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

# BÁO CÁO CUỐI KỲ MÔN
# NHẬP MÔN HỌC MÁY

*Người hướng dẫn*: **GS.TS. LÊ ANH CƯỜNG**

*Người thực hiện*: **TRẦN VĂN HUY – 520H538**

Khoá **: 24**

**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**

# BÁO CÁO CUỐI KỲ MÔN
# NHẬP MÔN HỌC MÁY

Người hướng dẫn: **GS.TS. LÊ ANH CƯỜNG**
Người thực hiện: **TRẦN VĂN HUY**
Khoá **: 24**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

# BÁO CÁO ĐƯỢC HOÀN THÀNH
# TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Em xin cam đoan đây là công trình nghiên cứu của riêng em và được sự hướng dẫn khoa học của GS.TS. Lê Anh Cường. Các nội dung nghiên cứu, kết quả trong bài báo cáo này là trung thực. 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.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung bài báo cáo của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do em gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 11 tháng 12 năm 2023*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Trần Văn Huy*

# MỤC LỤC

# DANH MỤC HÌNH VẼ

# DANH MỤC BẢNG BIỂU

# DANH MỤC CÁC CHỮ VIẾT TẮT

| | |
|---|---|
| SGD | Stochastic Gradient Descent |
| SGDM | Stochastic Gradient Descent with Momentum |
| RMS | Root Mean Square |
| EWMA | Exponential Weighted Moving Average |
| Adagrad | Adaptive Gradient Descent |

# CHƯƠNG 1 – PHẦN LÝ THUYẾT

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

### *1.1 Stochastic Gradient Descent(SGD)*

Một trong những phương pháp optimizer được sửa dụng nhiều nhất trong học máy nói chung và trong bài toán deep learning nói riêng đó là Stochastic Gradient Descent hay còn gọi là suy giảm độ dốc ngẫu nhiên, đây là một optimizer biến thể từ thuật toán Gradient Descent. Trong một model học máy, ta luôn muốn giảm sự khác nhau giữa kết quả đầu ra và giá trị thực tế càng nhiều càng tốt hay nói cách khác là cực tiểu hóa hàm loss. Thuật toán GD có thể giúp chúng ta cực tiểu hóa hàm loss này bằng cách lặp lại nhiều lần quá trình tìm hướng dốc nhất của hàm để cập nhật tham số và cuối cùng có thể đi đến cực tiểu. Về cơ bản, các tham số weight và bias được cập nhật qua công thức sau:

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial_{w_{t-1}}}$$

$$b_t = b_{t-1} - \eta \frac{\partial L}{\partial_{b_{t-1}}}$$

Trong đó $\eta$ biểu thị cho tốc độ học và tốc độ học phải do ta tự chọn trước khi train mô hình.

Tuy nhiên nếu mà để GD là phương pháp optimizer thì đó có lẽ không phải là một lựa chọn tuyệt vời, bởi vì nếu như kích thước của bộ dữ liệu quá lớn thì việc tính toán độ dốc sẽ rất phức tạp và tốn kém. Do nó thực hiện việc tính toán trên toàn bộ tập trainning cho mỗi lần lặp, bằng cách lấy trung bình các gradient của hàm loss của các ví dụ học để cập nhật tham số nên cần nhiều bộ nhớ để lưu hết toàn bộ dữ liệu, vì điều đó mà người ta ưu tiên sử dụng SGD hơn do SGD sẽ giải quyết được vấn đề này.

Thuật toán của SGD hoạt động tương tự như GD nhưng thay vì thực hiện tính toán trên toàn bộ tập dữ liệu trong mỗi lần lặp như GD thì nó sẽ chọn ngẫu nhiên từng

bộ dữ liệu con được tạo từ m ví dụ học trong tập trainning, hay còn gọi là một batch một trong mỗi lần học để xử lý. Tuy nhiên, cũng vì thế mà phương pháp này gây nên tình trạng bị nhiễu và khiến cho hướng độ dốc dao động liên tục qua các lần học, vì thế mà SGD cần nhiều lần lặp hơn để đạt cực tiểu địa phương. Việc cần lặp nhiều hơn dẫn đến việc thời gian tính toán tăng theo làm cho quá trình hội tụ bị chậm lại, nhưng xét về độ phức tạp tính toán vẫn thấp hơn GD do chỉ xét một vài quan sát ở một thời điểm.

