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

ĐỒ ÁN CUỐI KÌ MÔN
NHẬP MÔN HỌC MÁY

TÌM HIỂU MỘT SỐ PHƯƠNG PHÁP TRONG HỌC MÁY

Người hướng dẫn: **GV. Lê Anh Cường**

Người thực hiện: **NGUYỄN TẤN THÀNH – 52100841**

Lớp **: 21050201**

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 KÌ MÔN
NHẬP MÔN HỌC MÁY

TÌM HIỂU MỘT SỐ PHƯƠNG PHÁP TRONG HỌC MÁY

Người hướng dẫn: **GV. NGUYỄN CHÍ THIỆN**

Người thực hiện: **NGUYỄN TẤN THÀNH – 52100841**

Lớp **: 21050201**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, em xin trân trọng cảm ơn giảng viên người đã trực tiếp chỉ bảo, và truyền đạt kiến thức để hoàn thành bài báo cáo này.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi. 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 20 tháng 12 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Thành

Nguyễn Tấn Thành

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

Đây là báo cáo về phương pháp tối ưu optimizer trong Deep learning của chúng tôi. Được viết bằng Tiếng Việt.

Trong báo cáo, chúng tôi có trình bày chi tiết về các phương pháp tối ưu phổ biến hiện nay trong cộng đồng deep learning. Từ những lý thuyết và tư tưởng căn bản nhất của gradient descent, chúng tôi giúp người đọc hiểu được ưu và nhược điểm của từng phương pháp, đi sâu vào bản chất toán học của chúng.

Với gradient descent, phương pháp này là khởi nguồn cho mọi phương pháp sau này. Tuy có những nhược điểm, nhưng nó vẫn đang là phương pháp dễ hiểu và phổ biến nhất. Gradient khó vượt qua điểm cực tiểu và nó không tuỳ chỉnh learning rate cho từng tham số, có những tham số cần cập nhật timestep lớn sau mỗi bước, có những tham số lại chỉ cần timestep nhỏ sau mỗi bước. Vì vậy, 1 loạt các phương pháp tối ưu sau này ra đời nhằm khắc phục các nhược điểm của Gradient Descent như: momentum, NAG, AdaGrad, Adadelta, RMSProp và Adam.

Continual Learning (CL) là một lĩnh vực quan trọng trong Machine Learning (ML) mà mô hình cần có khả năng liên tục học từ dữ liệu mới mà không quên đi kiến thức đã học từ dữ liệu cũ. Trong quá trình triển khai giải pháp học máy để giải quyết một bài toán, việc đối mặt với dữ liệu mới và thay đổi liên tục là điều khá phổ biến.

Test Production là quá trình tạo và triển khai các tập kiểm thử để đánh giá hiệu suất của mô hình.

Tôi cũng thực nghiệm code lại các thí nghiệm và public code của các bài báo cho bạn đọc tham khảo. Các bạn có thể xem báo cáo ở file báo cáo pdf và xem code ở các file jupyter.

Mục Lục

CHƯƠNG 1 – CÁC PHƯƠNG PHÁP OPTIMIZER

online learning là khi dữ liệu cập nhật liên tục (ví dụ như thêm người dùng đăng kí) thì mỗi lần thêm dữ liệu ta phải tính lại đạo hàm trên toàn bộ dữ liệu => thời gian tính toán lâu, thuật toán không online nữa.

1.1 Optimizer là gì?

- Một optimizer có nhiệm vụ điều chỉnh các trọng số của mô hình để giảm thiểu hoặc tối đa hóa một hàm chi phí (cost function) nào đó.

- Trước khi bắt đầu khám phá sâu vào quá trình tối ưu hóa trong mô hình neural network, chúng ta cần hiểu rõ về thuật toán tối ưu (optimizer) và vai trò quan trọng của nó. Mục tiêu của thuật toán tối ưu là điều chỉnh các tham số (weights và bias) của mô hình để mô hình có thể "học" và tự điều chỉnh dự đoán theo dữ liệu đầu vào.
- Tính "học" này không đơn giản là việc đặt ngẫu nhiên các giá trị cho weights và bias một cách hữu hạn và hi vọng rằng mô hình sẽ tìm ra lời giải tối ưu. Điều này không khả thi và làm lãng phí tài nguyên. Thay vào đó, chúng ta cần một phương pháp thông minh để cải thiện từng bước và điều chỉnh weights và bias một cách hiệu quả.
- Thuật toán optimizer chính là người hướng dẫn mô hình qua quá trình này. Thay vì sử dụng phương pháp ngẫu nhiên, optimizer giúp mô hình di chuyển theo hướng tối ưu trên bề mặt hàm chi phí. Điều này thường được thực hiện bằng cách tính gradient của hàm chi phí đối với các tham số và điều chỉnh chúng dựa trên hướng và độ lớn của gradient đó.

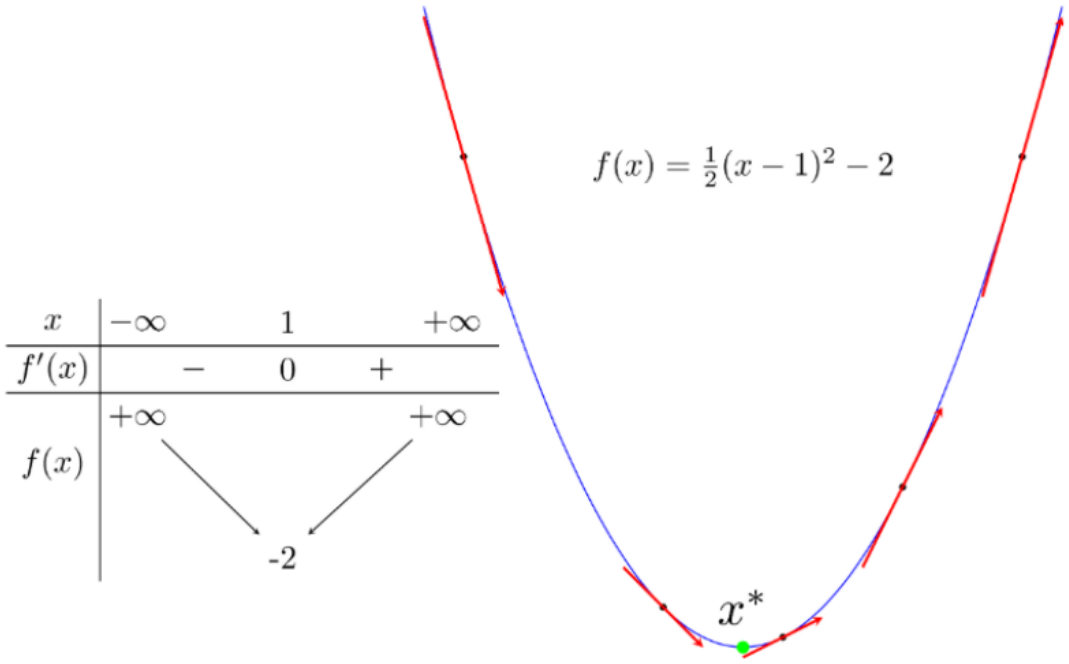
- Tối ưu hóa được sử dụng trong học máy vì các mô hình thường có nhiều thâm số việc tìm ra giá trị tốt nhất cho các tham số đó có thể tự động tìm kiếm các thông số tốt thay vì thủ công
- Từ đó, chúng ta có các thuật toán tối ưu như Stochastic Gradient Descent (SGD), Adam, RMSprop, v.v. Các thuật toán này không chỉ giúp tối ưu hóa mô hình mà còn đối mặt với các vấn đề như tốc độ học, độ nhanh chóng của hội tụ, và tránh các đặc điểm bất lợi như local minima. Nhờ vào sự phát triển của các thuật toán optimizer, quá trình học của mô hình neural network trở nên mạnh mẽ và hiệu quả hơn.

