AI 프로그래밍

ANN training과 Gradient descent

01

ANN의 Training

ANN의 Training

- ANN의 training
  - ANN은 해결하고자 하는 문제에 따라 training 방법이 나누어짐
    - Supervised learning (지도 학습)
      - 데이터에 input과는 다른 별도의 target이 필요한 문제에 활용
      - 예시: 사진 속 물체가 어떤 것인지 맞추는 문제(classification)
    - Unsupervised learning (비지도 학습)
      - 데이터에 별도의 정답 없이 input 자체가 target으로 활용 가능한 문제에 활용
      - 예시: 비슷한 사진끼리 분류하는 문제(clustering)
    - Reinforcement learning (강화 학습)
      - 행동에 대한 결과가 시간이 흐른 뒤에야 평가할 수 있는 문제에 활용
      - 예시: 보드 게임, 로봇의 움직임 제어 등

ANN의 Training

- ANN의 training
  - Supervised learning

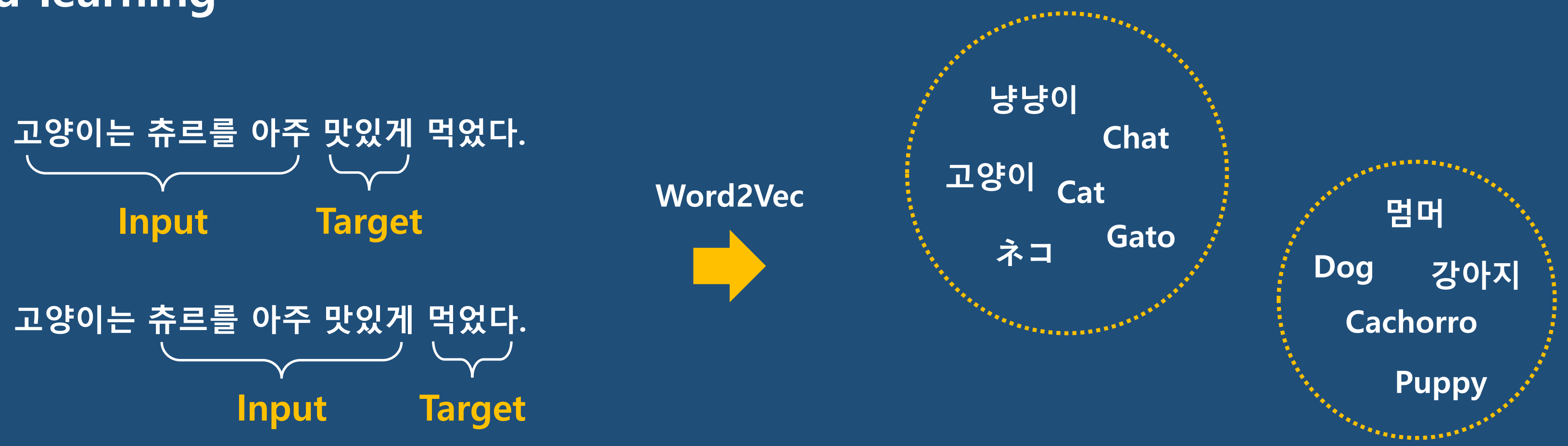
ImageNet

- 강아지
- 고양이
- 참새
- 앵무새

- 데이터의 input으로 target을 만들 수 없어,
  Dataset을 만들 때 ground truth를 통해 target 값을 포함해야 함
- Training이 가장 효과적이지만 dataset을 만들기 어려움

ANN의 Training

- ANN의 training
  - Unsupervised learning



- 데이터의 input으로 target을 만들 수 있음
- Dataset을 만들기 쉽지만, training 방법 설계가 비교적 까다로움

ANN의 Training

- ANN의 training
  - Reinforcement learning

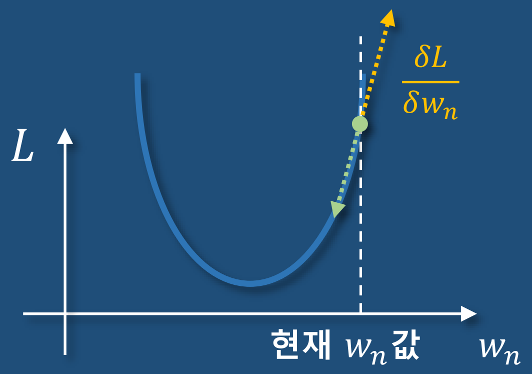
- 연속된 행동을 실행한 후 그 결과를 통해 이전 행동들에 점수를 매기는 방식
- 당장의 target을 주기 어려운 경우 유용하지만,
  안전한 환경이나 별도의 시뮬레이터가 필요함

