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

Đống Thạc Nhân - 52100914

BÁO CÁO CUỐI KÌ
MÔN NHẬP MÔN HỌC MÁY

Người hướng dẫn
TS Lê Anh Cường

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, em cảm ơn thầy Lê Anh Cường đã nhiệt tình giảng dạy ,giúp chúng em hiểu rõ hơn về học máy, cũng như hỗ trợ chúng em suốt quá trình học tập môn học này.

Tiếp theo, chúng em xin gửi lời cảm ơn tới khoa Công Nghệ Thông Tin trường Đại học Tôn Đức Thắng. Khoa đã tạo mọi điều kiện cho chúng em được học tập và nghiên cứu môn học này.

Cuối cùng, do giới hạn về mặt kiến thức, chúng em biết bài báo cáo của mình còn nhiều thiếu sót và hạn chế, kính mong được sự hướng dẫn và đóng góp của quý thầy cô để bài báo cáo của chúng em được hoàn thiện hơn. Chúc quý thầy cô tràn đầy sức khỏe.

TP. Hồ Chí Minh, ngày 23 tháng 12 năm 2023

Tác giả

Đống Thạc Nhân

(Ký tên và ghi rõ họ tên)

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 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 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 23 tháng 12 năm 2023

Tác giả

Đống Thạc Nhân

(Ký tên và ghi rõ họ tên)

Mục lục

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

1.1 Optimizer là gì và tại sao cần sử dụng optimizer

Thuật toán tối ưu (Optimizer) là một thành phần không thể thiếu trong các mô hình machine learning, đặc biệt là deep learning. Nó là cơ sở để xây dựng lên một mô hình neural network với mục đích học được các pattern của dữ liệu đầu vào trong quá trình huấn luyện. Từ đó điều chỉnh cặp tham số weights và bias sao cho phù hợp để giảm thiểu mất mát cũng như tăng chất lượng mô hình. Mục tiêu chính của thuật toán tối ưu là tìm ra cặp tham số giúp dữ liệu dự đoán càng giống với dữ liệu thực tế càng tốt.

1.2 Các loại optimizer

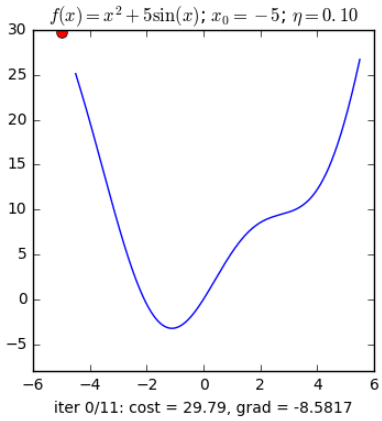
1.2.1 *Gradient Descent*

Đây là thuật toán tối ưu cơ bản, khá dễ hiểu so với các optimizer khác. Khi huấn luyện mô hình, chúng ta luôn muốn giá trị hàm loss nhỏ nhất. Để như vậy thì đạo hàm hàm số đó phải bằng 0, nhưng với các hàm phức tạp thì việc đạo hàm là rất khó, gần như không thể. Nên thay vào đó chúng ta có thể sử dụng gradient descent để tìm điểm gần với điểm cực tiểu nhất. Công thức của gradient descent:

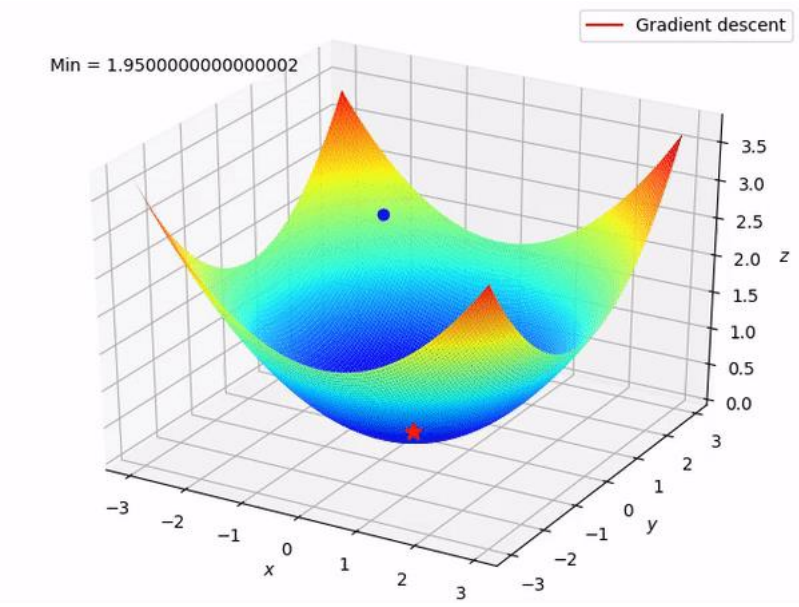
X_new = X_old – learning_rate*f’(x)

