

Optimization Methods and Regularization

Lê Nhựt Nam¹ Huỳnh Viết Thám¹ Lê Công Luận¹ Huỳnh Hoàng Huy¹

¹Faculty of Information Technology, University Of Science, VNU-HCMC

Seminar, July 2022

- 1 Giới thiệu
- 2 Gradient Descent
- 3 Stochastic Gradient Descent
- 4 Regularization
- 5 Các thuật toán tối ưu hoá Gradient Descent
- 6 Thực nghiệm kết quả và đánh giá
- 7 Kết luận
- 8 Tài liệu tham khảo
- 9 Phụ lục: Các bộ tối ưu gần đây; Phương pháp tối ưu toàn phương; Mã nguồn

Để mô hình đạt độ chính xác cao thì output của mô hình phải tiến gần đến giá trị target.

=> Cần một bộ trọng số weights W và bias b tốt.

- Random nhiều lần và lấy kết quả tốt nhất ?
- **Gradient Descent**

Thuật toán Gradient Descent có 2 phiên bản phổ biến:

- Phiên bản gốc.
- Phiên bản sử dụng thuật toán tối ưu hóa ngẫu nhiên “stochastic” (được sử dụng nhiều).

Loss Landscape và Optimization Surface

“Landscape” là các features có thể tìm thấy trong khu vực tìm kiếm.

Quá trình tối ưu hóa bề mặt (optimization surface) cũng chính là việc tối ưu hóa những sai khác giữa các cặp features khi đối sánh. Thuật toán gradient descent sẽ liên tục thực hiện việc tối ưu hóa bề mặt nhằm mục đích tìm ra điểm tối ưu nhất.

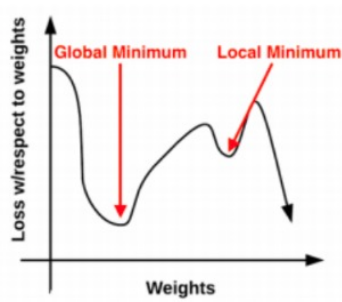


Figure 1: Mô phỏng các điểm tối ưu ứng với từng bộ trọng số.

Loss Landscape và Optimization Surface

Ở 1 góc độ 3D, với 2 trọng số θ_1 , θ_2 , đồ thị sẽ có dạng thung lũng (Hình 2). Có rất nhiều bộ tổ hợp θ_1 , θ_2 ứng với 1 giá trị lỗi. Nếu cứ duyệt một cách không có kế hoạch, chúng ta sẽ rất khó để đạt được mục tiêu.

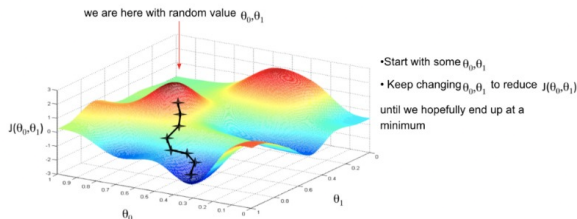


Figure 2: Mô phỏng 3D mối tương quan giữa hàm lỗi và trọng số θ_1 , θ_2 .

Vấn đề: Khi đang ở thời điểm thứ t với bộ trọng số θ , làm sao ta có thể tìm được bộ trọng số mới θ' trong thời điểm thứ $t+1$?

“Gradient” trong Gradient Descent

Robot R đang thực hiện nhiệm vụ tìm kiếm điểm thấp nhất trong cái tô (Hình 3). Vấn đề: làm sao để R có thể biết được *phương hướng* để đi đến điểm đó?

=> Dựa vào độ dốc.



Figure 3: Mô phỏng việc tìm kiếm điểm cực tiểu của Robot R trong cái tô .

“Gradient” trong Gradient Descent

Thuật toán Gradient Descent có nhiệm vụ tìm ra độ dốc của vị trí hiện tại, từ đó chọn được hướng đi kế tiếp.

Cụ thể, giả sử với bộ trọng số $W = w_1, w_2, \dots, w_n$, ta sẽ tìm được đầu ra dự đoán và tính được độ lỗi L từ đó. Công việc kế đến chính là đạo hàm riêng phần hàm L cho từng w_i , điều này đồng nghĩa với việc, ta đang xem xét sự biến thiên của hàm lỗi ứng với sự thay đổi của từng trọng số. Từ đó ta có thể biết được trọng số nào ảnh hưởng mạnh và trọng số nào ảnh hưởng thấp.

Tập hợp tất cả các đạo hàm riêng phần ta gọi là gradient. Công thức đạo hàm riêng phần được thể hiện bên dưới:

$$\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Thuật toán 1 Giải thuật Gradient Descent

- 1: **while** True **do**
 - 2: $W_{gradient} \leftarrow evaluate_gradient(loss, data, W)$
 - 3: $W += -alpha * W_{gradient}$
 - 4: **end while**
-

Stochastic Gradient Descent

Tổng quát

Stochastic Gradient Descent và các biến thể của nó là các thuật toán tối ưu được sử dụng phổ biến nhất trong học máy nói chung và học sâu nói riêng. Stochastic Gradient Descent là mở rộng của thuật toán Gradient Descent được giới thiệu ở trên.

Ý tưởng chung

Thay vì sau mỗi epoch chúng ta sẽ cập nhật trọng số (Weight) 1 lần thì trong mỗi epoch có N điểm dữ liệu chúng ta sẽ cập nhật trọng số N lần.

Nhận xét ý tưởng

- Nhìn vào 1 mặt, SGD sẽ làm giảm đi tốc độ của 1 epoch.
- Nhìn theo 1 hướng khác, SGD sẽ hội tụ rất nhanh chỉ sau vài epoch.

Công thức SGD cũng tương tự như GD nhưng thực hiện trên từng điểm dữ liệu. 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 online learning.

Mini-batch Gradient Descent

Xét thuật toán vanilla GD, rõ ràng là phương thức này sẽ chạy rất chậm trên các tập dữ liệu lớn.

Lý do: Mỗi lần lặp lại của gradient, chúng ta cần tính toán dự đoán cho từng điểm đào tạo trong dữ liệu đào tạo trước khi chúng ta được phép cập nhật ma trận trọng số.

Sự khác biệt với Vanilla GD và Stochastic GD

Sự khác biệt duy nhất với Vanilla GD và SGD là sự bổ sung của hàm `next_training_batch`. Thay vì tính toán gradient trên toàn bộ tập dữ liệu, chúng ta thay vào đó mẫu dữ liệu của chúng ta, thu được một batch. Chúng ta đánh giá gradient trên batch và cập nhật trọng số ma trận W .

Thuật toán 2 Giải thuật Mini-batch Gradient Descent

```
1: while True do  
2:   batch = next_training_batch(data, 256)  
3:   Wgradient  $\leftarrow$  evaluate_gradient(loss, data, W)  
4:   W += -alpha*Wgradient  
5: end while
```

Ý tưởng

Ở mỗi bước của thuật toán, lấy mẫu một mini-batch các mẫu dữ liệu ngẫu nhiên đồng đều từ tập huấn luyện. Kích thước của mini-batch thường được chọn là số tương đối nhỏ có giá trị từ một đến vài trăm. Quan trọng hơn, kích thước này thường được giữ nguyên ngay cả khi kích thước của tập huấn luyện tăng lên. Như vậy, khi đó tập huấn luyện hàng tỉ mẫu sẽ chỉ còn khoảng một trăm mẫu.

Mini-batch Stochastic Gradient Descent

Tại sao phải sử dụng kích thước batch > 1 ?

- Để bắt đầu, kích thước batch > 1 giúp giảm phương sai trong cập nhật tham số, dẫn đến sự hội tụ ổn định hơn.
- Thứ hai, hàm bình phương thường được sử dụng cho kích thước batch vì chúng cho phép các thư viện tối ưu hóa đại số tuyến tính bên trong để có hiệu quả hơn.

Regularization là một kỹ thuật được áp dụng nhằm tránh bị overfitting trong quá trình học.

Thông thường, regularization thay đổi hàm loss để **penalize** weights trong quá trình học.

$$\text{Cost function} = \text{Loss} + \text{Regularization term}$$

Ghi chú: Regularization thực hiện dựa trên giả định rằng những giá trị weights nhỏ hơn sẽ tạo ra model đơn giản hơn, giúp tránh bị overfitting.

