TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**NGUYỄN TƯỜNG AN - 52100380**

**LƯU MINH NHIỀU - 52100456**

**BÁO CÁO CUỐI KÌ**

**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**



**NGUYỄN TƯỜNG AN - 52100380**

**LƯU MINH NHIỀU - 52100456**

**BÁO CÁO CUỐI KÌ**

**NHẬP MÔN HỌC MÁY**

Người hướng dẫn

**GV. Lê Anh Cường**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

**LỜI CẢM ƠN**

Trong quá trình học môn "Nhập môn Học máy," chúng em may mắn được hướng dẫn và hỗ trợ nhiệt tình của Thầy Lê Anh Cường. Chúng em xin gửi lời cảm ơn sâu sắc và tri ân đến Thầy vì những kiến thức quý báu và kỹ năng quan trọng mà Thầy đã chia sẻ. Thầy Lê Anh Cường không chỉ là một người hướng dẫn giáo viên xuất sắc mà còn là người truyền đạt những kiến thức phức tạp một cách dễ hiểu và hấp dẫn. Những bài giảng, bài thực hành và hướng dẫn của Thầy đã giúp chúng em có cái nhìn tổng quan về lĩnh vực học máy và ứng dụng chúng vào thực tế. Chúng em không thể không đề cập đến sự kiên nhẫn và tận tâm của Thầy trong việc giải đáp mọi thắc mắc của chúng em. Nhờ sự hỗ trợ và sự giúp đỡ của Thầy, chúng em đã có cơ hội vận dụng những kiến thức mới nhất vào bài tập và dự án của mình. Cuối cùng, chúng tôi xin bày tỏ lòng biết ơn chân thành đến Thầy Lê Anh Cường, người đã giúp định hình cho chúng tôi một cách học máy và góp phần tạo nên hành trang kiến thức quan trọng cho hành trình sắp tới. Chúng tôi chân thành cảm ơn!

*TP. Hồ Chí Minh, ngày 16 tháng 12 năm 2023*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Lưu Minh Nhiều*

*Nguyễn Tường An*

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của GV. Lê Anh Cường. 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 Dự án 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 Dự án 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 16 tháng 12 năm 2023*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Lưu Minh Nhiều*

*Nguyễn Tường An*

**TÓM TẮT**

(Time New Romans – 13)

**ABSTRACT**

(Time New Romans – 13)

**MỤC LỤC**

[DANH MỤC HÌNH VẼ vii](#_Toc17902)

[DANH MỤC BẢNG BIỂU viii](#_Toc25102)

[DANH MỤC CÁC CHỮ VIẾT TẮT ix](#_Toc17619)

[CHƯƠNG 1. CÁC PHƯƠNG PHÁP OPTIMIZER 1](#_Toc17220)

[1.1 Tổng quan 1](#_Toc12065)

[1.2 Các loại Optimizer: 2](#_Toc25515)

[1.2.1 Gradient Descent (GD): 2](#_Toc5276)

[1.2.2 Adaptive Gradient Algorithm (Adagrad): 3](#_Toc12625)

[1.2.3 Root Mean Square Propagation (RMSprop): 4](#_Toc15111)

[1.2.4 Adaptive Moment Estimation (Adam): 5](#_Toc29461)

[1.2.5 Adadelta: 6](#_Toc11594)

[1.2.6 Nesterov-accelerated Adaptive Moment Estimation (Nadam): 7](#_Toc29113)

[1.2.7 FTRL-Proximal: 8](#_Toc15539)

[1.2.8 Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS): 9](#_Toc32417)

[1.2.9 So sánh các phương pháp Optimizer: 9](#_Toc19768)

[CHƯƠNG 2. CONTINUAL LEARNING 12](#_Toc28032)

[2.1 Tổng quan 12](#_Toc19694)

[2.2 Mục tiêu của Continual Learning 12](#_Toc133)

[2.3 Stateless retraining và Stateful retraining: 13](#_Toc30733)

[2.3.1 Stateless retraining: 13](#_Toc703)

[2.3.2 Stateful retraining ( fine-tuning, incremental learning): 13](#_Toc22071)

[2.4 Log and Wait: 14](#_Toc27348)

[2.4.1 Khái niệm: 14](#_Toc737)

[2.4.2 Mục đích sử dụng: 14](#_Toc28735)

[2.5 Thách thức của Continual Learning: 15](#_Toc8712)

[2.6 Các giai đoạn của một Continual Learning: 15](#_Toc4016)

[2.7 Khi nào thì nên cập nhật lại mô hình? 17](#_Toc11169)

[CHƯƠNG 3. TEST PRODUCTION 18](#_Toc8006)

[3.1 Tổng quan: 18](#_Toc19291)

[3.2 Phương pháp đánh giá: 18](#_Toc22971)

[3.2.1 Model Offline Evaluation (Hay đánh giá trước khi triển khai): 18](#_Toc16308)

[3.2.2 Testing in Production Strategies (Hay đánh giá trong triển khai): 20](#_Toc16305)

[TÀI LIỆU THAM KHẢO 28](#_Toc14957)

# DANH MỤC HÌNH VẼ

[Hình 2.1: Scaled Dot-Product Attention 3](#_Toc142677545)

# DANH MỤC BẢNG BIỂU

[Bảng 4.1: Thống kê kiểu thực thể trong tập VLSP 2016 4](#_Toc142677565)

# DANH MỤC CÁC CHỮ VIẾT TẮT

|  |  |
| --- | --- |
| BERT | Bidirectional Encoder Representations from Transformers |
| GEC | Grammatical Error Correction |
| MLM | Masked Language Model |
| NLP | Natural Language Processing |
| NSP | Next Sentence Prediction |

# CÁC PHƯƠNG PHÁP OPTIMIZER

## Tổng quan

Optimizer (Thuật toán tối ưu): thường dùng để tìm giá nhỏ nhất (đôi khi là lớn nhất) của một hàm số nào đó. Ví dụ như Loss Function trong Linear Regression. Về cơ bản, thuật toán tối ưu là thuật toán quan trọng, cũng là cơ sở trong mô hình Neural Network với mục đích học được các Features (hay các pattern) của dữ liệu đầu vào. Hay dễ hiểu hơn là tìm cặp hệ số weight, bias tối ưu nhất cho bài toán (model).

Bài toán tối ưu dạng tổng quát:

Thoả mãn 2 điều kiện:

Trong đó:

là biến tối ưu (Optimizer variable)

là hàm mục tiêu (Objective/cost/lost function)

ràng buộc dạng bất đẳng thức (inequality constraint function). Trong đó, i = 1, 2, … m

ràng buộc dạng đẳng thức (equality constraint function). Trong đó, j = 1, 2, … p

Tập hợp tất cả các điểm mà các hàm mục tiêu và D: tập xác định (domain).

Hay tổng quát dưới dạng lời văn: Tìm giá trị x để tối thiểu hàm mục tiêu thoả mãn hệ điều kiện ràng buộc bất đẳng thức và đẳng thức.

