TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**LÝ HOÀNG GIA HUY - 52100894**

**BÁO CÁO CUỐI KỲ**

**NHẬP MÔN HỌC MÁY**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**LÝ HOÀNG GIA HUY - 52100894**

**BÁO CÁO CUỐI KỲ**

**NHẬP MÔN HỌC MÁY**

Người hướng dẫn

**PGS.TS. Lê Anh Cường**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

**LỜI CẢM ƠN**

Chúng em xin chân thành cảm ơn thầy Lê Anh Cường đã tận tình giảng dạy em trong quá trình học tập và thực hiện báo cáo này. Em xin chân thành cảm ơn khoa Công Nghệ Thông Tin cũng như Trường Đại học Tôn Đức Thắng đã tạo điều kiện cho em thực hiện báo cáo

*TP. Hồ Chí Minh, ngày 23 tháng 12 năm 2023.*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Lý Hoàng Gia Huy*

**TÓM TẮT**

Trong báo cáo này em sẽ trình bày về các phương pháp tối ưu học mày, continual learning và việc kiểm thử một hệ thống machine learning

**MỤC LỤC**

[DANH MỤC HÌNH VẼ v](#_Toc154314396)

[DANH MỤC BẢNG BIỂU vi](#_Toc154314397)

[DANH MỤC CÁC CHỮ VIẾT TẮT vii](#_Toc154314398)

[CHƯƠNG 1. MỞ ĐẦU 1](#_Toc154314399)

[1.1 Optimizers 1](#_Toc154314400)

[CHƯƠNG 2. CÁC PHƯƠNG PHÁP OPTIMIZER TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY 2](#_Toc154314401)

[2.1 Gradient Descent 2](#_Toc154314402)

[2.1.1 Định nghĩa 2](#_Toc154314403)

[2.1.2 Cách thức hoạt động 2](#_Toc154314404)

[2.1.3 Ưu và nhược điểm 3](#_Toc154314405)

[2.2 Stochastic Gradient Descent 4](#_Toc154314406)

[2.2.1 Định nghĩa 4](#_Toc154314407)

[2.2.2 Ưu và nhược điểm 4](#_Toc154314408)

[2.3 Momentum 6](#_Toc154314409)

[2.3.1 Gradient Descent dưới góc nhìn vật lý 6](#_Toc154314410)

[2.3.2 Gradient Descent với Momentum 7](#_Toc154314411)

[2.3.3 Ưu và nhược điểm 8](#_Toc154314412)

[2.4 AdaGrad (Adaptive Gradient) 8](#_Toc154314413)

[2.4.1 Định nghĩa 8](#_Toc154314414)

[2.4.2 Ưu và nhược điểm 11](#_Toc154314415)

[CHƯƠNG 3. CONTINUAL LEARNING VÀ TEST PRODUCTION 12](#_Toc154314416)

[3.1 Continual Learning 12](#_Toc154314417)

[3.2 Test Production 13](#_Toc154314418)

[3.2.1 Đánh giá mô hình 14](#_Toc154314419)

[3.2.2 Unit testing 14](#_Toc154314420)

[3.2.3 Post Training Test 15](#_Toc154314421)

# DANH MỤC HÌNH VẼ

[Hình 2.1: Đường đi của Gradient Descent 3](#_Toc142677545)  
Hình 2.2: Nhược điểm của Gradient Descent đối với learning rate 4  
Hình 2.3: Đường đi của SGD so với GD 4  
Hình 2.4: GD và GD với momentum 7  
Hình 2.5: So sánh đường đi của GD và GD với momentum 8  
Hình 2.6: AdaGrad trên cây sơ đồ các optimizers 9  
Hình 2.7: Vấn đề vể tỉ lệ học của GD 10  
Hình 2.8: Giá trị mất mát giảm dần khi áp dụng AdaGrad 10  
Hình 3.1: Continual Learning 13  
Hình 3.2: Thử thách catastrophic forgetting đối với CL 13  
Hình 3.3: Kiểm thử phần mềm và kiểm thử hệ thống ML 13  
Hình 3.4: ML test flow 14  
Hình 3.5: Các kỹ thuật đánh giá mô hình 14  
Hình 3.6: Các kiểu Unit Testing 15  
Hình 3.7: Latency Testing và Load Testing 16

