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

BÁO CÁO CUỐI KÌ

MÔN HỌC: NHẬP MÔN HỌC MÁY

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

Người thực hiện: **PHAN THỊ THÙY LINH – 52100697**

Lớp **: 21050401**

Khoá **: 25**

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 KÌ

MÔN HỌC: NHẬP MÔN HỌC MÁY

Mã môn học: 503044

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

Người thực hiện: **PHAN THỊ THÙY LINH – 52100697**

Lớp **: 21050401**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiền, em xin được gửi lời cảm ơn đến quý thầy cô khoa Công nghệ thông tin trường Đại học Tôn Đức Thắng đã đưa bộ môn vào giảng dạy và tạo điều kiện cho chúng em được tìm hiểu và học tập. Đặc biệt, em xin gửi lời cảm ơn đền thầy PGS.TS. Lê Anh Cường đã quan tâm, giúp đỡ, hướng dẫn tận tình cho chúng em trong quá trình giảng dạy. Các buổi học của thầy giúp cho chúng em có cái nhìn tổng quan về môn học Nhập môn Học máy.

Mặc dù đã được tiếp cận với những kiến thức cơ bản, nhưng sự tiếp thu của bản thân còn có những hạn chế nhất định. Do đó, trong quá trình hoàn thành bài báo cáo, còn tồn tại nhiều thiếu sót. Hy vọng sẽ nhận được sự góp ý từ thầy để bài báo cáo của em được hoàn thiện hơn.

Cuối cùng, em xin gửi lời chúc sức khỏe đến quý thầy cô, chúc thầy cô luôn thành công trong sự nghiệp trồng người vĩ đại của mình. Em xin chân thành cảm ơn!

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

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy PGS.TS. 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 đồ 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 đồ 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 22 tháng 12 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Linh

Phan Thị Thùy Linh

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

Đối với nội dung bài một, em tiến hành tìm hiểu, nghiên cứu và đưa ra đánh giá, so sánh các phương pháp Optimizer trong huấn luyện mô hình học máy. Ngoài ra, cũng tiến hành tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

MỤC LỤC

DANH MỤC CÁC CHỮ VIẾT TẮT

Stochastic Gradient Descent - SGD

Gradient Descent – GD

Support Vector Machines – SVM

Root Mean Square Propogation - RMSProp

Adaptive Moment Estimation - Adam

Knowlegde Base - KB

DANH SÁCH CÁC BẢNG BIỂU, HÌNH VẼ

PHẦN 1 – CÁC PHƯƠNG PHÁP OPTIMIZER

1.1 Tìm hiểu các phương pháp Optimizer:

Mô hình học máy ngày càng trở nên quan trọng trong nhiều lĩnh vực, từ thị giác máy tính đến xử lý ngôn ngữ tự nhiên. Một yếu tố quyết định quan trọng trong quá trình huấn luyện là bộ tối ưu hóa (Optimizer), giúp tối ưu hóa hàm mất mát và cập nhật trọng số mô hình. Dưới đây là một số phương pháp Optimizer phổ biến.

1.1.1 Stochastic Gradient Descent:

Stochastic Gradient Descent (SGD) là một phương pháp tối ưu hóa thường được sử dụng trong quá trình huấn luyện mô hình học máy. Nó là một biến thể của Gradient Descent (GD).

Gradient Descent là một trong những thuật toán thông dụng khi tối ưu hóa Neural Network.

Gradient Descent tối thiểu hóa hàm loss function f (θ) trong đó θ là tập hợp các trọng số của mô hình cần tối ưu. Quy tắc cập nhật của GD ở dạng tổng quát sau:

$$\theta_{t+1} = \theta_t - \eta \Delta_\theta f(\theta_t)$$

Với $\Delta_\theta f(\theta)$ là đạo hàm của hàm số tại một điểm θ bất kì; n là learning rate.

Có một số biến thể khác nhau của GD tùy thuộc vào số lượng dữ liệu được sử dụng để tính gradient của loss function. Batch Gradient Descent là phương pháp tối ưu hóa cơ bản, sử dụng toàn bộ tập dữ liệu đào tạo để tính gradient và cập nhật trọng số mô hình. Tuy nhiên, thuật toán chỉ phù hợp cho dữ liệu nhỏ và không gian trạng thái bộ nhớ lớn. Để khắc phục nhược điểm đó, thuật toán Stochastic Gradient Descent thực hiện việc cập nhật trọng số với mỗi mẫu dữ liệu $x^{(i)}$ có nhãn tương ứng $y^{(i)}$ như sau:

$$\theta_{t+1} = \theta_t - \eta \Delta_\theta J(\theta_t; x^{(i)}; y^{(i)})$$

Bằng cách cập nhật này, SGD thường nhanh hơn Batch GD và có thể sử dụng để học trực tuyến khi tập dữ liệu huấn luyện được cập nhật liên lục. Bộ trọng số θ được cập nhật thường xuyên hơn so với Batch GD, do đó loss function cũng dao động nhiều hơn.

Sự dao động này làm SGD có vẻ không ổn định nhưng lại có điểm tích cực là nó giúp di chuyển đến những điểm cực tiểu địa phương mới có tiềm năng hơn. Với tốc độ học giảm, khả năng hội tụ của SGD cũng tương đương với Batch GD.

Trong một số trường hợp, SGD được hiểu là mini-batch GD. Đây là thuật toán kết hợp giữa Batch Gradient Descent và SGD bằng cách sử dụng một số lượng nhỏ mẫu dữ liệu (mini-batch) trong mỗi lần cập nhật. Điều này giúp giảm tính biến động của gradient so với SGD và có thể cỉa thiện độ ổn định của quá trình đào tạo.

- Ưu điểm:
  - Hiệu suất với dữ liệu lớn: Hoạt động hiệu quả trên các tập dữ liệu lớn vì chỉ cần sử dụng một mẫu ngẫu nhiên trong mỗi lần cập nhật thay vì toàn bộ dữ liệu.
  - Khả năng học Online: Phù hợp cho việc học trực tuyến khi dữ liệu cập nhật liên tục.
  - Khả năng vượt qua địa phương tối ưu: Có thể "vượt qua" các địa phương tối ưu hóa và giúp mô hình thoát khỏi điểm tối ưu cục bộ.
  - Tính tương thích dữ liệu phi tuyến: Được biết đến là hiệu quả khi áp dụng cho các mô hình có dữ liệu phi tuyến.
  - Tính tổng quát: Phù hợp cho nhiều loại mô hình học máy và học sâu.
- Nhược điểm:
  - Khả năng dao động lớn: SGD có thể dao động nhiều do sự biến động của mẫu ngẫu nhiên, đặc biệt là khi kích thước mẫu nhỏ.
  - Đòi hỏi tinh chỉnh thêm: Cần tinh chỉnh thêm hyperparameters như learning rate để đảm bảo hội tụ và hiệu suất tốt.
  - Không ổn định: Không ổn định trên các hàm mất mát không liên tục hoặc có nhiều điểm tối ưu cục bộ.
  - Learning Rate là vấn đề: Cần lựa chọn kích thước learning rate phù hợp để tránh hiện tượng overshooting hoặc diverging.

