

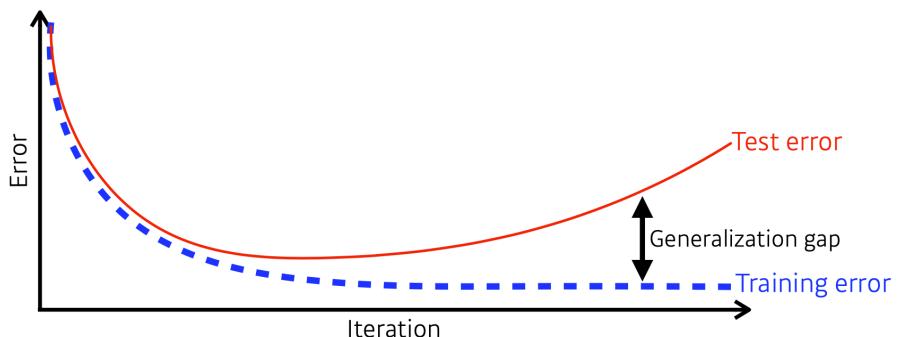
Optimization(최적화)

- **Gradient Descent**

First-order(일차미분값만 사용) iterative optimization algorithm for finding a local minimum of a differentiable function

- **Generalization**

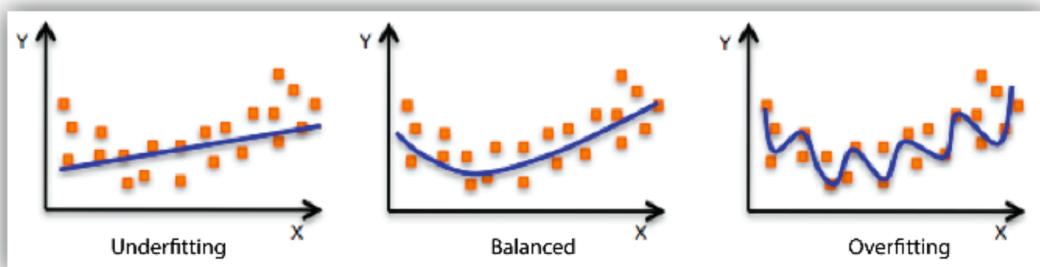
How well the learned model will behave on unseen data



캡이 작을 수록 well generalized

학습데이터 성능이 안 좋으면 generalization performance가 높아도, 테스트 데이터에서의 성능 보장 X
=> generalization performance가 높다는 게 네트워크 성능 좋은 것 X

- **Under-fitting <=> over-fitting**



가장 이상적인 방향: **balanced-fitting**과 동시에 **generalization performance** 높이기

- **Under-fitting**

학습을 마친 후, 모델의 성능이 기대치보다 떨어지는 현상

이상적인 결정경계(Decision Boundary)에 비해 지나치게 단순한 경우, 학습데이터조차 제대로 학습 못한 상황

- 발생 이유

작은 학습 반복 횟수

학습 데이터 부족

데이터의 feature에 비해 간단한 모델

- **Over-fitting**

학습을 마친 후, 새로운 데이터에 대해 그 결과가 매우 다르게 나오는 현상

이상적인 결정경계(Decision Boundary)보다 지나치게 복잡한 경우, 학습데이터에 과하게 학습된 상황

- 발생 이유

- 편중된 학습데이터

- 무분별한 noise 수용

- 너무 많은 데이터 feature로 모델 복잡

- 충분한 학습데이터 확보, 모델 및 하이퍼파라미터 조정 => under-fitting 해결
 - 다양한 학습데이터 확보, regularization => over-fitting 해결
 - trade-off 관계를 가지므로 적절한 파라미터 설정이 중요

- **Cross-validation(교차검증)**

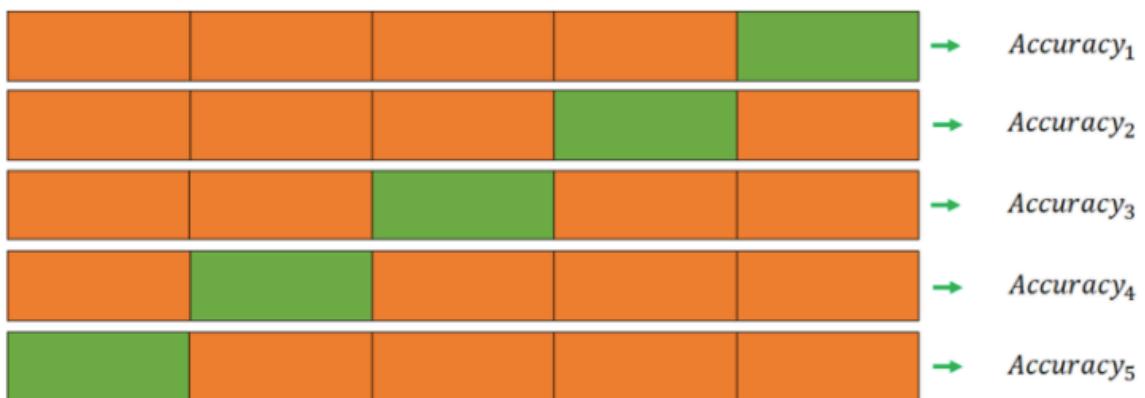
- 특정한 테스트 데이터셋에 과적합 문제, 테스트 데이터가 고정되어 있기 때문