### *1.2 Stochastic Gradient Descent with Momentum*

Như ở phần 1.1 đã đề cập SGD cần nhiều lần lặp để đạt cực tiểu địa phương do nó chỉ thực hiện tính toán trên một vài quan sát một lần khiến cho hướng độ dốc dao động liên tục qua các lần học, từ đó dẫn đến việc SGD tốn nhiều thời gian tính toán. Stochastic Gradient Descent with Momentum là phương pháp có thể giải quyết được nhược điểm này. Về cơ bản thì SGDM gần như là giống với SGD nhưng mà có thêm thuật ngữ "Momentum" dịch ra là quán tính, quán tính ở đây có nghĩa là nó sử dụng lại các giá trị trước đó để thực hiện tính toán. Ý tưởng của thuật toán này là nó sẽ cố làm giảm tình trạng nhiễu sử dụng trung bình trượt trọng số mũ (Exponential Weighted Moving Average) hay EWMA. Ở mỗi lần lặp nó sẽ tích lũy thêm một phần của lần cập nhật trước đó và gần vào lần lặp hiện tại và điều này sẽ làm tăng nhanh quá trình hội tụ hay tìm ra điểm cực tiểu, khắc phục được nhược điểm của SGD như đã đề cập. Trọng số sẽ được cập nhật qua công thức sau:

$$w_t = w_t - \eta V d_{w_t}$$

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1 - \beta) \frac{\partial L}{\partial_{w_{t-1}}}$$

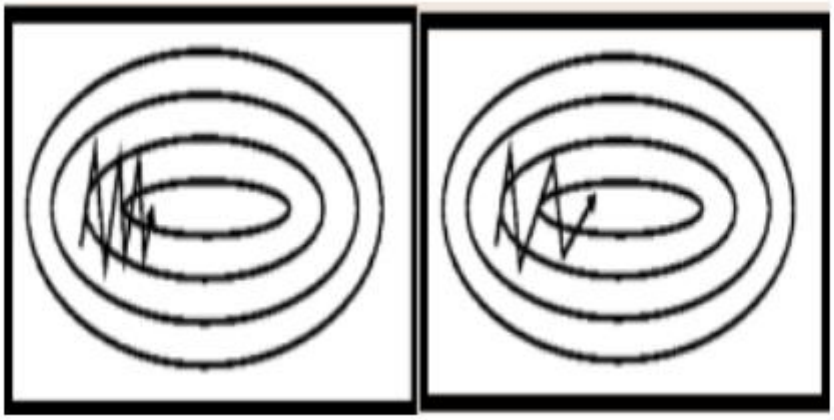

Figure 1. Quá trình hội tụ giữa SGD(trái) và SGDM(phải)

Hai hình ở trên cho chúng ta thấy sự khác nhau trong quá trình hội tụ giữa SGD và SGDM và có thể thấy SGDM có quá trình hội tụ nhanh hơn hẳn SGD. Mặc dù vậy ta không thể đốt cháy giai đoạn bằng cách tăng momentum và learning rate lên để tăng quá trình hội tụ được, bởi vì nếu như chúng ta càng tăng momentum lên thì khả năng chúng ta bỏ lỡ điểm cực tiểu càng cao và cũng tương tự như vậy đối với learning rate. Momentum là một tham số quan trọng trong SGDM, ảnh hưởng đến tốc độ và độ chính xác của việc tính toán. Để đạt được hiệu quả tối ưu, cần chọn momentum phù hợp với từng bài toán cụ thể. Nếu chọn momentum cao, cần giảm learning rate để tránh trường hợp bỏ lỡ điểm cực tiểu.

### *1.3 Adagrad (Apdaptive Gradient Descent)*

Adagrad optimizer là một phương pháp thay vì dùng một tốc độ học cố định như các biến thể của GD thì nó có khả năng điều chỉnh tốc độ học khác nhau ở mỗi lần học khác nhau. Nó sẽ điều chỉnh tốc độ học của các tham số qua mỗi lần lặp bằng cách chia cho tổng bình phương các gradient từ các lần tính trước, tham số nào có giá trị đạo hàm từng phần của hàm loss càng cao thì tốc độ học càng bé và ngược lại. Để hiểu kỹ hơn về phương pháp tự điều chỉnh tốc độ học này thì ta có công thức sau:

Giả sử ta có một trọng số $w_t$, phương pháp optimizer này sẽ cực tiểu hóa trọng số này qua công thức như sau:

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial_{w_{t-1}}}$$

Với tốc độ học được tính như sau:

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

Trong đó alpha t là tổng bình phương của các gradient của các lần trước:

$$\alpha_t = \sum_{i=1}^{t} (\frac{\partial L}{\partial w_i})^2$$

Alpha t đại diện cho những thông tin về các lần tính gradient phía trước và dùng nó để điều chỉnh tốc độ học mới, ε là một số nguyên dương giá trị nhỏ để tránh trường hợp phép chia cho 0.

