4. 최적화 알고리즘

Mini Batch Gradient Descent

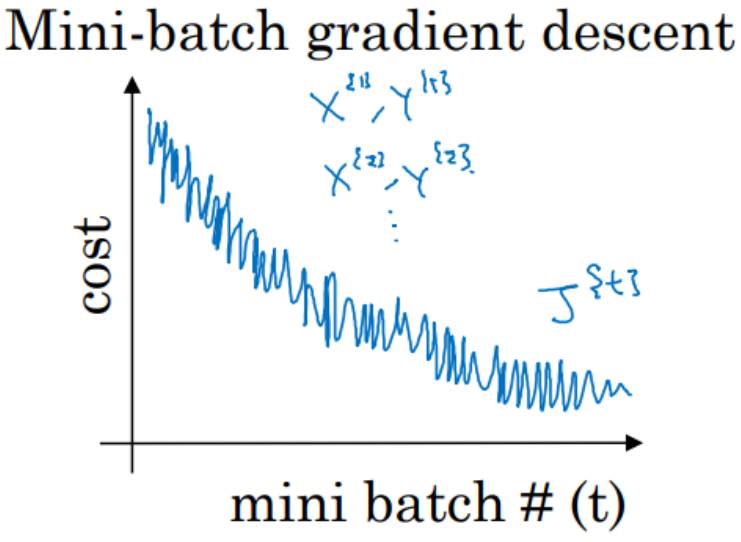
Batch vs. mini-batch gradient descent

- Batch: 전체 훈련 샘플에 대해 training을 진행 후 gradient descent 진행
- Mini-batch: 전체 훈련 샘플을 mini batch로 나눈 후, mini-batch를 training한 후 gradient descent 진행

  ex. 50만개의 데이터를 1000개의 데이터 샘플로 분할
- 표기법
  - i번째 훈련 세트 : $x(i)$
  - l번째 신경망의 z값 : $z[l]$
  - t번째 미니배치 : $X\{t\}, Y\{t\}$

Understanding Mini-batch gradient descent

- mini-batch gradient descent에서는 cost function이 매끄럽게 감소하지 않고 노이즈가 있음 → 미니배치마다 cost 가 다르기 때문



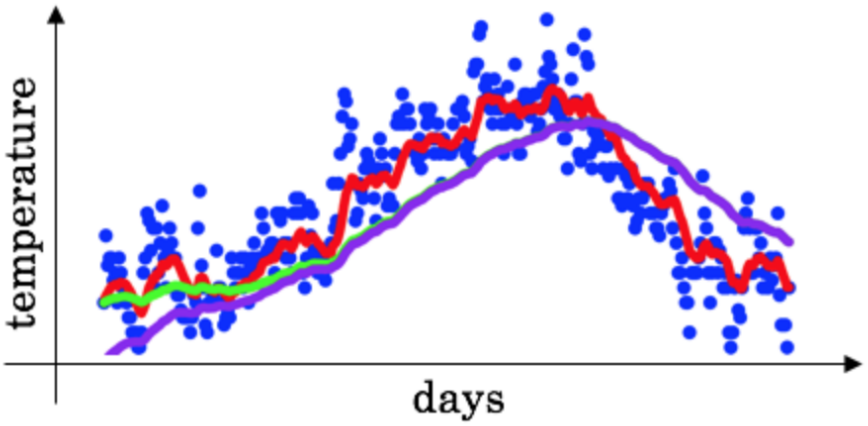
choosing mini-batch size

- m: size of training data
- size = m: 그냥 gradient descent와 같음
  - 각 반복마다 시간이 너무 오래걸림
- size = 1: Stochastic gradient descent, 각각의 샘플이 하나의 미니배치
  - 최솟값에 근사하지만 수렴하지 않음
  - 시간 오래걸림
- 1 ≤ size ≤ m
  - 적당한 크기의 배치 사이즈는 학습 속도를 빨라지게 함
  - 많은 벡터화를 얻음
  - 전체 훈련 세트가 진행되기를 기다리지 않고 진행 가능
- 작은 훈련 세트인 경우: batch gradient descent (m≤2000)
- typical mini-batch size: 64, 128, 256, 512… (2의 제곱수)
- 미니배치에서 X(t), Y(t)가 CPU와 GPU 메모리에 맞는지 확인

