

Bài 5:

Huấn luyện mạng nơ-ron

(Phần 2)

Nội dung

1. Các giải thuật tối ưu cho mạng nơ-ron
2. Chiến lược thay đổi tốc độ học
3. Một số kỹ thuật chống overfitting
4. Làm giàu dữ liệu (data augmentation)
5. Lựa chọn siêu tham số
6. Kỹ thuật kết hợp nhiều mô hình (ensemble)
7. Kỹ thuật học tái sử dụng (transfer learning)

Các giải thuật tối ưu

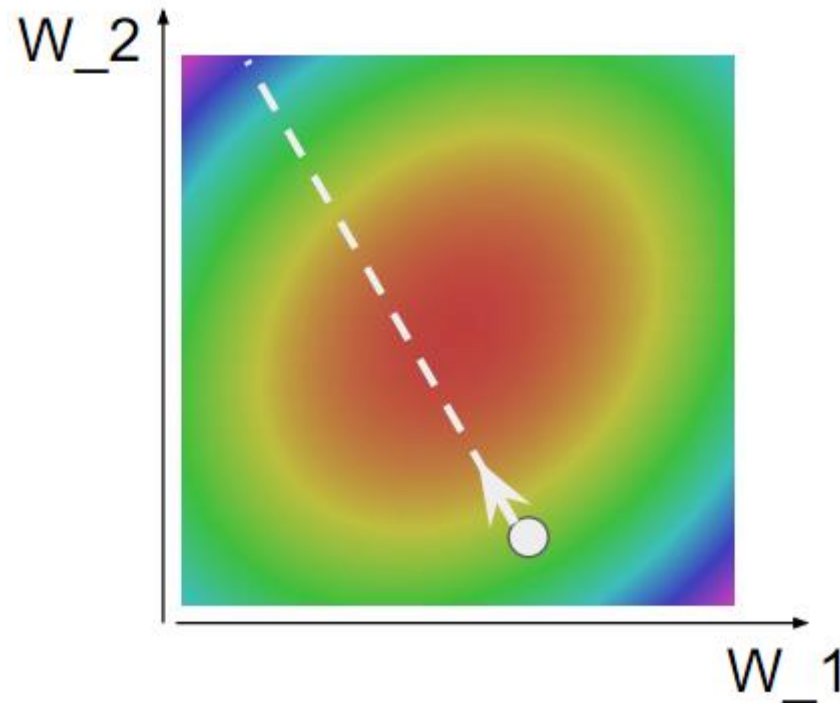
Phương pháp SGD

```
# Vanilla Gradient Descent
```

```
while True:
```

