10주차

태그	완료

미니 배치 경사 하강법

- 배치 경사 하강법: 전체 훈련 샘플에 대해 훈련 후 경사 하강 진행
- 벡터화는 m개의 샘플에 대해 계산을 효율적으로 만들어준다.

  → 만약 m = 5,000,000개라면? → 느림
- 미니 배치 경사 하강법: 전체 훈련 샘플을 작은 훈련 샘플인 미니 배치로 나눈 후, 미니 배치 훈련 후 경사 하강 진행

  ⇒ mini batch 사용: mini batch 사이즈가 1,000개라면 5,000개의 mini batch가 있는 것

$X^{(i)}$: i번째 훈련 샘플 ($X^{(i)} \in (n\_x, m)$)

$Z^{[l]}$: l 번째 layer

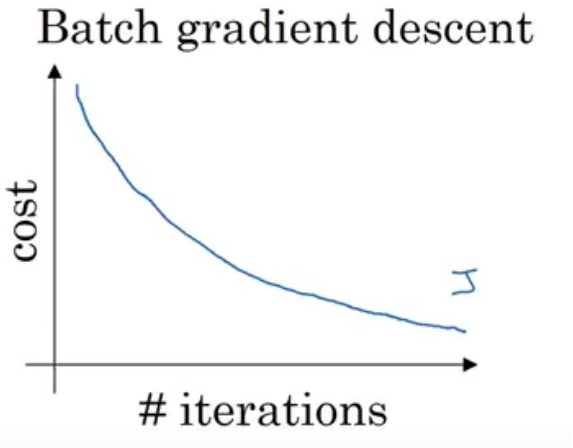
$X^{[t]}, Y^{[t]}$: t번째 mini batch ($X^{[t]} \in (n\_x, 1000), Y^{[t]} \in (1, 1000)$)

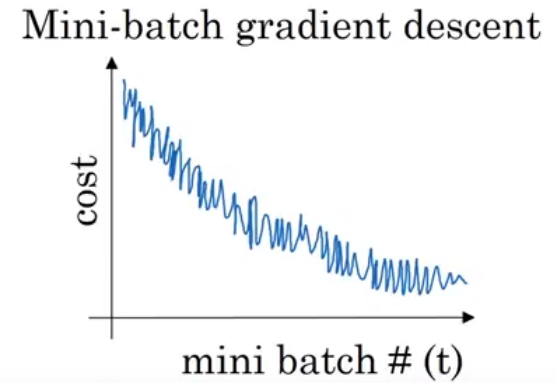
```
for t = 1, 2, ... 5000

    forward propagation of ◗ # 벡터화된 X
        Z^{[1]} = W^{[1]}X^{t} + b^{[1]}
        A^{[1]} = g(Z^{[1]})
        ...
        A^{[L]} = g(Z^{[L]})

    compute cost J^{{t}} = 1/1000◗ + \lambda/2*1000*froben
```

```
    backpropagation to compute gradient of J^{{t}}
    W^{[L]}:= W^{[L]} - \alpha dW^{[L]}, b^{[L]}:= b^{[L]}
```

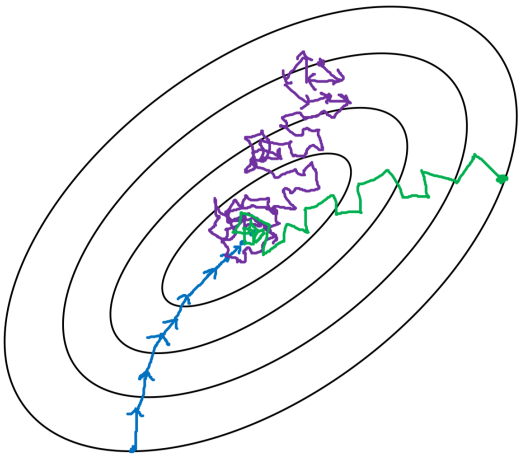
미니 배치 경사 하강법 이해하기

- 배치 경사 하강법에서는 비용 함수 J가 계속 감소하지만, 미니 배치 경사 하강법에서는 매 반복마다 다른 훈련 세트에서 훈련되기 때문에 전체적으로 비용 함수 J는 감소하나 약간의 노이즈가 발생한다.