1.2 Gradient Descent

- Trong Machine learning nói riêng và Toán tối ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó. Bài toán này có tầm quan trọng trong nhiều lĩnh vực khoa học và kỹ thuật. Ta có thể tìm các điểm cực tiểu hay cực đại của hàm số bằng cách giải phương trình đạo hàm bằng 0. Tuy nhiên, trong đa số các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc có quá nhiều điểm dữ liệu.
- Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép lặp nào đó để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0.
- Gradient descent (viết tắt là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất. Mọi thư viện deep learning hiện nay đều triển khai các thuật toán đa dạng khác nhau dựa trên GD để tối ưu.

1.2.1 Ý tưởng thuật toán

- Xét bài toán với hàm 1 biến, xét hàm số f(x) = $\frac{1}{2}(\boldsymbol{x-1})^2-\mathbf{2}$ có bảng biến thiên và đồ thị hàm số như hình 1.1. Điểm x* là điểm cực điểm của hàm số. Tại đó f ‘ (x*) = 0. Đường tiếp tuyến với đồ thị hàm số đó tại một điểm bất kỳ có hệ số góc chính bằng đạo hàm của hàm số tại điểm đó. Trong hình 1.1, các điểm bên trái của x* có đạo hàm âm, các điểm bên phải có đạo hàm dương. Và đối với hàm số này, càng xa về phía trái của x* thì đạo hàm càng âm, càng xa về phía bên phải thì đạo hàm càng dương.
- Giả sử $\mathbf{x_t}$ là điểm ta tìm được sau vòng lặp thứ t. Ta cần tìm một thuật toán để đưa $\mathbf{x_t}$ về càng gần x* càng tốt.



Hình 1.1: Minh hoạ hàm số $f(x) = \frac{1}{2}(x-1)^2 - 2$

Từ hình vẽ, chúng ta có quan sát sau:

- Nếu đạo hàm của hàm số tại $\mathbf{x_t}$: f ‘ ($\mathbf{x_t}$) > 0 thì xt nằm về phía bên phải so với x* (và ngược lại). Để điểm tiếp theo $\mathbf{x_{t+1}}$ gần với x* hơn, chúng ta cần di

chuyển $\mathbf{x_t}$ về phía bên trái, tức là về phía âm. Hay nói cách khác, chúng ta cần di chuyển ngược dấu với đạo hàm: $\mathbf{x_{t+1} = x_t + \nabla}$

Trong đó **∇ là một đại lượng ngược dấu với đạo hàm** f ' ($\mathbf{x_t}$)

- $\mathbf{x_t}$ càng xa x* về phía bên phải thì f'($\mathbf{x_t}$) càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển Δ, một cách trực quan, là tỉ lệ thuận với −f '($\mathbf{x_t}$).
- Từ nhận xét phía trên, chúng ta có một cập nhật đơn giản là:

$$x_{t+1} = x_t - \eta f'(x_t)$$

  - Trong đó η là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc chúng ta phải đi ngược dấu đạo hàm (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - đi ngược đạo hàm).

1.2.2 Optimizer với Gradient descent

Gradient Descent là một quá trình tối ưu hóa phức tạp và phụ thuộc vào nhiều yếu tố quan trọng. Đầu tiên, sự lựa chọn của điểm khởi đầu có thể ảnh hưởng đến quá trình hội tụ. Một khởi đầu khác nhau có thể dẫn đến kết quả khác nhau.

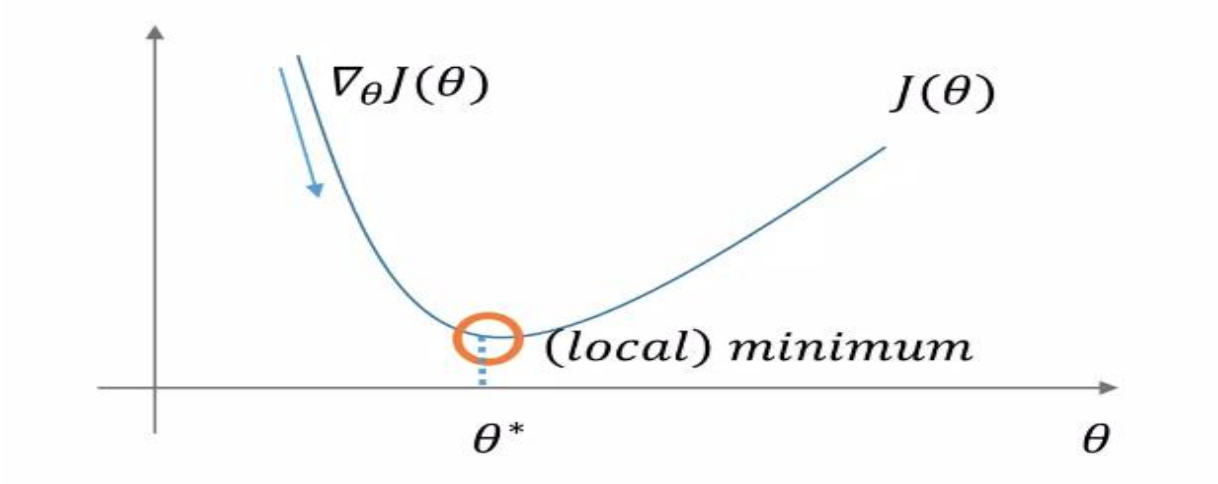
Thứ hai, tốc độ học (learning rate) đóng vai trò quan trọng. Nếu learning rate quá nhỏ, quá trình hội tụ sẽ rất chậm, trong khi nếu quá lớn, có thể dẫn đến việc không hội tụ và dao động xung quanh mục tiêu với bước nhảy quá lớn.

Tóm lại, sự cân nhắc cẩn thận về điểm khởi đầu và lựa chọn learning rate là quan trọng để đảm bảo Gradient Descent hội tụ một cách hiệu quả và đạt được kết quả tối ưu trong quá trình đào tạo mô hình.

- Gradient descent là một cách để cực tiểu hóa hàm mục tiêu J(θ)
  - $\theta \in R^d$: tham số của mô hình

- $n : learning\ rate$
- ∇_θ J(θ) : gradient của hàm mục tiêu theo các tham số

- Cập nhật tham số theo hướng ngược lại với gradient
- Cập nhật phương trình : $\theta = \theta - n.\nabla_\theta$ J(θ)



Optimization with gradient descent

1.2.3 Batch Gradient Descent

Batch Gradient Descent tức là tính toán gradient cho hàm mục tiêu với tham số θ cho toàn bộ tập dữ liệu.Ở phần trên, chúng ta đã xét hàm một biến. Tiếp theo chúng ta sẽ xét tổng quát với hàm nhiều biến.

Giả sử ta cần tìm global minimum cho hàm f(θ) trong đó θ là một vector, thường được dùng để ký hiệu tập hợp các tham số của mô hình cần tối ưu. Đạo hàm của hàm số đó tại một điểm θ bất kỳ được ký hiệu là $\boldsymbol{\nabla_\theta}$ f($\boldsymbol{\theta}$). Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán $\boldsymbol{\theta_0}$, sau đó, tại vòng lặp thứ t, quy tắc cập nhật là:

$\boldsymbol{\theta_t} = \boldsymbol{\theta_{t-1}} - \boldsymbol{n}.\boldsymbol{\nabla_\theta}$ f($\boldsymbol{\theta_t}$)

```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(
    loss_function, data, params)
  params = params - learning_rate * params_grad
```

Code for Batch gradient descent update

- Ưu điểm :
    - Dảm bảo hội tụ đến mức global minimun với các bề mặt lỗi lồi và local minimun với các bề mặt không lồi
- Nhược điểm :
    - Rất chậm
    - Khó sửa đổi với dữ liệu không fit với bộ nhớ
    - Không online learning
    - Hạn chế của Batch GD là khi tập dữ liệu lớn, việc tính gradient sẽ tốn nhiều thời gian và chi phí tính toán.

