Dự án Cuối kỳ
Môn: Nhập môn Học Máy

Bài 1 (3 điểm): làm riêng từng người
Trình bày một bài nghiên cứu, đánh giá của em về các vấn đề sau:

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

Giải đáp

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

❖ Optimizer là gì?

Optimizer là các thuật toán điều chỉnh các tham số của mô hình trong quá trình huấn luyện để giảm thiểu hàm mất(loss function). Ta cho phép mạng nơ-ron học hỏi từ dữ liệu bằng cách cập nhật lặp đi lặp lại trọng số và độ lệch.

Optimizer có tác vụ quan trọng là tính toán gradient của hàm mất mát (loss function) theo các tham số của mô hình. Gradient là vector đạo hàm riêng của hàm mất mát theo từng tham số, và cho biết hướng và mức độ thay đổi của hàm mất mát khi các tham số thay đổi. Gradient sẽ điều chỉnh các tham số của mô hình để di chuyển mô hình gần hơn đến điểm tối ưu của hàm mất mát.

❖ Các đặc trưng quan trọng của một optimizer

- Tốc độ hội tụ là khả năng của optimizer để đạt tới điểm cực tiểu của hàm mục tiêu trong thời gian ngắn.

- Khả năng tránh điểm cực tiểu cục bộ là điểm mà hàm mục tiêu có giá trị nhỏ nhất trong một khu vực nhưng không phải là giá trị nhỏ nhất toàn cục.

- Độ ổn định đo lường khả năng của optimizer để giữ được độ chính xác và tính khả dụng ngay cả khi có nhiễu trong dữ liệu hoặc khi tham số khởi tạo ban đầu không tốt.

- Khả năng xử lý các tham số thưa (sparse parameters). Các bài toán machine learning có thể có các tham số không quan trọng hoặc không được sử dụng rộng rãi trong quá trình tối ưu hóa.

- Khả năng xử lý tập dữ liệu lớn: có khả năng xử lý tốt trên các tập dữ liệu lớn.
- Độ phức tạp tính toán của optimizer đo lường số lượng phép tính toán và bộ nhớ cần thiết để thực hiện quá trình tối ưu hóa.
- Dễ triển khai và sử dụng: Optimizer nên có sẵn trong các thư viện và framework phổ biến và có các giao diện dễ sử dụng để cấu hình và điều chỉnh các tham số.

❖ Các phương pháp Optimizer phổ biến trong huấn luyện mô hình học máy:

1. Gradient Descent (GD)

Gradient descent là một thuật toán tối ưu hóa được sử dụng để tìm giá trị tối thiểu cục bộ của một hàm khác biệt. Trong machine learning, gradient descent được sử dụng để tìm các giá trị của các tham số của một mô hình học máy sao cho hàm chi phí của mô hình được tối thiểu hóa.

Gradient Descent sử dụng gradient của hàm mục tiêu để điều chỉnh các tham số. Hơn nữa, nó có hai biến thể: Batch Gradient Descent (BGD) và Stochastic Gradient Descent (SGD). BGD tính toán gradient trên toàn bộ tập dữ liệu đào tạo, trong khi SGD tính toán gradient trên từng mẫu dữ liệu một cách ngẫu nhiên.

Độ tin cậy: Trong trường hợp hàm mục tiêu là lồi và tốtích cực được đạt tới, GD có thể đạt được độ tin cậy cao.

Các bước thực hiện của gradient descent:

+ Bước 1: Khỏi tạo các tham số của mô hình tức là các giá trị cầ được tìm kiếm. Các tham số này có thể là các trọng số, góc hoặc các tham số khác. Các tham số thường được khởi tạo ngẫu nhiên.

+ Bước 2: Tính gradient của hàm chi phí là hướng mà hàm chi phí giảm nhanh nhất. Gradient được tính bằng cách sử dụng đạo hàm của hàm chi phí.