Regularization

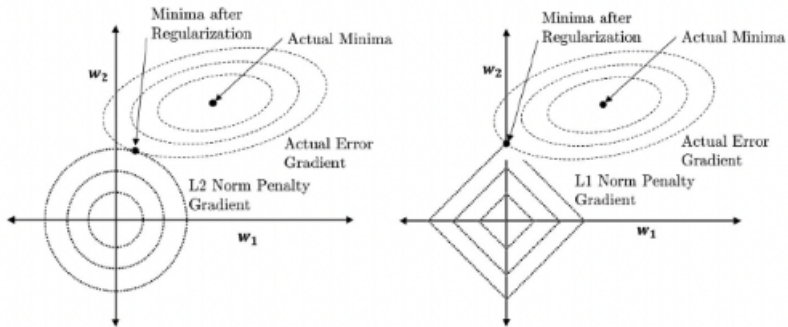


Figure 4: Tham số norm penalties: L2 norm regularization (trái) và L1 norm regularization (phải)

L1 regularization: Cộng thêm vào hàm loss tổng giá trị tuyệt đối của trọng số.

Hàm loss sử dụng L1

$$\mathcal{J}_{L1}(w; X, y) = \mathcal{J}(w; X, y) + \lambda \sum_i |\omega_i|$$

L2 regularization: Cộng thêm vào hàm loss tổng giá trị bình phương của trọng số.

Hàm loss sử dụng L2

$$\mathcal{J}_{L2}(w; X, y) = \mathcal{J}(w; X, y) + \lambda \sum_i \omega_i^2$$

Ảnh hưởng của hệ số λ .

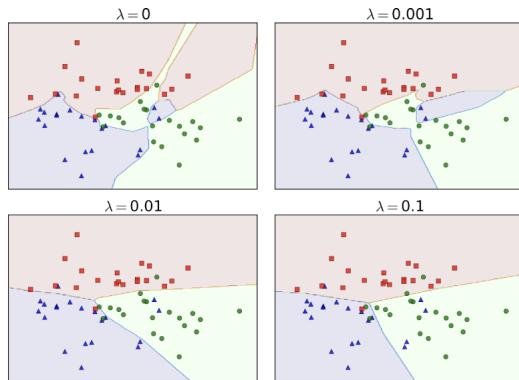


Figure 5: Ảnh hưởng của λ lên kết quả training

Elastic Net regularization: kết hợp cả L2 và L1 regularization.

Hàm loss kết hợp L1 và L2

$$\mathcal{J}_{L12}(w; X, y) = \mathcal{J}_{L12}(w; X, y) + \lambda_1 \sum_i |\omega_i| + \lambda_2 \sum_i \omega_i^2$$

Gradient descent optimization algorithms: Momentum

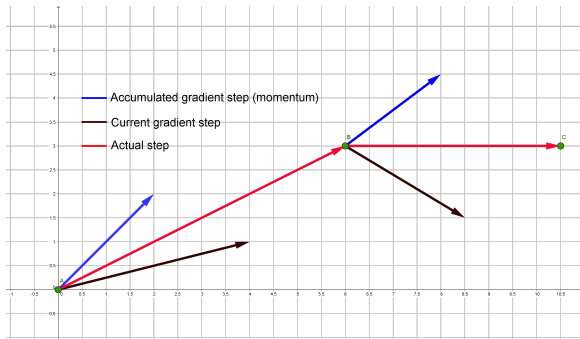


Figure 6: Nguyên lý hoạt động của Momentum

Xét hàm thế năng (potential function):

$$U = mgh \rightarrow U \propto h \quad (1)$$

Gọi F là lực tác động lên viên bi

$$F = -\nabla U = ma \quad (2)$$

Biểu thức luật cập nhật

$$v^{(k+1)} = \beta v^{(k)} - \alpha \nabla f(x^{(k)}) \quad (3)$$

$$x^{(k+1)} = x^{(k)} + v^{(k+1)} \quad (4)$$

Gradient descent optimization algorithms: Nesterov Momentum

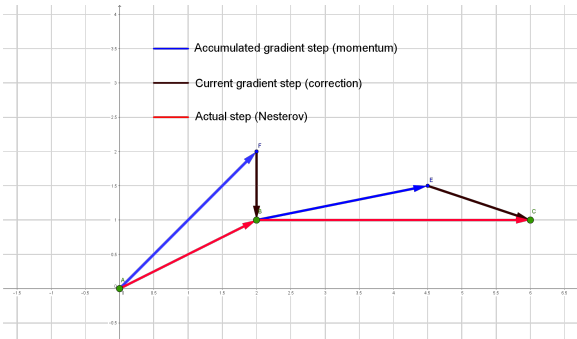


Figure 7: Nguyên lý hoạt động của Nesterov Momentum

Ý tưởng: tính toán đạo hàm tại điểm ở vị trí tương lai $x^{(k)} + \beta v^{(k)}$ - overhead, tức là một điểm có thể gần với điểm tối ưu nhất.

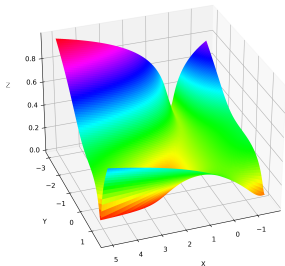
Biểu thức luật cập nhật

$$v^{(k+1)} = \beta v^{(k)} - \alpha \nabla f(x^{(k)} + \beta v^{(k)}) \quad (5)$$

$$x^{(k+1)} = x^{(k)} + v^{(k+1)} \quad (6)$$

Thực nghiệm, phân tích kết quả và đánh giá

Đánh giá khả năng tối ưu hàm mục tiêu Beale



Hàm beale

$$f(x, y) = 1 + (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^2)^2 \quad (7)$$

Không gian tìm kiếm: $-1.5 \leq x \leq 5$ và $-3 \leq y \leq 1.5$

Cực tiểu toàn cục: $f(3, 0.5) = 0$

Figure 8: Trực quan hóa hàm Beale 3D.

Đánh giá khả năng tối ưu hàm mục tiêu Beale

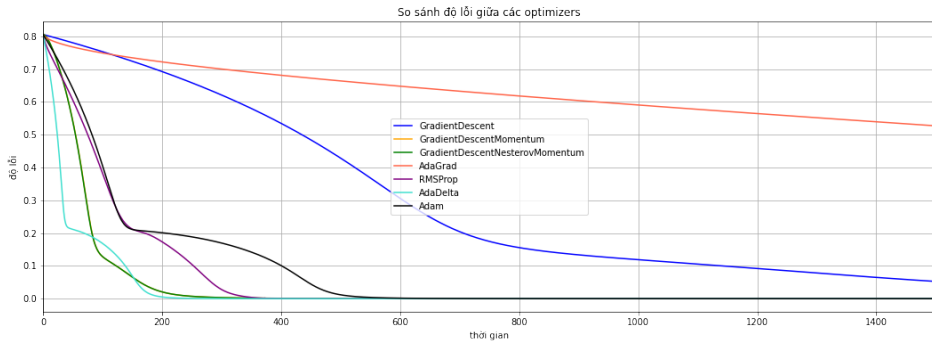


Figure 9: So sánh độ lỗi giữa các bộ tối ưu.

Đánh giá khả năng tối ưu hàm mục tiêu Beale

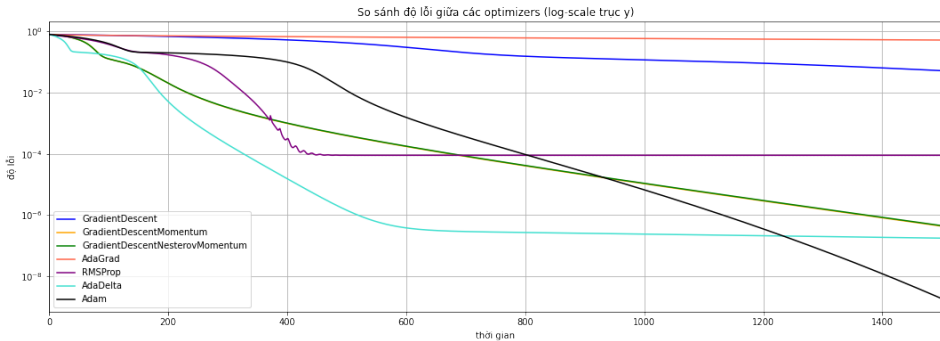


Figure 10: So sánh độ lỗi giữa các bộ tối ưu với log-scale trục y.