Với batch gradient descent, chúng ta cần tính toán gradient cho toàn bộ dataset để biểu diễn một cập nhật, chính vì thế batch gradient descent có thể rất chậm và khó khăn để tính toán cho toàn bộ dataset vì không thể fit hết data vào bộ nhớ. Batch gradient descent cũng không thế cập nhật dưới dạng online learning, tức là cập nhật khi có một mẫu dữ liệu mới đưa vào thì thuật toán phải cập nhật. Nếu làm theo Batch Gradient Descent, tức là tính lại đạo hàm của hàm mục tiêu tại tất cả các điểm dữ liệu, thì thời gian tính toán sẽ rất lâu, và thuật toán của chúng ta coi như là không online nữa do mất quá nhiều thời gian tính toán.

Trên thực tế, có một thuật toán đơn giản hơn và tỏ ra hiệu quả hơn, có tên gọi là Stochastic Gradient Descent (SGD).

1.2.4 Stochastic Gradient Descent

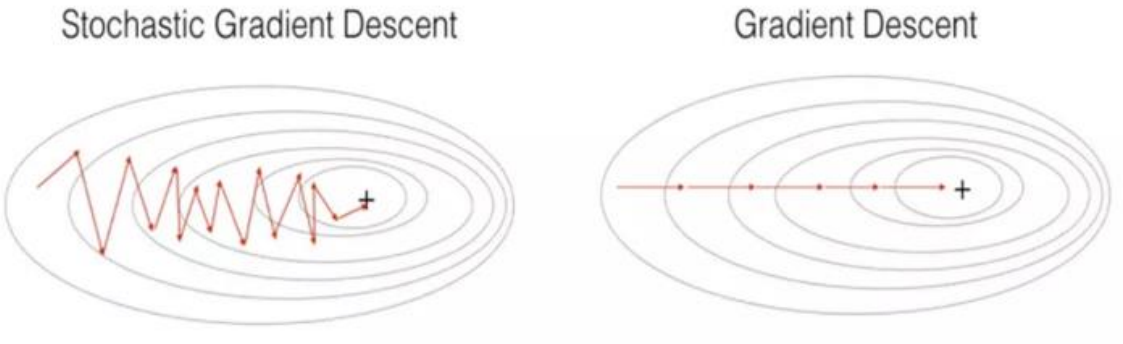
- Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mục tiêu dựa trên chỉ một điểm dữ liệu xi rồi cập nhật θ dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán đơn giản này trên thực tế làm việc rất hiệu quả.
- Tính toán trên mỗi điểm dữ liệu $x^{(i)}y^{(i)}$
- Cập nhật phương trình : $\theta = \theta - n.\nabla_\theta \, J(\theta; x^{(i)}; y^{(i)})$

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(
      loss_function, example, params)
    params = params - learning_rate * params_grad
```

Code for stochastic gradient descent update

- Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với Gradient Descent thông thường thì mỗi epoch ứng với 1 lần cập nhật θ, với SGD thì mỗi epoch ứng với N lần cập nhật θ với N là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu mọt lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). Vì vậy, SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn và các bài toán yêu cầu mô hình thay đổi liên tục, tức là online learning.
- Một điểm cần lưu ý là sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các điểm dữ liệu để đảm bảo tính ngẫu nhiên. Việc này ảnh hưởng tới hiệu năng của SGD.

- Trong Giảm dần độ dốc ngẫu nhiên, bạn chỉ sử dụng 1 ví dụ đào tạo trước khi cập nhật độ dốc. Khi tập huấn luyện lớn, SGD có thể nhanh hơn. Nhưng các thông số sẽ “dao động” về phía tối thiểu chứ không hội tụ một cách suôn sẻ. Đây là những gì trông giống như:



("+" biểu thị chi phí tối thiểu. SGD dẫn đến nhiều dao động để đạt đến sự hội tụ, nhưng mỗi bước tính toán cho SGD nhanh hơn rất nhiều so với GD, vì nó chỉ sử dụng một mẫu huấn luyện (so với toàn bộ lô cho GD))

- Ưu điểm :
    - Nhanh hơn nhiều BGD
    - Online learning (mỗi lần thêm dữ liệu mới vào chỉ cần cập nhật trên 1 điểm dữ liệu đó thôi, phù hợp với online learning)
- Nhược điểm :
    - Cập nhật phương sai cao
    - Vẫn còn phụ thuộc vào nghiệm khởi tạo ban đầu và learning rate.(Ví dụ 1 hàm số có 2 global minimum thì tùy thuộc vào 2 điểm khởi tạo ban đầu sẽ cho ra 2 nghiệm cuối cùng khác nhau.)

➢ Trong khi Batch Gradient Descent cho một đồ thị "mượt" giảm dần, thì SGD giúp chúng ta có thể nhảy đến một tham số tốt hơn. Mặt khác, nó làm phức tạp hoá sự hội tụ chính xác, vì SGD có thể tiếp tục vượt qua điểm cực tiểu. Đồ thị hội tụ của nó không ổn định. Tuy nhiên, các nghiên cứu cho thấy rằng khi chúng

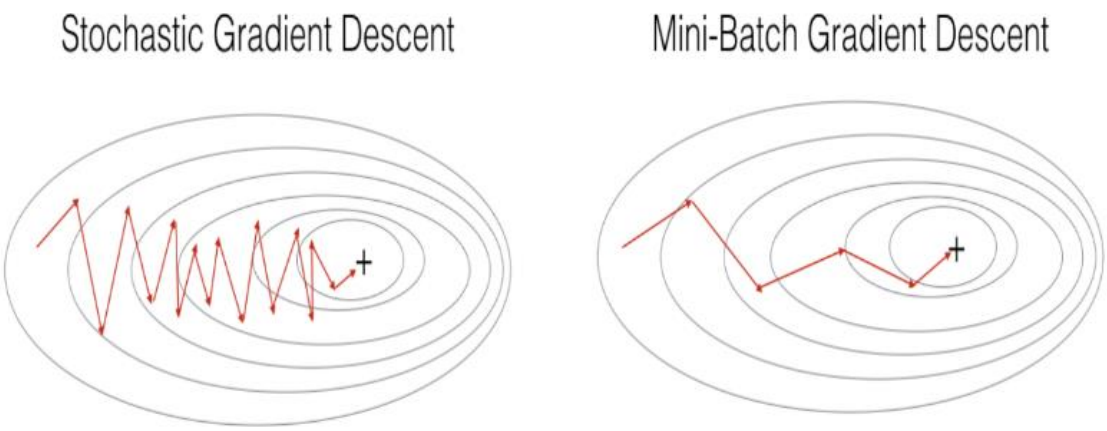
ta giảm dần learning rate. SGD cho thấy nó hội tụ nhanh hơn batch gradient descent, gần như chắc chắn hội tụ đến một điểm cục bộ hoặc toàn cục để tối ưu hàm mục tiêu.

1.2.5 Mini batch Gradient Descent

- Khác với SGD, mini-batch sử dụng mọt số lượng n lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu N rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu (trừ mini-batch cuối cùng có thể ít hơn nếu N không chia hết cho n). Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật.
- Cập nhật cho mỗi mini batch của n mẫu
- Cập nhật phương trình : $\theta = \theta - n.\nabla_\theta \, J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(
      loss_function, batch, params)
    params = params - learning_rate * params_grad
```

Code for mini batch gradient descent update



("+" biểu thị chi phí tối thiểu. Việc sử dụng các lô nhỏ trong thuật toán tối ưu hóa của bạn thường dẫn đến việc tối ưu hóa nhanh hơn.)

- Ưu điểm :
    - Giảm phương sai
    - Mini-batch GD giảm sự dao động của hàm mất mát so với SGD và chi phí tính gradient với k điểm dữ liệu là chấp nhận được. Mini-batch GD thường được lựa chọn khi huấn luyện mạng nơron và vì vậy trong một số trường hợp, SGD được hiểu là Mini-batch GD.
- Nhược điểm :
    - Mini batch size là siêu tham số, phổ biến 50-256

Với $x^{(i:i+n)}$ được hiểu là dữ liệu từ thứ i đến thứ i + n − 1 (theo ký hiệu của Python). Dữ liệu này sau mỗi epoch khác nhau chúng ta cần được xáo trộn. Các thuật toán khác cho GD như momentum, adagrad, adadelta ... mà chúng ta sẽ tìm hiểu các phần sau cũng có thể áp dụng vào đây.