+ Bước 3 : Cập nhật các tham số được cập nhật theo hướng ngược lại với gradient của hàm chi phí. Cập nhật các tham số được thực hiện bằng cách sử dụng công thức sau:

Công thức : xnew = xold – learning_rate*gradient(x)

- xnew : là giá trị mới của tham số,
- xold : là giá trị hiện tị của tham số,
- learning_rate : là hệ số học và gradient là gradient tính toán được.

+ Bước 4: lặp lại các bước 2 và 3 được lặp lại cho đến khi hàm chi phí đạt đến một giá trị thỏa mãn hoặc cho đến khi đạt đến một số giới hạn vòng lặp nhất định.

2. Momentum

Monmentum là một kỹ thuật tối ưu hóa được sử dụng để cải thiện tốc độ hội tụ của thuật toán Gradient Descent. Momentum giúp giảm độ dao động và giúp vượt qua các vùng yên tĩnh (flat regions) của hàm mất mát nhanh hơn. Nó cũng có thể giúp tránh rơi vào các điểm tối thiểu cục bộ và điều chỉnh tốt hơn khi hàm mất mát có độ cong thay đổi đáng kể.

Momentum Optimization sử dụng khái niệm "momentum" để tăng tốc quá trình tối ưu hóa. Momentum tính toán trung bình của các gradient trước đó và cập nhật tham số.

Độ tin cậy: Momentum Optimization cải thiện tốc độ hội tụ bằng cách tích lũy một lượng momentum từ các bước trước đó. Điều này giúp vượt qua các điểm cực tiểu cục bộ và giảm dao động.

Cách hoạt động:

+ Bước 1: Tại mỗi bước cập nhật, tính toán gradient của hàm mất mát tại tham số hiện tại.

+ Bước 2: Cập nhật vector momentum bằng cách kết hợp gradient hiện tại với vector momentum trước đó.

+ Bước 3: Cập nhật tham số bằng cách sử dụng vector momentum tính toán được và learning rate (tỷ lệ học). Vector momentum tạo ra một "đà" cho quá trình cập nhật, giúp tránh những dao động và hướng thẳng tiến hơn đến điểm tối ưu.

Các bước thực hiện Gradient Descent with Momentum:

+ Bước 1: Khởi tạo các thông số ban đầu cho tham số mô hình, learning rate (tỷ lệ học), hệ số momentum và các tham số khác liên quan.

+ Bước 2: Khởi tạo vector momentum ban đầu với các phần tử có giá trị ban đầu là 0 hoặc một giá trị khác tùy thuộc vào triển khai cụ thể.

+ Bước 3: Vòng lặp huấn luyện

- Tính toán gradient của hàm mất mát tại tham số hiện tại bằng cách sử dụng một mini-batch hoặc một mẫu dữ liệu.

- Sử dụng công thức sau để cập nhật vector momentum tại bước hiện tại

Công thức:
momentum = momentum * momentum_rate + learning_rate * gradient

- momentum là vector momentum
- momentum_rate là hệ số momentum (thường nằm trong khoảng từ 0 đến 1)
- learning_rate là tỷ lệ học
- gradient là gradient tính toán được.

- Sử dụng vector momentum tính toán được để cập nhật tham số mô hình
  Công thức
  tham_số_mô_hình = tham_số_mô_hình – momentum

+ Bước 4 : Lặp lại vòng lặp huấn luyện cho đến khi đạt được điều kiện dừng hoặc đủ số lần lặp.

- Sử dụng Momentum khi
  - Gradient Descent bị chậm hội tụ hoặc dễ mắc kẹt ở cực tiểu cục bộ.
  - Hàm chi phí phức tạp hoặc nhiều cực tiểu cục bộ.
  - Muốn cải thiện hiệu suất tìm kiếm điểm tối ưu.

3. Stochastic Gradient Descent (SGD)

SGD là một phương pháp tối ưu hóa được sử dụng rộng rãi trong machine learning và optimization. Nó là một biến thể của Gradient Descent (GD) truyền thống, nhưng thay vì tính toán gradient dựa trên toàn bộ tập dữ liệu huấn luyện, SGD tính toán gradient dựa trên từng mẫu dữ liệu đơn lẻ.

