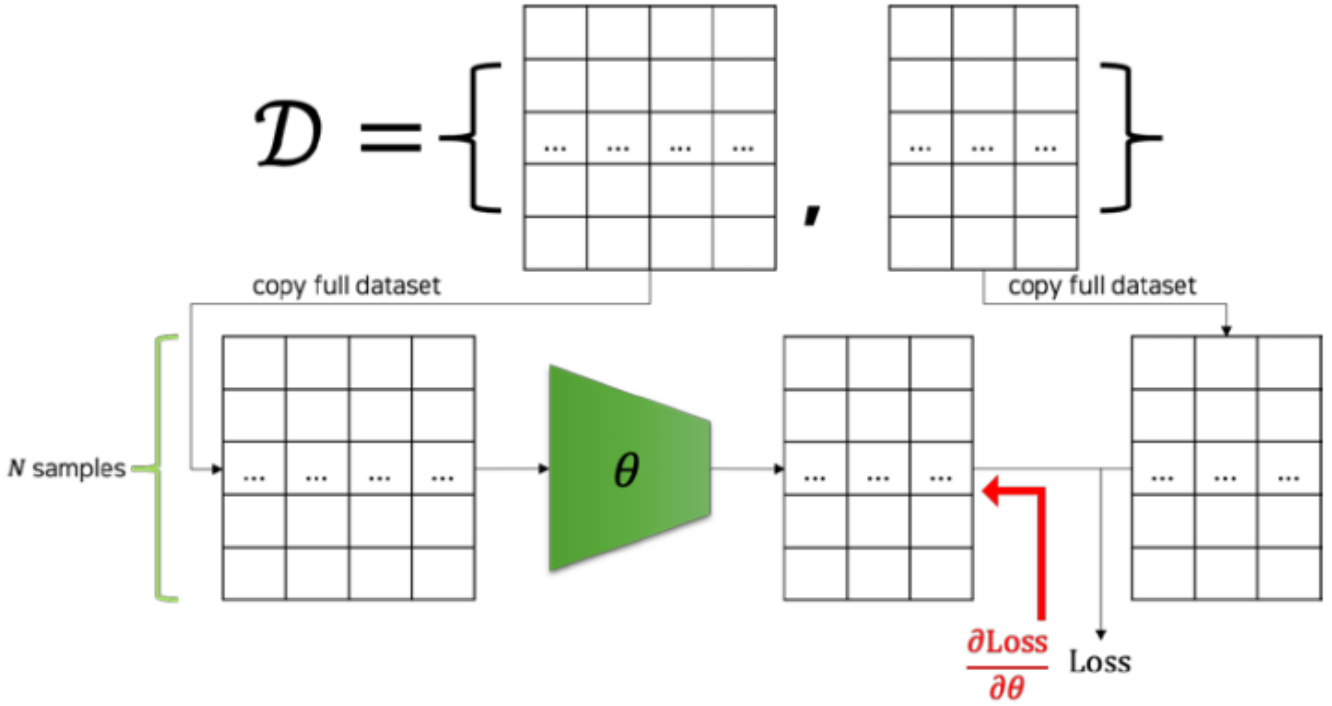
8. 최적화와 오버피팅 방지, 정규화

1. 확률적 경사하강법
2. 최적화
3. 오버피팅 방지
4. 정규화

1. 확률적 경사하강법

❖ 기존의 파라미터 업데이트 과정

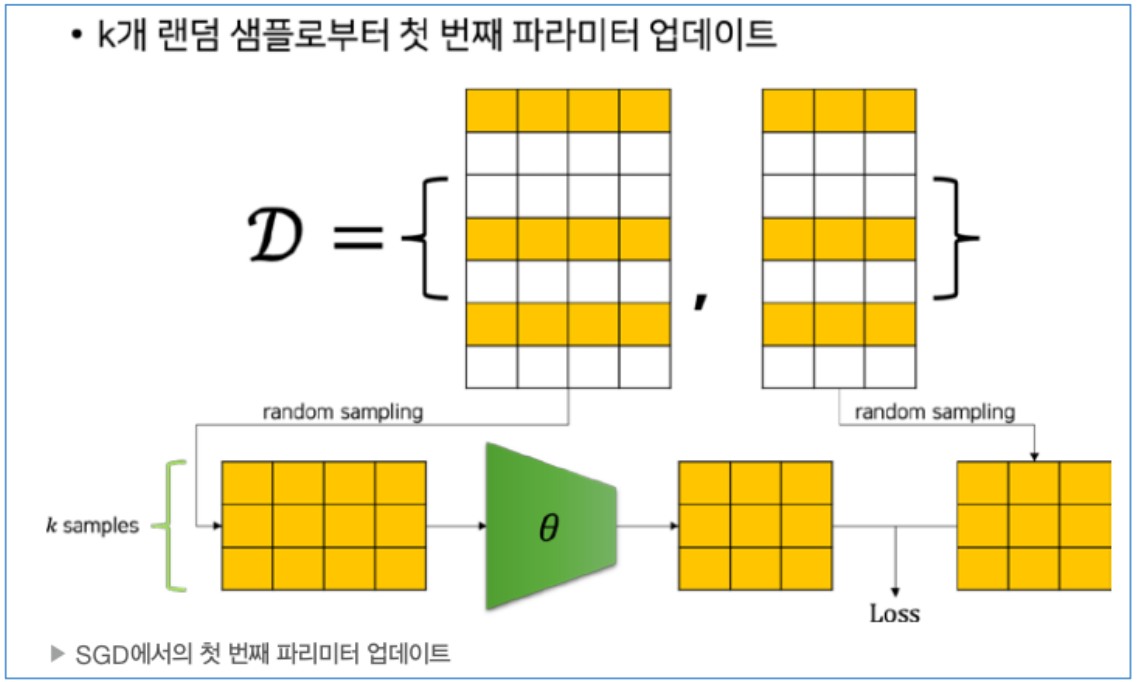
- 데이터셋의 모든 샘플들을 모델에 통과 시킨 후 손실 값 계산
- 데이터 셋이 클 경우 문제 발생 : GPU 메모리 한계로 큰 데이터셋 한번에 계산하기 어렵고, 학습속도 느림