Đánh giá khả năng tối ưu hàm mục tiêu Beale

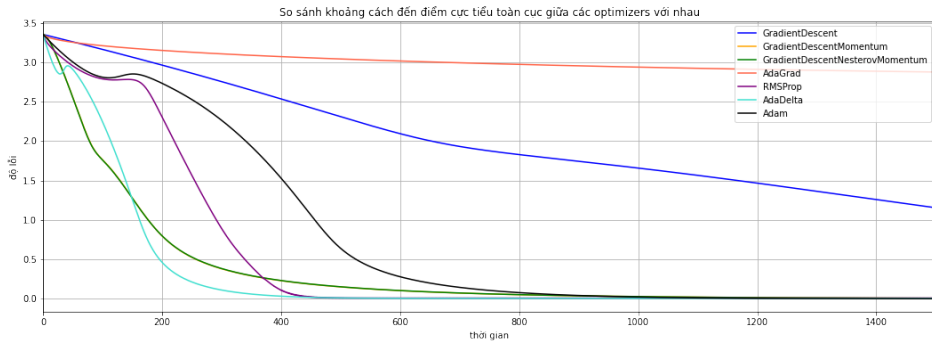


Figure 11: So sánh khoảng cách đến cực tiểu toàn cục giữa các bộ tối ưu.

Dữ liệu thực nghiệm MNIST



Figure 12: Minh họa hình ảnh từ bộ dữ liệu.

Thực nghiệm với mô hình Multilayer Perceptron

Optimizer	Validation Accuracy	Test Accuracy
Gradient Descent	0.95573	0.95830
Momentum	0.97906	0.97840
Nesterov Momentum	0.97896	0.97720
AdaGrad	0.97720	0.95120
AdaDelta	0.78698	0.79380
RMSProp	0.98021	0.97870
Adam	0.97823	0.98090

Table 1: Độ chính xác trên tập xác nhận và tập kiểm tra với các bộ tối ưu khác nhau đối với mô hình Multilayer Perceptron.

Thực nghiệm với mô hình Multilayer Perceptron

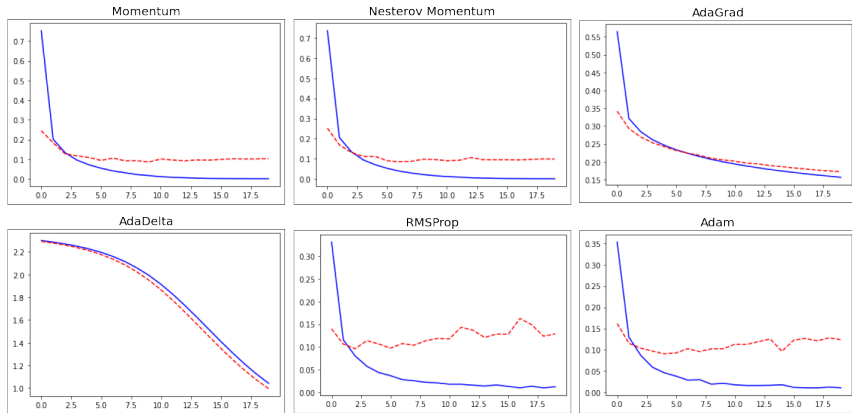


Figure 13: Kết quả thực nghiệm với mô hình MLP. Đường màu xanh thể hiện độ lỗi trên tập train, đường màu đỏ đứt gãy thể hiện độ lỗi tập test.

Thực nghiệm với mô hình Convolutional Neural Networks đơn giản

Optimizer	Validation Accuracy	Test Accuracy
Gradient Descent	0.98510	0.98520
Momentum	0.98729	0.98930
Nesterov Momentum	0.98677	0.98690
AdaGrad	0.97646	0.97980
AdaDelta	0.90823	0.91770
RMSProp	0.98802	0.98860
Adam	0.98594	0.98590

Table 2: Độ chính xác trên tập xác nhận và tập kiểm tra với các bộ tối ưu khác nhau đối với mô hình Convolutional Neural Networks 2 tầng.

Thực nghiệm với mô hình Convolutional Neural Networks đơn giản

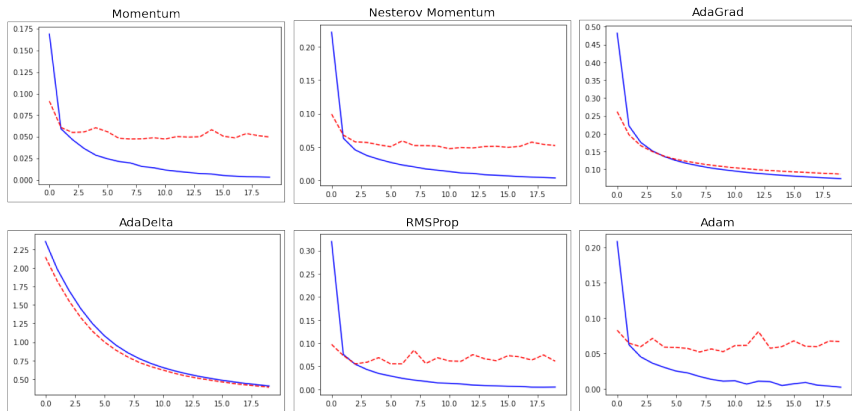


Figure 14: Kết quả thực nghiệm với mô hình CNN. Đường màu xanh thể hiện độ lỗi trên tập train, đường màu đỏ đứt gãy thể hiện độ lỗi tập test.

Thực nghiệm với mô hình Deep Residual Networks

Optimizer	Validation Accuracy	Test Accuracy
Gradient Descent	0.98604	0.98770
Momentum	0.99219	0.99200
Nesterov Momentum	0.99167	0.99340
AdaGrad	0.98656	0.98580
AdaDelta	0.96667	0.96760
RMSProp	0.99115	0.99190
Adam	0.99177	0.99170

Table 3: Độ chính xác trên tập xác nhận và tập kiểm tra với các bộ tối ưu khác nhau đối với mô hình ResNet18.

Thực nghiệm với mô hình Deep Residual Networks

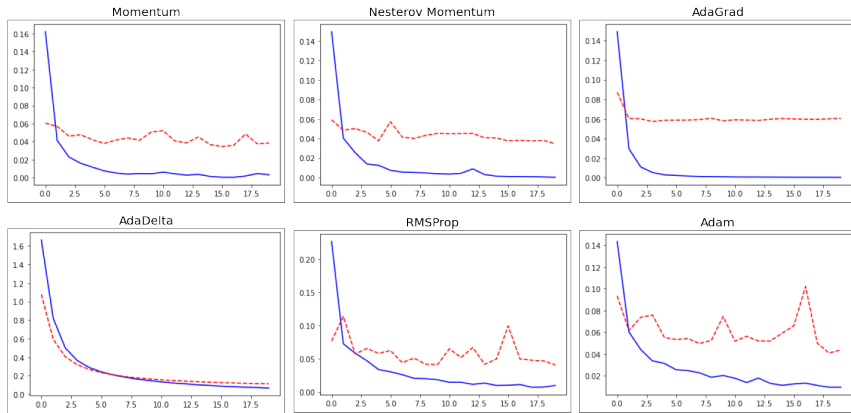


Figure 15: Kết quả thực nghiệm với mô hình ResNet-18. Đường màu xanh thể hiện độ lỗi trên tập train, đường màu đỏ đứt gãy thể hiện độ lỗi tập test.

Optimizer	Validation Accuracy	Test Accuracy
Gradient Descent	0.99365	0.99420
Momentum	0.98833	0.98900
Nesterov Momentum	0.98469	0.98700
AdaGrad	0.99063	0.99310
AdaDelta	0.96667	0.96760
RMSProp	0.99000	0.99030
Adam	0.98948	0.99070

Table 4: Độ chính xác trên tập xác nhận và tập kiểm tra với các bộ tối ưu khác nhau đối với mô hình VGG16.

Thực nghiệm với mô hình VGG

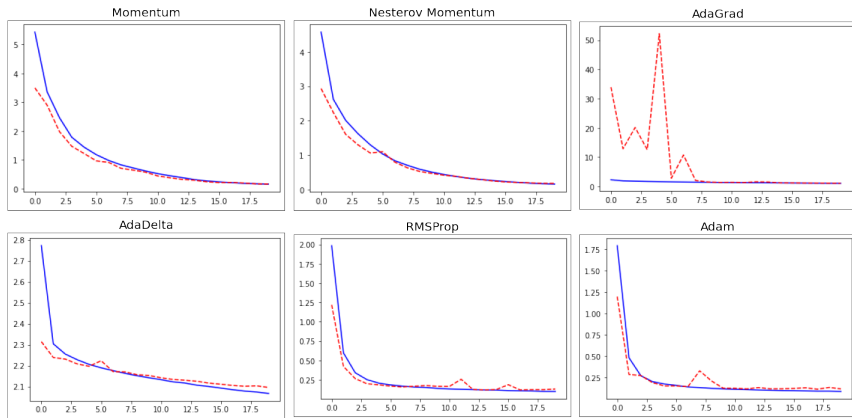


Figure 16: Kết quả thực nghiệm với mô hình VGG16. Đường màu xanh thể hiện độ lỗi trên tập train, đường màu đỏ đứt gãy thể hiện độ lỗi tập test.

