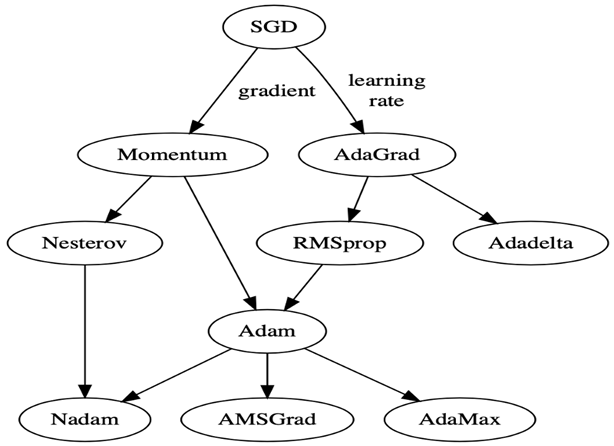
Trình bày một bài nghiên cứu, đánh giá của em về các vấn đề sau:

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.

Optimizer



Nguồn: <https://insideaiml.com/blog/Optimizers-in-Machine-Learning-and-Deep-Learning.-1048>

Trong machine learning hay cụ thể hơn là deep learning, nhiệm vụ của các optimizer hay optimizer algorithms (các thuật toán tối ưu) là những thuật toán giúp điều chỉnh các parameters (weights, biases, learning rates, ...) của model trong suốt quá trình training để làm sao giải thiểu được loss function. Những optimizers thường lặp đi lặp lại việc cập nhật các weights và biases.

Các optimizers có mục tiêu chung là phải giúp cho việc training càng nhanh càng tốt và cho ra các parameters sao cho dự đoán kết quả chính xác.

Một trong những vấn đề trong việc lựa chọn các optimizers là làm thế nào để có thể chọn đúng optimizer cho vấn đề hay ứng dụng cho việc gì đó? Vì trong thực tế, bộ dữ liệu có thể chứa hàng triệu features, hàng triệu instances, chỉ với một lần lặp để cập nhật các parameters cũng đã tốn một khoảng thời gian đáng kể.

Vì để tránh việc các áp dụng các optimizers không hiệu quả, sau đây tôi xin trình bày về những hiểu biết về các optimizers mà tôi tìm hiểu được, như Gradient Descent, Stochastic Gradient Descent, Stochastic Gradient descent with momentum, Mini-Batch Gradient Descent, Adagrad, RMSProp, AdaDelta, và Adam. Và tiếp theo đó là những so sánh các optimizers nhằm phần nào lựu chọn được optimizer phù hợp cho vấn đề của mình.

Những thành phần chung và quan trọng đối với một optimizer:

* Epoch: Số lần lặp mà thuật toán chạy với toàn bộ training dataset.
* Sample: Từng hàng trong một dataset.
* Batch: Biểu thị số lượng samples được lấy cho mỗi lần cập nhật các parameters của model.
* Learning rate: một parameter cung cấp cho model một scale (có thể hiểu là bước nhảy) dành cho việc cập nhật weights và biases.
* Cost Function/ Loss Function: hàm thể hiện độ sai lệch giữa giá trị dự đoán và giá trị thật sự.
* Weights và biases: các parameters của model, các parameters này sẽ liên tục được cập nhật, để cuối cùng có thể giúp model dự đoán chính xác.

Gradient Descent

Gradient Descent hay Batch Gradient Descent là một trong những optimizer cơ bản nhất. Nó được áp dụng cho cả hai vấn đề là regression và classification. Ngoài ra, cả Backpropagation trong neural network có sử dụng Gradient Descent.

Gradient Descent chủ yếu là đạo hàm từng phần các parameters gồm weights và bias trong loss function và sau đó cập nhật các parameters này sao cho loss function có giá trị nhỏ nhất (đạo hàm từng phần của loss function tiến tới giá trị là 0).