Độ tin cậy: SGD có thể hội tụ nhanh hơn GD vì nó chỉ sử dụng một mẫu dữ liệu ngẫu nhiên trong mỗi bước cập nhật.

Các bước thực hiện:

+ Bước 1: Khởi tạo tham số
+ Bước 2: Đối với mỗi mẫu dữ liệu trong tập huấn luyện, làm các bước sau:
  - Tính toán đầu ra của mô hình dựa trên tham số hiện tại.
  - Tính gradient của hàm mất mát theo các tham số của mô hình. Gradient được tính bằng cách sử dụng lan truyền ngược (backward

propagation), trong đó tính toán đạo hàm riêng của hàm mất mát theo từng tham số.

- cập nhật các tham số của mô hình theo công thức:

  thamsoNew = thamsoOld - learning_rate * gradient

  trong đó,

- thamsoNew là giá trị mới của tham số
- thamsoOld là giá trị hiện tại của tham số
- learning_rate là tỷ lệ học (learning rate)

+ Bước 3: Tiếp tục lặp lại bước 2 cho đến khi hoàn thành một số lượng xác định các mẫu dữ liệu (được gọi là epoch) hoặc đạt đủ tiêu chí dừng,

+ Bước 4: Sau khi hoàn thành quá trình huấn luyện, sử dụng tập dữ liệu kiểm tra để đánh giá hiệu suất của mô hình.

- Sử dụng SGD khi
  - Cần xử lý dữ liệu lớn và muốn tiết kiệm thời gian tính toán.
  - Dữ liệu ồn ào và bạn muốn mô hình ổn định.
  - Cần cập nhật mô hình liên tục với dữ liệu mới.

4. RMSprop

RMSprop được thiết kế để giảm độ dốc của hàm mất mát và điều chỉnh learning rate cho từng tham số.

RMSprop sử dụng bình phương độ dốc trung bình để điều chỉnh tốc độ học, giúp thuật toán hội tụ nhanh hơn và tránh bị mắc kẹt ở các cực tiểu cục bộ.

RMSprop giải quyết vấn đề tỷ lệ học giảm dần của Adagrad bằng cách chia tỷ lệ học cho trung bình của bình phương gradient. Nó tương tự như AdaGrad, RMSProp cũng điều chỉnh learning rate cho từng tham số.

RMSprop sử dụng một biến gọi là moving average để lưu trữ bình phương độ dốc trung bình:

$E_t = \beta * E_{(t-1)} + (1 - \beta) * (\nabla\theta\_cost(\theta\_t))^2$

Trong đó:

- E_t là moving average tại thời điểm t
- E_(t-1) là moving average tại thời điểm t-1
- β là một tham số điều chỉnh moving average
- ∇θ_cost(θ_t) là độ dốc của hàm chi phí tại điểm θ_t

Các bước thực hiện RMSprop:

+ Bước 1: Khởi tạo các tham số bao gồm các tham số mô hình chính (ví dụ: trọng số và bias) và các tham số RMSprop như learning rate (alpha) và hệ số giảm giá trị độ lớn của gradient (beta).

+ Bước 2: Tạo một biến đệm để lưu trữ lịch sử của gradient bình phương. Ban đầu, biến đệm được khởi tạo với giá trị 0.

+ Bước 3: Lặp lại các epochs, lặp qua từng mini-batch để tính toán gradient và cập nhật tham số theo công thức RMSprop.

+ Bước 4: tính toán gradient của hàm mất mát theo tham số hiện tại bằng cách sử dụng phương pháp lan truyền ngược (backpropagation) và các kỹ thuật tự động tính gradient.

+ Bước 5: Tính toán và cập nhật biến đệm để lưu trữ lịch sử của gradient bình phương.

- Tính toán gradient bình phương hiện tại:
  cache = beta * cache + (1 - beta) * (gradient ** 2)
- cache là biến đệm lưu trữ lịch sử của gradient bình phương.

+ Bước 6: Sử dụng biến đệm đã tính toán, cập nhật tham số mô hình theo công thức RMSprop:

tham_số_mô_hình_mới = tham_số_mô_hình_cũ - learning_rate * gradient / (sqrt(cache) + epsilon)

- epsilon là một giá trị rất nhỏ (thường là 1e-8) được thêm vào mẫu để tránh chia cho 0.