- => 데이터 중 일부분을 테스트 데이터로 두지 않고 데이터의 모든 부분 사용, 교차 검증

- => 성능 검증

- Model validation technique for assessing how the model will generalize to an independent (test) data set

- **K-fold Cross-validation**



$$Accuracy = \text{Average}(Accuracy_1, \dots, Accuracy_k)$$

- 전체 데이터 셋 K등분의 부분집합으로 분할

- K - 1개의 부분집합은 학습 데이터셋, 나머지 1개의 부분집합은 테스트 데이터셋 할당

- 교차 검증 총 K번만큼 반복

- **Hold-out Validation**

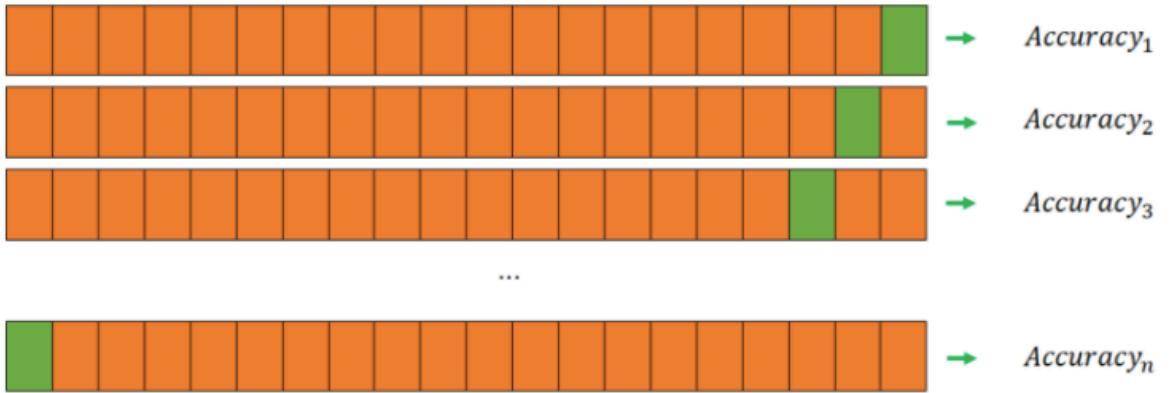


- 전체 데이터셋을 학습 데이터셋과 테스트 데이터셋으로 나눔

- 분리된 학습 데이터셋에서 다시 검증 데이터셋을 따로 떼어내어 교차 검증

- 장점: 교차 검증을 한 번만 진행 => 계산 시간 적음

- **Leave-one-out Cross-validation**



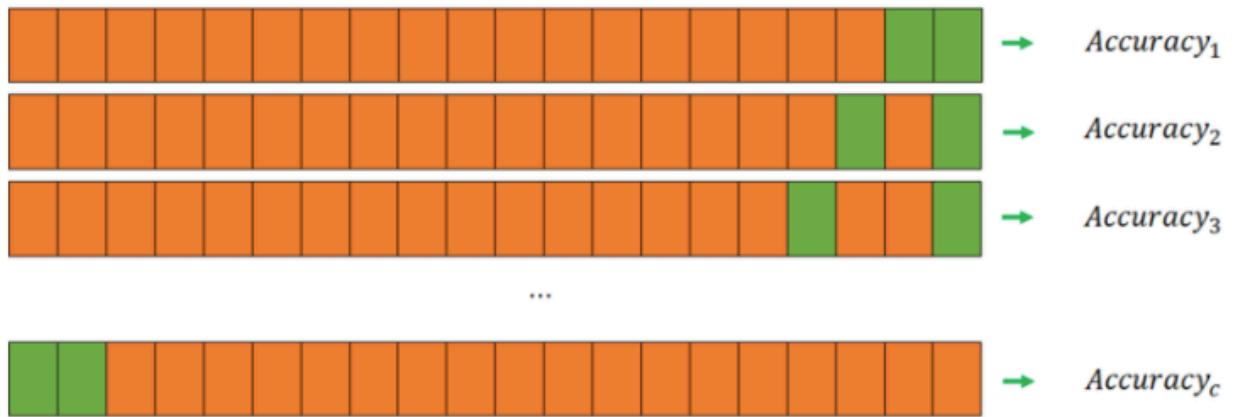
$$\text{Accuracy} = \text{Average}(\text{Accuracy}_1, \dots, \text{Accuracy}_n), \quad \text{where } n = \# \text{ of data sample}$$