Thực nghiệm Regularization với CNN

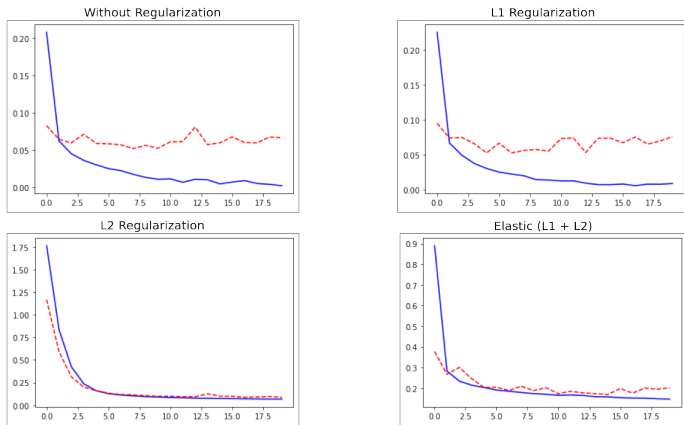





Figure 17: Kết quả thực nghiệm với mô hình CNN với các phương pháp regularization. Đường màu xanh thể hiện độ lỗi trên tập train, đường màu đỏ đứt gãy thể hiện độ lỗi tập test.

Kết luận

- Các phương pháp tối ưu local descent, mà cụ thể là các thuật toán dựa trên Gradient Descent được áp dụng phổ biến trong tối ưu các mô hình mạng học sâu và được cài đặt trong nhiều thư viện như TensorFlow, PyTorch.
- Gradient Descent là một phương pháp tối ưu hàm mục tiêu được tham số hóa bởi các tham số của mô hình học, thuật toán sử dụng cập nhật ngược hướng gradient của hàm này để hy vọng đạt tới cực tiểu toàn cục. Các thuật toán cải tiến với Momentum và Nesterov đem lại hiệu quả với sự điều chỉnh hướng dịch chuyển tiếp theo tốt hơn.
- Hơn nữa, những thuật toán cải tiến gần đây như AdaGrad, RMSProp, Adam ... sử dụng các kỹ thuật cụ thể để mà tăng tốc trong việc chạm tới điểm cực tiểu (vượt qua cực tiểu toàn cục, các điểm yên ngựa).
- Các phương pháp chính quy hóa sử dụng một số ràng buộc chính quy cho thuật toán học, ta có thể giúp nó ít nhạy cảm với dữ liệu hơn và ổn định quá trình huấn luyện mô hình. Tuy có thể gây ra giảm độ lỗi tổng quát và có thể làm tăng độ lỗi trong quá trình huấn luyện, nhưng mô hình ổn định hơn (tăng bias và giảm phương sai).

-  Kochenderfer, Mykel J and Tim A Wheeler (2019). *Algorithms for optimization*. Mit Press.
-  LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.
-  Luenberger, David G, Yinyu Ye, et al. (1984). *Linear and nonlinear programming*. Vol. 2. Springer.

Phụ lục

Đánh giá tối ưu hàm mục tiêu: Momentum

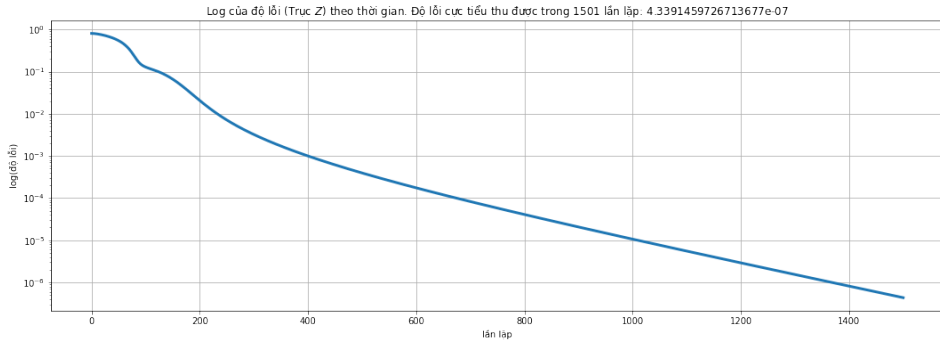


Figure 18: Thuật toán Gradient Descent Momentum: Logarit độ lỗi theo trục z theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Momentum

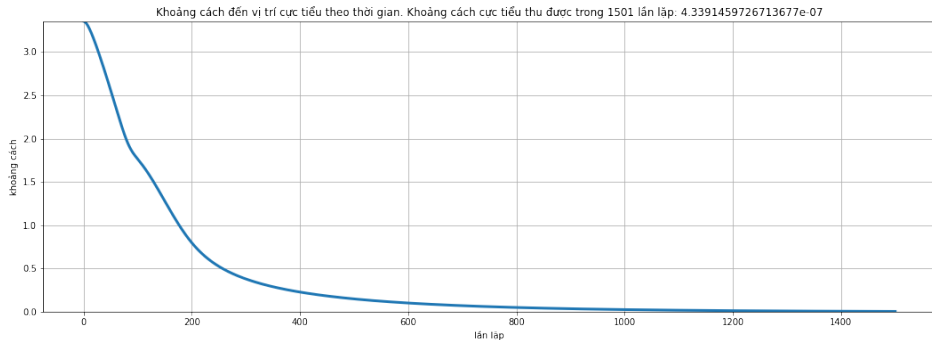


Figure 19: Thuật toán Gradient Descent Momentum: Khoảng cách đến vị trí cực tiểu toàn cục theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Momentum

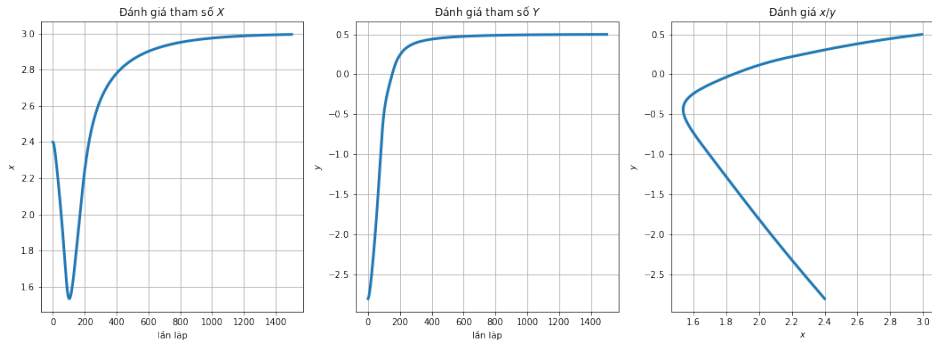


Figure 20: Thuật toán Gradient Descent Momentum: Đánh giá tham số X , Y và tỷ lệ X/Y theo thời gian

Đánh giá tối ưu hàm mục tiêu: Momentum

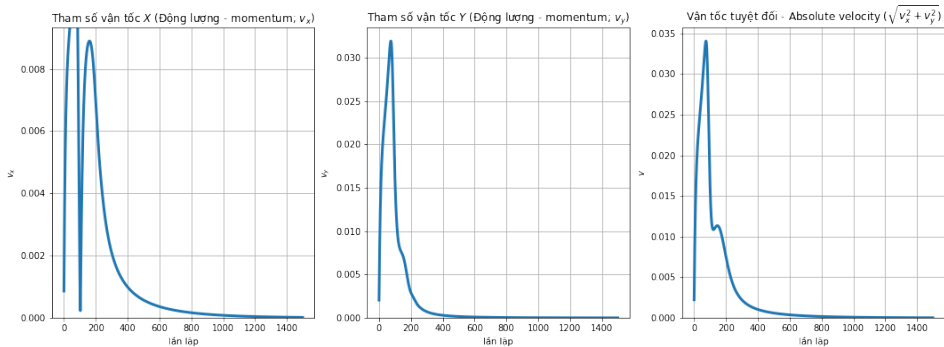


Figure 21: Thuật toán Gradient Descent Momentum: vận tốc theo phương x , y và vận tốc tuyệt đối