- Tính không đồng đều trong việc cập nhật tham số: Các tham số có thể được cập nhật không đồng đều, đặc biệt là khi sử dụng kích thước mini-batch lớn.

- Ứng dụng:

Stochastic Gradient Descent (SGD) thường được áp dụng cho nhiều loại bài toán trong machine learning, đặc biệt là trong các bài toán tối ưu hóa tham số của mô hình. Cụ thể, SGD thường được sử dụng trong các bài toán như:

- Học máy (Machine Learning) Supervised:
  - Hồi quy tuyến tính (Linear Regression): SGD có thể được sử dụng để tối ưu hóa các tham số của mô hình hồi quy tuyến tính.
  - Phân loại (Classification): Cả trong bài toán phân loại như Logistic Regression và các mô hình phức tạp hơn như Support Vector Machines (SVM), Neural Networks, SGD đều được sử dụng.
- Học máy không giám sát (Unsupervised Learning):
  - Phân cụm (Clustering): Trong các thuật toán như K-Means, SGD có thể được sử dụng để tối ưu hóa các trọng số của các tâm cụm.
  - Phân loại không giám sát (Self-supervised learning): SGD cũng có thể được áp dụng để tối ưu hóa các tham số của mô hình trong các bài toán như autoencoders.
- Học máy tăng cường (Reinforcement Learning):
  - Học theo phần thưởng (Reward-based Learning): Trong các mô hình học tăng cường, SGD có thể được sử dụng để cập nhật trọng số của chính sách hành động để tối ưu hóa hàm phần thưởng.
- Mạng nơ-ron (Neural Networks):
  - Học sâu (Deep Learning): Trong các mô hình nơ-ron sâu, SGD và các biến thể như Mini-batch SGD thường được sử dụng để cập nhật trọng số của mạng nơ-ron.
- Học máy chuyển giao (Transfer Learning):

- Fine-tuning: SGD có thể được sử dụng để fine-tune (tinh chỉnh) các mô hình đã được huấn luyện trước đó trên tập dữ liệu mới hoặc trong các bài toán khác nhau.

SGD giúp giảm bớt độ phức tạp tính toán so với các phương pháp tối ưu hóa toàn bộ dữ liệu (batch optimization), đặc biệt là khi tập dữ liệu lớn. Thay vì cập nhật trọng số dựa trên toàn bộ dữ liệu, SGD chỉ cập nhật dựa trên một mẫu dữ liệu ngẫu nhiên (stochastic). Điều này làm giảm bớt bộ nhớ cần thiết và tăng tốc quá trình học.

1.1.2 RMSProp (Root Mean Square Propogation):

RMSProp là một thuật toán tối ưu hóa thường được sử dụng để huấn luyện mô hình học máy. Nó là một biến thể của Stochastic Gradient Descent (SGD) được thiết kế để điều chỉnh tỷ lệ học (learning rate) tự động để cải thiện hiệu suất của mô hình.

- Cơ chế hoạt động: RMSProp giữ một trung bình gia quyền (exponentially weighted moving average - EMA) của bình phương gradient trước đó.

Cụ thể, tại mỗi bước thời gian t, RMSProp tính toán v_t , bình phương có trọng số của gradient theo công thức:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2$$

Trong đó v_t là trung bình gia quyền tại thời điểm t, g_t là gradient tại thời điểm t và b là hệ số giảm trọng số, thường có giá trị khoảng 0.9.

- Cập nhật trọng số: Tỷ lệ học (learning rate) của RMSProp được cập nhật tại mỗi bước theo công thức:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \varepsilon} \odot g_t$$

Trong đó θ_t là vector trọng số tại thời điểm t, n là tỷ lệ học, v_t là trung bình gia quyền, e là một giá trị nhỏ (thường được chọn là 10-8) để tránh chia cho 0.

So với SGD, RMSProp có khả năng điều chỉnh tỷ lệ học tự động. Đối với AdaGrad, RMDProp giảm bớt hiệu ứng của lịch sử gradient bằng cách sử dụng trung bình gia quyền.

- Ưu điểm:
  - Điều chỉnh tự động tỷ lệ học (Learning Rate): RMSProp có khả năng tự động điều chỉnh tỷ lệ học dựa trên trung bình gia quyền của bình phương gradient. Điều này giúp ổn định và tối ưu hóa quá trình học mà không yêu cầu người lập trình chọn một tỷ lệ học cố định.
  - Hiệu suất trên dữ liệu Non-Stationary: RMSProp thường hoạt động hiệu quả trên các tập dữ liệu có tính không đồng nhất theo thời gian (non-stationary) hoặc dữ liệu thưa thớt. Điều này là do nó giảm bớt ảnh hưởng của gradient lớn và giúp mô hình thích ứng linh hoạt với biến động.
  - Giảm nguy cơ dao động (Oscillation): RMSProp giảm khả năng dao động trong quá trình học, đặc biệt là so với các thuật toán như AdaGrad.
  - Dễ cài đặt: Thuật toán này không có quá nhiều tham số để cài đặt, giúp đơn giản hóa quá trình lựa chọn hyperparameter.
- Nhược điểm:
  - Khả năng mắc kẹt ở cực tiểu cục bộ: Trong một số trường hợp, RMSProp có thể mắc kẹt ở cực tiểu cục bộ và không thể thoát ra do tỷ lệ học tự động giảm quá nhanh.
  - Yêu cầu lập trình viên lựa chọn thêm tham số: Mặc dù ít tham số hơn so với một số thuật toán khác, nhưng vẫn cần lựa chọn giá trị cho β và ϵ, và những giá trị này có thể ảnh hưởng đến hiệu suất.
  - Khả năng Overfitting: Trong một số trường hợp, RMSProp có thể dẫn đến việc overfitting, đặc biệt là khi tập dữ liệu đào tạo nhỏ hoặc khi mô hình quá phức tạp.
  - Chưa hoàn toàn phù hợp cho tất cả các bài toán: Mặc dù hiệu suất tốt trên nhiều bài toán, nhưng không phải tất cả mọi bài toán đều được cải thiện bởi RMSProp. Có những trường hợp nơi các thuật toán khác có thể hoạt động tốt hơn.