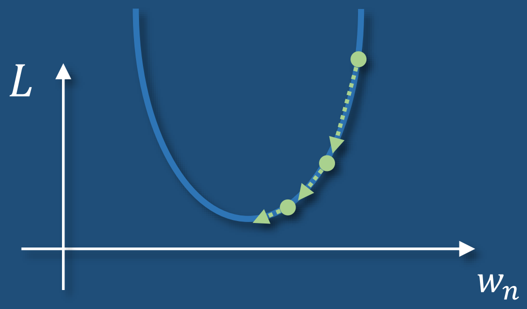
02

Gradient Descent

Gradient Descent

- Gradient descent

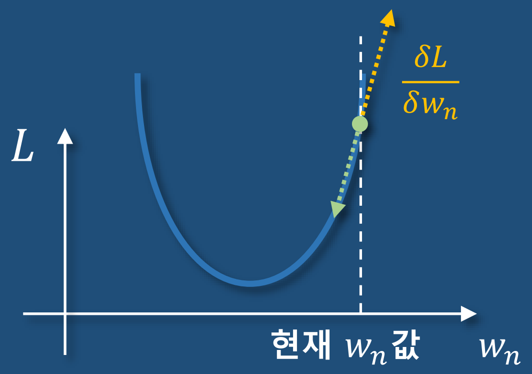


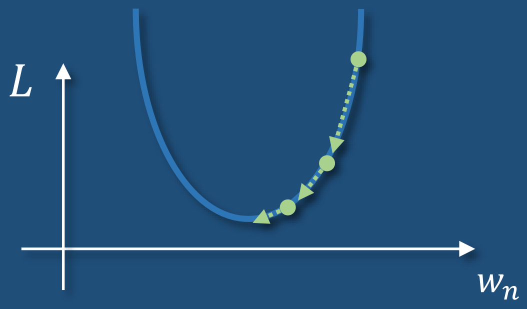


- 미분을 통해 각 weight 및 bias에 대한 loss 함수의 gradient(기울기)를 계산
- Loss를 줄이기 위해 각 weight와 bias를 gradient 반대 방향으로 조금씩 반복적으로 수정하여 loss 함수의 local minima(극소점)를 찾음
- 이러한 방식의 ANN training을 gradient descent라 함

Gradient Descent

- Gradient descent

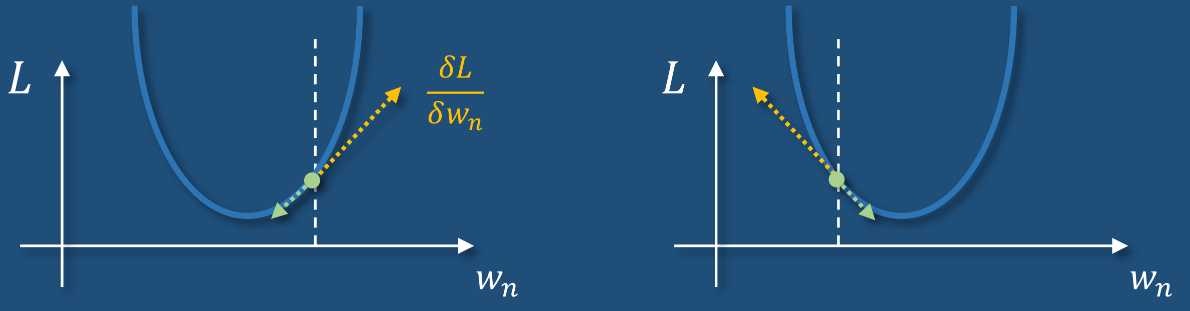




- Loss를 줄이기 위해 weight와 bias를 loss 함수의 gradient(기울기) 반대 방향으로 조금씩($\gamma < 1$) 수정
  - $w_n \leftarrow w_n - \gamma \frac{\delta L}{\delta w_n}$
  - $b_n \leftarrow b_n - \gamma \frac{\delta L}{\delta b_n}$

Gradient Descent

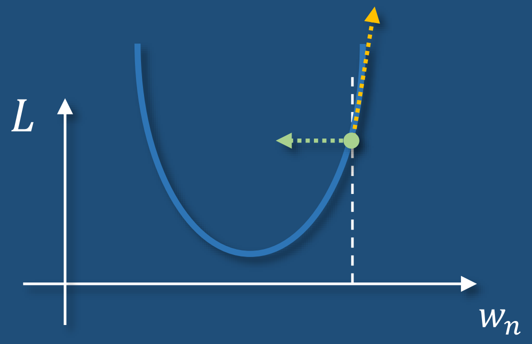
- Gradient descent



- Gradient에 따른 weight와 bias의 변화
  - $\frac{\delta L}{\delta w_n} > 0$인 경우, weight 및 bias가 줄어들면서 loss도 감소
  - $\frac{\delta L}{\delta w_n} < 0$인 경우, weight 및 bias가 증가하면서 loss도 감소
  - Local minima가 아니라면 어떤 경우에도 loss를 감소시킴