Như vậy ta thấy nhờ có ưu điểm tự động điều chỉnh tốc độ học mà Adagrad giúp chúng ta không phải điều chỉnh nó bằng cách thủ công, việc có một tốc độ học được tự động điều chỉnh cũng giúp cho quá trình hội tụ khi sử dụng optimizer này nhanh hơn hẳn gradient descent và các biến thể. Tuy nhiên, trên thực tế, người ta thấy rằng đối với việc huấn luyện các mô hình mạng nơ-ron sâu việc tích lũy các bình phương của gradient từ đầu quá trình huấn luyện có thể dẫn đến giảm tốc độ học hiệu quả một cách đột ngột và quá mức. AdaGrad hoạt động tốt cho một số mô hình học sâu nhưng không phải tất cả.

### *1.4 RMS Prop(Root Mean Square)*

RMS prop là một phương pháp optimizer được cải thiện từ Adagrad. Nếu như Adagrad điều chỉnh tốc độ học bằng cách lưu lại các gradient của các lần trước thì nó sẽ gây ra tình trạng learning rate của các lần về sau sẽ giảm dần và trọng số của các lần học sau sẽ gần như không thay đổi. Để giải quyết tình trạng này, thay vì tích lũy các gradient từ đầu thì RMS prop là nó sẽ dùng trung bình trượt số mũ (Exponential Weighted Moving Average) hay còn gọi là EWMA. EWMA một cách tính toán mức độ quan trọng của các gradient trong quá khứ, sao cho các gradient gần đây được coi trọng hơn các gradient xa, với cách tính này thì RMS Prop chỉ chọn một vài gradient để tính từ các lần ngay phía trước chứ không quan tâm đến các gradient đã được tính từ rất lâu trong quá trình huấn luyện, từ đó chia gradient cho căn của EWMA với ép xi lông và chúng ta sẽ có được một learning rate ổn định qua mỗi lần học với công thức như sau:

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial_{w_{t-1}}}$$

$$\eta'_t = \frac{\eta}{\sqrt{w_{Avg(t)} + \varepsilon}}$$

$$w_{Avg(t)} = \beta w_{Avg(t-1)} + (1-\beta)(\frac{\partial L}{\partial_{w_{t-1}}})^2$$

Ngoài việc khắc phục được điểm yếu của Adagrad thì RMS Prop có nhược điểm là phải dựa vào một siêu tham số $\beta$ để điều chỉnh sự ảnh hưởng của các gradient ở phía trước. Chọn một siêu tham số quá lớn sẽ làm đến tốc độ quá trình hội tụ bị chậm lại, hay một siêu tham số quá nhỏ sẽ làm optimizer bị ảnh hưởng bởi độ nhiễu và trở nên không ổn định.

### *1.5 Adam*

Phương pháp optimizer Adam viết tắt cho Adaptive moment, được coi như là một biến thể kết hợp giữa RMSProp và momentum với 2 điểm khác biệt. Thứ nhất Adam tích hợp momentum trực tiếp vào ước lượng mômen bậc nhất. Điều này có nghĩa là Adam sử dụng momentum để điều chỉnh tốc độ học của các tham số, dựa trên các gradient gần đây. Thứ hai, Adam bao gồm các điều chỉnh độ lệch cho cả ước lượng của các mômen bậc nhất và bậc hai. Điều này giúp giảm thiểu độ lệch của các ước lượng mômen, đặc biệt là trong giai đoạn đầu của quá trình huấn luyện. Để hiểu rõ hơn thì Adam được trình bày qua công thức sau:

Như ta đã biết SGDM cập nhật trọng số theo công thức sau:

$$w_t = w_t - \eta V_{dw_t}$$

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1 - \beta)\frac{\partial L}{\partial_{w_{t-1}}}$$