Trong đó:

|  |  |
| --- | --- |
|  | : tâp hợp các parameters (weights và bias) |
|  | : loss function |
|  | : đạo hàm của loss function tại một bất kì |
|  | : learning rate, gia tốc cho việc tìm global minimum |

Ví dụ loss function trong linear regression:

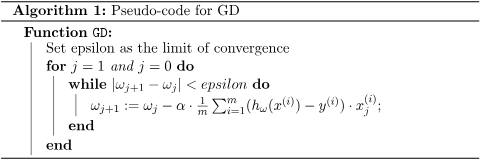
Trong đó:

|  |  |
| --- | --- |
|  | : điểm dữ liệu thứ , ví dụ   là số lượng features |
|  | : target thứ |
|  | : tổng số lượng điểm dữ liệu |

Thuật toán sẽ ngừng lại khi:

* Loss function có giá trị xấp xĩ bằng là 0 (hoặc đủ nhỏ khi so sánh với 2 hoặc 3 nghiệm trước đó).
* Đã hết số lần lặp (epoch) đã cho (dành trong việc cài đặt thuật toán).

Các biến thể của Gradient Descent được đề cập ở các phần sau cũng dựa vào các điều kiện trên để dừng thuật toán.



Pseudocode cho GD

Nguồn: <https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch>

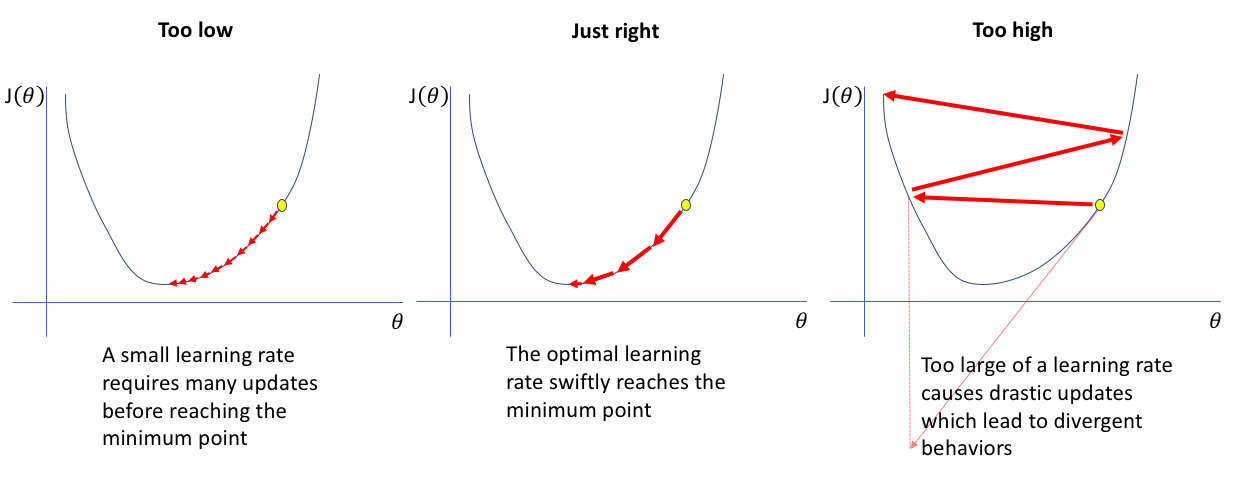
Ưu điểm:

Thuật toán Gradient Descent cơ bản dễ hiểu.

Nhược điểm:

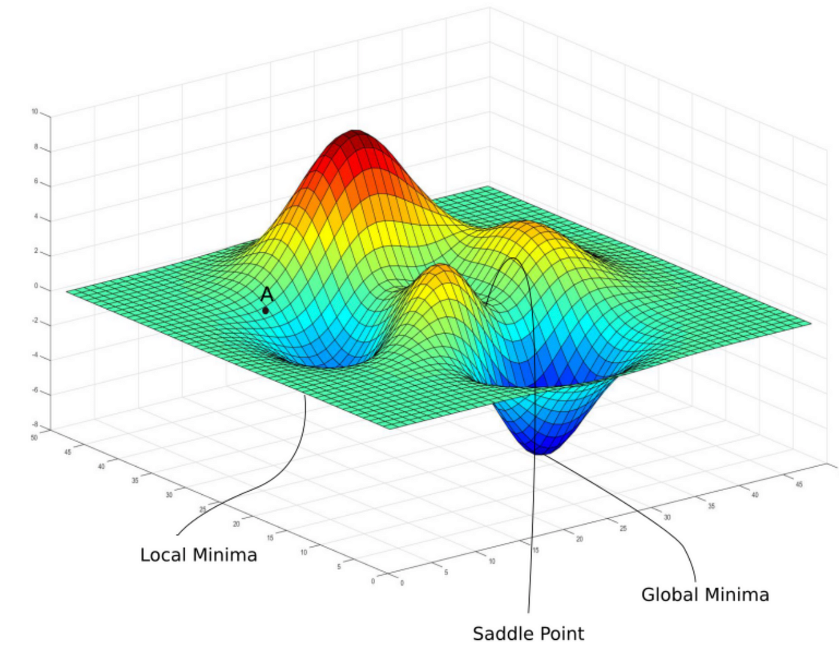
Cần một lượng lớn bộ nhớ để lưu trữ dataset.

