Ghi chép bài giảng: Stanford CS231n

Lecture 3: Regularization and Optimization

Ho Hong Phuc Nguyen

28 tháng 1, 2026

Mục lục

1 Regularization (Sự chuẩn hóa)

Mục tiêu của Regularization là giúp mô hình tránh hiện tượng **Overfitting** (quá khớp). Trong huấn luyện, chúng ta không chỉ tối ưu hóa việc dự đoán đúng dữ liệu huấn luyện (Data Loss) mà còn phải kiểm soát độ phức tạp của mô hình (Regularization Loss).

Hàm mất mát tổng quát:

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \tag{1}$$

Trong đó $R(W)$ là hàm chuẩn hóa và λ là siêu tham số (hyperparameter) điều chỉnh cường độ chuẩn hóa.

1.1 Các loại chuẩn hóa phổ biến

- **L2 Regularization (Weight Decay):** $R(W) = \sum_k \sum_l W_{k,l}^2$. Ưu tiên các trọng số nhỏ và phân bổ đều. Nó ngăn chặn việc bất kỳ một đặc trưng nào chiếm ưu thế quá lớn.
- **L1 Regularization:** $R(W) = \sum_k \sum_l |W_{k,l}|$. Có xu hướng đẩy các trọng số không quan trọng về đúng bằng 0, tạo ra các ma trận thưa (**Sparsity**).
- **Elastic Net:** Kết hợp cả hai: $R(W) = \sum_k \sum_l \left(\beta W_{k,l}^2 + |W_{k,l}|\right)$.

1.2 Khi nào sử dụng L1, L2 hay Elastic?

- **L2:** Sử dụng mặc định trong hầu hết các trường hợp để giữ mô hình ổn định và tránh phụ thuộc vào các pixel nhiễu.
- **L1:** Sử dụng khi bạn cần trích xuất đặc trưng (feature selection), muốn biết yếu tố nào thực sự quan trọng.
- **Elastic Net:** Hữu ích khi có nhiều đặc trưng tương quan (correlated) với nhau; L1 sẽ chọn ngẫu nhiên 1 cái, còn Elastic Net sẽ giữ lại cả nhóm một cách ổn định hơn.

2 Optimization (Tối ưu hóa)

Làm thế nào để tìm W sao cho L nhỏ nhất? Đó là bài toán tìm điểm thấp nhất trong một "cảnh quan"(landscape) mất mát phức tạp.

2.1 Gradient Descent và Chain Rule

- **Gradient Descent:** Cập nhật trọng số ngược hướng với đạo hàm: $W_{new} = W_{old} - \eta \nabla_W L$.
- **Chain Rule (Quy tắc xích):** Công cụ để tính đạo hàm trong các mô hình nhiều lớp. Nếu z phụ thuộc y, y phụ thuộc x, thì đạo hàm của z theo x là: $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$.

2.2 Stochastic Gradient Descent (SGD)

Thay vì tính gradient cho hàng triệu tấm ảnh (Full-batch), ta chỉ tính trên một nhóm nhỏ gọi là **Mini-batch** (thường từ 32 đến 256 ảnh). Điều này giúp tốc độ tính toán nhanh hơn và mang lại tính ngẫu nhiên giúp thoát khỏi các cực tiểu cục bộ.

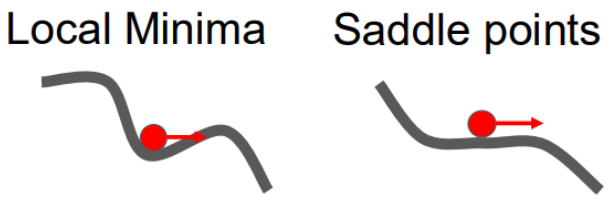
3 Các thuật toán tối ưu nâng cao

3.1 SGD + Momentum

Giải quyết vấn đề SGD bị dao động mạnh ở thung lũng hẹp hoặc kẹt ở điểm yên ngựa (saddle point).

- **SGD + Momentum (Cục đá lăn xuống đồi):** Hãy tưởng tượng một cục đá đang lăn từ trên đỉnh đồi xuống. Khi lăn, nó không chỉ di chuyển dựa trên lực đẩy hiện tại mà còn tích lũy **động năng (momentum)**.

– Cục đá sẽ càng lúc càng lăn nhanh hơn nếu nó tiếp tục đi đúng hướng xuống dốc.
– Nhờ vận tốc tích lũy, nó có đủ đà để vượt qua những cái hố nhỏ (cực tiểu cục bộ - local minima) hoặc lướt nhanh qua những đoạn bằng phẳng (điểm yên ngựa - saddle point).



Momentum tích lũy "vận tốc"từ các bước trước đó:

- $v_{t+1} = \rho v_t + \nabla L(W_t)$
- $W_{t+1} = W_t - \eta v_{t+1}$

3.2 RMSprop (Root Mean Square Propagation)

Được đề xuất bởi Geoffrey Hinton, RMSprop giải quyết vấn đề tốc độ học bị giảm quá nhanh trong các thuật toán cũ (như Adagrad) bằng cách sử dụng trung bình trượt bình phương của các gradient.

