VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PROJECT

INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Authors: **Luong Khanh Vy – 521H0337**

Class**:** **21H50301**

Admission Course**:** **25**

Supervisor: **Prof. LE ANH CUONG**

HO CHI MINH CITY, 2023

VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

PROJECT

INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

Authors: **Luong Khanh Vy – 521H0337**

Class**:** **21H50301**

Admission Course**:** **25**

Supervisor: **Prof. LE ANH CUONG**

HO CHI MINH CITY, 2023

ACKNOWLEDGEMENTS

We are really grateful because we managed to complete our report assignment about Introduction To Machine Learning within the time given. We sincerely thank our lecturer, Mr. Le Anh Cuong for the guidance and encouragement in finishing this assignment and also teaching us in this course.

PROJECT COMPLETED AT TON DUC THANG UNIVERSITY

We hereby declare that this is our own project and is under the guidance of Mr. Le Anh Cuong. The research contents and results in this topic are honest and have not been published in any publication before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data of other authors, other agencies and organizations, with citations and source annotations.

If we find any fraud, we will take full responsibility for the content of our project. Ton Duc Thang University is not related to copyright and copyright violations caused by us (if any).

Ho Chi Minh City, 24th December , 2023

Authors

(signature and full name)

Luong Khanh Vy

LECTURER'S ASSESSMENT

Ho Chi Minh, date

(sign)

ABSTRACT

CONTENT

LIST OF ABBREVIATION

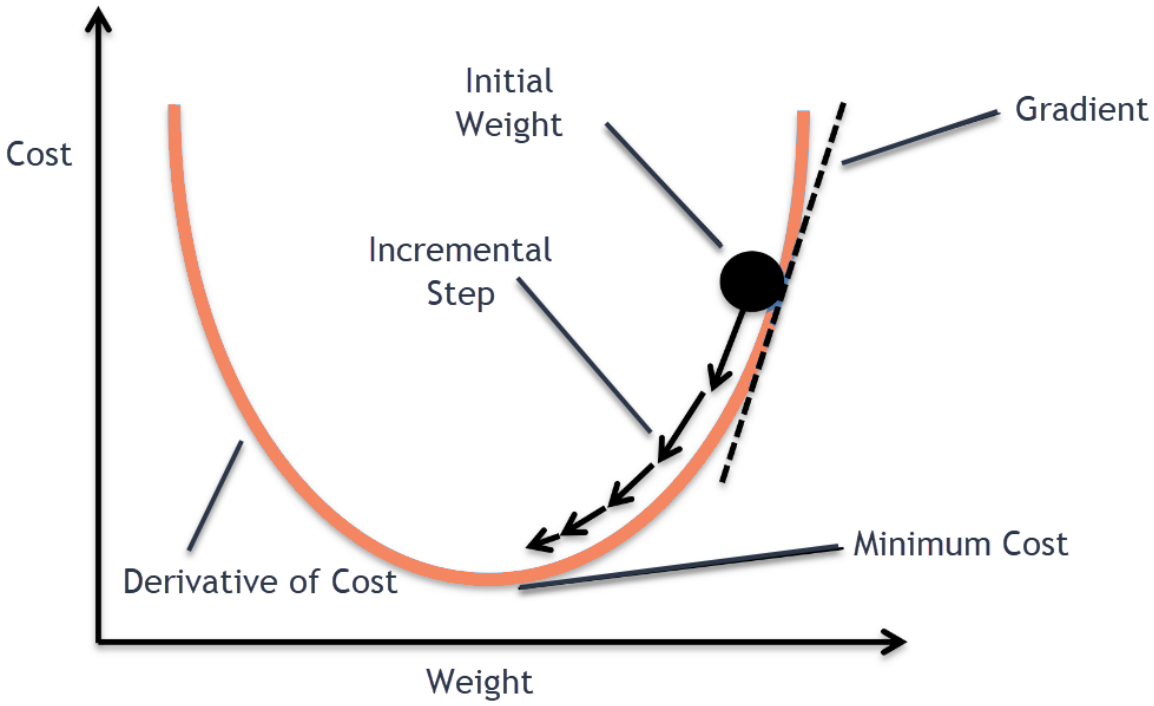
LIST OF FIGURE

Câu 1:

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

1.1 Gradient Descent (GD):

- GD là một khái niệm cơ bản trong học sâu nhằm phối hợp quá trình tối ưu hóa mô hình phức tạp. Về cơ bản, đây là một kỹ thuật tối ưu hóa số nhằm tìm cách giảm thiểu một hàm chi phí cụ thể để xác định các tham số mạng thần kinh lý tưởng, chẳng hạn như trọng số và độ lệch.
- Trong khi đào tạo mô hình dựa trên mạng thần kinh, việc học diễn ra trong quá trình lan truyền ngược. GD là một thuật ngữ được sử dụng để tối ưu hóa độ lệch và trọng số theo hàm chi phí. Sự khác biệt giữa đầu ra dự kiến và đầu ra thực tế được đánh giá bằng cách sử dụng hàm chi phí.
- Gradient Descent(GD) là một toán tối ưu hóa được sử dụng rộng rãi trong học sâu, được sử dụng để giảm thiểu hàm chi phí của mô hình mạng thần kinh trong quá trình đào tạo. Nó hoạt động bằng cách điều chỉnh lặp đi lặp lại các trọng số hoặc tham số của mô hình theo hướng gradient âm của hàm chi phí cho đến khi đạt được mức tối thiểu của hàm chi phí.



- Gradient Descent là một thuật toán tối ưu lặp (iterative optimization algorithm) được sử dụng trong các bài toán Machine Learning và Deep Learning (thường là các bài toán tối ưu lồi — Convex Optimization) với mục tiêu là tìm một tập các biến nội tại (internal parameters) cho việc tối ưu models. Trong đó:
  - Gradient: là tỷ lệ độ nghiêng của đường dốc (rate of inclination or declination of a slope). Về mặt toán học, Gradient của một hàm số là đạo hàm của hàm số đó tương ứng với mỗi biến của hàm. Đối với hàm số đơn biến, chúng ta sử dụng khái niệm Derivative thay cho Gradient.