# DANH MỤC BẢNG BIỂU

# DANH MỤC CÁC CHỮ VIẾT TẮT

1. GD : Gradient Descent
2. SGD : Stochastic Gradient Descent
3. CL : Continual Learning
4. ML : Machine Learning

# MỞ ĐẦU

## Optimizers

Trong machine learning, các optimizers là các thuật toán hoặc phương pháp giúp điều chỉnh các **tham số** (*weight, bias*) của mô hình sao cho **giá trị của hàm mất mát** (*loss function*) là *tối thiểu* nhất có thể. Mục tiêu của các optimizer là tìm một tập các giá trị tham số tối ưu mang lại hiệu suất tốt nhất có thể của mô hình cho một task cho trước.

Trong quá trình huấn luyện, optimizer sẽ cập nhật liên tục các tham số của mô hình dựa trên việc tính toán **gradients** (độ dốc) của **hàm mất mát** dựa trên các tham số đó. Các gradients sẽ cho chúng ta biết hướng mà các tham số cần được điều chỉnh sao cho giảm thiểu giá trị mất mát. Optimizer sẽ dùng các gradients này để cập nhật các tham số theo một cách giảm dần giá trị mất mát và tăng hiệu suất mô hình.

Có rất nhiều dạng thuật toán tối ưu cho nhiều dạng mô hình học máy. Trong đó có một số optimizers thường được sử dụng như: Gradient Descent, Stochastic Gradient Descent (biến thể của Gradient Descent), Momentum (thuật toán tối ưu cho Gradient Descent) và AdaGra (thay đổi tỉ lệ học dựa trên các gradient trước đó).

# CÁC PHƯƠNG PHÁP OPTIMIZER TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY

## Gradient Descent

### Định nghĩa

Gradient Descent (suy giảm độ dốc) là một thuật toán đi tìm các hướng đi trên hàm mất mát sao cho giá trị mất mát giảm nhanh nhất và cập nhật lại các tham số của mô hình cho phù hợp.

### Cách thức hoạt động

Để có thể tìm được hướng đi sao cho giá trị mất mát giảm, nghĩa là ta đang tìm điểm cực tiểu của hàm mất mát nhưng không phải lúc nào chúng ta cũng tìm được điểm này nên thuật toán sẽ chạy cho đến khi hội tụ gần điểm cực tiểu hoặc đã đến số lần lặp quy định trước.

Xét một đạo hàm tại nếu nghĩa là nằm bên phải và ngược lại với Chúng ta cần đi ngược dấu đạo hàm.

Từ ý tưởng trên ta có được công thức:

Trong đó:

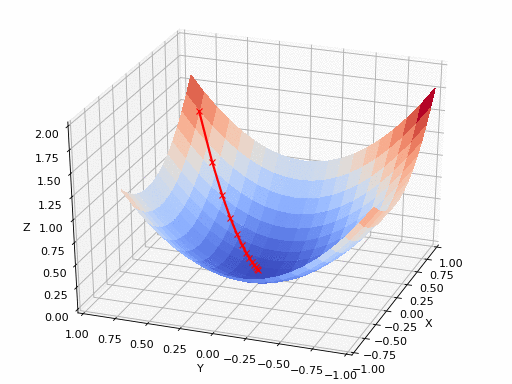
* là tham số tại lần lặp *t*
* là tỉ lệ học (learning rate)
* là đạo hàm tại
* Dấu “-“ ám chỉ cho việc ta đi ngược dấu đạo hàm

Đối với hàm nhiều biến:

Trong đó:

* là một vector
* là đạo hàm của hàm mất mát tại một điểm bất kỳ

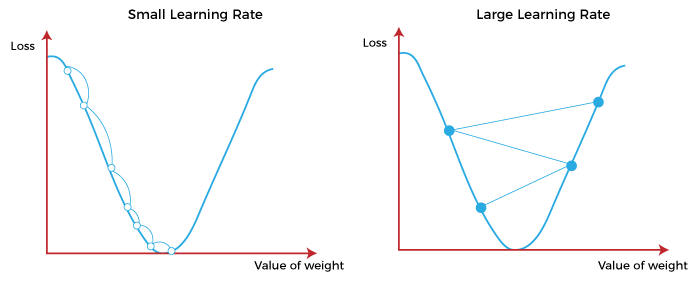
Từ đây để cài đặt thuật toán đầu tiên chúng ta sẽ khởi tạo các tham số ngẫu nhiên sau đó lặp công thức cho đến khi hội tụ



Hình 2.1: Đường đi của Gradient Descent

### Ưu và nhược điểm

* Ưu điểm: Thuật toán khá cơ bản, dễ hiểu và dễ cài đạt
* Nhược điểm:
* Phụ thuộc vào tham số khởi tạo và learning rate.
* Giả sử có nhiều local minimum trong 1 hàm số, thì tùy thuộc vào các điểm khởi tạo ban đầu sẽ cho ra các nghiệm khác nhau
* Tốc độ học quá lớn hoặc quá nhỏ sẽ khiến thuật toán khó hội tụ.



Hình 2.2: Nhược điểm của Gradient Descent đối với learning rate

## Stochastic Gradient Descent

### Định nghĩa

SGD là một biến thể của GD, thay vì mỗi lần lặp chúng ta sẽ dùng toàn bộ tập dữ liệu như GD, SGD chỉ sẽ sử dụng *N* phần tử ngẫu nhiên cho việc tính toán gradient và cập nhật tham số của mô hình. Với hướng tiếp cận này, SGD sẽ có độ phức tạp thời gian và không gian ở mỗi lần lặp rất nhỏ.

Nhìn chung SGD sẽ làm giảm tốc độ với cùng số lần lặp nhưng vì chi phí trong mỗi lần lặp rất ít nên ta có thể lặp nhiều hơn thế trong cùng một khoảng thời gian, khiến cho tốc độ hội tụ của SGD vượt trội GD.

A diagram of a celestial mass

Description automatically generated with medium confidence

Hình 2.3: Đường đi của SGD so với GD

### Ưu và nhược điểm

* Ưu điểm:
* Thuật toán giải quyết được đối với cơ sở dữ liệu lớn mà GD không làm được.
* Do chỉ tính một mẫu dữ liệu trong toàn bộ tập dữ liệu, SGD có thể tính toán trên tập dữ liệu lớn (big data) hay luồng dữ liệu (stream data)
* Khi bắt gặp các **điểm yên ngựa** (saddle points), GD có thể bị kẹt tại đó, nhưng vì có đường đi khá zig zag nên SGD có thể \_thoát khỏi\_ được các điểm yên ngựa đó. Cũng vì có đặc điểm đó, SGD cũng có thể \_thoát khỏi\_ được các điểm cực tiểu cục bộ hẹp

A blue grid with white lines

Description automatically generated

A blue square object with white lines

Description automatically generated

* Nhươc điểm:
* Vẫn chưa giải quyết được 2 nhược điểm lớn của GD (learning rate và điểm dữ liệu ban đầu)

## Momentum

### Gradient Descent dưới góc nhìn vật lý

Thuật toán GD được ví như là tác dụng trọng lực lên một hòn bi được đặt trên một mặt có dạng như thung lũng