Một điểm x thuộc tập xác định (D) nếu nó thỏa mãn ràng buộc và bài toán tối ưu được gọi là khả thi (feasible) nếu tồn tại ít nhất 1 điểm x. Ngược lại, nếu không tồn tại bất kì điểm nào ta gọi nó là không khả thi (infeasible). Tập hợp tất cả các điểm feasible (feasible point) gọi là feasible set hay constraint set.

Nghiệm tối ưu (optimal value) p của bài toán còn được định nghĩa theo infimum là:

Infimum (viết tắt là "inf") là giới hạn dưới lớn nhất của một tập hợp , được định nghĩa là một giá trị sao cho không có phần tử nào của tập hợp nhỏ hơn , nhưng nếu là bất kì giá trị dương nào, dù cho nhỏ đến đâu, luôn tồn tại một thành phần của tập hợp nhỏ hơn (Jeffreys and Jeffreys 1988). Khi nó tồn tại (điều này không được yêu cầu bởi định nghĩa này, ví dụ là không tồn tại). Infimum được ký hiệu là hoặc . Infimum được triển khai trong Ngôn ngữ Wolfram dưới dạng MinValue[f, constr, vars].

## Các loại Optimizer:

### Gradient Descent (GD):

Gradient Descent là một trong những phương pháp tối ưu hóa thường được sử dụng để hiệu chỉnh trọng số của mô hình học máy sao cho hàm mất mát (Loss function/ Cost function) đạt giá trị nhỏ nhất. Gradient Descent bao gồm 3 biến thể:

- Stochastic Gradient (SGD): sẽ cập nhật trọng số qua mỗi 1 sample (dòng dữ liệu). Phương pháp này giúp giảm bớt bộ nhớ và tăng tốc quá trình đào tạo, đặc biệt là khi có một lượng lớn dữ liệu.

- Mini-batch Gradient Descent: tập dữ liệu sẽ được chia thành các mini-batches nhỏ, và mỗi lần cập nhật trọng số chỉ sử dụng 1 mini batch.

- Batch Gradient Descent (BGD): lấy toàn bộ dữ liệu (1 batch) để cập nhật các trọng số.

Công thức tổng quát:

Ưu điểm:

- Thuật toán dễ hiểu, cơ bản. Giải quyết được vấn đề tối ưu của Model Neural Network bằng cách cập nhật trọng số sau mỗi vòng lặp.

- Các biến thể của Gradient Descent giải quyết được vấn đề của chính bản thân nó mang lại như: có thể hoạt động tốt trên tập dữ liệu (bài toán) lớn.

- Dễ tối ưu hóa và hiệu chỉnh: Các biến thể của Gradient Descent có thể điều chỉnh siêu tham số (hyper parameter) để tối ưu hóa hiệu suất.

Nhược điểm:

- Phụ thuộc vào nghiệm khởi tạo và learning rate.

- Dễ rơi vào Local minimum, vì bài toán yêu cầu là tìm Global minimum.

- Bài toán càng lớn và phức tạp thì chi phí sẽ càng cao hơn so với các mô hình tối ưu khác.

- Không thực hiện được đối với bài toán không tính được đạo hàm.

Đặc biệt: **Momentum** là một kỹ thuật được sử dụng để giảm độ dao động (oscillation) và giúp quá trình hội tụ nhanh hơn trong thuật toán tối ưu hóa, đặc biệt là trong các phương pháp như Gradient Descent. Momentum giữ thông tin về hướng trước đó của các bước cập nhật và sử dụng nó để "giữ đà" cho quá trình cập nhật tiếp theo. Điều này giúp giảm bớt dao động và giúp thuật toán vượt qua các điểm nhỏ (local minima) trên đồ thị hàm mất mát. Hệ số momentum giúp giảm độ dao động của quá trình cập nhật, đặc biệt hiệu quả khi quá trình huấn luyện đi qua các vùng cong và lồi lõm. Các giá trị momentum từ các bước cập nhật trước giúp tăng tốc độ và ổn định cho bước cập nhật tiếp theo.

Ưu điểm của momentum: giải quyết được vấn đề gradient descen không tiến tới được điểm global minimum

Nhược điểm của momentum: Mất khá nhiều thời gian và tài nguyên để có thể tìm đến global minimum.

### Adaptive Gradient Algorithm (Adagrad):

Adaptive Gradient Algorithm là thuật toán cập nhật trọng số của mô hình dựa trên tần suất xuất hiện của các đặc trưng. Thuật toán này đặc trưng bởi việc sử dụng tốc độ học (learning rate) được điều chỉnh tự động cho từng tham số. Điều này là do nó sử dụng các tốc độ học khác nhau cho mỗi lần lặp. Sự thay đổi trong tốc độ học phụ thuộc vào sự khác biệt giữa các tham số trong quá trình huấn luyện. Càng nhiều tham số thay đổi, thì thay đổi tốc độ học càng nhỏ. Sửa đổi này rất hữu ích vì dữ liệu thực tế chứa các đặc trưng thưa thớt cũng như đặc trưng dày đặc. Do đó, việc có cùng một giá trị tốc độ học cho tất cả các đặc trưng là không công bằng.

Công thức:

Trong đó:

n là tốc độ học

là gradient tại thời điểm t

là hệ số tránh lỗi (do chia cho mẫu bằng 0)

là ma trận chéo mà mỗi phần tử trên đường chéo (i,i) là bình phương của đạo hàm vectơ tham số tại thời điểm t

Ưu điểm: loại bỏ nhu cầu điều chỉnh tốc độ học thủ công. Nó đáng tin cậy hơn so với các thuật toán gradient descent và các biến thể của chúng, và đạt được sự hội tụ ở tốc độ cao.

Nhược điểm: nó giảm tốc độ học mạnh mẽ và liên tục. Có thể có một điểm mà tốc độ học trở nên cực kỳ nhỏ. Điều này là do đạo hàm bình phương trong mẫu số tiếp tục tích tụ, và do đó, phần mẫu tiếp tục tăng lên. Do tốc độ học nhỏ, mô hình cuối cùng trở nên không thể tiếp thu thêm kiến thức, và do đó, độ chính xác của mô hình bị đánh giá thấp.

### Root Mean Square Propagation (RMSprop):

RMSprop là một thuật toán tối ưu hóa Gradient Descent chủ yếu được thiết kế để giảm vấn đề về tốc độ học (learning rate) trong quá trình huấn luyện mô hình học máy bằng cách chia tỷ lệ học cho trung bình của bình phương gradient.

Công thức:

Trong đó:

n là tốc độ học

là giá trị kỳ vọng của bình phương gradient tại thời điểm t

là gradient tại thời điểm t

là hệ số tránh lỗi (do chia cho mẫu bằng 0)

Ưu điểm: khả năng tự động điều chỉnh tốc độ học và giảm vấn đề của tốc độ học biến đổi đột ngột.