Gradient Descent

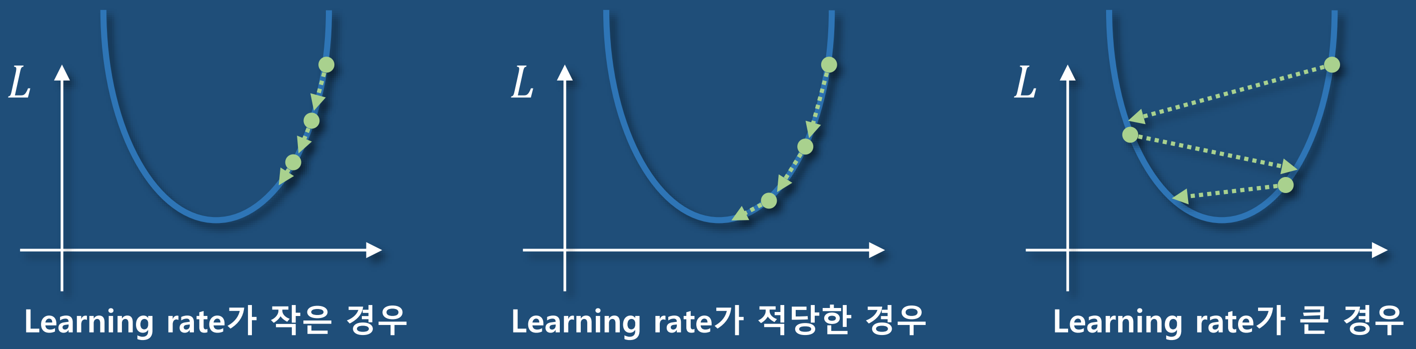
- Gradient descent



- Gradient에 따른 weight와 bias의 변화
  - $\left|\frac{\delta L}{\delta w_n}\right|$의 크기가 클수록 weight 및 bias를 과감하게 수정

Gradient Descent

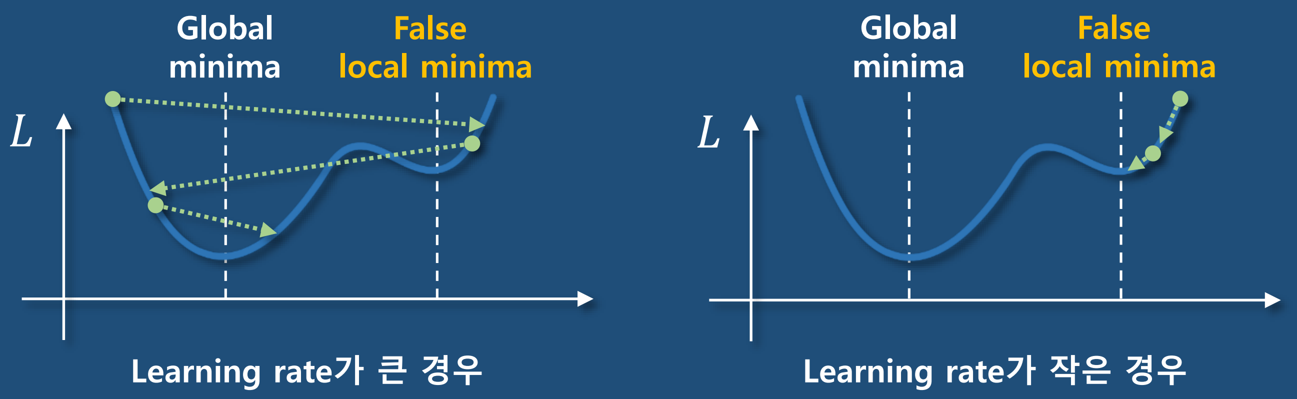
- Gradient descent



- Learning rate(γ)에 따른 학습 과정
  - Learning rate가 너무 작은 경우 loss의 local minima에 도달하는데 너무 오래 걸림
  - Learning rate가 너무 큰 경우 진동(oscillation)이 일어나 local minima에서 멈추기 어려움

Gradient Descent

- Gradient descent



- Learning rate(γ)에 따른 학습 과정
  - 실제 문제의 경우 loss 함수가 복잡하여 많은 false local minima가 존재할 수 있음
  - Learning rate가 큰 경우 진동은 있지만, false local minima에서 빠져나올(shoot) 수 있음
  - Learning rate가 너무 작으면 false local minima에서 빠져나올 수 없음

Gradient Descent

- Gradient descent
  - ANN의 수많은 weight와 bias들을 통틀어 parameter(Θ)라 부름
  - 모든 parameter에 대한 gradient를 $\nabla\boldsymbol{\Theta}$로 표기함

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \gamma\nabla\boldsymbol{\Theta}$$

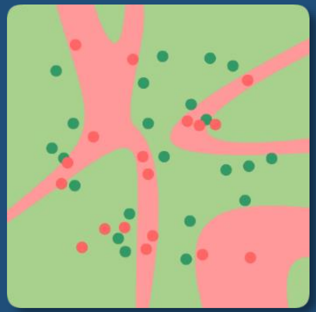
  - $\boldsymbol{\Theta}$: ANN의 모든 weight와 bias를 원소로 갖는 tensor
  - $\nabla\boldsymbol{\Theta}$: Loss를 각 weight와 bias로 편미분한 값을 원소로 갖는 tensor, $\nabla\boldsymbol{\Theta} = \frac{\delta L}{\delta \boldsymbol{\Theta}}$

