4. 최적화 알고리즘

1. 미니 배치 경사하강법

1. **배경**

- 머신러닝을 적용하는 것은 매우 실험적인 과정

→ 잘 작동되는 모델을 찾기 위해 많은 훈련을 거쳐야 하기 때문

- 큰 데이터 세트에서 훈련하는 것: 매우 느림

→ 좋은 최적화 알고리즘을 찾아 효율성을 높여야 함.

2. **벡터화 vs 미니 배치 경사 하강법**

- **벡터화**: m개의 샘플에 대한 계산을 효율적으로 만들어 줌
    - 명시적인 반복문 없이, 훈련 세트 진행 가능

$$\underset{(n_x,m)}{X} = [x^{(1)}\ x^{(2)}\ x^{(3)}\ \dots \qquad \dots\ x^{(m)}]$$

$$\underset{(1,m)}{Y} = [y^{(1)}\ y^{(2)}\ y^{(3)}\ \dots \qquad \dots\ y^{(m)}]$$

What if $m = 5{,}000{,}000$?

- m이 매우 크면, 여전히 느릴 수 있음.

- **배치 경사 하강법**: 전체 훈련 샘플에 대해 훈련 후 경사 하강법을 구현
    - 경사 하강법의 작은 한단계를 밟기 전에 모든 훈련세트를 처리 필요

    - 또 경사하강법의 다음 단계를 밟기 전에 다시 오백만개의 전체 훈련샘플을 처리

  → **오백만 개의 거대한 훈련 샘플을 모두 처리하기 전에 경사하강법이 진행되도록 하면 더 빠른 알고리즘을 얻을 수 있다.**

- **미니 배치 경사 하강법:** 전체 훈련 샘플을 작은 훈련세트들로 나누고(이를 미니배치라고 함), 미니배치 훈련후 경사하강법을 진행
    - ex) 전체 훈련세트가 5,000,000일 때, 각각의 미니배치가 1,000개의 샘플을 갖는다고(사이즈가 1,000인 미니배치 5,000개) 가정하고 훈련 및 경사 하강법을 진행
        - i번째 훈련 세트 : x^(i)
        - l번째 신경망의 z값 : z^[l]
        - t번째 미니배치 : X^{t},Y^{t}

      → X^{t} 의 차원: (n_x, m), Y^{t}의 차원: (1, m)

3. **미니 배치 경사 하강법**

- 훈련세트에서 미니배치 경사 하강법을 실행하기 위해 t=1... 5,000 반복문 돌리기. 반복문 안에서는 <u>**한 단계의 경사하강법**</u>을 구현.
    - X^{t},Y^{t}를 사용하여 구현
    - 입력 X^{t}에 대한 **정방향 전파** 구현: z^[1] = W^[1]X^{t} + b^[1]
    - A^[1] = g^[1](Z^[1]), … ,A^[l] = g^[l](Z^[l])
    - 비용함수 J 계산:

$$J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|w^{[l]}\|_F^2 .$$

    - **역전파 구현**: J^{t}에 대응하는 경사를 계산하기 위함.
        - X^{t},Y^{t}를 사용, 가중치 업데이트

Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$, $b^{[l]} := b^{[l]} - \alpha db^{[l]}$

- **1 epoch(에포크) : 훈련세트를 거치는 한 반복**

→ 배치 경사 하강법에서 훈련 세트를 거치는 1 반복은 오직 하나의 경사 하강 단계 만을 할 수 있게 한다.

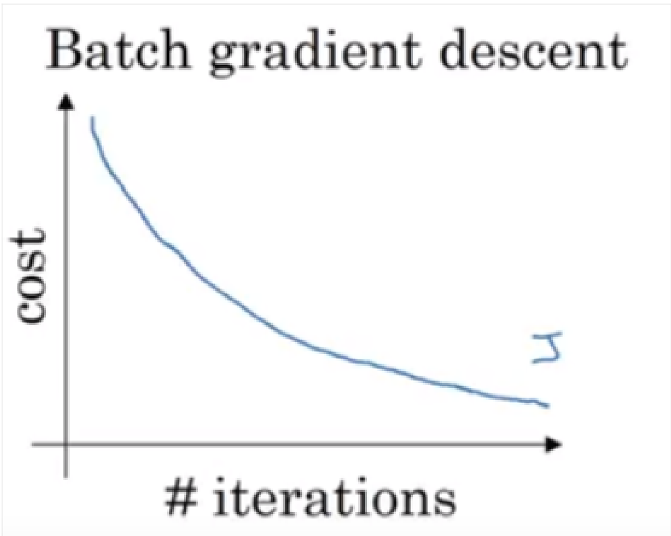
→ 미니 배치 경사 하강법의 경우, 훈련 세트를 거치는 1 반복은 5,000개의 경사 하강 단계를 거치도록 한다.