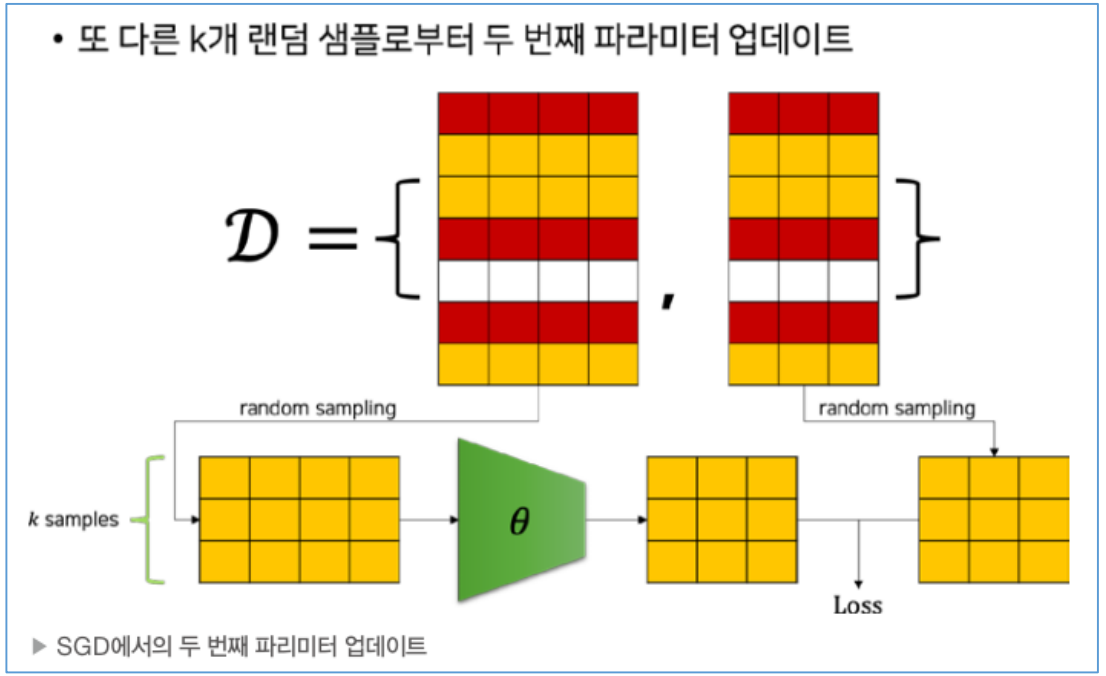


1. 확률적 경사하강법

❖ 확률적 경사하강법(Stochastic Gradient Descent, SGD)

- 전체 데이터셋을 모델에 통과 시키는 대신 랜덤 샘플링한 k개의 샘플을 모델에 통과시켜 손실 값을 계산하고, 파라미터 업데이트를 수행함.
- 비복원 추출 수행 : 한번 활용된 샘플은 모든 샘플이들 학습될 때까지 다시 학습에 활용되지 않음
- k개의 샘플 묶음을 미니배치라고 함





1. 확률적 경사하강법

❖ 확률적 경사하강법(Stochastic Gradient Descent, SGD)

- 전체 데이터셋의 샘플들이 전부 모델을 통과하는 것을 한번의 에포크(epoch)이라고 함
- 한 개의 미니배치를 모델에 통과시키 것을 이터레이션(iteration)이라고 함
- 셋이터셋 크기 N, 파라미터 업데이트(이터레이션) 횟수, 에포크 횟수(Epochs), 미니배치 크기 k 일 때 다음과 같은 관계를 가짐

$$\#\text{Iterations/Epoch} = \left\lceil \frac{N}{k} \right\rceil$$

$$\downarrow$$

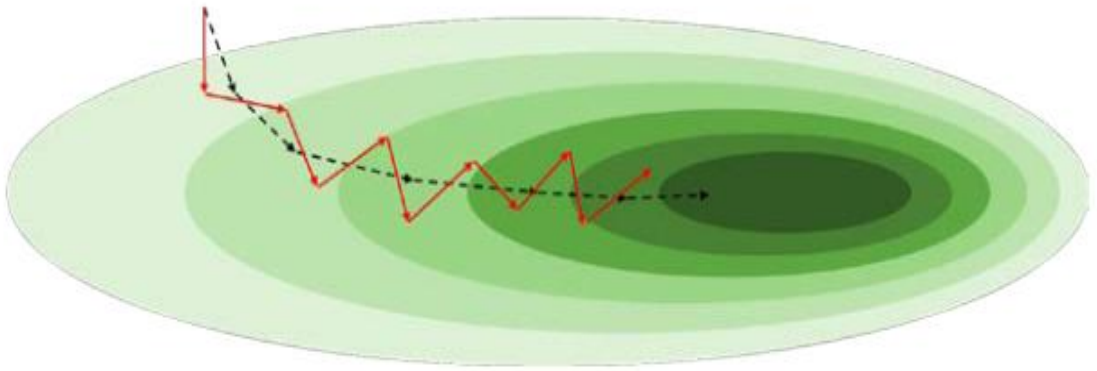
$$\#\text{Iterations} = \left\lceil \frac{N}{k} \right\rceil \times \#\text{Epochs}$$

- 미니배치:
  - 작을 수록 파라미터 업데이터 횟수가 늘어남,
  - 실무에서 GPU 메모리 허용 범위 안에서 크게 잡으나 4000 이상이면 성능 저하됨, 256~512 정도가 적당

1. 확률적 경사하강법

❖ 확률적 경사하강법(Stochastic Gradient Descent, SGD), 최소값 찾기

- 검은색 실선 : GD에 의한 최소 값 찾기
- 빨간색 실선 : SGD에 의한 최소 값 찾기
- 미니배치의 크기가 크면 실제 그라디언트와 비슷할 확률이 높고, 미니배치 크기가 작으면 실제 그라디언트와 달라질 확률이 높음



▶ SGD에서의 최소값을 찾는 과정

1. 확률적 경사하강법

❖ 미니배치 크기에 따른 SGD

- 미니배치가 너무 작은 경우
  - 미니배치의 분포가 전체 데이터셋 분포와 달라 편향을 가지게 될 가능성이 높음
  - 장점은 : 지역 최소점을 탈출할 수도 있음
  - 그래드언트에 심한 노이즈가 생길 수 있어 올바른 방향으로 학습되는 것을 저해함
  - 256 정도로 시작하여 메모리 허용 범위와 성능 저하가 없다면 크기를 늘리는 방향으로 튜닝해 나감

❖ 미니 배치 크기에 따른 파라미터 업데이트 횟수

- 한 에포크 내에서 파라미터 업데이트 횟수는 이터레이션 횟수와 같음
- 이터레이션이 많을 수록 신경망은 학습할 기회가 많아지나, 그래디언트에 노이즈 발생 가능
- 미니배치 작으면 그래디언트 방향이 정확해 질 수 있음, 신경망 학습률이 떨어짐
- 적절한 미니배치 크기 선택이 필요

SGD 실습

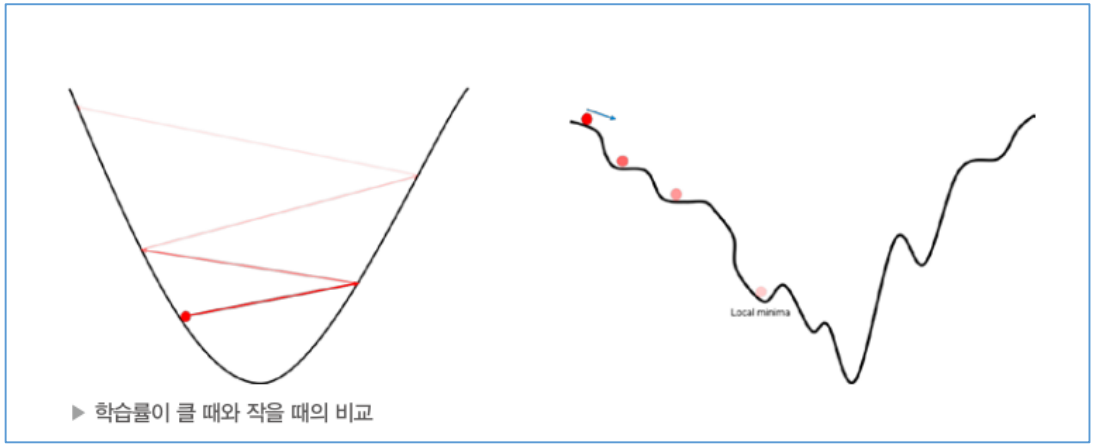
2. 최적화

❖ 하이퍼파라미터란?