03

Stochastic Gradient Descent

Stochastic Gradient Descent

- Batch gradient descent
  - 실제 ANN은 굉장히 큰 dataset으로 training을 진행
  - Dataset에서 최대한 많은 데이터에 적합한 parameter를 찾으려 함



  - Parameter는 dataset의 개별 데이터에 따라 늘려야 할 수도, 줄여야 할 수도 있음
    - 개별 데이터를 하나씩 따로 학습하면 parameter가 진동하여 수렴하기 어려움
  - 이를 해결하기 위해 dataset의 모든 데이터를 동시에 training하는 것을 batch gradient descent라 함

Stochastic Gradient Descent

- Batch gradient descent
  - Python pseudo code

```
> iteration = 0
> Θ를 random 값으로 초기화

> while(iteration < n):
>     iteration += 1
>     ∇Θ = 0

>     for(i_번째_데이터 in dateset):     # 최대 연산 양은 n × dataset_크기
>         ∇Θ_i = δL_i / δΘ               # 개별 데이터의 loss를 통해 gradient를 계산, Θ의 현재 값은 바뀌지 않음
>         ∇Θ += ∇Θ_i                     # Gradient의 부호 반대일 때 발생하는 진동을 상쇄시킴
>     Θ = Θ − γ∇Θ                        # 진동이 상쇄된 gradient로 parameter 수정

>     if(‖∇Θ‖ < ϵ):                      # Gradient가 아주 작은 값 ϵ보다 작아지면 parameter가 수렴했다고 간주
>         break
```

Stochastic Gradient Descent

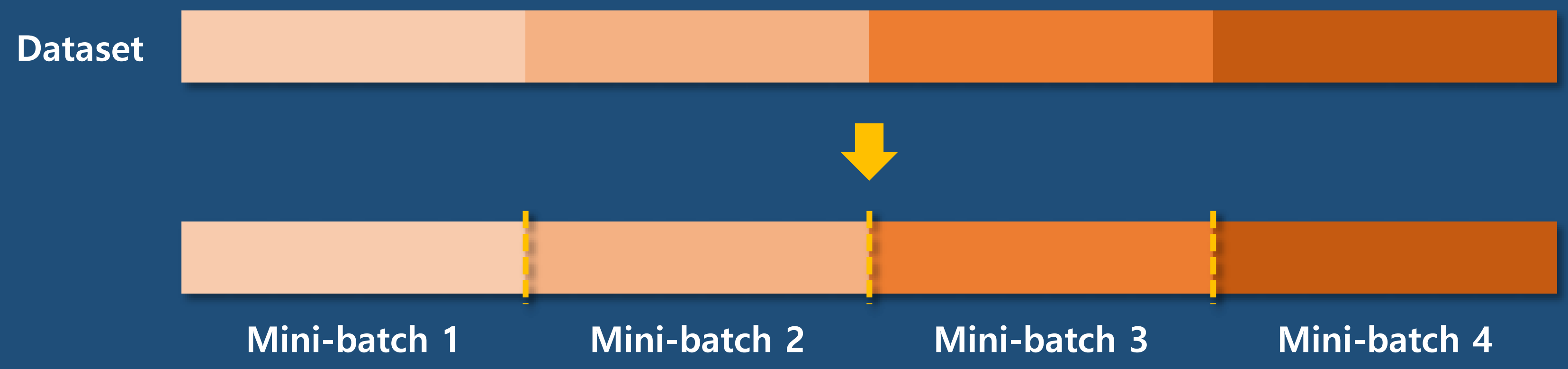
- Batch gradient descent
  - Dataset의 각 데이터로 계산한 모든 gradient를 합산하여 training을 진행
    - 일부 데이터에 의해 parameter가 진동할 여지가 없음
    - 일부 데이터에 의해 false local minima에 빠질 확률이 현저히 낮음

  - Training의 성공 확률을 높여줌
  - 단, dataset이 클수록 연산 양이 많아져 training 시간이 너무 길어짐

  - 빅데이터가 기본인 현대 AI에서는 training의 성공 확률과 시간 사이의 trade-off가 필요

Stochastic Gradient Descent

- Mini-batch gradient descent
  - 연산 양을 줄이기 위해, dataset를 일정한 크기의 mini-batch로 나누어 training을 진행
    - Mini-batch가 작을 수록 최대 연산 양이 줄어듬
    - Mini-batch가 클 수록 training 성공 확률이 높아짐