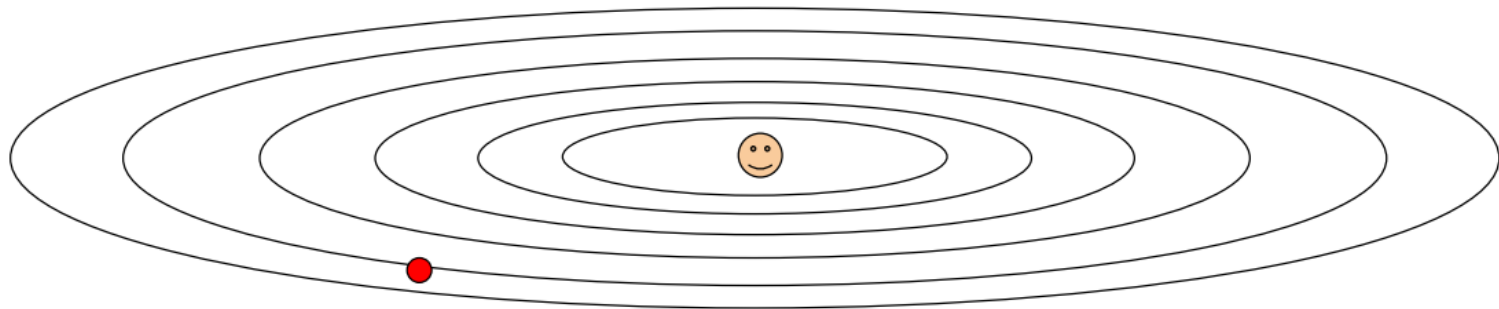
```
    weights_grad = evaluate_gradient(loss_fun, data, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```



Vấn đề với SGD

- Điều gì sẽ xảy ra khi hàm mục tiêu thay đổi nhanh theo một chiều và thay đổi chậm theo chiều khác?
- Khi đó SGD sẽ làm việc như thế nào?

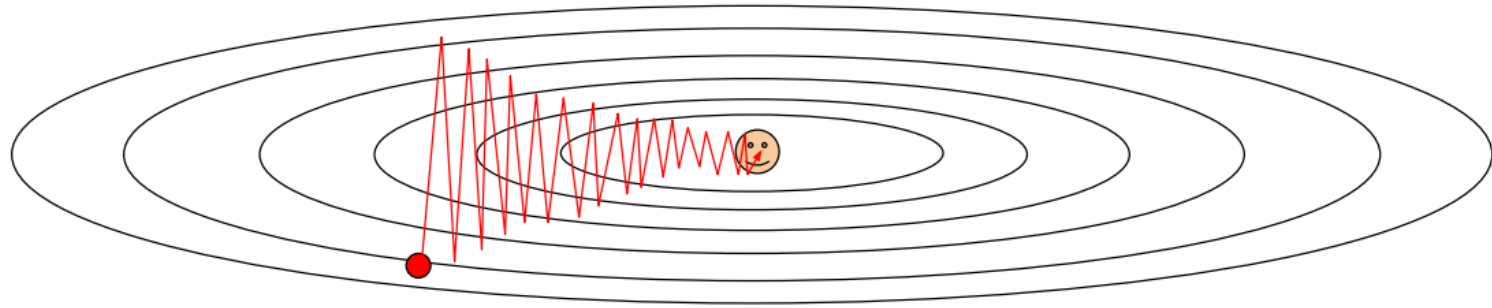


Hàm mục tiêu có **số điều kiện** lớn: tỉ lệ giữa giá trị riêng lớn nhất và giá trị riêng nhỏ nhất của ma trận Hessian là lớn.

Vấn đề với SGD

- Điều gì sẽ xảy ra khi hàm mục tiêu thay đổi nhanh theo một chiều và thay đổi chậm theo chiều khác?
- Khi đó SGD sẽ làm việc như thế nào?

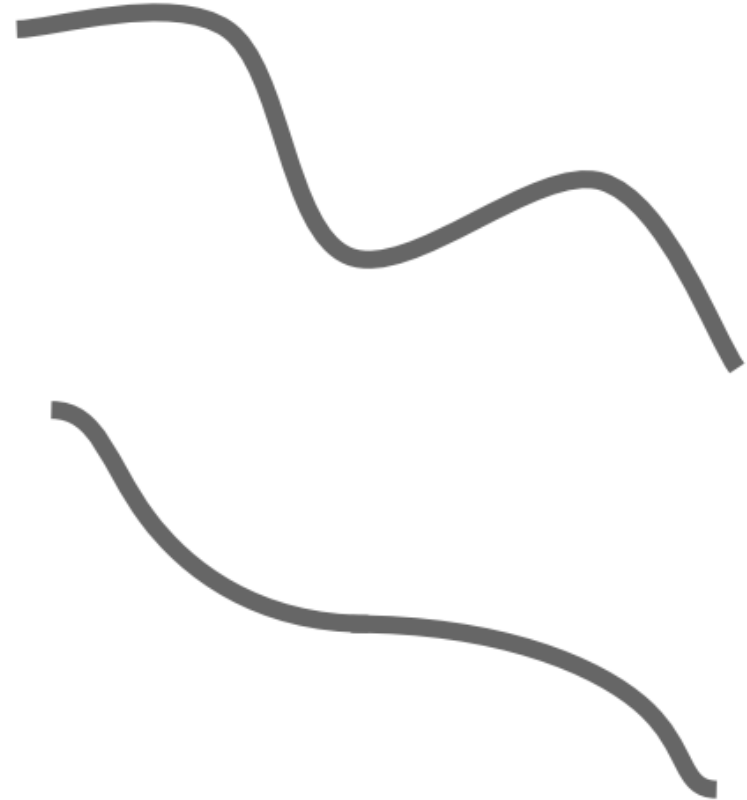
Thuật toán hội tụ rất chậm, nhảy từ bên này qua bên kia bề mặt hàm mục tiêu



Hàm mục tiêu có **số điều kiện** lớn: tỉ lệ giữa giá trị riêng lớn nhất và giá trị riêng nhỏ nhất của ma trận Hessian là lớn.

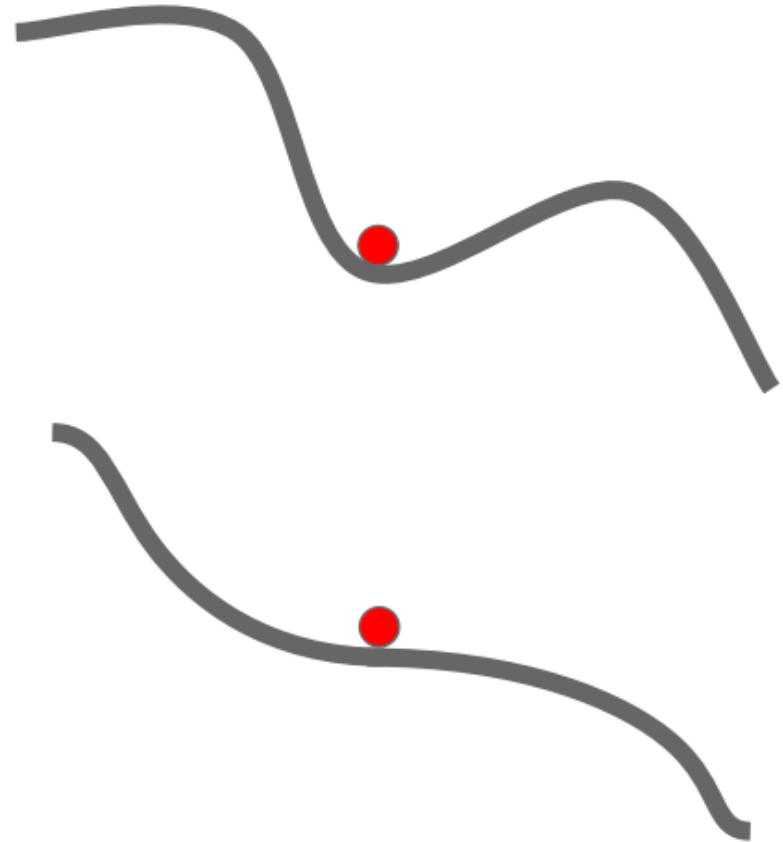
Vấn đề với SGD

- Chuyện gì xảy ra nếu hàm mục tiêu có cực tiểu địa phương hoặc điểm yên ngựa (saddle point)?



Vấn đề với SGD

- Chuyện gì xảy ra nếu hàm mục tiêu có cực tiểu địa phương hoặc điểm yên ngựa (saddle point)?
- Gradient bằng 0, thuật toán SGD bị tắc
- Điểm yên ngựa thường xuất hiện với các hàm mục tiêu nhiều biến

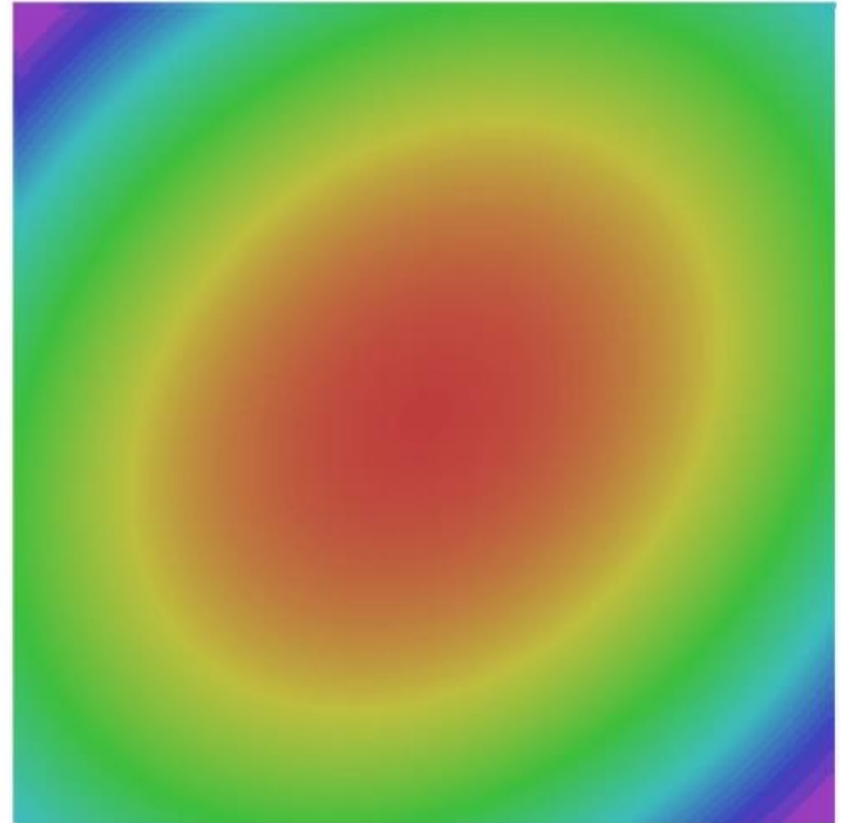


Vấn đề với SGD

- SGD xấp xỉ gradient theo từng lô dữ liệu nên thường rất nhiễu

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

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W)$$



SGD + momentum



SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
    dx = compute_gradient(x)
    x -= learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

- Xây dựng đại lượng “vận tốc” bằng trung bình dịch chuyển của gradients
- Lực ma sát rho thường bằng 0.9 hoặc 0.99.
- Tại thời điểm ban đầu rho có thể thấp hơn do hướng di chuyển chưa rõ ràng, ví dụ rho = 0.5

SGD + momentum

SGD+Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx - learning_rate * dx
    x += vx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

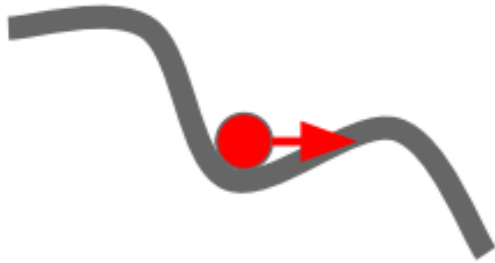
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
    dx = compute_gradient(x)
    vx = rho * vx + dx
    x -= learning_rate * vx
```

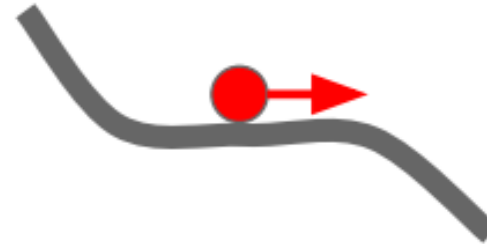
- SGD + momentum có thể phát biểu theo nhiều cách khác nhau nhưng chúng tương đương nhau và đều đưa ra cùng một dãy x

