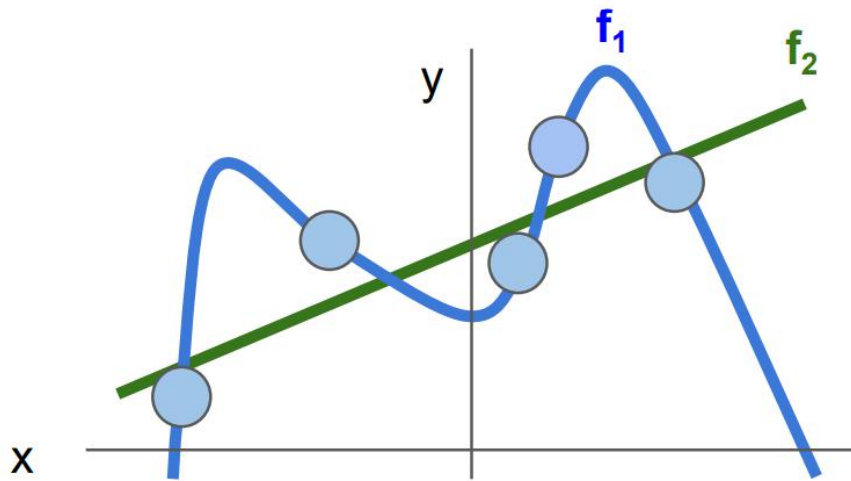
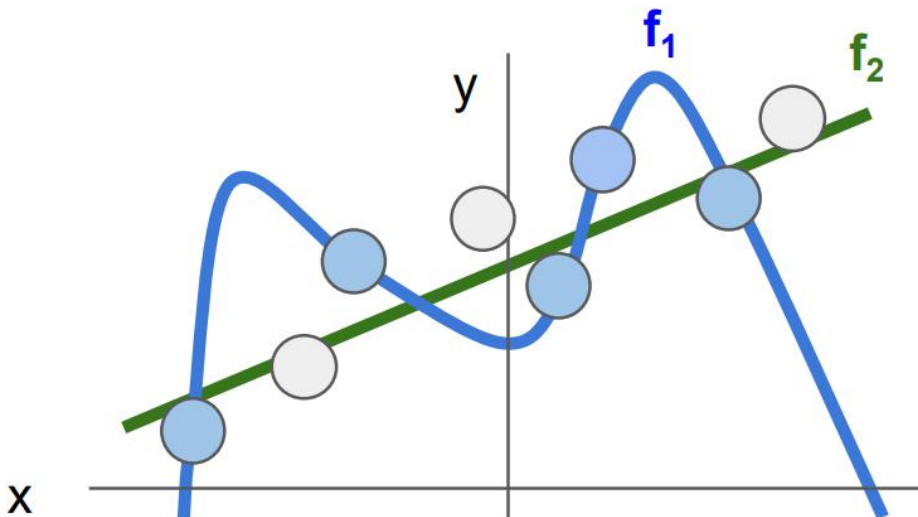


## 1. Regularization

### 1.1. Vấn đề đặt ra



Chúng ta có thể nhìn thấy 2 đường  $f_1$  và  $f_2$ .  $f_1$  đang fit hết với tất cả data training (Loss = 0) trong khi đó  $f_2$  chỉ gần các điểm data training (Loss nhỏ nhưng không bằng 0).



Khi có các điểm test data lên biểu đồ, ta có thể thấy  $f_1$  không có nằm gần các điểm nữa trong khi  $f_2$  nó vẫn giữ được độ chính xác giữa tập data training và data test (Loss vẫn nhỏ). Trường hợp này  $f_1$  đã bị overfitting tức là fit hết với tập data training nhưng sai số lớn trên tập test. Vì vậy, Regularization được sinh ra để ngăn cho mô hình bị rơi vào trường hợp overfitting.

### 1.2. Regularization là gì?

Trong công thức loss ở bài trước, hàm loss đang buộc phải giảm nhỏ nhất sai số gần như bằng 0 của mô hình:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Vì vậy, Regularization giúp *kìm* mô hình đừng fit quá tốt train. Công thức tổng quát:

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

Trong đó  $\lambda$  là độ ảnh hưởng của regularization(hyperparameter).  $\lambda$  thường được chọn từ 0 (không sử dụng regularization) tới vô tận và được tối ưu hóa thông qua tập training và test.

### 1.3. Các Regularization đơn giản

- L2 regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Thường làm weights nhỏ và mượt hơn (spread out). Vì đạo hàm của L2 là  $2\lambda w$  tức weight càng lớn  $\rightarrow$  càng mạnh  $\rightarrow$  giảm các weight lớn xuống  $\rightarrow$  phân tán đều hơn.

