[딥러닝 2단계] 4. 최적화 알고리즘

1. 미니 배치 경사하강법

Batch vs mini-batch gradient descent

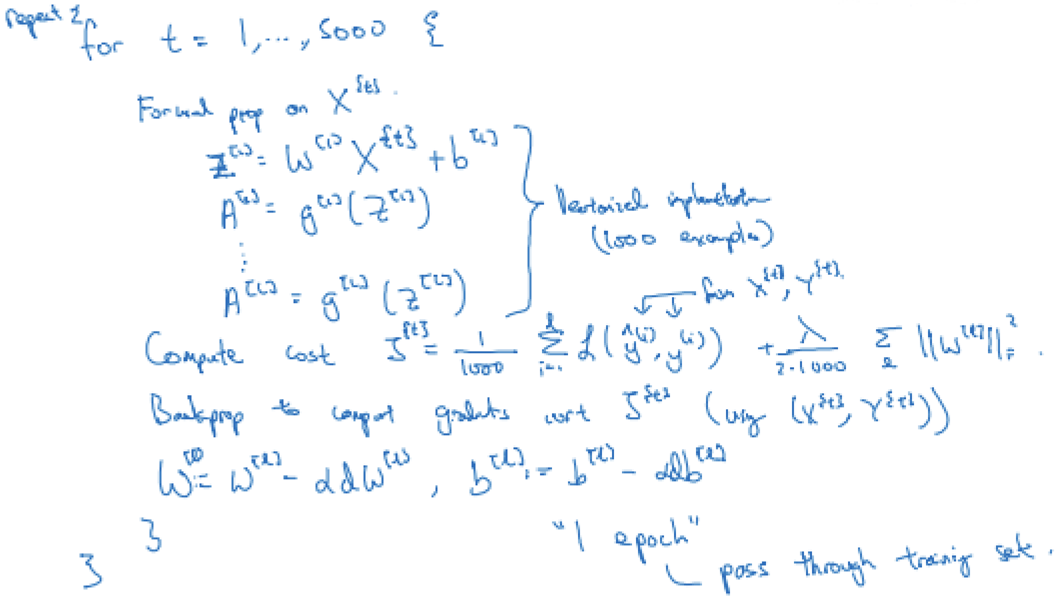
- Batch gradient descent: 전체 훈련 샘플에 대해 훈련 후 경사 하강 진행
- Mini-batch gradient descet: 전체 훈련 샘플을 작은 훈련 세트인 미니배치로 나누어 훈련 후 경사 하강 진행
- 벡터화는 m개의 샘플에 대한 계산을 효율적으로 만들어줌
- X의 차원:(n_x,m)
- Y의 차원: (1,m)
- 배치 경사 하강법 -> 데이터 세트가 크다면 훈련하는데 많은 시간이 들고, 경사 하강을 진행하기까지 오랜 시간이 걸림
    - m=5000000
- => 작은 훈련 세트인 미니배치로 나누어 훈련 후 경사 하강 진행
    - 사이즈가 1000인 5000개의 미니배치로 나눔

$$\underset{(n_x,m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \cdots x^{(1000)}}_{X^{\{1\}}\ (n_x,1000)} \mid \underbrace{x^{(1001)} \cdots x^{(2000)}}_{X^{\{2\}}\ (n_x,1000)} \mid \cdots \mid \underbrace{\cdots x^{(m)}}_{X^{\{5,000\}}\ (n_x,1000)}]$$

$$\underset{(1,m)}{Y} = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \cdots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)} \cdots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \cdots \mid \underbrace{\cdots y^{(m)}}_{Y^{\{5,000\}}\ (1,1000)}]$$

- 미니배치 t: X^{t}, Y^{t}
    - 각 미내배치의 차원: (n_x, 1000)

