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

ĐỒ ÁN CUỐI KỲ

MÔN NHẬP MÔN HỌC MÁY

Người hướng dẫn: **GV. LÊ ANH CƯỜNG**

Người thực hiện: **LÊ NGUYỄN NHẬT ANH - 52100597**

Lớp **: 21050301**

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

ĐỒ ÁN CUỐI KỲ

MÔN NHẬP MÔN HỌC MÁY

Người hướng dẫn: **GV. LÊ ANH CƯỜNG**

Người thực hiện: **LÊ NGUYỄN NHẬT ANH - 52100597**

Lớp **: 21050301**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Trước tiên, em xin gửi lời cảm ơn đến các thầy cô Phòng Khoa Công nghệ thông tin đã tạo điều kiện để chúng em được học hỏi, tiếp cận với một môn học mới. Đặc biệt là thầy Lê Anh Cường, cảm ơn thầy vẫn luôn nhiệt tình giúp đỡ, truyền đạt kiến thức môn học cho em qua từng buổi học. Nhờ vậy nhóm em mới có đủ cơ sở để hoàn thành bài báo cáo cuối kỳ môn nhập môn học máy.

Cảm ơn thầy đã đồng hành và bên cạnh chúng em trong quá trình học tập vừa qua, chúc thầy có nhiều sức khỏe và đạt được nhiều thành công trong cuộc sống.

TP. Hồ Chí Minh, ngày 21 tháng 12 năm 2023
Tác giả
(Ký tên và ghi rõ họ tên)
Lê Nguyễn Nhật Anh

ĐỒ ÁN / 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 thầy 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 cuối kỳ 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 cuối kỳ 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 18 tháng 12 năm 2023
Tác giả
(Ký tên và ghi rõ họ tên)

(Đã ký)

Lê Nguyễn Nhật Anh

TÓM TẮT

Bài báo cáo này trình bày những nội dung tìm hiểu về sánh các phương pháp Optimizer trong huấn luyện mô hình học máy, so sánh giữa các phương pháp; định nghĩa Continual Learning và Test Production và áp dụng nó khi xây dựng một giải pháp học máy để giải quyết một bài toán.

MỤC LỤC

CHƯƠNG 1 – PHƯƠNG PHÁP OPTIMIZER TRONG HUẤN LUYỆN MÔ HÌNH HỌC MÁY

1.1 GIỚI THIỆU VỀ OPTIMIZER

- Phương pháp optimizer (hay còn gọi là thuật toán optimizer) là một thành phần quan trọng trong quá trình tối ưu hóa các mô hình máy học và mạng nơ-ron. Mục tiêu của phương pháp optimizer là tìm kiếm các tham số của mô hình để giảm thiểu hoặc tối đa hóa một hàm mục tiêu.

- Trong quá trình huấn luyện mô hình, chúng ta thường xác định một hàm mất mát (loss function) để đo lường sự khác biệt giữa giá trị dự đoán của mô hình và giá trị thực tế. Mục tiêu của phương pháp optimizer là điều chỉnh các tham số của mô hình (như trọng số và bias) để giảm thiểu giá trị của hàm mất mát.

- Các phương pháp optimizer phổ biến trong machine learning và deep learning bao gồm:

+ Gradient Descent: Là phương pháp tối ưu thông dụng, dựa trên việc tính toán đạo hàm của hàm mất mát theo các tham số và điều chỉnh các tham số theo hướng giảm độ dốc của hàm mất mát.

+ Adam: Là một phương pháp tối ưu kết hợp giữa Momentum và RMSprop. Nó kết hợp việc lưu trữ lịch sử của gradient và hướng đi trước đó (momentum) để điều chỉnh learning rate cho từng tham số.

+ Adagrad: Là phương pháp tối ưu trong đó learning rate của mỗi tham số được điều chỉnh tự động theo lịch sử của gradient. Nó ưu tiên cập nhật những tham số có gradient lớn nhất ít hơn những tham số có gradient nhỏ hơn.

+ RMSprop: Là một phương pháp tối ưu khác trong đó learning rate được điều chỉnh tự động theo giá trị trung bình bình phương của gradient trước đó.

+ AdaDelta: AdaDelta là một phương pháp tối ưu hóa khác dựa trên gradient. Nó sử dụng tỷ lệ học tập được điều chỉnh tự động và giữ trọng số của các bước cập nhật trước đó.

1.2 GRADIENT DESCENT (GD)

- Gradient Descent (GD) là một phương pháp tối ưu hóa cơ bản trong máy học và tối ưu hóa. Nó được sử dụng để tìm giá trị tối ưu của một hàm số thông qua việc điều chỉnh các tham số của nó dựa trên đạo hàm của hàm mất mát.

- Ý tưởng cơ bản của Gradient Descent là dựa vào đạo hàm của hàm mất mát để xác định hướng và tỷ lệ cập nhật các tham số của mô hình. Mục tiêu là đi dần dần theo hướng giảm dốc của hàm mất mát để đạt được giá trị tối ưu.

- Các bước chính của Gradient Descent:

  + Bước 1: Khởi tạo các tham số ban đầu của mô hình.
  + Bước 2: Tính toán đạo hàm riêng của hàm mất mát theo từng tham số. Đạo hàm này cho biết hướng tăng hoặc giảm nhanh nhất của hàm mất mát tại điểm đó.
  + Bước 3: Cập nhật các tham số bằng cách di chuyển theo hướng đối nghịch với đạo hàm tính được và với một tỷ lệ học tập (learning rate) nhất định. Tỷ lệ học tập quyết định độ lớn của bước cập nhật.
  + Bước 4: Lặp lại bước 2 và 3 cho đến khi đạt được điều kiện dừng, ví dụ như đạt được số lần lặp tối đa hoặc đạt được sự hội tụ của các tham số.