Gradient Descent còn nhiều hạn chế như phụ thuộc vào khởi tạo ban đầu, learning rate không phù hợp như quá lớn chẳng hạn sẽ làm cho parameter dao động xung quanh hoặc tán xạ ra khỏi parameter tối ưu nhất.



Nguồn: <https://www.jeremyjordan.me/nn-learning-rate/>

Việc loss function có nhiều điểm local minimum cũng ảnh hưởng đến việc tối ưu các parameters.



Nguồn: <https://medium.com/analytics-vidhya/journey-of-gradient-descent-from-local-to-global-c851eba3d367>

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) một biến thể của Gradient Descent (GD), nó khác GD ở chỗ, với GD mỗi epoch ta sử dụng toàn bộ dữ liệu để cập nhật duy nhất 1 lần cho các , còn với SGD ta sử dụng từng điểm dữ liệu trong toàn bộ dữ liệu để cập nhật cho các , tức là mỗi epoch ta cập nhật lần các .

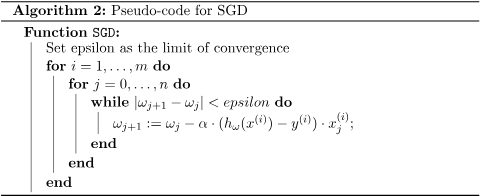
Việc cập nhật các làm giảm đi tốc độ thực hiện epoch, tuy nhiên nó chỉ cần một lượng epoch rất nhỏ để tối ưu . Vì thế nó phù hợp cho các bài toán sử dụng cơ sở dữ liệu lớn.

Với mỗi lần thực hiện xong 1 epoch ta cần shuffle (xáo trộn) thứ tự dữ liệu để đảm bảo tính ngẫu nhiên và hiệu năng của SGD.

Quy tắc cập nhật của SGD:

Trong đó:

|  |  |
| --- | --- |
|  | : loss function với cặp dữ liệu |

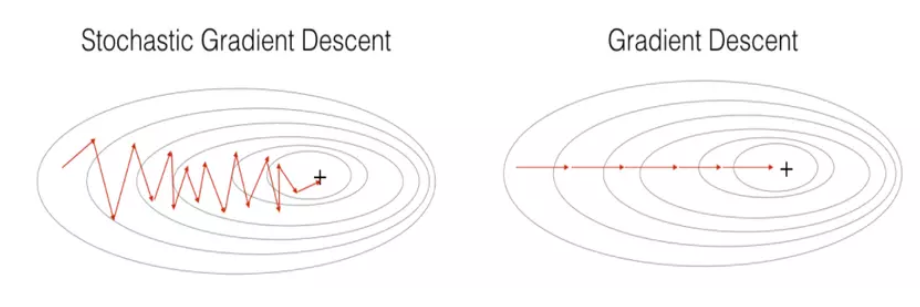


Pseudocode cho SGD

Nguồn: <https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch>

Ưu điểm:

SGD hội tụ rất nhanh, hình dưới đây minh họa về việc cập nhật của GD và SGD, ta thấy rằng đường đi của SGD là zigzag, chính đường zigzag này đã giúp cho SGD ít bị rơi vào local minimum mà chỉ fluctuate ở các điểm đó, tuy nhiên vẫn không loại trừ trường hợp SGD có thể không hướng về global minimum và có thể SGD cũng gây ra high variance. Trong hình 2 trục tung và hoành thể hiện cho model có 2 parameters (có thể là 1 weight và 1 bias).



Nguồn: <https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/>

SGD phù hợp cho dữ liệu được cập nhật liên tục (online learning), vì với nó chỉ cần cập nhật theo điểm dữ liệu mới có, chứ không phải toàn bộ như GD.