- Descent: là từ viết tắt của descending, nghĩa là giảm dần.

Gradient Descent có nhiều dạng khác nhau như Stochastic Gradient Descent (SGD), Mini-batch SDG. Nhưng về cơ bản thì đều được thực thi như sau:

- Khởi tạo biến nội tại.
- Đánh giá model dựa vào biến nội tại và hàm mất mát (Loss function).
- Cập nhật các biến nội tại theo hướng tối ưu hàm mất mát (finding optimal points).
- Lặp lại bước 2, 3 cho tới khi thỏa điều kiện dừng.
- Công thức cập nhật cho GD có thể được viết là:

$$(\mathrm{w})_{\mathrm{new}} = (\mathrm{w})_{\mathrm{old}} - \eta \frac{\partial \mathrm{L}}{\mathrm{dw}(\ \mathrm{old}\ )}$$

Trong đó là tập các biến cần cập nhật, là tốc độ học (learning rate), Θ là Gradient của hàm mất mát f theo tập θ.

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w(t-1)}$$

- Ưu điểm của phương pháp Gradient Descent (GD) và các biến thể của nó:
  + Được sử dụng rộng rãi: GD và các biến thể của nó được sử dụng rộng rãi trong các vấn đề tối ưu hóa và học máy vì chúng hiệu quả và dễ thực hiện.
  + Hội tụ: GD và các biến thể của nó có thể hội tụ đến mức tối thiểu toàn cầu hoặc mức tối thiểu cục bộ tốt của hàm chi phí, tùy thuộc vào vấn đề và biến thể được sử dụng.
  + Khả năng mở rộng: Nhiều biến thể của phương pháp GD có thể được song song hóa và có thể mở rộng thành các bộ dữ liệu lớn và mô hình nhiều chiều.
  + Tính linh hoạt: Các biến thể khác nhau của phương pháp GD mang lại sự cân bằng giữa độ chính xác và tốc độ và có thể được điều chỉnh để tối ưu hóa hiệu suất của một vấn đề cụ thể.
- Nhược điểm của việc Gradient Descent(GD) và các biến thể của nó:
  + Lựa chọn tốc độ học: Việc lựa chọn tốc độ học là rất quan trọng đối với sự hội tụ của độ dốc giảm dần và các biến thể của nó. Chọn tốc độ học quá lớn có thể dẫn đến dao động hoặc vọt lố trong khi chọn tốc độ học quá nhỏ có thể dẫn đến hội tụ chậm hoặc bị kẹt ở cực tiểu cục bộ.
  + Độ nhạy đối với việc khởi tạo: Độ dốc giảm dần và các biến thể của nó có thể nhạy cảm với việc khởi tạo các tham số của mô hình, điều này có thể ảnh hưởng đến độ hội tụ và chất lượng của giải pháp.
  + Tốn thời gian: Giảm dần độ dốc và các biến thể của nó có thể tốn thời gian, đặc biệt là khi xử lý các tập dữ liệu lớn và mô hình nhiều chiều. Tốc độ hội tụ cũng có thể khác nhau tùy thuộc vào biến thể được sử dụng và vấn đề cụ thể.
  + Tối ưu cục bộ: Độ dốc giảm dần và các biến thể của nó có thể hội tụ

đến mức tối thiểu cục bộ thay vì mức tối thiểu toàn cầu của hàm chi phí, đặc biệt là trong các vấn đề không lồi. Điều này có thể ảnh hưởng đến chất lượng của giải pháp và các kỹ thuật như khởi tạo ngẫu nhiên và khởi động lại nhiều lần có thể được sử dụng để giảm thiểu vấn đề này.