SGD + momentum

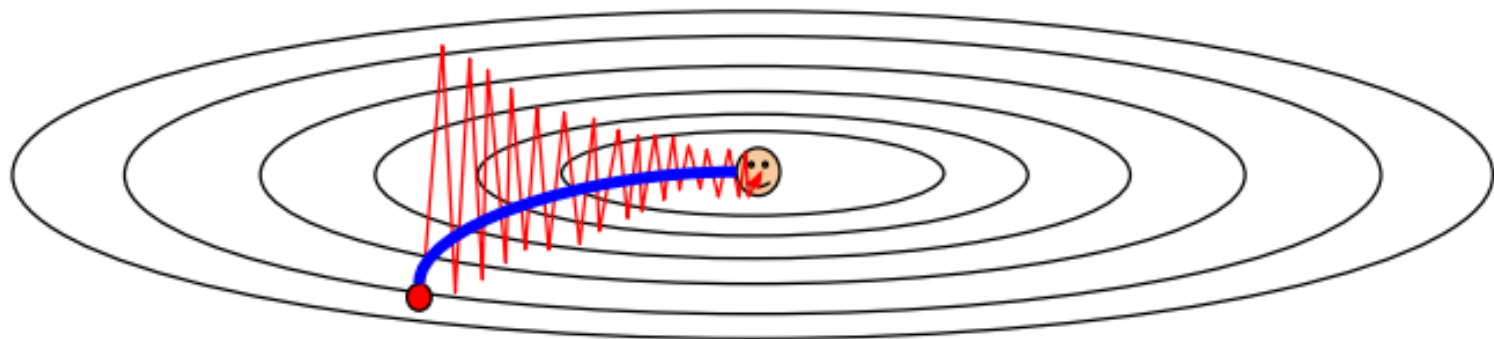
Local Minima




Saddle points



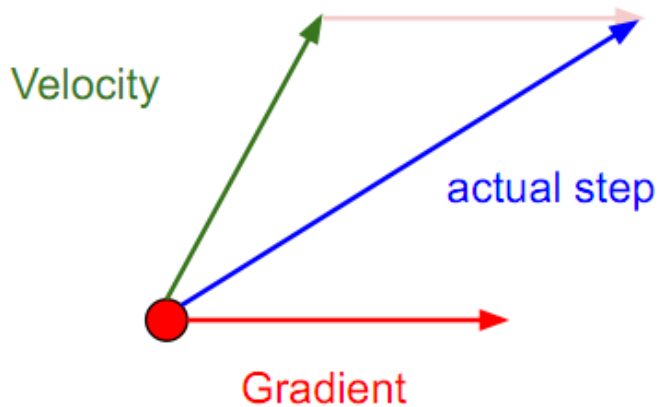
Poor Conditioning



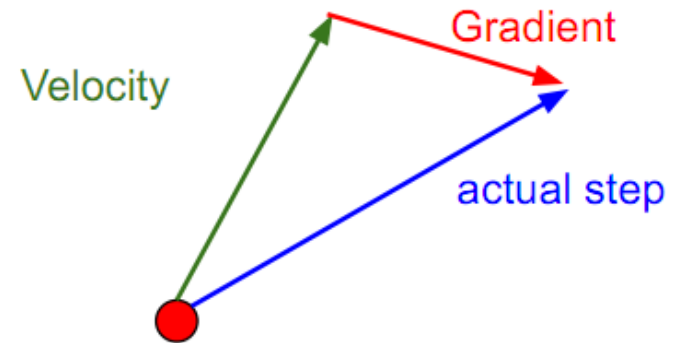
 SGD+Momentum

Nesterov Momentum

Momentum update:



Nesterov Momentum



$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

Nesterov Momentum

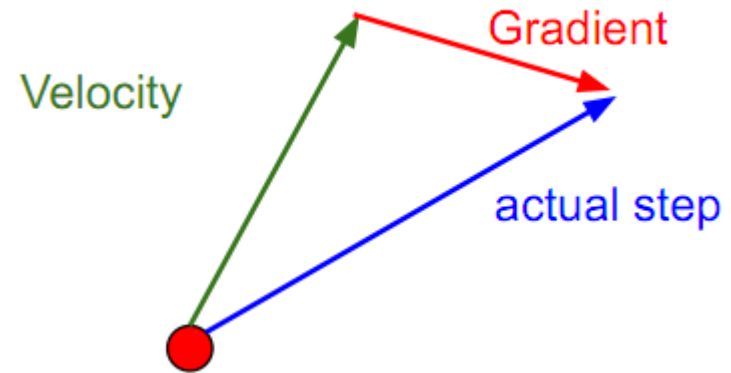
$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

- Thường người ta muốn tính theo $x_t, \nabla f(x_t)$
- Đặt $\tilde{x}_t = x_t + \rho v_t$ và chuyển về

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$

$$\begin{aligned} \tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + (1 + \rho)v_{t+1} \\ &= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t) \end{aligned}$$



```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

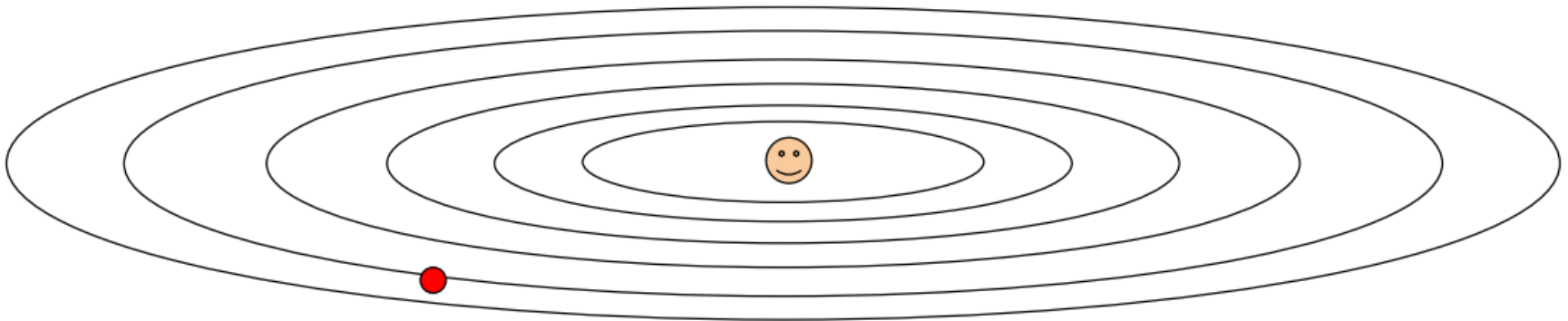
AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

- Mỗi trọng số có tốc độ học riêng: “Per-parameter learning rates” hoặc “adaptive learning rates”
- Tốc độ học của mỗi trọng số tỉ lệ nghịch với tổng bình phương độ lớn đạo hàm riêng của hàm mục tiêu đối với trọng số đó ở các bước trước

AdaGrad

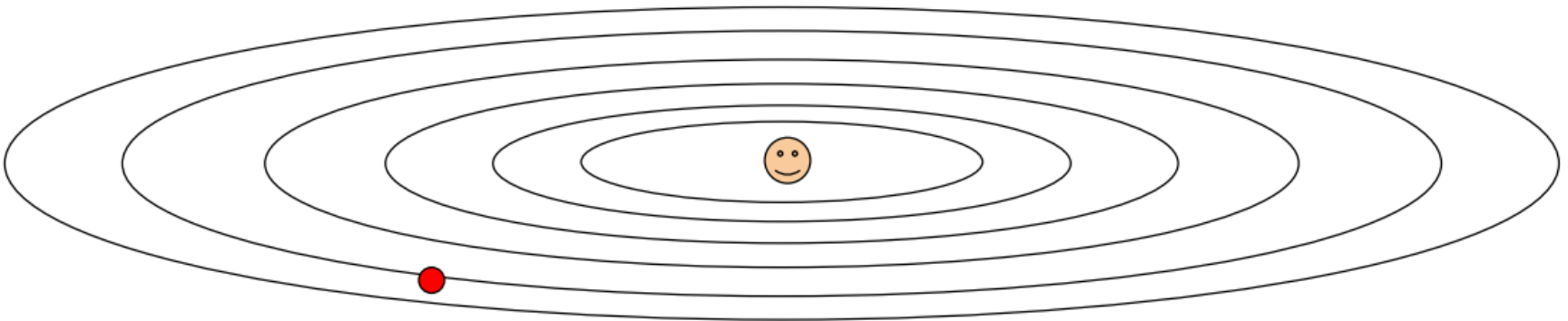
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



- Q1: Điều gì xảy ra với AdaGrad?

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



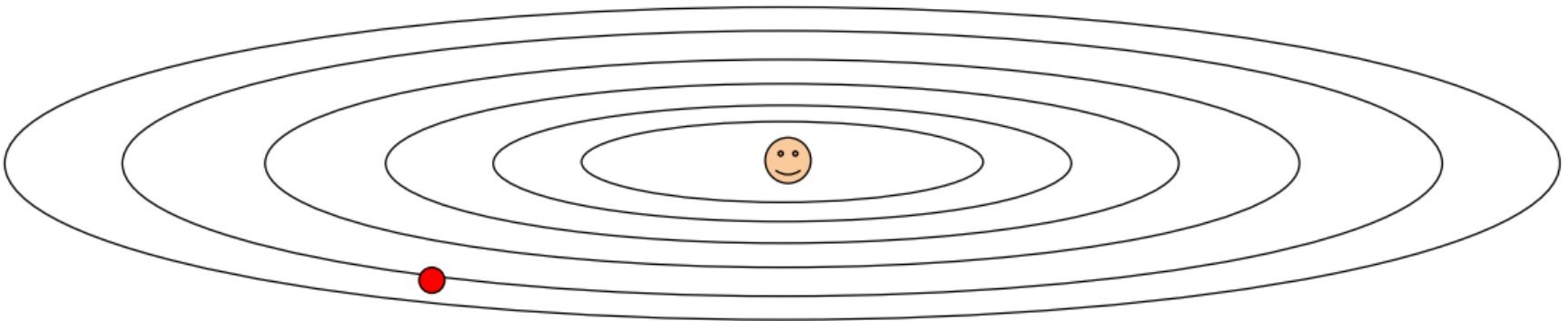
- Q1: Điều gì xảy ra với AdaGrad?

Tốc độ di chuyển theo hướng dốc được hãm dần

Tốc độ di chuyển theo hướng thoải được tăng tốc

AdaGrad

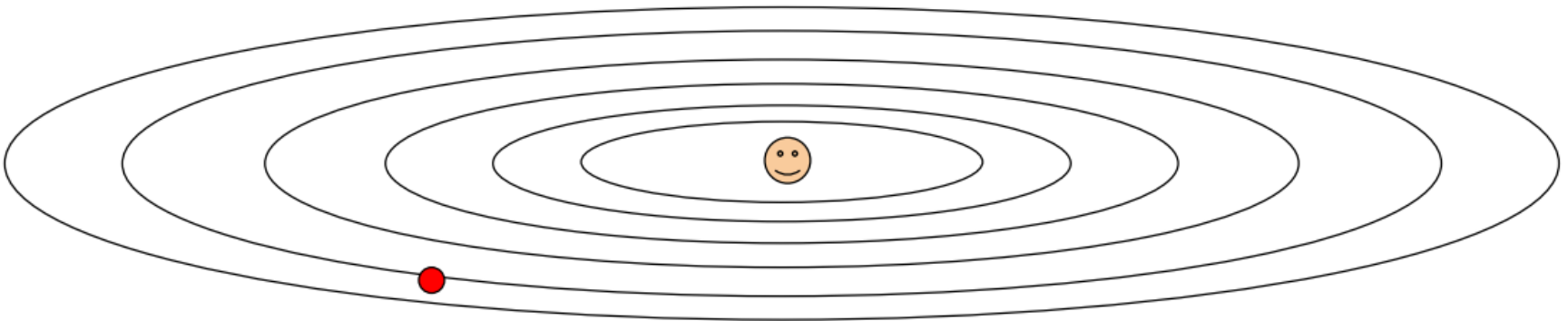
```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