Nhược điểm:

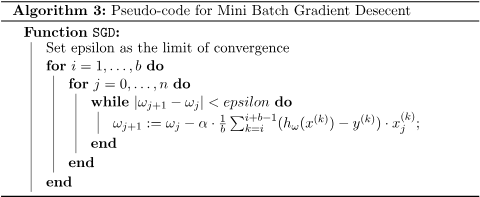
Tuy giải quyết được vấn đề cơ sở dữ liệu lớn, SGD vẫn không tránh được các hạn chế như phụ thuộc vào khởi tạo ban đầu, learning rate của GD.

Mini Batch Gradient Descent

Mini Batch Gradient Descent (MBGD) là một sự trộn lẫn giữa GD và SGD, khắc phục phần tính toán lâu của SGD khi lấy từng sample ngẫu nhiên, MBGD sử dụng chia toàn bộ dataset thành các mini-batch có samples với mỗi epoch (mini-batch cuối có thể có số lượng samples bé hơn ), với , việc lấy nhiều samples đã giúp MBGD có thể vector hóa như GD.

Trong đó:

|  |  |
| --- | --- |
|  | : tập hợp các cặp input, output từ đến |



Pseudocode cho MBGD

Nguồn: <https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch>

Ưu điểm:

Không cần một lượng lớn bộ nhớ để lưu trữ khi tính toán.

Hội tụ nhanh.

Nhược điểm

Vẫn không giải quyết được vấn đề khi chọn learning rate .

Gradient Descent with Momentun

Gradient Descent with Momentun là cách nói chung về việc áp dụng Gradient Descent nguyên bản hoặc các biến thể của nó như Stochastic Gradient Descent với Momentun.

Dưới góc nhìn của vật lý, Momentun đi cùng với Gradient Descent, đơn giản là một đại lượng mang thông tin là độ dốc và đà (vận tốc trước đó, coi ) , thuật toán này đơn giản là cộng thêm đại lượng này vào công thức cập nhật parameters:

Trong đó:

|  |  |
| --- | --- |
|  | : thường được lấy là |
|  | : vận tốc tại thời điểm hiện tai |

Ưu điểm:

Bổ trợ cho Gradient Descent vượt local minimum.

Nhược điểm:

Khi tới điểm tối ưu cần một khoảng thời gian để hội tụ, vì đà vẫn còn.

Vẫn còn hạn chế về learning rate.

Apdative Gradient Descent

Apdative Gradient Descent (Adagrad) giúp giải quyết vấn đề về việc chọn hằng số learning rate.

Trong đó:

|  |  |
| --- | --- |
|  | : vector chứa đạo hàm (hoặc là gradient) các parameters tại thời điểm |
|  | : vector chứa tích trữ gradient trước đó, |
|  | : hằng số thêm vào để tránh việc chia cho , |

Ưu điểm:

Tự điều chỉnh được learning rate.

Nhược điểm:

Learnig rate nhỏ dần vì thế có thể làm chậm quá trình training.

Root Mean Squared Propagation

Root Mean Squared Propagation (RMSProp) là một phiên bản đặc biệt của Adagrad, với Adagrad ngày càng tăng chính vì thế làm cho learning rate ngày càng nhỏ, có một cách giải quyết là lấy , lúc này sẽ hội tụ nhưng nó cũng mất khá nhiều thời gian, chính vì thế RMSProp cập nhật như sau:

Trong đó:

|  |  |
| --- | --- |
|  | : tham số tự chọn, |

Ưu điểm:

Giải quyết được vấn đề learning rate giảm dần.

Nhược điểm:

Có thể cho nghiệm là local minimum.

Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) là sự kết hợp giữa RMSProp và Momentun nhằm cải thiện việc RMSProp rơi vào local minimum.

Ưu điểm:

Có những ưu điểm của Momentun, RMSProp

Nhược điểm:

Chỉ là về việc cần bộ nhớ để lưu trữ tính toán có thể xem dường như không có

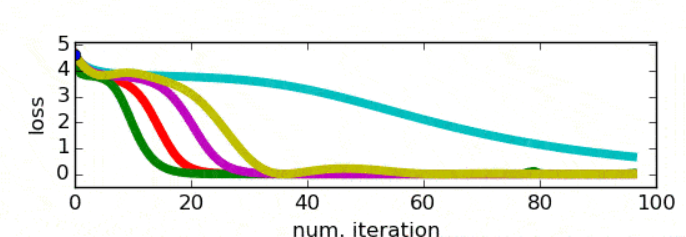
Dưới đây là bảng so sánh các optimizers

|  |  |  |
| --- | --- | --- |
| Tên các opimizers | Ưu điểm | Khuyết điểm |
| Gadient Descent (GD) | Thuật toán đơn giản.  Phù hợp với dữ liệu nhỏ. | Cần bộ nhớ lớn.  Khả năng cao rơi vào local minimum.  Phụ thuộc vào learning rate, các parameters khởi tạo. |
| Stochastic Gradient Descent (SGD) | Không cần bộ nhớ lớn.  Nhìn chung hội tụ nhanh. | Phụ thuộc vào learning rate, các parameters khởi tạo.  Có thể bị high variance.  Thời gian cho mỗi epoch khá lâu. |
| Mini Batch Gradient Descent (MBSGD) | Không cần bộ nhớ lớn.  Nhìn chung hội tụ nhanh. | Phụ thuộc vào learning rate. |
| Gradient Descent with Momentun | Thoát khỏi local minimum. | Cần một khoảng thời gian để hội tụ tại global minimum. |
| Adaptive Gradient Descent  (Adagrad) | Tự điều chỉnh được learning rate. | Learning rate giảm dần làm chậm quá trình training. |
| Root Mean Squared Propagation  (RMSProp) | Giả quyết được vấn đề learning rate giảm dần. | Có thể rơi vào local minimum. |
| Adaptive Moment Estimation (Adam) | Tự điều chỉnh được learning rate giảm dần.  Thoát được local minimum.  Hội tụ nhanh. | Cần bộ nhớ để lưu trữ dữ liệu tính toán. |

A graph of a weather forecast

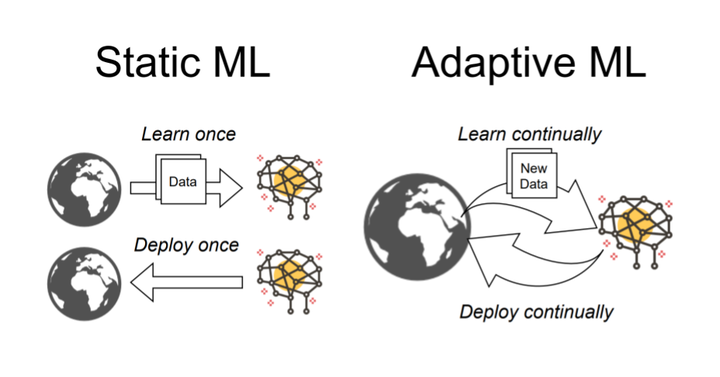
Description automatically generated with medium confidence

Nguồn: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>



Nguồn: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>

Continual Learning



Nguồn: <https://ai.kuleuven.be/stories/post/2021-05-10-continual-learning/>

Continual Learning hay còn được gọi với nhiều tên khác như Incremental Learning, Lifelong Learning, Never Ending Learning. Trong bài này tôi xin gõi chung là Continual Learning (CL).

Các hướng tiếp cận của Machine Learning thông thường là từ stream of data (dòng chảy dữ liệu) được lấy mẫu ngẫu nhiên từ một phân phối dữ liệu tĩnh (stationary data distribution). Điều này tạo ra sự hiệu quả trong việc học. Tuy nhiên, trong thực tế, việc này điều này thường không phổ biến. Continual Learning giải quyết vấn vấn đề về việc dữ liệu thay đổi liên tục và những kiến thức dung hợp (knowledge fusion).