Nhược điểm: có thể dẫn đến việc giảm tốc độ học quá nhanh, và mô hình có thể hội tụ chưa đủ khi đến gần điểm tối ưu.

### Adaptive Moment Estimation (Adam):

Adaptive Moment Estimation (Adam) là một thuật toán tối ưu phổ biến nhất trong các thuật toán tối ưu trong deep learning và machine learning. Thuật toán được xây dựng dựa trên ý tưởng kết hợp 2 thuật toán optimizer khác là: momentum và RMSprop để cùng một lúc “giữ đà” để vượt qua các local minimum (momentum) và thay đổi tốc độ học (learning rate) dựa trên độ lớn của gradient để tiến về global minimum nhanh hơn (RMSprop).

Công thức:

Trong đó:

là tập hợp các tham số (weights)

n là tỷ lệ học (learning rate)

là trung bình động của gradient theo bậc hai

là trung bình động của gradient

là hệ số tránh lỗi (do chia cho mẫu bằng 0)

Ưu điểm:

- Adam là một thuật toán tối ưu hóa hiệu quả và linh hoạt nhờ cơ chế tự động điều chỉnh learning rate dựa trên trung bình động của gradient, giúp nó hoạt động tốt trong các trường hợp gradient biến động.

- Adam hoạt động tốt với lượng dữ liệu lớn và không yêu cầu quá nhiều bộ nhớ để lưu trữ toàn bộ tập dữ liệu.

Nhược điểm:

- Trong một số trường hợp, Adam có thể dẫn đến over-optimization, đặc biệt với dữ liệu nhỏ hoặc khi mô hình quá phức tạp.

- Hiệu suất của Adam có thể phụ thuộc nhiều vào việc lựa chọn siêu tham số (hyper-parameter) như learning rate và beta.

- Một vài trường hợp cần nhiều bộ nhớ cho việc lưu trữ so với các thuật toán khác.

### Adadelta:

Adadelta là một biến thể khác của AdaGrad. Điểm khác biệt chính là Adadelta giảm mức độ mà tốc độ học sẽ thay đổi. Adadelta thường được biết đến là thuật toán không sử dụng tốc độ học vì nó dựa trên chính lượng thay đổi hiện tại để tính toán lượng thay đổi trong tương lai.

Các bước cập nhật của mô hình dựa trên các công thức:

- Tính gradient trung bình:

- Tính delta x trung bình:

- Tính tỷ lệ học mới:

- Cập nhật tham số:

Trong đó:

là gradient tại thời điểm t

là thay đổi của tham số tại thời điểm t

p là hệ số giảm dần gradient

là hệ số tránh lỗi (do chia cho mẫu bằng 0)

Ưu điểm:

- Linh hoạt trong tốc độ học, giảm áp lực cho việc chọn siêu tham số

- Hiệu quả trên dữ liệu lớn, ít tiêu tốn tài nguyên hơn so với các thuật toán khác.

Nhược điểm:

- Khó hiểu và tính phức tạp cao hơn so với các thuật toán optimizer khác.

### Nesterov-accelerated Adaptive Moment Estimation (Nadam):

Nadam là một thuật toán tối ưu hóa được kết hợp bởi 2 thuật toán tối ưu hóa là: Nesterov Accelerated Gradient (NAG) và Adaptive Moment Estimation (Adam). Tên “Nadam” là sự kết hợp của “Nesterov” và “Adam”. Nadam có tất cả các ưu điểm mà 2 thuật toán cấu thành mang lại, thậm chí có phần cải tiến hơn.

Công thức:

Trong đó:

là gradient tại thời điểm t

và là các trung bình động tuyến tính và không gian tốc độ của gradient.

và là các hệ số giảm dần gradient.

là tỷ lệ học

là hệ số tránh lỗi (do chia cho mẫu bằng 0)

Ưu điểm:

- Hiệu suất tốt, hay vượt trội theo từng loại bài toán.

- Khả năng linh động trong tốc độ học

- Tiết kiệm tài nguyên, không yêu cầu nhiều về bộ nhớ

Nhược điểm: Công thức và nguyên tắc hoạt động của Nadam có thể rất phức tạp và khó hiểu so với người vừa bắt đầu học về các thuật toán optimizer.

### FTRL-Proximal:

FTRL-Proximal là một thuật toán tối ưu hóa được thiết kế cho bài toán tối ưu hóa đa biến (multivariate optimization) trong học máy. Tên đầy đủ của thuật toán là "Follow the Regularized Leader - Proximal" (FTRL-Proximal).

Nguyên tắc hoạt động:

FTRL-Proximal là một thuật toán tối ưu hóa dựa trên nguyên tắc "Follow the Regularized Leader". Nguyên tắc này kết hợp việc tối ưu hóa loss function và áp dụng kỹ thuật Cross-validation cost để hạn chế overfitting. Cụ thể, FTRL-Proximal cập nhật các trọng số của mô hình dựa trên gradient của hàm chi phí và một Cross-validation cost (regularization term).

Ưu điểm:

- Khả năng “từ chối” overfitting nhờ việc có kết hợp từ kỹ thuật Cross-validation giúp hạn chế vấn đề cực kỳ quan trọng này.

- Hiệu quả với dũ liệu lớn và hiệu quả với các dữ liệu thưa thớt.

Nhược điểm:

- Tính phức tạp về cả lý thuyết và triển khai là một rào cản lớn đối với người mới bắt đầu học

- Yêu cầu về đặc tính (feature) của dữ liệu, Hiệu suất của mô hình phụ thuộc nhiều vào đặc tính của dữ liệu.

### Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS):

Là một biến thể của thuật toán BFGS (Broyden-Fletcher-Goldfarb-Shanno) mà giữ lại một số lượng hữu hạn các vectơ gradient trước đó để giảm bộ nhớ cần thiết. Với độ phức tạp cực cao nên công thức thường sẽ không được biểu diễn dễ dàng. Nó liên quan đến các bước cập nhật gradient và ma trận Hessian.

Ưu điểm:

- Hiệu quả với bài toán có số chiều lớn

- Không yêu cầu ma trận Hessian đầy đủ, giúp giảm bớt gánh nặng tính toán của việc lưu trữ hoặc tính toán ma trận Hessian.

Nhược điểm:

- Khả năng hội tụ không được đảm bảo, có thể bị mắc kẹt tại điểm local minimum.

- Khả năng quyết định khoảng cách bước nhảy không đảm bảo, kéo đến việc hiệu suất của bài toán bị sụt giảm.

