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: **NGUYỄN CHÍ ANH – 520H0335**

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: **NGUYỄN CHÍ ANH**
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)

Nguyễn Chí Anh

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

GD	Gradient Descent
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 Batch Gradient Descent

Batch Gradient descent hay còn gọi là suy giảm độ dốc được coi là một trong những optimizer phổ biến nhất dựa trên thuật toán gradient descent. Thuật toán của phương pháp này dùng vi tích phân để điều chỉnh các giá trị theo một hướng nhất quán từ đó đạt được cực tiểu địa phương. Cơ chế hoạt động của nó là sẽ lặp đi lặp lại quá trình cập nhật tham số của mô hình là weight và bias theo hướng dốc nhất, hay còn gọi là hướng cực tiểu hóa. Cách hoạt động của thuật toán như sau:

- Bắt đầu với một tập tham số ban đầu
- Tính gradient descent của hàm mất mát. Gradient là một vec-tơ chỉ theo hướng tăng trưởng lớn nhất của hàm.
- Cập nhật lại tham số bằng cách trừ đi một hệ số tỷ lệ của gradient. Hệ số tỷ lệ còn được gọi là tốc độ học(learning rate) và nó xác định kích thước của các bước mà thuật toán thực hiện
- Lặp lại quá trình 2 và 3 đến khi hội tụ hay đạt được đến cực tiểu địa phương.

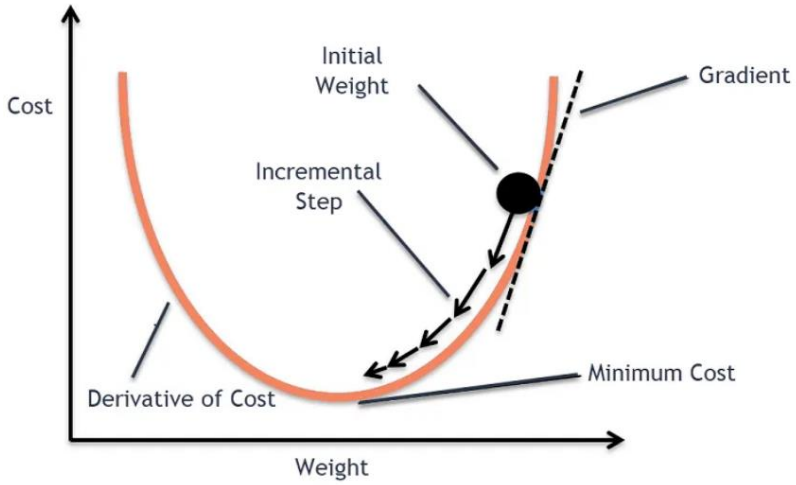

Figure 1. Minh họa thuật toán GD

Batch gradient descent optimizer có thể áp dụng được cho đa số các phương pháp học sâu. Tuy nhiên nó có nhược điểm là nếu 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. Bên cạnh đó batch GD hoạt động tốt đối với những hàm lồi thì đối với những hàm không lồi thì gradient descent có thể gặp khó trong việc tìm ra cực tiểu do không thể tính toán được nên đi bao xa theo độ dốc.

1.2 Stochastic Gradient Descent

Stochastic Gradient Descent hay còn gọi là suy giảm độ dốc ngẫu nhiên là một optimizer dựa trên biến thể của thuật toán Gradient Descent. Như ở phần 1.1 đã đề cập batch gradient descent có lẽ không phải là lựa chọn tốt để thực hiện trên một bộ dữ liệu lớn thì 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 batch GD thì nó sẽ xử lý từng dữ liệu một trong mỗi lần lặp. Do phương pháp optimizer này chỉ thực hiện tính toán trên một dữ liệu một lần nên độ mất mát cũng biến thiên theo trong mỗi lần học, vì thế mà SGD cần nhiều lần lặp hơn để đạt cực tiểu địa phương. Lần lặp nhiều hơn dẫn đến việc thời gian tính toán tăng theo, nhưng độ phức tạp tính toán vẫn thấp hơn GD optimizer. Như vậy nếu như chúng ta xử lý trên một bộ dữ liệu rất lớn và ưu tiên về mức độ phức tạp tính toán thì SGD optimizer sẽ phù hợp hơn GD optimizer.