A green graph with red and yellow dots

Description automatically generated

Hình 2.4: GD và GD với momentum

Như hình b phía trên, nếu ta thả 2 viên bi tại 2 điểm khác nhau A và B thì viên bị A sẽ trượt xuống điểm C còn viên bi B sẽ trượt xuống điểm D, nhưng ta lại không mong muốn viên bi B sẽ dừng ở điểm D (local minimum) mà sẽ tiếp tục lăn tới điểm C (global minimum). Nếu tại điểm B, viên bi có một vận tốc đủ lớn,khi bi lăn đến điểm D, theo ***đà***, bi có thể tiếp tục di chuyển lên dốc phía bên trái của D. Và nếu giả sử vận tốc ban đầu lớn hơn nữa, bi có thể vượt dốc tới điểm E rồi lăn xuống C như hình c

Dựa trên hiện tượng này, một thuật toán được ra đời nhằm khắc phục việc nghiệm của GD rơi vào một điểm local minimum không mong muốn. Thuật toán đó có tên là Momentum

### Gradient Descent với Momentum

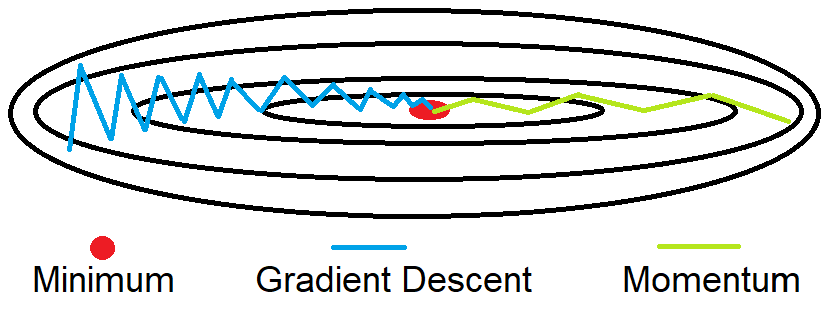
Trong GD, chúng ta cần tính lượng thay đổi ở thời điểm *t* để cập nhật vị trí mới cho nghiệm(tức *hòn bi*). Nếu chúng ta coi đại lượng này như vận tốc  trong vật lý, vị trí mới của *hòn bi* sẽ là .

Bây giờ là tính đại lượng sao cho nó vừa mang thông tin của *độ dốc* (tức đạo hàm), vừa mang thông tin của *đà*, tức vận tốc trước đó  (chúng ta coi như vận tốc ban đầu ):

Trong đó:

* thường được chọn là một giá trị khoảng 0.9
* là vận tốc tại thời điểm trước đó
* chính là độ dốc của điểm trước đó

Sau đó vị trí mới của *hòn bi* được xác định như sau:



Hình 2.5: So sánh đường đi của GD và GD với momentum

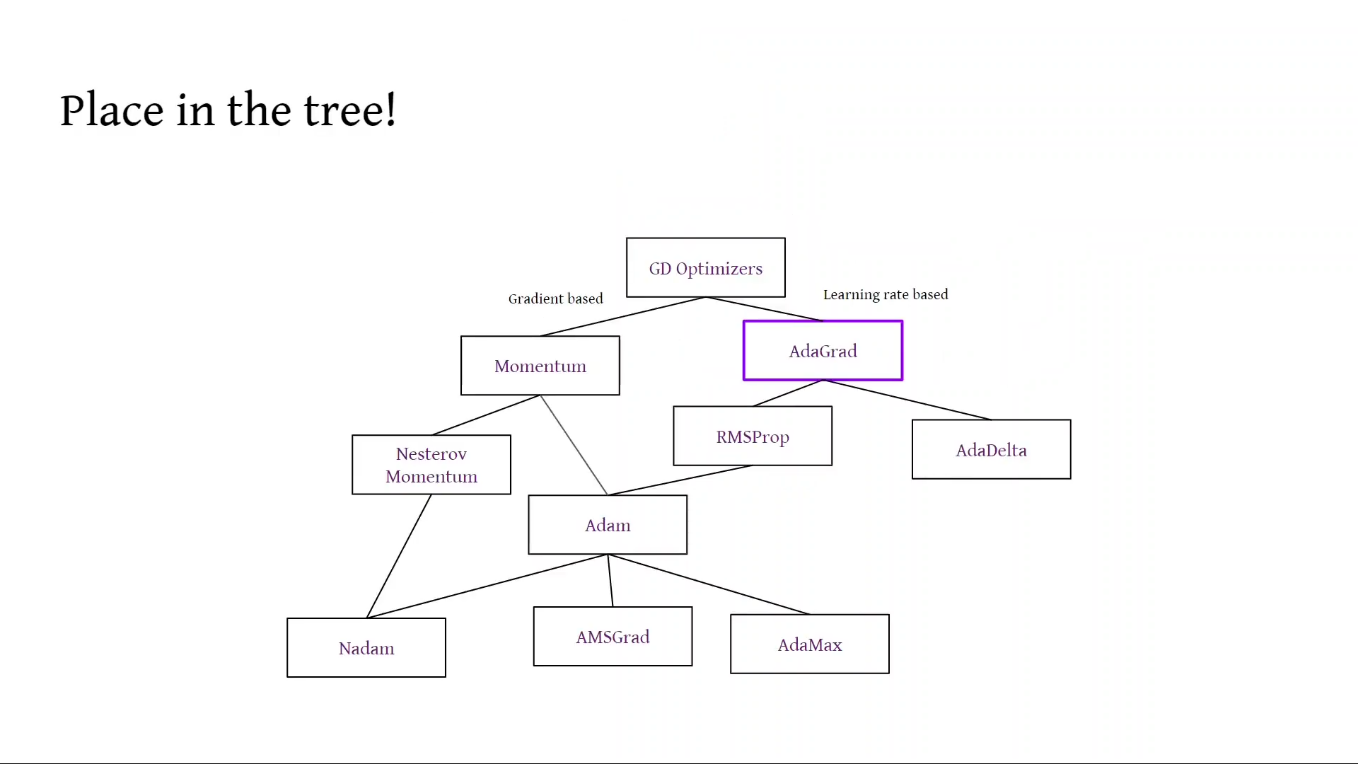
### Ưu và nhược điểm

* Ưu điểm: Thuật toán tối ưu giải quyết được vấn đề: Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.
* Nhược điểm: Tuy momentum giúp hòn bi vượt dốc tiến tới điểm đích, tuy nhiên khi tới gần đích, nó vẫn mất khá nhiều thời gian giao động qua lại trước khi dừng hẳn, điều này được giải thích vì viên bi có đà.

## AdaGrad (Adaptive Gradient)

### Định nghĩa

Khác với các thuật toán tối ưu đã nói phía trên, AdaGrad là một thuật toán tối ưu dựa trên **tốc độ học** (learning rate).



Hình 2.6: AdaGrad trên cây sơ đồ các optimizers

Trở về với vấn đề của GD, khi tốc độ học nhỏ, mỗi lần hàm số giảm rất ít nên phải qua rất nhiều lần lặp để hàm số đạt giá trị nhỏ nhất và ngoài ra hàm số có thể hội tụ đến một **điểm cực tiểu cục bộ chưa tối ưu** (non-optimal local minimum). Còn khi tốc độ học quá lớn, hàm số có thể sẽ hội tụ nhanh hơn nhưng sẽ gây ra hiện tượng **overshoot** (nghĩa là vượt quá điểm giá trị nhỏ nhất của hàm) và không bao giờ đạt được giá trị nhỏ nhất của hàm.