전체 N개의 샘플 데이터셋을 N-1개의 학습 데이터셋과 1개의 테스트 데이터셋으로 나눔

총 N번만큼 교차 검증 반복

단점: 계산량이 많음

- **Leave-p-out Cross-validation**



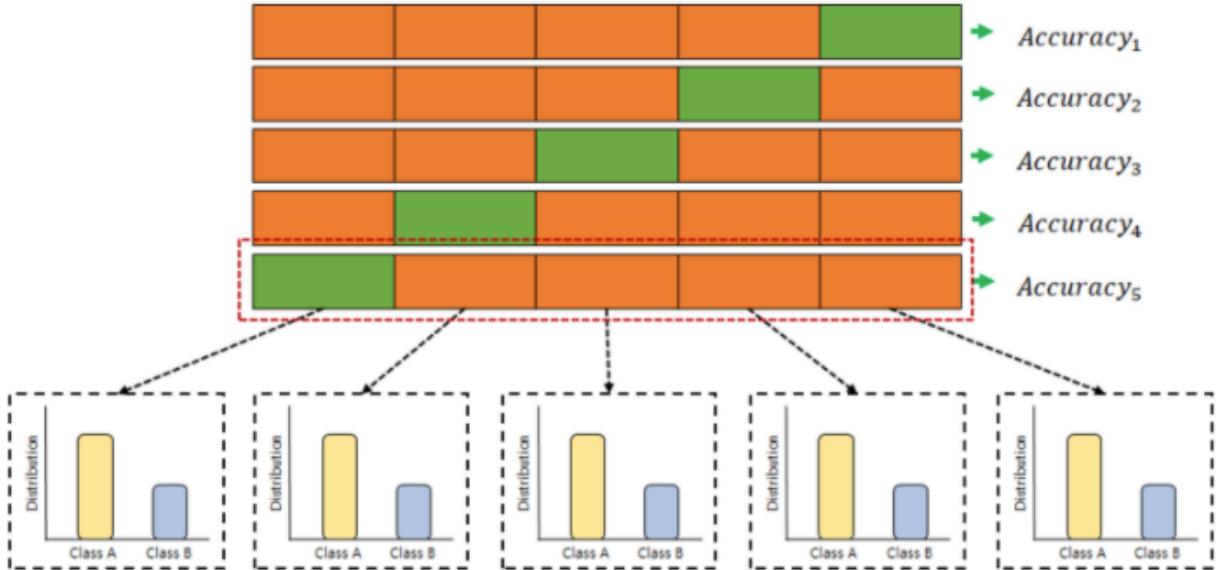
$$\text{Accuracy} = \text{Average}(\text{Accuracy}_1, \dots, \text{Accuracy}_c), \quad \text{where } c = \# \text{ of combinations of data sample}$$

전체 N개의 샘플 데이터셋을 N-p개의 학습 데이터셋과 p개의 테스트 데이터셋으로 나눔

총 nCp번만큼 교차 검증 반복

Leave-one-out 교차 검증 기법보다 더 계산량 많음 => 교차 검증 반복 횟수를 늘리고자 할 때 사용

- **Stratified K-fold Cross-validation**

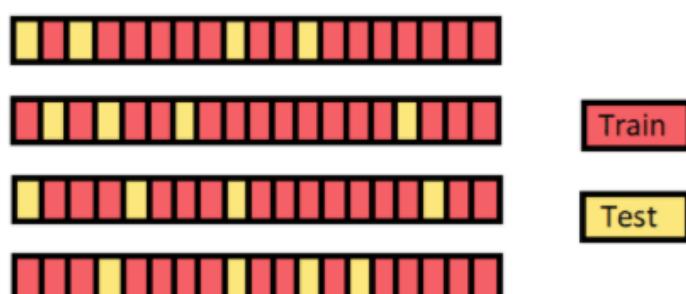


$$Accuracy = \text{Average}(Accuracy_1, \dots, Accuracy_k)$$

데이터 label 분포까지 고려해서 각 fold의 라벨분포가 전체 데이터의 라벨분포에 근사해서 훈련 및 검증 label 분포, 불균형한 상태 검증 => 오류 일으킴