- L1 regularization

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Khuyến khích sparsity (nhiều weight = 0)  $\rightarrow$  giống chọn đặc trưng. Vì đạo hàm là 1 hằng số  $\lambda \text{sign}(w)$   $\rightarrow$  làm cho các weight dễ dính 0

- Elastic Net = L2 regularization + L1 regularization

Ngoài ra còn các Regularization khác như: Drop out, Batch Normalization, Stochastic Depth,...

## 2. Optimization

### 2.1. Optimization là gì?

Mục tiêu: Tìm tham số  $W$  sao cho hàm loss  $L(W)$  là nhỏ nhất

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

Vậy optimization là cách update  $W$  để  $L(W)$  nhỏ nhất.

### 2.2. Follow the slope

- 1D: đạo hàm là ra slope

- Nhiều chiều, vector gradient là đạo hàm theo từng chiều

- Hướng giảm nhanh nhất: negative gradient.

### 2.3. Tính Gradient

- Numerical Gradient là gradient xấp xỉ bằng cách thử dịch tham số một chút rồi đo loss thay đổi bao nhiêu.  $\rightarrow$  không cần biết công thức đạo hàm

$$\frac{\partial L}{\partial w_i} \approx \frac{L(w_i + h) - L(w_i)}{h}$$

Trong đó  $h$  thay đổi rất nhỏ.

+ Ưu điểm: Dễ viết code

+ Nhược điểm: Chỉ xấp xỉ, chậm (loop nhiều)

- Analytic Gradient là gradient được tính bằng toán học (giải tích), tức là lấy đạo hàm công thức loss một cách chính xác.

+ Ưu điểm: Chính xác, nhanh

+ Nhược điểm: dễ bug

Trong thực tế, luôn xài Analytic Gradient, còn dùng Numerical Gradient để check lại coi gradient này phù hợp không  $\rightarrow$  Gradient check

### 2.4. Gradient Descent

Cách update  $W$ :

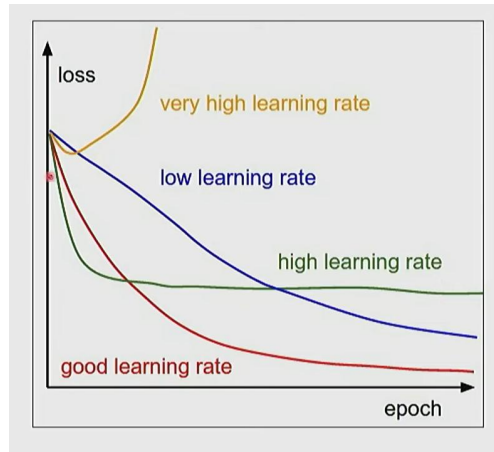
$$W_{t+1} = W_t - \eta \nabla_w L(W_t)$$

$\eta$ : tốc độ học của hàm

Nếu  $\eta$  quá lớn  $\rightarrow$  sẽ dễ dàng bỏ qua các điểm tối ưu

Nếu  $\eta$  quá nhỏ  $\rightarrow$  tốn nhiều thời gian tới điểm tối ưu

$\rightarrow$  Cần chọn  $\eta$  hợp lý



1 vấn đề khác: Gradient Descent dùng full dataset mỗi bước → rất tốn compute

## 2.5. Stochastic Gradient Descent – SGD

Ý tưởng: Approximate gradient bằng mini-batch. Batch size thường: 32 / 64 / 128

$$\nabla L \approx \frac{1}{N} \sum_{i \in N} \nabla L_i$$

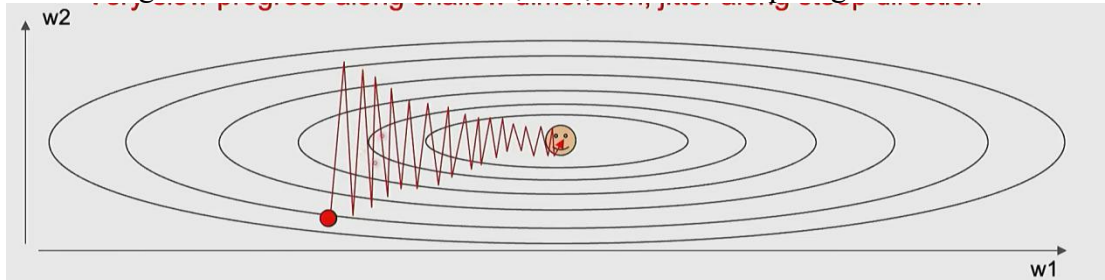
Ưu điểm: Nhanh

Nhược điểm: gradient noisy

## 2.6. Các problem với SGD

Problem 1: Poor conditioning

- SGD rung lắc theo chiều dọc và tiến cực chậm theo chiều phẳng



Vd: w2 thì quá dốc gây rung lắc mạnh trong khi w1 quá thoải nên đi chậm

Problem 2: Local minima và Saddle point

- Gradient = 0 → SGD đứng yên

Problem 3: Gradient noise

- Gradient đến từ nhiều local gradient → có thể có noise trong đó

## 2.7. SGD + Momentum

Ý tưởng: Thêm quán tính để update nhanh hơn.