- 미니 배치 사이즈 고르기 (hyperparameter): 속도를 빠르게 하기 위한 최적의 미니 배치 사이즈를 찾아내야 한다.
    - if mini batch size = m: 배치 경사 하강법 → 전체 훈련 데이터를 모두 훈련시키기 때문에, too long time per iteration
    - if mini batch size = 1: stochastic gradient descent (하나의 샘플이 하나의 batch가 된다) → 한 번에 하나의 샘플만 훈련시키기 때문에 (vectorization이 진행되지 않은 것과 마찬가지) 비효율적인 진행 방식이다.
    - In practice: 1과 m 사이의 값을 사용한다. → 가장 좋음
        - 여러 개의 vectorization을 얻기 때문에 빠르다.
        - 전체를 여러 개의 epoch으로 나눔, 전체 훈련 세트가 진행되기를 기다리지 않고 진행할 수 있다.



- how to choose the size of mini batch?

  -If small train set (2000개 이): 배치 경사 하강법을 사용 (전체 훈련 샘플을 하나의 batch로 설정)

  -typical mini batch size: 64, 128, 256, 512 → 2의 제곱수를 사용하는 것을 추천

  -make sure mini batch fits in CPU / GPU memory: 정해진 CPU / GPU memory 크기 이내에서 memory를 사용하도록 mini batch size를 잘 정해야 한다.

지수 가중 이동 평균

- 경사 하강법보다 빠른 최적화 알고리즘을 이해하기 위해선 지수 가중 이동 평균을 이해해야 한다. 최근 데이터에 더 많은 영향을 받는 데이터의 평균 흐름을 계산하기 위해 지수 가중 이동 평균을 구한다. **지수 가중 이동 평균은 최근 데이터에 더 높은 가중치를 준다.**
- θ_t를 t번째 날의 온도, v_{t-1}은 그 전 날의 온도라고 했을 때, 지수 가중 이동 평균의 식은 다음과 같다. v_t는 t번째 날의 온도의 경향이다. 첫째 항은 과거의 경향성, 둘째 항은 새로운 경향성이다.

$$$$

-v_t approximates to average over $\frac{1}{1-\beta}$*days temperature

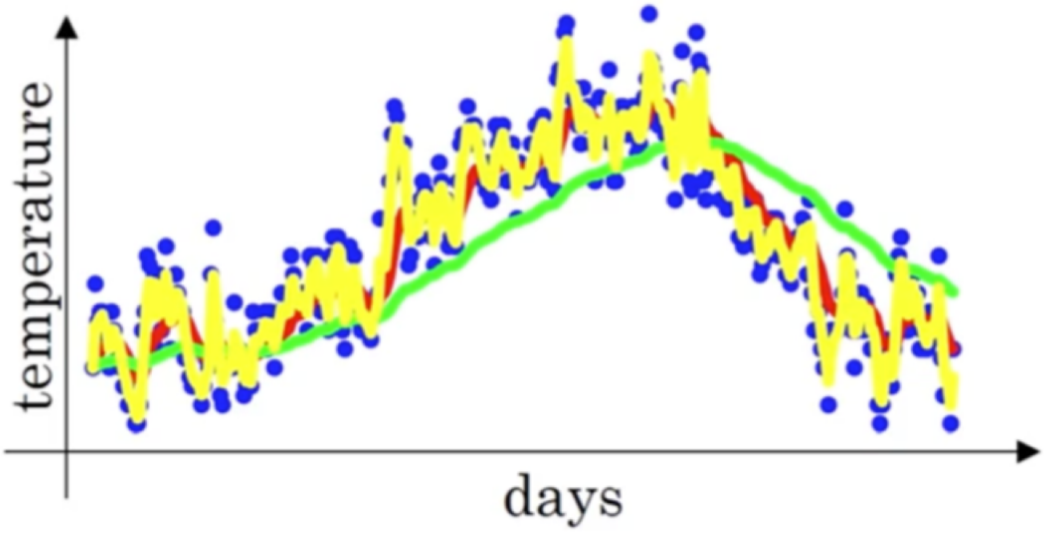
-if β = 0.9, then v_t = 10일 동안 기온의 평균(red)

-if β = 0.98, then v_t = 50일 동안 기온의 평균(green)

→ β 값이 클수록 더 많은 날짜의 기온을 사용하기 때문에 곡선이 더 부드러워진다. 그러나 다양한 날짜의 기온을 사용하기 때문에, 이 곡선은 올바른 값에서 더 멀어진다.

