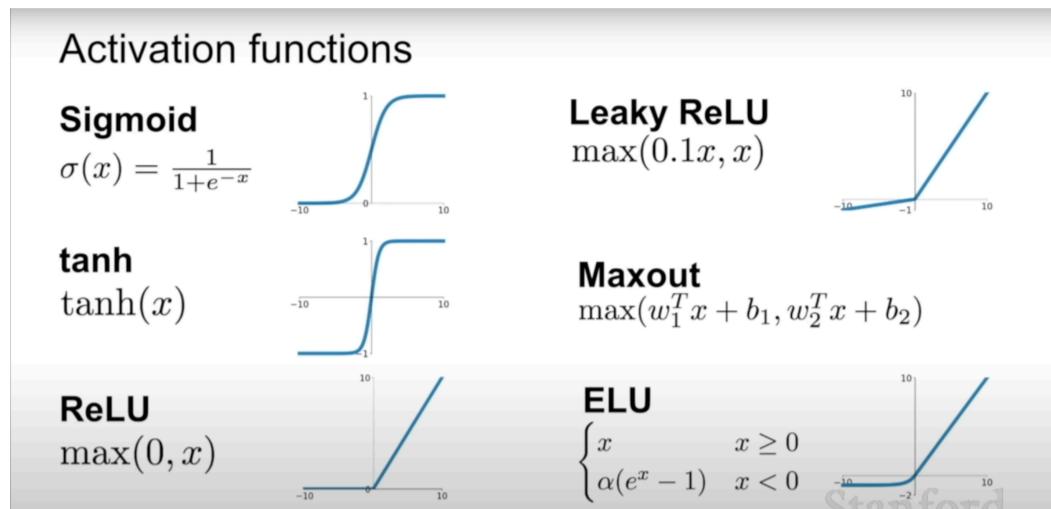


Training Neural Networks: Part 1

Activation Functions



Sigmoid

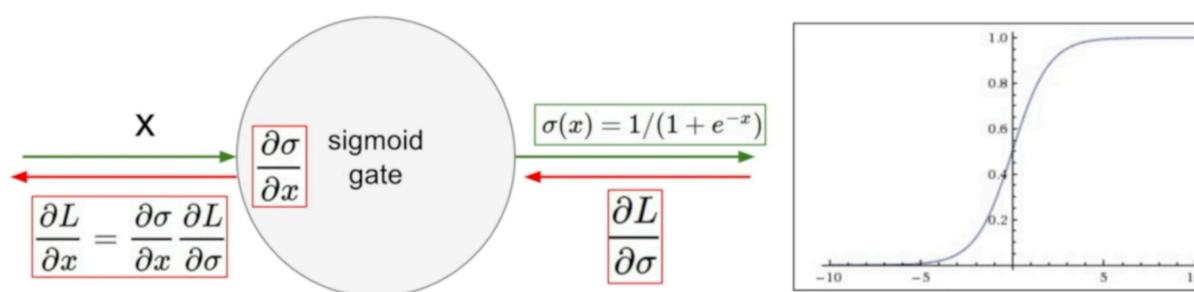
$$\sigma(x) = \frac{1}{(1 + e^{-x})}$$

숫자를 범위 [0,1]로 압축

뉴런의 발화율이 포화하는 것(역치)을 나타낼 때 좋은 해석이 되기에 많이 사용

Sigmoid function의 한계

1. 포화된 뉴런이 기울기 소실 문제를 일으킴



위와 같은 상황에서

if $x = -10$ or $10 \rightarrow$ sigmoid gate에서 0을 전달 (sigmoid function이 평평한 지점)

- Backprop의 chain rule을 계산할 때, 매우 작은 gradient로 인해 네트워크가 제대로 학습하지 못함
→ function이 평평하다는 것은 기울기(gradient)가 near-zero임

2. Sigmoid outputs이 zero-centered 하지 않음

: Sigmoid function은 [0,1]의 범위를 가지므로 x는 항상 양수를 띨

if input neuron x 의 부호가 모두 같다면

- w 에 대한 gradient가 '모두' 양수 혹은 음수가 되어버림
:
 w 의 local gradient는 x 이기에 ($\frac{df}{dw} = x$) 부호가 모두 같아짐