- Ứng dụng:

RMSProp (Root Mean Square Propagation) là một phương pháp tối ưu hóa được sử dụng trong quá trình đào tạo mô hình máy học. Phương pháp này thường được áp dụng cho các bài toán tối ưu hóa trong machine learning, đặc biệt là trong việc đào tạo các mạng nơ-ron sâu (deep neural networks).

RMSProp giúp giảm vấn đề của learning rate đổi đột ngột và quá lớn trong quá trình tối ưu hóa. Nó sử dụng ý tưởng của việc điều chỉnh learning rate cho từng tham số của mô hình dựa trên độ lớn của gradient tương ứng với tham số đó. Điều này giúp ổn định quá trình học và giảm khả năng overshooting (quá mức) hay undershooting (thiếu mức) của learning rate.

RMSProp tính toán một trung bình độ lớn của bình phương của gradient trước đó (với trọng số giảm dần theo thời gian), và sử dụng nó để điều chỉ learning rate. Điều này giúp mô hình tự động thích ứng với dữ liệu và tăng cường khả năng học trong các tình huống khác nhau.

Vì RMSProp giải quyết một số vấn đề liên quan đến learning rate trong quá trình đào tạo mạng nơ-ron sâu, nó thường được ưa chuộng trong các bài toán liên quan đến deep learning, chẳng hạn như nhận diện hình ảnh, dịch máy, và các nhiệm vụ khác mà yêu cầu đào tạo mô hình sâu.

1.1.3 Adagrad:

Adagrad cũng là một thuật toán để tối ưu hóa dựa trên gradient descent, thuật toán này giúp điều chỉnh tốc độ học ứng với các tính năng khác nhau, thực hiện các cập nhật nhỏ hơn (tức là tốc độ học thấp) cho các tham số liên quan đến các tính năng thường xuyên xuất hiện và các cập nhật lớn hơn (tức là tốc độ học cao) cho các tham số liên quan đến các tính năng không thường xuyên. Vì lý do này, Adagrad rất thích hợp để xử lý dữ liệu thưa thớt. Một vài dẫn chứng về hiệu quả của Adagrad là Dean đã phát hiện ra rằng Adagrad đã cải thiện đáng kể độ bền của SGD và sử dụng nó để đào tạo mạng lưới thần kinh quy mô lớn tại Google, Pennington đã sử dụng Adagrad để đào tạo nhúng từ

GloVe vì những từ không thường xuyên xuất hiện yêu cầu cập nhật lớn hơn nhiều so với những từ xuất hiện thường xuyên.

Bình phương Gradient: Bình phương mỗi thành phần của gradient. Điều này có nghĩa là với mỗi tham số θ_i tính toán $g_{t,i}^2$ là bình phương của gradient $\nabla J(\theta_t)_i$ tại thời điểm t.

Quy tắc cập nhật Adagrad:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Theo quy tắc cập nhật, Adagrad điều chỉnh tốc độ học n tại bước t tương ứng với trong số θ_i xác định dựa trên các gradienr đã tính được theo θ_i. Mẫu số là chuẩn L2 của ma trận đường chéo G_t trong đó phần tử i,i là tổng bình phương của các gradient tương ứng với θ_i tính đến bước t. e là một số dương khó nhỏ nhằm tránh trường hợp mẫu số bằng 0.

- Ưu điểm:

- Adaptive Learning Rate: Tính chất quan trọng nhất của Adagrad là khả năng tự động điều chỉnh tỷ lệ học cho từng tham số. Nó giúp giảm tỷ lệ học đối với các tham số có gradient lớn và tăng tỷ lệ học đối với các tham số có gradient nhỏ, giúp cân bằng tốt hơn.
- Phù hợp cho dữ liệu thưa (Sparse Data): Hiệu quả khi làm việc với dữ liệu thưa vì nó tự động điều chỉnh tỷ lệ học cho từng tham số dựa trên lịch sử gradient.
- Dễ cài đặt: Thuật toán Adagrad có thể cài đặt một cách đơn giản và không đòi hỏi nhiều hyperparameters.

- Nhược điểm:

- Accumulation of Squared Gradients: Một vấn đề của Adagrad là tích tụ bình phương của gradient theo thời gian. Điều này có thể dẫn đến tổng quát quá lớn của các bình phương gradient, làm cho tỷ lệ học giảm đáng kể về sau.

- Khả năng dừng sớm (Early Stopping): Vì tỷ lệ học giảm theo thời gian, có thể dẫn đến dừng sớm của quá trình đào tạo trước khi đạt được tối ưu tốt.
- Không hiệu quả cho dữ liệu Non-Stationary: Adagrad không hiệu quả cho các dữ liệu không ổn định theo thời gian, vì nó giả định rằng tất cả các tham số cần tỷ lệ học khác nhau, trong khi thực tế có thể có sự biến động.
- Memory Requirement: Do tích tụ bình phương gradient, Adagrad có thể đòi hỏi bộ nhớ lớn và có thể trở nên không hiệu quả cho các mô hình với số lượng tham số lớn.
- Khả năng chia sẻ mô hình: Không phù hợp cho các mô hình được chia sẻ (shared model) vì mỗi tham số có thể có tỷ lệ học khác nhau.

- Ứng dụng:

Thuật toán tối ưu hóa Adagrad thường được áp dụng cho các bài toán học máy khi có đặc trưng (feature) không đồng nhất về mức độ quan trọng. Đặc biệt, nó thường được sử dụng trong các bài toán:

Xử lý ngôn ngữ tự nhiên (NLP): Trong xử lý ngôn ngữ tự nhiên, các từ ngữ thường có độ quan trọng khác nhau đối với mô hình. Adagrad giúp tự động điều chỉnh learning rate (tốc độ học) của từng tham số theo mức độ quan trọng của chúng.

Học máy và học sâu với dữ liệu thưa (sparse data): Khi bạn có một lượng lớn các đặc trưng, nhưng mỗi mẫu dữ liệu chỉ có một số ít đặc trưng được kích thích, Adagrad có thể giúp mô hình học hiệu quả từ các đặc trưng quan trọng.

Bài toán đa lớp (multiclass classification): Khi phải xử lý nhiều lớp, có thể có sự chênh lệch về độ quan trọng giữa các lớp. Adagrad có thể giúp tối ưu hóa cho các tham số của mỗi lớp một cách hiệu quả.

Tuy nhiên, cũng cần lưu ý rằng Adagrad có một số nhược điểm, như việc giảm learning rate quá nhanh theo thời gian, có thể dẫn đến vấn đề của việc học quá mức (overshooting). Do đó, một số biến thể như Adadelta và RMSprop đã được phát triển để khắc phục một số vấn đề của Adagrad.