Và RMS Prop cập nhật trọng số như sau :

$$w_t = w_{t-1} - \eta'_t \frac{\partial L}{\partial_{w_{t-1}}}$$

$$\eta'_t = \frac{\eta}{\sqrt{w_{Avg(t)} + \varepsilon}}$$

$$w_{Avg(t)} = \beta w_{Avg(t-1)} + (1-\beta)(\frac{\partial L}{\partial_{w_{t-1}}})^2$$

Từ 2 công thức cập nhật trọng số trên Adam kết hợp cả 2 và ra một công thức cập nhật trọng số như sau:

$$w_t = w_{t-1} - \eta \frac{V_{dw_t}}{\sqrt{w_{Avg(t)} + \varepsilon}}$$

Nhìn chung, Adam là một thuật toán tối ưu hóa hiệu quả và linh hoạt và có thể được sử dụng cho nhiều loại bài toán máy học khác nhau. Các ưu điểm có thể kể đến như có tốc độ học tự thích nghi, thời gian hội tụ nhanh, và không cần phải điều chỉnh thủ công nhiều so với các optimizer khác do Adam ít nhạy cảm hơn với việc lựa chọn các siêu tham số.

Ngoài ra, Adam cũng có nhược điểm của riêng nó là sử dụng nhiều bộ nhớ do Adam lưu trữ hai thông tin quan trọng, đó là bình phương các gradient dùng để ước lượng mức độ biến động của gradient, giúp điều chỉnh tốc độ học phù hợp và mômen bậc nhất dùng để tính toán momentum, giúp tăng tốc độ hội tụ và tránh bị kẹt trong các điểm cực tiểu cục bộ. Khi xử lý dữ liệu rất lớn, lượng lưu trữ thông tin tăng lên đáng kể và có thể dẫn đến thiếu hụt bộ nhớ và xảy ra lỗi trong quá trình học. Bên cạnh đó, phương pháp này thiên về ưu tiên thời gian tính toán hơn trong khi các phương pháp như SGD thì ưu tiên về xử lý các điểm dữ liệu hơn nên SGD có thể tổng quát hóa dữ liệu tốt hơn nhưng phải đánh đổi với thời gian xử lý bị chậm lại.

### *1.6 So sánh các phương pháp*

| | Ưu điểm | Nhược điểm |
|---|---|---|
| SGD | Xử lý được trên một bộ dữ liệu rất lớn, độ phức tạp tính toán nhỏ | Quá trình hội tụ hay đạt cực tiểu chậm |
| SGD with momentum | Xử lý được trên một bộ dữ liệu rất lớn, độ phức tạp tính toán nhỏ đồng thời giảm được thời gian tính toán | Phải lưu ý trong việc chọn một momentum thích hợp |

| | | |
|---|---|---|
| Adagrad | Có khả năng điều chỉnh tốc độ học khác nhau ở mỗi lần học, quá trình hội tụ nhanh | Tốc độ học giảm đáng kể do lưu thông tin của tất cả các gradent trước. |
| RMS Prop | Khắc phục được nhược điểm của Adagrad sử dụng Exponential Weighted Moving Average(trung bình trượt số mũ) | Lưu ý chọn siêu tham số phù hợp với từng bài toán.<br>Thiếu hụt bộ nhớ nếu xử lý trên bộ dữ liệu quá lớn. |
| Adam | Thời gian xử lý nhanh, không tốn quá nhiều bộ nhớ và không cần phải điều chỉnh thủ công nhiều so với các optimizer khác | Uu tiên thời gian tính toán hơn hơn ưu tiên xử lý các điểm dữ liệu, tổng quát hóa dữ liệu kém hơn.<br>Thiếu hụt bộ nhớ nếu xử lý trên bộ dữ liệu quá lớn. |

Table 1. Bảng so sánh các phương pháp optimzer

## 2. Tìm hiểu về Continual Learning và Test Production

Continual learning và test production là hai kỹ thuật quan trọng trong học máy. Học liên tục giúp mô hình học hỏi và thích nghi với dữ liệu mới, trong khi test production giúp đảm bảo rằng mô hình hoạt động chính xác trong môi trường sản xuất.