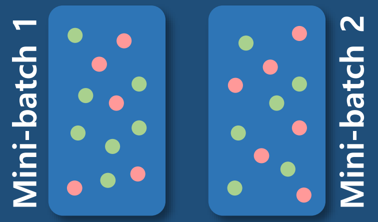
Stochastic Gradient Descent

- Mini-batch gradient descent
  - Python pseudo code

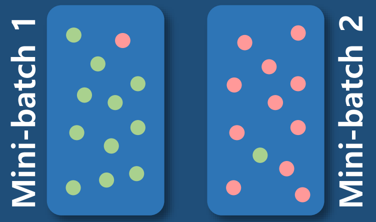
> iteration = 0
> Θ를 random 값으로 초기화
>
> while(iteration < n):
>   iteration += 1
> $\nabla\Theta = 0$
>
>   j = iteration % mini_batch_개수
>   for(i_번째_데이터 in j_번째_mini_batch): # 최대 연산 양은 n × dataset_크기에서 n × mini_batch_크기로 감소
> $\nabla\Theta_i = \frac{\delta L_i}{\delta\Theta}$
> $\nabla\Theta \mathrel{+}= \nabla\Theta_i$
> $\Theta = \Theta - \gamma\nabla\Theta$ # 각 mini-batch 안에서 진동이 상쇄된 gradient로 전체 parameter 수정
>
>   if($\|\nabla\Theta\| < \epsilon$):
>   break

Stochastic Gradient Descent

- Stochastic gradient descent
  - Mini-batch 내 데이터간 관계성(correlation)이 높은 경우 training하는 mini-batch가 바뀔 때마다 parameter가 크게 변하면서 진동이 발생, 수렴하기가 어려워짐



낮은 데이터간 관계성

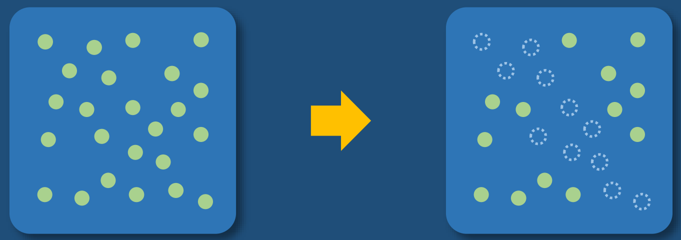


높은 데이터간 관계성

  - 데이터간 관계성이 높은 경우
    - Mini-batch 1을 training할 때 초록색 데이터에 맞도록 parameter를 대폭 수정
    - Mini-batch 2를 training할 때 빨간색 데이터에 맞도록 parameter를 대폭 수정
    - 두 mini-batch간의 합의점에 수렴하기 어려움

Stochastic Gradient Descent

- Stochastic gradient descent
  - 일반적으로, dataset에서 근접한 데이터간의 관계성이 클 확률이 높음
  - 멀리 떨어진 데이터들로 mini-batch를 구성하기 위해 확률적으로 sampling함
  - 확률적 sampling을 수행하므로 stochastic(확률적) gradient descent라 함
    - 엄밀히 구분하면 sampling을 통해 mini-batch를 만들어 training하는 방법을 MSGD (mini-batch stochastic gradient descent)라 부르지만, 현대에는 mini-batch를 기본적으로 활용하므로, 간편하게 stochastic gradient descent로 부름
  - 하지만, 확률적 sampling은 데이터를 중복 또는 누락해 데이터 간 관계성이 커지기 쉬움