Mini-batch gradient descent

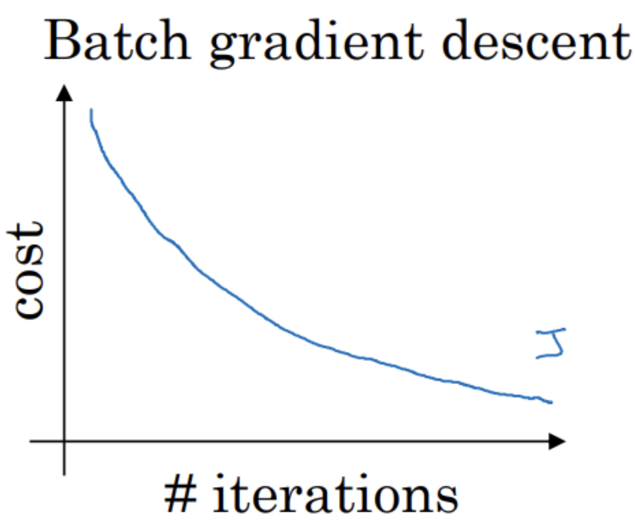


- for문: 크기가 1000인 미니배치가 5000개
  - 1단계의 경사 하강법 구현
  - 1 epoch: 훈련 세트를 거치는 한 반복
- repeat문: 훈련 세트를 여러번 반복

2. 미니 배치 경사하강법 이해하기

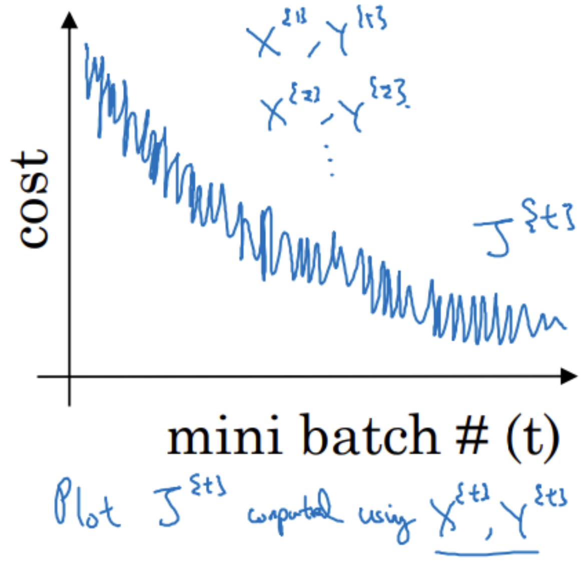
Training with mini batch gradient descent

배치 경사 하강법



- 모든 반복마다 비용함수가 감소하지 않으면 잘못된 것

미니배치 경사 하강법



- 모든 반복마다 비용함수 감소하지 않음
- X^{t}, Y^{t}로 계산한 비용함수 J^{t}를 그림
- 전체적인 흐름은 감소하나 약간의 노이즈 발생

Choosing your mini-batch size

- 훈련 세트의 크기:m

미니배치 크기 = m

- 배치 경사 하강법과 동일
  - (X^{1},Y^{1}) = (X,Y)
- 한 반복에서 너무 오랜 시간이 걸림

미니배치 크기 = 1

- 확률적 경사 하강법
- 각 훈련 샘플은 하나의 미니배치
- 대부분의 경우 전역 최솟값으로 가지만 잘못된 곳으로 가기도 함
  - 노이즈 클 수 있으나 평균적으로는 좋은 방향으로 감
  - 절대 수렴하지 않음

        - 최솟값 주변을 진동
- 장점: 하나의 샘플만 처리한 뒤에 진행할 수 있어 매우 간단, 노이즈도 작은 학습률로 줄일 수 있음
- 단점: 벡터화에서 얻을 수 있는 속도 향상을 잃게 됨, 비효율적

미니배치 크기 = 1과 m 사이의 값

- 실제 미니배치크기는 1부터 m
- 미니 배치크기 너무 크거나 작지 않을 때 가장 잘 작동
- 가장 빠른 학습을 제공
- 장점:
    1. 많은 벡터화를 얻음 -> 속도 향상
    2. 전체 훈련 세트가 진행되기를 기다리지 않고 진행 할 수 있음