⇒ upstream gradient의 부호가 모두 같아짐을 직관적으로 알 수 있음

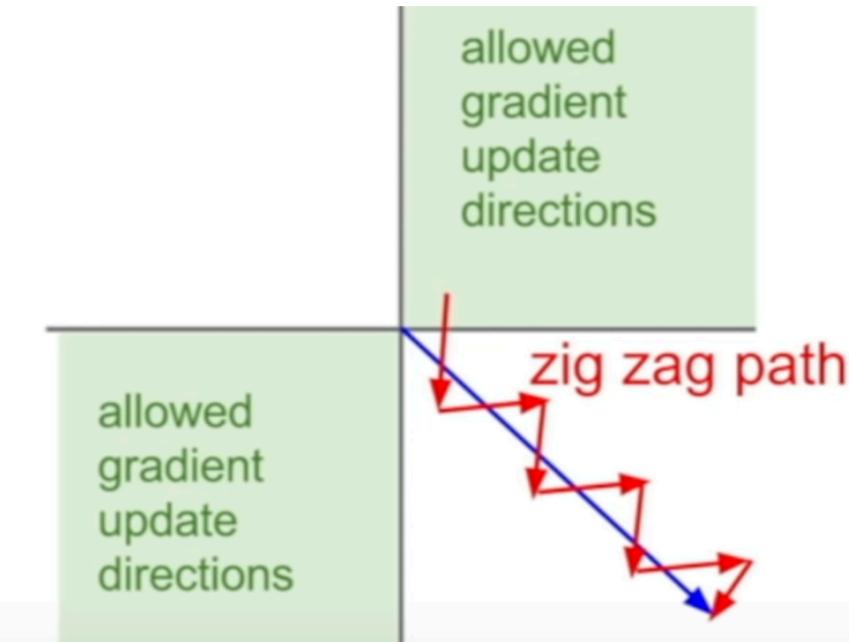
w 의 gradient 부호가 모두 같아지면 다음과 같이 항상 같은 방향으로 움직일 수 밖에 없음

→ 즉, 업데이트할 때

- 부호가 모두 양수라면, W 는 항상 상승
- 부호가 모두 음수라면, W 는 항상 감소

⇒ 매우 비효율적인 최적화과정

+) 이 지점이 우리가 평균이 0인 데이터를 선호하는 이유



W 가 2- d 일 때, 업데이트 방향이 항상 같다면 위의 빨간선(zigzag path)과 같이 gradient 업데이트를 할 수 있는 방향이 고정됨.

* 최적의 업데이트 방향은 파란선(=hypothetical optimal
 w vector)

3. 지수함수는 계산비용이 큼

: 다른 layer의 비용이 매우 높기에 부수적인 문제정도로 간주

tanh (Hyperbolic Tangent)

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

숫자를 범위 [-1,1]로 압축 (= zero-centered)