1.3 Stochastic Gradient Descent with Momentum

Như ở phần 1.2 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 khiến cho độ mất mát biến thiên theo từng lần lặp, từ đó dẫn đến việc SGD tốn nhiều thời gian tính toán. Để khắc phục được tình trạng này thì ta có một optimizer gọi là Stochastic Gradient Descent with Momentum. Về cơ bản thì SGDM không khác gì SGD nhưng mà có thêm thuật ngữ

“Momentum” dịch ra là quán tính. Ý tưởng của thuật toán này là ở mỗi lần lặp nó sẽ tích lũy thêm một phần của lần cập nhật trước đó vào lần lặp hiện tại và điều này sẽ làm tăng nhanh quá trình hội tụ hay tìm ra điểm cực tiểu, đồng thời khắc phục được nhược điểm của SGD như đã đề cập. Trọng số sẽ được cập nhật qua công thức sau:

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

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

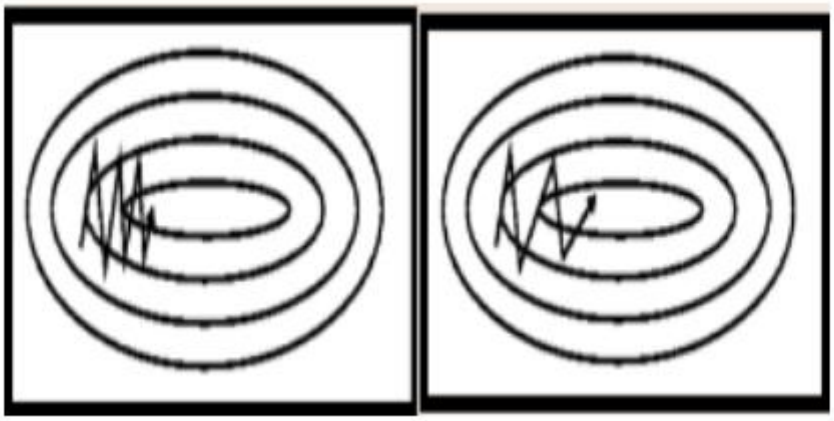

Figure 2. So sánh quá trình hội tụ giữa SGD(trái) và SGDM(phải)

Hai hình ở trên cho chúng ta thấy được quá trình hội tụ giữa SGD và SGDM, chúng ta có thể thấy SGDM có quá trình hội tụ nhanh hơn hẳn SGD. Nhưng nếu vậy thì có một câu hỏi đặt ra là như vậy chúng ta có thể đẩy nhanh hơn nữa quá trình hội tụ bằng cách tăng momentum và learning rate lên một cách đáng kể hay không?

Câu trả lời là không, 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. Như vậy đối với SGDM, điều thiết yếu là chúng ta phải chọn một momentum phù hợp dể tối đa hóa việc tính toán chính xác cũng như chú ý việc giảm learning rate nếu như chúng ta chọn momentum cao.

1.4 Mini Batch Gradient Descent

Mini batch gradient descent là một phương pháp optimizer kết hợp cả hai khái niệm từ batch GD và SGD. Trong phương pháp này, nó sẽ không thực hiện tính toán đối với toàn bộ tập dữ liệu hay đối với từng dữ liệu một, mà sẽ dùng một "batch" hay một tập các dữ liệu có kích thước bé hơn tập dữ liệu ban đầu để thực hiện tính toán, một tập các dữ liệu có kích thước bé hơn tập dữ liệu ban đầu ở đây được gọi là "mini batch". Với cách thực hiện như vậy phương pháp này giúp chúng ta có được các lợi thể của cả 2 phương pháp GD và SGD. Bên cạnh đó, mini-batch GD cũng bị một nhược điểm tương tự SGD đó là sự mất mát dao động liên tục qua mỗi batch được học bởi vì ta đang tính toán trên một phần các ví dụ học trong một lần lặp, tuy nhiên sự dao động này nếu so với SGD thì vẫn ít hơn nên phương pháp optimizer này đem lại cho chúng ta sự cân bằng giữa tốc độ và độ chính xác.