- Q2: Bước di chuyển thay đổi như thế nào khi số vòng lặp tăng dần?

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



- Q2: Bước di chuyển thay đổi như thế nào khi số vòng lặp tăng dần?

Tiến tới 0

RMSProp

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Adam đơn giản

```
first_moment = 0
second_moment = 0
while True:
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Có thể xem như là RMSProp + Momentum

Adam đầy đủ



```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
```

Momentum

```
first_moment = beta1 * first_moment + (1 - beta1) * dx
```

```
second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
```

Bias correction

```
first_unbias = first_moment / (1 - beta1 ** t)
```

```
second_unbias = second_moment / (1 - beta2 ** t)
```

```
x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

AdaGrad / RMSProp

- Hiệu chỉnh bias để thuật toán đỡ bị ảnh hưởng bởi giá trị của β_1 và β_2 . Đồng thời giúp cho thuật toán ổn định hơn trong quá trình *warm up* tại một số bước đầu tiên khi cả hai moment đều khởi tạo bằng 0.
- Chứng minh chi tiết có thể tham khảo tại [Tài liệu tham khảo số 2](#) hoặc trong [bài báo gốc](#)
- Adam với $\beta_1 = 0.9$, $\beta_2 = 0.999$, và $\text{learning_rate} = 1e-3$ hoặc $5e-4$ là tham số mặc định tốt cho nhiều mô hình!

SOTA optimizers

- NAdam = Adam + NAG
- RAdam (Rectified Adam)
- LookAhead
- Ranger = RAdam + LookAhead

Trong thực tế

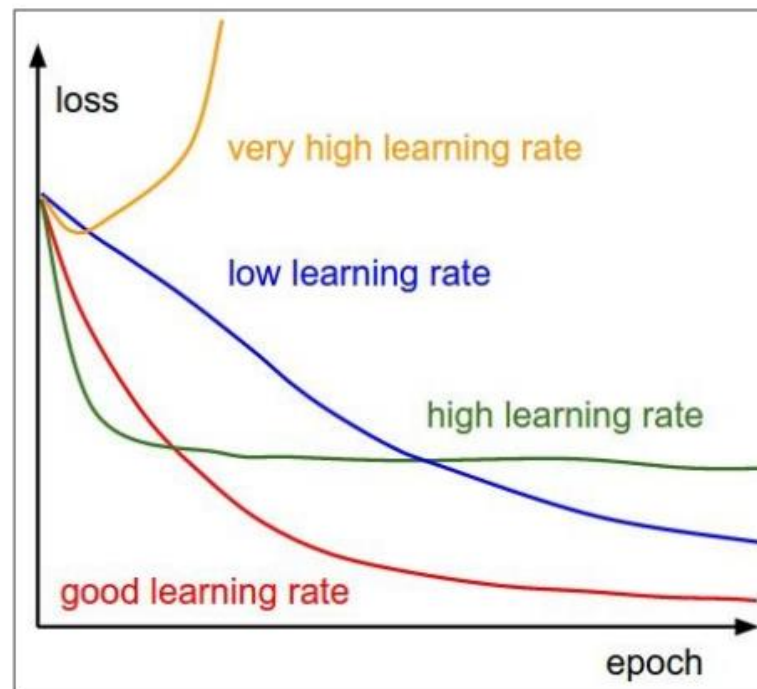


- Adam là lựa chọn mặc định tốt trong nhiều trường hợp
- SGD+Momentum thường tốt hơn Adam nhưng cần phải tinh chỉnh tốc độ học và lên chiến lược thay đổi tốc độ học hợp lý

Chiến lược thay đổi tốc độ học

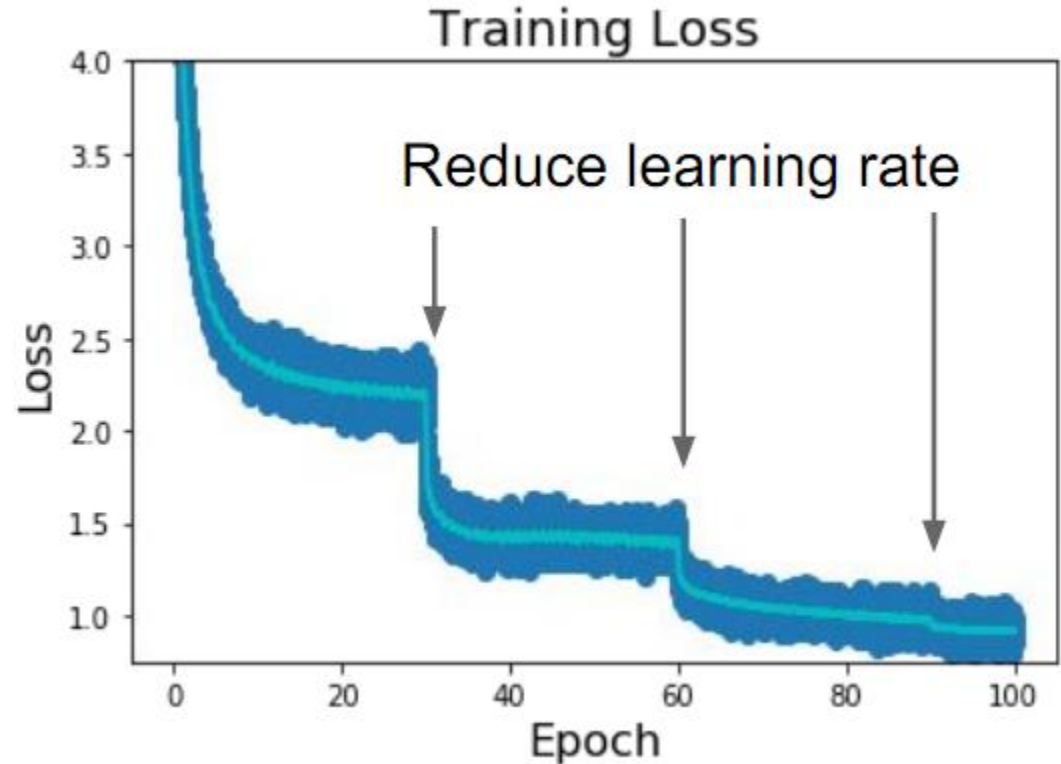
Tốc độ học

- Tốc độ học là siêu tham số (hyperparameter) của tất cả các thuật toán tối ưu SGD, SGD+Momentum, Adagrad, RMSProp, Adam...
- Thường bắt đầu với giá trị lớn và giảm dần theo thời gian



Chiến lược thay đổi tốc độ học

- Step: Thay đổi tốc độ học tại một số thời điểm cố định.
- Ví dụ: với ResNets có thể giảm lr 10 lần tại các epochs 30, 60 và 90.