=> 주로 classification에서 사용

- **Shuffle-split Cross-validation**

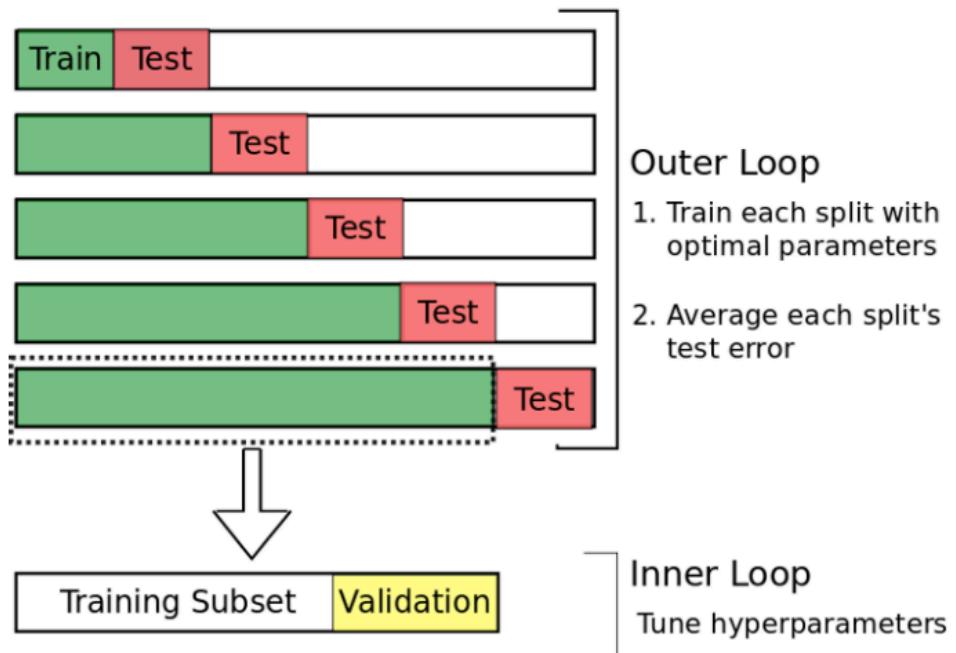


train dataset과 test dataset을 랜덤하게 나누고 반복

랜덤 => 어떤 fold는 여러 번 선택, 한번도 선택되지 않은 fold 발생

- **Nested Cross-validation**

Nested Cross-Validation



기존 교차검증 중첩

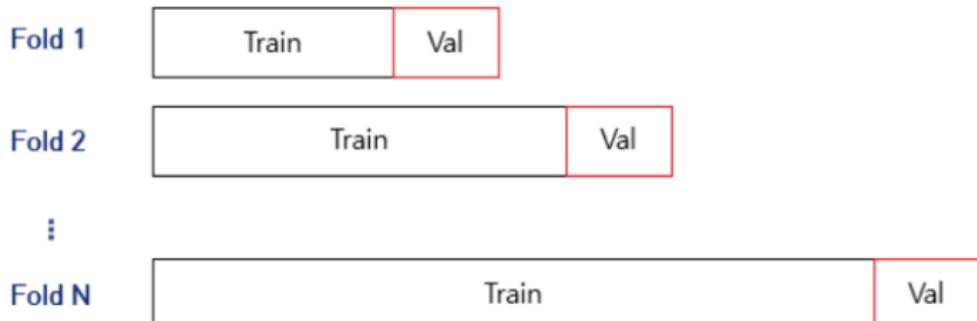
outer loop에서 기존 validation set 나누는 것 대신, testset을 다양한 fold로 나눠서 구성

outer loop에서 생성된 train set에 inner loop 적용

=> train set + validation set으로 나눠서 검증, 파라미터 튜닝

최적의 파라미터 통해 outerloop에서 생성된 test set으로 평가

- **Time-series Cross-validation**



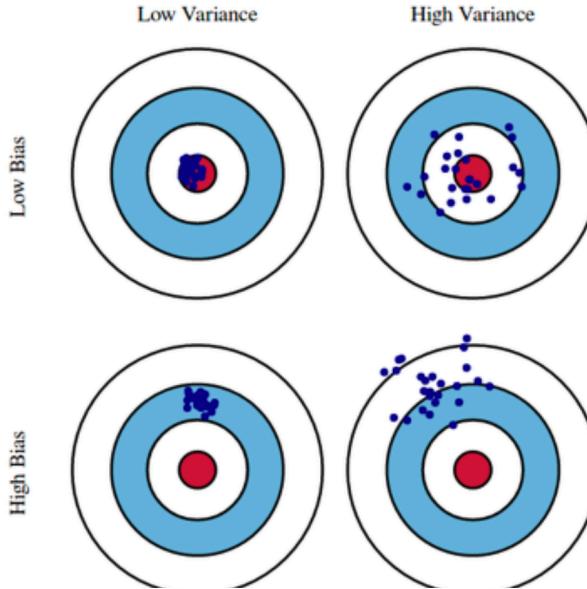
시계열 데이터 경우, 시간정보 결과에 영향

high variance 땐 => over-fitting 발생

=> validation / test set에 train set보다 미래 데이터 사용 => 합리적 검증

- **Bias and Variance**

- **Bias**: aim값에서 평균적으로 얼마나 벗어났는지
- **Variance**: 비슷한 입력에 대한 일관적인 출력 (분산)



- **Bias and Variance Tradeoff**

$$\text{cost} = \text{bias}^2 + \text{variance} + \text{noise}$$

$$\begin{aligned}
 \mathbb{E} [(t - \hat{f})^2] &= \mathbb{E} [(t - f + f - \hat{f})^2] \\
 \text{cost} &= \dots \\
 &= \mathbb{E} [(f - \mathbb{E}[\hat{f}]^2)^2] + \mathbb{E} [(\mathbb{E}[\hat{f}] - \hat{f})^2] + \mathbb{E} [\epsilon] \\
 &\quad \text{bias}^2 \qquad \qquad \text{variance} \qquad \text{noise}
 \end{aligned}$$