1.1.4 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 với các tọa độ. Hơn nữa, 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 căn chỉnh lượng thay đổi trong tương lai.

Trong quá trình thực hiện, thay vì luuw trữ w bình phương của gradient theo cách thông thường, Adadalta thực hiện tích lũy dưới dạng mô-men bậc 2 của gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

Công thức trên thể hiện trung bình các gradienr $E[g^2]$, ở bước t phụ thuộc vào trung bình xác gradient $E[g^2]_{t-1}$ ở bước t-1 và gradient g_t ở bước t. Hệ số γ thường có giá trị bằng 0.9 với ý nghĩa rằng gradient ở hiện tại sẽ phụ thuộc phần lớn vào gradient ở các bước trước đó.

- Tính toán RMS (Root Mean Square): Adadelta tính toán RMS của trung bình công bình phương gradient:

  $$RMS_t = \sqrt{E[g^2]_t + \varepsilon}$$

  Trong đó: ε là một giá trị nhỏ được thêm vào để tránh chia cho 0.
- Tính toán tỷ lệ học mới: Tính toán tỷ lệ học mới dực trên RMS và lịch sử delta tham số:

  $$Learning\ Rate_t = \frac{RMS_{t-1}}{RMS_t}$$

  Các bước cập nhật tham số được thực hiện như sau:

  $$\theta_{t+1} = \theta_t - Learning\ rate_t . g_t$$

  Trong đó θ_t là tham số tại thời điểm t.
- Lịch sử Delta tham số: Tính toán trung bình cộng trọng số của bình phương delta tham số (độ chênh lệch giữa các bước cập nhật):

  $$E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2$$

  Trong đó: $\Delta\theta_t = -Learning\ rate\ . g_t$

- Cập nhật delta tham số:

$$\Delta\theta_{t+1} = \frac{-RMS_{t-1}}{RMS_t} . g_t$$

- Ưu điểm:
  - Khả năng tự điều chỉnh Learning Rate: Tự động điều chỉnh tỷ lệ học cho mỗi tham số mà không cần phải lựa chọn tỷ lệ học trước.
  - Khả năng phù hợp cho dữ liệu Non-Stationary: Hiệu quả cho các dữ liệu không ổn định theo thời gian.
  - Giảm tích tụ bình phương Gradient: Giảm vấn đề tích tụ bình phương gradient, giúp tránh việc giảm động của tỷ lệ học theo thời gian.
- Nhược điểm:
  - Yêu cầu thêm Hyperparameters: Adadelta có thêm một hyperparameter p so với Adagrad, tăng độ phức tạp của việc lựa chọn hyperparameters.
  - Dễ bị quá mức thực hiện cập nhật: Có thể dẫn đến việc cập nhật quá mức trong mô hình với nhiều tham số.
- Ứng dụng:

Adadelta là một thuật toán tối ưu hóa được thiết kế để cải thiện hiệu suất của các mô hình máy học trong quá trình đào tạo. Nó thường được sử dụng cho bài toán tối ưu hóa trong các mô hình học sâu (deep learning). Đặc biệt, Adadelta thường được ưa chuộng khi làm việc với các mô hình sử dụng hàm mất mát phức tạp và có số lượng lớn các tham số cần được tối ưu hóa.

Thuật toán này được thiết kế để giảm độ nhạy cảm của tỷ lệ học (learning rate) đối với các tham số và giảm độ phụ thuộc vào việc lựa chọn tỷ lệ học thích hợp. Điều này làm cho Adadelta trở nên mạnh mẽ hơn trong việc tự điều chỉnh tỷ lệ học một cách tự động.

Do đó, Adadelta thường được áp dụng trong các bài toán đòi hỏi sự linh hoạt và hiệu quả cao của thuật toán tối ưu hóa, đặc biệt là trong lĩnh vực của học sâu (deep learning), nơi mà việc huấn luyện các mô hình với số lượng lớn các tham số là phổ biến.

1.1.5 Adam - Adaptive Moment Estimation:

Adam được xem như là sự kết hợp của RMSprop và Stochastic Gradient Descent với động lượng. Adam là một phương pháp tỉ lệ học thích ứng, nó tính toán tỉ lệ học tập cá nhân cho các tham số khác nhau. Adam sử dụng ước tính của khoảng thời gian thứ nhất và thứ hai của độ dốc để điều chỉnh tỉ lệ học cho từng trọng số của mạng nơ-ron.

Tuy nhiên, qua nghiên cứu thực nghiệm, trong một số trường hợp, Adam vẫn còn gặp phải nhiều thiếu sót so với thuật toán SGD. Thuật toán Adam được mô tả:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Trong đó: g_t là gradient tại bước thời gian t.

m_t là first moment của gradient.

v_t là second moment của gradient.

β_1 và β_2 là các hệ số giảm momentums (thường được đặt giá trị mặc định là 0.9 và 0.999).

Nếu khởi tạo m_t và v_t là các vector 0, các giá trị này có khuynh hướng nghiêng về 0, đặc biệt là khi β1 và β2 xấp xỉ bằng 1. Do vậy, để khắc phục, các giá trị này được ước lượng bằng cách:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Sau đó cập nhật các trọng số theo công thức:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t, \varepsilon \, th\grave{u}\grave{o}ng \, b\check{a}ng \, 10^{-8}$$

Trong đó: θ_t là tham số cần cập nhật; anphal là tốc độ học; e là một giá trị nhỏ để tránh chia cho 0.

Để việc descent được thực hiện nhanh hơn, thuật toán đã sử dụng hai kỹ thuật:

- Tính exponential moving average của giá trị đạo hàm lưu vào biến m và sử dụng nó là tử số của việc cập nhật hướng. Với ý nghĩa là nếu m có giá trị lớn, thì việc descent đang đi đúng hướng và chúng ta cần bước nhảy lớn hơn để đi nhanh hơn. Tương tự, nếu giá trị m nhỏ, phần descent có thể không đi về hướng tối tiểu và chúng ta nên đi 1 bước nhỏ để thăm dò. Đây là phần momentum của thuật toán.

- Tính exponential moving average của bình phương gía trị đạo hàm lưu vào biến v và sử dụng nó là phần mẫu số của việc cập nhật hướng. Với ý nghĩa như sau: Giả sử gradient mang các giá trị dương, âm lẫn lộn, thì khi cộng các giá trị lại theo công thức tính m ta sẽ được giá trị m gần số 0. Do âm dương lẫn lộn nên nó bị triệt tiêu lẫn nhau. Nhưng trong trường hợp này thì v sẽ mang giá trị lớn. Do đó, trong trường hợp này, chúng ta sẽ không hướng tới cực tiểu, chúng ta sẽ không muốn đi theo hướng đạo hàm trong trường hợp này. Chúng ta để v ở phần mẫu vì khi chia cho một giá trị cao, giá trị của các phần cập nhật sẽ nhỏ, và khi v có giá trị thấp, phần cập nhật sẽ lớn. Đây chính là phần tối ưu RMSProp của thuật toán.