- Có ba biến thể chính của Gradient Descent:

  1. Batch Gradient Descent: Trong biến thể này, toàn bộ tập dữ liệu huấn luyện được sử dụng để tính gradient và cập nhật tham số trong mỗi vòng lặp. Điều này đòi hỏi nhiều tài nguyên tính toán và không phù hợp cho các tập dữ liệu lớn.

  2. Stochastic Gradient Descent (SGD): Thay vì sử dụng toàn bộ tập dữ liệu, SGD chỉ chọn ngẫu nhiên một mẫu trong mỗi vòng lặp để tính gradient và cập nhật tham số. Điều này giúp giảm tài nguyên tính toán và thích hợp cho các tập dữ liệu lớn. Tuy nhiên, SGD có thể gây ra sự dao động trong quá trình tối ưu và có thể không đạt được giá trị tối thiểu chính xác.

  3. Mini-batch Gradient Descent: Đây là một biến thể kết hợp giữa Batch Gradient Descent và SGD. Thay vì sử dụng toàn bộ tập dữ liệu hoặc một mẫu duy nhất, Mini-batch Gradient Descent sử dụng một lượng nhỏ (mini-batch) mẫu để tính gradient và cập nhật tham số. Điều này giúp tiết kiệm tài nguyên tính toán và hội tụ nhanh hơn so với SGD.

- Xét một ví dụ đơn giản về việc sử dụng Gradient Descent để tìm giá trị tối thiểu của một hàm đơn giản là hàm bậc hai: f(x) = x^4 - 3x^3 + 2.

Bước 1: Khởi tạo giá trị ban đầu cho tham số x, chẳng hạn x = 2.
Bước 2: Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2.
Bước 3: Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx = 4x^3 - 9x^2.
Bước 4: Cập nhật tham số x bằng cách di chuyển ngược dọc theo đạo hàm với một learning rate (tỷ lệ học) α:

x = x - α * dJ(x)/dx = x - α * (4x^3 - 9x^2).

Tiếp tục lặp lại các bước trên cho một số vòng lặp cho đến khi đạt được điều kiện dừng hoặc đạt được giá trị tối thiểu mong muốn.

- Ví dụ, giả sử chúng ta chọn learning rate α = 0.1 và số vòng lặp là 100:

Vòng lặp 1:

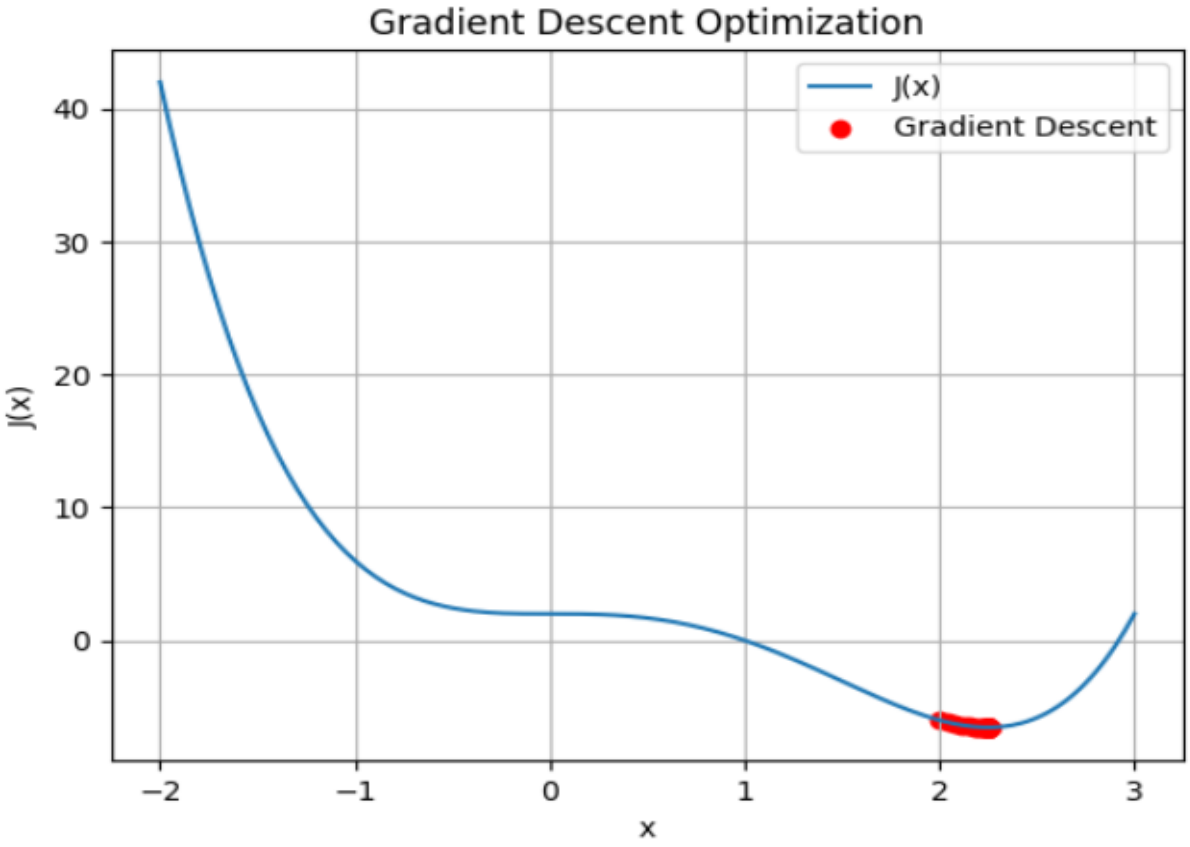
- Giá trị ban đầu: x = 2
- Đạo hàm riêng: dJ(x)/dx = 4x^3 - 9x^2 = 4 * 2^3 - 9 * 2^2 = 32 - 36 = -4
- Cập nhật tham số: x = x - α * dJ(x)/dx = 2 - 0.01 * (-4) = 2.04