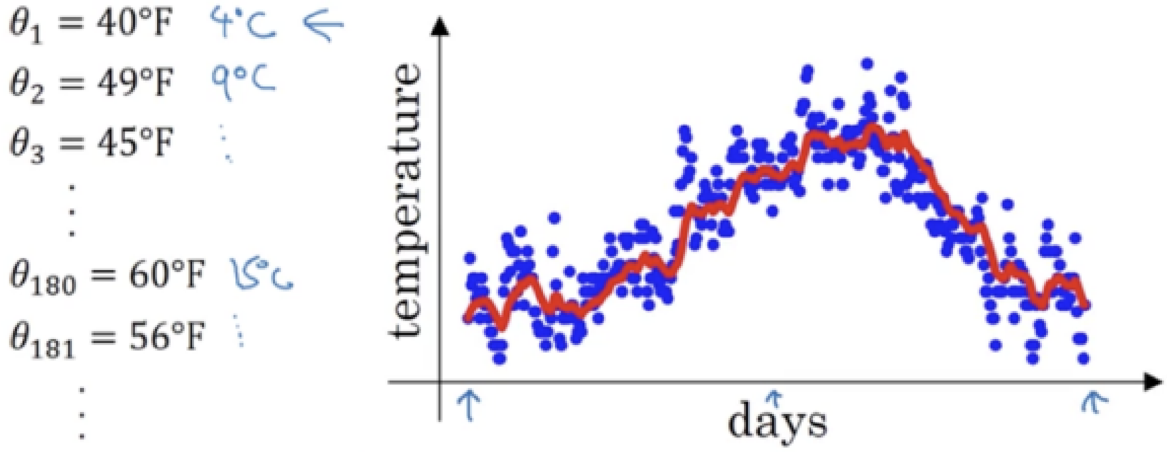
미니배치 사이즈 선택 방법

- 작은 훈련 세트 -> 배치 경사 하강법 사용
    - 샘플이 2000개보다 적은 경우
- 큰 훈련 세트 -> 미니배치 크기: 64~512가 가장 일반적
    - 컴퓨터 메모리의 접근 방식 때문에 미니배치 크기가 2의 제곱인 것이 빠름
    - 64, 128, 256, 512
- 미니배치에서 모든 X^{t}, Y^{t}가 CPU, GPU 메모리에 맞는지 확인

3. 지수 가중 이동 평균

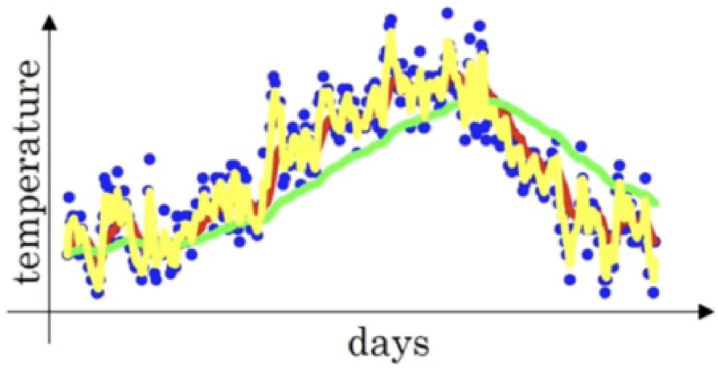
- 경사 하강법보다 더 빠른 알고리즘을 이해하기 위해 지수 가중 이동 평균을 알아야함

Temperature in London



v0=0
v1=0.9v0+0.1θ1
v2=0.9v1+0.1θ2
...
vt=0.9vt-1+0.1θt
-> 그래프에 나타내면 일별 기온의 지수가중평균

Exponentially weighted averages



$v_t = \beta v_{t-1} + (1-\beta)\theta_t$

- v_t는 $\frac{1}{1-\beta}$ * 일별 기온의 평균
- β=0.9 -> 10일 동안 기온의 평균
- β=0.98 -> 50일의 기온의 평균 -> 초록색 그래프
- β값이 클수록 더 많은 날짜의 기온의 평균을 이용하기 때문에 곡선이 더 부드러워짐
그러나, 더 큰 범위에서 기온을 평균하므로 곡선이 올바른 값에서 더 멀어짐