Chiến lược thay đổi tốc độ học

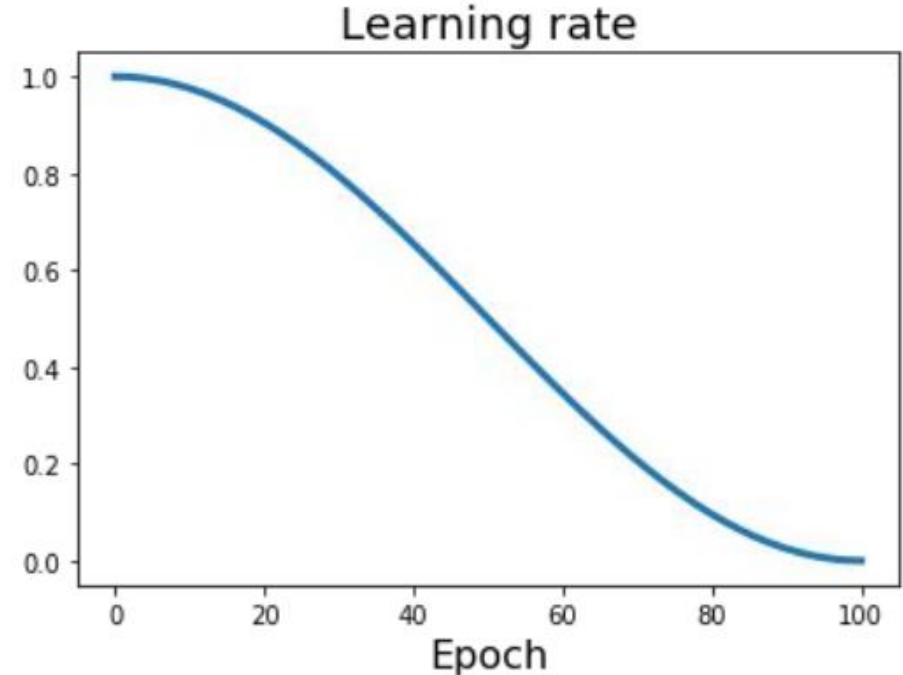
- Giảm theo cosin

$$\alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(t\pi/T))$$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs



Chiến lược thay đổi tốc độ học

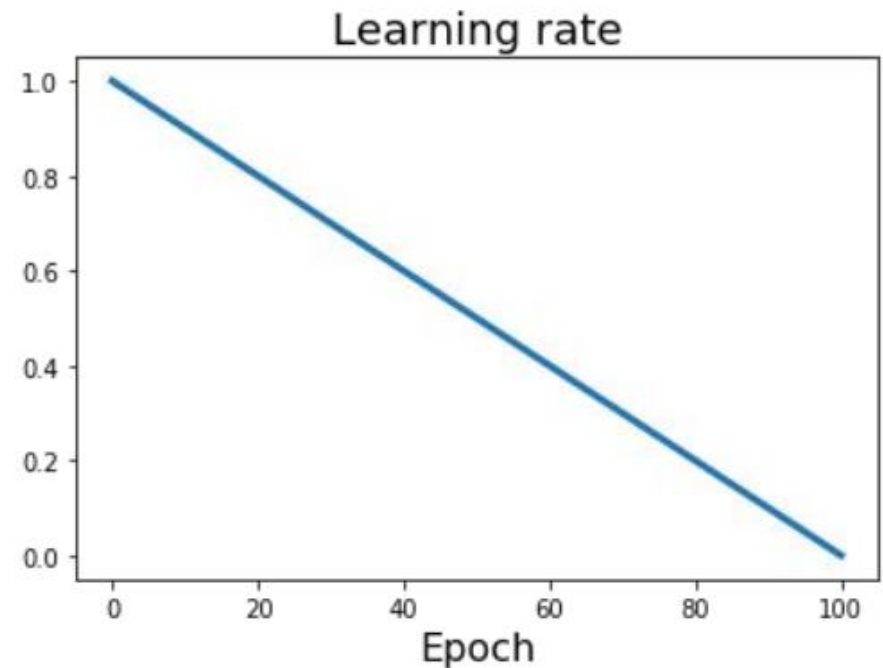
- Giảm tuyến tính

$$\alpha_t = \alpha_0(1 - t/T)$$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs



Chiến lược thay đổi tốc độ học

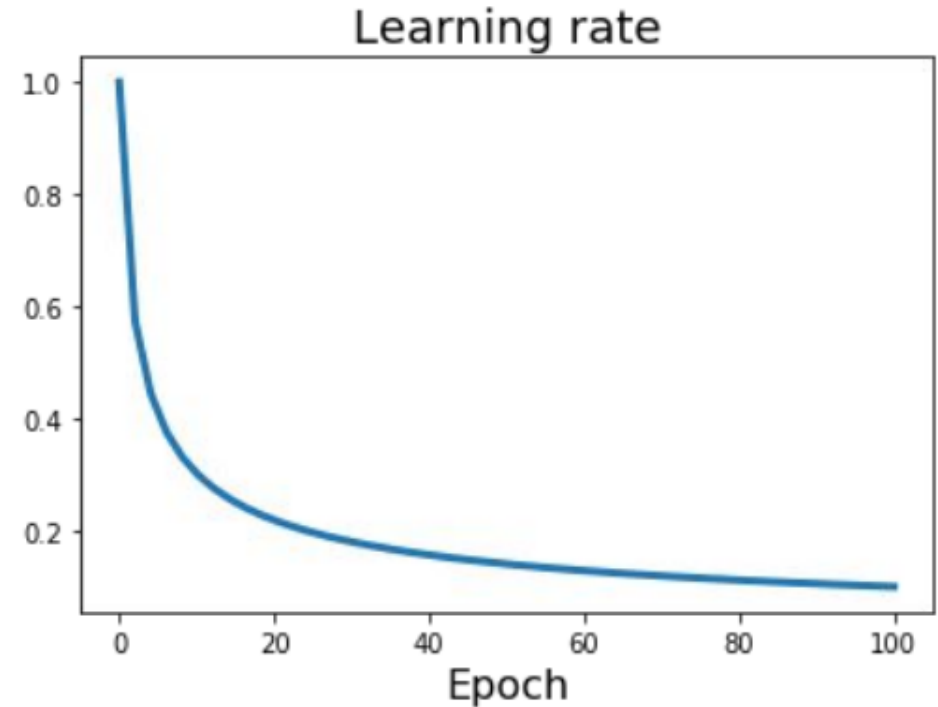
- Tỉ lệ nghịch căn bậc hai số epoch:

$$\alpha_t = \alpha_0 / \sqrt{t}$$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