1.2 Stochastic Gradient Descent (SGD)

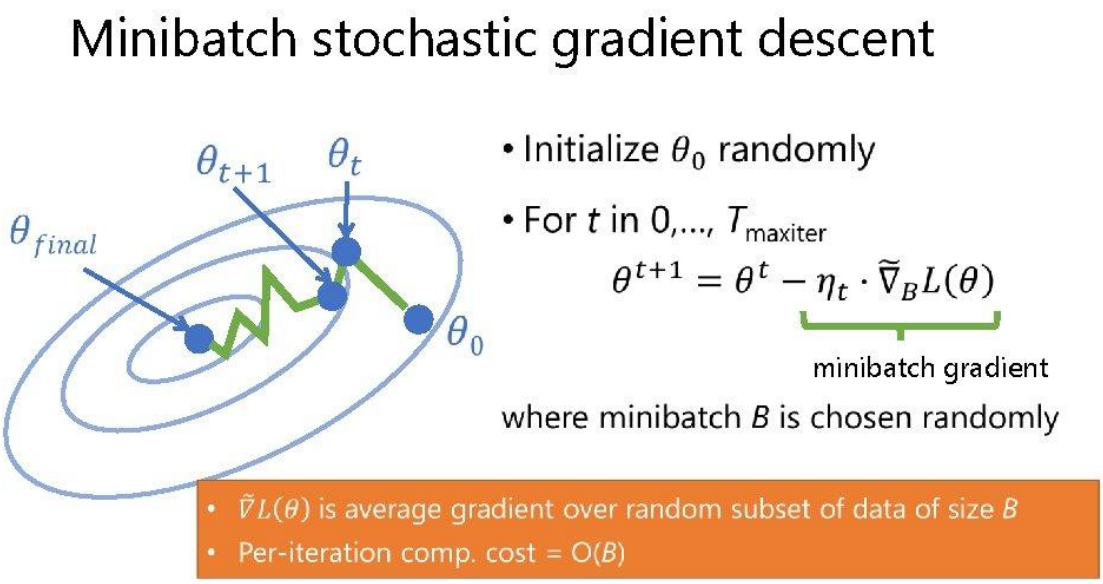
- Stochastic Gradient Descent (SGD) là một biến thể của thuật toán Gradient Descent(GD) được sử dụng để tối ưu hóa các mô hình học máy. Nó giải quyết sự thiếu hiệu quả tính toán của các phương pháp GD truyền thống khi xử lý các tập dữ liệu lớn trong các dự án học máy.
- Trong SGD, thay vì sử dụng toàn bộ tập dữ liệu cho mỗi lần lặp, chỉ một mẫu huấn luyện ngẫu nhiên duy nhất (hoặc một lô nhỏ) được chọn để tính gradient và cập nhật các tham số mô hình. Lựa chọn ngẫu nhiên này đưa tính ngẫu nhiên vào quá trình tối ưu hóa, do đó có thuật ngữ "ngẫu nhiên" trong Giảm dần độ dốc ngẫu nhiên. Ưu điểm của việc sử dụng SGD là hiệu quả tính toán, đặc biệt khi xử lý các tập dữ liệu lớn. Bằng cách sử dụng một ví dụ duy nhất hoặc một lô nhỏ, chi phí tính toán cho mỗi lần lặp lại giảm đáng kể so với các phương pháp Giảm dần độ dốc truyền thống yêu cầu xử lý toàn bộ tập dữ liệu.
- Trong SGD, chỉ một mẫu huấn luyện được sử dụng để tính gradient và cập nhật các tham số ở mỗi lần lặp. Điều này có thể nhanh hơn so với việc giảm độ dốc hàng loạt nhưng có thể dẫn đến nhiều tiếng ồn hơn trong các bản cập nhật.
- Ưu điểm của phương pháp Stochastic Gradient Descent (SGD):
  + Tốc độ: SGD nhanh hơn các biến thể khác của Giảm dần độ dốc hàng loạt và Giảm dần độ dốc hàng loạt vì nó chỉ sử dụng một ví dụ để cập nhật các tham số.
  + Hiệu quả bộ nhớ: Vì SGD cập nhật từng tham số cho từng ví dụ huấn luyện nên nó tiết kiệm bộ nhớ và có thể xử lý các tập dữ liệu lớn không vừa bộ nhớ.
  + Tránh cực tiểu cục bộ: Do các cập nhật ồn ào trong SGD, nó có khả năng thoát khỏi cực tiểu cục bộ và hội tụ đến mức tối thiểu toàn cầu.
- Nhược điểm của phương pháp Stochastic Gradient Descent (SGD):
  + Cập nhật ồn ào: Các cập nhật trong SGD rất ồn ào và có phương sai cao, điều này có thể làm cho quá trình tối ưu hóa kém ổn định hơn và dẫn đến dao động quanh mức tối thiểu.
  + Hội tụ chậm: SGD có thể yêu cầu nhiều lần lặp hơn để hội tụ đến mức tối thiểu vì nó cập nhật từng tham số cho từng ví dụ huấn luyện tại một thời điểm.
  + Độ nhạy đối với tốc độ học: Việc lựa chọn tốc độ học có thể rất quan trọng trong SGD vì việc sử dụng tốc độ học cao có thể khiến thuật toán vượt quá mức tối thiểu, trong khi tốc độ học thấp có thể khiến thuật toán hội tụ chậm.
  + Kém chính xác hơn: Do cập nhật ồn ào, SGD có thể không hội tụ đến mức tối thiểu toàn cục chính xác và có thể dẫn đến giải pháp dưới mức tối ưu. Điều này có thể được giảm thiểu bằng cách sử dụng các kỹ thuật như lập kế hoạch tốc độ học tập và cập nhật dựa trên động lượng

1.3 Mini-batch Gradient Descent:

- Mini-batch Gradient Descent là một phương pháp tối ưu hóa thường được sử dụng trong việc huấn luyện mô hình học máy. Đây là sự kết hợp giữa

Gradient Descent (GD) và Stochastic Gradient Descent (SGD).

- Tức là các tập con nhỏ hơn của dữ liệu trong Mini-batch Gradient Descent, huấn luyện được chia thành các mini-batch. Mini-batch GD cập nhật trọng số sau mỗi lượt duyệt toàn bộ dữ liệu như trong GD, mỗi mẫu như trong SGD hoặc sau mỗi lượt duyệt toàn bộ dữ liệu như trong GD.
- Trong quy trình giảm độ dốc hàng loạt nhỏ, một loạt ví dụ huấn luyện nhỏ được sử dụng để tính toán độ dốc và cập nhật các tham số ở mỗi lần lặp. Đây có thể là sự kết hợp tốt giữa giảm dần độ dốc hàng loạt và Giảm dần độ dốc ngẫu nhiên, vì nó có thể nhanh hơn so với giảm độ dốc hàng loạt và ít nhiễu hơn so với Giảm dần độ dốc ngẫu nhiên.



- Mini-batch GD tính gradients dựa trên 1 tập nhỏ ngẫu nhiên các điểm dữ liệu được gọi là mini- batches
  - Ưu điểm của phương pháp Mini-batch GD:
    + Hiệu suất cao hơn: Thường nhanh hơn so với GD vì nó sử dụng thông tin gradient từ một lượng dữ liệu nhỏ hơn.
    + Ước lượng gradient ổn định hơn: Đối với một số bài toán, ước lượng gradient từ một mini-batch có thể ổn định hơn so với SGD, giúp việc di chuyển đến điểm cực tiểu mất mát tốt hơn.
  - Nhược điểm của phương pháp Mini-batch GD:
    + Lựa chọn kích thước mini-batch: Việc chọn kích thước mini-batch phù hợp có thể ảnh hưởng đến tốc độ học và sự ổn định của việc hội tụ.
    + Cần quản lý bộ nhớ: Kích thước mini-batch lớn hơn có thể đòi hỏi nhiều bộ nhớ hơn.

1.4 RMSprop:

- Tối ưu hóa Adam là một phương pháp giảm độ dốc ngẫu nhiên dựa trên ước tính thích ứng của các khoảnh khắc bậc nhất và bậc hai.
- Theo Kingma và cộng sự, năm 2014, phương pháp này "hiệu quả về mặt tính toán, yêu cầu ít bộ nhớ, không thay đổi tỷ lệ theo đường chéo của độ dốc và rất phù hợp cho các vấn đề lớn về dữ liệu/tham số".
- Trong biến thể này, tốc độ học được điều chỉnh thích ứng cho từng tham số dựa trên đường trung bình động của gradient bình phương. Điều này giúp

thuật toán hội tụ nhanh hơn khi có độ dốc nhiễu.