-> 기온이 바뀔 경우 지수가중평균 공식은 더 느리게 적응

- β=0.5 -> 2일 기온 평균 -> 노란색 그래프
  - 노이즈 많고 이상치에 더 민감
  - 기온 변화에 더 빠르게 적응

4. 지수 가중 이동 평균 이해하기

Exponentially weighted averages

$v_{100} = 0.1\theta_{100} + 0.9v_{99}$

$= 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9v_{98})$

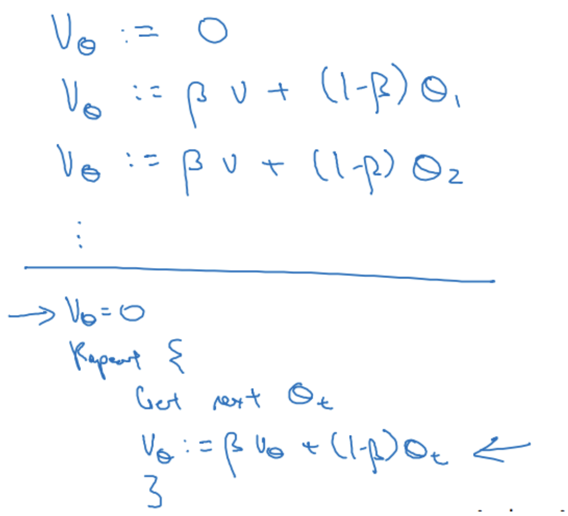
$= ...$

$v_{100} = 0.1\theta_{100} + 0.10.9\theta_{99} + 0.1(0.9)^2\theta_{98} + ...$

- 그림으로 표현하면 지수적으로 감소하는 그래프 (v_100을 기준으로 보았을 때)
  - v_100은 각각의 요소에 지수적으로 감소하는 요소를 곱해서 더한 것이기 때문
- 얼마의 기간동안 평균이 구하는가?
  - $0.9^{10} \approx 0.35 \approx 1/e$
  - β=1-ε
  - $(1-\epsilon)^{1/\epsilon} = 1/e$

  1. 만약 e=0.1 -> 약 10일 걸림
  2. 만약 β=0.98 -> 약 50일

Implementing exponentially weighted averages



- 장점: 아주 적은 메모리를 사용
    - v_θ 하나의 실수만을 컴퓨터 메모리에 저장하고 가장 최근에 얻은 값을 덮어쓰면 되기 때문

5. 지수 가중 이동 평균의 편향 보정

- 편향 보정으로 평균을 더 정확하게 계산할 수 있음

Bias correction

지수 가중 이동 평균

> v0=0
>
> v1=0.02θ1
>
> v2=0.0196θ1+0.02θ2

- 초반 좋지 않은 추정 (보라색선)

편향 보정

> 💡 $\frac{V_t}{1-\beta^t}$를 계산
>
> $t = 2 : 1 - \beta^t = 1 - (0.98)^2 = 0.0396$
>
> $\frac{V_2}{0.0396} = \frac{0.0196\theta_1 + 0.02\theta_2}{0.0396}$

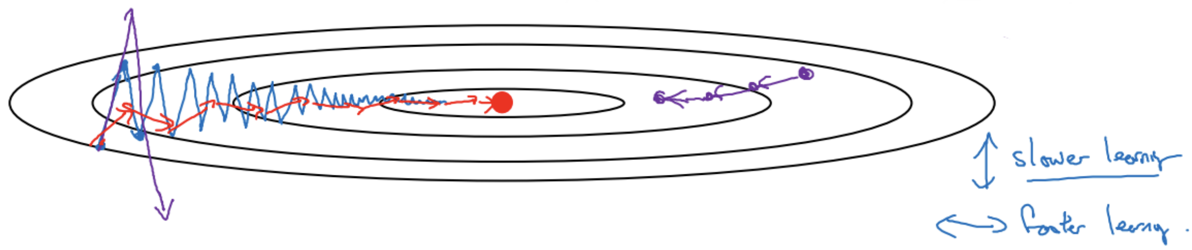
- t가 충분히 커지면 편향 보정은 효과가 거의 없어짐

- 초기 단계의 학습에서 더 나은 추정값을 얻을 수 있게 도와줌 (초록색선)
- 머신러닝에서 지수가중평균을 구현하는 경우 대부분 편향 보정을 거의 구현하지 않음