Vòng lặp 2:

- Giá trị ban đầu: x = 2.04
- Đạo hàm riêng: dJ(x)/dx = 4x^3 - 9x^2 = 4 * 2.04^3 - 9 * 2.04^2 = 33.055 - 36.734 = -3.679
- Cập nhật tham số: x = x - α * dJ(x)/dx = 2.04 - 0.01 * (-3.679) = 2.07779

Tiếp tục lặp lại các bước trên cho đến khi đạt được số vòng lặp đã cho. Kết quả cuối cùng có thể tiến gần đến giá trị tối thiểu của hàm mất mát.

```
Iteration 10: x = 2.2161, J(x) = -6.5316
Iteration 20: x = 2.2464, J(x) = -6.5428
Iteration 30: x = 2.2496, J(x) = -6.5430
Iteration 40: x = 2.2500, J(x) = -6.5430
Iteration 50: x = 2.2500, J(x) = -6.5430
Iteration 60: x = 2.2500, J(x) = -6.5430
Iteration 70: x = 2.2500, J(x) = -6.5430
Iteration 80: x = 2.2500, J(x) = -6.5430
Iteration 90: x = 2.2500, J(x) = -6.5430
Iteration 100: x = 2.2500, J(x) = -6.5430
```



- Mục tiêu của Gradient Descent là tìm giá trị tối thiểu của hàm mất mát bằng cách di chuyển dần các tham số theo hướng ngược với gradient. Quá trình này tiếp tục cho đến khi đạt được điều kiện dừng, chẳng hạn như số lượng vòng lặp đã đủ hoặc thay đổi của hàm mất mát rất nhỏ.

1.3 ADAM (ADAPTIVE MOMENT ESTIMATION)

- Adam (Adaptive Moment Estimation) là một thuật toán tối ưu hóa gradient được sử dụng trong quá trình huấn luyện mạng neural và các mô hình học máy khác. Nó kết hợp cả hai khái niệm chính là Momentum và RMSProp để tối ưu hóa việc cập nhật trọng số của mạng.

- Thuật toán Adam tính toán một mức độ cập nhật động cho từng tham số dựa trên lịch sử của gradient trước đó. Nó lưu trữ một trung bình chuyển động (moving average) của gradient và một trung bình chuyển động của bình phương gradient. Các trung bình chuyển động này được sử dụng để tính toán hệ số cập nhật cho từng tham số trong quá trình huấn luyện.

- Các bước chính của Adam:
  + Bước 1: Khởi tạo các tham số ban đầu của mô hình và các biến đệm (buffers) cho các bước tính toán.
  + Bước 2: Tính toán gradient của hàm mất mát theo từng tham số.
  + Bước 3: Cập nhật các tham số bằng cách tính toán một hệ số điều chỉnh dựa trên gradient, các biến đệm và tỷ lệ học tập (learning rate). Adam sử dụng hai biến đệm để theo dõi độ mượt của gradient và độ lớn của gradient trung bình. Hai biến đệm này được tính toán thông qua các phép tính trung bình động (exponential moving average). Các tham số được cập nhật theo công thức:

    m_t = beta1 * m_t-1 + (1 - beta1) * g

    v_t = beta2 * v_t-1 + (1 - beta2) * g^2

    m_hat = m_t / (1 - beta1^t)

    v_hat = v_t / (1 - beta2^t)

    theta = theta - alpha * m_hat / (sqrt(v_hat) + epsilon)

Trong đó:

- m_t và v_t là các biến đệm của gradient và gradient bình phương trung bình.
- m_hat và v_hat là các phiên bản điều chỉnh của biến đệm để khắc phục hiện tượng bias ban đầu.
- beta1 và beta2 là các hệ số giữ lại (decay rates) cho các biến đệm.
- alpha là tỷ lệ học tập (learning rate).
- t là chỉ số vòng lặp.
- epsilon là một giá trị rất nhỏ được thêm vào mẫu số để tránh chia cho 0.

+ Bước 4: Lặp lại các bước 2 và 3 cho đến khi đạt được điều kiện dừng, ví dụ như đạt được số lần lặp tối đa hoặc đạt được sự hội tụ của các tham số.

- Adam kết hợp cả yếu tố của Momentum và RMSprop để đạt được sự kết hợp giữa việc điều chỉnh đường đi và tỷ lệ học tập. Điều này giúp thuật toán Adam có khả năng tìm được giá trị tối ưu nhanh hơn và ổn định hơn trong nhiều tình huống.