→ **훈련 세트가 많다면, 미니 배치 경사하강법이 훨씬 더 빠르게 실행됨.**

2. 미니 배치 경사하강법 이해하기

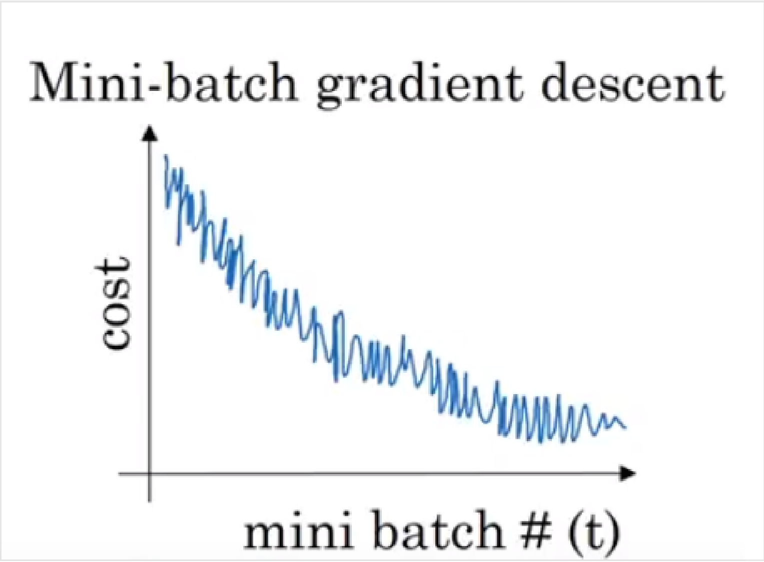
1. **배치 경사 하강법**

- 모든 반복에서 전체 훈련 세트를 진행
- 각 반복마다 비용이 감소



2. **미니 배치 경사 하강법**

- 모든 반복마다 감소 X
- 전체적으로는 **비용함수가 감소하는 경향**을 보이지만 **노이즈가 많이 발생.**
- 노이즈(진동)가 발생하는 이유: X^{1}과 Y^{1}이 상대적으로 쉬운 미니배치라서 비용이 약간 낮은데 우연적으로 X^{2}와 Y^{2}가 더 어려운 미니배치라서 등의 이유로 비용이 약간 더 높아질 수 있다.



3. **미니 배치 사이즈**

i. **m이 훈련세트 크기일 때, 미니배치 크기가 m과 같은 경우 = 배치 경사 하강법**

- 하나의 미니배치만을 갖게 되고 이 미니배치의 크기는 전체 훈련 세트와 같다.
- 미니배치 크기를 m으로 설정하는 것은 일반적인 경사 하강법과 같다.

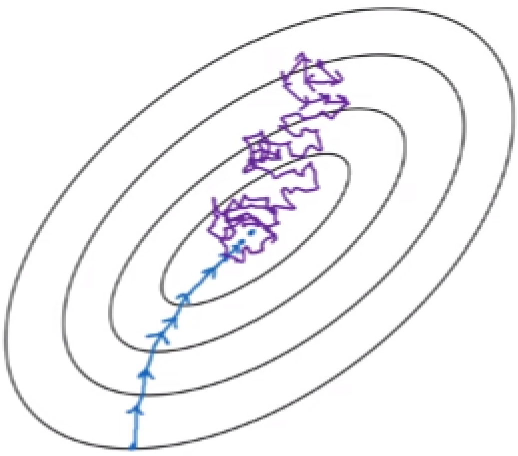
ii. **미니배치 크기 = 1 인 경우; 확률적 경사 하강법**

- 각각의 샘플은 하나의 미니배치.
- 첫번째 미니 배치인 X^{1}과 Y^{1}을 살펴보면 미니배치 크기가 1일 때: 첫번째 훈련샘플과 같다.