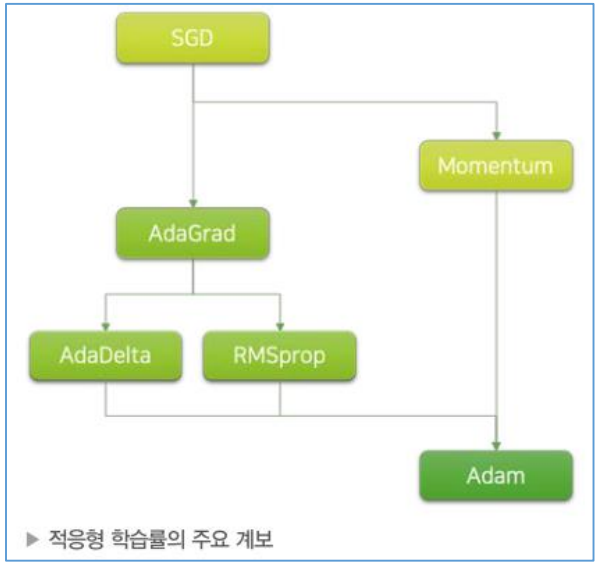
- 모델 성능에 영향을 미치지만 자동으로 최적화되지 않는 파라미터, 경험적 또는 휴리스틱(heuristic)한 방법을 통해 찾음, 대표적 하이퍼파라미터로 학습률, 미니배치 크기, 신경망의 깊이와 너비 등

❖ 적응형 학습률

- 학습률의 크기에 따른 특성과 동적 학습률 필요성
  - 학습률이 큰 경우 발산할 수 있으며, 학습률이 작은 경우 지역 최소점에 빠질 수 있음
  - 초반에 큰 학습율에 진행하다가 후반에 학습률을 작게 진행하도록 학습률을 동적으로 변경



▶ 학습률이 클 때와 작을 때의 비교

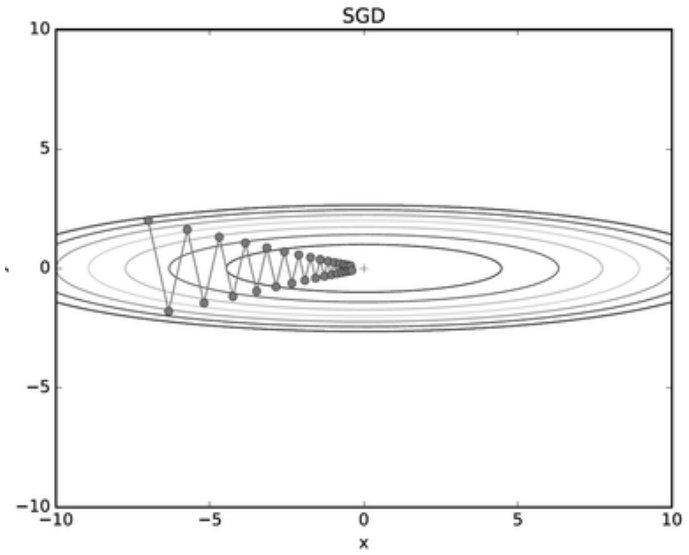


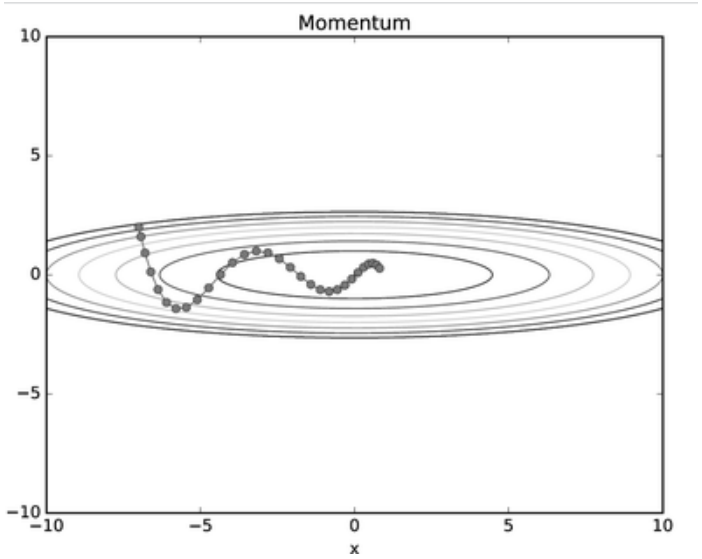
▶ 적응형 학습률의 주요 계보

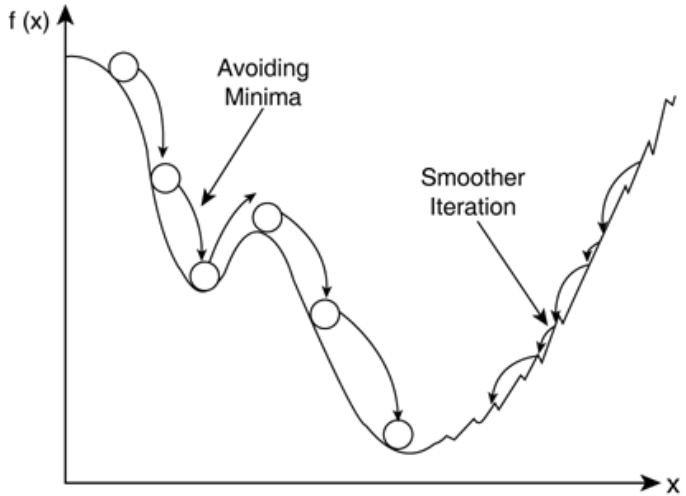
2. 최적화

❖ 모멘텀

- 모멘텀은 운동량을 의미하며 Momentum Optimizer는 매개변수의 이동에 속도를 부여하는 것을 의미
- 구슬을 떨어트리면 급한 경사에서는 더 빨라지듯이 매개변수의 변화에도 속도를 부여하는 것
- 모멘텀을 사용할 경우 x축은 항상 같은 방향을 가르키므로 양의 가속도로 작용하여 SGD에 비해서 지그재그의 비율이 줌. 그러나 y축은 번갈아 상충하여 속도가 안정적이지 못함
- 장점 : 지역 최소점을 쉽게 탈출, 그래디언트 노이즈 많은 경우에도 가속 시켜 줌