Hướng tiếp cận là ta xuất phát từ một điểm được coi là gần với nghiệm, sau đó dùng phép toán lặp để tiến dần về điểm cần tìm, tức là đạo hàm gần với 0. Ta sử dụng dấu trừ trong công thức vì để tiến tới điểm cực tiểu ta phải đi ngược đạo hàm.

Thuật toán sẽ bị ảnh hưởng bởi nhiều yếu tố như điểm x ban đầu hay tốc độ học (learning rate). Nếu bước nhảy quá nhỏ thì thuật toán vẫn hội tụ nhưng tốc độ rất chậm. Ngược lại nếu bước nhảy lớn thì sẽ nhanh hơn nhưng không hội tụ vì có thể vượt qua đích. Hình minh họa:



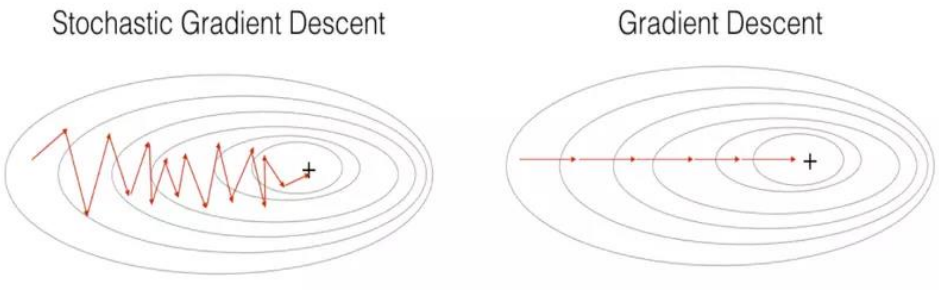
Hình 1: Gradient descent 1 biến



Hình 2: Gradient descent nhiều biến

1.2.2 *Stochastic Gradient Descent (SGD)*

SGD là một biến thể của Gradient descent. Vấn đề của GD là ta cần sử dụng toàn bộ dữ liệu để tính toán, với dữ liệu lớn thì GD tỏ ra mất thời gian. Để giải quyết việc đó thì SGD ra đời. Với SGD chúng ta chỉ cần lấy một số mẫu từ tập dữ liệu, ví dụ epoch đó có N điểm dữ liệu thì chúng ta sẽ cập nhật trọng số N lần và xáo trộn dữ liệu sau mỗi lần lặp thay vì mỗi epoch cập nhật trọng số 1 lần như GD.

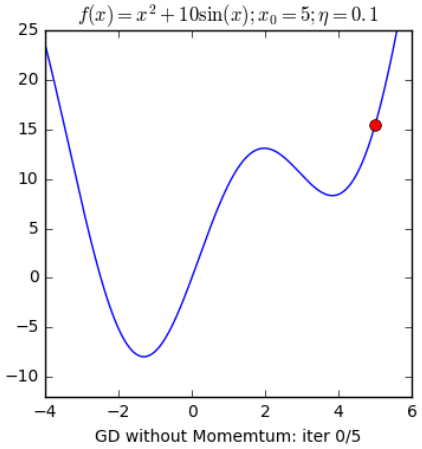


Hình 3: SGD và GD

Có thể thấy đường đi của SGD khá nhiễu, do đó SGD sử dụng 1 lượng lớn lần lặp để tìm ra điểm tối thiểu cục bộ. Nhưng nhìn tổng quan tuy thời gian tính toán một epoch lâu hơn GD nhưng chi phí tính toán cuối cùng vẫn thấp hơn nhiều so với GD. Ngoài ra, với trường hợp dữ liệu cập nhật liên tục, SGD sẽ rất hiệu quả vì chỉ cần cập nhật trên điểm dữ liệu đó thôi.