- Ưu điểm:
    - Hiệu suất cao ban đầu: Adam thường cho hiệu suất tốt ngay từ giai đoạn đầu của quá trình huấn luyện, đặc biệt là khi sử dụng trong các mô hình học máy sâu.
    - Tự điều chỉnh tỷ lệ học (learning rate): Adam tự động điều chỉnh tỷ lệ học cho từng tham số riêng lẻ, giúp nhanh chóng hội tụ và giảm khả năng bị mắc kẹt ở điểm cực tiểu cục bộ.

- Hiệu quả với dữ liệu lớn và nhiều tham số: Adam thường hoạt động hiệu quả với các mô hình có lượng dữ liệu lớn và nhiều tham số.
- Tính chất động (adaptive): Adam sử dụng thông tin từ gradient cũ để điều chỉnh trạng thái hiện tại, giúp mô hình linh hoạt và chống lại vấn đề của các phương pháp có tỷ lệ học cố định.

- Nhược điểm:
  - Yêu cầu bộ nhớ: Adam cần lưu trữ thêm thông tin như gradient bình phương trung bình và độ nghiêng bình phương trung bình cho mỗi tham số, làm tăng yêu cầu về bộ nhớ.
  - Khả năng quá mức cập nhật: Có thể xảy ra trường hợp Adam cập nhật quá mức, dẫn đến những độ lớn của trọng số trở nên quá lớn, đôi khi làm giảm hiệu suất.
  - Không ổn định trên một số bài toán: Có những bài toán nơi Adam không hoạt động tốt và thậm chí có thể làm tăng độ biến động của quá trình học.
  - Cần lựa chọn thủ công cho tỷ lệ học khởi tạo (learning rate): Mặc dù Adam có khả năng tự điều chỉnh tỷ lệ học, nhưng vẫn cần lựa chọn tỷ lệ học khởi tạo thủ công, và chọn sai có thể ảnh hưởng đến hiệu suất của thuật toán.
- Ứng dụng:

Adaptive Moment Estimation (Adam) là một thuật toán tối ưu hóa được thiết kế để cập nhật trọng số của mô hình trong quá trình huấn luyện mạng nơ-ron trong machine learning. Adam thường được áp dụng cho nhiều loại bài toán trong machine learning, bao gồm:

- Học máy sâu (Deep Learning): Adam thường được sử dụng trong các mô hình học máy sâu như mạng nơ-ron đa tầng (Multilayer Perceptron), mạng nơ-ron tích chập (Convolutional Neural Networks), và mạng nơ-ron hồi quy (Recurrent Neural Networks).

- Phân loại (Classification): Adam có thể được áp dụng cho các bài toán phân loại, nơi mục tiêu là dự đoán lớp của một điểm dữ liệu.
- Học máy không giám sát (Unsupervised Learning): Adam cũng có thể được sử dụng trong các bài toán không giám sát như phân cụm (clustering) hoặc giảm chiều dữ liệu (dimensionality reduction).
- Học máy tăng cường (Reinforcement Learning): Trong một số trường hợp, Adam có thể được sử dụng để tối ưu hóa chính sách trong các mô hình học máy tăng cường.
- Học máy chuyển giao (Transfer Learning): Adam thường được tích hợp vào các phương pháp học máy chuyển giao để cập nhật trọng số của mô hình khi chúng được chuyển giao từ một tác vụ học tập sang một tác vụ khác.
- Adam cung cấp khả năng tự điều chỉnh tỷ lệ học (learning rate) cho từng tham số riêng lẻ của mô hình, giúp cải thiện khả năng tối ưu hóa và ổn định quá trình huấn luyện. Tuy nhiên, việc lựa chọn thuật toán tối ưu hóa phụ thuộc vào nhiều yếu tố như đặc tính của dữ liệu, kích thước mô hình, và yêu cầu của bài toán cụ thể.

1.2 Áp dụng vào bài toán cụ thể:

Tiến hành khảo sát và đánh giá các thuật toán tối ưu với bài toán phân loại hình ảnh trên tập cơ sở dữ liệu MNIST. Bộ dữ liệu MNIST là bộ dữ liệu gồm các hình ảnh xám (grayscale picture) các chữ số viết tay được chia sẻ bởi Yann Lecun bao gồm 70000 ảnh chữ số viết tay được chia thành 2 tập: tập huấn luyện gồm 60000 ảnh và tập kiểm tra 10000 ảnh. Các chữ số viết tay ở tập MNIST được chia thành 10 nhóm tương ứng với các chữ số từ 0 đến 9. Tất cả hình ảnh trong tập MNIST đều được chuẩn hóa với kích thước 28 x 28 điểm ảnh. Dưới đây là một số hình ảnh được trích xuất từ bộ dữ liệu.

Sử dụng mô hình convolutional neural network đơn giản để huấn luyện và đánh giá với chu kì học là 10 và áp dụng trên các thuật toán SGD, RMSprop, Adagrad,

AdaDelta, Adam. Phương pháp thực hiện đánh giá kết quả sử dụng trong bài là loss function và tỉ lệ nhận dạng đúng trên các tập dữ liệu được xét.

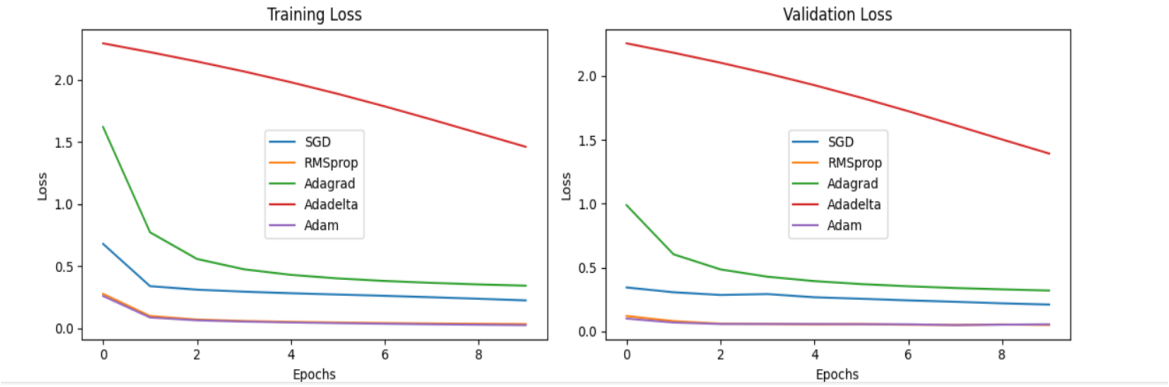
```python
# Define a simple convolutional neural network model
def create_model(optimizer):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(10, activation='softmax'))
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Hình 1. 1 CNN model