Exponentially Weighted Averages

가중 이동 평균 식

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



- *vt* 는 1/1−*β* 기간 동안 기온의 평균을 의미

- *β* = 0.9일 때 10일의 기온 평균

- *β* = 0.5일 때 2일의 기온 평균

- β가 커질수록 line이 smooth해지고, 작을수록 이상치와 노이즈에 민감해짐

Understanding Exponentially Weighted Average

vt=βvt−1+(1−β)θt

β = 0.9일 때 위 식을 하나의 식으로 정리하면 아래와 같음

$$v_100 = 0.1\theta_{100} + 0.1 \times 0.9\theta_{99} + 0.1 \times (0.9)^2\theta_{98} + \cdots$$

얼마의 기간이 이동하면서 평균이 구해졌는가

- *β*=(1−*ε*) 일 때 1/*ε*

구현

```
v[theta] = 0

Repeat {
  Get next theta[t]
  v[theta] = beta * v[theta] + (1-beta)*theta[t]
```

- 구현시 적은 메모리 사용함

Bias Corretion of Exponentially Weighted Averages

Bias Correction

- 구현 식
  - $\boldsymbol{vt/(1-\beta t)}$
- bias corection으로 평균을 더 정확히 계산할 수 있음
- 추정 초기 단계의 값이 정확하지 않음

→ $\boldsymbol{vt/(1-\beta t)}$ 를 취해서 초기 값에서 실제값과 비슷해지게 함

→ theta1, thea2의 가중평균에 편향을 없앤 값이 됨

- t가 커질수록 βt의 값은 0에 가까워지면서 편향 보정 효과가 없어짐

Gradient Descent With Momentum

Momentum

- gradient descent보다 빠르게 동작함
- 구현

On iteration t:

Compute dw, db on current mini-batch.

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

β: friction, V_{dw}: velocity, dw: acceleration

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

⇒ 수직 방향으로는 작은 진동, 수평 방향으로는 더 빠르게 이동.

- 하이퍼파라미터 설정
  - 일반적을 beta = 0.9로 설정하는 것이 일반적임
- momentum 알고리즘에서는 보통 평향 추정을 실행하지 않음

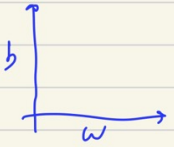
RMSProp

On iteration t:

Compute dw, db on current mini-batch

$$S_{dw} = \beta \cdot S_{dw} + (1-\beta) dw^2 \quad \text{← element-wise}$$
$$S_{db} = \beta \cdot S_{db} + (1-\beta) db^2$$

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}}}, \quad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