1.2.3 *Gradient descent with Momentum*

Đây là phương pháp cải tiến của Gradient descent. Với những hàm số phức tạp thì đồ thị sẽ xuất hiện những điểm local minimum khiến kết quả có thể bị sai lệch. Hình minh họa:



Hình 4: GD without momentum

Có thể thấy, kết quả đã hội tụ chỉ sau 5 lần lặp, nhưng đó là điểm local minimum. Để vượt qua những trường hợp như vậy ta cần tạo "vận tốc ban đầu" đủ lớn để hay còn gọi là tạo đà. Công thức của GD with momentum:

$v = \beta . v + (1 - \beta) . \nabla J(w)$

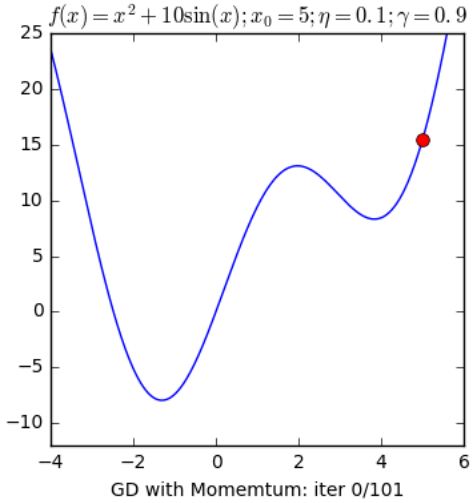
$w = w - \eta . v$

Với:

$\nabla J(w)$ là đạo hàm hàm loss

η là learning rate

β là tham số cho momentum (thường là số gần 1 như 0.9)



Hình 5: GD with momentum

Khi có momentum, tuy tốn nhiều vòng lặp hơn nhưng nghiệm đã vượt tốc qua đoạn dốc để tiến tới global minimum. Tóm lại GD with momentum phù hợp cho những bài toán phức tạp có đường cong cao và rãnh nhỏ, giúp giảm thiểu sự dao động và tăng tốc sự hội tụ.

1.2.4 *Adaptive Gradient Descent(Adagrad)*

Thuật toán này được gọi là gradient descent thích ứng vì nó sử dụng các tốc độ học khác nhau cho mỗi lần lặp. Tốc độ sẽ thay đổi phụ thuộc vào sự khác biệt giữa các tham số trong quá trình huấn luyện. Khi càng nhiều tham số thay đổi, tốc độ học càng nhỏ. Việc này rất hữu ích vì trên thực tế, dữ liệu sẽ chứa các đặc trưng thưa hoặc dày đặc. Việc sử dụng tốc độ học cố định sẽ kém hiệu quả hơn. Công thức của Adagrad:

$$\mathrm{w_t} = \mathrm{w_{t-1}} - \eta'_{\mathrm{t}} \frac{\partial \mathrm{L}}{\partial \mathrm{w(t-1)}}$$

$$\eta'_{\mathrm{t}} = \frac{\eta}{\mathrm{sqrt}(\alpha_{\mathrm{t}} + \epsilon)}$$

Trong đó:

$\alpha(t)$: biểu thị các tốc độ học khác nhau ở mỗi lần lặp

η : là một hằng số

ϵ: là giá trị dương nhỏ để tránh chia cho 0

Tóm lại, adagrad sẽ giúp loại bỏ việc điều chỉnh tốc độ học thủ công. Nó đáng tin cậy hơn so với các thuật toán gradient descent cũng như các biến thể của nó, đồng thời đạt được sự hội tụ ở tốc độ cao hơn.