-if β = 0.5, then v_t = 2일 동안 기온의 평균(yellow)

→ 온도 변화에 더욱 민감하게 반응한다.



지수 가중 이동 평균 이해하기

$V_{100} = 0.1 * \theta_{100} + 0.9 * V_{99}$ | 이때 $V_{99} = 0.1 * \theta_{99} + 0.9 * V_{98}$

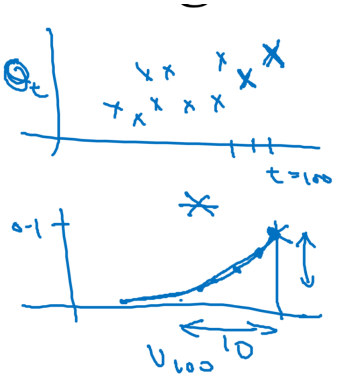
그렇다면,

$V_{100} = 0.1 * \theta_{100} + 0.9 * (0.1 * \theta_{99} + 0.9 * V_{98})$ | $V_{98} = 0.1 * \theta_{98} + 0.9 * V_{98}$

$V_{100} = 0.1 * \theta_{100} + 0.9 * (0.1 * \theta_{99} + 0.9 * [0.1 * \theta_{98} + 0.9 * V_{98}]) + \cdots$

$\Rightarrow V_{100} = 0.1 * \theta_{100} + 0.1 * 0.9 * \theta_{99} + 0.1 * 0.9^2 * \theta_{98} + 0.1 * 0.9^3 * \theta_{98} + \cdots$ 이런식으로 계속된다.

- 과거의 θ일수록 현재의 경향 v_t를 표현하는 데 더 적은 영향을 끼치고 있다.
- 위의 그래프는 θ의 그래프, 아래의 그래프는 θ의 계수에 대한 그래프이다. 아래의 그래프는 지수적으로 감소하는 형태이다. v_t는 두 그래프를 element-wise하게 곱함으로써 구할 수 있다.



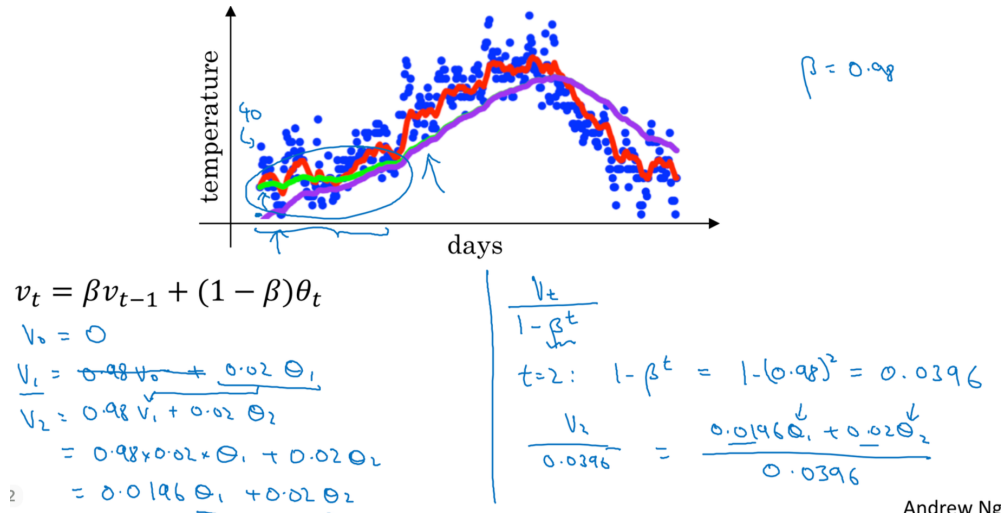
- 얼마의 기간에 걸쳐 구한 평균인가?
  - $(1-\varepsilon)^{\frac{1}{\varepsilon}} = \frac{1}{e}$
    - $1-\varepsilon = \beta$
    - $\frac{1}{\varepsilon}$은 몇 일에 걸친 평균 온도인지를 알 수 있다.
    - 예를 들어 ε이 0.1이라면 β = 0.9이고 이는 10일에 걸친 온도의 평균이다.

```
v = 0
repeat{
    get next \theta_{t}
    v:= \beta*v + (1-\beta)*\theta_{t}
}
```

- 지수 가중 이동 평균의 장점: 반복해도 v에 계속 값을 덮어씌우면 되기 때문에 메모리를 적게 사용할 수 있다. 또한 업데이트 식도 한 줄이라서 편리 & 효율적이다.