포화로 인한 기울기 소실 문제 발생

ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

(양수 영역에서) 포화하지 않음

계산 비용 측면에서 매우 효율적임

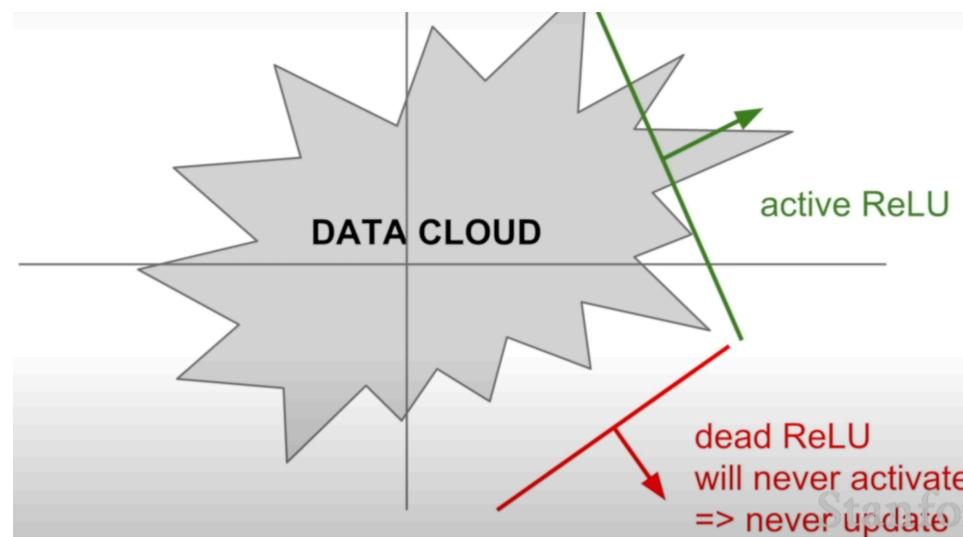
sigmoid/tanh에 비해 (약 6배) 빠르게 수렴

sigmoid보다 생물학적으로 그럴듯하다고 함

output이 zero-centered 하지 않음

음수 영역에서 포화함

Dead ReLU 현상



좋지 않은 영역(Data cloud에서 벗어난 영역)에 있을 때 발생

→ ReLU가 활성화되지 않아 업데이트 되지 않음

발생원인

1. 초깃값이 좋지 않은 경우

난수로 설정된 초깃값이 포화에 취약하다면 Dead ReLU가 발생할 수 있음

(예방책으로 slight positive biases를 사용해 ReLU 뉴런의 초깃값 설정을 하기도 함)

2. 학습률이 너무 높은 경우

가중치 w 업데이트가 너무 큰 폭으로 진행되어 데이터 매니폴드에서 벗어날 수 있음

⇒ ReLU 뉴런이 비활동적이거나 데이터를 나타낼 수 없게 되어버림 (Dead ReLU)

Reaky ReLU

$$f(x) = \max(0.01x, x)$$

포화하지 않음

계산 비용 측면에서 매우 효율적임

sigmoid/tanh에 비해 (약 6배) 빠르게 수렴

기울기가 소실하지 않음

PReLU (Parametric Rectifier)

$$f(x) = \max(\alpha x, x)$$

파라미터 α 를 사용해 최적의 α 를 학습(backprop)으로 얻는 방식

→ 조금 더 유동적인 방법

ELU (Exponential Linear Units)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

ReLU의 모든 장점

평균이 0에 가까운 output

Leaky ReLU에 비하면 음수 영역에서 포화를 보이기에 노이즈에 강건함

: Leaky ReLU는 매우 큰 음수값(노이즈)도 훈련에 반영

지수함수의 계산량이 많음

Maxout "Neuron"