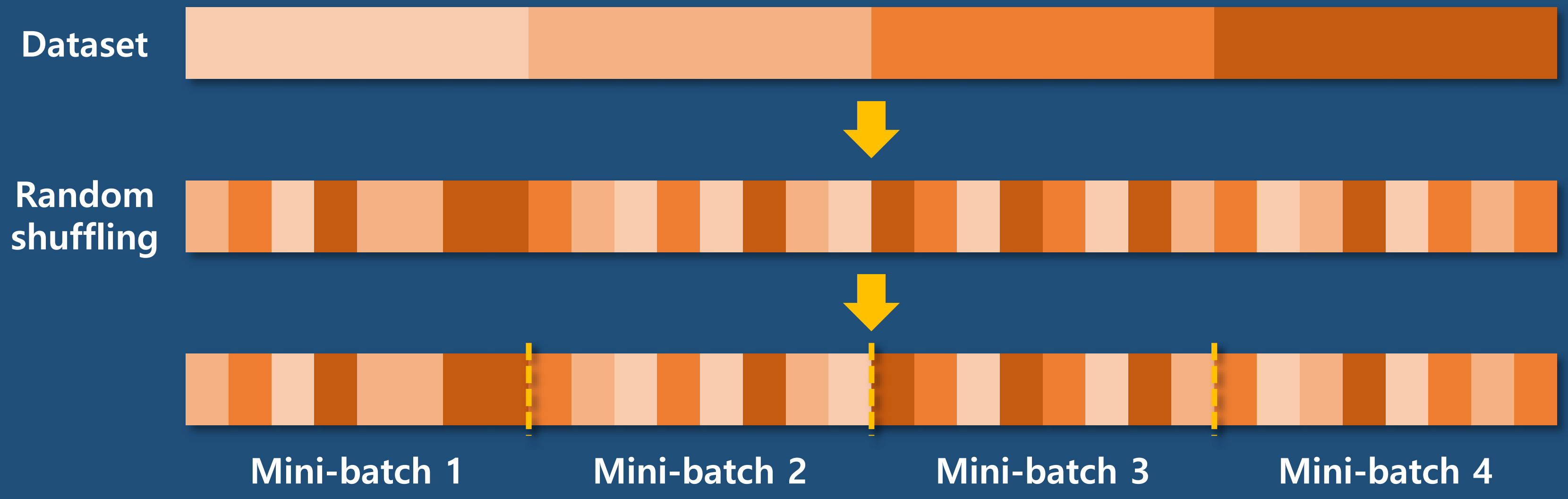


데이터 누락으로 인해 높아진 데이터간 관계성

  - 따라서 sampling을 하더라도 dataset의 모든 데이터를 동일한 횟수로 학습해야 함

Stochastic Gradient Descent

- Stochastic gradient descent
  - Training 성공 확률을 높이기 위한 mini-batch 생성
    1. Dataset의 데이터의 순서를 random으로 shuffle
    2. Shuffle된 dataset을 mini-batch 크기로 나누어 training



Stochastic Gradient Descent

- Stochastic gradient descent
  - Python pseudo code

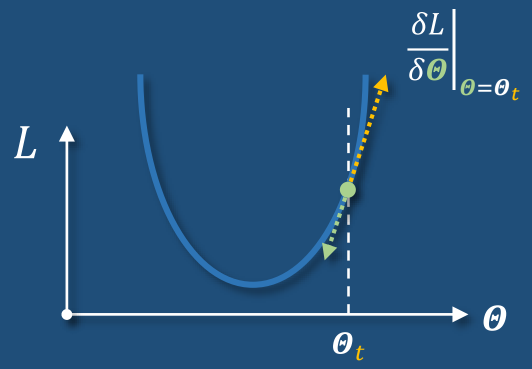
> iteration = 0
> $\boldsymbol{\Theta}$를 random 값으로 초기화
> dataset_random_shuffle()   # Dataset의 데이터 순서를 random으로 shuffle
>
> while(iteration < n):   # Mini-batch gradient descent와 동일
>   iteration += 1
> $\nabla\boldsymbol{\Theta} = 0$
>   j = iteration % mini_batch_개수
>   for(i_번째_데이터 in j_번째_mini_batch):
> $\nabla\boldsymbol{\Theta}_i = \frac{\delta L_i}{\delta\boldsymbol{\Theta}}$
> $\nabla\boldsymbol{\Theta} \mathrel{+}= \nabla\boldsymbol{\Theta}_i$
> $\boldsymbol{\Theta} = \boldsymbol{\Theta} - \gamma\nabla\boldsymbol{\Theta}$
>   if($\|\nabla\boldsymbol{\Theta}\| < \epsilon$):
>   break

04

Optimizer

Optimizer

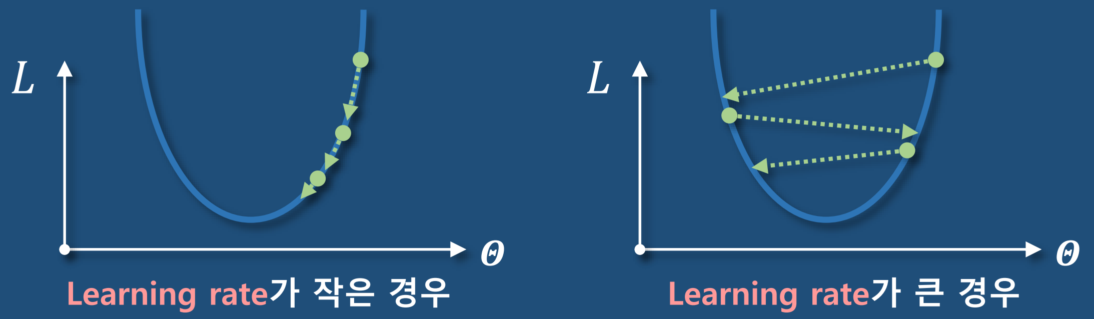
- Optimizer
  - SGD (Steepest Gradient Descent)
    - Parameter 수정 방법
      - $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \gamma \nabla \boldsymbol{\Theta}_t$
      - $\nabla \boldsymbol{\Theta}_t = \left.\frac{\delta L}{\delta \boldsymbol{\Theta}}\right|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}_t}$



    - Notation
      - $\boldsymbol{\Theta}_t$: 현재 parameter
      - $\boldsymbol{\Theta}_{t+1}$: 수정 후 parameter
      - γ : Learning rate
      - $\left.\frac{\delta L}{\delta \boldsymbol{\Theta}}\right|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}_t}$: 현재 parameter에서의 gradient

Optimizer

- Optimizer
  - SGD (Steepest Gradient Descent)



- Parameter를 경사면(loss의 변화량)이 가장 가파른 방향으로 이동시키는 방법
- Gradient는 항상 가장 가파른 방향을 가리키므로, 이를 그대로 활용
- Learning rate의 영향을 많이 받음

