là các chỉ số hiệu quả cho chất lượng của mô hình.

#### 2.2.2.2 Kiểm thử trước huấn luyện (Pre-training test)

Là các bài kiểm tra được thực hiện trước khi huấn luyện mô hình để kiểm tra xem mô hình có đáp ứng yêu cầu hay không, chẳng hạn như định dạng dữ liệu đúng, đủ dữ liệu, v.v.

#### 2.2.2.3 Kiểm thử sau huấn luyện (Post-training testing)

Là các bài kiểm tra sau khi hoàn tất giai đoạn huấn luyện. Trong học theo lô (batch learning), kiểm thử sau huấn luyện được thực hiện sau khi hoàn tất giai đoạn huấn luyện, trong khi trong học trực tuyến (online learning), các bài kiểm tra này diễn ra trong quá trình huấn luyện. Trong học trực tuyến, kiểm thử độ trễ (Latency Test) đảm bảo rằng dự đoán được thực hiện trong một phần của giây để đảm bảo hiệu suất mạnh mẽ và hiệu quả trong lưu lượng truy cập. Đây là đặc biệt quan trọng đối với các phương pháp học máy trực tuyến.

#### 2.2.2.4 Kiểm thử A/B

Kiểm thử A/B đối với việc huấn luyện lại mô hình sau thời gian, khi các đặc trưng dữ liệu có thể thay đổi, dẫn đến thách thức trong các mô hình học máy và trí tuệ nhân tạo. Đào tạo lại trở nên cần thiết sau khi triển khai mô hình học máy để xử lý các sự thay đổi này.

#### 2.2.2.5 Kiểm thử tạm (Stage test/Shadow test)

Kiểm thử tạm là một trong các kiểm thử cuối cùng để kiểm tra xem mô hình có đưa ra kết quả mong muốn không, sử dụng dữ liệu kiểm thử đa dạng như đặc tính dữ liệu thực tế.

#### 2.2.2.6 Kiểm thử API (Application Programming Interface)

Đây là giai đoạn cuối cùng, tập trung chủ yếu vào cách người dùng cuối thực sự trải nghiệm các phản hồi từ mô hình thông qua API của nó. Giai đoạn này nhằm mô phỏng các tương tác thực tế của người dùng với mô hình.

### *2.2.3 Ưu điểm*

Tính chính xác: Kiểm thử sản xuất giúp đảm bảo mô hình hoạt động chính xác trong môi trường thực tế.

Tính ổn định: Mô hình được kiểm thử để đảm bảo nó hoạt động ổn định và hiệu suất ở mức cao trong thời gian dài.

Tối ưu hóa liên tục: Cập nhật và tinh chỉnh mô hình để duy trì hoặc nâng cao hiệu suất theo thời gian.

### *2.2.4 Nhược điểm*

Chi phí và thời gian: Quá trình kiểm thử sản xuất có thể tốn kém về thời gian và nguồn lực.

Rủi ro khi triển khai: Mô hình có thể không hoạt động tốt trong môi trường thực tế do các yếu tố không dự đoán được.