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

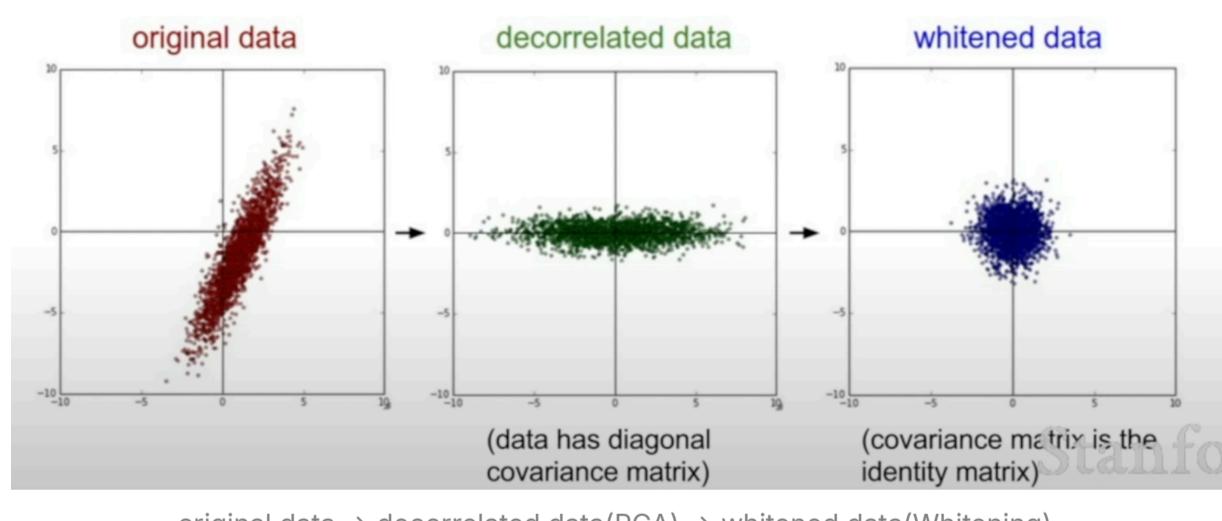
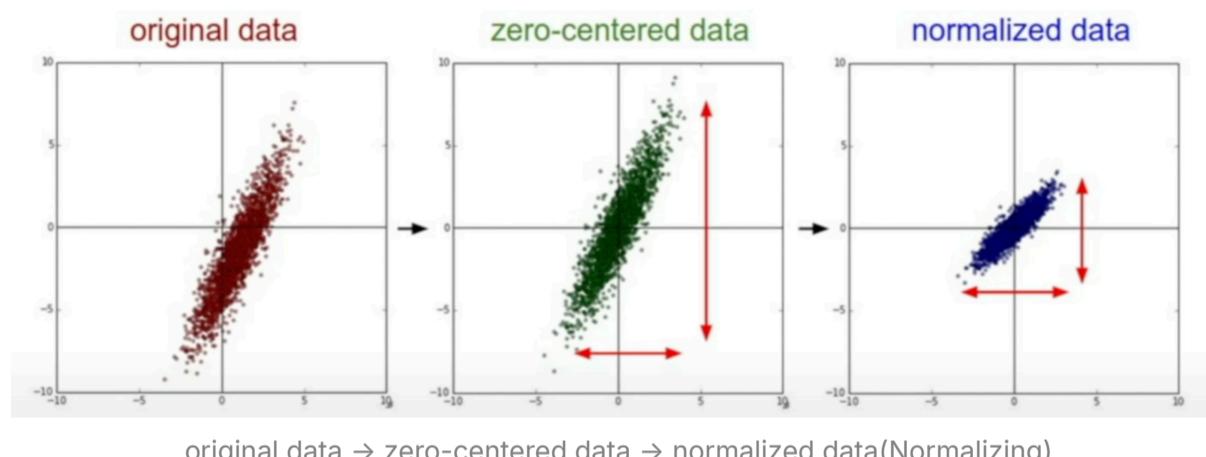
dot product의 기본적인 형태를 가지고있지 않음 → 비선형성

ReLU와 Leaky ReLU를 일반화함

선형 영역에 있고 포화하지 않으며 기울기 소실이 일어나지도 않음

파라미터가 w_1, w_2 로 계산량이 2배

Data Preprocessing



실제로 Image를 다룰 때: center only

image를 다루는 과제는 Conv 모델을 통해 본래 image에 대한 공간 구조를 얻는 것이 목적

⇒ 그러므로 center only = zero-centered 처리만 진행

(Normalize, PCA, Whitening 등의 복잡한 전처리 과정은 실제로 잘 쓰이지 않음)

[32,32,3] 차원의 image인 CIFAR-10 데이터를 예시로 생각해보면

- mean image를 추출 (e.g. AlexNet)
→ mean image = [32,32,3] 행렬
- channel 마다 mean 추출 (e.g. VGGNet)
→ 각 channel 마다 mean은 3개 숫자

Weight Initialization

Small Random numbers

(gaussian with zero mean and 1e-2(=0.01) standard deviation)

- code: `W = 0.01 * np.random.randn(D, H)`

⇒ 단순한 network에서는 잘 작동하지만, 깊은 network에서 문제 발생

→ forward, backprop을 거듭하면서 모든 activation이 0이 됨 (즉, 업데이트가 일어나지 않음)

Xavier initialization

- code(tanh): `W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)`
- code(ReLU): `W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2)`

Batch Normalization

: 총마다 batch의 activations를 정규분포 범위로 만들기 위해 사용하는 방법

→ 주로 FC와 Conv layer 뒤, activation function 앞에 들어감

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{(\text{Var}[x^{(k)}])}}$$

Procedure

Training time

1. 모든 Batch내 통계치, μ_d 과 σ_d^2 를 차원($d=\text{feature}$)에 대해서 계산

$$\mu_d = \frac{1}{B} \sum_{i=1}^B x_{i,d} \quad , \quad \sigma_d^2 = \frac{1}{B} \sum_{i=1}^B (x_{i,d} - \mu_d)^2$$