- Nhìn chung, gradient descent làm giảm phương sai cập nhật tham số (hội tụ ổn định hơn) so với SGD.

1.2.6 Những vấn đề của Gradient Descent

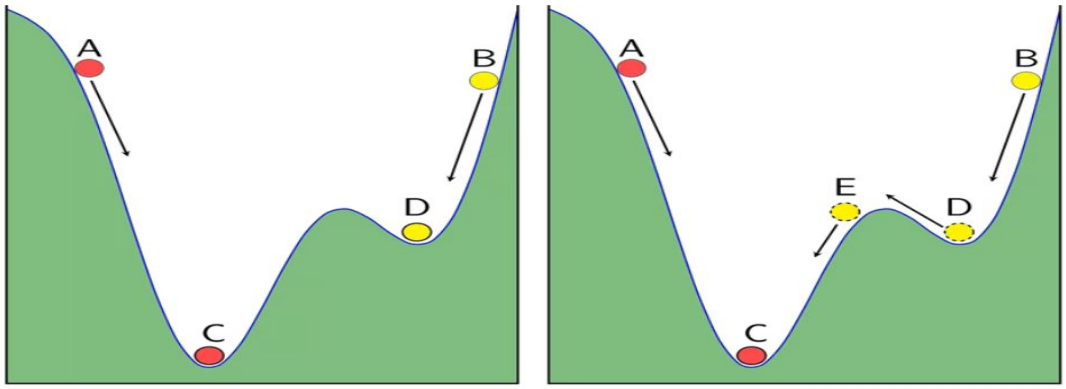
Method	Accuracy	Update Speed	Memory Usage	Online Learning
Batch gradient descent	Good	Slow	High	No
Stochastic gradient descent	Good (with annealing)	High	Low	Yes
Mini-batch gradient descent	Good	Medium	Medium	Yes

- Chọn Learning Rate Phù Hợp:
    - Learning rate quá nhỏ dẫn đến hội tụ chậm và khó khăn.
    - Learning rate quá lớn có thể gây phân kỳ và làm hàm mất mát dao động xung quanh mức tối thiểu hoặc thậm chí là không hội tụ.
- Lên Lịch Cho Tỷ Lệ Học Tập:
    - Cố gắng điều chỉnh learning rate trong quá trình huấn luyện.
    - xác định lịch trước, không thể thích ứng với đặc điểm của mọi bộ dữ liệu.
- Xử Lý Các Tham Số Khác Nhau:
    - Gradient Descent áp dụng cùng một learning rate cho tất cả các tham số, trong khi các tham số khác nhau có thể đòi hỏi learning rate khác nhau.
- Vượt Qua Điểm Cực Tiểu Địa Phương:
    - Khó khăn trong việc vượt qua các điểm cực tiểu địa phương, đặc biệt là các điểm yên ngựa (đạo hàm gần như bằng 0 ở mọi chiều).
    - Có thể cần thử nghiệm nhiều lần với các điểm khởi tạo khác nhau để tránh mắc kẹt ở điểm cực tiểu địa phương.
- Thách Thức Với Điểm Cực Tiểu Toàn Cục:

- Không có đảm bảo tìm ra điểm cực tiểu toàn cục, đặc biệt là trong bài toán có nhiều điểm cực tiểu.

- Tóm gọn lại, Stochastic Gradient Descent (SGD) thường phù hợp hơn để tối ưu hóa các hàm phi lồi (non-convex) vì nó có thể thoát khỏi các điểm yên tĩnh (local minima) và tìm được điểm tối thiểu toàn cục. Ngược lại, Batch Gradient Descent có thể bị mắc kẹt trong các điểm yên tĩnh.
- Để diễn đạt một cách đơn giản: Batch Gradient Descent chậm nhưng chính xác, trong khi Stochastic Gradient Descent nhanh nhưng không chính xác bằng. Sự lựa chọn giữa hai thuật toán này phụ thuộc vào vấn đề cụ thể, tập dữ liệu, và tài nguyên máy tính có sẵn.
- Mini batch sinh ra để giao thoa hai cái trên nó nhanh có cái chậm (BGD) và ít nhiễu hơi cái nhiễu (SGD)

1.3 Momentum



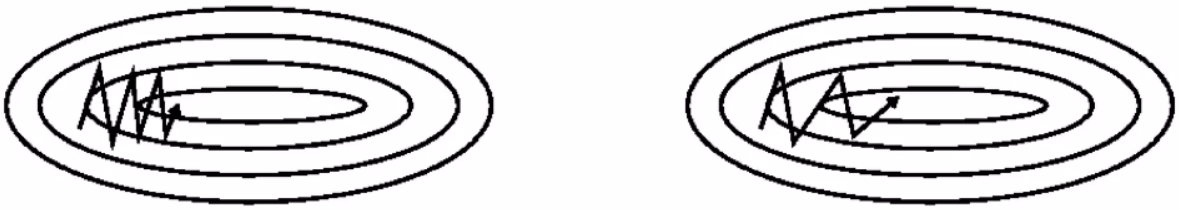
b) GD c) GD with momentum

- Dễ dàng thấy GD gặp khó khi di chuyển xuống global minimun (bị mắc kẹt ở local minimum)

- Dựa vào ý tưởng ở hình trên người ta add thêm γ vào vector ở bước trước v_{t-1} để cập nhật vector v_t, γ thường = 0.9
- Với công thức sau:

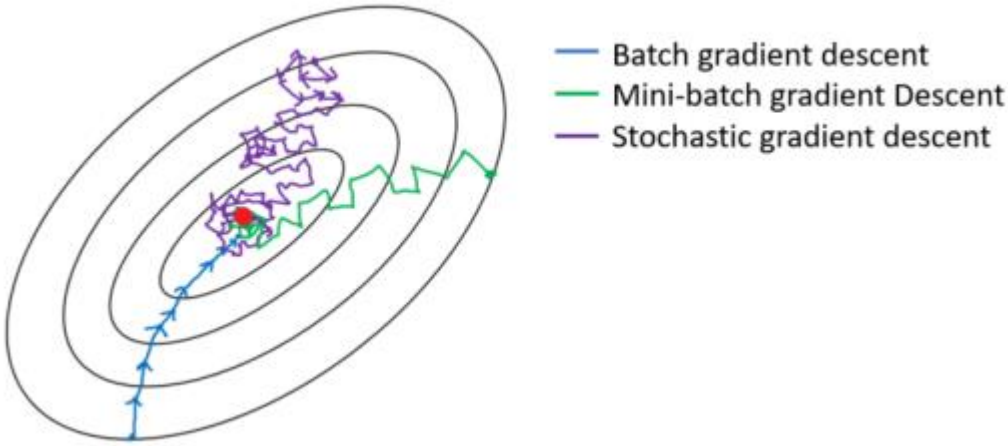
$$v_t = \gamma\ v_{t-1} + n.\nabla_\theta\ \mathrm{J}(\theta)$$

$$\theta = \theta - v_t$$



(a) SGD without momentum (b) SGD with momentum

- Ưu điểm
  - Giải quyết được vấn đề Gradient Descent không tiến được tới điểm global minimum mà chỉ dừng lại ở local minimum.



Gradient descent Là một thuật toán tối ưu nhưng việc học của nói liên quan tới learning rate, learning rate set cứng cho mỗi nút, vậy ở các dữ liệu đầu vào như

thế nào thì learning rate của các lần duyệt gradient descent thì learning rate đều như thế.