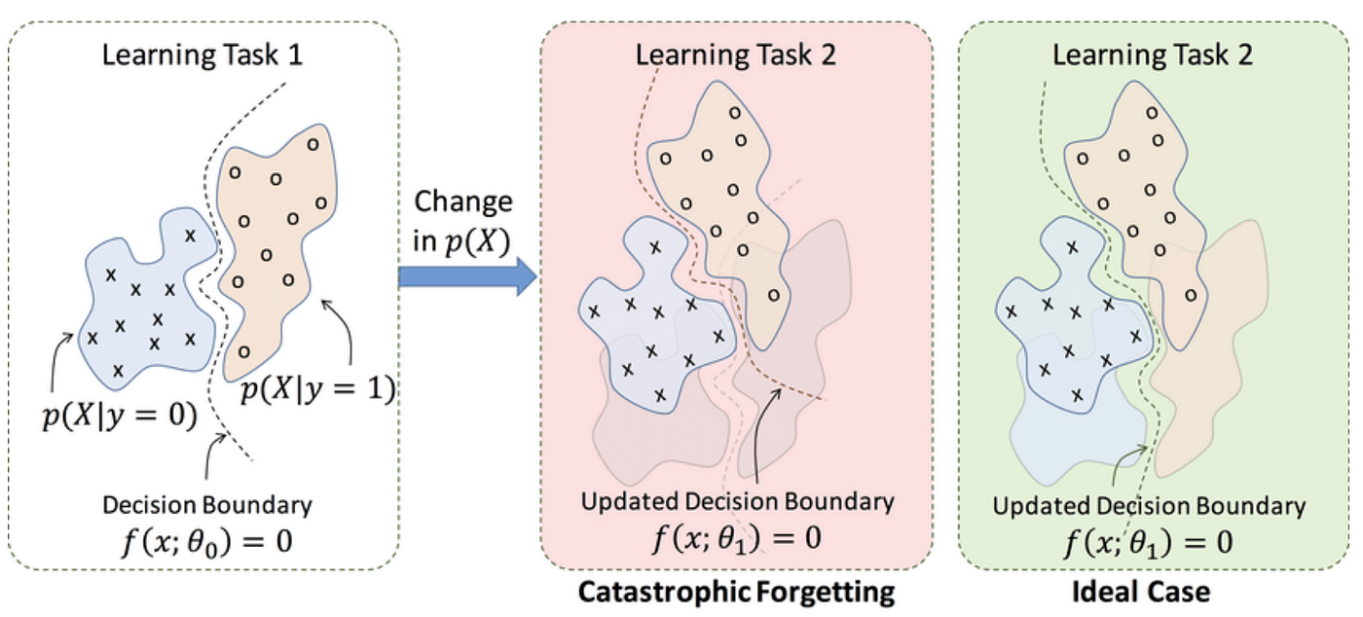
Cho một stream of data không giới hạn, một thuật toán Continual Learning sẽ học từng phần của dữ liệu đó (sequence of partial exprerinces) vì dữ liệu không thể luôn luôn có sẵn. Một non-continual learning (như Machine Learning model truyền thống chẳng hạn) thì luôn có thể truy cập toàn bộ dữ liệu một lần và có thể xử lý nếu muốn. Một Continual Learning thì đối mặt với vấn đền thiếu thốn dữ liệu (scarce data problems), quên nghiêm trọng (catastrophic forgetting), phân phối dữ liệu theo ca (data distribution shifts).

Với những ràng buộc bởi Online Learning, CL cần phải cụ thể hoặc trừu tượng cho việc knowledge fusion tại từng mức độ khác nhau. Đầu tiên, CL cần phải hỗ trợ cho data-level fusion và đồng thời phải bảo lưu các kiến thức đã học có khả năng sẽ quên. Cuối cùng, fusion cần phải ở mức độ là kiến thức (knowledge) hoặc khái niệm (conceptual) vì để tránh các dữ liệu thô và để không bị catastrophic forgetting.

Những thách thức cần được giải quyết bởi CL

Catastrophic Forgetting

Catastrophic Forgetting được là một hiện tượng mà neural network quên những kiến thức đã học trước đó trong khi đang học các kiến thức mới.



Nguồn: <https://towardsdatascience.com/forgetting-in-deep-learning-4672e8843a7f>

Hading Memories

Để giải quyết vấn đề catastrophic forgetting, mỗi stategy cần phải tìm ra cách để nhớ những gì (parameters) gradient descent có thể quên. CL cần có cơ chế để lưu trữ memories của các tasks trước đó, với memories có nhiều hình thái khác nhau. Điều quan trọng cần lưu ý là các memories cần có cách quản lý, tổ chức khác nhau, như: raw data, representations, model weights, regularization matrices, ....

Detecting Distribution Shifts

Khi mà phân phối dữ liệu không thuộc dạng cố định hay tĩnh, quá trình chuyển sang data stream cần phải được nhận biết sớm. CL model cần phải giải quyết được thách thức trên. Một cuộc chuyển data stream không được nhận biết kĩ càng sẽ làm liên đới đến việc “quên lãng”.