2. Normalize(정규화)

$$\hat{x}_{i,d} = \frac{x_{i,d} - \mu_d}{\sqrt{\sigma_d^2 + \epsilon}}$$

3. 각 feature마다 학습 가능한 파라미터인 γ , β 추가

→

*모델(network)이 원하는 표현 범위로 재조정

$$y_{i,d} = \gamma_d \cdot \hat{x}_{i,d} + \beta_d$$

- γ_d (scale): feature의 분산(variance) 조절
→ 정보가 너무 축소/확장되었을 때 다시 키우거나 줄이는 역할
- β_d (shift): feature의 평균(mean) 조절
→ 평균이 0으로 정규화된 걸 다시 임의의 위치로 이동시키는 역할

▼ * 모델(network)이 원하는 표현 범위

Normalization(정규화)는 학습을 쉽게 만들지만 표현을 제한함

: 단순하게 모든 입력이 항상 평균 0, 분산 1을 갖게 만듦

그래서 모델이 원하는 표현 범위란?

⇒ 다음 층(activation, conv, ...)이 학습하기 가장 좋고 유용한 출력값의 분포

→ γ, β 는 학습을 통해 각 feature에 대해 모델이 어떤 값 분포가 더 유용한지 조절

Examples

- 다음 층이 ReLU → 음수는 포화하므로 출력 분포가 양수 중심이면 좋음
- 특징 feature가 중요하다면 → 해당 feature 값이 크게 강조되어야 함 (γ 상승)
- 일부 feature가 억제되어야 한다면 → 해당 feature 평균, 분산을 줄여야 함 (γ, β 감소)

+) 파라미터 γ, β 를 통해 입력을 그대로 재복원 가능

BN의 기본 형태는

$$y^{(k)} = \gamma^{(k)} \cdot \frac{x^{(k)} - \mu^{(k)}}{\sqrt{\sigma^2(k)} + \epsilon} + \beta^{(k)}$$

여기서 다음과 같을 때,

- $\mu^{(k)} = E[x^{(k)}]$
- $\sigma^2 = \text{Var}[x^{(k)}]$
- $\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]} = \sqrt{\sigma^2(k)}$
- $\beta^{(k)} = E[x^{(k)}] = \mu^{(k)}$

$$y^{(k)} = \gamma^{(k)} \cdot x^{(k)} + \beta^{(k)} = \sqrt{\sigma^2(k)} \cdot \frac{x^{(k)} - \mu^{(k)}}{\sqrt{\sigma^2(k)}} + \mu^{(k)} = x^{(k)}$$

결과적으로 정규화 후 다시 scaling, shifting하면 원래 입력값 복원 가능

⇒ 이전의 값과 같은 표현력을 유지하기 때문에, BN은 표현력을 잃지 않음

4. 전체 평균/분산 running average로 누적해서 저장

⇒ running_mean $\rightarrow (1 - \alpha) \cdot \text{running_mean} + \alpha \cdot \mu_B$

Test time

Training 동안 저장한 *empirical mean/variance 사용

→ 출력값이 deterministic(결정적)해지고 성능이 안정적으로 유지

▼ *empirical mean

학습 과정에서 관측된 값들(미니배치들)의 평균

→ 모든 배치 평균의 누적값 = 데이터 분포를 근사한 전역 평균

Batch Normalization 특징

Network에서 gradient가 원활하게 작동 가능

더 높은 learning rates(학습률)을 사용 가능

초깃값 설정에 대한 높은 의존도 감소

Regularization 기법의 효과가 있고 dropout의 필요성을 약간 경감

- batch마다 mean, variance가 다르므로 입력은 같아도 출력이 약간 달라져 noise(jitter)처럼 작용 \Rightarrow overfitting 방지(Regularization 효과)

Babysitting the Learning Process

Step 1: Preprocess the data

Step 2: Choose the architecture

Step 3: Check the loss works properly

Step 4: Train

- 우선 training data에서 적은 양의 sample을 추출해 훈련이 제대로 일어나는지 확인해보기
- Regularization을 조금 적용시켜 loss가 적어지도록 하는 learning rate 찾기
 - Loss가 줄어들지 않는다면 \rightarrow LR이 너무 작음
 - Loss가 발산한다면(NaN) \rightarrow LR이 너무 큼