- Tuy nhiên, việc sử dụng Adam cũng đòi hỏi nhiều tham số để được cấu hình, và không phải lúc nào cũng mang lại hiệu suất tốt nhất. Việc lựa chọn thuật toán tối ưu hóa phụ thuộc vào loại bài toán và các yếu tố khác nhau như kích thước tập dữ liệu, độ lớn của gradient, và yêu cầu về tốc độ và hiệu suất.

1.4 RMSPROP (ROOT MEAN SQUARE PROPAGATION)

- Thuật toán RMSPROP (Root Mean Square Propagation) là một phương pháp tối ưu hóa được sử dụng trong máy học và tối ưu hóa. Nó là một biến thể của Gradient Descent và được thiết kế để giảm tốc độ học của các tham số khi chúng gần đến giá trị tối ưu.

- Ý tưởng cơ bản của RMSPROP là dựa vào đạo hàm của hàm mất mát để điều chỉnh tỷ lệ cập nhật của các tham số, tương tự như Gradient Descent. Tuy nhiên, RMSPROP sử dụng một kỹ thuật đặc biệt để điều chỉnh learning rate dựa trên lịch sử của các gradient đã tính toán trước đó.

- Các bước chính của RMSprop:

+ Bước 1: Khởi tạo các tham số ban đầu của mô hình.
+ Bước 2: Khởi tạo một bộ đệm (buffer) để lưu trữ và cập nhật lịch sử của các gradient đã tính toán trước đó.
+ Bước 3: Tính toán đạo hàm riêng của hàm mất mát theo từng tham số.
+ Bước 4: Cập nhật bộ đệm bằng cách tính toán trung bình bình phương của các gradient đã tính toán trước đó.

$$E[g^2]_t = 0{,}9E[g^2]_{t\text{-}1} + 0{,}1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

+ Bước 5: Cập nhật các tham số bằng cách di chuyển theo hướng đối nghịch với đạo hàm và với một tỷ lệ học tập được điều chỉnh bởi bộ đệm đã tính toán ở bước trước.
+ Bước 6: Lặp lại bước 3 đến bước 5 cho đến khi đạt được điều kiện dừng.

- Ví dụ, để áp dụng thuật toán RMSProp với hàm mục tiêu f(x) = x^4-3x^3+ 2, cần thêm một số tham số như learning rate α và β. Giả sử chúng ta chọn α = 0.1 và β = 0.9.

+ Bước 1: Khởi tạo giá trị ban đầu cho tham số x, chẳng hạn x = 2, và khởi tạo giá trị ban đầu cho bộ nhớ đệm S là 0.

+ Bước 2: Lặp lại các bước sau cho mỗi vòng lặp:

Vòng lặp 1:

- Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2 = 2^4 - 3 * 2^3 + 2 = 2.
- Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx = 4x^3 - 9x^2 = 4 * 2^3 - 9 * 2^2 = 32 - 36 = -4.
- Cập nhật bộ nhớ đệm S: S = β * S + (1 - β) * (gradient^2) = 0.9 * 0 + (1 - 0.9) * (-4)^2 = 3.6.
- Cập nhật tham số x bằng cách sử dụng learning rate α và bộ nhớ đệm S: x = x - α * gradient / sqrt(S + ε) = 2 - 0.1 * (-4) / sqrt(3.6 + 10^-8) ≈ 2.021.

Vòng lặp 2:

- Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2 = 2.021^4 - 3 * 2.021^3 + 2 ≈ 2.036.
- Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx = 4x^3 - 9x^2 ≈ -3.768.
- Cập nhật bộ nhớ đệm S: S = β * S + (1 - β) * (gradient^2) ≈ 0.9 * 3.6 + (1 - 0.9) * (-3.768)^2 ≈ 3.733.
- Cập nhật tham số x bằng cách sử dụng learning rate α và bộ nhớ đệm S: x = x - α * gradient / sqrt(S + ε) ≈ 2.021 - 0.1 * (-3.768) / sqrt(3.733 + 10^-8) ≈ 2.041.

Tiếp tục lặp lại các bước trên cho đến khi đạt được điều kiện dừng (ví dụ như đạt đến một số lần lặp tối đa hoặc giá trị hàm mất mát hội tụ đến một ngưỡng nhất định).

- Điểm mạnh của thuật toán RMSProp là nó có khả năng tự điều chỉnh learning rate dựa trên lịch sử các gradient trước đó, giúp tăng tốc quá trình hội tụ và tránh việc learning rate quá lớn hoặc quá nhỏ. Nó thường được sử dụng trong các mô hình học sâu như mạng nơ-ron hồi quy (RNN) và mạng nơ-ron tích chập (CNN).

1.5 ADAGRAD (ADAPTIVE GRADIENT)

- Thuật toán ADAGRAD (Adaptive Gradient) là một phương pháp tối ưu hóa trong máy học và tối ưu hóa, được thiết kế để điều chỉnh tỷ lệ học của các tham số dựa trên tần suất xuất hiện của các gradient trong quá trình huấn luyện.

- Ý tưởng cơ bản của ADAGRAD là sử dụng bộ nhớ đệm để tích lũy các bình phương của các gradient đã tính toán trước đó. Việc tích lũy này sẽ tạo ra một giá trị đệm riêng cho mỗi tham số, và các giá trị này sẽ được sử dụng để điều chỉnh learning rate của từng tham số trong quá trình cập nhật.