5. Adam

Adam (Adaptive Moment Estimation) là một phương pháp tối ưu hóa trong quá trình huấn luyện mô hình machine learning, kết hợp cả Momentum và RMSprop. Adam

được thiết kế để tự động điều chỉnh learning rate cho từng tham số dựa trên gradient và lịch sử của gradient. Adam giúp cải thiện tốc độ hội tụ và độ ổn định của mô hình.

Adam sử dụng hai thành phần để cập nhật tham số:

- Momentum: Adam sử dụng momen để giúp thuật toán hội tụ nhanh hơn và tránh bị mắc kẹt ở các cực tiểu cục bộ.

- Adaptive Learning Rate: Adam sử dụng adaptive learning rate để điều chỉnh tốc độ học dựa trên độ biến thiên của hàm chi phí.

Độ tin cậy: Adam (Adaptive Moment Estimation) kết hợp các lợi ích của Momentum Optimization và RMSProp. Nó có thể hội tụ nhanh chóng và ổn định trên nhiều bài toán.

Các bước thực hiện Adam :

+ Bước 1: Khởi tạo các tham số ban đầu, learning rate (alpha), hệ số giảm giá trị gradient (beta1) và hệ số giảm giá trị độ lớn của gradient (beta2).

+ Bước 2: Tạo các biến đệm để lưu trữ lịch sử của gradient và gradient bình phương. Ban đầu, các biến đệm được khởi tạo với giá trị 0.

+ Bước 3: Lặp lại các epochs , lặp qua từng mini-batch để tính toán gradient và cập nhật tham số theo công thức Adam.

+ Bước 4: tính toán gradient của hàm mất mát theo tham số hiện tại bằng cách sử dụng phương pháp lan truyền ngược (backpropagation) và các kỹ thuật tự động tính gradient.

+ Bước 5: Adam tính toán và cập nhật các biến đệm để lưu trữ lịch sử của gradient và gradient bình phương.

- Tính toán gradient bình phương hiện tại:
  v = beta2 * v + (1 - beta2) * (gradient ** 2)
- Tính toán gradient trung bình hiện tại:
  m = beta1 * m + (1 - beta1) * gradient

Trong đó,

- v và m lần lượt là các biến đệm lưu trữ lịch sử của gradient bình phương và gradient.

+ Bước 6: Điều chỉnh biến đệm bằng cách sử dụng các dạng điều chỉnh bias correction để giảm thiểu ảnh hưởng của giá trị ban đầu:

- Điều chỉnh biến đệm v:
  v_corrected = v / (1 - beta2^t)
- Điều chỉnh biến đệm m:
  m_corrected = m / (1 - beta1^t)

  Trong đó,
- t là số lần cập nhật tham số trong quá trình huấn luyện.

+ Bước 7: Sử dụng các biến đệm đã điều chỉnh, cập nhật tham số mô hình theo công thức Adam:

tham_số_mô_hình_mới = tham_số_mô_hình_cũ - learning_rate * m_corrected / (sqrt(v_corrected) + epsilon)

Trong đó,
- epsilon là một giá trị rất nhỏ (thường là 1e-8) được thêm vào mẫu để tránh chia cho 0.

❖ Bảng so sánh các phương pháp tối ưu hóa (optimizer)

Phương pháp tối ưu hóa	**Ưu điểm**	**Nhược điểm**
Gradient Descent (GD)	- Dễ hiểu và triển khai. - Đơn giản và không yêu cầu nhiều siêu tham số.	- Chậm trong việc hội tụ đến điểm cực tiểu địa phương. - Không hiệu quả khi có nhiễu trong dữ liệu.
Stochastic Gradient Descent (SGD)	- Nhanh hơn GD vì chỉ sử dụng một mẫu dữ liệu trong mỗi lần cập nhật. - Hướng tới điểm cực tiểu địa phương tốt hơn GD. - Phù hợp khi dữ liệu lớn.	- Không ổn định và dao động nhiều hơn GD. - Cần định nghĩa tỷ lệ học (learning rate) thích hợp.