Ngoài ra, phương pháp này còn một nhược điểm nữa là nó cần một siêu tham số là kích thước của mini-batch được điều chỉnh làm sao để đạt được độ chính xác mong muốn. Mặc dù trong đa số trường hợp kích thước của mini-batch là 32, nhưng cũng có tồn tại một số trường hợp mà kích thước này cho ra độ chính xác rất kém.

1.5 Adagrad (Apdaptive Gradient Descent)

Đối với gradient descent và các biến thể của nó thì trong suốt quá trình huấn luyện các phương pháp này luôn dùng một tốc độ học không đổi. Tuy nhiên có một vấn đề được đặt ra, lấy ví dụ chúng ta đang cố cực tiểu hóa tham số weight với một tốc độ học không đổi là 0,1, thì tham số weight sẽ giảm dần đều 0,1 qua mỗi lần học đến khi đạt cực tiểu. Vậy thì có cách nào mà chúng ta có thể điều chỉnh được tốc độ học sao mà khi tham số weight ở xa điểm cực tiểu thì tốc độ học nhanh hơn để tiết kiệm thời gian, và khi weight ở càng gần điểm cực tiểu thì tốc độ học càng giảm để không bị bỏ lỡ điểm cực tiểu hay không? Để trả lời cho câu hỏi này chúng ta có phương pháp optimizer Adagrad.

Adagrad optimizer có khả năng điều chỉnh tốc độ học khác nhau ở mỗi lần lặp khác nhau bằng cách lưu thông tin của các tham số của lần tính gradient của lần trước

sau đó điều chỉnh tốc độ học sao cho phù hợp, tham số có giá trị đạo hàm từng phần lớn nhất của hàm loss thì sẽ có tốc độ học giảm dần và ngược lại.

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 có ý nghĩa là để lưu trữ 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.

Adagrad có ưu điểm là có thể tự động điều chỉnh được learning rate giúp chúng ta không phải điều chỉnh nó bằng cách thủ công, bên cạnh đó quá trình hội tụ khi sử dụng optimizer này nhanh hơn hẳn gradient descent và các biến thể. Tuy nhiên, nếu như việc tính tổng bình phương của các gradient phía trước mà cho ra giá trị rất cao thì khiến cho giá trị alpha t rất lớn, và điều đó khiến cho tốc độ học giảm đáng kể. Đến một lúc nào đó sẽ khiến cho việc tối ưu hóa các trọng số gần như không có thay đổi nhiều giữa các lần tối ưu khiến cho độ chính xác trong model bị giảm.

1.6 RMS Prop(Root Mean Square)

RMS prop là một phương pháp optimizer giải quyết tình trạng giảm dần tỷ lệ học của Adagrad. Như ở phần 1.5 đã đề cập đến, Adagrad sẽ lưu lại các gradient của các lần trước và sử dụng nó để điều chỉnh cho learning rate ở thời điểm hiện tại, nhưng nếu dần dần các gradient của các lần trước cộng lại có giá trị quá lớn thì 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 chia cho căn của tổng bình phương các gradeint với ép xi lông thì ý tưởng chính của RMS prop là nó sẽ dùng trung bình trượt số mũ (Exponential Weighted Moving Average) hay EWMA. EWMA chỉ chọn một vài gradient để tính từ các lần trước, 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. Để hiểu rõ hơn thì trọng số được cập nhật theo công thức sau:

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

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

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

Tuy nhiên trong EWMA có một siêu tham số β để điều chỉnh trọng số cho các gradient ở phía trước và việc lựa chọn siêu tham số sao cho phù hợp cho từng bài toán cần phải có sự xem xét kỹ càng. Chọn một siêu tham số quá lớn hay quá nhỏ sẽ làm ảnh hưởng đến tốc độ quá trình hội tụ, hay độ chính xác của thuật toán.