1.4 Adagrad

Trước đây, chúng ta thực hiện một update cho tất cả các tham số θ, cùng một lúc mỗi tham số θi sử dụng learning rate giống nhau η. Adagrad sử dụng learning rate khác nhau cho mỗi tham số θi tại mỗi bước t. Gọi gt là gradient tại bước t. G(t,i) là đạo hàm riêng của hàm mục tiêu theo tham số $\boldsymbol{\theta_i}$ tại bước thứ t:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

Khi đó, GD update cho mỗi tham số $\boldsymbol{\theta_i}$ tại bước thứ t trở thành:

$$\theta_{t+1,i} = \theta_{t,i} - \eta . g_{t,i}$$

Trong quy tắc cập nhật của mình, Adagrad sửa đổi sinh learning rate cho mỗi bước t cho mỗi tham số θi dựa trên các gradient quá khứ đã được tính toán cho $\boldsymbol{\theta_i}$:

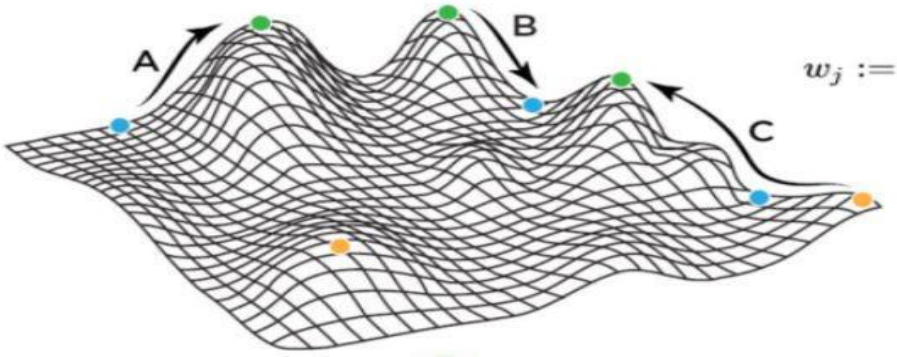
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} . g_{t,i}$$

Trong đó, Gt ∈ R^(dxd) là một ma trận đường chéo mà các phần tử trên đường chéo i, i là tổng bình phương của các gradient theo θi cho đến bước thứ t, và ε là hệ số để tránh chia cho 0 (thường là 10−8). Theo thử nghiệm, nếu không có căn bậc 2 dưới mẫu, thuật toán lại hoạt động tệ hơn nhiều.

Như vậy, Gt gồm tổng bình phương các gradient quá khứ của tất cả các tham số θ dọc theo đường chéo, chúng ta có thể viết tổng quát lại công thức cập nhật cho Adagrad như sau (Thông thường, η = 0.01.):

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

- Ví dụ ở nút B muốn xuống nút màu xanh dương thì nó sẽ nhận 2 dữ liệu đầu vào và điều chỉnh tốc độ học phải làm sao cho nó xuống điểm màu xanh dương này một cách sớm nhất



$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j.$$

- Vì G(t,i) được tính bằng cách tích lũy bình phương của độ dốc, các tham số có độ dốc lớn sẽ có G(t,i) lớn và do đó, learing rate η(t,i) sẽ giảm đi đáng kể, giúp ổn định quá trình học và ngăn chặn bước nhảy quá lớn.

- Ưu điểm:
    - Phù hợp với dữ liệu thưa
    - Ít điều chỉnh learning rate theo cách thủ công
- Nhược điểm:
    - Tích lũy gradient bình phương ở mẫu, làm cho learning rate giảm xuống và trở nên cực kỳ nhỏ (làm việc tranining trở nên đóng băng)

- Với những điều trên, cho các tham số có biến độ dốc thay đổi nhanh và không thường xuyên, learning rate sẽ giảm đi, giúp cải thiện sự ổn định của quá trình tối ưu hóa. Ngược lại, đối với các tham số có độ dốc thay đổi nhỏ và thường xuyên, learning rate sẽ giữ ổn định hoặc giảm ít, giúp duy trì độ nhạy của mô hình đối với các biến độ dốc thay đổi nhỏ.

1.5 Rmsprop

- RMSProp là phương pháp điều chỉnh tỷ lệ học (learning rate)
- Sử dụng trung bình bình phương của gradient để chuẩn hóa gradient (chia learing rate cho giá trị trung bình của gradient bình phương)
- Điều này giúp tránh việc có bước nhảy quá lớn (Exploding Gradient) khi độ dốc quá lớn và ngăn chặn tình trạng bất trắc dữ liệu. Ngược lại, với các độ dốc nhỏ, nó giúp tăng bước nhảy để tránh tình trạng biến mất độ dốc (Vanishing Gradient).

- RMSProp tự động điều chỉnh tốc độ học tập, và chọn một tỉ lệ học tập khác nhau cho mỗi tham số.

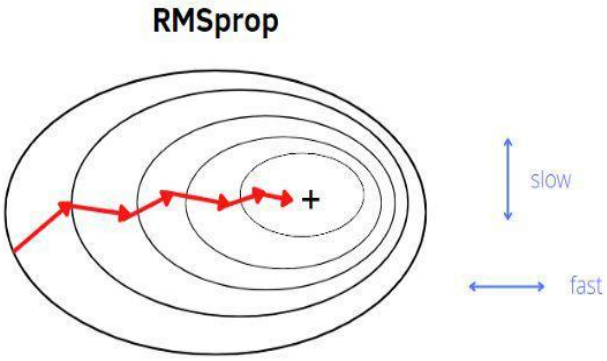
$$E[g^2]_t = 0.9[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$

RMSProp cũng chia learning rate cho $E[g^2]_t$.

Hinton đề nghị $\gamma = 0.9$ và giá trị mặc định tốt cho $\eta = 0.001$

Càng gần vào trung tâm thì các bước này diễn ra càng nhiều và chạy chậm lại (điểm cực tiểu mong muốn có được)



1.6 Adam

1.6.1 Adam là gì?

- Adam (Adaptive Moment Estimation) là một thuật toán tối ưu hóa các hàm mục tiêu ngẫu nhiên dựa trên gradient bậc nhất và sự ước tính thích ứng của các mô-men (moment) bậc thấp.
- Một phương pháp tối ưu hóa ngẫu nhiên hiệu quả chỉ yêu cầu gradient bậc nhất với bộ nhớ thấp.
- Phương pháp này tính toán từng tỷ lệ học thích ứng cho các tham số khác nhau.
- Adam sử dụng ước tính của khoảng thời gian thứ nhất và thứ hai để điều chỉnh tốc độ học cho mỗi trọng số của mạng nơ-ron.

1.6.2 Adam sinh ra từ đâu?

- Là sự kết hợp những lợi thế của hai thuật toán tối ưu AdaGrad và RMSProp nhằm giảm dần độ dốc ngẫu nhiên.
- Adam thay vì điều chỉnh các tham số learning rate dựa trên thời điểm trung bình đầu tiên (giá trị trung bình) như RMSProp, Adam sử dụng giá trị trung bình của thời điểm thứ hai của các gradient (Phương sai không tập trung)

1.6.4 Adam thuật toán?

Nó tính toán mức trung bình có trọng số theo cấp số nhân của độ dốc trong quá khứ và lưu trữ nó trong các biến (trước khi hiệu chỉnh sai lệch) và (có hiệu chỉnh sai lệch).
Nó tính toán giá trị trung bình theo cấp số nhân của bình phương của các gradient trong quá khứ và lưu trữ nó trong các biến (trước khi hiệu chỉnh sai lệch) và (với hiệu chỉnh sai lệch).
Nó cập nhật các tham số theo hướng dựa trên việc kết hợp thông tin từ "1" và "2".

Algorithm 1: Adam, ký hiệu g_t^2 là bình phương phép nhân element-wise $g_t \odot g_t$, các tham số khuyến nghị là $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ và $\varepsilon = 10^{-8}$.