Momentum	- Giúp tăng tốc độ hội tụ bằng cách tích lũy đạo hàm trước đó. - Giảm độ dao động và giúp thoát khỏi các cực tiểu địa phương. - Hiệu quả khi có nhiễu trong dữ liệu.	- Cần chọn giá trị cho hệ số momentum. - Có thể bị mắc kẹt ở điểm cực tiểu địa phương.
RMSProp	- Giảm ảnh hưởng của squared gradient bằng cách giới hạn lịch sử gradient. - Hiệu quả khi mô hình có các tham số thưa (sparse).	- Cần chọn giá trị cho hệ số decay rate. - Không phù hợp cho các bài toán với dữ liệu phi tuyến.
Adam	- Kết hợp ưu điểm của Momentum và RMSProp. - Hiệu quả trong nhiều bài toán và dữ liệu. - Tự điều chỉnh tỷ lệ học.	- Cần chọn giá trị cho các siêu tham số (learning rate, beta1, beta2). - Đôi khi có thể bị mắc kẹt ở điểm cực tiểu địa phương.

❖ Bảng so sánh độ tin cây của các phương pháp Optimiszer

Phương pháp tối ưu hóa	**Tốc độ hội tụ**	**Tránh điểm cực tiểu cục bộ**	**Độ ổn định**
Gradient Descent (GD)	Trung bình	Không	Trung bình
Stochastic Gradient Descent (SGD)	Trung bình	Không	Không

Momentum	Nhanh	Có	Tốt
RMSProp	Trung bình - Nhanh	Trung bình	Tốt
Adam	Nhanh	Có	Tốt

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.

❖ **Continual Learning (CL)**

Continual Learning là một lĩnh vực nghiên cứu quan trọng trong machine learning mà mô hình học máy được thiết kế để liên tục học từ các tập dữ liệu mới mà nó gặp phải, mà không làm mất đi khả năng thích ứng với dữ liệu đã học được trước đó.

Continual Learning là giúp mô hình duy trì và mở rộng kiến thức của mình theo thời gian, ngăn chặn hiện tượng quên mô hình (catastrophic forgetting) khi học từ các nguồn dữ liệu mới.

CL có nhiều ứng dụng tiềm năng trong các lĩnh vực khác nhau, chẳng hạn như:

- Robot học các kỹ năng mới liên tục trong môi trường thay đổi.
- Phân tích tài chính thích ứng với thị trường biến động.
- Chăm sóc sức khỏe cá nhân hóa bằng cách cập nhật kiến thức về bệnh nhân theo thời gian.
- Xử lý ngôn ngữ tự nhiên hiểu được các bối cảnh mới và các từ vựng mới.
- Những thách thức của CL

Các approaches của CL:

- Regularization techniques: Các kỹ thuật như weight decay và synaptic pruning giúp ngăn chặn các tham số cũ bị cập nhật quá nhiều khi học các nhiệm vụ mới.
- Memory replay: Lưu trữ và lặp lại một tập con dữ liệu cũ trong khi học các nhiệm vụ mới giúp củng cố kiến thức cũ.
- Parameter sharing and distillation: Chia sẻ một số tham số giữa các nhiệm vụ hoặc chưng cất kiến thức cũ thành các mô hình nhỏ hơn, hiệu quả hơn có thể giúp tiết kiệm bộ nhớ và cải thiện việc học.
- Meta-learning: Học cách học hiệu quả hơn từ kinh nghiệm trước có thể giúp các mô hình CL thích ứng nhanh hơn với các nhiệm vụ mới.

- **Thách Thức Chính Catastrophic Forgetting**
    - Catastrophic Forgetting (Hiện tượng quên mô hình) là một vấn đề lớn trong Continual Learning. Khi một mô hình được đào tạo với dữ liệu mới, nó có thể quên đi tri thức đã học từ dữ liệu cũ, dẫn đến mất mát đáng kể trong hiệu suất trên nhiệm vụ trước đó.