1.7 AdaDelta

Tương tự như RMSProp, phương pháp optimizer AdaDelta là một phương pháp khác để giải quyết tình trạng giảm dần tỷ lệ học của Adagrad với cơ chế tương tự RMSProp.

1.8 Adam

Phương pháp optimizer Adam hiện tại đang là một trong những phương pháp được dùng thường xuyên nhất. Phương pháp này hoạt động dựa trên cơ chế của cả 2 phương pháp SGDM và Ada delta, với ý tưởng là kết hợp khái niệm momentum từ SGDM và khả năng tự điều chỉnh tỷ lệ học từ RMS Prop. Để hiểu rõ hơn về sự kết hợp này thì ta sẽ làm rõ như 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:

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

Nhờ sự kết hợp độc đáo từ phương pháp optimizer RMS Prop và SGDM, Adam opimizer có rất nhiều các ưu điểm và hay được ứng dụng trong những mô hình học máy. Các ưu điểm có thể kể đến như dễ dàng sử dụng, 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. Tuy nhiên nếu Adam thật sự tuyệt vời như vậy sao chúng ta không dùng nó vào mọi mô hình mà phải cân nhắc các phương pháp khác như SGD, SGD with momentum, RMS Prop,… Đó là vì Adam cũng có nhược điểm của riêng nó, 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.

1.9 So sánh các phương pháp

	Ưu điểm	Nhược điểm
Batch Gradient descent	Ap dụng được cho đa số các phương pháp học sâu	Nếu kích thước của bộ dữ liệu quá lớn thì việc tính toán độ dốc sẽ rất phức tạp và tốn kém
SGD	Xử lý được trên một bộ dữ liệu rất lớn, độ phức tạp tính toán nhỏ	Thời gian tính toán cao

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
Mini batch Gradient Descent	Có tất cả các ưu điểm của cả 2 phương pháp Batch GD và SGD.	Lưu ý chọn kích thước của mini-batch phù hợp với từng bài toán
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 tham số phù hợp với từng bài toán
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.

Table 1. Bảng so sánh các phương pháp optimzer

2. Tìm hiểu về Continual Learning và Test Production

Khi nói về continual learning cho một giải pháp học máy nào đó là đang nói đến khả năng học liên tục dữ liệu của model. Khả năng này không phải tự nhiên mà có mà trên thực tế chúng ta phải lập trình model sao cho nó có thể tự động học và thích nghi khi có luồng dữ liệu mới được đưa vào. Ý tưởng đằng sau continual learning chính là dựa trên khả năng tiếp thu của con người. Xuyên suốt vòng đời, con người liên tục được tiếp cận kiến thức mới, sau đó hiểu và ghi nhớ và sau cùng là ứng dụng vào thực tiễn. Đối với học máy cũng vậy, khi ta cho ra một model hoàn chỉnh, ta mong muốn model có thể tự thích nghi được với những luồng dữ liệu mới. Mục tiêu của continual learning là để từ những dữ liệu đầu vào, model sẽ dùng những dữ liệu đó và tự động train lại chính bản thân nó để có thể đạt được độ chính xác cao hơn.

Còn test production đề cập đến một giai đoạn trong quá trình học liên tục của một giải pháp học máy. Giai đoạn này bao gồm một chiến lược thử nghiệm model mà em sẽ đề cập sau ở phần ví dụ dưới đây.

Ví dụ đối với bài toán nhận dạng chữ viết tay MNIST, giả sử chúng ta muốn viết ra một app ngân hàng điện tử có tính năng dùng camera nhận dạng séc để thực hiện nạp tiền vào tài khoản. Để có thể nhận dạng được séc thì model này phải nhận ra được các thông tin bằng chữ viết tay của người dùng có trong tờ séc đó. Do càng ngày lượng khách hàng càng tăng nên chúng ta muốn nhận dạng được càng nhiều phong cách chữ cũng như số viết tay từ nhiều người dùng khác nhau, để app không bị gặp lỗi trong quá trình sử dụng nếu không nhận dạng được chữ viết tay. Để đạt được điều này đòi hỏi model của ta phải liên tục học mỗi khi xuất hiện người dùng mới để có thể 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:

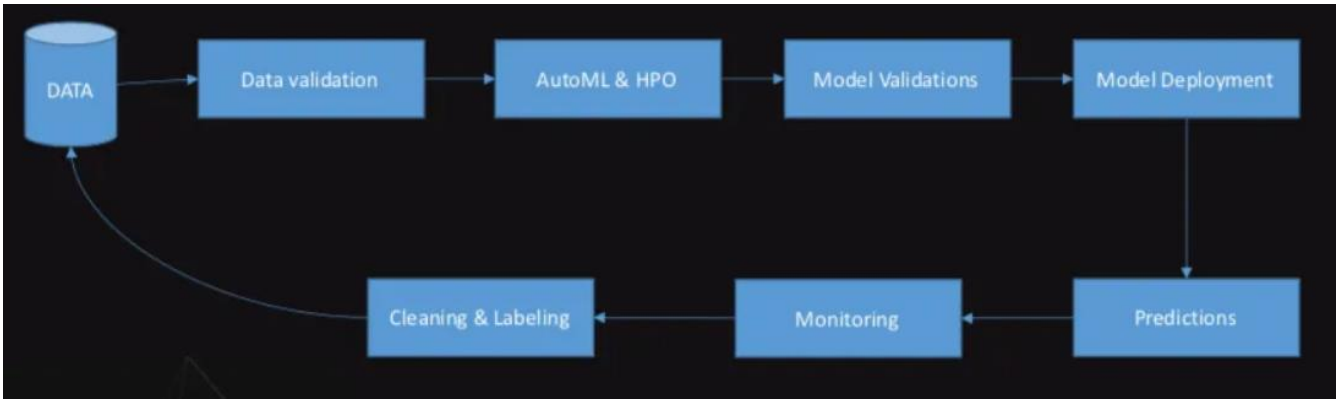

Figure 3. Giải pháp học máy cho bài toán MNIST

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.

Để huấn luyện model trước tiên ta phải chọn được một model phù hợp sau đó thực hiện fine-tune nó, một trong những model được dùng nhiều trong bài toán này có thể kể đến 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 có ích về sau trong việc re-train, re-deploy model.

Do để khởi động một mô hình deep learning từ đầu tốn nhiều thời gian nên một nhân tố quan trọng nữa trong bước AutoML này nó chính là phải tự động hóa cơ sở hạ tầng học máy, 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.

Một khi đã chọn và train được model thích hợp thì bước tiếp theo là bước deploy model. Trong bước này ta phải lưu ý việc deploy model phải duy trì được sự 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. Canary chia lượng user trong hệ thống ra làm hai phần, một phần sẽ được tiếp xúc với phiên bản mới nhất của phần mềm và một phần sẽ vẫn dùng phiên bản cũ và sẽ dần dần cập nhật tới tất cả người dùng . Cách này giúp người dùng có một cái nhìn tổng quan về phiên bản mới và có thời gian thích nghi, đồng thời nếu như phiên bản mới nhất có xảy ra lỗi thì sẽ chỉ có một phần người dùng bị ảnh hưởng chứ không ảnh hưởng đến toàn bộ. Bên cạnh đó Canary giúp ta đánh giá được các ưu nhược điểm của 2 phiên bản khi deploy thực tế để từ đó đưa ra các điều chỉnh phù hợp với model.

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 là tạo thủ tục để giải pháp học máy có thể biết khi nào đến thời điểm để có thể tự re-train lại chính bản thân và thời điểm này phải do chúng ta quyết định, có thể đối với bài toán MNIST thì thủ tục để retrain là sau khi có một khách hàng mới đăng ký thành công ứng dụng chẳng hạn. Lưu ý rằng chúng ta tạo ra một cái thủ tục để giải pháp học máy có thể tự retrain nhưng cũng phải nên theo dõi và kiểm tra sát sao sau mỗi lần tự retrain để có thể có những phương pháp xử lý kịp thời nếu 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