→ 즉, 첫번째 훈련샘플로 경사 하강법을 하는 것.

- 두번째 미니배치: 두번째 훈련샘플과 같고 이에 대한 경사하강단계를 취하게 됨.

→ 이런 식으로 한번에 하나의 훈련 샘플만을 살펴보며 계속 진행하는 것.



i), ii)의 두가지 극단적인 경우가 비용함수를 최적화할 때, i)은 파란색 화살표의 경로를 따르고 ii)는 보라색 화살표의 경로를 따른다.

i) 상대적으로 노이즈가 적고, 큰 단계를 취함.

ii) 대부분의 경우, 전역 최솟값으로 가게 되지만 어떤 경우는 잘못된 곳으로 가기도 한다. 극단적으로 노이즈가 많을 수도 있지만, 평균적으로 좋은 방향으로 향한다.

iii. 실제로 우리가 사용하는 미니배치 크기는 1과 m 사이일 것

(1은 상대적으로 작고 m은 상대적으로 큰 값)

- **배치 경사 하강법**: 미니배치의 크기는 m과 같다.

→ 매우 큰 훈련 세트를 모든 반복에서 진행하게 됨

→ 단점: 한 반복에서 너무 오랜 시간이 걸림

- **확률적 경사 하강법**: 하나의 샘플만 처리한 뒤에 계속 진행 가능해 간단, 노이즈도 작은 학습률을 사용해 줄일 수 있음

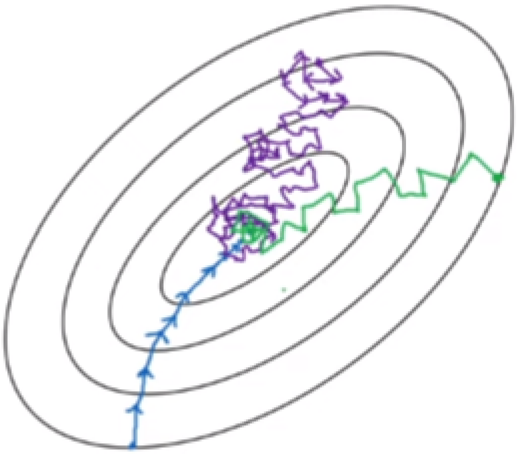
→ 단점: 벡터화에서 얻을 수 있는 속도 향상을 잃게 됨 + 한 번에 하나의 훈련세트를 진행하기 때문에 각 샘플을 진행하는 방식이 매우 비효율적

따라서, 가장 잘 작동하는 값은 1과 m사이의 있는 값이다.

장점

1. 많은 벡터화를 얻음 : 미니배치의 크기가 1000개의 샘플이라면 1000개의 샘플에 벡터화를 하게 된다. 그렇다면 한 번에 샘플을 진행하는 속도가 더 빨라지게 된다.

2. 전체 훈련세트가 진행되기를 기다리지 않고 진행가능 : 각각의 훈련세트의 에포크는 5000번의 경사 하강 단계를 허용



iii)은 초록색 화살표를 따른다.

- 항상 최솟값으로 수렴한다고 보장할 수 X
- 더 일관되게 전역의 최솟값으로 향하는 경향이 있음.
- 매우 작은 영역에서 항상 정확하게 수렴 or 진

4. **미니배치 크기가 m이나 1이 아닌 그 사이의 값이어야 한다면 그 값을 어떻게 선택?**

- 훈련세트가 **작다면**(2000개 이하) 모든 훈련세트를 한번에 학습시키는 **배치 경사하강**을 진행한다.
- 2000개 이상의 훈련세트가 **클 경우** 전형적으로 선택하는 **미니배치 사이즈: 64, 128, 256, 512와 같은 2의 제곱수**.
- 미니배치에서 모든 X^{t}와 Y^{t}가 CPU와 GPU 메모리에 맞는지 확인.