1.2.5 *Root Mean Square Propagation (RMSprop)*

Đây là thuật toán tối ưu gradient descent giúp giải quyết việc giảm tốc độ học đơn điệu của adagrad. Nguyên tắc cơ bản của RMSProp là giảm tốc độ học của các tham số mô hình dựa trên độ lớn của gradient cho từng tham số. Nếu một tham số có gradient lớn, thì tốc độ học của nó sẽ giảm, và ngược lại. Điều này giúp kiểm soát tốc độ học của mỗi tham số tùy thuộc vào cách gradient của nó biến đổi theo thời gian. Công thức RMSprop:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\nabla J(w_t))^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla J(w_t)$$

$E[g^2]_t$ là trung bình chuyển động của bình phương gradient

β là hệ số quên(khoảng 0.9)

ϵ là hệ số dương nhỏ tránh chia cho 0

RMSProp giúp cải thiện khả năng hội tụ của thuật toán gradient descent, đặc biệt là trong các tình huống mà tốc độ học cần được điều chỉnh linh hoạt để tránh việc overshooting (tốc độ học quá cao) hay còn gọi là vanishing gradient (tốc độ học quá nhỏ).

1.2.6 *Adaptive Moment Estimation(Adam)*

Đây là một trong những thuật toán tối ưu hiệu quả và được sử dụng nhiều nhất trong các thuật toán. Nó là sự mở rộng của SGD và cũng được thiết kế để cập nhật trọng số của mạng nơ-ron trong quá trình huấn luyện. Khác với SGD, giữ một tốc độ học duy nhất trong suốt quá trình huấn luyện, Adam optimizer tính toán các tốc độ học cá nhân dựa trên gradient quá khứ và moment thứ hai của chúng. Người sáng tạo của Adam optimizer cũng tích hợp các tính năng có lợi từ các thuật toán tối ưu hóa khác như AdaGrad và RMSProp.

Nguyên tắc cơ bản của thuật toán Adam:

1. Moment đầu (First Moment): Tương tự như thuật toán Momentum, Adam tính trung bình chuyển động của gradient, được gọi là moment đầu.

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

Trong đó:

g_t là gradient của hàm chi phí tại thời điểm t.

β_1 là hệ số đánh giá moment đầu (thường là 0.9).

2. Moment thứ hai (Second Moment): Adam tính trung bình chuyển động của bình phương gradient, được gọi là moment thứ hai

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

Trong đó:

$(g_t)^2$ là bình phương của gradient tại thời điểm t.

β_2 là hệ số đánh giá moment thứ hai (thường là 0.999).

3. Hiệu chỉnh moment đầu và moment thứ hai: Để tránh sự chệch lệch đầu tiên khi m_t và v_t ban đầu gần 0, Adam thực hiện hiệu chỉnh moment đầu và moment thứ hai như sau:

$\widehat{m_t} = \frac{m_t}{1 - \beta_2^t}$

$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$

4. Cập nhật trọng số: Adam sử dụng moment đầu và moment thứ hai để cập nhật trọng số w_t như sau:

$$w_{t+1} = w_t - \eta . \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}$$

Tóm lại thuật toán Adam kết hợp giữa ước lượng moment đầu và moment thứ hai, giúp nó thích ứng với các tốc độ học khác nhau cho từng tham số và tăng tốc quá trình hội tụ.

1.3 So sánh

	Ưu điểm	Nhược điểm
Gradient descent	_Cơ bản, dễ hiểu, dễ triển khai _Hoạt động tốt trên dữ liệu nhỏ, ít nhiễu	_Tốc độ chậm nếu dữ liệu lớn _Dễ bị kẹt ở local minimum
SGD	_Phù hợp với dữ liệu lớn. _Giảm tính toán so với GD vì chỉ sử dụng một mẫu dữ liệu mỗi lần cập nhật.	_Dao động lớn do sử dụng một mẫu ngẫu nhiên, có thể làm chậm quá trình hội tụ
GD with momentum	_Giảm dao động và giúp tránh các điểm địa phương cực tiểu.	_ Cần điều chỉnh thêm hệ số quán tính
Adagrad	_Tốc độ học tự động thích ứng với từng tham số. _Hiệu quả trên các dữ liệu có đặc trưng thưa.	_Tốc độ học giảm quá nhanh và có thể trở thành vấn đề trên dữ liệu lớn.
RMSProp	_Kiểm soát tốc độ học tốt. _Hiệu quả trên nhiều loại hàm chi phí.	_ Có thể bị kẹt ở local minimum
Adam	_Kết hợp ưu điểm của Momentum và RMSProp.	