6. Momentum 최적화 알고리즘

- 모멘텀 알고리즘은 일반적인 경사 하강법보다 거의 항상 빠르게 동작
- 경사에 대한 지수가중평균을 계산

Gradient descent example



- 진동: 경사 하강법 속도를 느리게 함, 더 큰 학습률을 사용하는 것을 막음
- 수직축: 진동을 막기 위해 학습이 더 느리게 일어나기를 바람
- 수평축: 더 빠른 학습을 원함

Momentum

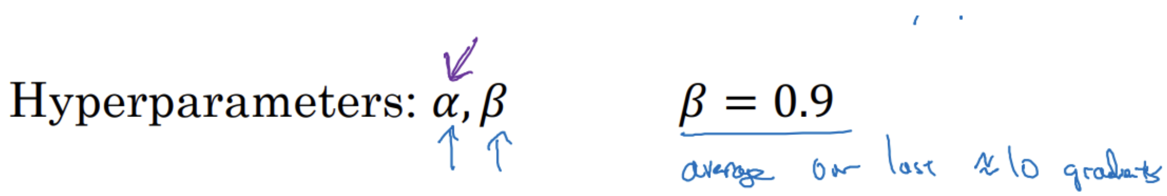
$V_{dw} = \beta V_{dw} + (1-\beta)dw$

$V_{db} = \beta V_{db} + (1-\beta)db$

$w := w - \alpha V_{dw}, b := b - \alpha V_{db}$

- 경사 하강법의 단계를 부드럽게 만들어줌
- 도함수 항들은 아래로 내려갈 때 가속을 제공
- 모멘텀 항들은 속도를 나타냄
- β는 마찰을 제공해서 속도가 제한 없이 빨라지는 것을 막음

Implementation details



- 두가지 하이퍼파라미터: 학습률 α, 지수가중평균을 제어하는 β
- β=0.9가 일반적: 지난 10일 간의 온도를 평균
- 편향 보정는 잘 사용 x
    - 이동평균이 충분히 진행 돼서 편향 추정이 더 이상 일어나지 않기 때문

$V_{dw}=0$, $V_{db}=0$

On iteration t:

Compute dW, db on the current mini-batch

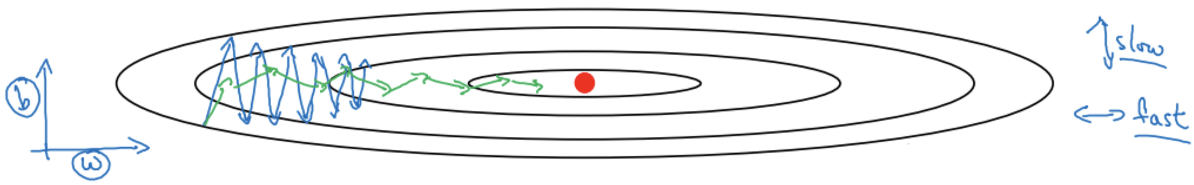
$$\rightarrow v_{dW} = \beta v_{dW} + \cancel{(1-\beta)}dW$$
$$\rightarrow v_{db} = \beta v_{db} + \cancel{(1-\beta)}db$$
$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

$V_{dw} = \beta V_{dw} + dW \leftarrow$

- V_dw=0, V_db=0으로 초기화
- 오른쪽 식은 v_dw, v_db의 스케일링에 영향을 주게 되고 학습률도 다시 보정해야 하므로 왼쪽을 선호

7. RMSProp 최적화 알고리즘

RMSprop



- 반복 t에서 현재의 미니배치에 대한 보통의 도함수 dw, db를 계산

💡
$$S_{dw} = \beta S_{dW} + (1-\beta)dW^2$$
$$S_{db} = \beta S_{db} + (1-\beta)db^2$$
$$w := w - \alpha dw/\sqrt{s_{dw}}$$
$$b := b - \alpha db/\sqrt{s_{db}}$$

- 제곱: element wise
- db는 매우 크고 dw는 상대적으로 작음
- b(수직방향)은 더 큰 숫자로 나눠서 업데이트하므로 진동이 줄어듦
- w(수평방향)는 작은 숫자로 나눠서 업데이트함
- 효과: 큰 학습률을 사용해 빠르게 학습하고 수직 방향으로 발산하지 않음
- 진동을 줄이는 효과가 있다는 점에서 모멘텀과 비슷