2. 최적화

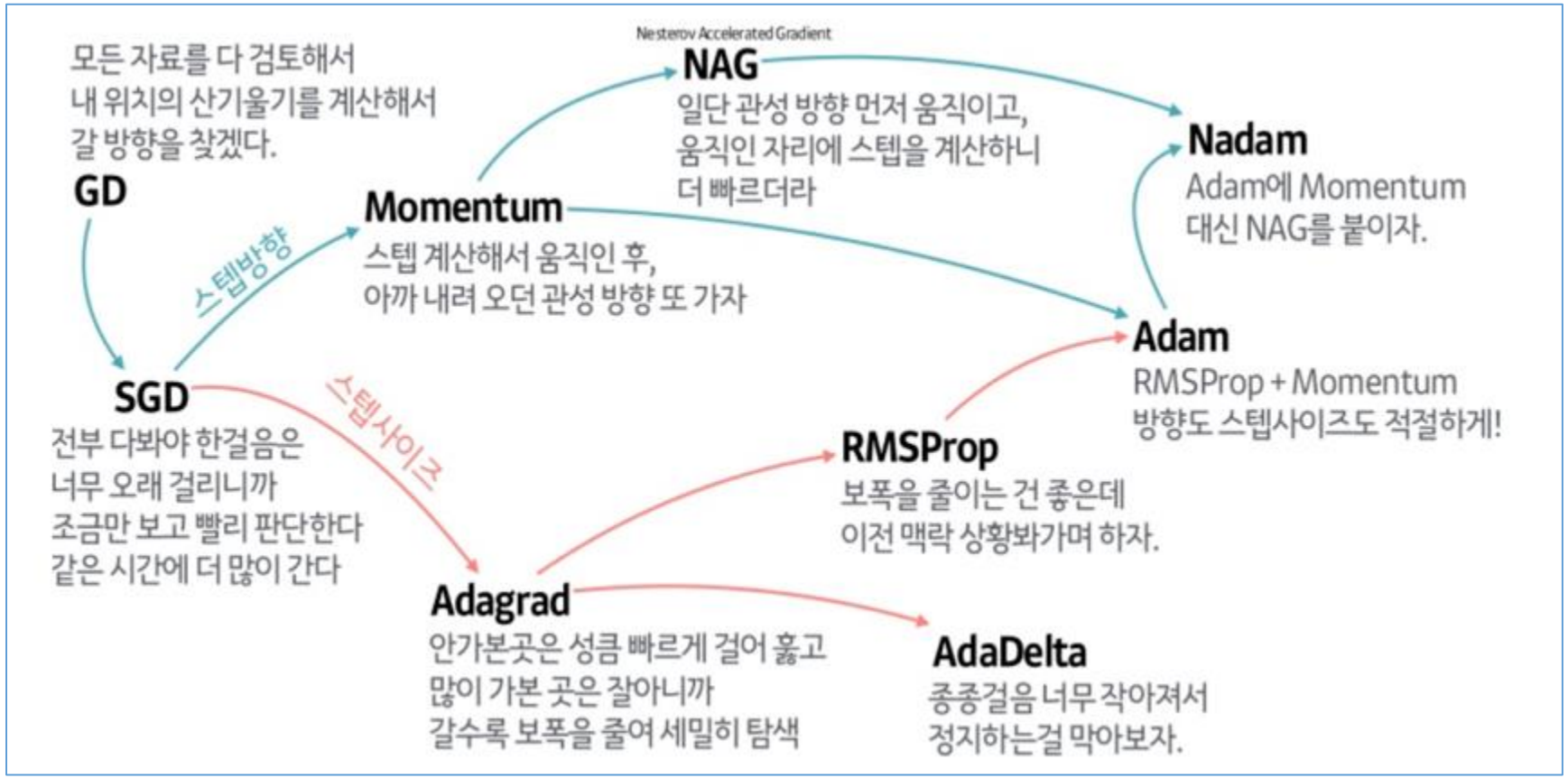
❖ 적응형 학습률

- 새로운 데이터셋과 새로운 모델 구조로 학습을 시작할 때 학습률을 고민하지 않고 기본 설정 값을 사용하더라고 상황에 따라 학습률이 자동으로 정해지는 형태

- 학습률 스케줄링
  - 학습초반은 학습률을 크게하고 후반으로 갈수록 미세한 가중치 파라미터 조정이 필요할 수 있기 때문 학습률을 감소 시킴
- 아드그래드 옵티마이저
  - 최초로 제안된 적응형 학습률 알고리즘
  - 가중치 파미터의 학습률은 가중치 파리미터가 업데이트될 수록 반비례하여 작아짐
- 아담 오티마이저
  - 진행하던 속도에 **관성**을 주고, 최근 경로의 곡면의 변화량에 따른 **적응적 학습률**을 갖은 알고리즘

2. 최적화

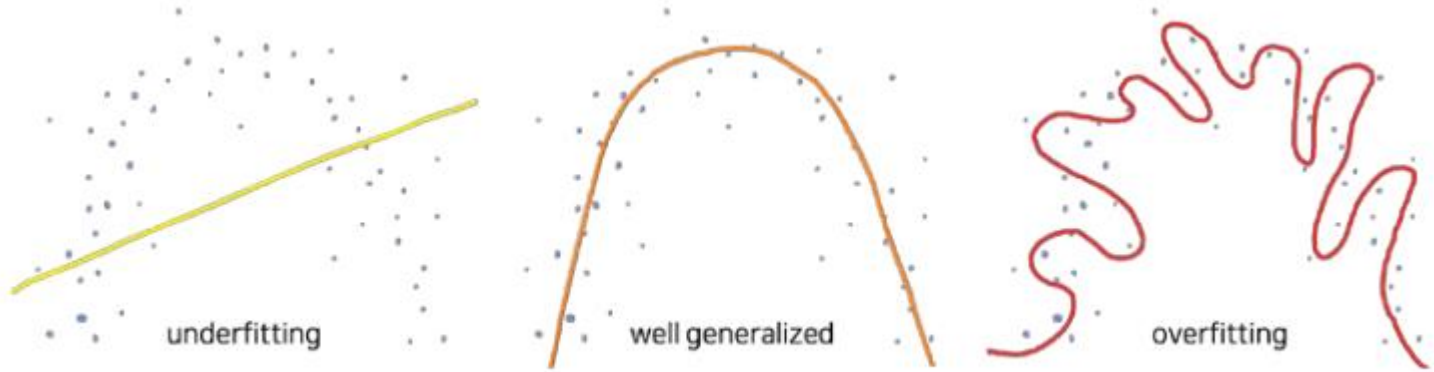
❖ 옵티마이저


모든 자료를 다 검토해서
내 위치의 산기울기를 계산해서
갈 방향을 찾겠다.
GD
스텝방향
SGD
전부 다봐야 한걸음은
너무 오래 걸리니까
조금만 보고 빨리 판단한다
같은 시간에 더 많이 간다
Momentum
스텝 계산해서 움직인 후,
아까 내려 오던 관성 방향 또 가자
Nesterov Accelerated Gradient
NAG
일단 관성 방향 먼저 움직이고,
움직인 자리에 스텝을 계산하니
더 빠르더라
Nadam
Adam에 Momentum
대신 NAG를 붙이자.
Adam
RMSProp + Momentum
방향도 스텝사이즈도 적절하게!
스텝사이즈
RMSProp
보폭을 줄이는 건 좋은데
이전 맥락 상황봐가며 하자.
Adagrad
안가본곳은 성큼 빠르게 걸어 훑고
많이 가본 곳은 잘아니까
갈수록 보폭을 줄여 세밀히 탐색
AdaDelta
종종걸음 너무 작아져서
정지하는걸 막아보자.