모델 학습 => cost 최소화

cost 를 최소화하는 것: bias 와 variance를 조절하는 문제

Noise: 데이터가 가지는 본질적인 한계치 이기 때문에 irreducible error

bias/variance : 모델에 따라 변하는 것이기에 reducible error

bias / variance => trade-off 관계를 가지므로 적절한 파라미터 설정이 주요문제

- **Bootstrapping**

Any test or metric that uses random sampling with replacement

고정된 데이터에 대해 random sampling with replacement 적용

=> 생성된 데이터 또는 이 데이터를 기반으로 만들어진 model/metric 생성

- **Bagging (Booststrapping aggregating)**

Multiple models are being trained with bootstrapping

주어진 데이터에서 여러 개의 bootstrap 이용, 각각의 모델 생성 후 최종 모델을 결정하는 방법

- 데이터로 부터 여러번의 복원 샘플링

=> 예측 모형의 분산을 줄여 줌

=> 예측력을 향상

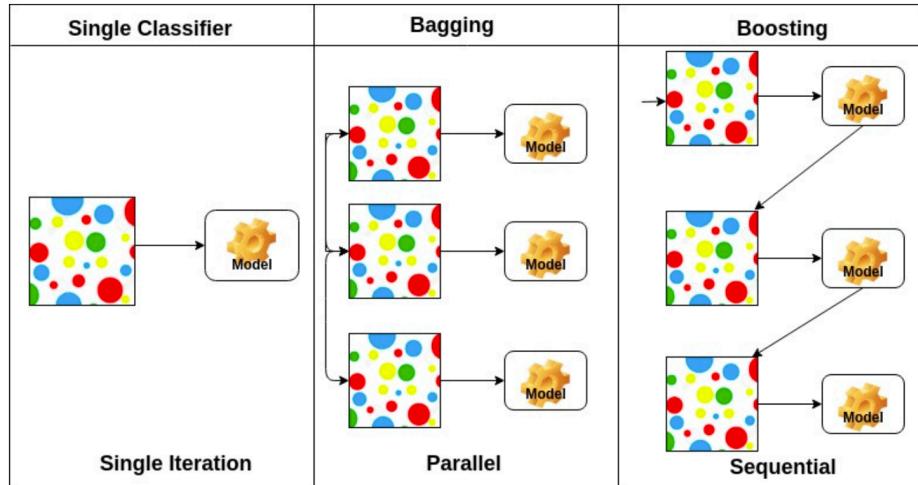
High variance,Low bias(Over-fitting) 인 모델에 사용

- **Boosting**

Focus on those specific training samples that are hard to classify

weak learners들을 결합하여 하나의 strong learner 만들

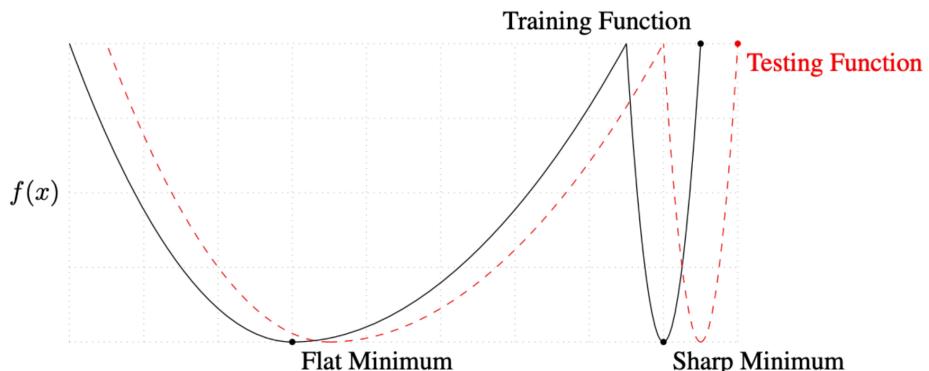
- 순차적으로 오분류된 개체들에게는 높은 가중치,
정확하게 분류된 개체들에게는 낮은 가중치 적용
=> 오분류된 객체들이 더 잘 분류하는 것



• Gradient Descent Methods

- BGD (Batch Gradient Descent) / 배치 전체
- SGD (Stochastic Gradient Descent) / 배치 1
- MSGD (Mini-batch Gradient Descent) / 배치 사용자 지정

• Batch-size Matters



batch: large => minimizers: sharp

batch: small => minimizers: flat

=> 작은 배치 사이즈가 더 학습이 잘됨, generalization performance 더 높음

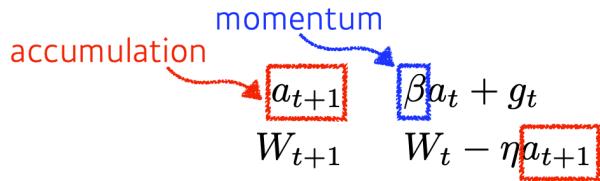
• Gradient Descent Methods

- Stochastic gradient descent

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

Learning rate Gradient

- Momentum (관성)



관성 존재 => gradient 요동쳐도 어느 정도 꾸준히 학습 가능

과거의 누적 gradient v_t (Accumulation)에 momentum μ 을 곱한 momentum step μv_t 을 이용해, 현재 gradient $\nabla f(\theta_t)$ 조정

보통 모멘텀 $\mu = 0.9$ 사용



- SGD가 oscillation(진동)을 겪을 때, 중앙의 최적점을 이동하는 힘을 줌
=> SGD에 비해 상대적으로 빠르게 이동 가능
- 기존에 이동했던 방향에 모멘텀 있음
=> local minima 빠져나오는 효과 기대 가능

○ Nesterov accelerated gradient(NAG)

The diagram illustrates the Nesterov accelerated gradient (NAG) update rule. It shows the lookahead gradient being used to calculate the next momentum step $\beta a_t + \nabla \mathcal{L}(W_t - \eta \beta a_t)$.