Tất cả các toán tử trên vector đều là phép element-wise (tức là hiện tượng tương ứng của từng phần tử)

Require: α : độ lớn bước nhảy

Require: $\beta_1, \beta_2 \in [0, 1)$: tỷ lệ giảm dần theo cấp số mũ cho các ước tính thời điểm

Require: f(θ) : hàm mục tiêu ngẫu nhiên với tham số θ

Require: θ_0 : khởi tạo tham số vector

$m_0 \leftarrow 0$;(Khởi tạo vector moment thứ nhất)

$v_0 \leftarrow 0$; (Khởi tạo vector moment thứ hai)

$t \leftarrow 0$ (Khởi tạo bước nhảy theo thời gian)

Mã giả:

while θ_t not converged **do**

$t \leftarrow t + 1$;

$g_t \leftarrow \nabla_t f_t(\theta_{t-1})$;

$m_t \leftarrow \beta_1.m_{t-1} + (1 - \beta_1).g_t$; (cập nhật moment lực thứ 1)

$v_t \leftarrow \beta_2.v_{t-1} + (1 - \beta_2).g_t^2$; (cập nhật moment lực thứ 2)

$\hat{m}_t \leftarrow m_t /(1 - \beta^t_1)$; (tính toán cập nhật moment lực 1 và 2)

$\hat{v}_t \leftarrow v_t /(1 - \beta^t_2)$;

end while

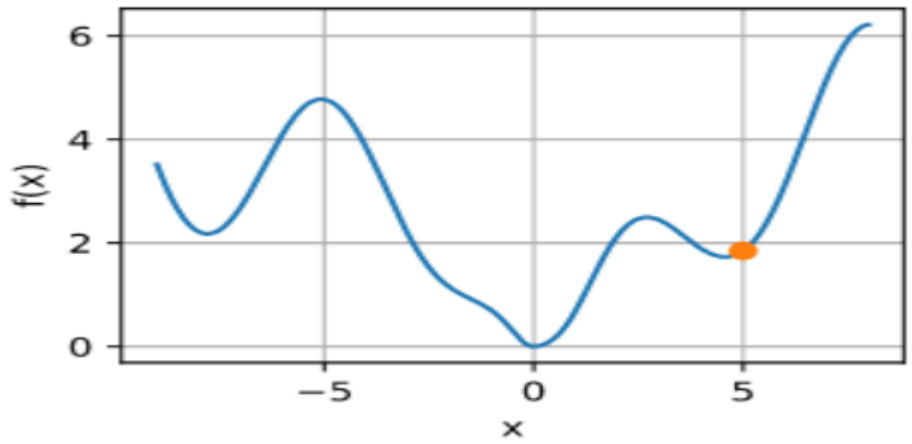
return θt (trả về theta lần chạy thứ t)

1.7 Kết Luận Từ Bài Phân Tích Và Đoạn Code?

1.7.1 Bài toán đồ thị hàm sin

$$f(x) = log(1 + (\mid x \mid)^{2+sinx})$$

- Đồ thị:



Với khởi tạo ban đầu x0 = 2, learning rate = 0.01, β1 = 0.9, β2 = 0.99 , kết quả của các thuật toán

```
gd epoch  868 x:  [-1.06607856e-08]
Tham số Tối ưu hóa Cuối cùng (GD): [-1.06607856e-08]
Số Epochs (GD): 868
BGD epoch  868 x:  [-1.06607856e-08]
Tham số Tối ưu hóa Cuối cùng (BGD): [-1.06607856e-08]
Số Epochs (BGD): 868
MBGD epoch  9999 x:  [-100.9199096]
Tham số Tối ưu hóa Cuối cùng (MBGD): [-100.9199096]
Số Epochs (MBGD): 9999
SGD epoch  9999 x:  [-100.9199096]
Tham số Tối ưu hóa Cuối cùng (SGD): [-100.9199096]
Số Epochs (SGD): 9999
gd_momentum epoch  265 x:  [-2.17763624e-06]
Tham số Tối ưu hóa Cuối cùng (Momentum): [-2.17763624e-06]
Số Epochs (Momentum): 265
adagrad epoch  9999 x:  [0.33193342]
Tham số Tối ưu hóa Cuối cùng (Adagrad): [0.33193342]
Số Epochs (Adagrad): 9999
RMSPROP epoch  242 x:  [-4.9640172e-07]
Tham số Tối ưu hóa Cuối cùng (RMSPROP): [-4.9640172e-07]
Số Epochs (RMSPROP): 242
ADAM: epoch  473 x:  [-2.87651112e-06]
Tham số Tối ưu hóa Cuối cùng (Adam): [-2.87651112e-06]
Số Epochs (Adam): 473
```

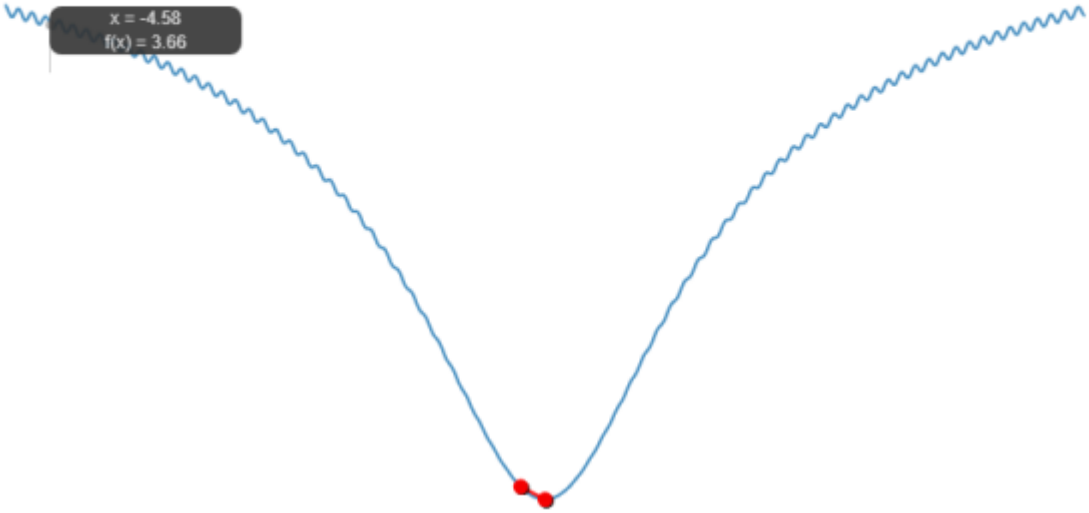
- Có thể thấy với khởi tạo ban đầu lý tưởng các thuật toán đều có thể hội tụ tốt , với thời gian hội tụ ngắn .
- Với khởi tạo khó hơn x0 = 8, $\alpha = 0.01$, $\beta 1 = 0.9$, $\beta 2 = 0.99$, kết quả của các thuật toán

```
gd epoch  1433 x:  [4.56611984]
Tham số Tối ưu hóa Cuối cùng (GD): [4.56611984]
Số Epochs (GD): 1433
BGD epoch  1433 x:  [4.56611984]
Tham số Tối ưu hóa Cuối cùng (BGD): [4.56611984]
Số Epochs (BGD): 1433
MBGD epoch  9999 x:  [0.90318148]
Tham số Tối ưu hóa Cuối cùng (MBGD): [0.90318148]
Số Epochs (MBGD): 9999
SGD epoch  9999 x:  [0.90318148]
Tham số Tối ưu hóa Cuối cùng (SGD): [0.90318148]
Số Epochs (SGD): 9999
gd_momentum epoch  296 x:  [4.56612179]
Tham số Tối ưu hóa Cuối cùng (Momentum): [4.56612179]
Số Epochs (Momentum): 296
adagrad epoch  9999 x:  [5.64208059]
Tham số Tối ưu hóa Cuối cùng (Adagrad): [5.64208059]
Số Epochs (Adagrad): 9999
RMSPROP epoch  382 x:  [4.56611898]
Tham số Tối ưu hóa Cuối cùng (RMSPROP): [4.56611898]
Số Epochs (RMSPROP): 382
ADAM: epoch  679 x:  [4.56611977]
Tham số Tối ưu hóa Cuối cùng (Adam): [4.56611977]
Số Epochs (Adam): 679
```

- Có thể thấy các thuật toán đều hội tụ đến điểm local minimum và không thể tới được điểm global minimum .