### So sánh các phương pháp Optimizer:

|  |  |  |
| --- | --- | --- |
| Thuật Toán | Ưu Điểm | Nhược Điểm |
| LBFGS | - Hiệu quả với số chiều lớn và không yêu cầu tính toán hoặc lưu trữ đầy đủ ma trận Hessian. | - Không đảm bảo hội tụ cho mọi vấn đề tối ưu và có thể bị mắc kẹt tại các điểm cực tiểu cục bộ. |
| FTRL-Proximal | - Hiệu quả khi xử lý các vấn đề học máy với dữ liệu lớn và thưa. | - Đòi hỏi đặc trưng về tính toán và điều chỉnh siêu tham số có thể là một thách thức. |
| Nadam | - Kết hợp ưu điểm của Adam và Nesterov Accelerated Gradient (NAG). - Cải thiện khả năng hội tụ và chống overshooting so với Adam. | - Cần hiểu rõ các tham số để tránh hiện tượng quá mức hoặc mất khả năng hội tụ. |
| Adadelta | - Không yêu cầu sự điều chỉnh thủ công của tỷ lệ học (learning rate). - Hiệu quả khi xử lý các vấn đề với dữ liệu thưa và không yêu cầu việc lưu trữ toàn bộ lịch sử gradient. | - Cần điều chỉnh một số tham số như rho và epsilon, và không phù hợp cho mọi loại bài toán. |
| Adam | - Hiệu quả với dữ liệu lớn, không yêu cầu thực hiện các điều chỉnh thủ công của learning rate và thường là một lựa chọn an toàn. | - Cần hiểu rõ các tham số và có thể đôi khi không hội tụ đúng cách nếu không được cấu hình đúng. |
| RMSprop | - Hiệu quả với dữ liệu không đồng nhất, giảm ảnh hưởng của gradient nhỏ hoặc lớn. - Tích hợp khả năng tự giảm learning rate dựa trên gradient lịch sử. | - Cần điều chỉnh tham số learning rate và không phải lựa chọn tốt cho mọi bài toán. |
| Adagrad | - Tích hợp khả năng tự điều chỉnh learning rate dựa trên lịch sử gradient. - Hiệu quả với các bài toán có đặc điểm dữ liệu thưa. | - Tăng learning rate theo thời gian có thể làm giảm hiệu suất và không phù hợp với mọi bài toán. |
|  |  |  |

# CONTINUAL LEARNING

## Tổng quan

Continual Learning (Học Liên Tục) là một phương pháp trong lĩnh vực học máy mà mô hình được liên tục cập nhật và mở rộng để học từ dữ liệu mới mà không quên kiến thức đã học trước đó. Điều này là quan trọng trong các hệ thống thời gian thực hoặc trong các ngữ cảnh mà dữ liệu thay đổi theo thời gian. Mục tiêu là giữ cho mô hình linh hoạt, có khả năng học từ những thay đổi liên tục mà không làm giảm hiệu suất trên dữ liệu cũ.

Continual learning (hay học tập liên tục) là ý tưởng cập nhật model (mô hình) của bạn khi có dữ liệu mới; điều này giúp mô hình của bạn theo kịp sự phân bổ dữ liệu hiện tại.

Sau khi mô hình của bạn được cập nhật, mô hình đó không thể được deploy (phát hành) một cách mù quáng vào sản xuất. Nó cần phải được thử nghiệm để đảm bảo rằng nó an toàn và tốt hơn so với mô hình hiện tại đang được sản xuất.

Continual learning thường bị hiểu sai:

- Continual learning KHÔNG đề cập đến một lớp thuật toán Machine Learning (ML) đặc biệt cho phép cập nhật mô hình dần dần khi có mọi điểm dữ liệu mới.

- Continual learning KHÔNG có nghĩa là bắt đầu công việc đào tạo lại mỗi khi có mẫu dữ liệu mới. Trên thực tế, điều này rất nguy hiểm vì nó khiến mạng lưới thần kinh dễ bị forgetting (lãng quên) một cách thảm họa.

## Mục tiêu của Continual Learning

Lý do cơ bản là để giúp mô hình của bạn theo kịp sự thay đổi phân phối dữ liệu. Có một số trường hợp sử dụng trong đó việc thích ứng nhanh chóng với việc thay đổi phân phối là rất quan trọng.

Ví dụ: có thể có một buổi hòa nhạc ở một khu vực ngẫu nhiên vào Thứ Hai ngẫu nhiên và "mô hình Machine Learning định giá buổi hòa nhạc vào Thứ Hai" có thể không được trang bị tốt để xử lý nó.

## Stateless retraining và Stateful retraining:

### Stateless retraining:

-Khái niệm: Đào tạo lại mô hình của bạn từ đầu mỗi lần, sử dụng trọng số được khởi tạo ngẫu nhiên và dữ liệu mới.

- Đặc điểm:

+ Có thể có một số trùng lặp với dữ liệu đã được sử dụng để huấn luyện phiên bản mô hình trước đó.

+ Hầu hết các công ty bắt đầu thực hiện việc học tập liên tục bằng cách sử dụng stateless retraining.

+ Thỉnh thoảng bạn sẽ cần phải chạy stateless retraining với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình.

### Stateful retraining ( fine-tuning, incremental learning):

- Khái niệm: Khởi tạo mô hình của bạn với các trọng số từ vòng huấn luyện trước và tiếp tục huấn luyện bằng cách sử dụng dữ liệu mới chưa thấy.

- Đặc điểm:

+ Cho phép mô hình của bạn cập nhật với lượng dữ liệu ít hơn đáng kể.

+ Cho phép mô hình của bạn hội tụ nhanh hơn và sử dụng ít năng lượng tính toán hơn.

+ Một số công ty đã báo cáo giảm 45% sức mạnh tính toán.

+ Về mặt lý thuyết, nó có thể tránh việc lưu trữ dữ liệu hoàn toàn sau khi dữ liệu đã được sử dụng để đào tạo (và để lại một khoảng thời gian an toàn). Về mặt lý thuyết, điều này giúp loại bỏ những lo ngại về quyền riêng tư dữ liệu.

+ Trên thực tế, hầu hết các công ty đều có thói quen theo dõi mọi thứ và không muốn vứt bỏ dữ liệu ngay cả khi nó không còn cần thiết nữa.

+ Sau khi cơ sở hạ tầng của bạn được thiết lập chính xác, việc thay đổi từ stateless retraining sang stateful retraining sẽ trở thành một nút nhấn.

+ Model iteration vs data iteration (Lặp lại mô hình và lặp lại dữ liệu): stateful retraining chủ yếu được sử dụng để kết hợp dữ liệu mới vào kiến trúc mô hình cố định và hiện có (tức là lặp lại dữ liệu). Nếu muốn thay đổi các tính năng hoặc kiến trúc của mô hình, bạn sẽ cần phải thực hiện bước đầu tiên là stateless retraining.

\* Đã có một số nghiên cứu về cách chuyển trọng số từ kiến trúc mô hình này sang kiến trúc mô hình mới (Net2Net knowledge transfer, model surgery). Có rất ít hoặc chưa có sự áp dụng những kỹ thuật này trong công nghiệp.

## Log and Wait:

### Khái niệm:

- Log (Ghi nhật ký): Ghi nhật ký trong quá trình học liên tục là quan trọng để theo dõi hiệu suất của mô hình theo thời gian và đối mặt với thách thức của việc tiếp tục học.