- $\nabla \mathcal{L}(W_t) = g_t$
- $-\eta \beta a_t$: momentum에 대한 피드백, gradient 값이 +/-로 바뀔 때 급격하게 바뀌도록 함

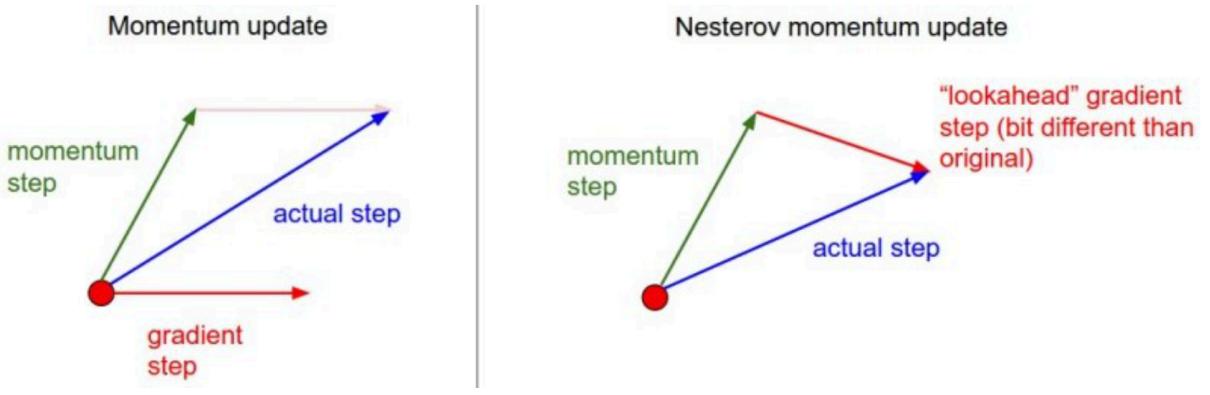
현재 위치의 gradient $\nabla f(\theta_t)$ 가 아닌 현재 위치에서 만 μv_t 큼 이동한 gradient $\nabla f(\theta + \mu v_t)$ 이용

momentum step 을 적용한 위치에서의 gradient $\nabla f(\theta + \mu v_t)$ (lookahead gradient step) 구한 후 업데이트

momentum이 local minimum에 converge하지 못하는, 느린 경우 예방

lookahead gradient step을 통해 어떤 방식으로 이동할지 결정

=> 유동적 이동 가능 => 기존의 momentum 방식에 비해 효과적



○ Adagrad(Adaptive Gradient)

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

for numerical stability

Sum of gradient squares

- G_t : 변화량 제곱합을 역수로 넣음
- ϵ : zero division 예방

많이 변화한 파라미터 일수록 적게 이동, 적게 변화한 파라미터 일수록 많이 이동하게 adapts

학습을 진행하는 동안: 보통 step size=0.01 정도를 사용한 뒤, step size decay 등을 신경 쓰지 X

- 학습이 긴 시간 진행될 경우 G_t 는 계속 증가

$$\Rightarrow \frac{\eta}{\sqrt{G_t + \epsilon}} \text{이 작아지는 문제 발생}$$

\Rightarrow 결국 거의 움직이지 않게 되는 문제 존재

○ Adadelta(Adaptive Delta)

$$G_t = \gamma G_{t-1} + (1 - \gamma) g_t^2$$

$$H_t = \gamma H_{t-1} + (1 - \gamma) (\Delta W_t)^2$$

$$W_{t+1} = W_t - \frac{\sqrt{H_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} g_t$$

EMA of gradient squares

EMA of difference squares

No learning rate

기존 Adagrad 보완

- 학습이 길어짐에 따라 **learning rate decay** 문제 해결
- 직접 **global learning rate**를 결정해야하는 문제 해결

EMA(Exponential Moving Average)를 이용해 최신 gradient 반영

\Rightarrow learning rate decay 문제 해결, 메모리 문제 해결

monotonically decreasing property 예방

○ RMSprop

An unpublished, adaptive learning rate method proposed by Geoff Hinton in his lecture.

EMA of gradient squares

$$G_t = \gamma G_{t-1} + (1 - \gamma) g_t^2$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

stepsize

기존 Adagrad 보완

EMA를 통해 최신의 gradient 반영

- **Adam**

Adaptive Moment Estimation (Adam) leverages both past gradients and squared gradients.

=> **adaptive + momentum**

Momentum

EMA of gradient squares

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

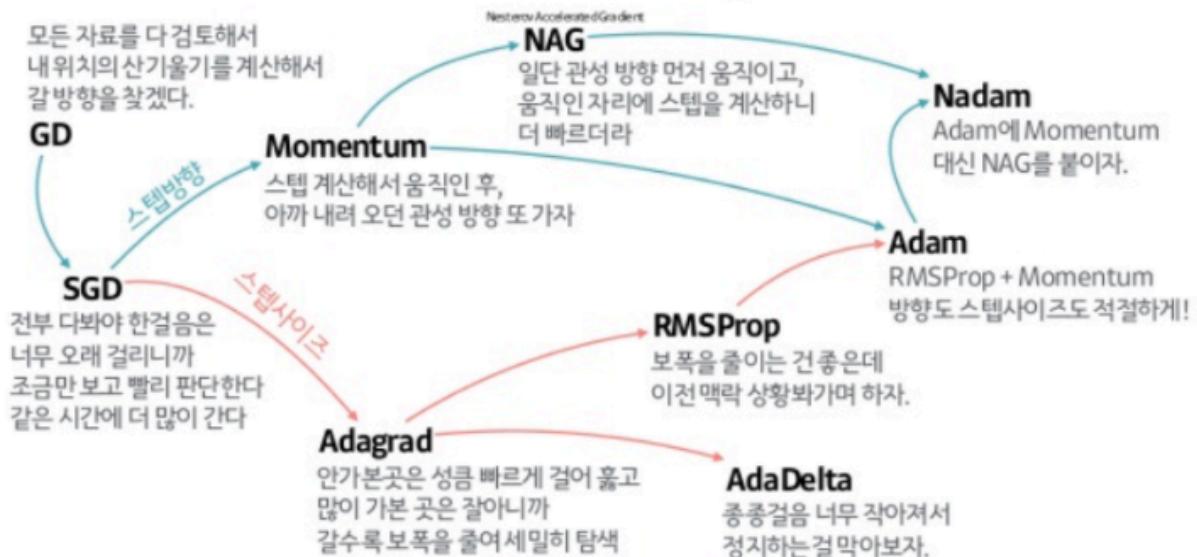
$$W_{t+1} = W_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} m_t$$