A diagram of a graph

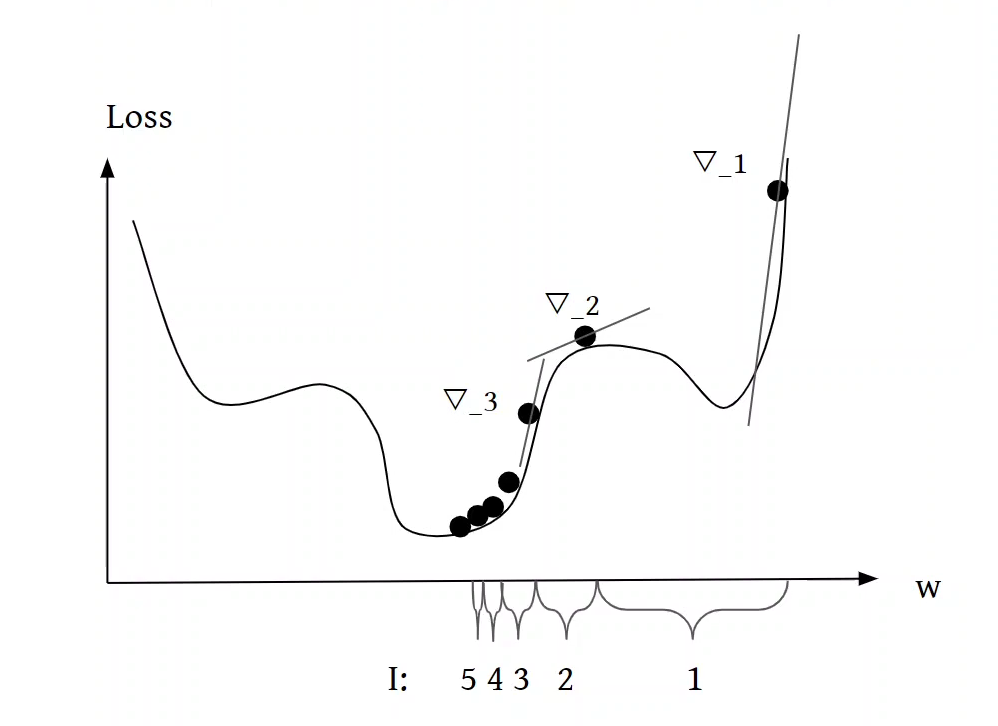
Description automatically generated

Hình 2.7: Vấn đề vể tỉ lệ học của GD

Vậy ý tưởng đưa ra là chúng ta có thể thay đổi tốc độ học này dựa trên **giá trị gradient trước đó theo một tham số cụ thể** (ký hiệu: , với *t* là các lần lặp)

Theo ý tưởng này ta có được công thức tính learning rate tại mỗi lần lặp *t* là:

Trong đó là độ lớn của vector chứa các giá trị gradient của mỗi lần lặp từ trước đó theo một tham số cụ thể.



Hình 2.8: Giá trị mất mát giảm dần khi áp dụng AdaGrad

Khi đó learning rate sẽ biến thiên sau mỗi thời điểm *t* và giúp cho thuật toán dần hội tụ về cực tiểu toàn cục.

Để thực hiện ý tưởng này, ở mỗi bước lặp ta sẽ cần lưu trữ tạm giá trị gradient bình phương tại bước lặp đó.

Giá trị tham số mới lúc này sẽ là:

Trong đó: là hệ số tránh lỗi (chia cho mẫu bằng 0)

Từ đây ta thấy rằng, các giá trị gradient trước đó càng lớn thì trọng số tỉ lệ học sẽ càng nhỏ.

### Ưu và nhược điểm

* Ưu điểm: Một lơi ích dễ thấy của Adagrad là tránh việc điều chỉnh learning rate bằng tay, chỉ cần để tốc độ học default là 0.01 thì thuật toán sẽ tự động điều chỉnh.
* Nhược điểm: Yếu điểm của Adagrad là tổng bình phương biến thiên sẽ lớn dần theo thời gian cho đến khi nó làm tốc độ học cực kì nhỏ, làm việc training trở nên đóng băng