- Wait (Chờ đợi): Trong quá trình học liên tục, "wait" có thể ám chỉ thời gian chờ đợi giữa các lượt học mới để đảm bảo rằng mô hình đã học đủ từ dữ liệu mới.

### Mục đích sử dụng:

- Log:

+ Ghi lại thông tin về hiệu suất của mô hình sau mỗi lượt học mới.

+ Ghi nhật ký về các vấn đề như quá mức quên (catastrophic forgetting) hay hiệu ứng ngữ cảnh trước đó.

- Wait:

+ Ghi lại thông tin về hiệu suất của mô hình sau mỗi lượt học mới.

+ Ghi nhật ký về các vấn đề như quá mức quên (catastrophic forgetting) hay hiệu ứng ngữ cảnh trước đó.

Một số công ty lưu trữ các features đã được tính toán cho mọi mẫu dữ liệu để có thể sử dụng lại chúng cho mục đích đào tạo học tập liên tục và bằng cách đó tiết kiệm được kha khá tài nguyên. Việc này được gọi là log and wait.

## Thách thức của Continual Learning:

- Catastrophic Forgetting: Khi học một nhiệm vụ mới, mô hình có thể quên mất kiến thức đã học từ các nhiệm vụ trước đó.

- Fresh data access challenge(Thách thức truy cập dữ liệu mới): Tốc độ lưu trữ dữ liệu vào kho dữ liệu, Tốc độ ghi nhãn.

- Knowledge Transfer: Làm thế nào để chia sẻ và sử dụng lại kiến thức đã học từ một nhiệm vụ cho những nhiệm vụ mới.

- Representation Management: Làm thế nào để biểu diễn mô hình sao cho nó có thể linh hoạt và thích ứng được với nhiều loại dữ liệu.

- Evaluation Challenge (Thách thức đánh giá): việc học hỏi liên tục sẽ mở ra cơ hội cho các cuộc tấn công đối nghịch phối hợp nhằm đầu độc các mô hình.

- Data scaling challenge (Thách thức mở rộng quy mô dữ liệu): Việc mở rộng quy mô yêu cầu quyền truy cập vào số liệu thống kê dữ liệu toàn cầu như tối thiểu, tối đa, trung bình và phương sai. (min, max, average and variance.)

- Algorithm challenge (Thách thức thuật toán): Thách thức này xuất hiện khi bạn sử dụng một số loại thuật toán nhất định và muốn cập nhật chúng thật nhanh (ví dụ: mỗi giờ).

## Các giai đoạn của một Continual Learning:

Giai đoạn 1 - Manual, stateless retraining (đào tạo (lại) thủ công bằng stateless): Các mô hình chỉ được đào tạo lại khi đáp ứng hai điều kiện: (1) hiệu suất của mô hình đã giảm sút đến mức hiện tại nó gây hại nhiều hơn là có lợi, (2) nhóm của bạn có thời gian để cập nhật mô hình.

Giai đoạn 2 - Fixed schedule automated stateless retraining (Đào tạo (lại) bằng stateless retraining theo lịch trình cố định): Giai đoạn này thường xảy ra khi các mô hình chính của một miền đã được phát triển và do đó ưu tiên của bạn không còn là tạo các mô hình mới mà là duy trì và cải thiện các mô hình hiện có. Tần suất đào tạo lại ở giai đoạn này thường dựa trên: "gut feeling". (cảm giác “gut”). Các bước trong 1 quy trình đào tạo lại bao gồm:

- Pull data: Lấy dữ liệu.

- Downsample or upsample data if necessary: Xuống mẫu hoặc lấy mẫu dữ liệu nếu cần thiết.

- Extract features: Trích xuất các đặc trưng.

- Process and/or annotate labels to create training data: Xử lý và/hoặc chú thích (gán) nhãn để tạo dữ liệu đào tạo.

- Kick off the training process: Bắt đầu quá trình đào tạo.

- Evaluate the new model: Đánh giá mô hình mới.

- Deploy: Triển khai

Giai đoạn 3 - Fixed schedule automated stateful training (Đào tạo stateful theo lịch trình cố định): Để đạt được điều này, bạn cần phải cấu hình lại tập lệnh của mình và cách theo dõi dòng dữ liệu cũng như mô hình của bạn.

Giai đoạn 4 - Continual learning: Trong giai đoạn này, phần lịch trình cố định của các giai đoạn trước được thay thế bằng một số re-training trigger mechanism (trình kích hoạt đào tạo lại cơ chế). Các triggers có thể là:

- Time-based: Dựa trên thời gian

- Performance-based: Dựa trên hiệu suất

- Volume-based: Dựa trên khối lượng

- Drift-based: Dựa trên trôi dạt

* Tần suất cập nhật mô hình của bạn: trước tiên bạn cần hiểu và xác định bạn nhận được lợi ích gì khi cập nhật mô hình của mình với dữ liệu mới. Càng đạt được nhiều lợi ích thì càng phải đào tạo lại thường xuyên hơn.
* Đo lường giá trị của độ mới dữ liệu: Một cách để định lượng giá trị của dữ liệu mới là huấn luyện cùng một kiến trúc mô hình với dữ liệu từ 3 khoảng thời gian khác nhau, sau đó kiểm tra từng mô hình dựa trên dữ liệu được gắn nhãn hiện tại (xem hình ảnh).
* Nếu bạn phát hiện ra rằng việc để mô hình cũ trong 3 tháng sẽ gây ra sự khác biệt 10% về độ chính xác của dữ liệu thử nghiệm hiện tại và 10% là không thể chấp nhận được, thì bạn cần đào tạo lại sau chưa đầy 3 tháng.

## Khi nào thì nên cập nhật lại mô hình?

Hầu hết các kiến thức tôi đã đề cập đều là liên quan đến việc cập nhật mô hình với dữ liệu mới (tức là lặp lại dữ liệu). Tuy nhiên, trong thực tế, đôi khi bạn cũng có thể cần thay đổi kiến trúc mô hình của mình (tức là lặp lại mô hình).Dưới đây là một số gợi ý về khi bạn nên và không nên xem xét việc lặp lại mô hình:

- Nếu bạn liên tục giảm ngưỡng kích hoạt tái đào tạo dữ liệu và bạn không đạt được nhiều hiệu suất, bạn có thể nên tìm kiếm một mô hình tốt hơn (nếu doanh nghiệp của bạn cần).

- Nếu việc chuyển sang một kiến trúc mô hình lớn hơn và yêu cầu 100 lần công suất tính toán mang lại 1% cải thiện hiệu suất, nhưng giảm ngưỡng kích hoạt tái đào tạo xuống 3 giờ cũng mang lại tăng hiệu suất 1% với 1 lần công suất tính toán, hãy ưu tiên lặp lại dữ liệu hơn là lặp lại mô hình.