3. 오버피팅 방지

❖ 오버비팅과 언더피팅

- 오버피팅 : 학습 오차가 일반 오차에 비해서 현저하게 낮아지는 현상
- 언더피팅 : 모델이 충분히 데이터를 학습하지 못하여 오차가 충분하게 낮지 않는 현상



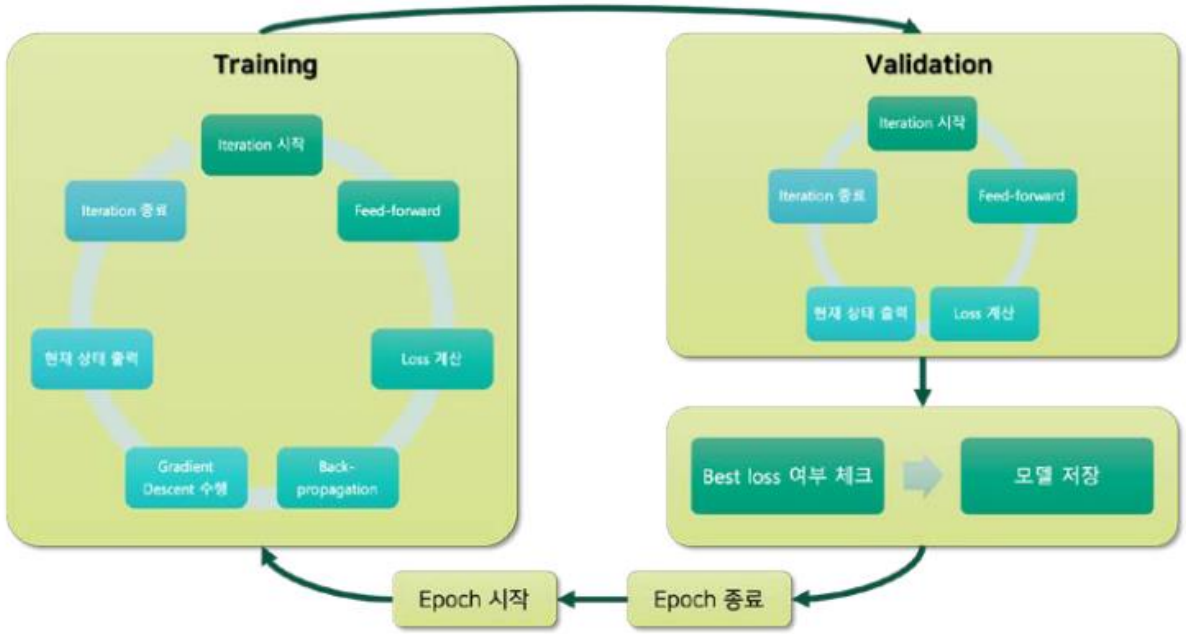
▶ 언더피팅 vs 일반화가 잘된 경우 vs 오버피팅

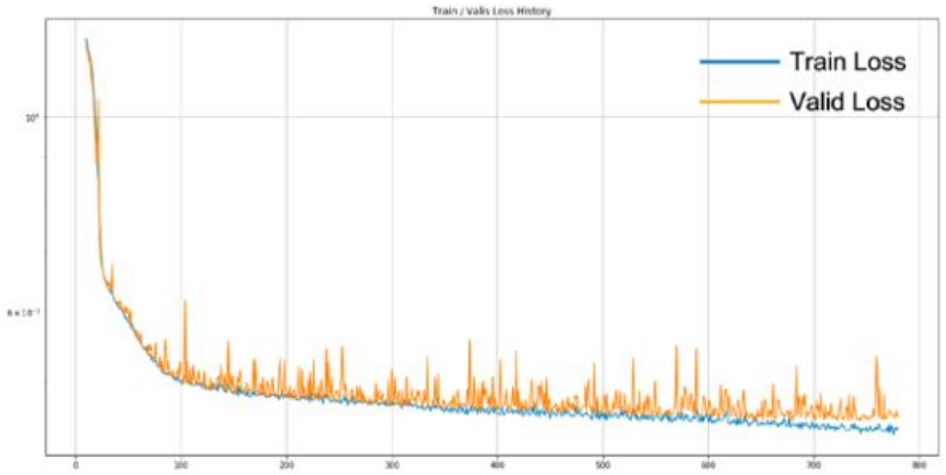
3. 오버피팅 방지

❖ 검증 데이터 셋

- 오버피팅 여부를 체크하기 위해 사용

▶ 학습 데이터 일부를 임의로 나누어 검증 데이터셋으로 구성





▶ 에포크에 따른 학습 손실 값과 검증 손실 값

3. 오버피팅 방지

❖ 테스트 데이터 셋

- 훈련된 모델을 평가하기 위한 데이터 셋

▶ 데이터 나누기(학습 데이터셋, 검증 데이터셋, 테스트셋)

	Train set	Valid set	Test set
Parameter	결정	**검증**	**검증**
Hyper-parameter		결정	**검증**
Algorithm			결정

▶ 나누어 구성한 각 데이터셋의 역할 비교

4. 정규화

❖ 오버피팅 복습

- 오버피팅이란 '학습 오차가 일반 오차에 비해 현저하게 낮아지는 현상 " 을 의미
- 학습데이터의 불필요한 편향이나 노이즈까지 학습하여 모델의 일반화 성능이 떨어지는 현상
- 검증 데이터셋과 테스트 데이터 셋을 도입하여 모델의 최적화 과정에 발생할 수 있는 오버피팅 현상을 방지
- 3가지 데이터셋에 대해서 가중치 파라미터와 하이퍼파라미터, 알고리즘의 결정과 검증 수행