Optimizer

- Optimizer
  - Parameter 수정양을 조정하여 training의 시간을 줄이는 동시에 성공 확률을 높여주는 함수

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - f(\gamma \nabla \boldsymbol{\Theta})$$

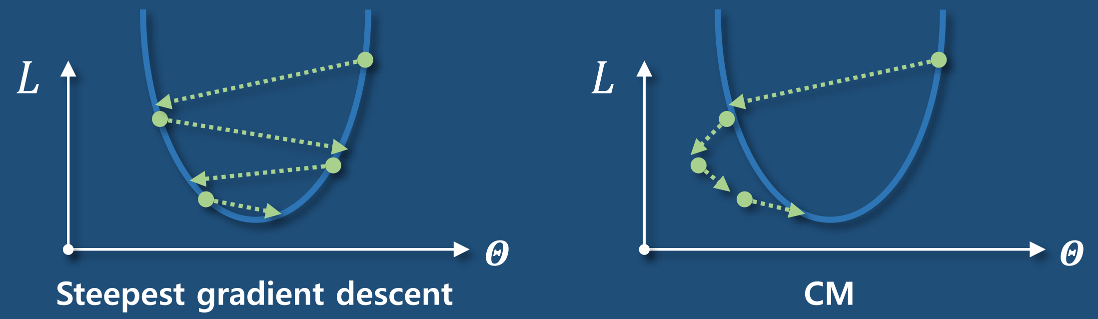
  - Optimizer의 역사
    - CM (Classical Momentum, 1986)
    - NAG (Nesterov Accelerated Gradient, 1991)
    - AdaGrad (Adaptive Gradient Algorithm, 2011)
    - RMSprop. (Root Mean Square Back-propagation, 2013)
    - ADAM (Adaptive Moment Estimation, 2014)
      - 외에도 AdaDelta, Nadam 등 다양한 optimizer들이 있음

Optimizer

- Optimizer
  - CM (Classical Momentum) optimizer
    - Loss 함수의 local minima를 찾아가는 과정을 굴러가는 구슬에 빗대어 만든 optimizer
    - 운동량(momentum) 개념을 추가, parameter가 관성을 가진 것처럼 이동함
    - Parameter 수정 방법
      - $\boldsymbol{M}_{t+1} = \alpha \boldsymbol{M}_t - \gamma \nabla \boldsymbol{\Theta}_t$
      - $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \boldsymbol{M}_{t+1}$
    - Notation
      - $\boldsymbol{M}_t$: 현재 운동량, $\boldsymbol{M}_0 = 0$
      - α : 운동량에 적용하는 discount factor, $\alpha < 1$

Optimizer

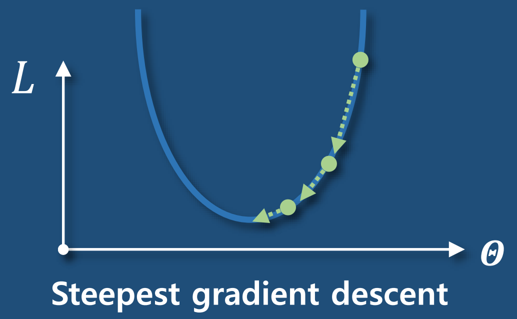
- Optimizer
  - CM (Classical Momentum) optimizer

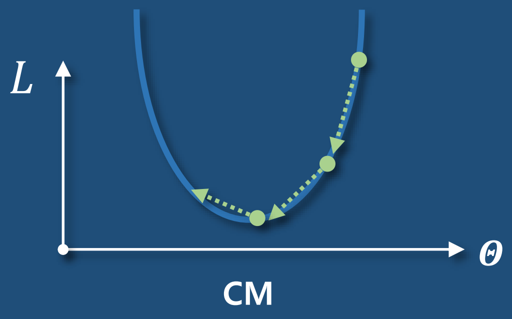


  - 현재와 이전 gradient의 방향이 서로 반대일 경우 parameter 변화량이 작아짐
  - Learning rate가 커도 parameter의 진동이 줄어듬

Optimizer

- Optimizer
  - CM (Classical Momentum) optimizer





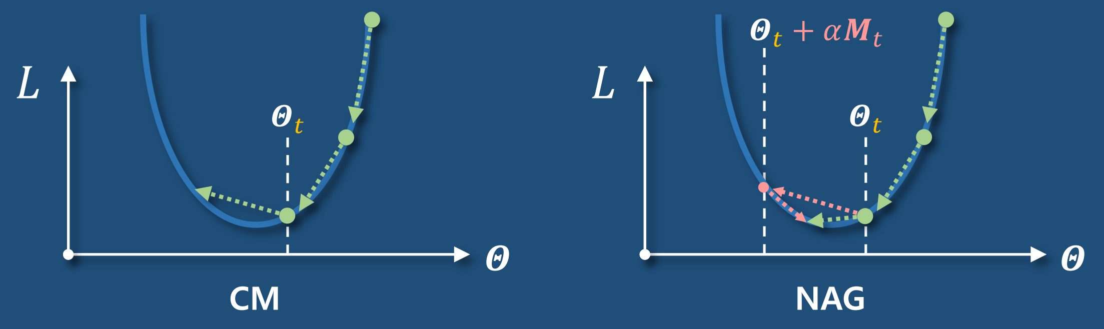
  - 현재와 이전 gradient의 방향이 서로 비슷한 경우 parameter 변화량이 더 커짐
  - Learning rate가 작아도 얕은 false local minima에서 빠져나올 확률이 커짐
  - 하지만, overshoot으로 인해 global minima에서도 빠져나와 parameter 수렴이 늦어질 수 있음