- **Phương Pháp Regularization:** Elastic Weight Consolidation (EWC): EWC là một phương pháp regularization được thiết kế để bảo vệ trọng số quan trọng của mô hình từ việc thay đổi quá mức khi học từ dữ liệu mới. Nó đặt trọng số lớn cho các trọng số quan trọng và giảm độ nhạy của chúng đối với điều chỉnh.
- **Bộ Nhớ Ngoại Vi (External Memory):** Sử dụng bộ nhớ ngoại vi là một chiến lược phổ biến để giải quyết hiện tượng quên. Mô hình có thể lưu trữ thông tin quan trọng từ dữ liệu cũ trong bộ nhớ để sử dụng lại khi cần thiết.
- **Biểu Diễn Đa Nhiệm:** Continual Learning thường kết hợp với biểu diễn đa nhiệm, trong đó mô hình cố gắng học đồng thời từ nhiều tập dữ liệu và giữ lại khả năng chia sẻ thông tin giữa các nhiệm vụ.
- **Task-Incremental và Domain-Incremental Learning:** Trong task-incremental learning, mỗi nhiệm vụ mới được giới thiệu đồng thời với dữ liệu mới. Trong domain-incremental learning, mô hình phải xử lý các dữ liệu từ các miền mới, có thể khác biệt đáng kể so với các miền đã biết.
- **Online và Offline Learning:** Continual Learning có thể triển khai online (học liên tục khi có dữ liệu mới) hoặc offline (đào tạo lại toàn bộ mô hình khi có dữ liệu mới).
- **Thách Thức Đa Nhiệm:** Đối mặt với thách thức của việc học đa nhiệm, nơi mà mô hình cần hiệu quả hóa trên nhiều nhiệm vụ đồng thời.
- **Tiếp Cận Meta-Learning:** Meta-learning cũng được kết hợp trong Continual Learning để giúp mô hình nhanh chóng thích ứng với dữ liệu mới.

Chuẩn Bị Dữ Liệu Kiểm Thử: Trước khi triển khai giải pháp học máy vào môi trường sản xuất, cần chuẩn bị một tập dữ liệu kiểm thử để representative và đa dạng để đảm bảo rằng giải pháp hoạt động tốt trên nhiều trường hợp. Để đảm bảo về giải pháp hoạt động tốt thì công cụ test production cần phải thực hiện các kiểm thử như sau:

- **Test Production** là quá trình tạo ra các bài kiểm thử để đánh giá hiệu suất của mô hình học máy đã được đào tạo như Kiểm Thử Tính Tổng Quát Hóa, Kiểm Thử Hiệu Suất, Kiểm Thử Độ Tin Cậy và Khả Năng Xử Lý Ngoại Lệ, Kiểm Thử Tích Hợp, Kiểm Thử Bảo Mật, Kiểm Thử Môi Trường

Test Production giúp đảm bảo rằng các mô hình học máy hoạt động tốt trong môi trường sản xuất. Điều này là cần thiết để tránh các lỗi hoặc sự cố có thể gây ra thiệt hại.

Quy trình này bao gồm các bước sau:

- Tạo một môi trường thử nghiệm (test environment) giống với môi trường sản xuất (production environment) càng nhiều càng tốt.
- Huấn luyện mô hình trên tập dữ liệu thử nghiệm (test data).
- Tiến hành kiểm tra mô hình (model testing) bằng cách sử dụng tập dữ liệu thử nghiệm.
- Triển khai mô hình vào môi trường sản xuất (production deployment).

Các kỹ thuật Test Production:

- Data sampling: Chọn mẫu dữ liệu từ tập dữ liệu sản xuất để sử dụng cho quá trình huấn luyện và kiểm tra mô hình.
- Data augmentation: Tạo dữ liệu mới từ dữ liệu sản xuất bằng cách sử dụng các kỹ thuật như xoay, lật, thay đổi cường độ ánh sáng,...
- Cross-validation: Phân chia tập dữ liệu sản xuất thành các tập con nhỏ hơn và sử dụng các tập con này để huấn luyện và kiểm tra mô hình.