Đánh giá tối ưu hàm mục tiêu: Nesterov Momentum

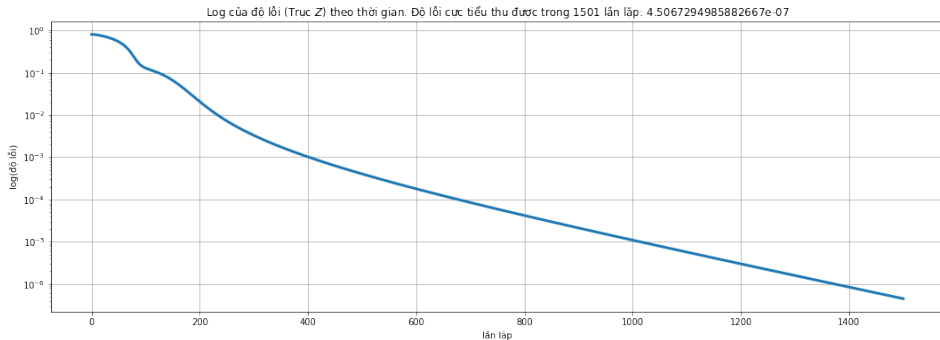


Figure 22: Thuật toán Gradient Descent Nesterov Momentum: Logarit độ lỗi theo trục z theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Nesterov Momentum

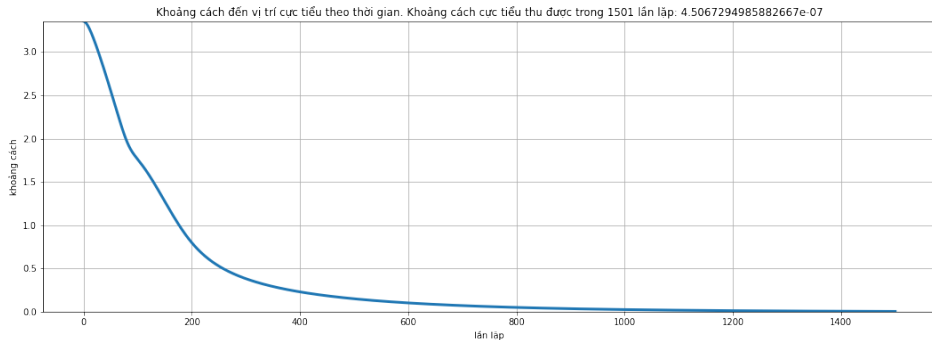


Figure 23: Thuật toán Gradient Descent Nesterov Momentum: Khoảng cách đến vị trí cực tiểu toàn cục theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Nesterov Momentum

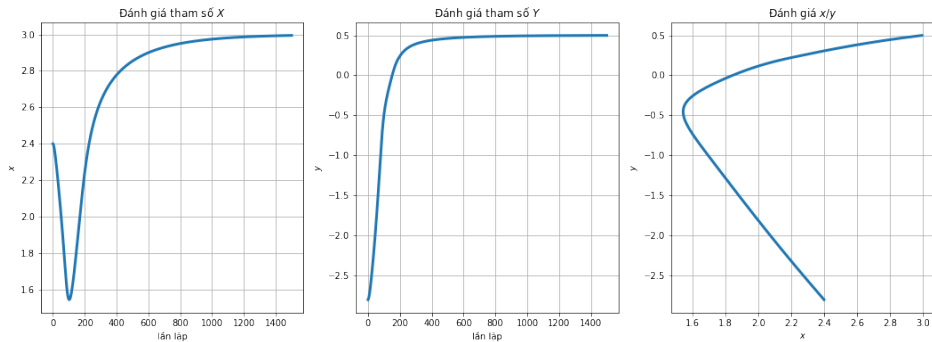


Figure 24: Thuật toán Gradient Descent Nesterov Momentum: Đánh giá tham số X , Y và tỷ lệ X/Y theo thời gian

Đánh giá tối ưu hàm mục tiêu: Nesterov Momentum

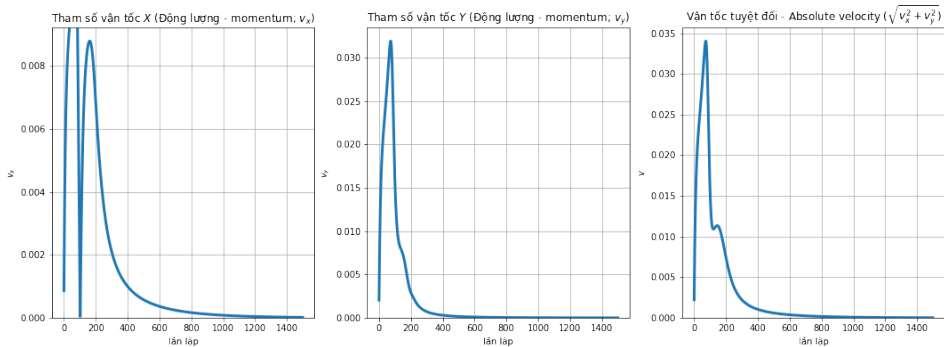


Figure 25: Thuật toán Gradient Descent Nesterov Momentum: vận tốc theo phương x, y và vận tốc tuyệt đối

Đánh giá tối ưu hàm mục tiêu: AdaGrad

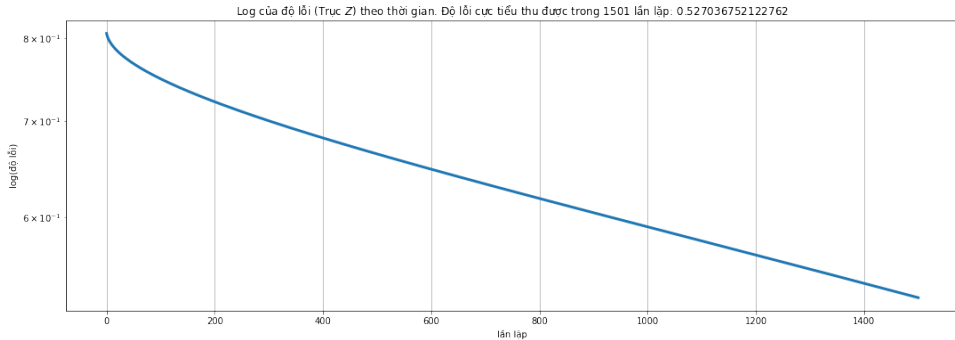


Figure 26: Thuật toán AdaGrad: Logarit độ lỗi trục z theo thời gian.

Đánh giá tối ưu hàm mục tiêu: AdaGrad

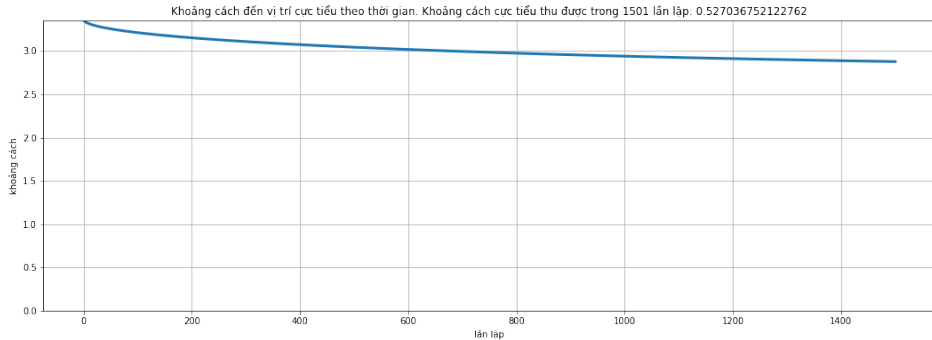


Figure 27: Thuật toán AdaGrad: Khoảng cách đến vị trí cực tiểu toàn cục theo thời gian.