- Thuật toán RMSprop sử dụng các đường trung bình động có trọng số theo cấp số nhân của gradient bình phương để cập nhật các tham số. Đây là phương trình toán học cho RMSprop:
- Khởi tạo các tham số:
- Tỷ lệ học tập: α
- Tốc độ phân rã theo cấp số nhân để tính trung bình: γ
- Hằng số nhỏ cho sự ổn định số: ε
- Giá trị tham số ban đầu: θ
- Khởi tạo độ dốc tích lũy (Trung bình có trọng số theo cấp số nhân):
- Độ dốc bình phương tích lũy cho từng tham số: Et= 0
- Lặp lại cho đến khi hội tụ hoặc lặp lại tối đa.
- Ưu điểm của phương pháp RMSprop:
    + Hiệu suất ổn định: RMSprop giúp ổn định quá trình huấn luyện bằng cách điều chỉnh tỷ lệ học tập (learning rate) dựa trên độ lớn của gradient gần đây. Điều này có thể giúp việc hội tụ đến điểm cực tiểu mất mát tốt hơn.
    + Xử lý vấn đề của tỷ lệ học tập không đồng nhất: Đôi khi, tỷ lệ học tập không đồng nhất trên các tham số có thể gây ra vấn đề trong việc huấn luyện. RMSprop giúp giảm thiểu vấn đề này bằng cách điều chỉnh tỷ lệ học tập theo cách phản ánh độ lớn của gradient.
- Nhược điểm của phương pháp RMSprop:
    + Chọn tham số: Cần điều chỉnh các siêu tham số của RMSprop như hệ số quan trọng (giảm cân) hoặc tham số beta để đảm bảo hiệu suất tốt nhất cho mô hình.
    + Khả năng đồng bộ hóa không tốt: RMSprop có thể không đồng bộ hóa tốt trong một số trường hợp, đặc biệt là khi áp dụng cho các mô hình phức tạp hoặc dữ liệu không đồng nhất.

1.5 Adagrad:

- Adagrad là một trình tối ưu hóa với tốc độ học theo tham số cụ thể, được điều chỉnh tương ứng với tần suất cập nhật của tham số trong quá trình đào tạo. Tham số càng nhận được nhiều cập nhật thì số lượng cập nhật càng nhỏ.
- Trong biến thể này, tốc độ học được điều chỉnh thích ứng cho từng tham số dựa trên thông tin độ dốc lịch sử. Điều này cho phép cập nhật lớn hơn cho các tham số không thường xuyên và cập nhật nhỏ hơn cho các tham số thường xuyên.
- Adagrad là viết tắt của Trình tối ưu hóa độ dốc thích ứng. Có các trình tối ưu hóa như Giảm dần độ dốc, Giảm độ dốc ngẫu nhiên, SGD lô nhỏ, tất cả đều được sử dụng để giảm hàm mất mát liên quan đến trọng số. Công thức cập nhật trọng số như sau:

$$(\mathrm{w})_{\mathrm{new}} = (\mathrm{w})_{\mathrm{old}} - \eta \frac{\partial \mathrm{L}}{\mathrm{dw}(\ \mathrm{old}\ )}$$

Dựa trên các lần lặp, công thức này có thể được viết là:

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w(t-1)}$$

w(t) = giá trị của w ở lần lặp hiện tại, w(t-1) = giá trị của w ở lần lặp trước đó và η = tốc độ học tập.

- Ưu điểm của phương pháp Adagrad:
  + Tỷ lệ điều chỉnh tự động học tập (tốc độ học tập): Adagrad điều chỉnh tỷ lệ học tập cho từng tham số dựa trên tần suất xuất hiện của gradient. Điều này có ý nghĩa là các tham số với độ dốc lớn sẽ có tỷ lệ học tập giảm và lùi, giúp ổn định quá trình hội tụ.
  + Phù hợp với dữ liệu thưa thớt: Adagrad hoạt động tốt trên các dữ liệu thưa thớt (dữ liệu thưa thớt) vì nó thích ứng với tốc độ xuất hiện của các đặc thù.
- Nhược điểm của phương pháp Adagrad:
  + Tính chất giảm dần của tốc độ học tập: Một vấn đề của Adagrad là tốc độ học tập giảm dần theo thời gian. Điều này có thể dẫn đến tốc độ học tập trở nên quá nhỏ sau một số lượt huấn luyện viên, làm chậm quá trình hội tụ.
  + Khả năng vượt quá mục tiêu (vượt quá mục tiêu): Trong một số trường hợp, Adagrad có thể quá tối ưu hóa các tham số có độ dốc lớn, dẫn đến việc vượt quá mục tiêu và không thể hội tụ đến điểm cực tiểu mong muốn.
  + Quản lý bộ nhớ: Adagrad yêu cầu lưu trữ lịch sử gradient cho từng tham số, dẫn đến bộ nhớ tốn kém khi làm việc với số lượng tham số lớn.

1.6 Adam:

- Tối ưu hóa Adam là một phương pháp giảm độ dốc ngẫu nhiên dựa trên ước tính thích ứng của các khoảnh khắc bậc nhất và bậc hai.
- Theo Kingma và cộng sự, năm 2014, phương pháp này "hiệu quả về mặt tính toán, yêu cầu ít bộ nhớ, không thay đổi tỷ lệ theo đường chéo của độ dốc và rất phù hợp cho các vấn đề lớn về dữ liệu/tham số".
- Adam là viết tắt của ước tính mô men thích ứng, nó kết hợp các lợi ích của Độ dốc gốc dựa trên Động lượng, Adagrad và RMSprop. Tốc độ học được điều chỉnh thích ứng cho từng tham số dựa trên mức trung bình di chuyển của độ dốc và độ dốc bình phương, cho phép hội tụ nhanh hơn và thực hiện tốt hơn các bài toán tối ưu không lồi. Nó theo dõi hai mức trung bình giảm dần theo cấp số nhân, ước tính thời điểm đầu tiên, là mức trung bình giảm dần theo cấp số nhân của độ dốc trong quá khứ và ước tính thời điểm thứ hai, là mức trung bình giảm dần theo cấp số nhân của độ dốc bình phương trong quá khứ. Ước tính khoảnh khắc đầu tiên được sử dụng để tính toán động lượng và ước tính khoảnh khắc thứ hai được sử dụng để chia tỷ lệ tốc độ học cho từng tham số. Đây là một trong những thuật toán tối ưu hóa phổ biến nhất cho deep learning.
- Ưu điểm của phương pháp Adam:
  + Hiệu suất cao: Adam thường hoạt động tốt trên nhiều loại mô hình và các dữ liệu khác nhau mà không yêu cầu nhiều siêu tham số cần điều

chỉnh.