▶ 학습 데이터셋, 검증 데이터셋, 테스트셋의 구분

범주	학습 데이터셋	검증 데이터셋	테스트 데이터셋
가중치 파라미터	결정	검증	검증
하이퍼파라미터		결정	검증
알고리즘			결정

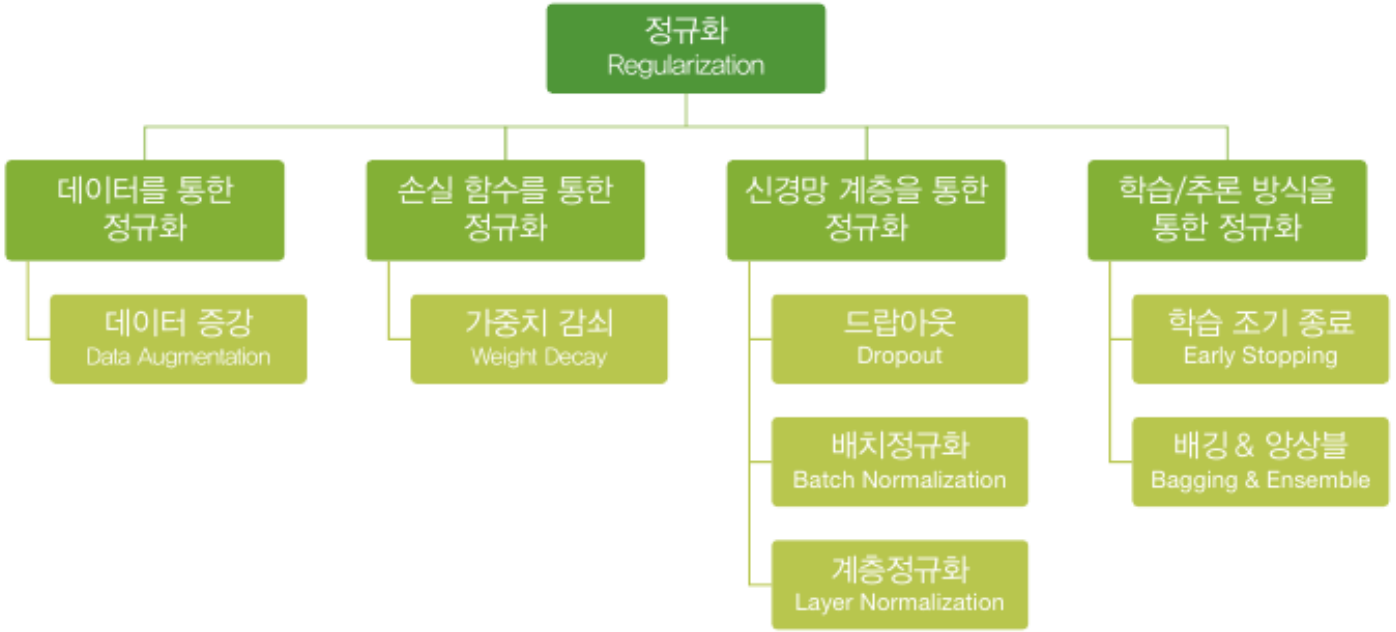
▶ 학습 데이터셋, 검증 데이터셋, 테스트셋의 역할 비교

4. 정규화

❖ 정규화란?

- 오버티팅을 늦추고 모델이 학습 데이터로부터 적절한 특징들을 학습하여 일반화 오차를 낮출 수 있는 기법

❖ 다양한 정규화 방법들



▶ 다양한 정규화의 방식

4. 정규화

❖ 가중치 감쇄(Weight Decay)

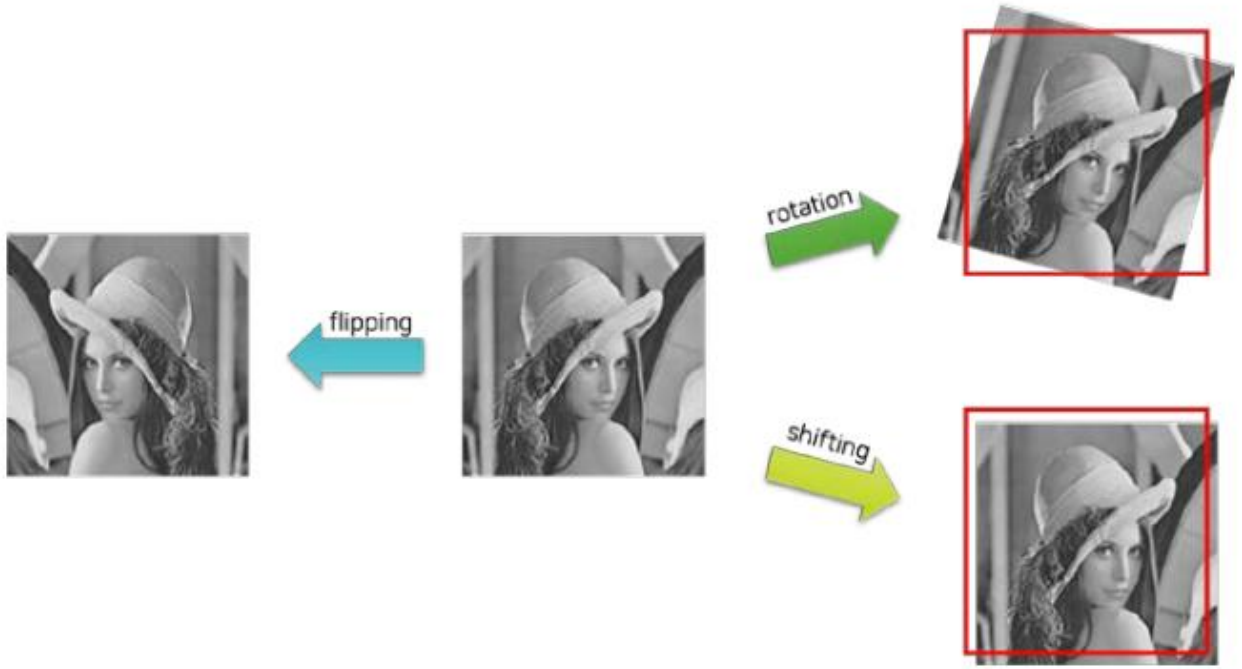
- 복잡한 모델이 간단한 모델보다 과적합될 가능성이 높음
- 간단한 모델은 적은 수의 매개변수를 가진 모델
- 복잡한 모델을 좀 더 간단하게 하는 방법으로 가중치 규제(Regularizaiton)가 있음