- Các bước chính của AdaGrad:

  + Bước 1: Khởi tạo các tham số ban đầu của mô hình.
  + Bước 2: Khởi tạo một bộ đệm (buffer) để tích lũy các bình phương của các gradient đã tính toán trước đó.
  + Bước 3: Tính toán đạo hàm riêng của hàm mất mát theo từng tham số.
  + Bước 4: Cập nhật bộ đệm bằng cách tích lũy bình phương của gradient hiện tại.
  + Bước 5: Cập nhật các tham số bằng cách di chuyển theo hướng đối nghịch với đạo hàm và với một tỷ lệ học tập được điều chỉnh bởi bộ đệm đã tính toán ở bước trước.
  + Bước 6: Lặp lại bước 3 đến bước 5 cho đến khi đạt được điều kiện dừng.

- Ví dụ, để áp dụng thuật toán ADAGRAD với hàm mục tiêu f(x) = x^4 - 3x^3 + 2, cần thêm một số tham số như learning rate α và chọn α = 0.1.

  + Bước 1: Khởi tạo giá trị ban đầu cho tham số x, chẳng hạn x = 2, và khởi tạo giá trị ban đầu cho bộ đệm là 0.
  + Bước 2: Lặp lại các bước sau cho mỗi vòng lặp:

Vòng lặp 1:

- Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2 = 2^4 - 3 * 2^3 + 2 = 2.
- Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx = 4x^3 - 9x^2 = 4 * 2^3 - 9 * 2^2 = 32 - 36 = -4.
- Cập nhật bộ đệm bằng cách tích lũy bình phương của gradient hiện tại: bộ đệm = bộ đệm + (gradient)^2 = 0 + (-4)^2 = 16.
- Cập nhật tham số x bằng cách sử dụng learning rate α và bộ đệm: x = x - α * gradient / (sqrt(bộ đệm) + ε) = 2 - 0.1 * (-4) / (sqrt(16) + 10^-8) ≈ 2.025.

Vòng lặp 2:

- Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2 = 2.025^4 - 3 * 2.025^3 + 2 ≈ 1.935.
- Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx = 4x^3 - 9x^2 = 4 * 2.025^3 - 9 * 2.025^2 ≈ -3.222.
- Cập nhật bộ đệm bằng cách tích lũy bình phương của gradient hiện tại: bộ đệm = bộ đệm + (gradient)^2 = 16 + (-3.222)^2 ≈ 27.21.
- Cập nhật tham số x bằng cách sử dụng learning rate α và bộ đệm: x = x - α * gradient / (sqrt(bộ đệm) + ε) = 2.025 - 0.1 * (-3.222) / (sqrt(27.21) + 10^-8) ≈ 2.041

- AdaGrad cung cấp quá trình tối ưu hóa tự động và hiệu quả cho các bài toán với dữ liệu thưa và các tham số có độ lớn gradient khác nhau. Nó giúp giảm tỷ lệ học tập đối với các tham số quan trọng và tăng tỷ lệ học tập đối với các tham số không quan trọng, giúp tăng tốc độ hội tụ và ổn định quá trình tối ưu hóa.

- Tuy nhiên, một nhược điểm của ADAGRAD là việc tích lũy bình phương của gradient có thể dẫn đến việc giá trị bộ đệm trở nên quá lớn và khiến tỷ lệ học tập rất nhỏ, dẫn đến hiện tượng dừng đột ngột của quá trình tối ưu hóa.

1.6 ADADELTA

- Adadelta là một thuật toán tối ưu hóa trong lĩnh vực học máy và mạng nơ-ron, là một biến thể của AdaGrad. Adadelta giải quyết nhược điểm của AdaGrad bằng cách giới hạn quỹ đạo lịch sử gradient và làm giảm độ giảm learning rate theo thời gian.

- ADADELTA (Adaptive Delta) cũng là một thuật toán tối ưu hóa trong máy học và tối ưu hóa, được phát triển dựa trên ý tưởng của ADAGRAD và giải quyết một số nhược điểm của nó, như việc tích lũy giá trị bộ đệm quá lớn.

- Các bước chính của Adadelta:

  + Bước 1: Khởi tạo các tham số ban đầu của mô hình.
  + Bước 2: Khởi tạo các bộ đệm để tích lũy các bình phương của các gradient đã tính toán trước đó.
  + Bước 3: Tính toán đạo hàm riêng của hàm mất mát theo từng tham số.
  + Bước 4: Cập nhật bộ đệm bằng cách tích lũy một phần của các bình phương gradient hiện tại và một phần của các bình phương gradient trước đó.
  + Bước 5: Cập nhật các tham số bằng cách di chuyển theo hướng đối nghịch với đạo hàm và sử dụng một tỷ lệ học tập được điều chỉnh bởi bộ đệm đã tính toán ở bước trước.
  + Bước 6: Lặp lại bước 3 đến bước 5 cho đến khi đạt được điều kiện dừng.

- Ví dụ, để áp dụng thuật toán AdaDelta cho hàm mục tiêu f(x) = x^4 - 3x^3 + 2, cần thêm một số tham số như rho và chọn rho = 0.9.

+ Bước 1: Khởi tạo giá trị ban đầu cho tham số x, chẳng hạn x = 2, và khởi tạo giá trị ban đầu cho các bộ đệm là 0.
+ Bước 2: Lặp lại các bước sau cho mỗi vòng lặp:

  Vòng lặp 1:

- Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2 = 2^4 - 3 * 2^3 + 2 = 2.

- Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx = 4x^3 - 9x^2 = 4 * 2^3 - 9 * 2^2 = 32 - 36 = -4.
- Cập nhật bộ đệm thứ nhất bằng cách tích lũy bình phương của gradient hiện tại: bộ đệm thứ nhất = rho * bộ đệm thứ nhất + (1 - rho) * (gradient)^2 = 0.9 * 0 + (1 - 0.9) * (-4)^2 = 7.2.
- Tính toán sự thay đổi của các tham số: delta = - (sqrt(bộ đệm trước đó) + ε) / (sqrt(bộ đệm) + ε) * gradient = - (sqrt(0) + 10^-8) / (sqrt(7.2) + 10^-8) * (-4) ≈ 0.001Tiếp tục từ vòng lặp trước đó:
- Cập nhật bộ đệm thứ hai bằng cách tích lũy bình phương của các thay đổi trước đó của các tham số: bộ đệm thứ hai = rho * bộ đệm thứ hai + (1 - rho) * delta^2 = 0.9 * 0 + (1 - 0.9) * 0.001^2 = 8.9999e-7.
- Cập nhật tham số x: x = x + delta = 2 + 0.001 = 2.001.

  Vòng lặp 2:
- Tính giá trị hàm mất mát J(x) tại điểm x hiện tại: J(x) = f(x) = x^4 - 3x^3 + 2 = 2.001^4 - 3 * 2.001^3 + 2 ≈ 2.002.
- Tính đạo hàm riêng của hàm mất mát J(x) theo tham số x: dJ(x)/dx ≈ -3.996.
- Cập nhật bộ đệm thứ nhất: bộ đệm thứ nhất = 0.9 * 7.2 + (1 - 0.9) * (-3.996)^2 ≈ 8.0976.
- Tính toán sự thay đổi của các tham số: delta ≈ - (sqrt(7.2) + 10^-8) / (sqrt(8.0976) + 10^-8) * (-3.996) ≈ 0.0004918.
- Cập nhật bộ đệm thứ hai: bộ đệm thứ hai = 0.9 * 8.9999e-7 + (1 - 0.9) * 0.0004918^2 ≈ 9.091e-7.
- Cập nhật tham số x: x = x + delta ≈ 2.001 + 0.0004918 ≈ 2.0014918.

Bước này được lặp lại cho đến khi đạt được điều kiện dừng, chẳng hạn là khi đạt tới một số vòng lặp tối đa hoặc khi gradient đạt được một ngưỡng nhất định.

- Điểm mạnh của Adadelta là nó tự động điều chỉnh tỷ lệ học tập dựa trên lịch sử của gradient và thay đổi tham số; cải thiện tốc độ hội tụ của quá trình tối ưu hóa và ổn định hơn; kiểm soát tỷ lệ học tập tự động và giảm thiểu sự điều chỉnh tỷ lệ học tập cần thiết từ phía người dùng.

- Mặc dù Adadelta là một thuật toán tối ưu hóa hiệu quả, nhưng nó không phải lúc nào cũng phù hợp cho mọi bài toán. Có những bài toán đặc thù hoặc có cấu trúc dữ liệu đặc biệt mà Adadelta có thể không đạt được hiệu suất tối ưu.

1.7 SO SÁNH CÁC PHƯƠNG PHÁP OPTIMIZER

Đặc điểm	GD	Adam	RMSprop	AdaGrad	Adadelta
Quản lý tỷ lệ học tập	Cần điều chỉnh thủ công	Tự động điều chỉnh (adaptive)	Tự động điều chỉnh (adaptive)	Tự động điều chỉnh (adaptive)	Tự động điều chỉnh (adaptive)
Momentum	Không	Có	Không	Không	Có
Tính năng thích ứng	Không	Có	Có	Có	Có
Hiệu suất hội tụ	Trung bình	Cao	Cao	Thấp	Cao
Sử dụng bộ nhớ	Thấp	Cao	Thấp	Cao	Cao
Tính toán hiệu suất	Thấp	Cao	Cao	Cao	Cao

Bảng so sánh các phương pháp Optimizer

CHƯƠNG 2 – CONTINUAL LEARNING VÀ TEST PRODUCTION

2.1 Khái niệm

- Continual Learning (CL), còn được gọi là Lifelong Learning, là một lĩnh vực trong học máy nhằm tìm hiểu và phát triển các phương pháp và thuật toán để cho phép mô hình học máy liên tục học và thích ứng với dữ liệu mới theo thời gian mà không cần huấn luyện lại từ đầu trên toàn bộ dữ liệu đã sử dụng trước đó.
- Test Production là quá trình xây dựng và triển khai hệ thống để kiểm tra hiệu suất và đánh giá mô hình học máy. Nó đảm bảo rằng mô hình hoạt động đúng đắn và có hiệu suất tốt trên dữ liệu mới và trong môi trường thực tế.