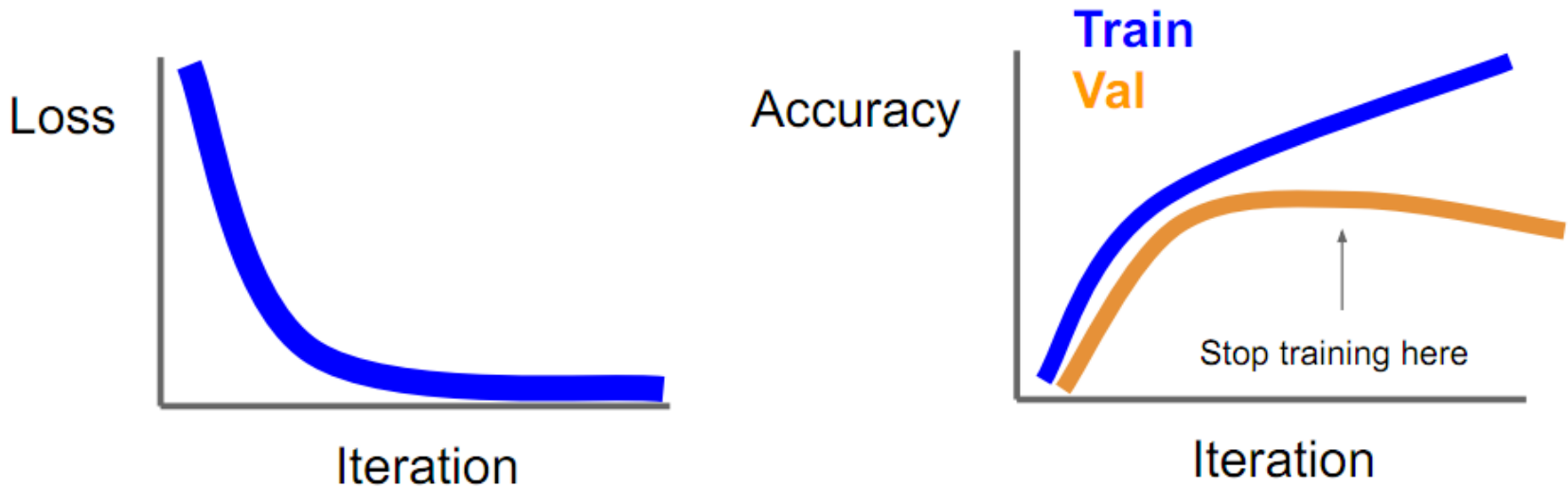
T : Total number of epochs



Một số kỹ thuật chống overfitting

Dừng sớm

- Dừng huấn luyện khi độ chính xác trên tập val bắt đầu giảm



Điều khiển quá trình huấn luyện



$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Một số ràng buộc hay sử dụng:

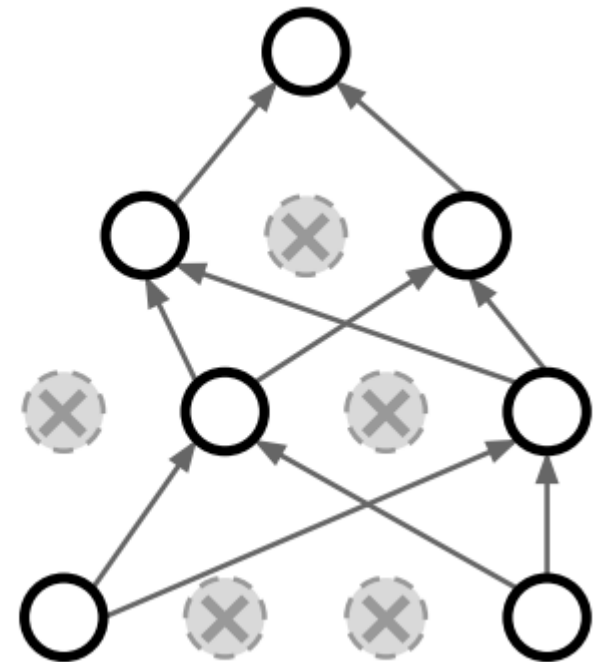
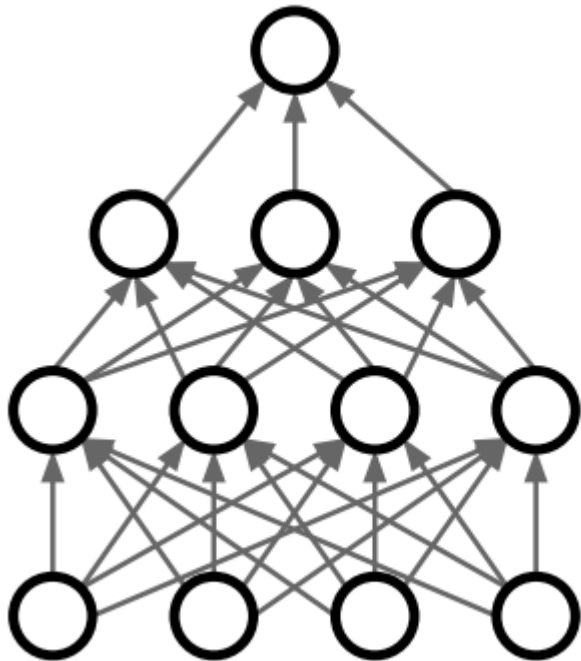
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Dropout

- Trong quá trình tính toán tiến (forward pass), ngẫu nhiên thiết lập đầu ra một số nơ-ron về 0.
- Xác suất drop thường là 0.5



Dropout

- Ví dụ quá trình tính toán tiến của một mạng nơ-ron 3 lớp sử dụng dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

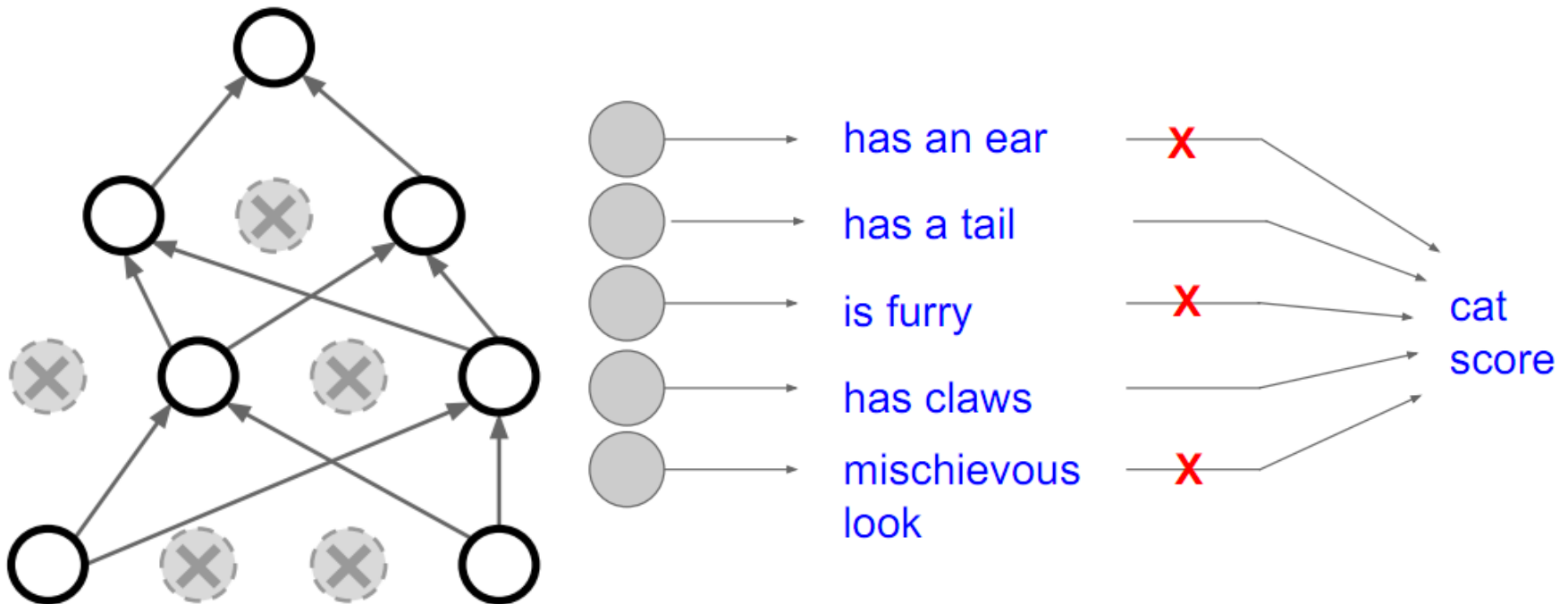
def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

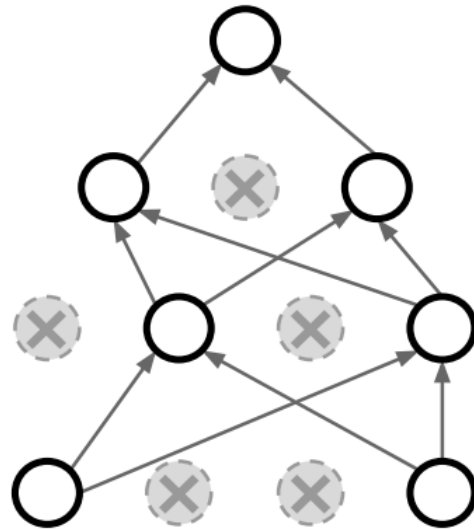
Tác dụng dropout

- Ép mạng nơ-ron phải học biểu diễn dư thừa (redundant representation)



Tác dụng dropout

- Dropout khi huấn luyện có thể diễn giải như huấn luyện đồng thời nhiều mô hình khác nhau
- Mỗi kiểu drop nơ-ron tương ứng với một mô hình
- Một lớp kết nối đầy đủ với 4096 nơ-ron sẽ có $2^{4096} \sim 10^{1233}$ phương án drop
- ... chỉ có cỡ 10^{82} nguyên tử trong toàn bộ vũ trụ!