```python
# Train and evaluate the model for each optimizer
optimizers = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam']
history_dict = {}

for optimizer_name in optimizers:
    model = create_model(optimizer_name)
    history = model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_data=(test_images, test_labels))
    history_dict[optimizer_name] = history.history
```

Hình 1. 2 Huấn luyện và đánh giá mô hình

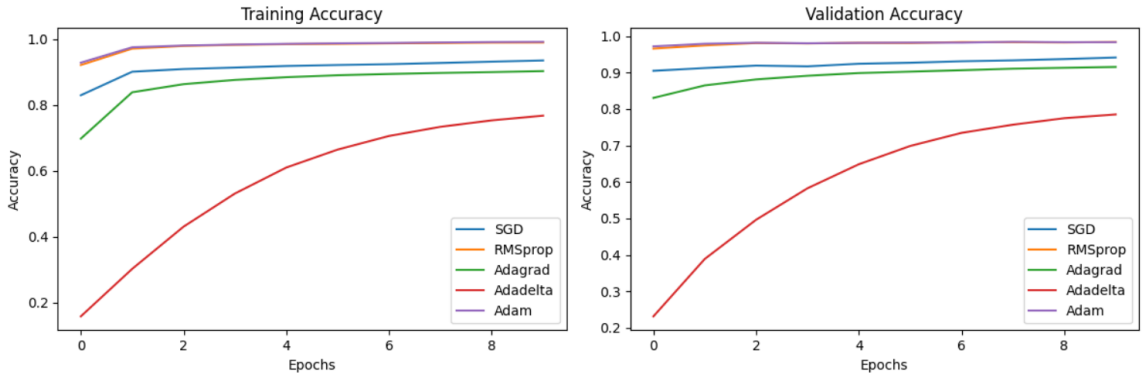


Hình 1. 3 Biểu đồ đánh giá tỉ lệ mất mát

Từ đồ thị, ta có thể thấy, Adam và RMSprop là hai thuật toán có biên độ giao động thấp nhất, gần như không thay đổi quá nhiều trong quá trình huấn luyện. Trong khi đó, AdaDelta và AdaGrad là hai thuật toán có sự biến động lớn nhất trong suốt các chu kì học. Bên cạnh đó, SGD là thuật toán có kết quả hội tụ nhanh với tỉ lệ mất mát thấp.

Các thuật toán Adam, RMSprop và Adagrad cũng có kết quả tốt nhưng Adelta có tỉ lệ mất mát cao nhất

Để có cái nhìn tổng quan hơn, ta quan sát biểu đồ tỉ lệ nhận dạng đúng của các mô hình với các thuật toán trên:



Hình 1. 4 Biểu đồ đánh giá tỉ lệ nhận dạng.

Ta có thể thấy, với thuật toán cho tỉ lệ mất mát cao như Adadelta hay Adagrad, tỉ lệ nhận dạng đúng khá thấp. Trong khi đó, các thuật toán cho tỉ lệ mất mát thấp như Adam, RMSProp, SGD cho tỉ lệ nhận dạng đúng khả quan hơn khi sử dụng trên cùng một mô hình kiến trúc mạng đề ra.

1.3 Bảng so sánh:

Qua dữ liệu phân tích và ví dụ ở trên, ta có bảng so sánh tóm tắt như sau:

	Cơ chế	Ưu điểm	Nhược điểm
SGD	Cập nhật trọng số bằng tỷ lệ học nhân với đạo hàm của hàm mất mát.	Dễ triển khai và hiểu. Yêu cầu ít bộ nhớ.	Dễ rơi vào các cục bộ tối ưu hóa. Có thể chậm nếu tốc độ học không được chọn đúng.

RMSProp	Sử dụng trung bình động của bình phương đạo hàm để điều chỉnh tỷ lệ học.	Hiệu quả hơn trong việc ổn định tốc độ học. Giảm nguy cơ rơi vào các cục bộ tối ưu hóa.	Cần phải chọn các tham số đúng để hoạt động tốt.
Adagrad	Điều chỉnh tỷ lệ học cho mỗi tham số dựa trên lịch sử đạo hàm.	Hiệu quả cho các mô hình có các tham số thưa thớt.	Có thể dẫn đến vấn đề của tỷ lệ học giảm quá nhanh, dẫn đến dừng lại sớm.
Adadelta	Sử dụng trung bình động của bình phương đạo hàm và trung bình động của thay đổi trọng số. Không yêu cầu tỷ lệ học ban đầu.	Khắc phục vấn đề của tỷ lệ học giảm quá nhanh trong Adagrad.	Có thể chậm trong một số trường hợp.
Adam	Kết hợp sự điều chỉnh tỷ lệ học của RMSProp và động lượng của Momentum.	Hiệu suất tốt trên nhiều loại mô hình và dữ liệu.	Cần cấu hình thêm một số tham số.

Bảng 1. 1 Bảng so sánh các phương pháp Optimizer

PHẦN 2 – CONTINUAL LEARNING VÀ TEST PRODUCTION

2.1 Continual Learning:

- Tổng quan:

Continual Learning (CL), còn được gọi là Lifelong Learning hoặc Incremental Learning, là một phương pháp trong lĩnh vực học máy mà mô hình được thiết kế để liên tục học từ dữ liệu mới mà nó gặp phải mà không quên đi kiến thức đã học trước đó.

Khái niệm học suốt đời được Thrun và Mitchell đề xuất vào khoảng năm 1996 và định nghĩa đầu tiên của continual learning được nêu như sau: Cho một hệ thống đã thực

hiện N bài toán. Khi đối mặt với bài toán thứ N + 1, nó sử dụng tri thức thu được từ N bài toán để trợ giúp bài toán N + 1. Sau đó, vào năm 2018, Z.Chen và B.Liu đã mở rộng định nghĩa này bằng cách cung cấp cho nó thêm các chi tiết và các tính năng bổ sung gồm: một hệ cơ sở tri thức tường minh (Knowlegde Base) được thêm vào để lưu lại tri thức đã học được từ các nhiệm vụ trước; (ii) khả năng khám phá các nhiệm vụ học mới; (iii) khả năng học trong khi làm (hoặc học trong công việc).

- Định nghĩa:

Học máy suốt đời là một quá trình học liên tục. Tại thời điểm bất kì, bộ học đã thực hiện một chuỗi N bài toán học, T_1, T_1, …, T_n. Các bài toán này, còn được gọi là các bài toán trước (previous tasks) có các tập dữ liệu tương ứng là D_1, D_2, …, D_2. Các bài toán có thể cùng kiểu hoặc thuộc các kiểu khác nhau và từ cùng một miền ứng dụng hoặc các miền ứng dụng khác nhau. Khi đối mặt với bài toán thứ N+1, T_{N+1} với dữ liệu D_{N+1}, bộ học có thể tận dụng tri thức quá khứ trong cơ sở tri thức để hỗ trợ học bài toán T_{N+1}.

Mục tiêu của LML thường là tối ưu hóa hiệu năng của bài toán mới T_{N+1}, song nó có thể tối ưu hóa bất kỳ bài toán nào bằng cách xử lý các bài toán còn lại như các bài toán trước đó. Cơ sở tri thức (KB) duy trì tri thức đã được học và được tích lũy từ việc học các bài toán trước đó. Sau khi hoàn thành bài toán học T_{N+1}, tri thức được cập nhật vào KB (chẳng hạn, kết quả trung gian cũng như các kết quả cuối cùng) thu được từ bài toán học T_{N+1}. Việc cập nhật tri thức có thể bao gồm liên quan đến kiểm tra tính nhất quán, lập luận và biến đổi của tri thức mức cao bổ sung vào KB.