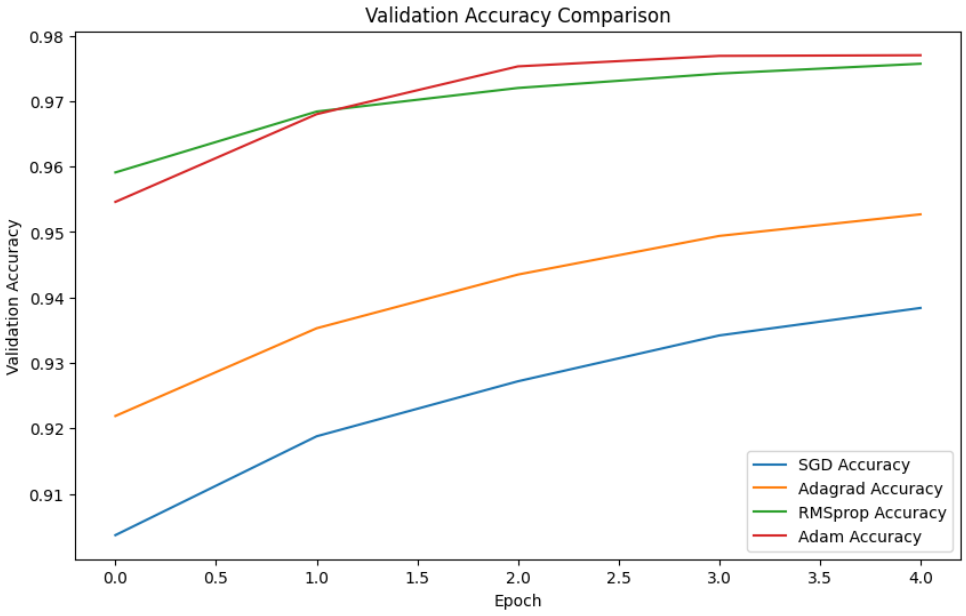
	_ Hiệu quả trên nhiều loại dữ liệu và hàm chi phí. _ Tốc độ hội tụ nhanh và tự động thích ứng với tốc độ học.	