Lúc suy diễn

- Dropout làm kết quả đầu ra ngẫu nhiên
-

$$\begin{array}{c} \text{Output} \\ \text{(label)} \end{array} \quad \begin{array}{c} \text{Input} \\ \text{(image)} \end{array} \quad \begin{array}{c} \text{Random} \\ \text{mask} \end{array}$$

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

- Cần phải lấy trung bình tất cả các kết quả

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$

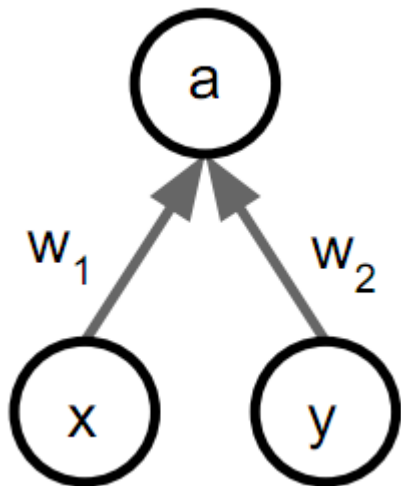
- Nhưng tích phân này là không thể...

Lúc suy diễn

- Xấp xỉ tích phân

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

- Ví dụ xét một nơ-ron



- Lúc suy diễn: $E[a] = w_1x + w_2y$
- Lúc huấn luyện:

$$\begin{aligned}
 E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\
 &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\
 &= \frac{1}{2}(w_1x + w_2y)
 \end{aligned}$$

Lúc suy diễn

- Lúc suy diễn tất cả nơ-ron đều hoạt động. Vì vậy phải scale đầu ra của mỗi nơ-ron:

Đầu ra khi suy diễn = kỳ vọng đầu ra khi huấn luyện

→ Nhân với tỉ lệ **keeping rate**

```
def predict(X):  
    # ensembled forward pass  
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations  
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations  
    out = np.dot(W3, H2) + b3
```

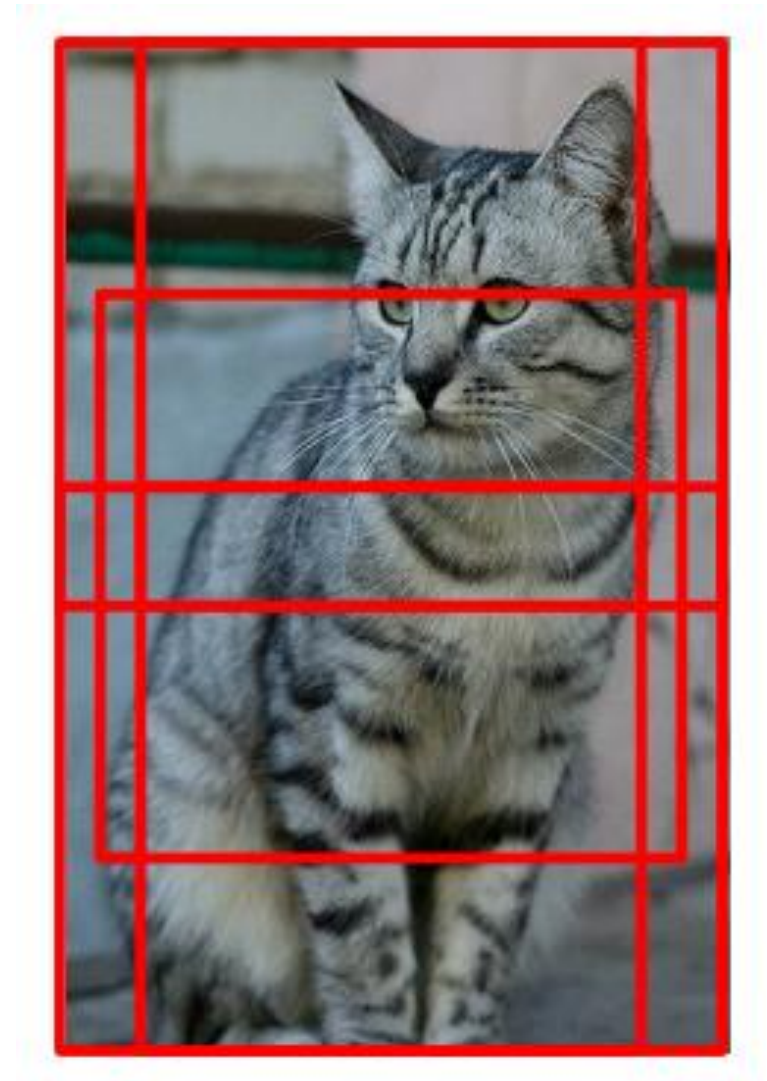

Làm giàu dữ liệu Data Augmentation

Flip ngang



Crop ngẫu nhiên và scale ảnh

- Ví dụ ResNet:
 1. Chọn ngẫu nhiên L trong khoảng $[256, 480]$
 2. Resize ảnh để chiều nhỏ nhất bằng L
 3. Crop ngẫu nhiên vùng kích thước 224×224



Thay đổi màu sắc



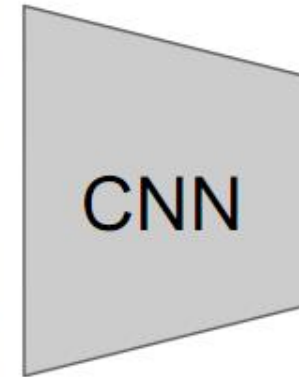
Các phép biến đổi khác...

- Tịnh tiến
- Xoay ảnh
- stretching
- shearing
- lens distortions...

Mixup



Randomly blend the pixels of pairs of training images, e.g. 40% cat, 60% dog



Target label:
cat: 0.4
dog: 0.6

Một số thư viện

1. Alumentations

<https://github.com/alumentations-team/alumentations>

2. Imgaug

<https://github.com/aleju/imgaug>

3. Augmentor

<https://github.com/mdbloice/Augmentor>

Lựa chọn siêu tham số

Siêu tham số

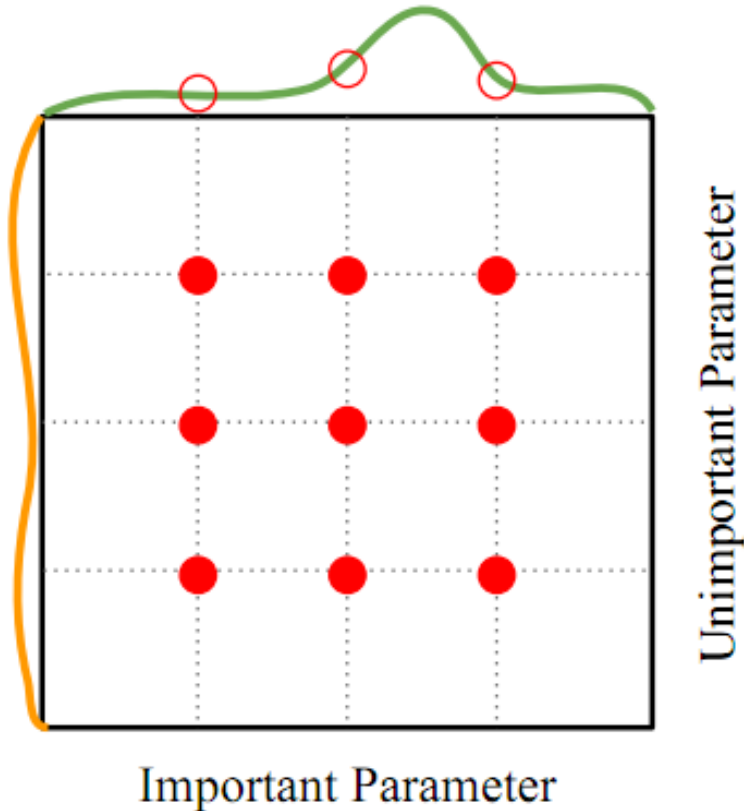
- Kiến trúc mạng
- Tốc độ học, tham số trong chiến lược thay đổi tốc độ học, thuật toán tối ưu
- Các hệ số điều khiển (L2 weight decay, drop rate)