- Đặc điểm:

LML có 3 đặc điểm chính:

  - Quá trình học liên tục
  - Tích luỹ và lưu giữ tri thức trong cơ sở tri thức (KB)
  - Khả năng sử dụng các tri thức đã học trước đó để xử lý các bài toán mới.

Kiến trúc hệ thống học máy suốt đời gồm 4 phần: Bộ quản lý bài toán (Task management), Cơ sở tri thức (Knowledge Base - KB), Bộ học dựa trên tri thức (Knowledge Base Learner - KBL) và Đầu ra (Output).

- Bộ quản lý bài toán (Task management): Nhận và quản lý các bài toán xuất hiện trong hệ thống. Xử lý sự chuyển bài toán và trình bày bài toán học mới cho bộ học (KBL) theo phương pháp học suốt đời.
- Cơ sở tri thức (Knowledge Base - KB): Lưu giữ lại các tri thức đã học được, gồm các thành phần:

  Kho thông tin quá khứ (Past Information Store - PIS): Lưu trữ thông tin đã học trong quá khứ, bao gồm các mô hình kết quả, mẫu hoặc các dạng kết quả, PIS cũng có thể bao gồm các kho con chứa các thông tin như:

  - Dữ liệu ban đầu được sử dụng trong mỗi bài toán trước đó,
  - Các kết quả trung gian từ mỗi bài toán trước
  - Mô hình kết quả hoặc các mẫu học được từ mỗi bài toán trước đó.

Những thông tin hoặc tri thức nào nên được giữ lại phụ thuộc vào bài toán học và thuật toán học. Trong một hệ thống cụ thể, người sử dụng cần quyết định những gì cần giữ lại để trợ giúp việc học trong tương lai.

Kho tri thức (Knowledge Store - KS): Lưu trữ kiến thức được khai thác hoặc củng cố, tổng hợp từ PIS.

- Bộ khai phá tri thức (Knowledge Miner - KM): Khai thác dữ liệu từ PIS, Kết quả được lưu ở KS.
- Bộ suy luận tri thức (Knowledge Resoner - KR): Suy luận dựa trên tri thức trong KB và PIS để tạo thêm tri thức bổ sung.
- Bộ học dựa trên tri thức (Knowledge Base Learner - KBL): Nhận kiến thức từ KS, Bộ học của LML có thể tận dụng kiến thức và thông tin trong PIS để học bài toán mới.

- Đầu ra (Output): Đây là kết quả học của người dùng, có thể là một mô hình dự báo hoặc bộ phân lớp trong học giám sát, các cụm hoặc chủ đề trong học không giám sát, chính sách trong học tăng cường, …

- Thách thức:

Continual Learning mang lại nhiều lợi ích trong học máy, nhưng cũng đối mặt với nhiều thách thức khó khăn. Dưới đây là một số thách thức chính mà Continual Learning phải đối mặt:

Cộng hưởng quên (Catastrophic Forgetting): Đây là hiện tượng mô hình quên kiến thức đã học khi được đào tạo trên dữ liệu mới. Khi mô hình cập nhật trọng số để học từ dữ liệu mới, nó có thể ghi đè lên thông tin quan trọng đã học trước đó. Điều này đặt ra thách thức lớn khi mô hình phải xử lý nhiều loại dữ liệu hoặc nhiều nhiệm vụ.

Khả năng mở rộng (Scalability): Continual Learning càng khó khăn khi mô hình cần học từ số lượng lớn các nhiệm vụ hoặc dữ liệu lớn. Quá trình này đòi hỏi sự hiệu quả và khả năng mở rộng của các thuật toán học máy.

Hiệu suất suy giảm (Performance Decay): Khi mô hình đã được đào tạo trên nhiều dữ liệu và nhiệm vụ, có thể xảy ra hiện tượng suy giảm hiệu suất toàn cầu của mô hình. Mô hình có thể trở nên quá phức tạp và khó kiểm soát khi học liên tục.

Quản lý Dữ liệu Đa Nhiệm (Multi-Task Learning): Khi mô hình phải xử lý nhiều nhiệm vụ đồng thời, cần có cách hiệu quả để quản lý và kết hợp thông tin từ các nhiệm vụ khác nhau mà không gây ảnh hưởng tiêu cực đến hiệu suất.

- Phương pháp đánh giá:

Đánh giá thử nghiệm một thuật toán Machine Learning (LML - Learning Machine Learning) là một phần quan trọng trong quá trình nghiên cứu. Dưới đây là một số phương pháp phổ biến để đánh giá hiệu suất của một thuật toán học máy:

- Cross-Validation (Chia dữ liệu kiểm tra chéo):
    - Mục đích: Đảm bảo tính ổn định và tổng quát của mô hình trên dữ liệu.

    - Thực hiện: Dữ liệu được chia thành các tập huấn luyện và kiểm tra, và mô hình được huấn luyện và đánh giá trên các tập này lặp đi lặp lại. Phổ biến nhất là k-fold cross-validation (ví dụ: 5-fold hoặc 10-fold).
- Holdout Validation (Kiểm tra giữ lại):
    - Mục đích: Đánh giá hiệu suất của mô hình trên một tập kiểm tra độc lập.
    - Thực hiện: Một phần của dữ liệu được giữ lại để kiểm tra mô hình, trong khi phần còn lại được sử dụng để huấn luyện.
- Đánh giá trên Metric phù hợp:
    - Mục đích: Sử dụng các metric phù hợp để đánh giá hiệu suất của mô hình.
    - Thực hiện: Tùy thuộc vào loại vấn đề, sử dụng các metric như accuracy, precision, recall, F1-score (đối với bài toán phân loại), hoặc mean squared error (đối với bài toán dự đoán).
- Learning Curves (Đồ thị học tập):
    - Mục đích: Đánh giá hiệu suất của mô hình trên tập huấn luyện và kiểm tra khi kích thước tập huấn luyện thay đổi.