- Câu hỏi "khi nào nên lặp lại mô hình so với lặp lại dữ liệu" vẫn chưa có câu trả lời tốt cho tất cả các nhiệm vụ. Bạn sẽ cần thực hiện các thử nghiệm trên nhiệm vụ cụ thể của mình để tìm ra lúc nào nên thực hiện lựa chọn nào.

# TEST PRODUCTION

## Tổng quan:

Test Production (Kiểm Thử Sản Xuất) là một giai đoạn quan trọng trong vòng đời của một mô hình học máy, chuyển từ giai đoạn phát triển và đào tạo sang triển khai trong các kịch bản thực tế. Trong quá trình Kiểm Thử Sản Xuất, mục tiêu chính là đánh giá hiệu suất của mô hình khi đối mặt với dữ liệu thực tế, thường là dữ liệu động từ môi trường mà nó được thiết kế để hoạt động.

Để kiểm tra đầy đủ các mô hình của bạn trước khi phổ biến rộng rãi, bạn cần cả đánh giá ngoại tuyến trước khi triển khai (pre-deployment offline evaluations) và thử nghiệm trong sản xuất (testing in production). Chỉ đánh giá ngoại tuyến là không đủ.

Tốt nhất là các quy trình đánh giá này được tự động hóa và khởi động khi có bản cập nhật mô hình mới. Việc nâng cấp giai đoạn cần được xem xét tương tự như cách đánh giá CI/CD ( Continuous Integration (tích hợp liên tục)/Continuous Delivery (chuyển giao liên tục) ) trong công nghệ phần mềm.

## Phương pháp đánh giá:

### Model Offline Evaluation (Hay đánh giá trước khi triển khai):

Chúng ta đã quen với việc đánh giá một mô hình Machine Learning trong đó chúng ta huấn luyện một mô hình và đánh giá hiệu suất của nó trên một bộ dữ liệu xác thực chưa được nhìn thấy; điều này được thực hiện thông qua các số liệu (accuracy: độ chính xác), và biểu đồ đường cong ( precision-recall curve). Việc đánh giá 1 mô hình Machine learning trong sản xuất có thể dựa trên nhiều yếu tố tùy thuộc vào mục đích sử dụng của mô hình đó. Ta có 2 yếu tố để có thể đánh giá một mô hình trước khi triển khai:

- Baseline (Đường cơ sở): bạn cần một cái gì đó để so sánh. Chúng ta có nhiều loại baseline:

+ Đường cơ sở ngẫu nhiên (Random baseline)

+ Heuristic đơn giản (Simple heuristic)

+ Đường cơ sở không có quy tắc (Zero rule baseline)

+ Cơ sở con người (Human baseline)

+ Cơ sở giải pháp hiện có (Existing solution baseline)

- Các phương pháp đánh giá vượt quá các thước đo tổng quát trong học máy::

+ Đánh giá độ bền bằng các thử nghiệm nhiễu loạn (Evaluating robustness with perturbation tests): Kiểm tra khả năng chịu nhiễu loạn của mô hình bằng cách thêm nhiễu loạn vào dữ liệu đầu vào và đánh giá sự biến động của dự đoán.

+ Đánh giá tính công bằng bằng kiểm định bất biến (Evaluating fairness with invariance tests): Đánh giá công bằng của mô hình thông qua các kiểm tra về tính không biến đổi trên các thuộc tính quan trọng, đảm bảo công bằng trong quá trình dự đoán.

+ Kiểm tra lựa chọn với các kiểm tra kỳ vọng rời rạc (Sanity checks with discretional expectation tests): Thực hiện các kiểm tra đối với giả định và mong đợi dữ liệu rời rạc để đảm bảo tính đúng đắn và an toàn của mô hình.

+ Đánh giá hiệu chuẩn (Evaluating calibration): Đánh giá khả năng của mô hình để đạt đến các dự đoán chuẩn xác và đồng nhất với dữ liệu thực tế.

+ Kiểm tra lựa chọn với đo lường sự tin cậy (Sanity checks with confidence measurement): Thực hiện các kiểm tra để đánh giá mức độ tự tin của mô hình trong việc đưa ra dự đoán.

+ Đánh giá hiệu suất và tính công bằng bằng các bài kiểm tra đánh giá dựa trên lát cắt (Evaluating performance and fairness with slice-based evaluation tests): Phân tích hiệu suất và công bằng của mô hình dựa trên các lát (phân khúc) của dữ liệu để xác định sự chệch và hiệu suất trong các nhóm cụ thể.

Trong đó được đề cập đến 2 giải pháp được sử dụng phổ biến nhất là: test split và Backtesting.

- Phần phân tách thử nghiệm (Test splits) thường ở dạng static (tĩnh) để ta có điểm chuẩn (dựa trên baseline) đáng tin cậy để so sánh nhiều mô hình. Điều này cũng có nghĩa là hiệu suất tốt trên phần test split tĩnh cũ không đảm bảo hiệu suất tốt trong điều kiện phân phối dữ liệu hiện tại trong sản xuất.

- Kiểm tra ngược (Backtesting) là ý tưởng sử dụng dữ liệu được gắn nhãn mới nhất mà mô hình chưa thấy trong quá trình đào tạo để kiểm tra hiệu suất (ví dụ: nếu ta đã sử dụng dữ liệu của ngày cuối cùng, hãy sử dụng dữ liệu của giờ cuối cùng để kiểm tra lại).

### Testing in Production Strategies (Hay đánh giá trong triển khai):

#### Shadow Deployment

- Ý tưởng: Triển Khai Mô Hình Thách Thức Song Song với Mô Hình Hiện Tại. Chuyển mọi yêu cầu đến cả hai mô hình, nhưng chỉ phục vụ dự đoán từ mô hình hiện tại. Ghi lại các dự đoán của cả hai mô hình để so sánh chúng.

- Ưu điểm:

+ Đây là cách triển khai an toàn nhất cho mô hình. Ngay cả khi mô hình mới có lỗi, các dự đoán sẽ không được phục vụ.

+ Nó mang tính khái niệm đơn giản.

+ Cuộc thử nghiệm của bạn sẽ thu thập đủ dữ liệu để đạt đến ý nghĩa thống kê nhanh hơn so với tất cả các chiến lược khác, vì tất cả các mô hình đều nhận toàn bộ lưu lượng.

- Nhược điểm:

+ Kỹ thuật này không thể sử dụng khi đo lường hiệu suất của mô hình phụ thuộc vào việc quan sát cách người dùng tương tác với các dự đoán. Ví dụ, các dự đoán từ các mô hình đề xuất đồng thời sẽ không được phục vụ, do đó, bạn sẽ không biết người dùng có nhấp chuột vào chúng hay không.

+ Kỹ thuật này tốn kém vì nó làm tăng gấp đôi số lượng dự đoán và do đó làm tăng số lượng tính toán cần thiết.

+ Nếu việc dự đoán xảy ra bằng bất kỳ chế độ dự đoán trực tuyến nào, bạn sẽ cần tìm cách xử lý các trường hợp biên như:

Điều gì xảy ra nếu mô hình đồng bóng mất nhiều thời gian hơn mô hình chính để phục vụ một dự đoán?