neural networks practitioner
music = loss function

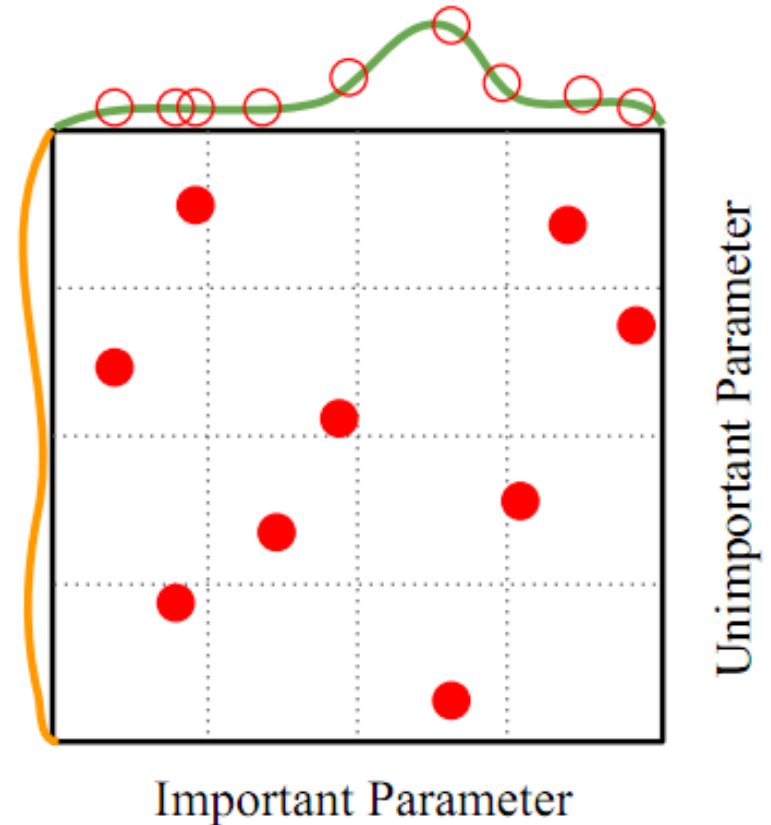


Random Search vs Grid Search

Grid Layout



Random Layout



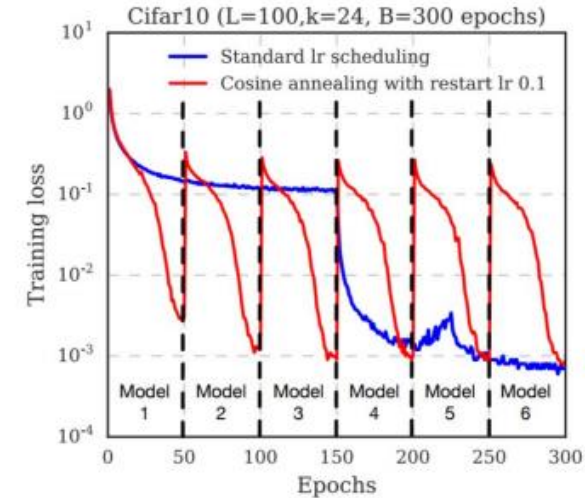
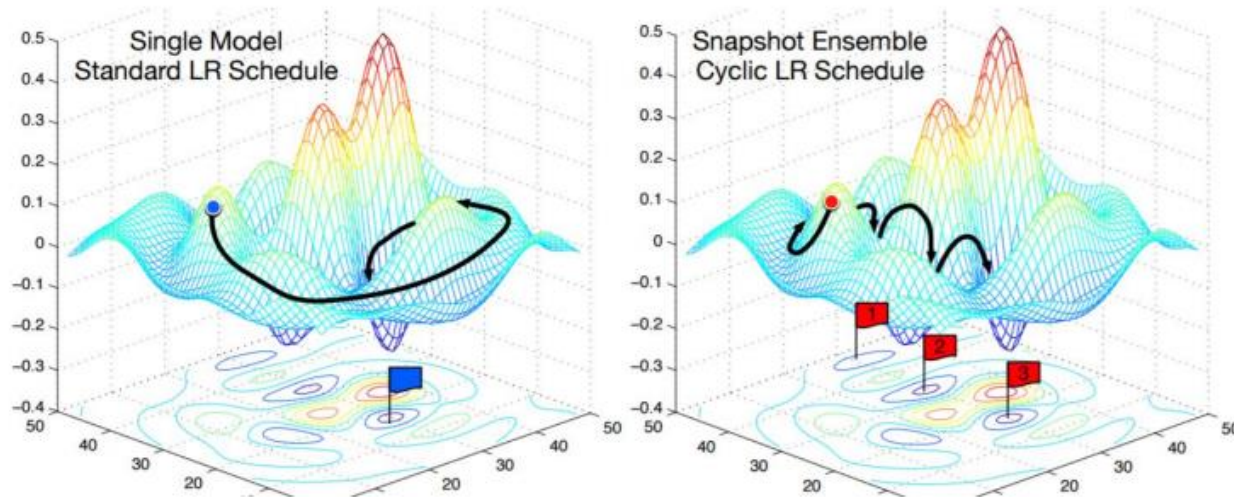
Kỹ thuật kết hợp nhiều mô hình (model ensemble)

Model Ensembles

- Huấn luyện nhiều mô hình độc lập
- Khi test kết hợp kết quả nhiều mô hình
- Độ chính xác thường tăng 2%

Model Ensembles

- Thay vì huấn luyện nhiều mô hình độc lập, có thể dùng nhiều snapshot của cùng một mô hình trong quá trình huấn luyện



Kỹ thuật học tái sử dụng (transfer learning)

Transfer learning

Huấn luyện mạng trên một tập dữ liệu lớn có sẵn, sau đó huấn luyện tiếp với tập dữ liệu của mình

1. Train on Imagenet



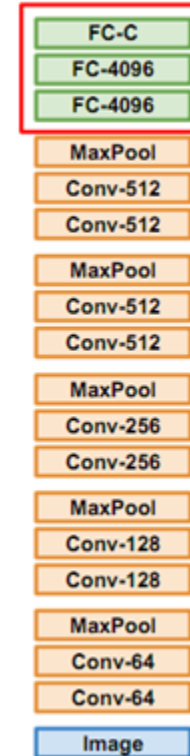
2. Small Dataset (C classes)



Khởi tạo khối này và huấn luyện

Đóng băng

3. Bigger dataset



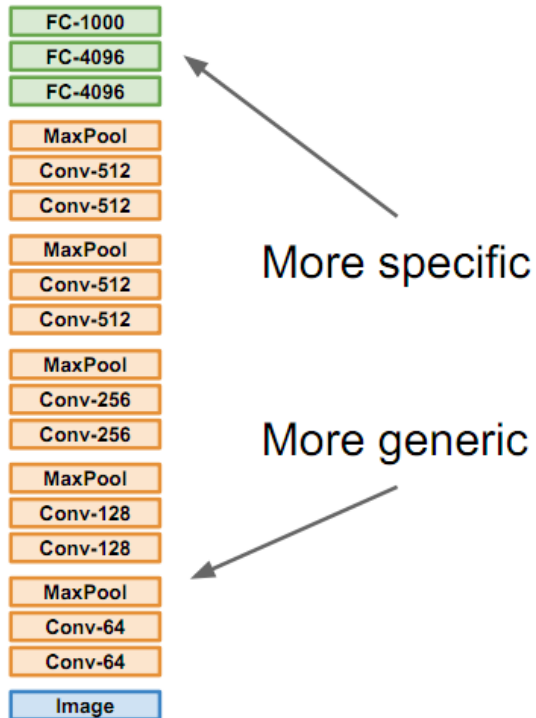
Huấn luyện các khối này

Dữ liệu nhiều hơn có thể huấn luyện nhiều lớp hơn

Đóng băng

Thường dùng lnr bé, ví dụ khoảng 1/10 lnr gốc ban đầu

Transfer learning



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

More tips and tricks

- Machine Learning Yearning by Andrew Ng

<https://d2wvfoqc9gyqzf.cloudfront.net/content/uploads/2018/09/Ng-MLY01-13.pdf>

Tài liệu tham khảo

1. Bài giảng biên soạn dựa trên khóa cs231n của Stanford, bài giảng số 8:

<http://cs231n.stanford.edu>

2. Adam:

<https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>

3. Stanford lecture note:

<http://cs231n.github.io/neural-networks-3/>