Đánh giá tối ưu hàm mục tiêu: AdaGrad

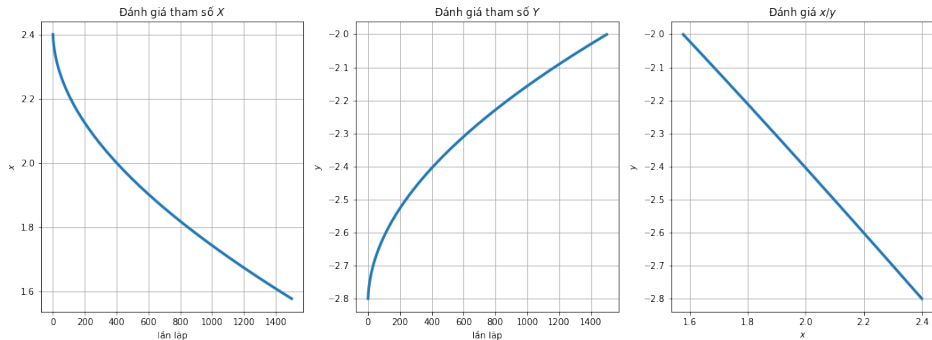


Figure 28: Thuật toán AdaGrad: Đánh giá tham số X , Y và tỷ lệ X/Y theo thời gian

Đánh giá tối ưu hàm mục tiêu: AdaGrad

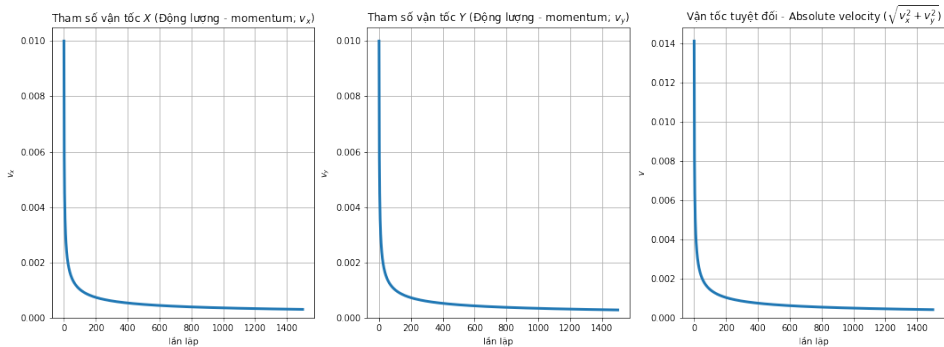


Figure 29: Thuật toán AdaGrad: vận tốc theo phương x , y và vận tốc tuyệt đối

Đánh giá tối ưu hàm mục tiêu: AdaDelta

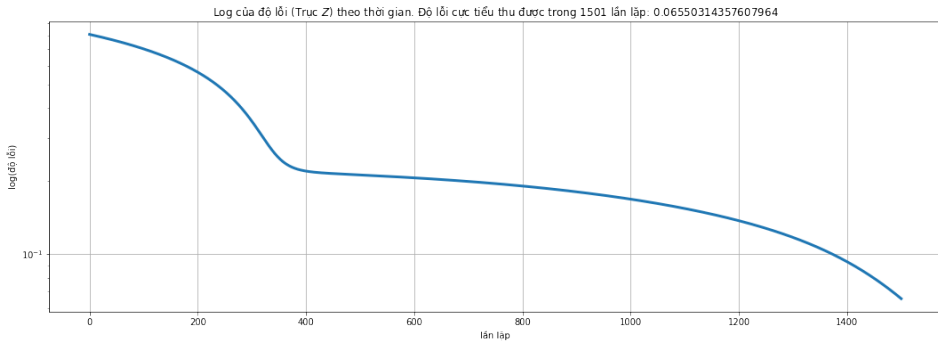


Figure 30: Thuật toán AdaDelta: Logarit độ lỗi trục z theo thời gian.

Đánh giá tối ưu hàm mục tiêu: AdaDelta

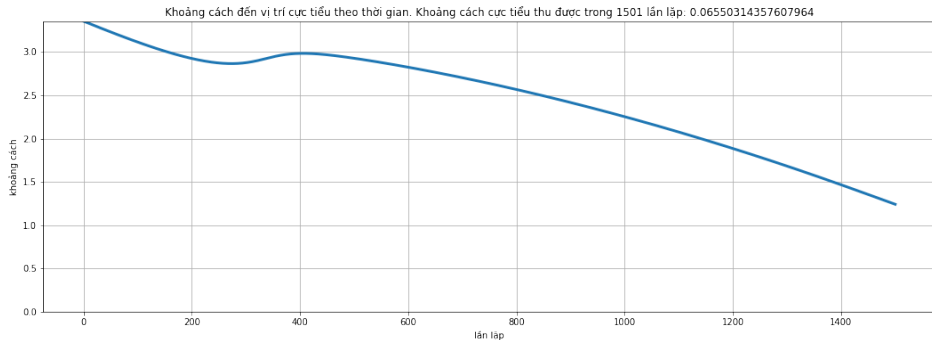


Figure 31: Thuật toán AdaDelta: Khoảng cách đến vị trí cực tiểu toàn cục theo thời gian.

Đánh giá tối ưu hàm mục tiêu: AdaDelta

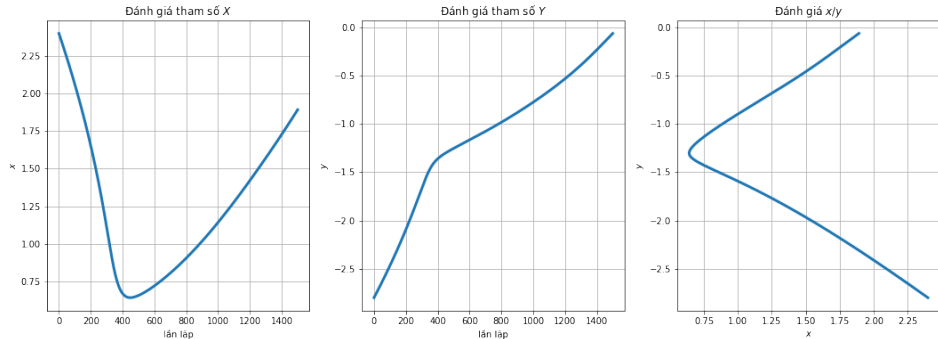


Figure 32: Thuật toán AdaDelta: Đánh giá tham số X , Y và tỷ lệ X/Y theo thời gian

Đánh giá tối ưu hàm mục tiêu: AdaDelta

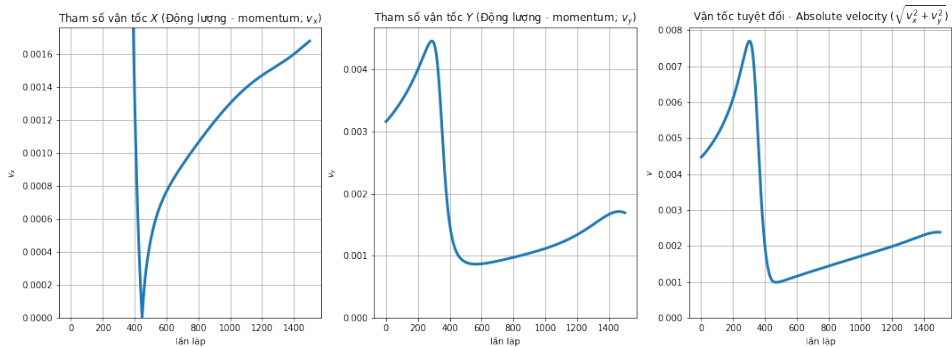


Figure 33: Thuật toán AdaDelta: vận tốc theo phương x , y và vận tốc tuyệt đối

Đánh giá tối ưu hàm mục tiêu: RMSPProp

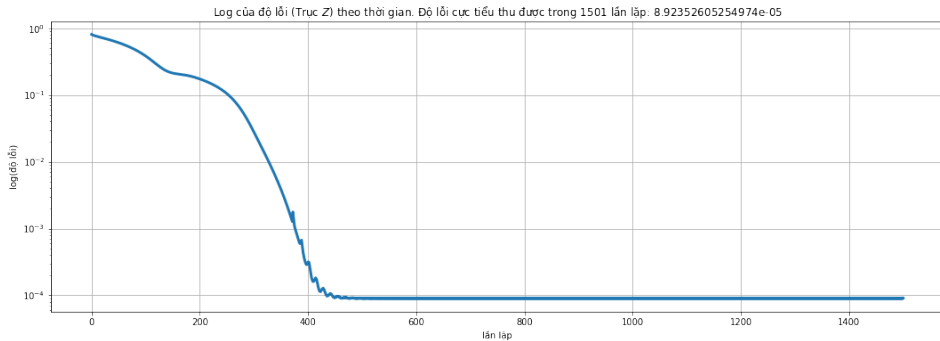


Figure 34: Thuật toán RMSPProp: Logarit độ lỗi theo trục z theo thời gian.

Đánh giá tối ưu hàm mục tiêu: RMSProp

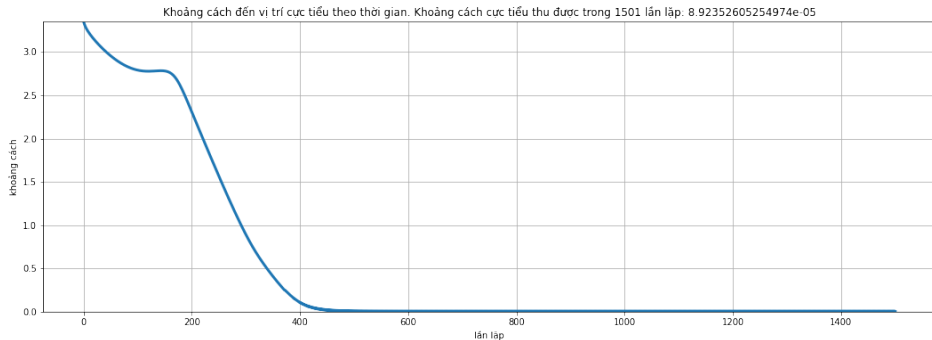


Figure 35: Thuật toán RMSProp: Khoảng cách đến vị trí cực tiểu toàn cục theo thời gian.