지수 가중 이동 평균의 편향보정

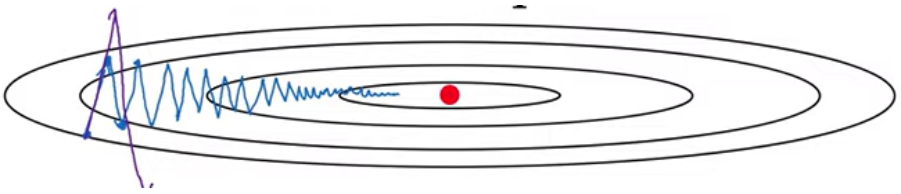
- bias correction



- 만약 지수 가중 이동 평균의 식대로만 v_t를 구하면 (왼쪽 식) 실제로 그려지는 곡선은 보라색 곡선이 된다. 그러나 보라색 곡선에는 너무 낮은 값이 존재한다 → 편향을 제거해야 한다!
- 오른쪽 식처럼 θ들의 가중 평균인 v_t를 $1-\beta^t$로 나눠주면 θ들의 가중 평균에서 편향이 제거된 초록색 그래프가 그려지고, 실제 값과 비슷하게 된.
- t가 커질수록 β^t는 0에 가까워지므로 편향 보정의 효과가 거의 없어지게 된다.
- 초기 단계에서 편향 보정으로 더 정확한 온도 평균의 추정값을 얻을 수 있다. (보라색 그래프에서 초록색 그래프로)

Momentum 최적화 알고리즘

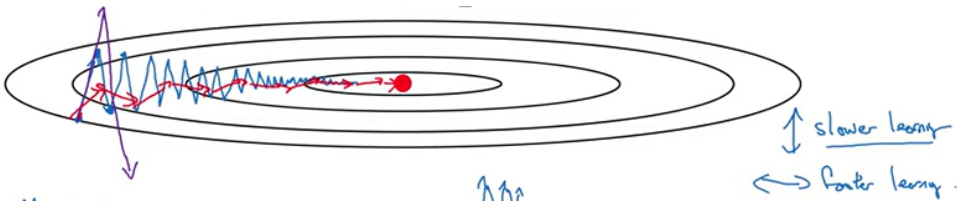
- 미니 배치 경사 하강법을 시행하면 다음과 같이 진동하며 최적점에 수렴한다. 그러나 이 방법은 경사 하강법의 속도를 느리게 하고 더 큰 learning rate를 사용하는 것을 막는다. 또한 수직축에서는 진동을 막기 위해 학습이 더 느려지기를 원하지만 수평축에서는 왼쪽에서 오른쪽으로 학습이 빠르게 이루어지기를 원한다.



- momentum 경사 하강법은 일반적인 경사 하강법보다 빠르게 작동한다.
  - 경사에 대한 지수 가중 평균 (VdW, Vdb)을 계산하고, 그 값으로 가중치를 업데이트한다.
  - 따라서 momentum 경사 하강법은 아래와 같은 알고리즘을 따른다.

```
VdW = 0; Vdb = 0
Iteration t:
    compute dW, db on current mini-batch
    VdW = \beta*VdW + (1 - \beta)*dW
    Vdb = \beta*Vdb + (1 - \beta)*db

    W:= W - \alpha*VdW
    b:= b -\alpha*Vdb

    # two hyperparameters: \alpha(learning rate), \beta
```

→ 이 알고리즘을 통해 진동을 줄여 훨씬 부드럽게 최적점에 수렴할 수 있게 된다. 수직축에서는 진동을 줄이면서도 수평축에서는 속도를 빠르게 한다.