Điều gì xảy ra nếu mô hình đồng bóng thất bại? Mô hình chính cũng sẽ thất bại hay không?

#### A/B Testing

- Ý Tưởng: Triển Khai Mô Hình Thách Thức Cùng Lúc với Mô Hình Hiện Tại (Mô Hình A) và Định Tuyến Một Phần Giao Thông (traffic) Đến Mô Hình Thách Thức (Mô Hình B). Dự đoán từ Mô Hình Thách Thức được hiển thị cho người dùng. Sử dụng giám sát và phân tích dự đoán trên cả hai mô hình để xác định xem hiệu suất của Mô Hình Thách Thức có độ chệch thống kê so với Mô Hình Hiện Tại hay không.

Một số trường hợp sử dụng không phù hợp với ý tưởng chia giao thông và sử dụng nhiều mô hình cùng một lúc. Trong những trường hợp này, kiểm thử A/B có thể được thực hiện bằng cách thực hiện chia đôi thời gian: một ngày Mô Hình A, ngày tiếp theo Mô Hình B.

Chia giao thông phải là một thử nghiệm ngẫu nhiên thực sự. Nếu bạn tạo ra bất kỳ độ chệch lựa chọn nào về việc ai nhận Mô Hình A so với Mô Hình B (ví dụ: người dùng máy tính để bàn nhận A và di động nhận B), các kết luận của bạn sẽ là không chính xác.

Thử nghiệm phải chạy đủ thời gian để thu thập đủ mẫu để đạt được độ chắc chắn thống kê về sự khác biệt.

Ý nghĩa thống kê không phải là điều hoàn toàn đảm bảo (đó là lý do vì sao nó có độ tin cậy). Nếu không có sự khác biệt thống kê nào giữa A và B, bạn có thể sử dụng bất kỳ mô hình nào.

Không có điều gì ngăn cản bạn chạy các thử nghiệm A/B/C/D nếu bạn muốn.

- Ưu Điểm:

+ Vì dự đoán được phục vụ cho người dùng, phương pháp này cho phép bạn hoàn toàn nắm bắt cách người dùng phản ứng với các mô hình khác nhau.

+ Thử nghiệm A/B là dễ hiểu và có nhiều thư viện và tài liệu xung quanh nó.

+ Phương pháp này tiết kiệm chi phí vì chỉ có một dự đoán cho mỗi yêu cầu.

+ Bạn không cần phải xem xét các trường hợp biên phát sinh từ việc định tuyến dự đoán song song cho các chế độ dự đoán trực tuyến (xem nhược điểm của triển khai đồng bóng).

- Nhược điểm:

+ Đây là phương pháp ít an toàn hơn so với triển khai đồng bóng. Bạn cần một đảm bảo đánh giá mạnh mẽ hơn về hiệu suất ngoại tuyến của mô hình để đảm bảo rằng nó sẽ không thất bại nặng nề, vì bạn sẽ đưa giao thông thực tế vào nó.

+ Bạn có sự lựa chọn tự nhiên giữa giả sử nhiều rủi ro hơn (định tuyến nhiều lưu lượng đến mô hình B) SO VỚI đạt được đủ mẫu để thực hiện phân tích nhanh hơn.

#### Canary Release (Phát Hành Canary)

- Ý Tưởng: Triển khai mô hình thách thức và mô hình hiện tại cùng nhau, nhưng bắt đầu bằng việc mô hình thách thức không nhận bất kỳ lưu lượng nào. Dần dần chuyển lưu lượng từ mô hình hiện tại sang mô hình thách thức (gọi là canary). Giám sát các chỉ số hiệu suất của mô hình thách thức, nếu chúng trông tốt, tiếp tục cho đến khi tất cả lưu lượng đều chuyển sang mô hình thách thức.

Phát hành Canary có thể được kết hợp với kiểm thử A/B để đo lường sự khác biệt hiệu suất một cách chặt chẽ.

Phát hành Canary cũng có thể chạy ở "chế độ YOLO" (You Only Look Once), trong đó bạn đánh giá sự khác biệt hiệu suất theo cách thủ công.

Một phiên bản khác của phát hành Canary có thể là việc phát hành mô hình thách thức cho một thị trường nhỏ trước và sau đó triển khai cho tất cả thị trường nếu mọi thứ đều ổn.

Nếu mô hình thách thức bắt đầu gặp vấn đề, định tuyến lại lưu lượng về mô hình hiện tại.

- Ưu điểm:

+ Dễ hiểu.

+ Đơn giản nhất trong tất cả các chiến lược để triển khai nếu bạn đã có cơ sở hạ tầng cờ tính nào đó trong công ty.

+ Vì các dự đoán của mô hình thách thức sẽ được phục vụ, bạn có thể sử dụng điều này với các mô hình yêu cầu tương tác của người dùng để nắm bắt hiệu suất.

+ So với triển khai đồng bóng, nó rẻ hơn để chạy. Một dự đoán cho mỗi yêu cầu.

+ Nếu kết hợp với kiểm thử A/B, nó cho phép bạn thay đổi động lực lượng mà mỗi mô hình đang nhận.

- Nhược điểm:

+ Nó mở ra khả năng không nghiêm túc trong việc xác định sự khác biệt hiệu suất.

+ Nếu các phiên bản không được giám sát cẩn thận, tai nạn có thể xảy ra. Điều này có thể coi là lựa chọn ít an toàn nhất nhưng nó rất dễ quay trở lại trạng thái trước.

#### Interleaving Experiments (Thử Nghiệm Xen Kẽ)

- Ý Tưởng: Trong kiểm thử A/B, một người dùng chỉ nhận các dự đoán từ mô hình A hoặc mô hình B. Trong thử nghiệm xen kẽ, một người dùng nhận dự đoán xen kẽ từ cả mô hình A và mô hình B. Sau đó, chúng ta theo dõi hiệu suất của từng mô hình bằng cách đo lường sự ưa thích của người dùng với các dự đoán của từng mô hình (ví dụ: người dùng nhấp nhiều hơn vào các đề xuất từ mô hình B).

Các nhiệm vụ đề xuất thường là một trường hợp sử dụng điển hình cho thử nghiệm xen kẽ. Không phải tất cả các nhiệm vụ đều phù hợp với chiến lược này.

Bạn muốn tránh tình trạng một mô hình có lợi thế ưa thích của người dùng không công bằng như luôn chọn lựa đầu tiên từ mô hình A. Nên có khả năng tương đối nhau giữa việc chọn lựa đầu tiên từ mô hình A và mô hình B. Các vị trí còn lại có thể được điền bằng phương pháp "team-drafting".

- Ưu điểm:

+ Netflix đã phát hiện thử nghiệm xen kẽ đáng tin cậy xác định mô hình tốt nhất với kích thước mẫu đáng kể nhỏ hơn so với kiểm thử A/B truyền thống.