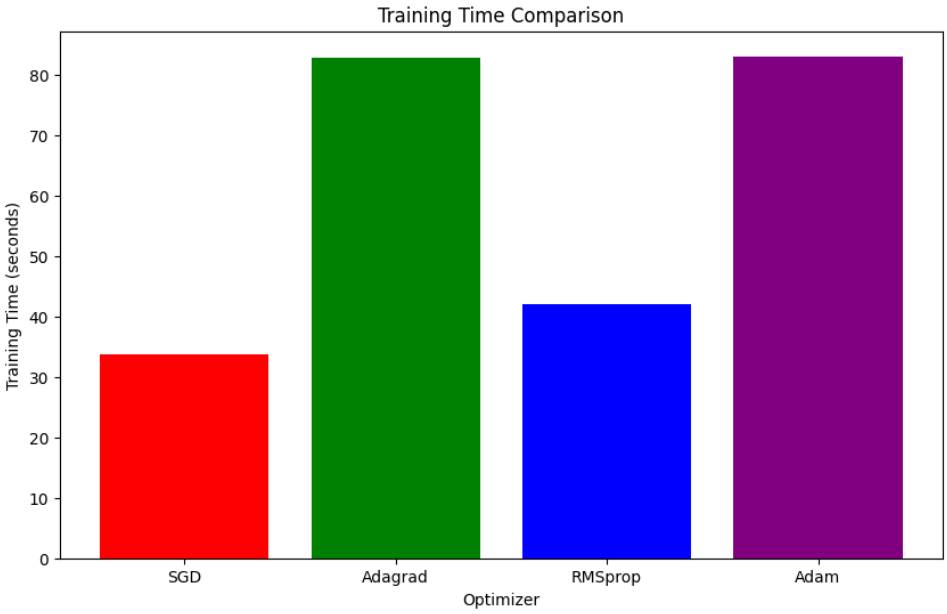
Xây dựng mô hình và so sánh

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Flatten the images va normalize pixel
x_train = x_train.reshape((60000, 28 * 28))
x_test = x_test.reshape((10000, 28 * 28))
#train
def train_model(optimizer, x_train, y_train, x_test, y_test):
  model = Sequential([
    Dense(128, activation='relu', input_shape=(28 * 28,)),
    Dense(10, activation='softmax')
  ])
  model.compile(optimizer=optimizer,loss=SparseCategoricalCrossentropy(),metrics=['accuracy'])
  start_time = time.time()
  history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
  end_time = time.time()
  training_time = end_time - start_time
  return history, training_time
sgd_history, sgd_time = train_model(SGD(learning_rate=0.01), x_train, y_train, x_test, y_test)
adagrad_history, adagrad_time = train_model(Adagrad(learning_rate=0.01), x_train, y_train, x_test, y_test)
rmsprop_history, rmsprop_time = train_model(RMSprop(learning_rate=0.001), x_train, y_train, x_test, y_test)
adam_history, adam_time = train_model(Adam(learning_rate=0.001), x_train, y_train, x_test, y_test)
```

Kết quả





Trong ví dụ trên có thể thấy, adam có có độ chính xác cao nhất nhưng thời gian train cũng lâu nhất.Sát sau đó là RMSprop nhưng RMSprop có tốc độ train vượt trội hơn. Tiếp theo là đến Adagrad nhưng adgrad lại có thời gian train gần bằng adam. Cuối cùng kém hiệu quả nhất là SGD tuy có thời gian train rất nhanh. Vậy với dữ liệu test mnist thì RMSprop hiệu quả nhất

CHƯƠNG 2. Tìm hiểu về Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán nào đó.

2.1 Continual learning

Cotinual learning hay học liên tục là phương pháp mà mô hình học máy tiếp tục phát triển và cải thiện theo thời gian khi nó tiếp xúc với dữ liệu mới. Việc này rất quan trọng vì:

1. Dữ liệu tiến hóa: Dữ liệu mà các mô hình máy học gặp có thể thay đổi theo thời gian. Không có cách nào tránh khỏi điều này. Để duy trì độ chính xác của dữ liệu và hiệu suất của mô hình, chúng cần học liên tục từ dữ liệu mới.

2. Cập nhật theo thời gian thực: Với các mô hình học hoạt động theo thời gian thực, việc học liên tục cho phép chúng thích ứng ngay lập tức với đầu vào mới để cung cấp thông tin phù hợp và chất lượng cao.

3.Hiệu suất: Học liên tục có thể làm cho quá trình cập nhật mô hình trở nên hiệu quả hơn. Thay vì huấn luyện lại một mô hình từ đầu với mỗi bộ dữ liệu mới, nó có thể được cập nhật từng phần nhỏ.

Sự khác biệt giữa học máy truyền thống và học liên tục

Với học máy truyền thống:

1.Mô hình được huấn luyện trên một bộ dữ liệu tĩnh.

2. Mô hình mới được triển khai, nhưng nó không học từ dữ liệu mới trừ khi được huấn luyện lại.

3. Phương pháp này không lý tưởng khi xử lý dữ liệu thay đổi qua thời gian hoặc trong các tình huống cần thiết phải thích ứng.

Với máy học liên tục:

1. Mô hình liên tục học và thích ứng với dữ liệu mới.