→ 또한, 밥그릇 모양에서 공을 굴려 최적점으로 보낼 때, β는 1보다 작은 값이므로 friction, VdW, Vdb는 공의 velocity, dW, db는 공의 momentum(acceleration)의 역할을 한다고 이해할 수 있다.

→ momentum 경사 하강법을 구현할 때는 편향 보정을 거의 하지 않는다. step이 10번 이상이면 이동 평균에 더 이상 편향 보정을 하는 효과가 미미하기 때문이다.

→ 몇몇 논문에서 VdW를 계산할 때 $1-\beta$를 생략하는데, 이는 VdW를 1/$(1-\beta)$로 scaling한 것이다. 따라서 $\alpha = \frac{1}{(1-\beta)}$가 되어야 한다. 그러나 실제로는 두 가지 방법 모두 잘 작동한다. (교수님은 1-β가 있는 수식을 더 선호한다)

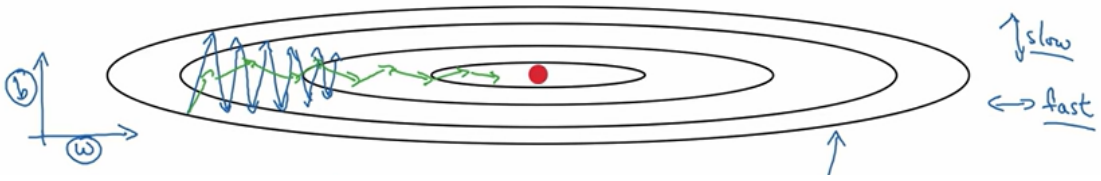
RMSProp 최적화 알고리즘

- root mean square prop algorithm → 마찬가지로 경사 하강법을 빠르게 한다. 과거의 기울기는 적게 반영하고 최근의 기울기는 많이 반영하는 지수 가중 평균을 사용한다.
    - 수평 방향(W): 빠르게 이루어지기를 원함/ 수직 방향(b): 느리게 이루어지기를 원함(진동 적게)

```
Iteration t:
    compute dW, db on current mini-batch
```

```
# 도함수의 제곱을 지수 가중 평균한 것
# dW^2은 element-wise하게 제곱한
Sdw = \beta*Sdw + (1 - \beta)*dW^2 <- small
Sdb = \beta*Sdb + (1 - \beta)*db^2 <- large

W:= W - \alpha*dW/\root{SdW}
b:= b - \alpha*db/\root{Sdb}
```

- 수평 방향(W)에서의 기울기는 완만해 미분값 SdW의 값은 상대적으로 작다. 따라서 수평 방향(W)은 기존의 learning rate보다 빠르게 업데이트되고, 이는 더 빠르게 수렴하도록 한다. 반면 수직 방향(b)에서의 기울기가 훨씬 가파르기 때문에 미분값 Sdb가 상대적으로 크다. 따라서 수직 방향(b)으로는 기존의 learning rate보다 느리게 업데이트되고, 이는 진동을 줄인다.
- 이 알고리즘을 사용하면 진동을 줄이면서도 빠르게 최적점으로 수렴할 수 있다.



- 주의할 점: W, b 업데이트 식에서 SdW, Sdb의 값이 너무 작지 않도록 한다(폭발할 수 있음) → SdW, Sdb에 ε = 10^(-8)을 더함으로써 보완 가능

Adam 최적화 알고리즘

- Adam 최적화 알고리즘 = momentum + RMSProp
- Adam = Adaptive moment estimation

```
# 초기화
VdW = 0; SdW = 0; Vdb = 0; Sdb = 0

Iteration t:
    compute dW, db using current mini-batch

    # momentum
    VdW = B1*VdW + (1 - B1)*dW
    Vdb = B1*Vdb + (1 - B1)*db
```

```
# RMSProp
SdW = B2*SdW + (1 - B2)*dW^2
Sdb = B2*Sdb + (1 - B2)*db^2

# 편향 보정 (일반적으로 Adam에서는 편향 보정을 한다.)
VdW_corrected = VdW / (1 - B1^t)
Vdb_corrected = Vdb / (1 - B1^t)
SdW_corrected = SdW / (1 - B2^t)
Sdb_corrected = Sdb / (1 - B2^t)

# W, b 업데이트
W:= W-A*VdW_corrected / \root(SdW_corrected) + E
b:= b-A*Vdb_corrected / \root(Sdb_corrected) + E
```