- **RMSprop (Điều tiết theo độ dốc):** Thuật toán này đóng vai trò như một bộ điều chỉnh thông minh dựa trên địa hình:
  – **Độ dốc cao (Steep):** Nếu địa hình quá dốc và hiểm trở (gradient lớn), RMSprop sẽ chủ động "phanh"lại bằng cách giảm tốc độ học, tránh việc mô hình bị nhảy quá đà và văng ra khỏi thung lũng.
  – **Độ dốc thoải (Flat):** Nếu địa hình bằng phẳng và ít dốc (gradient nhỏ), RMSprop sẽ tự động "tăng ga"để mô hình bước những bước dài hơn, giúp quá trình hội tụ không bị dậm chân tại chỗ.
- **Bước 1: Tích lũy bình phương gradient (với decay rate α):**

$$s_{t+1} = \alpha s_t + (1-\alpha)\nabla L(W_t)^2 \tag{2}$$

- **Bước 2: Cập nhật trọng số:**

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{s_{t+1}+\epsilon}}\nabla L(W_t) \tag{3}$$

Trong đó ϵ là một số cực nhỏ (thường là 10^{-8}) để tránh lỗi chia cho 0. **Ý nghĩa:** Nếu một tham số có gradient lớn thường xuyên (s lớn), mẫu số sẽ lớn làm bước nhảy nhỏ lại. Ngược lại, tham số có gradient nhỏ sẽ được đẩy cho bước nhảy lớn hơn để nhanh chóng đạt đến tối ưu.

3.3 Adam

Adam (Adaptive Moment Estimation) là thuật toán tối ưu phổ biến nhất hiện nay, kết hợp ưu điểm của cả Momentum và RMSprop.

3.3.1 Adam (Almost) - Phiên bản sơ khai

Ý tưởng đầu tiên là kết hợp trực tiếp Moment bậc 1 (m_t - tương tự Momentum) và Moment bậc 2 (v_t - tương tự RMSprop):

- $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
- $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
- $W_{t+1} = W_t - \frac{\eta}{\sqrt{v_t}+\epsilon} m_t$

3.3.2 Vấn đề bước nhảy đầu tiên và Full-form Adam

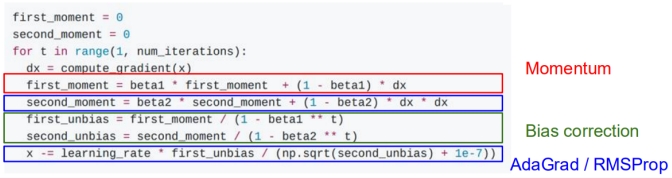
Vấn đề: Do m_t và v_t được khởi tạo bằng 0, và các hệ số β thường rất lớn (gần bằng 1), nên trong những bước đầu tiên, giá trị của m_t và v_t bị kéo về 0 rất mạnh (*bias towards zero*). Điều này khiến bước nhảy ban đầu bị sai lệch hoàn toàn.

Giải pháp (Bias Correction): Để bù đắp sai số này, ta chia cho một lượng hiệu chỉnh phụ thuộc vào thời gian t:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4}$$

Full-form Adam update rule:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{5}$$



3.3.3 AdamW: Vấn đề với L2 Regularization

Trong bài giảng có nhấn mạnh sự khác biệt tinh tế giữa Adam và AdamW khi sử dụng cùng với L2 Regularization (Weight Decay):

- **Adam (truyền thống):** Cộng đạo hàm của phần L2 trực tiếp vào gradient chung g_t. Điều này khiến phần Weight Decay cũng bị điều chỉnh bởi các biến m_t và v_t, làm mất đi bản chất của việc giảm trọng số đơn thuần.
- **AdamW (Weight Decay Decoupled):** Tách biệt hoàn toàn Weight Decay ra khỏi các bước tính toán Moment. Ta chỉ dùng Data Loss để tính m_t, v_t, còn phần giảm trọng số L2 sẽ được trừ trực tiếp vào W ở bước cuối cùng:

$$W_{t+1} = (1-\lambda)W_t - \text{Adam\_Update} \tag{6}$$

Kết luận: AdamW thường cho kết quả tốt hơn và ổn định hơn, đặc biệt là trong các mô hình lớn như Transformer.

4 Learning Rate Scheduling

Việc giữ nguyên một tốc độ học (*learning rate*) suốt quá trình là không tối ưu. Chúng ta cần giảm dần nó khi mô hình tiến gần đến đáy.

- **Step Decay:** Giảm theo bậc thang, tạo ra các cú "sụp"Loss đặc trưng mỗi khi hạ Learning Rate.
- **Cosine Decay:** Giảm theo hàm cos, rất mượt mà và phổ biến.
- **Linear Decay:** Giảm đều theo đường thẳng.
- **Inverse Sqrt:** Giảm theo nghịch đảo căn bậc hai của thời gian.
- **Linear Warm-up:** Tăng dần tốc độ học từ 0 lên giá trị cực đại trong vài trăm bước đầu để ổn định gradient, sau đó mới áp dụng các loại Decay.

5 Quy tắc tăng Batch và Learning Rate

Một quy tắc thực nghiệm quan trọng (**Linear Scaling Rule**): Khi bạn tăng kích thước **Batch** lên k lần, bạn nên tăng **Learning Rate** lên k lần tương ứng. *Lý do:* Batch lớn hơn cho một ước lượng Gradient chính xác hơn (ít nhiễu hơn), nên ta có thể "mạnh dạn"bước những bước lớn hơn mà không sợ bị chệch hướng.