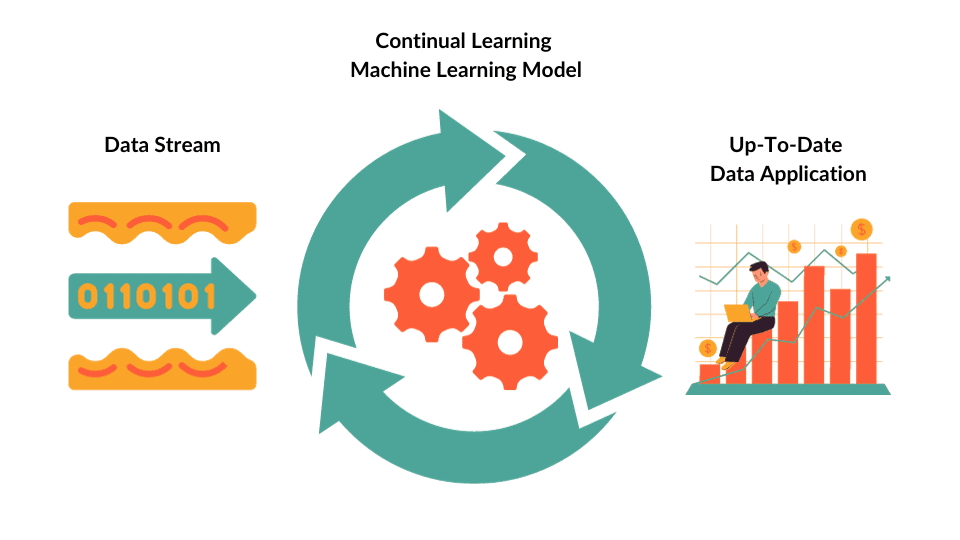
# CONTINUAL LEARNING VÀ TEST PRODUCTION

## Continual Learning

Continual Learning hay còn được gọi với cái tên khác là lifelong learning, increasement learning là một **hình mẫu học máy** mà tập trung vào một hệ thống học có khả năng tiếp thu được các kiến thức và kỹ năng mới **theo thời gian** mà **không quên đi các thông tin đã học** trước đó. Mục đích của hệ thống này là làm cho trí tuệ nhân tạo có khả năng học và thích nghi giống với con người.

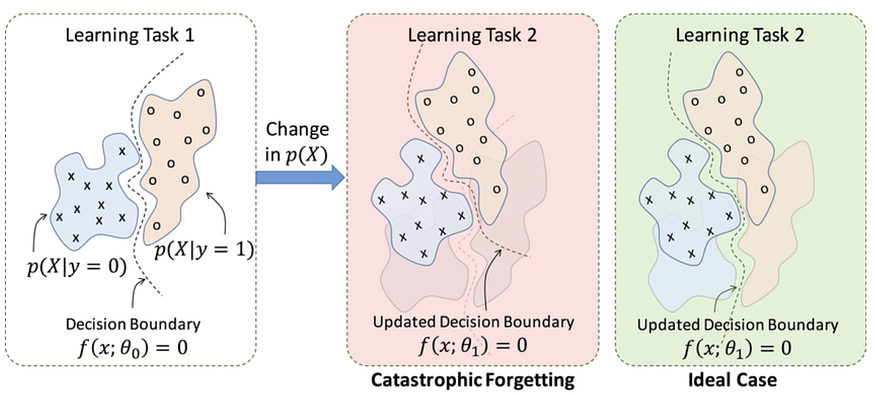
Ở các mô hình học máy truyền thống, các mô hình thường được huyến luyện với các dữ liệu tĩnh (fixed data), sau khi được huấn luyện, mô hình sẽ được triển khai và sử dụng để suy luận trên tập dữ liệu mới. Tuy nhiên, hướng tiếp cận này có những hạn chế khi xử lý các phân phối dữ liệu đang phát triển hoặc học từ các tasks mới mà không cần phải huấn luyện lại từ đầu.