2.2 Các phương pháp và kỹ thuật Continual Learning

- Elastic Weight Consolidation (EWC): EWC là một phương pháp quan trọng trong CL giúp giữ kiến thức cũ trong mô hình bằng cách áp dụng một hàm mất mát phạt các thay đổi quá lớn của trọng số quan trọng trong quá trình huấn luyện mới. Nó cho phép mô hình ưu tiên giữ lại những thông tin quan trọng từ nhiệm vụ trước đó.
- Generative Replay: Generative Replay sử dụng mô hình sinh dữ liệu (như mạng tổng quát hoá (GAN)) để tạo ra các mẫu dữ liệu từ các nhiệm vụ trước đó và sử dụng chúng để huấn luyện lại mô hình trên các nhiệm vụ mới. Điều này giúp mô hình không quên kiến thức đã học và đồng thời cung cấp dữ liệu đa dạng cho quá trình huấn luyện.
- Memory-based Methods: Các phương pháp dựa trên bộ nhớ sử dụng một bộ nhớ ngoài để lưu trữ các mẫu dữ liệu quan trọng từ các nhiệm vụ trước đó. Khi huấn luyện trên nhiệm vụ mới, mô hình sẽ truy cập vào bộ nhớ và sử dụng các mẫu dữ liệu cũ để đưa ra dự đoán và cập nhật trọng số.
- Progressive Neural Networks (PNN): PNN là một kiến trúc mạng nơ-ron đa tầng cho phép mô hình mở rộng bằng cách thêm lớp mới cho mỗi nhiệm vụ mới. Các lớp mới sẽ học cách xử lý thông tin của nhiệm vụ mới trong khi giữ nguyên kiến thức của các nhiệm vụ trước đó trong các lớp cũ.

- Dynamic Architectures: Kỹ thuật này thay đổi kiến trúc của mô hình để phù hợp với các nhiệm vụ mới. Các phương pháp như Network Expansion, Network Splitting và Network Masking cho phép mô hình thích ứng với sự biến đổi của đầu vào và yêu cầu của nhiệm vụ mới.
- Meta-learning: Meta-learning, còn được gọi là học học, là một lĩnh vực trong Continual Learning tập trung vào việc học cách học. Phương pháp này tập trung vào việc xây dựng các thuật toán và mô hình có khả năng học nhanh và thích ứng với các tác vụ mới một cách hiệu quả.

2.3 Các phương pháp Test Production

* Có nhiều phương pháp và kỹ thuật được sử dụng trong quá trình Test Production để kiểm tra hiệu suất của mô hình học máy:

- Unit Testing: Unit Testing tập trung vào việc kiểm tra từng phần riêng lẻ của mô hình hoặc các thành phần phụ thuộc như hàm, lớp hoặc module. Điều này đảm bảo rằng các phần của mô hình hoạt động đúng đắn và đáng tin cậy.
- Integration Testing: Integration Testing kiểm tra sự tương tác và tích hợp giữa các thành phần khác nhau của mô hình. Nó đảm bảo rằng các thành phần hoạt động hợp lý khi được kết hợp và giao tiếp với nhau.
- End-to-End Testing: End-to-End Testing kiểm tra hoạt động của toàn bộ hệ thống từ đầu đến cuối, bao gồm cả quá trình tiền xử lý dữ liệu, huấn luyện mô hình, dự đoán và đầu ra kết quả. Nó đánh giá hiệu suất của mô hình trong môi trường thực tế.
- Performance Testing: Performance Testing đánh giá hiệu suất và khả năng mở rộng của mô hình trong nhiều tình huống khác nhau. Nó có thể đo lường thời gian phản hồi, tốc độ xử lý, tài nguyên tiêu thụ và khả năng xử lý tải công việc lớn.
- A/B Testing: A/B Testing so sánh hiệu suất của hai phiên bản mô hình hoạt động song song. Một phiên bản là phiên bản hiện tại đang được sử dụng, và phiên bản còn lại là một phiên bản mới hoặc đã được cải tiến. Điều này giúp đánh giá liệu phiên bản mới có hiệu suất tốt hơn hay không.

- Cross-Validation: Cross-Validation được sử dụng để đánh giá hiệu suất của mô hình trên một tập dữ liệu đã biết. Nó chia dữ liệu thành các tập con và thực hiện kiểm tra và đánh giá trên các tập con này để đánh giá hiệu suất trung bình của mô hình.\

* Sự lựa chọn phương pháp và kỹ thuật phụ thuộc vào loại mô hình, bài toán cụ thể và mục tiêu của dự án.

2.4 Sử dụng Continual Learning và Test Production khi xây dựng một giải pháp học máy để giải quyết một bài toán

2.4.1 Bài toán 1:

* Bài toán phân loại hình ảnh. Giả sử có nhiệm vụ phân loại hình ảnh khác nhau: phân loại loài hoa, phân loại động vật và phân loại phương tiện giao thông.

- Bước 1: Huấn luyện mô hình ban đầu
    + Sử dụng dữ liệu huấn luyện ban đầu để huấn luyện mô hình phân loại hình ảnh cho nhiệm vụ phân loại loài hoa. Mô hình này sẽ được huấn luyện trên tập dữ liệu loài hoa và đạt được độ chính xác đáng kể trên nhiệm vụ này.
- Bước 2: Giữ vững kiến thức đã học
    + Áp dụng phương pháp Continual Learning (ví dụ: Elastic Weight Consolidation) để giữ vững kiến thức đã học từ nhiệm vụ phân loại loài hoa. Đảm bảo rằng các trọng số quan trọng cho nhiệm vụ phân loại loài hoa không bị thay đổi quá nhiều trong quá trình học nhiệm vụ mới.
- Bước 3: Tiếp tục học trên nhiệm vụ phân loại động vật
    + Thêm dữ liệu huấn luyện cho nhiệm vụ phân loại động vật vào mô hình. Tiếp tục huấn luyện mô hình trên tập dữ liệu động vật. Sử dụng phương pháp Continual Learning đã áp dụng ở bước trước để đảm bảo rằng mô hình không quên kiến thức đã học từ nhiệm vụ phân loại loài hoa.
- Bước 4: Tiếp tục học trên nhiệm vụ phân loại phương tiện giao thông
    + Thêm dữ liệu huấn luyện cho nhiệm vụ phân loại phương tiện giao thông vào mô hình. Tiếp tục huấn luyện mô hình trên tập dữ liệu phương tiện giao thông.