Stepsize

Momentum + Adaptive 방식으로 적절한 step size & step 방향 결정

- 보통 $\beta_1 = 0.9$ 로는 0.9, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ 정도의 값 사용

산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



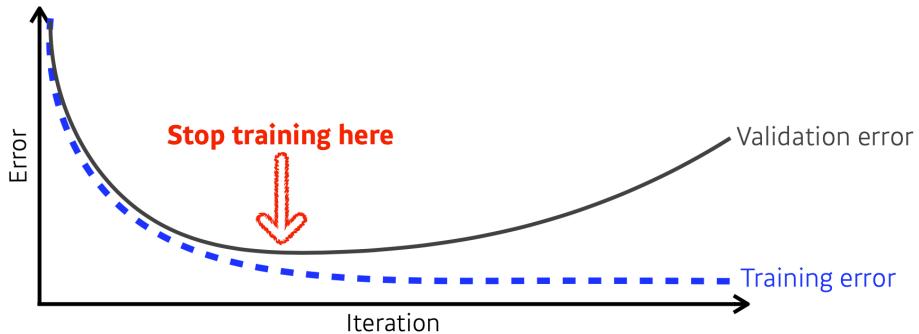
출처: 하용호, 자습해도 모르겠던 딥러닝, 머리속에 인스톨 시켜드립니다

- **Regularization**

Over-fitting을 막기 위한 regularization 종류

- **Early stopping**

training 중 validation error가 증가하는 부분에서 early stop



- 주의: loss 값이 상하로 움직이는 경우 생기므로, 직접 학습의 loss 만을 비교해서 종료하면 X
=> 일정 epoch 동안 계속해서 loss 가 증가 시, 학습 중단

- Parameter norm penalty(Weight Decay)

weight 숫자 작게 규제

=> add sommthness to the function space

Parameter Norm Penalty

$$\text{total cost} = \text{loss}(\mathcal{D}; W) + \frac{\alpha}{2} \|W\|_2^2$$

- L1-norm penalty => Ridge model

$$\begin{aligned} J(\theta) &= Loss + \frac{\lambda}{2m} \sum_{\theta} \theta^2 \\ \theta &:= \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \\ &= \theta - \alpha \frac{\partial Loss}{\partial \theta} + \frac{\lambda}{m} \theta \\ &= \left(1 - \frac{\alpha \lambda}{m}\right) \theta - \alpha \frac{\partial Loss}{\partial \theta} \end{aligned}$$

θ 가 작아지는 방향으로 진행

=> weight decay 를 통해 outlier 의 영향을 적게 받도록 만들어 generalization performance를 높이는 것

- L2-norm penalty => Lasso model

$$\begin{aligned} J(\theta) &= Loss + \frac{\lambda}{m} \sum_{\theta} |\theta| \\ \theta &:= \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \\ &= \theta - \frac{\lambda \alpha}{m} sgn(\theta) - \alpha \frac{\partial Loss}{\partial \theta} \end{aligned}$$

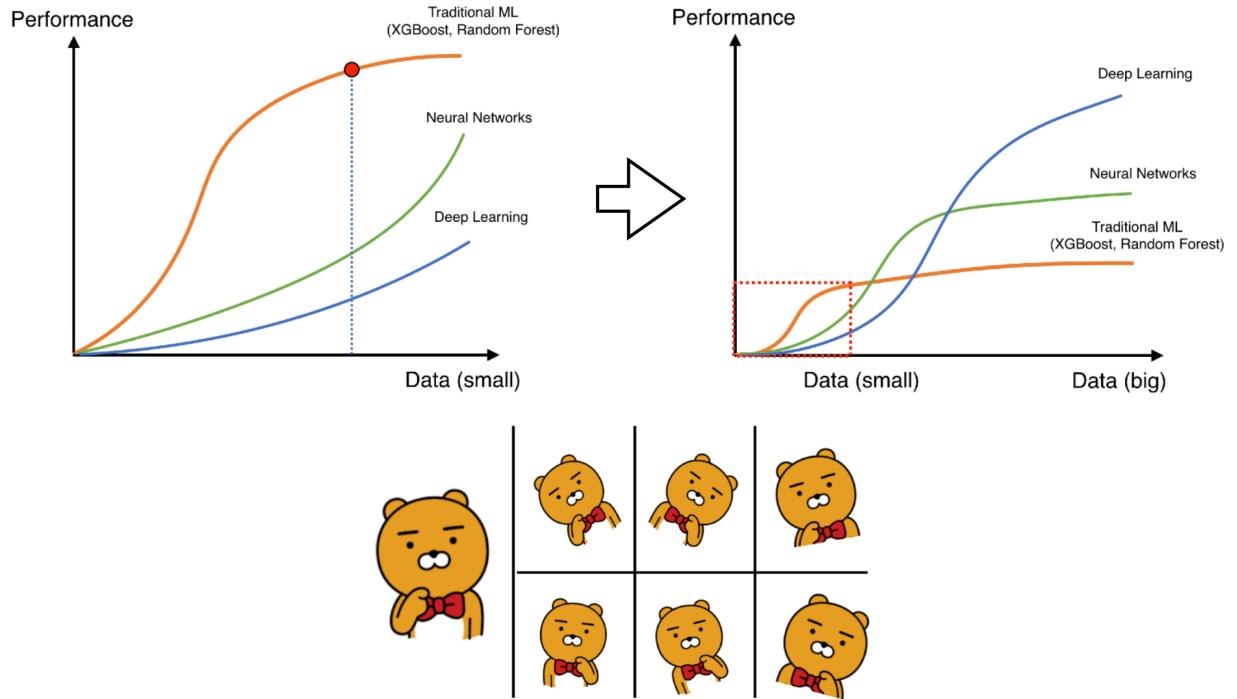
θ 값 자체를 줄이는 것이 아닌 θ 부호에 따른 상수값을 빼주는 방향으로 진행