Continual learning, còn được gọi là học liên tục hoặc học gia tăng, đề cập đến khả năng của mô hình học hỏi và thích nghi liên tục khi nhận được dữ liệu mới trong suốt vòng đời của nó. Không giống như các mô hình truyền thống được đào tạo trên tập dữ liệu cố định, các mô hình học liên tục có thể cải thiện hiệu suất và mở rộng cơ sở kiến thức của chúng bằng cách tích hợp thông tin mới theo thời gian đồng thời không quên đi những kiến thức mà đã được học trước đó rất lâu.

Trong khi đó test production là giai đoạn thử nghiệm này giúp đảm bảo mô hình hoạt động hiệu quả trong điều kiện thực tế, gặp phải dữ liệu người dùng thực tế và các trường hợp cạnh tiềm ẩn không được xem xét rõ ràng trong quá trình phát triển. Trong giai đoạn này có một chiến lược thử nghiệm model mà sẽ được đề cập ở bài toán MNIST dưới đây.

Trong bài toán nhận dạng chữ viết tay MNIST, giả sử chúng ta muốn xây dựng một ứng dụng ngân hàng điện tử có tính năng dùng camera nhận dạng séc. Để có thể nhận dạng được séc, ứng dụng cần nhận ra được các thông tin bằng chữ viết tay của người dùng có trong tờ séc đó. Do lượng khách hàng ngày càng tăng, chúng ta muốn ứng dụng có thể nhận dạng được càng nhiều phong cách chữ cũng như số viết tay từ nhiều người dùng khác nhau. Điều này giúp ứng dụng tránh gặp lỗi trong quá trình sử dụng nếu không nhận dạng được chữ viết tay. Để đạt được điều này, ứng dụng cần liên tục học hỏi mỗi khi xuất hiện người dùng mới. Bằng cách này, ứng dụng có thể cập nhật các mẫu chữ viết tay mới để hoạt động một cách chính xác nhất.

Dưới đây là hình minh họa một giải pháp học máy cho bài toán MNIST sử dụng continual learning:

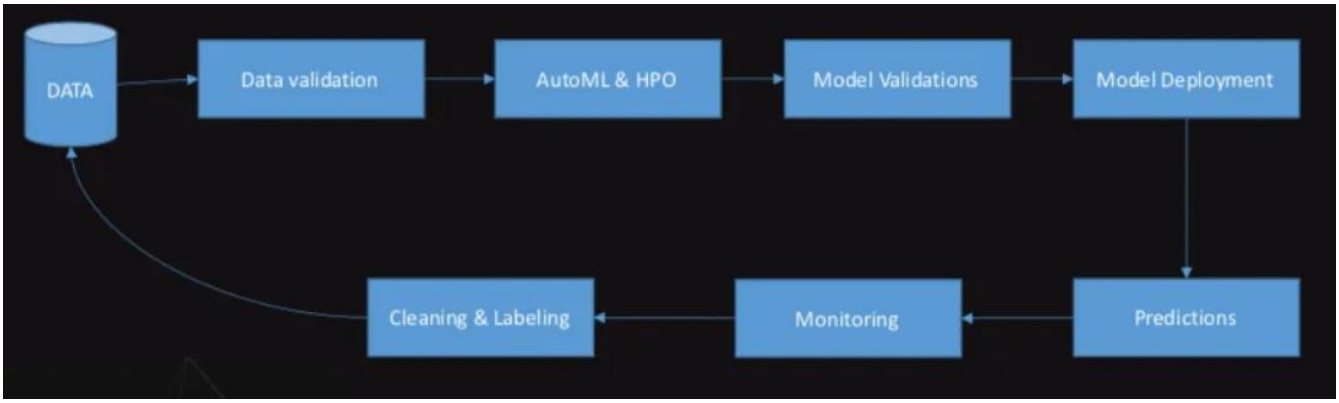


Figure 2. Giải pháp học máy cho bài toán MNIST

Với một bộ dữ liệu đầu vào đầu tiên chúng ta có bước validation có nghĩa là để kiểm tra dữ liệu, việc này có thể được thực hiện thông qua những bước tiền xử lý. Tiếp đến là bước AutoML & HPO, đây là một trong những bước quan trọng, ta có thể hiểu đơn giản đây là giai đoạn huấn luyện model. AutoML là một công cụ quan trọng trong việc áp dụng trong continual learning do chúng ta liên tục nhận được luồng dữ liệu mới. Ta có thể chỉ đơn giản tái huấn luyện thuật toán cũ với các tham số cũ, nhưng để đạt độ chính xác cao, AutoML là lựa chọn tốt hơn.