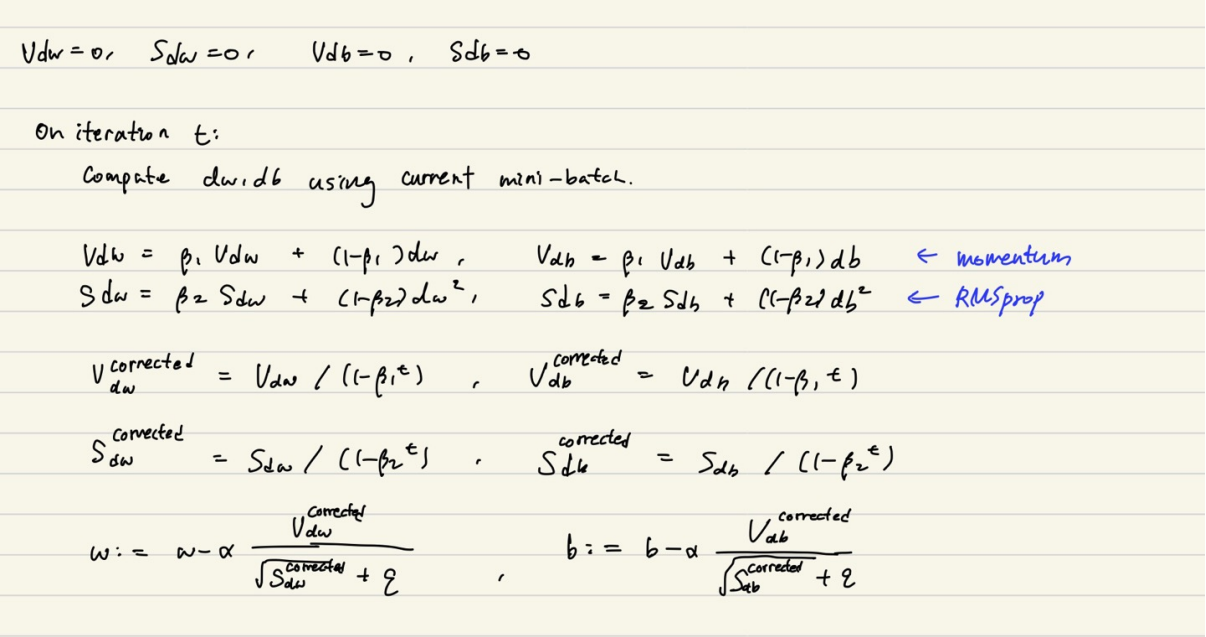


dw은 크게, db는 작게 하여 w 증감은 작게 (horizontal) b 증감은 크게 (vertical).

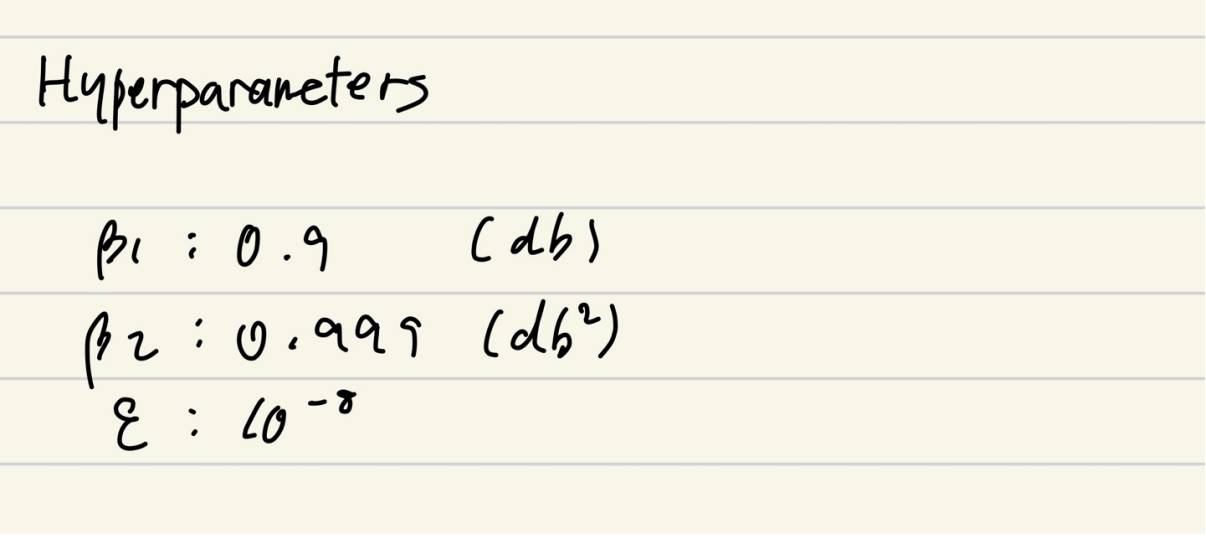
- 미분값이 큰 곳에서는 기존 학습률보다 작은 값으로 업데이트 되고 미분값이 작은 곳에서는 기존 학습률보다 큰 값으로 업데이트 되기 때문에 더 빠르게 수렴할 수 있음

Adam

- momentum + RMSProp
- 구현



- 하이퍼파라미터 설정



Learning Rate Decay

학습이 진행될수록 learning rate가 감소되어야하는 이유

- 학습이 진행될수록 최솟값에 가까워지지만 수렴하지는 않음
- 하지만 learning rate를 점점 줄이면, 학습 초기에는 큰 스텝으로 학습을 진행하고 최솟값에 가까워지면 learning rate가 줄어 한 곳에 수렴할 수 있음
- 구현