Sử dụng phương pháp Continual Learning để đảm bảo rằng mô hình không quên kiến thức đã học từ các nhiệm vụ trước đó.

- Bước 5: Tạo dữ liệu kiểm tra đáng tin cậy
    + Áp dụng phương pháp Test Production như Cross-Validation, Stratified Sampling hoặc Bootstrapping để tạo ra các tập dữ liệu kiểm tra đáng tin cậy cho cả ba nhiệm vụ. Điều này giúp đánh giá hiệu suất của mô hình trên các nhiệm vụ phân loại khác nhau.
- Bước 6: Đánh giá mô hình
    + Sử dụng dữ liệu kiểm tra đã tạo ra để đánh giá hiệu suất của mô hình trên các nhiệm vụ phân loại loài hoa, động vật và phương tiện giao thông. Điều này giúp đảm bảo tính chính xác và độ tin cậy của mô hình trên các nhiệm vụ khác nhau.
- Qua các bước trên, ta có thể xây dựng một giải pháp học máy để phân loại hình ảnh trong môi trường học liên tục, đồng thời đảm bảo tính liên tục, tính chính xác và độ tin cậy của mô hình trên các nhiệm vụ khác nhau.

2.4.2 Bài toán 2:

Bài toán phân loại văn bản. Giả sử có nhiệm vụ phân loại văn bản khác nhau: phân loại tin tức, phân loại đánh giá sản phẩm và phân loại email.

- Bước 1: Huấn luyện mô hình ban đầu
    + Sử dụng dữ liệu huấn luyện ban đầu để huấn luyện mô hình phân loại văn bản cho nhiệm vụ phân loại tin tức. Mô hình này sẽ được huấn luyện trên tập dữ liệu tin tức và đạt được độ chính xác đáng kể trên nhiệm vụ này.
- Bước 2: Giữ vững kiến thức đã học
    + Áp dụng phương pháp Continual Learning (ví dụ: Elastic Weight Consolidation) để giữ vững kiến thức đã học từ nhiệm vụ phân loại tin tức. Đảm bảo rằng các trọng số quan trọng cho nhiệm vụ phân loại tin tức không bị thay đổi quá nhiều trong quá trình học nhiệm vụ mới.

- Bước 3: Tiếp tục học trên nhiệm vụ phân loại đánh giá sản phẩm
    + Thêm dữ liệu huấn luyện cho nhiệm vụ phân loại đánh giá sản phẩm vào mô hình. Tiếp tục huấn luyện mô hình trên tập dữ liệu đánh giá sản phẩm. Sử dụng phương pháp Continual Learning đã áp dụng ở bước trước để đảm bảo rằng mô hình không quên kiến thức đã học từ nhiệm vụ phân loại tin tức.
- Bước 4: Tiếp tục học trên nhiệm vụ phân loại email
    + Thêm dữ liệu huấn luyện cho nhiệm vụ phân loại email vào mô hình. Tiếp tục huấn luyện mô hình trên tập dữ liệu email. Sử dụng phương pháp Continual Learning để đảm bảo rằng mô hình không quên kiến thức đã học từ các nhiệm vụ trước đó.
- Bước 5: Tạo dữ liệu kiểm tra đáng tin cậy
    + Áp dụng phương pháp Test Production như Cross-Validation, Stratified Sampling hoặc Bootstrapping để tạo ra các tập dữ liệu kiểm tra đáng tin cậy cho cả ba nhiệm vụ. Điều này giúp đánh giá hiệu suất của mô hình trên các nhiệm vụ phân loại khác nhau.
- Bước 6: Đánh giá mô hình
    + Sử dụng dữ liệu kiểm tra đã tạo ra để đánh giá hiệu suất của mô hình trên các nhiệm vụ phân loại tin tức, đánh giá sản phẩm và phân loại email. Điều này giúp đảm bảo tính chính xác và độ tin cậy của mô hình trên các nhiệm vụ khác nhau.

* Qua các bước trên, ta có thể xây dựng một giải pháp học máy để phân loại văn bản trong môi trường học liên tục, đồng thời đảm bảo tính liên tục, tính chính xác và độ tin cậy của mô hình trên các nhiệm vụ khác nhau.

TÀI LIỆU THAM KHẢO

[1] Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,..) https://viblo.asia/p/optimizer-hieu-sau-ve-cac-thuat-toan-toi-uu-gdsgdadam-Qbq5QQ9E5D8

[2] Gradient Descent https://machinelearningcoban.com/2017/01/12/gradientdescent/#gradient-descent

[3] Continual Learning https://www.pnas.org/doi/10.1073/pnas.1611835114

[4] Optimizer- Hiểu sâu về các thuật toán tối ưu (GD,SGD,Adam,.) https://sentayho.com.vn/optimizer-la-gi

[5] Test production https://applyingml.com/resources/testing-ml/

[6] EWC https://paperswithcode.com/method/ewc

[7] AdaGrad - Optimization Wiki https://tek4.vn/khoa-hoc/machine-learning-co-ban/thuat-toan-adagrad-va-van-de-hieu-chinh-learning-rate

[8] RMSPROP VS ADAM https://www.lightly.ai/post/which-optimizer-should-i-use-for-my-machine-learning-project