+ Điều này chủ yếu là do cả hai mô hình đều nhận lưu lượng đầy đủ.

+ Ngược lại với triển khai đồng bóng, chiến lược này cho phép bạn nắm bắt cách người dùng đang tương tác với dự đoán của bạn (vì dự đoán được phục vụ).

- Nhược điểm:

+ Việc triển khai phức tạp hơn so với kiểm thử A/B.

+ Bạn cần lo lắng về các trường hợp biên như phải làm gì nếu một trong những mô hình xen kẽ mất quá nhiều thời gian để phản hồi hoặc gặp sự cố.

+ Gấp đôi công suất tính toán cần thiết vì mỗi yêu cầu đều nhận dự đoán từ nhiều mô hình.

+ Nó không thể được sử dụng cho tất cả các loại nhiệm vụ. Ví dụ, nó hoạt động cho các nhiệm vụ xếp hạng/gợi ý nhưng không có ý nghĩa cho các nhiệm vụ hồi quy.

+ Nó không dễ dàng mở rộng lên một số lượng lớn các mô hình thách thức. 2-3 mô hình xen kẽ có vẻ là điểm ngọt.

#### Bandits

- Ý Tưởng: Bandits là một thuật toán theo dõi hiệu suất hiện tại của mỗi biến thể mô hình và đưa ra quyết định động trên mỗi yêu cầu về việc sử dụng mô hình có hiệu suất tốt nhất cho đến nay (tức là khai thác kiến thức hiện tại) hoặc thử nghiệm bất kỳ mô hình nào khác để có thêm thông tin về chúng (tức là khám phá trong trường hợp một trong những mô hình khác tốt hơn).

Bandits thêm một khái niệm khác vào quyết định về việc sử dụng mô hình nào: chi phí cơ hội.

Bandits không thể được áp dụng cho tất cả các trường hợp. Để Bandits có thể áp dụng, nhiệm vụ của bạn cần:

+ Thực hiện dự đoán trực tuyến. Dự đoán ngoại trực trực tuyến không tương thích với Bandits.

+ Bạn cần có các chu kỳ phản hồi ngắn để xác định xem dự đoán có tốt hay không và một cơ chế để truyền phản hồi đó vào thuật toán Bandit để cập nhật phần thưởng của từng mô hình.

Có nhiều thuật toán Bandits. Thuật toán đơn giản nhất được gọi là epsilon-greedy. Hai thuật toán mạnh mẽ và phổ biến nhất là Thompson Sampling và Upper Confidence Bound (UCB).

- Ưu điểm:

+ Bandits cần ít dữ liệu hơn so với A/B testing để xác định mô hình nào tốt hơn. Một ví dụ được đề cập trong sách là cần 630K mẫu để đạt được 95% tin cậy với A/B testing so với 12K với Bandits.

+ Bandits hiệu quả dữ liệu hơn đồng thời giảm thiểu chi phí cơ hội. Trong nhiều trường hợp, Bandits được coi là tối ưu.

+ So với A/B testing, Bandits an toàn hơn vì nếu một mô hình thực sự kém, thuật toán sẽ chọn nó ít hơn. Hơn nữa, sự hội tụ sẽ nhanh chóng, giúp bạn loại bỏ mô hình thách thức xấu nhanh chóng.

- Nhược điểm:

+ So với tất cả các chiến lược khác, Bandits khó triển khai hơn rất nhiều do cần phải truyền phản hồi vào thuật toán liên tục.

+ Bandits chỉ có thể được sử dụng trong một số trường hợp sử dụng cụ thể (xem trên).

+ Nó không an toàn như Shadow Deployment vì những mô hình thách thức nhận lưu lượng trực tiếp.

Có một phân khúc nhỏ của thuật toán bandit được gọi là contextual bandit. Contextual bandits không chỉ xem xét "phần thưởng" lịch sử của mỗi tùy chọn để đưa ra quyết định "khám phá so với khai thác". Chúng cũng xem xét thông tin bổ sung (tức là ngữ cảnh). Ngữ cảnh này có thể là: thông tin về người dùng, thông tin về thời gian trong ngày/năm, thông tin về sản phẩm. Với cả phần thưởng và ngữ cảnh, contextual bandits có thể đưa ra quyết định "khám phá so với khai thác" cá nhân hóa.

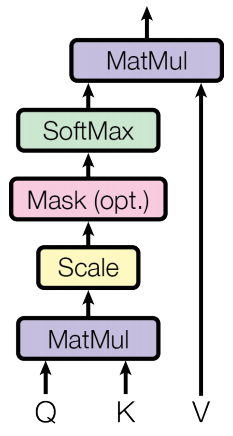
Contextual bandits đã được sử dụng thành công trong các nhiệm vụ đề xuất, cải thiện hiệu suất đáng kể. Ví dụ, trong môi trường thương mại điện tử, chúng có thể được sử dụng để quyết định liệu có nên hiển thị cho BẠN một sản phẩm MÀ BẠN có khả năng mua (khai thác) hay một sản phẩm KHÁC MÀ BẠN chưa từng thấy trước đó (khám phá). Lưu ý rằng quyết định này được đưa ra cho BẠN, xem xét thông tin về BẠN và có thể là thông tin về các sản phẩm có thể làm ngữ cảnh.

Contextual bandits có thể được sử dụng "đơn độc" như là các thuật toán đề xuất không cần dữ liệu huấn luyện. Chúng bắt đầu không có ngữ cảnh và phần thưởng, đưa ra một đoán về điều gì nên đề xuất và học khi chúng đi.

Chúng cũng có thể kết hợp với học sâu để cải thiện hiệu suất đề xuất. Đọc bài báo này của Twitter engineering nếu bạn quan tâm.

Contextual bandits cũng có thể được sử dụng để chống lại các vòng lặp phản hồi suy giảm.

Nhược điểm chính của contextual bandits là chúng còn khó triển khai hơn so với bandits "bình thường". Việc triển khai của chúng phụ thuộc vào kiến trúc mô hình học máy cơ bản (ví dụ như cây so với mạng nơ-ron), làm cho chúng ít có thể tổng quát hóa hơn.



Hình 2.1: Scaled Dot-Product Attention

(Nguồn: (Vaswani et al., 2023))

Bảng 4.1: Thống kê kiểu thực thể trong tập VLSP 2016

|  |  |  |
| --- | --- | --- |
| **Label** | **Train** | **Test** |
| **LOC** | 1210 | 1377 |
| **ORG** | 7478 | 274 |
| **PER** | 6230 | 1294 |
| **MISC** | 250 | 47 |
| **Total** | 15168 | 2992 |

# TÀI LIỆU THAM KHẢO

Tiếng Việt

<https://machinelearningcoban.com/>

<https://phamdinhkhanh.github.io/>

<https://viblo.asia/>

Tiếng Anh

<https://www.upgrad.com/>

<https://machinelearningmastery.com/>