2. Khi dữ liệu mới xuất hiện, mô hình cập nhật và làm rõ hiểu biết của nó để cung cấp dự báo hoặc quyết định chính xác hơn.

3.Mô hình học máy duy trì hiệu suất cao qua thời gian.

Cơ chế của học máy liên tục

1. Huấn luyện mô hình ban đầu: Trước tiên tạo một mô hình học bằng cách sử dụng một bộ dữ liệu huấn luyện cơ bản.

2. Dữ liệu mới: Khi thông tin mới xuất hiện, mô hình máy học được cập nhật tương ứng. Điều này có thể xảy ra theo lịch trình đều đặn (ví dụ, hàng ngày hoặc hàng tuần) hoặc theo thời gian thực tùy mục đính thực tế.

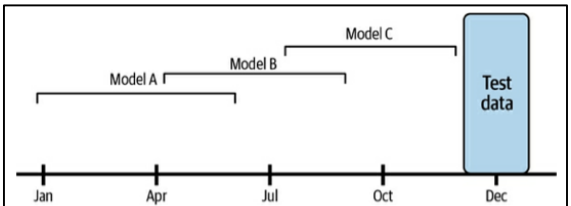


Figure 9-5. To get a sense of the performance gain you can get from fresher data, train your model on data from different time windows in the past and test on data from today to see how the performance changes

3. Cập nhật mô hình: Mô hình sử dụng dữ liệu mới để cập nhật các tham số của nó. Điều này có thể liên quan đến việc huấn luyện lại toàn bộ mô hình hoặc, phổ biến hơn, là thêm một cập nhật trong đó nó điều chỉnh các tham số dựa trên dữ liệu mới.

4. Đánh giá: Hiệu suất của mô hình huấn luyện được cập nhật được đánh giá. Nếu nó đã cải thiện, mô hình sẽ thay thế mô hình cũ. Nếu không, mô hình cũ được giữ nguyên.

5. Lặp lại: Quá trình này của dữ liệu mới xuất hiện, cập nhật mô hình và đánh giá được lặp đi lặp lại liên tục, cho phép mô hình học và thích ứng theo thời gian để tạo ra kết quả tốt hơn.

Stateless retraining và stateful retraining

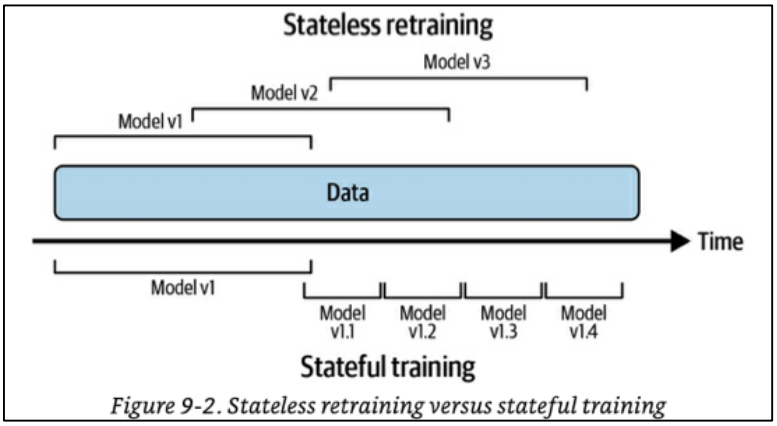


Figure 9-2. Stateless retraining versus stateful training

Huấn luyện không lưu trạng thái (stateless)

Huấn luyện lại mô hình của bạn từ đầu mỗi lần, sử dụng trọng số được khởi tạo ngẫu nhiên và dữ liệu mới hơn.

- Có thể trùng dữ liệu đã được sử dụng trong phiên bản mô hình trước đó và tốn thời gian.
- Hầu hết các công ty bắt đầu thực hiện học liên tục bằng cách sử dụng huấn luyện không lưu trạng thái.

Huấn luyện có lưu trạng thái (còn được gọi là fine-tuning, học tăng cường)

Khởi tạo mô hình của bạn với trọng số từ lần huấn luyện trước đó và tiếp tục huấn luyện bằng cách sử dụng dữ liệu mới.

- Cho phép mô hình của bạn cập nhật với lượng dữ liệu ít hơn.
- Cho phép mô hình hội tụ nhanh hơn và sử dụng ít tài nguyên tính toán hơn.
- Đôi khi bạn sẽ cần chạy huấn luyện không lưu trạng thái với một lượng lớn dữ liệu để hiệu chỉnh lại mô hình.
- Huấn luyện có lưu trạng thái thường được sử dụng để tích hợp dữ liệu mới vào một kiến trúc mô hình sẵn có và cố định. Nếu bạn muốn thay đổi các đặc điểm hoặc kiến trúc của mô hình, sẽ cần thực hiện một lượt huấn luyện không lưu trạng thái đầu tiên.

2.2 Test production

Sau khi mô hình của bạn được cập nhật, nó không thể được đưa vào sử dụng mà không kiểm tra. Nó cần được thử nghiệm để đảm bảo rằng nó an toàn và tốt hơn so với mô hình hiện tại đang chạy. Tuy nhiên, chỉ kiểm tra mô hình offline trước khi triển khai là không đủ, cần kiểm tra sau khi mô hình đi vào hoạt động.

Kiểm tra ngoại tuyến trước triển khai