Optimizer

- Optimizer
  - NAG (Nesterov Accelerated Gradient) optimizer
    - 운동량을 구할 때 현재가 아닌 미래 시점의 gradient를 활용
    - Overshoot으로 인한 global minima에서 빠져나가는 현상을 방지
    - Parameter 수정 방법
      - $\boldsymbol{M}_{t+1} = \alpha \boldsymbol{M}_t - \gamma \frac{\delta L}{\delta \boldsymbol{\Theta}}\Big|_{\boldsymbol{\Theta}=\boldsymbol{\Theta}_t+\alpha \boldsymbol{M}_t}$
      - $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \boldsymbol{M}_{t+1}$

Optimizer

- Optimizer
  - NAG (Nesterov Accelerated Gradient) optimizer



- 미래 시점의 gradient가 현재 진행 방향과 반대일 경우, parameter 변화량이 줄어듬
- 깊은 global minima에서 빠져나갈 확률을 크게 줄여, parameter 수렴이 빠르게 일어남

Optimizer

- Optimizer
  - AdaGrad (Adaptive Gradient Algorithm) optimizer
    - 상수였던 learning rate를 각 parameter에 따라 다르게 적용
    - 총 이동 거리가 긴 parameter일수록 learning rate를 줄여나감
    - Parameter 수정 방법
      - $G_{t+1} = G_t + (\nabla\Theta_t^{\ T}\nabla\Theta_t)$
      - $\Theta_{t+1} = \Theta_t - \frac{\gamma}{\epsilon+\sqrt{G_{t+1}}} \odot \nabla\Theta_t$
    - Notation
      - G_t : 현재까지의 각 parameter 총 이동 거리를 원소로 갖는 tensor
      - $\odot$: Element-wise dot product, 같은 위치의 원소끼리만 곱함
      - ϵ : 분모가 0이 되지 않도록 하기 위해 추가한 아주 작은 값의 상수

Optimizer

- Optimizer
  - AdaGrad (Adaptive Gradient Algorithm) optimizer
    - 장점
      - 각 parameter에 다른 learning rate가 적용되어 더 효율적인 training 가능
    - 단점
      - Parameter의 수렴 여부에 상관없이 이동 거리가 길면 learning rate가 무조건 줄어듬
        - Global minima에 도달하기 전에 training이 끝날 수 있음

Optimizer

- Optimizer
  - RMSprop. (Root Mean Square Back-propagation) optimizer
    - AdaGrad에서 parameter의 수렴 여부와 상관없이 training이 끝나는 문제를 해결
    - Discount factor를 이용해, 분모가 되는 이동 거리가 무작정 커지지 않도록 함
    - Parameter 수정 방법
      - $\boldsymbol{G}_{t+1} = \alpha \boldsymbol{G}_t + (1-\alpha)(\nabla\boldsymbol{\Theta}_t \odot \nabla\boldsymbol{\Theta}_t)$
      - $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \frac{\gamma}{\epsilon + \sqrt{G_{t+1}}} \odot \nabla\boldsymbol{\Theta}_t$
    - Notation
      - α : 이동 거리에 적용하는 discount factor, $\alpha < 1$

Optimizer

- Optimizer
  - ADAM (Adaptive Moment Estimation) optimizer
    - 각 parameter의 운동량과 이동 거리를 모두 활용하는 optimizer
    - 현대의 많은 ANN 시스템이 ADAM optimizer를 통해 training 진행
    - Parameter 수정 방법
      - $\boldsymbol{M}_{t+1} = \beta_1 \boldsymbol{M}_t + (1-\beta_1)\nabla\boldsymbol{\Theta}_t$
      - $\boldsymbol{G}_{t+1} = \beta_2 \boldsymbol{G}_t + (1-\beta_2)(\nabla\boldsymbol{\Theta}_t \odot \nabla\boldsymbol{\Theta}_t)$
      - $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \frac{\gamma}{\epsilon + \sqrt{\boldsymbol{G}_{t+1}}} \odot \boldsymbol{M}_{t+1}$
    - Notation
      - β_1 : 운동량에 적용하는 discount factor, $\beta_1 < 1$
      - β_2 : 이동 거리에 적용하는 discount factor, $\beta_2 < 1$

감사합니다

Fin