Auto ML không nhất thiết phải là phương pháp học tăng cường quá phức tạp, ta có thể chọn các model được xây dựng sẵn phù hợp và thực hiện fine-tune nó, như VGG, Inception, ResNet,… Sau đó phải liên tục theo dõi model trong suốt quá trình huấn luyện như là theo dõi loại thuật toán nào đang được sử dụng và sử dụng với các siêu tham số nào, mức tiêu thụ bộ nhớ, số liệu, độ chính xác,… Các dữ liệu theo dõi này có thể được dùng để tái triển khai và tái huấn luyện mô hình trong tương lai. Tự động hóa cơ sở hạ học máy là yếu tố then chốt của AutoML, đặc biệt trong lĩnh vực học sâu (deep learning). Bên cạnh đó, việc thiết lập môi trường chạy học sâu thường mất nhiều thời gian, bao gồm cài đặt CUDA, thư viện phụ thuộc, dữ liệu, mã code, v.v. Một trong những cách tự động hóa có thể kể đến là Kubernetes. Kubernetes là một hệ thống điều phối vùng chứa mã nguồn mở để tự động hóa việc triển khai, mở rộng quy mô và quản lý phần mềm nên việc dùng nó giúp quá trình tự động hóa của model trở nên dễ dàng hơn và giúp ta theo dõi được quá trình học.

Sau khi đã chọn và huấn luyện được mô hình phù hợp, bước tiếp theo là triển khai mô hình. Trong bước này, cần lưu ý việc triển khai mô hình phải duy trì được độ chính xác cao mà không gây ảnh hưởng tiêu cực đến người dùng. Để có được điều này thì như em đã đề cập ở trên là ta có một chiến lược thử nghiệm để giúp ích cho việc deploy model. Chiến lược này có tên là kỹ thuật triển khai Canary hay còn được biết đến là A/B testing. Đây là một kỹ thuật để triển khai phần mềm mới hoặc các thay đổi cho một lượng nhỏ người dùng trước khi thực hiện triển khai đầy đủ. Điều này cho phép thu thập dữ liệu thực tế về hiệu suất của thay đổi và khắc phục bất kỳ vấn đề nào trước khi ảnh hưởng đến tất cả người dùng. Cách thức hoạt động của nó là sẽ chọn một lượng nhỏ người dùng và triển khai phiên bản mới cho nhóm đó. Nếu phiên bản mới hoạt động tốt thì có thể dần dần triển khai nó cho các nhóm người dùng lớn hơn, cho đến khi tất cả người dùng đều nhận được phiên bản mới. Còn nếu không thì có thể nhanh chóng quay lại phiên bản cũ và khắc phục sự cố. Khi đã deploy được model thì bước tiếp theo chúng ta phải liên tục giám sát giải pháp học máy để phòng trường hợp model gặp lỗi trong quá trình chạy thì có cơ chế phù hợp để xử lý.

Cuối cùng, trong giải pháp học máy, chúng ta cần thiết lập một quy trình để giải pháp học máy có khả năng tự động nhận biết khi nào nên tự động huấn luyện lại mô hình của mình. Quy trình này cần được xác định bởi chúng ta, ví dụ như trong trường hợp bài toán MNIST, có thể là sau mỗi lần có một khách hàng mới đăng ký thành công ứng dụng. Lưu ý rằng mặc dù chúng ta thiết lập một quy trình để mô hình có thể tự động huấn luyện lại, nhưng cũng quan trọng để theo dõi và kiểm tra mỗi lần tự động huấn luyện để có thể xử lý kịp thời nếu phát hiện có lỗi.

# TÀI LIỆU THAM KHẢO

[1]: MIT Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville(chương 8)

[2]: Optimizers in Deep Learning: A Comprehensive Guide (analyticsvidhya.com)

[3]: Calculating Gradient Descent Manually | by Chi-Feng Wang | Towards Data Science

[4]: Deep Learning Optimizers. SGD with momentum, Adagrad, Adadelta… | by Gunand Mayanglambam | Towards Data Science.

[5]: Batch, Mini Batch & Stochastic Gradient Descent | by Sushant Patrikar | Towards Data Science

[6]: What is Continuous Machine Learning? (levity.ai)

[7]: How to apply continual learning to your machine learning models | by yochze | Towards Data Science