+ Tự động điều chỉnh tỷ lệ học tập (tốc độ học tập): Adam tự động điều chỉnh tỷ lệ học tập theo từng tham số dựa trên độ dốc lớn và độ trung bình của độ dốc. Điều này có thể giúp tiến trình chuỗi nhanh và ổn định.

+ Hiệu quả với dữ liệu lớn và thưa thớt: Adam thường xuyên hoạt động tốt trên dữ liệu lớn và có khả năng tự điều chỉnh tỷ lệ học tập dựa trên tần suất xuất hiện của các đặc thù.

- Nhược điểm của phương pháp Adam:

+ Yêu cầu bộ nhớ: Adam cần lưu trữ bổ sung các thông tin như gradient bậc hai và thứ hai cho từng tham số, dẫn đến bộ nhớ tiêu tốn nhiều hơn so với một số phương pháp tối ưu hóa khác.

+ Khả năng vượt mức: Trong một số trường hợp, Adam có thể vượt quá tiêu cực tiểu và không hội tụ một ổn định.

+ Yêu cầu điều chỉnh siêu tham số: Mặc dù Adam tự điều chỉnh tỷ lệ học tập, nhưng việc điều chỉnh các siêu tham số khác như beta1, beta2 vẫn cần thiết để đạt được hiệu suất tốt nhất trên mỗi bài toán cụ thể.

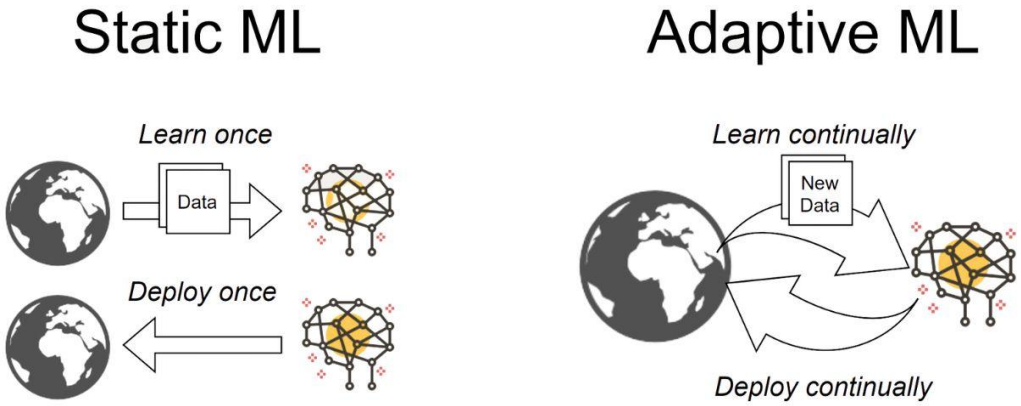
Câu 2:

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.

2.1Continual Learning

2.1.1 Khái niệm:

- Continual Learning (CL) tập trung vào việc phát triển các mô hình để học các nhiệm vụ mới trong khi vẫn giữ lại thông tin từ các nhiệm vụ trước đó. CL là một lĩnh vực nghiên cứu quan trọng vì nó giải quyết tình huống thực tế trong đó dữ liệu và nhiệm vụ liên tục thay đổi và mô hình phải thích ứng với những thay đổi này mà không quên kiến thức trước đó.

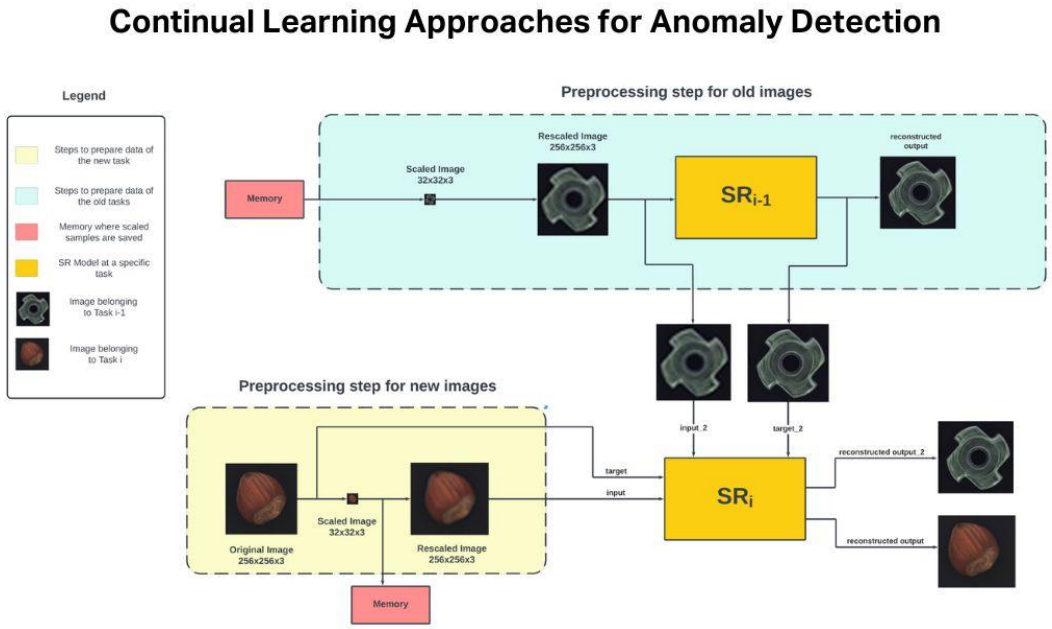


- Trong học máy truyền thống, một mô hình được đào tạo trên một tập dữ liệu cố định và dự kiến sẽ thực hiện một nhiệm vụ duy nhất. Tuy nhiên, cách tiếp cận này trở nên có vấn đề khi dữ liệu và nhiệm vụ thay đổi và linh hoạt, vì mô hình phải có khả năng

thích ứng và học hỏi từ dữ liệu mới theo thời gian. Đây là lúc CL phát huy tác dụng, cho phép mô hình liên tục học hỏi và cải thiện mà không quên kiến thức trước đó.