- 가중치 파라미터가 학습되는 것을 방해하여 오버피팅을 방지하는 정규화 기법
- 가중치 감쇠는 손실함수 수정을 통해 적용됨

4. 데이터 증강

❖ 이미지 증강

- 회전, 이동, 뒤집기 등을 사용하여 이미지를 수를 증가시킴

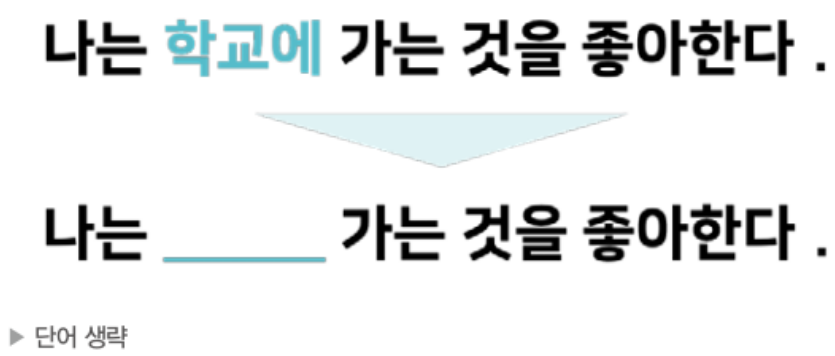


▶이미지의 회전, 이동, 뒤집기

4. 데이터 증강

❖ 텍스트 증강

- 단어 생략, 단어 교환, 단어 이동 등



4. 데이터 증강

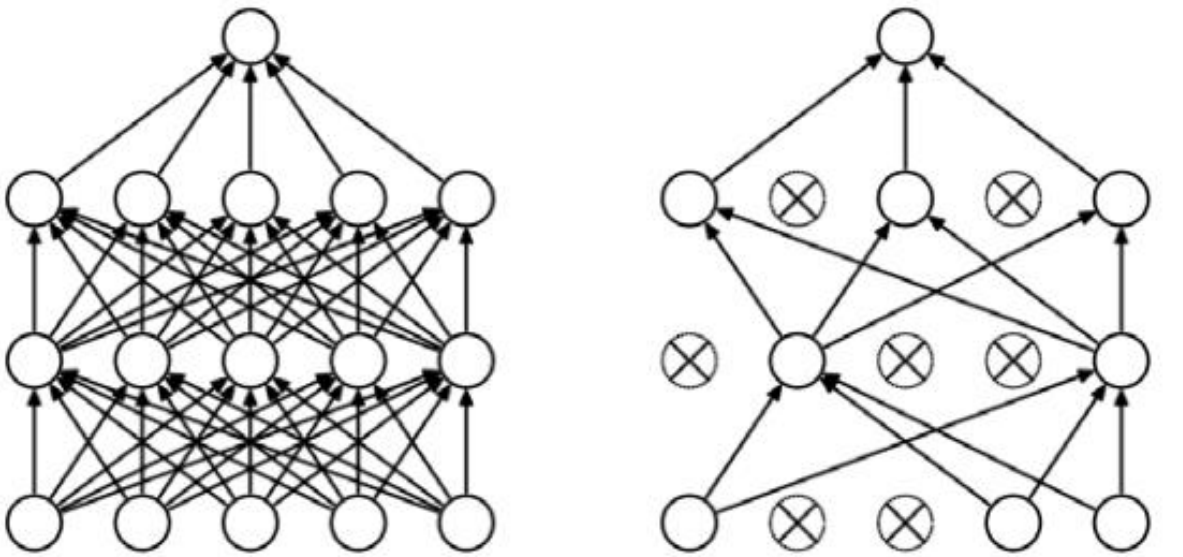
❖ 데이터 증감의 장점과 한계점

- 쉽게 데이터를 확장시킬 수 있음
- 신경망의 구조나 학습 기법 수정 없이 정규화 적용 가능
- 실제로 데이터를 더 수집하는 것에 비해 성능 개선이 낮음
- 데이터를 증강한다는 것은 새로운 지식을 배우는 것이 아니라 최적화를 수월하게 수행할 수 있도록 도와주는 측면이 강함

4. 드롭아웃

❖ 드롭아웃

- 신경망 학습에서 임의의 노드를 일정 확률로 드롭해서 학습에 참여하지 않도록 하는 방법
- 드롭되는 노드는 매 미니배치마다 이항분포를 활용하여 랜덤으로 선정
- 노드의 드롭확률 p : 하이퍼파라미터



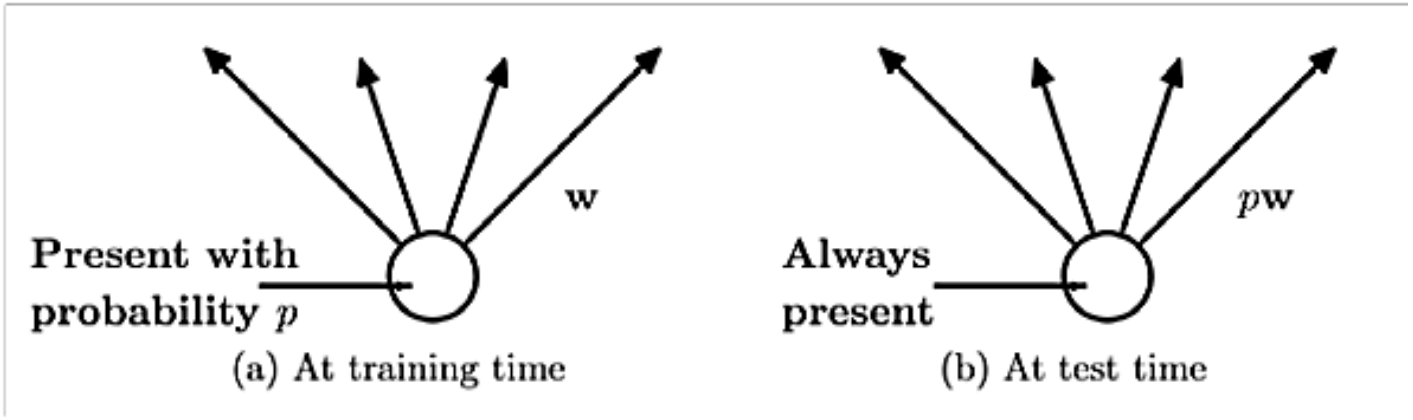
(a) Standard Neural Net (b) After applying dropout.

▶ 드롭아웃 동작 방식

5. 드롭아웃

❖ 학습과 추론 방식의 차이

- 드롭아웃은 학습에서 만 적용, 추론에서는 드롭되는 노드 없이 모든 노드가 항상 추론에 참여
- 가중치 파라미터 W에 (1-p)를 곱해 주어야 함
- 예 : 입력노드 3개인 선형 계층일 때

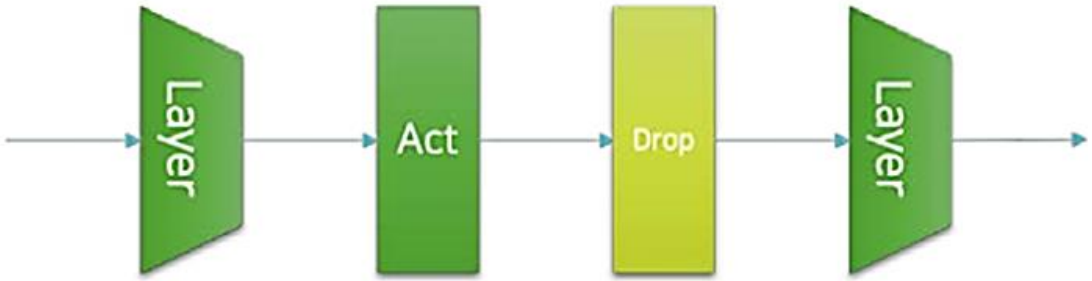


▶ 드롭아웃의 학습 할 때와 추론할 때의 차이

4. 드롭아웃

❖ 드롭 아웃 구현

- 위 그림의 오른쪽과 같이 드롭아웃이 구현되는 위치는 다음과 같이 활성 활수와 다음 계층 사이이다.