1.7.2 Bài toán đồ thị hàm tan

$$f(x) = (2 + \frac{sin50x}{50})(arctanx)^2$$



- Với khởi tạo ban đầu : x0 = 4.03, α = 0.06, β1 = 0.9, β2 = 0.99 kết quả đạt được

```
gd epoch  23 x:  [3.9866718]
Final Optimized Parameters (GD): [3.9866718]
Number of Epochs (GD): 23
BGD epoch  23 x:  [3.9866718]
Final Optimized Parameters (bgd): [3.9866718]
Number of Epochs (BGD): 23
MBGD epoch  9999 x:  [-101.92637317]
Final Optimized Parameters (mbgd): [-101.92637317]
Number of Epochs (MBGD): 9999
SGD epoch  9999 x:  [-101.92637317]
Final Optimized Parameters (sgd): [-101.92637317]
Number of Epochs (SGD): 9999
gd_momentum epoch  246 x:  [3.9866717]
Final Optimized Parameters (Momentum): [3.9866717]
Number of Epochs (Momentum): 246
adagrad epoch  61 x:  [3.98667183]
Final Optimized Parameters (Adagrad): [3.98667183]
Number of Epochs (Adagrad): 61
RMSPROP epoch  9999 x:  [3.98156969]
Final Optimized Parameters (RMSPROP): [3.98156969]
Number of Epochs (RMSPROP): 9999
ADAM: epoch  168 x:  [3.98666652]
Final Optimized Parameters (Adam): [3.98666652]
Number of Epochs (Adam): 168
```

- Trường hợp này các thuật toán khác có thể hội tụ tốt nhưng adam lại không qua được local minimum do điểm khởi tạo ban đầu quá gần với local minimum mà tốc độ học ban đầu lại quá nhỏ dẫn tới "ma sát" quá lớn khiến cho điểm k vượt qua được local minimum
- Tăng learning rate lên 1 và giữ nguyên những hyper-parameter khác thì kết quả :

```
gd epoch  9999 x:  [-0.13295658]
Final Optimized Parameters (GD): [-0.13295658]
Number of Epochs (GD): 9999
BGD epoch  9999 x:  [-0.13295658]
Final Optimized Parameters (bgd): [-0.13295658]
Number of Epochs (BGD): 9999
MBGD epoch  9999 x:  [-10591.60731716]
Final Optimized Parameters (mbgd): [-10591.60731716]
Number of Epochs (MBGD): 9999
SGD epoch  9999 x:  [-10591.60731716]
Final Optimized Parameters (sgd): [-10591.60731716]
Number of Epochs (SGD): 9999
gd_momentum epoch  9999 x:  [140.58435864]
Final Optimized Parameters (Momentum): [140.58435864]
Number of Epochs (Momentum): 9999
adagrad epoch  6848 x:  [-0.03020883]
Final Optimized Parameters (Adagrad): [-0.03020883]
Number of Epochs (Adagrad): 6848
RMSPROP epoch  9999 x:  [0.30484974]
Final Optimized Parameters (RMSPROP): [0.30484974]
Number of Epochs (RMSPROP): 9999
ADAM: epoch  9999 x:  [-0.20488129]
Final Optimized Parameters (Adam): [-0.20488129]
Number of Epochs (Adam): 9999
```

- Vì learning rate quá lớn nên trong 10000 epoch momentum chưa kịp hội tụ còn adam đã hội tụ tốt.

- Thuật toán SGD có thời gian huấn luyện thấp, tốc độ hội tụ chậm nhưng ổn định, và trong hai thực nghiệm đều đạt được ngưỡng độ chính xác đề ra. Tuy SGD luôn đem lại một trong các tần suất lỗi nhỏ nhất.
- Thuật toán Adagrad yêu cầu thời gian huấn luyện lớn hơn SGD.
- Thuật toán Adam hội tụ nhanh nhưng không đạt được ngưỡng độ chính xác đề ra.

CHƯƠNG 2 – CONTINUAL LEARNING VÀ TEST PRODUCTION

2.1 Continual Learning

2.1.1 Continual Learning là gì?

Ở cấp độ cơ bản, Machine Learning sử dụng các thuật toán để cung cấp cho máy tính khả năng tìm hiểu dữ liệu, tìm mẫu và đưa ra kết quả dự đoán. Tuy nhiên, vấn đề là mô hình Machine Learning truyền thống giả định rằng dữ liệu sẽ giống với dữ liệu mà nó đã được đào tạo, điều này không phải lúc nào cũng đúng.

Học máy liên tục (CML) giải quyết vấn đề này bằng cách giám sát và đào tạo lại các mô hình với dữ liệu cập nhật. Mục tiêu của CML là mô phỏng khả năng thu thập và tinh chỉnh thông tin liên tục của con người.

Học tập liên tục "Continual learning" (CL) tập trung vào việc phát triển các mô hình để học các nhiệm vụ mới trong khi vẫn giữ lại thông tin từ các nhiệm vụ trước đó. CL là một lĩnh vực nghiên cứu quan trọng vì nó giải quyết tình huống thực tế trong đó dữ liệu và nhiệm vụ liên tục thay đổi và mô hình phải thích ứng với những thay đổi này mà không quên kiến thức trước đó.

Ví dụ: Với app Netflix, hệ thống này có tính năng “Up Next” phát các chương trình tương tự như những chương trình bạn đã xem gần đây, thì bạn đã thấy mô hình CML hoạt động. Để theo kịp nguồn cung cấp chương trình mới không ngừng nghỉ, cũng như sự thay đổi sở thích và xu hướng của khách hàng Netflix, dữ liệu đến phải được nhập liên tục. Và mô hình phải cập nhật liên tục để có khả năng đề xuất các chương trình hoặc phim phù hợp.

2.1.2 Continual Learning Type?

- Incremental Learning:
    - Định nghĩa: Học tăng cường liên quan đến việc cập nhật mô hình với các ví dụ dữ liệu mới một cách từng bước, dần dần thích ứng với sự thay đổi trong phân phối dữ liệu.
    - Ứng dụng: Thường được sử dụng khi xử lý dữ liệu trực tuyến hoặc trong những tình huống nơi tập dữ liệu thay đổi theo thời gian.
- Transfer Learning:
    - Định nghĩa: Học truyền chuyển liên quan đến việc tận dụng kiến thức được học từ việc huấn luyện trên một nhiệm vụ và áp dụng nó vào một nhiệm vụ khác liên quan.
    - Ứng dụng: Hữu ích khi kiến thức thu được trong một lĩnh vực có thể hữu ích cho việc học trong một lĩnh vực khác, giảm thiểu tài nguyên tính toán và thời gian huấn luyện.
- Lifelong Learning (Continual Learning):
    - Định nghĩa: Học suốt đời đề cập đến khả năng của mô hình thích ứng và học từ một luồng dữ liệu liên tục trong một khoảng thời gian dài mà không quên kiến thức trước đó.
    - Ứng dụng: Quan trọng trong các tình huống mô hình cần giữ lại kiến thức về nhiều nhiệm vụ và thích ứng với các nhiệm vụ mới mà không cần bắt đầu lại từ đầu.

2.1.3 Continual Learning Process?

Quy trình:

1. Tiền xử lý (Pre-processing): Chuẩn bị dữ liệu trước khi đưa vào mô hình, bao gồm các công đoạn như chuẩn hóa, mã hóa, và xử lý dữ liệu nhiễu.

2. Lựa chọn mô hình (Model Selection): Chọn mô hình phù hợp với bài toán cụ thể, dựa trên đặc điểm của dữ liệu và yêu cầu hiệu suất.
3. Điều chỉnh siêu tham số (Hyperparameter Tuning): Tối ưu hóa các siêu tham số để cải thiện hiệu suất của mô hình.
4. Huấn luyện (Training): Huấn luyện mô hình trên tập dữ liệu huấn luyện để nó có khả năng dự đoán tốt trên dữ liệu mới.
5. Triển khai (Deployment): Triển khai mô hình để sử dụng trong môi trường thực tế.
6. Theo dõi (Monitoring): Giám sát hiệu suất của mô hình sau khi triển khai, đảm bảo nó duy trì độ chính xác.