Hyperparameter Optimization

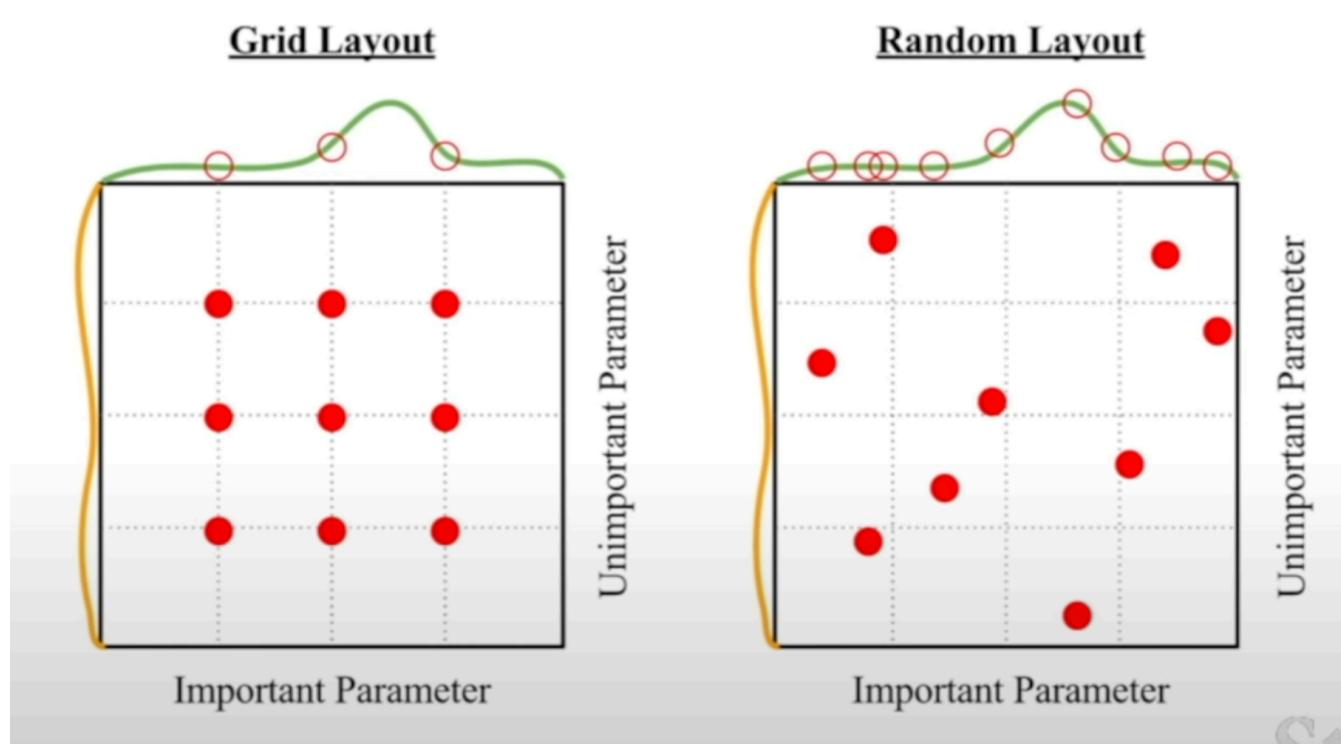
Cross-validation strategy

우선 적은 횟수의 epochs를 돌려서 hyperparam의 대략적인 범위 잡기

\rightarrow loss가 기준 값에 대비해 3배 이상 커지면 훈련 도중 발산하는 것이므로, 빠르게 훈련 중단

대략적인 범위에서 충분히 epochs를 반복해 가장 좋은 hyperparam 찾기

Random Search vs. Grid Search



Grid Search는 한정된 영역만 찾을 수 있음. Random search가 더 효율적인 방식

Hyperparameters 종류

모델 구조

Learning rate와 전략(decay schedule, update type)

Regularization

시각화를 통해 Hyperparameter 최적화