- Một trong những thách thức đáng kể trong CL là vấn đề quên thảm khốc, trong đó một mô hình được đào tạo về nhiều nhiệm vụ cần ghi nhớ thông tin đã học được từ các nhiệm vụ trước đó khi tiếp xúc với dữ liệu mới. Nhiều kỹ thuật khác nhau đã được phát triển để vượt qua thách thức này, bao gồm cả mạng chính quy hóa và mạng tăng cường bộ nhớ.
- Tóm lại, CL là một lĩnh vực quan trọng trong học máy nhằm giải quyết thách thức của các mô hình đào tạo có thể liên tục học hỏi và thích ứng với các nhiệm vụ và dữ liệu mới mà không quên kiến thức trước đó. Việc sử dụng các kỹ thuật như mạng chính quy hóa và tăng cường bộ nhớ, cũng như thiết kế kiến trúc mô hình, đóng một vai trò quan trọng trong khả năng thích ứng của các mô hình CL.

2.1.2 Trường hợp sử dụng:



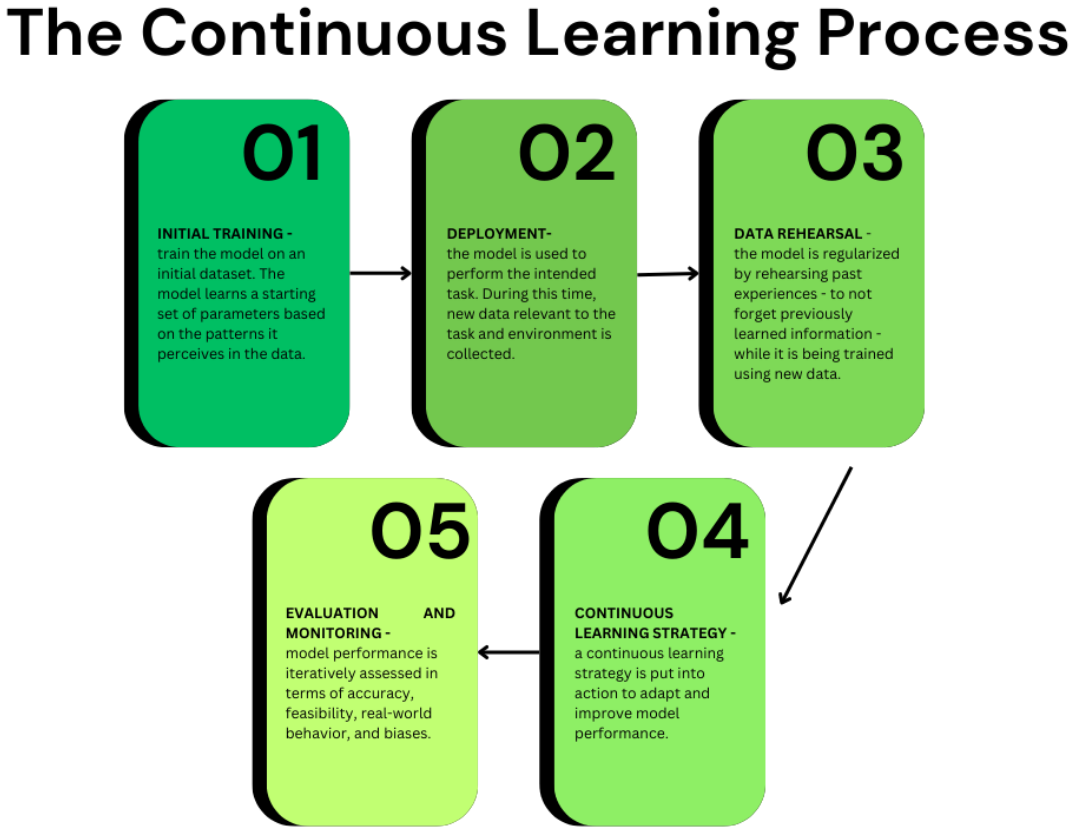
- Phát hiện bất thường – CL có thể đặc biệt hữu ích trong các tình huống phát hiện bất thường trong đó việc phân phối dữ liệu thay đổi theo thời gian và các thuật toán học máy truyền thống có thể không hiệu quả. Mục tiêu của kiểu học liên tục này là liên tục theo dõi hành vi bình thường của hệ thống để tìm hiểu các cấp độ vận hành, quy trình hoặc luồng dữ liệu tiêu chuẩn của nó và phát hiện các điểm bất thường bằng cách so sánh dữ liệu mới với hành vi bình thường đã học. Để thực hiện điều này, thuật toán học liên tục được huấn luyện trên luồng dữ liệu để tìm hiểu hành vi bình thường của hệ thống. Khi có dữ liệu mới, dữ liệu đó sẽ được so sánh với hành vi bình thường đã học được và những sai lệch so với tiêu chuẩn sẽ được gắn cờ là bất thường. Thuật toán học liên tục sau đó có thể được cập nhật để kết hợp dữ liệu mới, cho phép nó thích ứng với những thay đổi trong hành vi thông thường theo thời gian.
- Cá nhân hóa – Một trường hợp sử dụng khác của CL là trong các hệ thống đề xuất được cá nhân hóa với mục tiêu là cung cấp các đề xuất cập nhật và có tính tùy chỉnh cao cho người dùng. Bằng cách liên tục tìm hiểu sở thích và hành vi của người dùng, hệ thống đề xuất có thể cải thiện khả năng đưa ra đề xuất chính xác và phù hợp. Để đạt được điều này, thuật toán học liên tục được đào tạo dựa trên dữ liệu tương tác

lịch sử của người dùng, chẳng hạn như lịch sử mua hàng hoặc truy vấn tìm kiếm, để tìm hiểu sở thích và kiểu hành vi của họ. Khi có dữ liệu mới, mô hình sẽ cập nhật hiểu biết về sở thích và hành vi của người dùng. Thông tin cập nhật này sau đó được sử dụng để đưa ra các khuyến nghị chính xác và phù hợp hơn cho người dùng.