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

$$W_{t+1} = W_t - \eta v_t$$

$\rho$ : momentum (0.9 hoặc 0.99)

Ưu điểm: vượt saddle point - làm mượt hướng đi (problem 2) → chủ yếu, giảm rung và tăng tốc theo hướng tốt (problem 1)

## 2.8. RMSProp

Ý tưởng: dùng gradient hiện tại để coi hướng và dùng gradient quá khứ để biết đi nhanh hay đi chậm trong hướng đó

$$dx = \nabla L(W_t)$$

$$grad\_square = \rho * grad\_square + (1 - \rho) * dx * dx$$

$$W_{t+1} = W_t - \eta \frac{dx}{\sqrt{grad\_square + \epsilon}}$$

dx là đạo hàm theo từng chiều

$\rho$  là decay rate và  $\eta$  là learning rate

grad\_square cho biết gradient lớn hay nhỏ -> lớn dốc -> nhỏ phẳng

$\frac{\eta}{\sqrt{grad\_square + \epsilon}}$  là learning rate riêng cho từng tham số

Ưu điểm: gặp dốc cao thì đi chậm lại còn phẳng thì đi nhanh -> tối ưu nhanh hơn, mỗi tham số sẽ được cập nhật riêng -> tăng chính xác

Nhìn thì RMSProp nó trái lại với SGD+Momentum. Tuy nhiên, SGD+Momentum cho biết nếu gradient ổn định theo thời gian thì sẽ đi nhanh hướng đó còn RMSProp cho biết nếu tham số đó thay đổi quá lớn thì chậm lại

## 2.9. Adam Optimizer

Tuy nhiên sẽ có trường hợp đường đi sẽ dốc và sẽ đi dốc hoài lúc đó thì SGD+Momentum sẽ tăng tốc trong khi RMSProp sẽ thắng lại hoài. Vì vậy, Adam ra đời để điều hòa điều đó.

Ý tưởng: Adam = Momentum + RMSProp -> sử dụng điểm mạnh của cả 2

$$dx = \nabla L(W_t)$$

$$v_t = \rho_1 v_{t-1} + (1 - \rho_1) * dx$$

$$grad\_square = \rho_2 * grad\_square + (1 - \rho_2) * dx * dx$$

$$W_{t+1} = W_t - \eta \frac{v_t}{\sqrt{grad\_square + \epsilon}} * dx$$

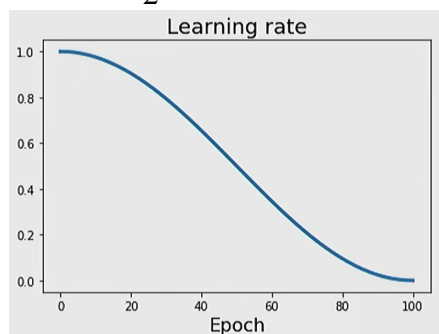
Ở đây dx là đạo hàm của hàm Loss có bao gồm Regularization trong đó. Còn nếu chỉ có data loss làm xong rồi cộng cập nhật của Regularization -> gọi là AdamW.

## 2.10. Learning rate decay

Ý tưởng: giảm learning rate giảm dần qua mỗi epoch -> đi nhanh lúc đầu chậm lại lúc sau

Vd cosine

$$\eta_t = \frac{1}{2} \eta_0 (1 + \cos(t\pi / T))$$



Vd ResNet sau mỗi epochs 30,60,90 thì nhân 0.1 vào learning rate.

Còn linear, Inverse sqrt, ...

Bên cạnh đó còn có Linear Warmup tức là lúc đầu chúng ta không dùng giá trị LR mong muốn mà tăng từ 0 bởi vì lúc đầu weight chưa ổn định nếu LR lớn thì sẽ khiến loss quá lớn khó kiểm soát.

### **2.11. First-order vs Second-order Optimization**

First-order: đạo hàm bậc 1 chỉ biết hướng đi  $\rightarrow$  Gradient Descent + SGD

Second-order: kết hợp với Hessian biết thêm độ cong