    - Thực hiện: Vẽ đồ thị của các metric đánh giá trên tập huấn luyện và kiểm tra theo thời gian hoặc số lượng vòng lặp.
- Đánh giá trên Dữ liệu Thực Tế (Real-world Data Evaluation):
    - Mục đích: Đảm bảo rằng mô hình hoạt động tốt trên dữ liệu thực tế.
    - Thực hiện: Sử dụng dữ liệu mới, không xuất hiện trong quá trình huấn luyện, để đánh giá hiệu suất của mô hình.

- Các hướng nghiên cứu:

Học có giám sát suốt đời: Một số kỹ thuật LL đã được đề xuất dựa trên mạng nơron, Naïve Bayesian, mô hình trường ngẫu nhiên có điều kiện (CRF).

Học liên tục sử dụng các mạng nơ ron sâu: Trong vài năm qua, do sự phổ biến của học sâu, nhiều nhà nghiên cứu đã nghiên cứu vấn đề liên tục học một chuỗi các nhiệm vụ sử dụng kỹ thuật học sâu. Trong cộng đồng nghiên cứu học sâu thì LL cũng

được gọi là học liên tục. Mục tiêu của nó xây dựng mạng nơ ron sâu có khả năng học thêm từng nhiệm vụ mới mà không quên đi các mô hình đã học cho các nhiệm vụ trước.

Học thế giới mở: Học có giám sát truyền thống đòi hỏi giả định thế giới đóng phải đúng: các lớp của các dữ liệu mới phải được nhìn thấy trong quá trình học/huấn luyện. Điều này không phù hợp để học trong các môi trường mở và biến động với những lớp mới luôn xuất hiện.

Học không giám sát suốt đời: Các nghiên cứu trong hướng này chủ yếu là về mô hình hoá chủ đề suốt đời và trích chọn thông tin suốt đời. Các kỹ thuật này đều dựa trên khai thác cấp độ meta, tức là khai thác kiến thức được chia sẻ qua các tác vụ.

2.2 Test Production:

- Định nghĩa:

Test Production là quá trình tạo ra các bài kiểm tra để đánh giá hiệu suất của mô hình học máy. Các bài kiểm tra này được thiết kế để đánh giá khả năng tổng quát hóa của mô hình, nghĩa là khả năng áp dụng kiến thức đã học từ tập dữ liệu huấn luyện vào dữ liệu mới.

- Các hình thức test:

- Test Code:
    - Unit tests: Kiểm thử các thành phần hoặc chức năng cá nhân của một hệ thống học máy một cách cô lập.
    - Integration tests: Xác minh các tương tác và giao diện giữa các thành phần khác nhau của hệ thống học máy.
    - System tests: các thử nghiệm về thiết kế của một hệ thống với các kết quả đầu ra dự kiến dựa trên các đầu vào.
    - Acceptance tests: các thử nghiệm để xác minh rằng các yêu cầu đã được đáp ứng, thường được gọi là Thử nghiệm chấp nhận của người dùng (UAT).

- Regression tests: Đảm bảo rằng các thay đổi vào hệ thống (ví dụ: cập nhật mô hình, sửa đổi mã nguồn) không gây ra vấn đề mới.

- Test Data:
  - Mục Đích: Xác minh chất lượng, tính nhất quán và tính phù hợp của dữ liệu đầu vào.
  - Tập Trung:
    - Kiểm Thử Chất Lượng Dữ Liệu: Đảm bảo dữ liệu đầu vào đáp ứng các tiêu chuẩn yêu cầu.
    - Kiểm Thử Phân Phối Dữ Liệu: Kiểm tra xem phân phối dữ liệu trong tập kiểm thử có tương đồng với tập huấn luyện hay không.
  - Kiểm Thử:
    - Xác định và xử lý dữ liệu thiếu hoặc không nhất quán.
    - Xác minh rằng luồng dữ liệu là mạnh mẽ và hiệu quả.
    - Sử dụng các tập dữ liệu đại diện cho quá trình kiểm thử.
- Test Model:
  - Mục Đích: Đánh giá hiệu suất, độ chính xác và khả năng tổng quát của mô hình học máy.
  - Tập Trung:
    - Kiểm Thử Hiệu Năng Mô Hình: Đánh giá các chỉ số hiệu suất của mô hình trên tập dữ liệu kiểm thử.
    - Kiểm Thử Độ Bền: Đánh giá khả năng của mô hình xử lý đầu vào không mong muốn hoặc tác động đối kháng.
  - Kiểm Thử:
    - Đánh giá độ chính xác, precision, recall và các chỉ số quan trọng khác.
    - Tiến hành kiểm thử đối kháng và biến đổi đầu vào để đánh giá độ bền.

- Kiểm tra sự công bằng và độ chệch của mô hình.

- Các bước quan trọng trong quá trình kiểm thử:

Kiểm thử tích cực (Positive Testing): Đảm bảo rằng mô hình đang làm đúng công việc của nó, có thể dự đoán và phân loại dữ liệu đầu vào đúng cách.

Kiểm thử tiêu cực (Negative Testing): Xác định khả năng của mô hình khi đối mặt với dữ liệu ngoại lệ, không thuộc phạm vi dữ liệu huấn luyện.

Kiểm thử hiệu suất (Performance Testing): Đánh giá hiệu suất của mô hình trên các tiêu chí như độ chính xác, tốc độ dự đoán và các yếu tố khác quan trọng tùy thuộc vào ứng dụng cụ thể.

- Ý nghĩa và ứng dụng:

Test Production là quan trọng để đảm bảo rằng mô hình không chỉ học thuộc lòng dữ liệu huấn luyện mà còn có khả năng tổng quát hóa đối với dữ liệu mới. Các bài kiểm tra phải được thiết kế một cách có thể đại diện cho các điều kiện thực tế mà mô hình có thể gặp phải.

DANH MỤC TÀI LIỆU THAM KHẢO

1. https://www.phamduytung.com/blog/2021-01-15---adabelief-optimizer/
2. https://madewithml.com/courses/mlops/testing/
3. http://lib.uet.vnu.edu.vn/bitstream/123456789/1082/1/TrangPTQ_LuanVan.pdf
4. https://machinelearningcoban.com/2017/01/16/gradientdescent2/