Đánh giá tối ưu hàm mục tiêu: RMSProp

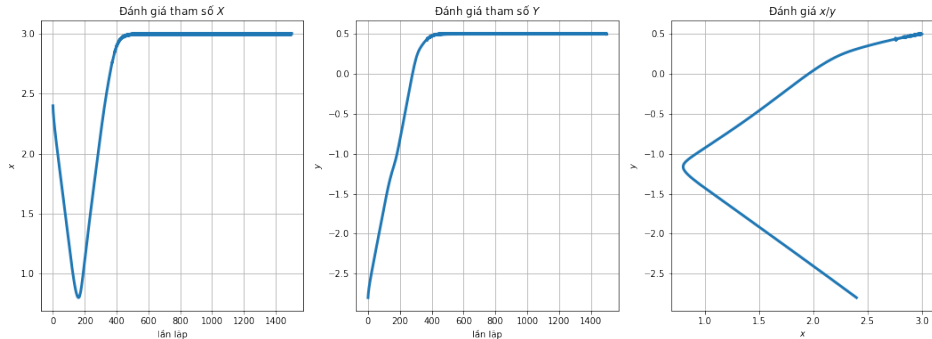


Figure 36: Thuật toán RMSProp: Đánh giá tham số X , Y và tỷ lệ X/Y theo thời gian

Đánh giá tối ưu hàm mục tiêu: RMSProp

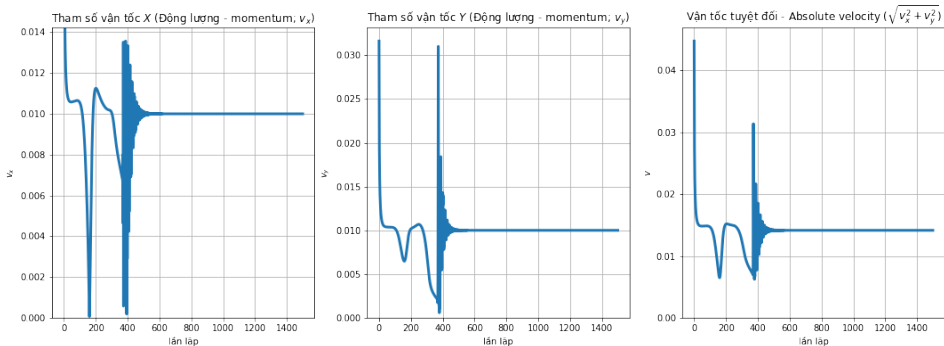


Figure 37: Thuật toán RMSProp: vận tốc theo phương x, y và vận tốc tuyệt đối

Đánh giá tối ưu hàm mục tiêu: Adam

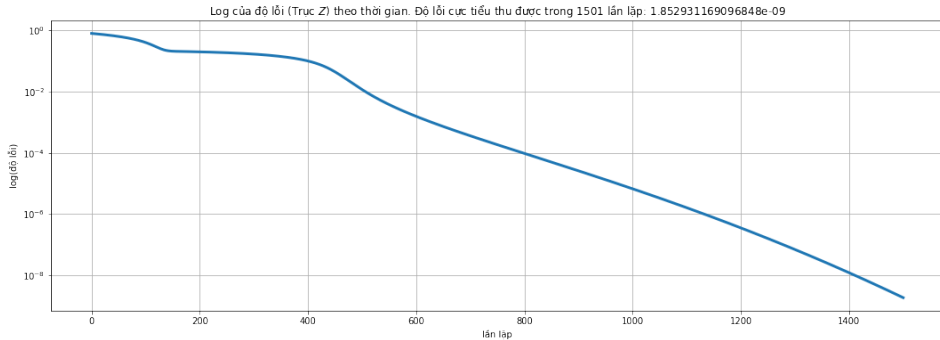


Figure 38: Thuật toán Adam: Logarit độ lỗi theo trục z theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Adam

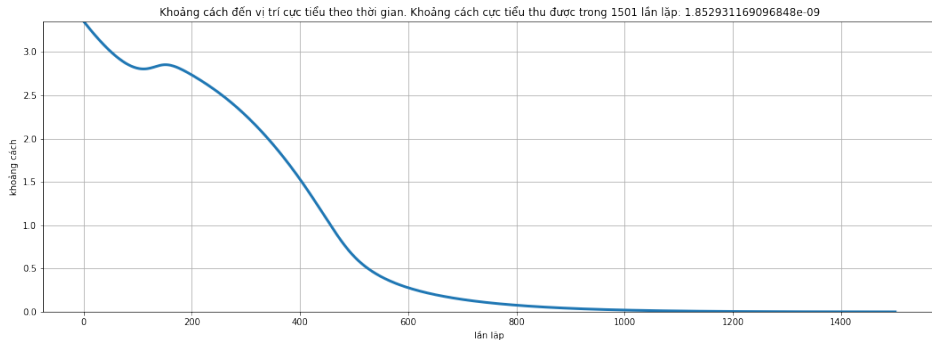


Figure 39: Thuật toán Adam: Khoảng cách đến vị trí cực tiểu toàn cục theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Adam

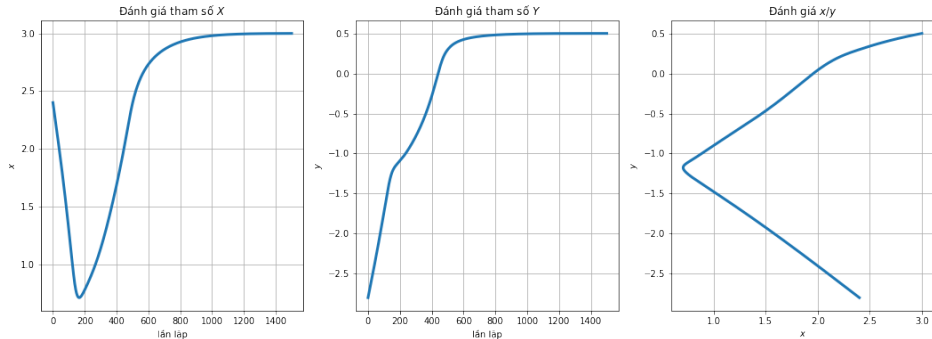


Figure 40: Thuật toán Adam: Đánh giá tham số X , Y và tỷ lệ X/Y theo thời gian.

Đánh giá tối ưu hàm mục tiêu: Adam

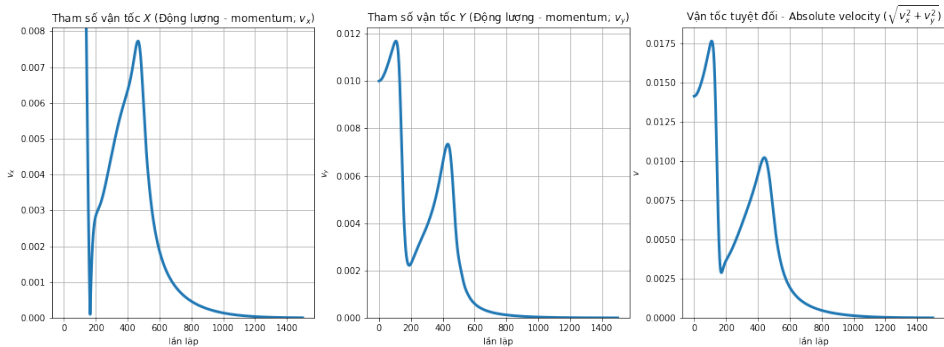


Figure 41: Thuật toán Adam: vận tốc theo phương x , y và vận tốc tuyệt đối.