Bổ sung:

1. Luyện Tập Dữ Liệu (Data Rehearsal): Là bước đặc biệt trong quá trình học liên tục, bao gồm việc đưa dữ liệu mới vào quá trình huấn luyện để mô hình cập nhật kiến thức từ dữ liệu mới.
2. Triển Khai Chiến Lược Học Liên Tục (Continuous Learning Strategy Implementation): Xác định và triển khai chiến lược cụ thể để mô hình có thể liên tục học từ dữ liệu mới một cách linh hoạt và hiệu quả.

2.1.4 Continual Learning Advantage?

- Tổng Quát Hóa: Mô hình trở nên mạnh mẽ và chính xác hơn đối diện với dữ liệu mới.
- Giữ Lại Thông Tin: Mô hình tích lũy kiến thức trước đó, duy trì thông tin qua các lần học liên tục.
- Tính Linh Hoạt: Mô hình thích ứng với kiến thức mới, có khả năng dự đoán mạnh mẽ hơn trong dài hạn.

2.1.5 Continual Learning Limited?

Từ Góc Độ Vận Hành:

- Chi Phí: Phức tạp tính toán và đòi hỏi nhiều nguồn lực về dữ liệu, nguồn lực con người và máy tính, dẫn đến chi phí kinh tế cao hơn.

Từ Góc Độ Mô Hình:

- Quản lý Mô Hình: Sự cập nhật thường xuyên tạo ra nhiều mô hình, làm phức tạp quá trình xác định mô hình hoạt động tốt nhất.
- Thay Đổi Dữ Liệu (Data Drift): Có rủi ro mất khả năng dự đoán nếu phân phối đặc trưng thay đổi đột ngột.

2.2 Testing In Production

2.2.1 Đặt vấn đề

Để đánh giá đủ mô hình của bạn, bạn cần sự kết hợp giữa đánh giá offline ư đánh giá trực tuyến được thảo luận trong phần này. Để hiểu tại sao đánh giá offline không đủ, hãy đi qua hai loại kiểm thử quan trọng cho đánh giá offline: test splits và backtests.

- Test Splits (Tập Thử Nghiệm):
  - Test splits được sử dụng để đánh giá mô hình offline, cung cấp một thước đo đáng tin cậy cho so sánh giữa các mô hình.
  - Thường là tập dữ liệu tĩnh và phải là tĩnh để bạn có một thước đo đáng tin cậy để so sánh giữa nhiều mô hình. và không thay đổi để tạo ra một bộ kiểm thử ổn định.
  - Sẽ khó để so sánh kết quả thử nghiệm của hai mô hình nếu chúng được thử nghiệm trên các tập thử nghiệm khác nhau.
  - Đánh giá mô hình trên test splits từ phân phối dữ liệu cũ không đủ khi mô hình được cập nhật để thích ứng với phân phối mới.

- Backtests (Kiểm Thử Quay Lại):

- Backtests là quá trình kiểm thử mô hình dự đoán trên dữ liệu mới nhất sau khi mô hình đã được cập nhật.
- Mục tiêu là đánh giá hiệu suất của mô hình trên dữ liệu từ một khoảng thời gian cụ thể trong quá khứ.
- Không đủ để thay thế test splits tĩnh vì nếu có vấn đề với dữ liệu gần đây, đánh giá chỉ dựa trên nó không đảm bảo.
- Nếu có vấn đề gì đó xảy ra với ống dữ liệu của bạn và một số dữ liệu từ giờ cuối cùng bị hỏng, việc đánh giá mô hình của bạn chỉ dựa trên dữ liệu gần đây này không đủ. Với backtests, bạn vẫn nên đánh giá mô hình của mình trên một bộ thử nghiệm tĩnh mà bạn đã nghiên cứu kỹ

⇨ Bởi vì phân phối dữ liệu thay đổi, việc một mô hình làm tốt trên dữ liệu từ giờ cuối cùng không có nghĩa là nó sẽ tiếp tục làm tốt trên dữ liệu trong tương lai. Cách duy nhất để biết liệu một mô hình có làm tốt trong sản xuất hay không là triển khai nó.

2.2.2 Shadow Deployment

- Triển khai bóng có thể là cách an toàn nhất để triển khai mô hình hoặc bất kỳ cập nhật phần mềm nào. Quy trình triển khai bóng hoạt động như sau:
    - Triển khai mô hình ứng cử viên song song với mô hình hiện tại.
    - Đối với mỗi yêu cầu đến, định tuyến nó đến cả hai mô hình để đưa ra dự đoán, nhưng chỉ phục vụ dự đoán của mô hình hiện tại cho người dùng.
    - Ghi lại các dự đoán từ mô hình mới cho mục đích phân tích.
    - Chỉ khi bạn xác định rằng các dự đoán từ mô hình mới là hài lòng, bạn mới thay thế mô hình hiện tại bằng mô hình mới.
    - ➢ Vì bạn không phục vụ các dự đoán từ mô hình mới cho người dùng cho đến khi bạn đảm bảo rằng các dự đoán của mô hình là hài lòng, rủi ro của

mô hình mới giảm đi, ít nhất là không cao hơn so với mô hình hiện tại. Tuy nhiên, kỹ thuật này không phải lúc nào cũng được ưa chuộng vì nó tăng chi phí tính toán dự đoán của hệ thống, gấp đôi so với trạng thái không triển khai bóng.

2.2.4 A/B Testing

- Định Nghĩa: A/B Testing là một phương pháp so sánh hai biến thể của một đối tượng để xác định biến thể nào hiệu quả hơn, thường thông qua việc đánh giá phản hồi của người dùng.
- Triển Khai:
    - Triển khai mô hình ứng cử viên cùng với mô hình hiện tại.
    - Một phần lưu lượng được chuyển hướng đến mô hình mới để đưa ra dự đoán, phần còn lại đến mô hình hiện tại.
    - Giám sát và phân tích dự đoán và phản hồi từ người dùng để xác định sự chênh lệch giữa hiệu suất của hai mô hình có ý nghĩa thống kê hay không.

2.2.4 Canary Release

- Định Nghĩa: Canary Release là một kỹ thuật giảm rủi ro khi triển khai một phiên bản phần mềm mới vào môi trường sản xuất bằng cách tung ra thay đổi một cách từ từ đối với một phần nhỏ người dùng trước khi triển khai cho toàn bộ cơ sở hạ tầng và làm cho nó sẵn có cho mọi người.
- Triển Khai Trong:
    - Triển khai mô hình ứng cử viên song song với mô hình hiện tại, mô hình ứng cử viên được gọi là "canary."
    - Một phần của lưu lượng được chuyển hướng đến mô hình ứng cử viên.

  - Nếu hiệu suất của nó làm bạn hài lòng, tăng lưu lượng đến mô hình ứng cử viên. Ngược lại, hủy bỏ canary và chuyển toàn bộ lưu lượng về mô hình hiện tại.
  - Dừng khi entiwer canary phục vụ toàn bộ lưu lượng (mô hình ứng cử viên đã thay thế mô hình hiện tại) hoặc khi canary bị hủy bỏ.
- Đánh giá hiệu suất
  - Đánh giá hiệu suất của mô hình ứng cử viên so với mô hình hiện tại dựa trên các chỉ số quan trọng mà bạn quan tâm.
  - Nếu các chỉ số chính của mô hình ứng cử viên giảm đáng kể, canary sẽ bị hủy bỏ và toàn bộ lưu lượng sẽ được chuyển về mô hình hiện tại.
- Liên kết:
  - Canary Release có thể được sử dụng để thực hiện A/B Testing vì sự tương đồng trong cách triển khai của chúng.
  - Tuy nhiên, có thể thực hiện phân tích canary mà không cần A/B Testing, chẳng hạn như không cần ngẫu nhiên hóa lưu lượng để chuyển hướng đến từng mô hình.