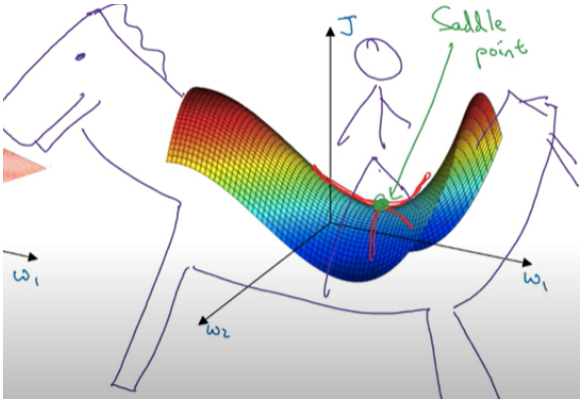
- 여러 개의 hyperparameter가 있다 (α 빼고 대부분 보정하지 않는다)
  - α: learning rate → needs to be tuned
  - β_1: 0.9 (dW)
  - β_2: 0.999 (dW^2)
  - ε: 10^(-8)

학습율 감쇠

- 학습율 감쇠: 시간에 따라 learning rate를 감소시키는 것 → 학습 알고리즘의 속도를 높이는 한 가지 방법
- 시간에 따라 학습율이 줄어들면 step size가 작아짐에 따라 최적점에 더 잘 수렴할 수 있다.
- 1 epoch = 1 pass through data
  - $\alpha = \frac{1}{1+decayrate*epochnum} * \alpha_0$: hyperparameter α_0, decay_rate 조절
  - exponential decay: $\alpha = 0.95^{epochnum} * \alpha_0$
  - $\alpha = \frac{k}{epochnum} * \alpha_0$
  - $\alpha = \frac{k}{t**0.5} * \alpha_0$: t는 미니 배치 개수

지역 최적값 문제

- 고차원 비용함수에서 경사가 0인 경우는 대부분 지역 최적값이 아니라 안장점이다.
    - 안장점 주위는 대부분 경사가 0이므로 지역 최적값에 갇힐 위험이 있다 → 그러나 big neural network를 사용하면 괜찮다.
    - 지역 최적값에 빠지면 대부분 경사가 0에 가깝기 때문에 학습이 느려질 수 있다. 또한, 다른 방향으로의 전환이 없다면 안장 지대에서 벗어나기 힘들다 → Adam, RMSProp과 같은 최적화 알고리즘을 통해 빠져나올 수 있다.



Quiz

✕ 3. 왜 최적의 미니배치 크기는 1이나 m이 아니라, 그 중간 값일까요? * 0/1

- [x] 만약 미니배치 크기가 1이라면, 미니배치 내의 예제들 간의 벡터화 장점을 상실하게 됩니다. ✓
- [ ] 만약 미니배치 크기가 m이라면, 진전을 이루기 전에 전체 학습 데이터 집합을 처리해야 하는 배치 경사 하강법으로 빠지게 됩니다.
- [ ] 만약 미니배치 크기가 1이라면, 진전을 이루기 전에 전체 학습 데이터 집합을 처리해야 하는 배치 경사 하강법으로 빠지게 됩니다.
- [ ] 만약 미니배치 크기가 m이라면, 미니배치 내의 예제들 간의 벡터화 장점을 상실하게 됩니다.

정답

- [x] 만약 미니배치 크기가 1이라면, 미니배치 내의 예제들 간의 벡터화 장점을 상실하게 됩니다.
- [x] 만약 미니배치 크기가 m이라면, 진전을 이루기 전에 전체 학습 데이터 집합을 처리해야 하는 배치 경사 하강법으로 빠지게 됩니다.

- 만약 미니배치의 크기가 1이라면, 미니배치 내의 훈련 샘플들 간의 벡터화의 장점을 상실한다.
- 만약 미니배치의 크기가 m이라면, 한 번의 진전을 이루기 전에 모든 훈련 샘플들을 처리해야 하는 배치 경사 하강법으로 빠지게 된다.