8. Adam 최적화 알고리즘

Adam optimization algorithm

- 넓은 범위에서 작동하는 딥러닝 알고리즘
- $V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0$
- 도함수 dw,db를 미니배치를 써서 계산

💡
$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dw$$
$$V_{db} = \beta_1 V_{db} + (1-\beta_1)db$$

- β1을 사용한 모멘텀 업데이트

💡
$$S_{dw} = \beta_2 S_{dW} + (1-\beta_2)dW^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$$

- β2을 사용한 RMSprop 업데이트
- 편향 보정+업데이트

$$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t), \quad V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t), \quad S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$$

$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

Hyperparameters choice

- α: 다양한 값을 시도해서 잘 맞는 것을 찾아야 함
- β1: 0.9 (dw의 가중이동평균)
  - 도함수의 평균을 계산하므로 첫번째 모멘트
- β2: 0.999 (dw^2의 가중이동평균)
  - 지수가중평균의 제곱을 계산하므로 두번째 모멘트
- ε: 크게 상관 없지만 10^(-8) 추천
- Adam은 Adaptive moment estimation의 약자

9. 학습률 감쇠

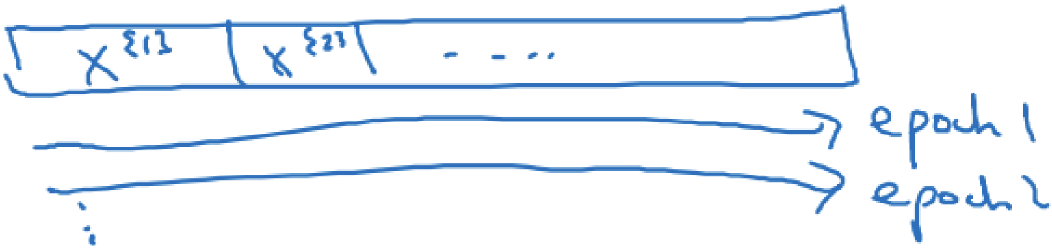
- **학습률 감쇠**: 학습 알고리즘의 속도를 높이기 위해 시간에 따라 학습률을 천천히 줄이는 것

Learning rate decay

- 상당히 작은 미니배치(64,128)에 대해 미니배치 경사 하강법을 구현한다고 가정
  - 노이즈가 잇지만 최솟값으로 향하는 경향
  - 정확하게 수렴하지는 않지만 주변을 돌아다니게 될 것
- 그러나 천천히 학습률 α를 줄인다면
  - α가 큰 초기 단계에서는 여전히 상대적으로 빠른 학습이 가능
  - α가 작아지면 단계마다 진행 정도가 작아지고 최솟값 주변의 밀집된 영역에서 진동하게 될 것

구현 방법

- 1 epoch = 데이터를 지나는 하나의 패스



- decay-rate(감쇠율)은 조정이 필요한 또 다른 하이퍼파라미터

> 💡 $\alpha = \frac{1}{1+decay-rate*epoch-num}\alpha_0$

- α0=0.2, decay-rate=1일 때

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮

- 에포크 수에 대한 함수에서 학습률은 점차적으로 감소
- α0, 감쇠율에 대해 다양한 값을 시도하고 잘 작동하는 값을 찾아야 함

Other learning rate decay methods

공식

- 지수적 감쇠(exponential decay): α<1
    - $\alpha = 0.95^{epoch-num}\alpha_0$
- $\alpha = k/\sqrt{epoch-num}\alpha_0$

- $k/\sqrt{t}\alpha_0$
    - t: 미니배치의 개수
- 이산적 단계로 감소하는 학습률 사용
    - 이산 계단

직접 조작

- α의 값을 시간이나 날마다 직접 보정
- 훈련이 작은 수의 모델로만 이루어진 경우에 가능

해당글은 부스트코스의 [딥러닝 2단계] 4. 최적화 알고리즘 강의를 듣고 작성한 글입니다.