- Dự báo – CL cũng có thể được áp dụng trong lĩnh vực dự báo để liên tục cập nhật các dự đoán dựa trên dữ liệu mới khi có sẵn. Cách tiếp cận này cho phép các mô hình thích ứng với những thay đổi trong phân phối dữ liệu và duy trì độ chính xác của chúng theo thời gian, ngay cả khi xử lý dữ liệu phức tạp và động. Trong hệ thống dự báo, thuật toán học liên tục được huấn luyện trên luồng dữ liệu lịch sử, chẳng hạn như dữ liệu tài chính hoặc dữ liệu chuỗi thời gian, để đưa ra dự đoán về các sự kiện trong tương lai. Khi có dữ liệu mới, mô hình sẽ cập nhật hiểu biết của nó về các mối quan hệ trong dữ liệu và sử dụng thông tin này để tinh chỉnh các dự đoán của nó.

2.1.3 Quá trình của Continual Learning (CL):

- CL là một sự phát triển của mô hình học máy truyền thống. Do đó, nó liên quan đến nhiều nguyên tắc lập mô hình giống nhau: tiền xử lý, lựa chọn mô hình, điều chỉnh siêu tham số, đào tạo, triển khai và giám sát.
- Hai bước bổ sung cần thiết trong quá trình CL: diễn tập dữ liệu và thực hiện chiến lược học tập liên tục. Các bước này nhằm đảm bảo rằng mô hình đang học từ các luồng dữ liệu mới một cách hiệu quả dựa trên ứng dụng và bối cảnh của nhiệm vụ dữ liệu.



2.1.4 Ưu điểm của Continual Learning (CL):

- Tiếp cận dữ liệu mới: Continual Learning cho phép mô hình học máy tiếp tục học từ dữ liệu mới, giúp cải thiện dần đề xuất hoặc dự đoán dựa trên thông tin mới.
- Tính linh hoạt: Có thể thích ứng với môi trường thay đổi, dữ liệu mới, và thậm chí là các bài toán khác nhau mà không cần đào tạo lại từ đầu.
- Tiết kiệm tài nguyên: Bằng cách tận dụng lại kiến thức đã học, Continual Learning có thể giảm thiểu việc phải đào tạo lại mô hình hoàn chỉnh từ đầu khi có dữ liệu mới.

2.1.5 Nhược điểm của Continual Learning (CL):

- Lãng phí thông tin cũ: Có thể xảy ra hiện tượng quên mất thông tin cũ khi mô hình học từ dữ liệu mới, dẫn đến việc giảm hiệu suất hoặc độ chính xác của mô hình đối với dữ liệu cũ.
- Hiện tượng "catastrophic forgetting": Khi tập trung vào học từ dữ liệu mới, mô hình có thể quên đi kiến thức đã học trước đó, dẫn đến hiện tượng quên mất nhanh chóng các kỹ năng hoặc dự đoán cũ.
- Khó khăn trong việc ổn định mô hình: Continual Learning yêu cầu sự cân nhắc cẩn thận giữa việc học mới và việc bảo tồn kiến thức cũ để đảm bảo mô hình không bị biến dạng hoặc quá mức tối ưu hóa cho dữ liệu mới.

2.2 Test Production

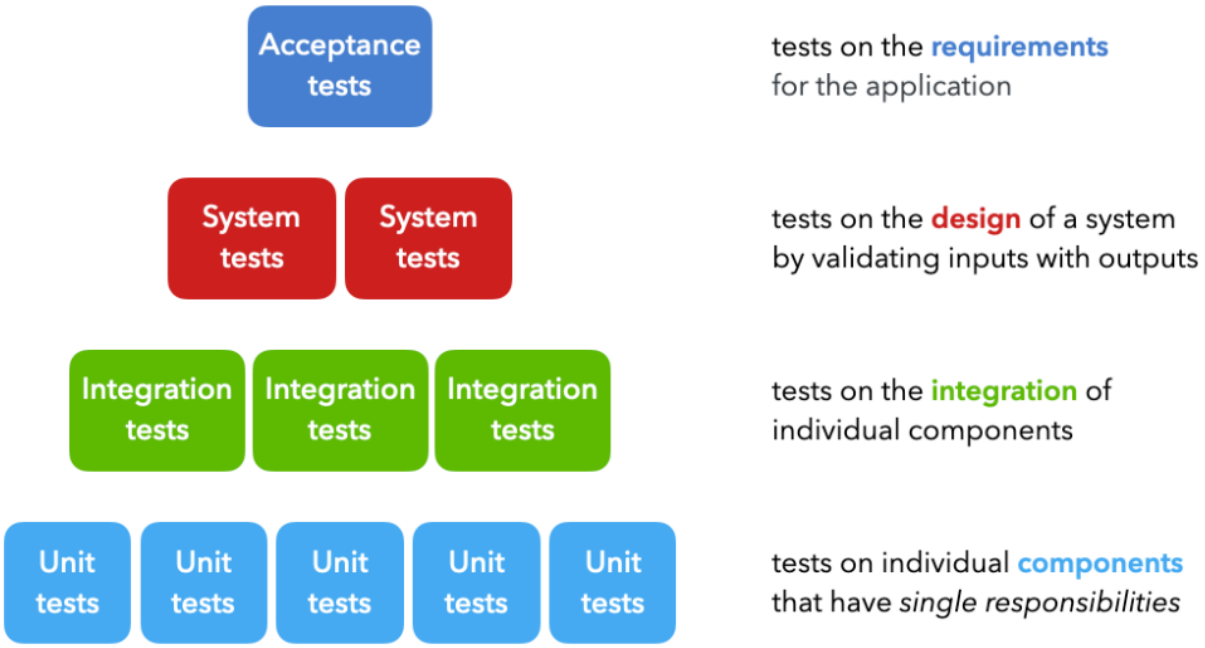
2.2.1 Khái Niệm

- Test Production là giai đoạn kiểm thử mô hình AI với dữ liệu thực tế trước khi triển khai. Mục đích là đánh giá độ chính xác, hiệu suất và ổn định của mô hình trong điều kiện thực tế.
- Các bước chính trong Test Production: thu thập dữ liệu thực tế, xây dựng tập test, chạy thử và đánh giá mô hình, xác định và khắc phục sự cố, lặp lại quy trình cho đến khi đạt yêu cầu.
- Một số kỹ thuật phổ biến trong Test Production: A/B testing, shadow mode, canary deployment.