▶ 신경망계층에 드롭아웃 삽입

❖ 드롭아웃의 한계

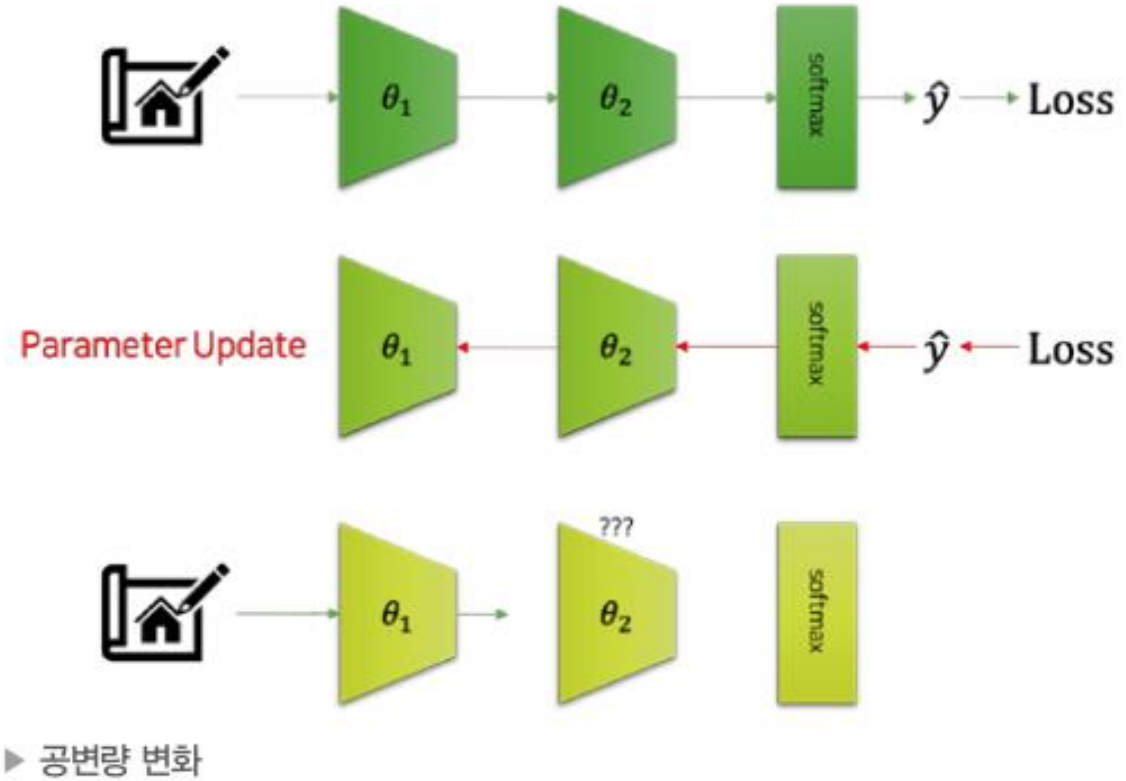
- 드롭아웃이 적용되면 일반화 성능이 개선될 수 있어도 손실 값의 수렴 속도가 저하될 수 있고 학습 오차가 증가할 수 있음

5. 배치 정규화

❖ 배치정규화 ?

- 기존의 정규화 기법 대부분 학습을 방해하는 형태로 적용, 일반화 성능을 개선하기 위해 학습 및 수렴 속도가 느려 짐
- 배치정규화는 학습 속도를 비약적으로 향상시킬 수 있을 뿐만 아니라 일반화 성능까지 대폭 개선할 수 있음

❖ 공변량 변화 문제 해결을 위해 제안



▶ 공변량 변화

5. 배치 정규화

❖ 배치정규화의 동작

- 미니배치의 분포를 정규화하여 문제를 해결
- 미니배치를 단위 가우시안 분포로 바꾸는 정규표준분포화를 한 이후에 스케일 파라미터 γ와 이동 파라미터 β를 적용, γ와 β 학습되는 가중치 파라미터로 신경망 내의 비선형적 성질을 유도

배치정규화식

$$\text{batch\_norm}(x) = \gamma \frac{(x-\mu)}{\sqrt{\sigma^2+\epsilon}} + \beta$$

$$\mu = x.\text{mean}(\text{dim} = 0)$$
$$\sigma = x.\text{std}(\text{dim} = 0)$$

$$\text{where } x \in \mathbb{R}^{N \times n}.$$

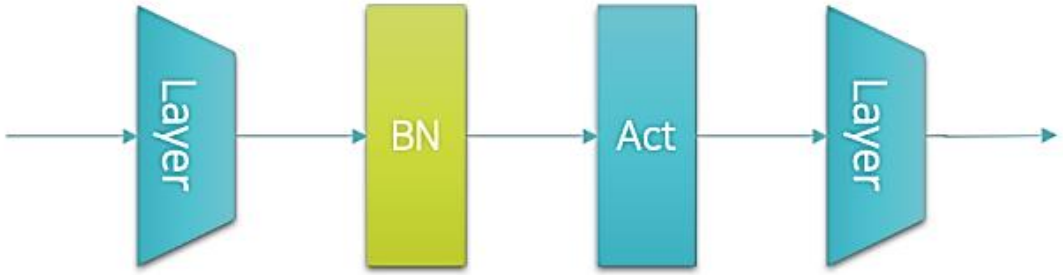
❖ 학습과 추론 동작 차이

- 학습과 추론 동작의 차이는 미니배치의 평균과 표준편차를 구하는 방식이 다름
- 학습 : 미니배치마다 평균과 표준편차를 구함
- 추론 : 추론과정에 들어오는 샘플들의 이동평균과 이에 따른 표준편차를 계산하여 활용

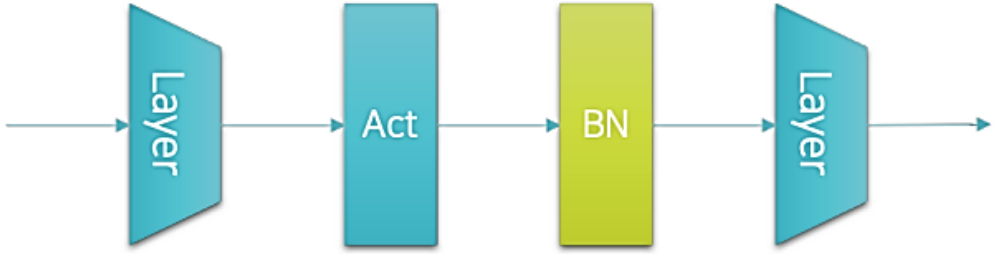
5. 배치 정규화

❖ 배치정규화 구현

- 선형계층과 활성화 함수 사이, 드롭아웃을 삽입하던 위치



▶ 선형 계층과 활성 함수 사이에 배치정규화 삽입



▶ 드롭아웃을 삽입하던 위치에 배치정규화 삽입

```
net = nn.Sequential(
    nn.Linear(300, 200),
    nn.LeakyReLU(),
    nn.BatchNorm1d(200),
    nn.Linear(200, 100),
    nn.LeakyReLU(),
    nn.BatchNorm1d(100),
    nn.Linear(100, 50),
    nn.LeakyReLU(),
    nn.BatchNorm1d(50),
    nn.Linear(50, 10)
)
```