Một số phương pháp tối ưu gần đây: AMSGrad

Thuật toán AMSGrad được tham số hóa bởi learning rate η , discount factors $\gamma_v < 1$, $\gamma_s < 1$

1) biased decaying momentum

$$\mathbf{v}^{(k+1)} = \gamma_v \mathbf{v}^{(k)} + (1 - \gamma_v) \mathbf{g}^{(k)} \quad (8)$$

2) corrected decaying momentum

$$\hat{\mathbf{v}}^{(k+1)} = \max(\hat{\mathbf{v}}^{(k)}, \mathbf{v}^{(k)}) \quad (9)$$

3) biased decaying sq. gradient

$$\mathbf{s}^{(k+1)} = \gamma_s \mathbf{s}^{(k)} + (1 - \gamma_s) \left(\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)} \right) \quad (10)$$

4) corrected decaying sq. gradient

$$\hat{\mathbf{s}}^{(k+1)} = \mathbf{s}^{(k+1)} / (1 - \gamma_s^k) \quad (11)$$

5) final update

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \hat{\mathbf{v}}^{(k+1)} / \left(\epsilon + \sqrt{\hat{\mathbf{s}}^{(k+1)}} \right) \quad (12)$$

Một số phương pháp tối ưu gần đây: AdamW

AdamW sửa đổi việc thực hiện điển hình của sự phân rã trọng số trong Adam, bằng cách tách rời mức phân rã trọng số từ bản cập nhật gradient.

$$\mathbf{g}^{(k)} = \nabla f(x^{(k)}) + w_t x^{(k)} \quad (13)$$

Thuật toán AdamW cơ bản giống với thuật toán Adam

Một số phương pháp tối ưu gần đây: Quasi-Hyperbolic Adam

1) Biased decaying momentum

$$\mathbf{v}^{(k+1)} = \gamma_v \mathbf{v}^{(k)} + (1 - \gamma_v) \mathbf{g}^{(k)} \quad (14)$$

2) Biased decaying sq. gradient

$$\mathbf{s}^{(k+1)} = \gamma_s \mathbf{s}^{(k)} + (1 - \gamma_s) \left(\mathbf{g}^{(k)} \odot \mathbf{g}^{(k)} \right) \quad (15)$$

3) Corrected decaying momentum

$$\hat{\mathbf{v}}^{(k+1)} = \mathbf{v}^{(k+1)} / (1 - \gamma_v^k) \quad (16)$$

4) Corrected decaying sq. gradient

$$\hat{\mathbf{s}}^{(k+1)} = \mathbf{s}^{(k+1)} / (1 - \gamma_s^k) \quad (17)$$

5) Update rule

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \left[\frac{(1 - \nu_1) \cdot \mathbf{g}^{(k)} + \nu_1 \cdot \hat{\mathbf{v}}^{(k+1)}}{\sqrt{(1 - \nu_2)(\mathbf{g}^{(k)})^2 + \nu_2 \hat{\mathbf{s}}^{(k+1)} + \epsilon}} \right] \quad (18)$$

Một số phương pháp tối ưu gần đây: AggMo

AggMo (Aggregated Momentum) là một thuật toán momentum thích ứng và giải quyết vấn đề trên bằng cách sử dụng một tổ hợp tuyến tính của nhiều momentum buffers. Nó duy trì K momentum buffers, mỗi chúng có một discount factor khác nhau và tính trung bình của chúng để mà cập nhật. Với K momentum buffers, có các discount factors $\beta \in \mathbb{R}^K$

$$(v^{(k+1)})^{(i)} = (\beta)^{(i)}(v^{(k)})^{(i)} + g^{(k)} \text{ (integrate velocity)}, \forall i \in [1, K] \quad (19)$$

$$x^{(k+1)} = x^{(k)} - \alpha \left[\frac{1}{K} \cdot \sum_{i=1}^K (v^{(k+1)})^{(i)} \right] \text{ (integrate position)} \quad (20)$$

Một số phương pháp tối ưu gần đây: Quasi-Hyperbolic Momentum

QHM (Quasi-Hyperbolic Momentum) là một thuật toán momentum thích ứng khác mà tách momentum ra khỏi gradient hiện tại khi cập nhật trọng số.

$$\mathbf{v}^{(k+1)} = \beta \mathbf{v}^{(k)} + (1 - \beta) \cdot \mathbf{g}^{(k)} \quad (21)$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \left[(1 - \nu) \cdot \mathbf{g}^{(k)} + \nu \cdot \mathbf{v}^{(k+1)} \right] \quad (22)$$

trong đó các hệ số $\nu = 0.7$ và $\beta = 0.999$

Một số phương pháp tối ưu gần đây: Demon

Demon (Decaying Momentum) là một luật phân rã động lượng (decaying momentum rule) được lấy ý tưởng từ mô hình học tuyến tính phân rã tốc độ học (linear learning rate decay models) mà giảm ảnh hưởng của một gradient đến các cập nhật thời điểm hiện tại và kế tiếp.

$$\sum_{i=1}^{\infty} \beta^i = \beta \sum_{i=0}^{\infty} \beta^i = \frac{\beta}{1 - \beta} \quad (23)$$

Phân rã tổng này trong thuật toán Demon. Ta gọi β_{init} là giá trị khởi tạo, β là giá trị hiện thời tại bước nhảy thứ t trong tổng cộng T bước nhảy, quy trình phân rã được đưa ra bằng cách giải nghiệm cho β_t

$$\frac{\beta_t}{(1 - \beta_t)} = \left(1 - \frac{t}{T}\right) \frac{\beta_{\text{init}}}{(1 - \beta_{\text{init}})} \quad (24)$$

Các phương pháp tối ưu toàn phương: Newton's method

Xem xét bài toán tối ưu hàm đơn biến, xấp xỉ bậc hai (quadratic approximation) tại một điểm $x^{(k)}$ có thể thu được bằng cách sử dụng khai triển Taylor bậc hai:

$$q(x) = f(x^{(k)}) + (x - x^{(k)}) f'(x^{(k)}) + \frac{(x - x^{(k)})^2}{2} f''(x^{(k)}) \quad (25)$$

Cho đạo hàm bằng 0 và giải nó, thu được biểu thức cập nhật như sau:

$$\frac{\partial}{\partial x} q(x) = f'(x^{(k)}) + (x - x^{(k)}) f''(x^{(k)}) = 0 \quad (26)$$

$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})} \quad (27)$$

Các phương pháp tối ưu toàn phương: Newton's method

Xem xét bài toán tối ưu hàm đa biến, xấp xỉ bậc hai (quadratic approximation) tại một điểm $x^{(k)}$ có thể thu được bằng cách sử dụng khai triển Taylor bậc hai:

$$f(x) \approx q(x) = f(x^{(k)}) + (g^{(k)})^\top (x - x^{(k)}) + \frac{1}{2} (x - x^{(k)})^\top H^{(k)} (x - x^{(k)}) \quad (28)$$

Cho đạo hàm bằng 0 và giải nghiệm

$$\nabla q(x^{(k)}) = g^{(k)} + H^{(k)} (x - x^{(k)}) = 0 \quad (29)$$

$$x^{(k+1)} = x^{(k)} - (H^{(k)})^{-1} g^{(k)} \quad (30)$$

Các phương pháp tối ưu toàn phương: Secant method

Phương pháp secant sử dụng hai lần lặp cuối cùng để xấp xỉ đạo hàm bậc hai như sau:

$$f''(x^{(k)}) \approx \frac{f'(x^{(k)}) - f'(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad (31)$$

Áp dụng đưa vào biểu thức của phương pháp Newton, ta được biểu thức ước lượng như sau:

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f'(x^{(k)}) - f'(x^{(k-1)})} f'(x^{(k)}) \quad (32)$$

Các phương pháp tối ưu toàn phương: Quasi-Newton method

Giống như cách xấp xỉ đạo hàm bậc hai f'' trong phương pháp secant khi xem xét bài toán tối ưu hàm đơn biến, quasi-Newton xấp xỉ ma trận nghịch đảo Hessian. Biểu thức cập nhật của Quasi-Newton có dạng:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha^{(k)} \mathbf{Q}^{(k)} \mathbf{g}^{(k)} \quad (33)$$

trong đó $\alpha^{(k)}$ là một hệ số cho bước nhảy (step factor) và $\mathbf{Q}^{(k)}$ xấp xỉ nghịch đảo của ma trận Hessian tại $\mathbf{x}^{(k)}$

Làm sao xấp xỉ $\mathbf{Q}^{(k)}$: Davidon-Fletcher-Powell (DFP) method;
Broyden-Fletcher-Goldfarb-Shanno (BFGS) method

- 1) Multilayer Peceptron
- 2) Convolutional Neural Networks
- 3) Deep Residual Networks (ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152)
- 4) Very Deep Convolutional Neural Networks (VGG11, VGG13, VGG16, VGG19)
- 5) Generative Adversarial Networks
- 6) Variational Auto-Encoder
- 7) L1 Regularization with PyTorch
- 8) L2 Regularization with PyTorch
- 9) Elastic (L1 + L2) Regularization
- 10) Cài đặt Batch gradient descent, Stochastic gradient descent và Mini-Batch gradient descent với Numpy

Cảm thầy cô, anh chị và các bạn đã chú ý lắng nghe.