2.2.2 Các dạng test

- Có bốn loại thử nghiệm chính được sử dụng ở các thời điểm khác nhau trong chu kỳ phát triển:
    1. **Kiểm tra đơn vị:** kiểm tra trên các thành phần riêng lẻ mà mỗi thành phần có một trách nhiệm duy nhất (ví dụ: chức năng lọc danh sách).
    2. **Kiểm tra tích hợp**: kiểm tra chức năng kết hợp của các thành phần riêng lẻ (ví dụ: xử lý dữ liệu).
    3. **Kiểm tra hệ thống**: kiểm tra thiết kế của hệ thống về kết quả đầu ra dự kiến dựa trên đầu vào (ví dụ: đào tạo, suy luận, v.v.).
    4. **Kiểm tra chấp nhận**: kiểm tra để xác minh rằng các yêu cầu đã được đáp ứng, thường được gọi là Kiểm tra chấp nhận của người dùng (UAT).
    5. **Kiểm tra hồi quy**: kiểm tra dựa trên các lỗi mà chúng tôi đã thấy trước đây để đảm bảo các thay đổi mới không tái hiện các lỗi đó.

=> Mặc dù các hệ thống ML có bản chất xác suất nhưng chúng bao gồm nhiều thành phần xác định có thể được kiểm tra theo cách tương tự như các hệ thống phần mềm truyền thống. Sự khác biệt giữa các hệ thống ML thử nghiệm bắt đầu khi chúng tôi chuyển từ thử nghiệm mã sang thử nghiệm dữ liệu và mô hình.



2.2.3 Chúng ta nên thử nghiệm như thế nào?

Khung để sử dụng khi soạn bài kiểm tra là phương pháp Xác nhận hành động sắp xếp.

- Sắp xếp: thiết lập các đầu vào khác nhau để kiểm tra.
- Hành động: áp dụng các đầu vào trên thành phần mà chúng tôi muốn kiểm tra.
- Khẳng định: xác nhận rằng chúng tôi đã nhận được kết quả mong đợi.

Trong Python, có nhiều công cụ, chẳng hạn như unittest, pytest, v.v. cho phép chúng ta dễ dàng thực hiện các thử nghiệm của mình trong khi tuân thủ khung Xác nhận Hành động Sắp xếp. Những công cụ này đi kèm với chức năng tích hợp mạnh mẽ như tham số hóa, bộ lọc, v.v. để kiểm tra nhiều điều kiện trên quy mô lớn.

2.2.4 Chúng ta nên kiểm tra những gì?

Khi sắp xếp đầu vào và khẳng định kết quả đầu ra dự kiến, chúng ta nên thử nghiệm một số khía cạnh của đầu vào và đầu ra là gì?

- đầu vào: loại dữ liệu, định dạng, độ dài, trường hợp cạnh (tối thiểu/tối đa, nhỏ/lớn, v.v.)
- đầu ra: kiểu dữ liệu, định dạng, ngoại lệ, đầu ra trung gian và cuối cùng

2.2.5 Thực hành.

Bất kể chúng ta sử dụng framework nào, điều quan trọng là phải gắn chặt việc thử nghiệm vào quá trình phát triển.

- Nguyên tử: khi tạo các hàm và lớp, chúng ta cần đảm bảo rằng chúng có một trách nhiệm duy nhất để chúng ta có thể dễ dàng kiểm tra chúng. Nếu không, chúng ta sẽ

cần chia chúng thành các thành phần chi tiết hơn.

- Soạn thảo: khi chúng tôi tạo các thành phần mới, chúng tôi muốn soạn các bài kiểm tra để xác thực chức năng của chúng. Đó là một cách tuyệt vời để đảm bảo độ tin cậy và phát hiện lỗi sớm.
- Tái sử dụng: chúng ta nên duy trì các kho lưu trữ trung tâm nơi chức năng cốt lõi được kiểm tra tại nguồn và được sử dụng lại trong nhiều dự án. Điều này làm giảm đáng kể nỗ lực thử nghiệm đối với cơ sở mã của từng dự án mới.
- Hồi quy: chúng tôi muốn tính đến các lỗi mới mà chúng tôi gặp phải khi kiểm tra hồi quy để có thể đảm bảo rằng chúng tôi không tái phạm các lỗi tương tự trong tương lai.
- Phạm vi bảo hiểm: chúng tôi muốn đảm bảo phạm vi bao phủ 100% cho cơ sở mã của chúng tôi. Điều này không có nghĩa là viết bài kiểm tra cho từng dòng mã mà là tính toán từng dòng một.
- Tự động hóa: trong trường hợp chúng tôi quên chạy thử nghiệm trước khi đưa vào kho lưu trữ, chúng tôi muốn tự động chạy thử nghiệm khi thực hiện thay đổi đối với cơ sở mã của mình. Chúng ta sẽ tìm hiểu cách thực hiện việc này cục bộ bằng cách sử dụng pre-commit hook và từ xa thông qua các hành động GitHub trong các bài học tiếp theo.

REFERENCE

1. https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/amp/
2. https://www.geeksforgeeks.org/intuition-behind-adagrad-optimizer/amp/
3. https://en.m.wikipedia.org/wiki/Stochastic_gradient_descent
4. https://wiki.continualai.org/the-continualai-wiki/introduction-to-continual-learning
5. https://madewithml.com/courses/mlops/testing/#types-of-tests