Continual learning có thể giải quyết được các hạn chế đó. Cho một luồng dữ liệu (stream data), với một thuật toán CL sẽ học từng phần của dữ liệu đó (sequence of partial experiences) vì dữ liệu không thể luôn có sẵn.



Hình 3.1: Continual Learning

Nhưng đi với nó là các thử thách. Một trong những thử thách lớn nhất của CL là catastrophic forgetting, tức là các mô hình thường có xu hướng quên đi các thông tin đã học từ trước khi tiếp xúc với dữ liệu mới. Vấn đề này phát sinh do sự mất cân bằng được tạo ra giữa tầm quan trọng của việc học thông tin mới và việc lưu trữ kiến ​​thức cũ.



Hình 3.2: Thử thách catastrophic forgetting đối với CL

## Test Production

Việc kiểm thử một hệ thống ML có thể được phân biệt với việc kiểm thử các phần mềm. Trong khi việc kiểm thử phần mềm là kiểm tra với một đầu vào là dữ liệu và logic, đầu ra là một kết quả mong muốn thì với việc kiểm thử hệ thống ML chúng ta sẽ có đầu vào là dữ liệu và kết quả mong muốn với đầu ra là logic đã được học.

A screenshot of a computer

Description automatically generated

Hình 3.3: Kiểm thử phần mềm và kiểm thử hệ thống ML

A diagram of a training process

Description automatically generated

Hình 3.4: ML test flow

### Đánh giá mô hình

Đánh giá mô hình là giai đoạn đầu tiên trong việc kiểm thử hệ thống ML và chỉ giới hạn ở các chức năng của mô hình. Chúng ta có thể sử dụng các kỹ thuật đánh giá với nhiều loại mô hình khác nhau

A blue rectangular box with black text

Description automatically generated

Hình 3.5: Các kỹ thuật đánh giá mô hình

### Unit testing

Khi chúng ta xây dựng mô hình theo hướng online learning hoặc batch learning, trước khi huấn luyện chúng ta sẽ cần các unit test cho mô hình với nhiều loại dữ liệu.

A diagram of a software development

Description automatically generated

Hình 3.6: Các kiểu Unit Testing

### Post Training Test

Post Training Test được thực hiện sau khi huấn luyện đối với batch learning và trong quá trình huấn luyện đối với online learning.

Ở giai đoạn này, chúng ta sẽ có hai loại test là Latency test và Load test

Đối với latency test, chúng ta sẽ kiểm tra việc dự đoán của mô hình có được thực hiện trong vòng một phần giây hay không để từ đó mô hình có thể mở rộng và xử lý lưu lượng. Nếu việc dự đoán mất từ một phút trở lên, hầu như chúng ta cần phải thay đổi thiết kế mô hình. Đây là một test rất quan trọng với online learning.

Đối với load test, chúng ta sẽ kiểm tra mô hình sẽ có thể xử lý bao nhiêu dữ liệu cùng một lúc. Việc kiểm thử này đều quan trọng cho cả batch learning và online learning.

A screenshot of a computer program

Description automatically generated

Hình 3.7: Latency Testing và Load TestingTÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Optimizer- Hiểu sâu về các thuật toán tối ưu ( GD,SGD,Adam,..) (<https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8#_2-stochastic-gradient-descent-sgd-3>)

[2] Bài 7: Gradient Descent (phần 1/2) (<https://machinelearningcoban.com/2017/01/12/gradientdescent/>)

[3] Bài 8: Gradient Descent (phần 2/2) (<https://machinelearningcoban.com/2017/01/16/gradientdescent2/>)

Tiếng Anh

[1] AdaGrad Optimizer For Gradient Descent (https://www.youtube.com/watch?v=RXgF14WblVo)

[2] Machine Learning Model Testing for Production (https://medium.com/bright-ml/machine-learning-artificial-intelligence-testing-for-production-c561590ff292)

[3] The Unreasonable Effectiveness of Stochastic Gradient Descent (in 3 minutes) (https://www.youtube.com/watch?v=UmathvAKj80)