Hai phương pháp phổ biến là sử dụng một tập dữ liệu thử nghiệm để so sánh với một mô hình cơ sở và backtests.

Tập dữ liệu thử nghiệm thường là cố định để làm thước đo so sánh nhiều mô hình. Điều này cũng có nghĩa là hiệu suất tốt trên một tập dữ liệu thử nghiệm không đảm bảo hiệu suất tốt dưới phân phối dữ liệu sau khi triển khai.

backtests là ý tưởng sử dụng dữ liệu mới nhất thu thập được mà mô hình chưa từng thấy trong quá trình huấn luyện để kiểm thử hiệu suất .

Testing in Production

Một số phương pháp test

- **Shadow Deployment**

Triển khai một mô hình phụ chạy song song với mô hình chính. Chuyển mọi yêu cầu đến cả hai mô hình, nhưng chỉ xài kết quả của mô hình chính. Ghi lại dự đoán của cả hai mô hình để sau đó so sánh và đánh giá chúng.

- **A/B Testing**

  Triển khai mô hình phụ cùng với mô hình chính (mô hình A) và định tuyến một số phần trăm lưu lượng đến mô hình phụ (mô hình B). Dự đoán từ B được hiển thị cho người dùng. Sử dụng theo dõi và phân tích dự đoán trên cả hai mô hình để xác định hiệu suất của mô hình B có tốt hơn so với mô hình A hay không.

  Một số trường hợp không phù hợp với việc chia lưu lượng 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 phân chia thời gian: một ngày cho mô hình A, ngày tiếp theo cho mô hình B.

  Phân chia lưu lượng phải là ngẫu nhiên thực sự. Nếu cố tình lựa chọn thì kết luận của bạn sẽ không chính xác.

- **Canary Release**

  Triển khai mô hình phụ và mô hình chính song song nhau, nhưng bắt đầu với mô hình phụ không nhận lưu lượng. Dần dần chuyển lưu lượng từ mô hình chính sang mô hình phụ (gọi là canary). Theo dõi các chỉ số hiệu suất của mô hình phụ, nếu kết quả tốt, tiếp tục cho đến khi toàn bộ lưu lượng đều chuyển sang mô hình phụ.

  Các phiên bản Canary có thể kết hợp với kiểm thử A/B để đo lường sự khác biệt về hiệu suất.

  Nếu mô hình phụ bắt đầu gặp vấn đề, đổi hướng lưu lượng về mô hình chính.

- **Interleaving Experiments**

  Trong A/B testing, 1 người dùng chỉ nhận 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 hai mô hình A và B. Sau đó, chúng ta theo dõi mô hình hoạt động bằng cách xem ưa thích của người dùng với dự đoán của từng mô hình (ví dụ: người dùng click nhiều hơn vào các đề xuất từ mô hình B).

Không phải tất cả các model đều phù hợp với chiến lược này. Recommendation là một trường hợp sử dụng điển hình cho thử nghiệm xen kẽ.

Cần tránh việc tạo lợi thế không công bằng cho một mô hình như việc luôn chọn lựa dự đoán tốt nhất từ mô hình A. Khả năng chiếm vị trí đầu tiên của mô hình A hoặc mô hình B phải như nhau. Các vị trí còn lại có thể được điền bằng phương pháp team-drafting.

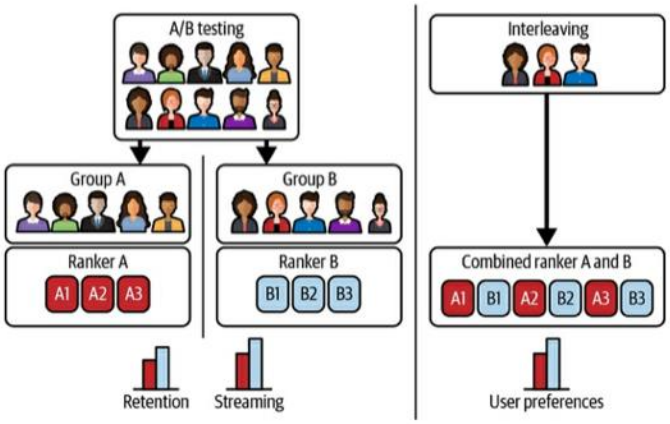


Figure 9-6. An illustration of interleaving versus A/B testing. Source: Adapted from an image by Parks et al.

Tham khảo:

[Chapter 9 - Continual learning and test in production]

[A Comprehensive Guide on Optimizers in Deep Learning]