=> 상수값을 빼줘 작은 θ 들은 거의 0으로 수렴하게 되어 특정 중요한 θ 들만 남음

- Data augmentation

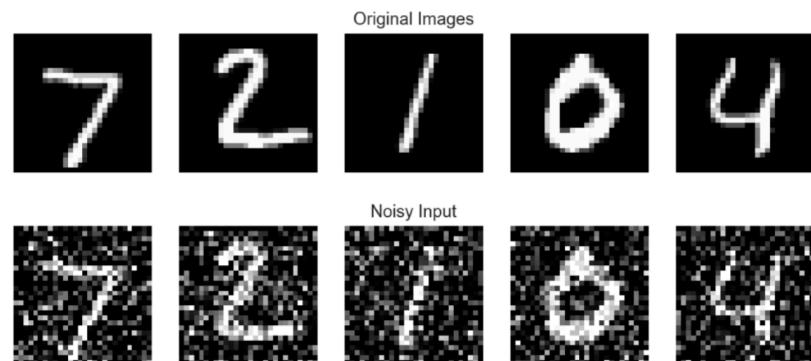
label preserving augmentation

class 에 영향을 끼치지 않는 선에서 Data Augmentation은 필수적



- **Noise robustness**

Add random noise inputs or weights



- **Label smoothing**

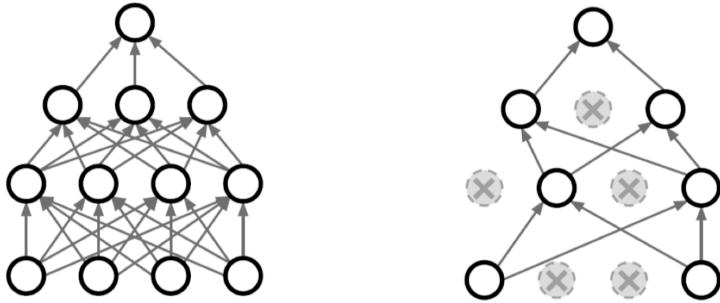
| | ResNet-50 | Mixup [48] | Cutout [3] | CutMix |
|-------|-----------|--------------------|------------|--------------------|
| Image | | | | |
| Label | Dog 1.0 | Dog 0.5 Cat 0.5 | Dog 1.0 | Dog 0.6 Cat 0.4 |

Hard target(label) 을 soft target(label) 로 바꾸는 것

label smoothing을 통해 mislabeling 데이터 다룸 + generalization performance를 올리는 방법

사진 합성, 알맞은 레이블링 => 성능 향상

- **Dropout**



Randomly set some neurons to zero (무작위로 일부 노드들을 생략하여 학습을 진행)

드롭아웃 $p = 0.5 \Rightarrow 50\%$ 의 뉴런 비활성화

각각 뉴런들이 좀 더 robust한 feature를 잡을 수 있음

\Rightarrow 모델의 generalization performance 올라감

over-fitting을 막음

매번 다른 형태의 노드로 학습 \Rightarrow 여러 형태의 네트워크를 생성하는 양상블 효과

- **Batch normalization**

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \ // \text{mini-batch mean}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2 \ // \text{mini-batch variance}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \ // \text{normalize}$$

$$y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(X_i) \ // \text{scale and shift}$$

internal covariate shift(네트워크의 각 층이나 Activation마다 input 의 distribution이 달라지는 현상)가 줄어듬

Learning rate 을 너무 높게 잡은 경우, gradient vanishing/exploding 가 발생

\Rightarrow Batch Normalization을 통해 parameter scale 영향 제거, learning rate를 크게 잡을수 있음, 빠른 학습을 가능

regularization 효과, Dropout와 같은 효과(Dropout의 경우 효과는 좋지만 학습속도가 다소 느려짐)

\Rightarrow Dropout을 제거, 학습속도 향상