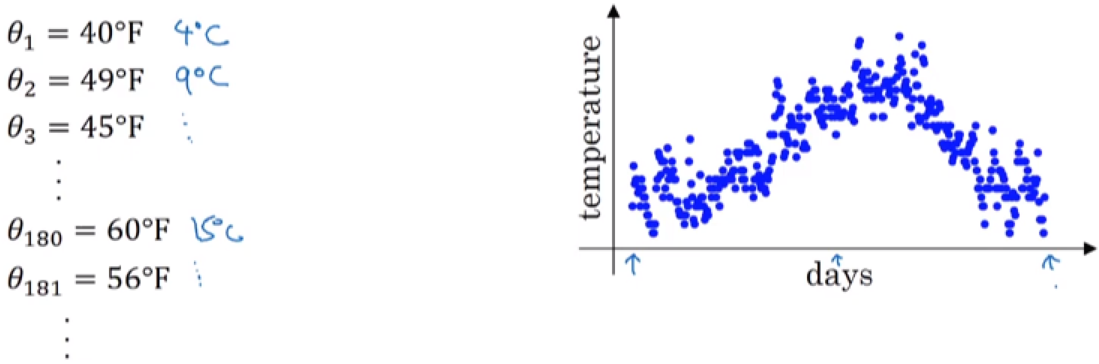
→ 애플리케이션과 하나의 훈련샘플의 크기에 달려있다. 그렇지 않으면 성능이 갑자기 떨어지고 훨씬 나빠지게 된다.

- 미니배치 크기는 빠른 탐색을 통해 찾아내야 하는 또 다른 하이퍼파라미터
- 가장 효율적이면서 비용함수 J를 줄이는 값을 찾아내야 한다.

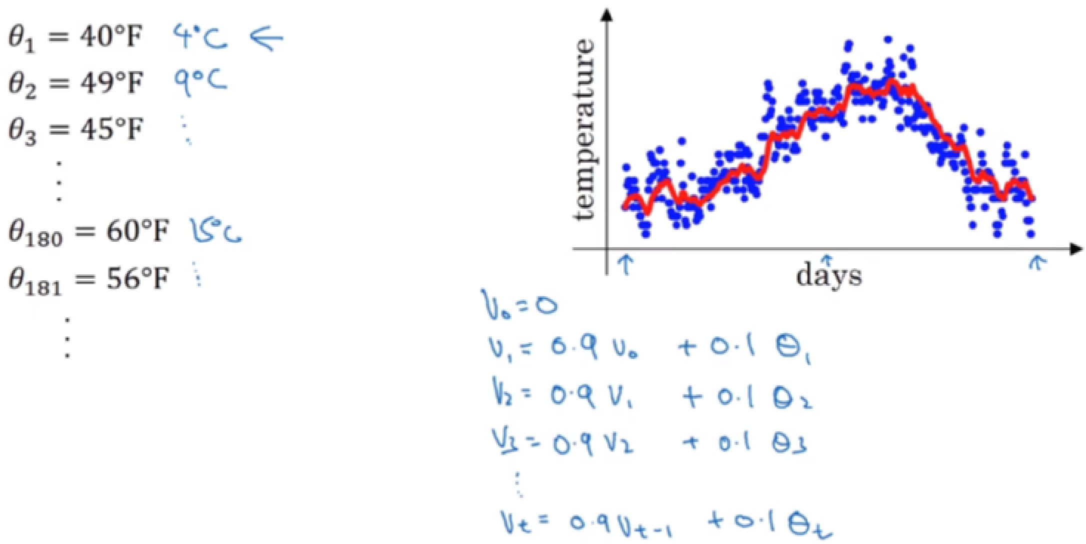
3. 지수 가중 이동 평균

1. **지수 가중 이동 평균 이해**

- ex) Temperatutr in London



- 데이터에 약간의 노이즈 존재
- 역 평균이나 이동 평균의 흐름을 계산하는 방법
  - v_0를 0으로 초기화
  - 매일 이전의 값에 0.9를 곱하고 여기에 0.1 곱하기 해당 날의 기온을 더하기
  - 일반적인 식: $V_t = 0.9V_{t-1} + 0.1\theta_t$
  - **일별 기온의 지수가중평균 그래프**



2. **Exponentially weighted averages**

- V_t를 계산한 값: 대략적으로 1/(1-β) 곱하기 일별 기온의 평균과 같다.

i) **β=0.9**와 같은 경우

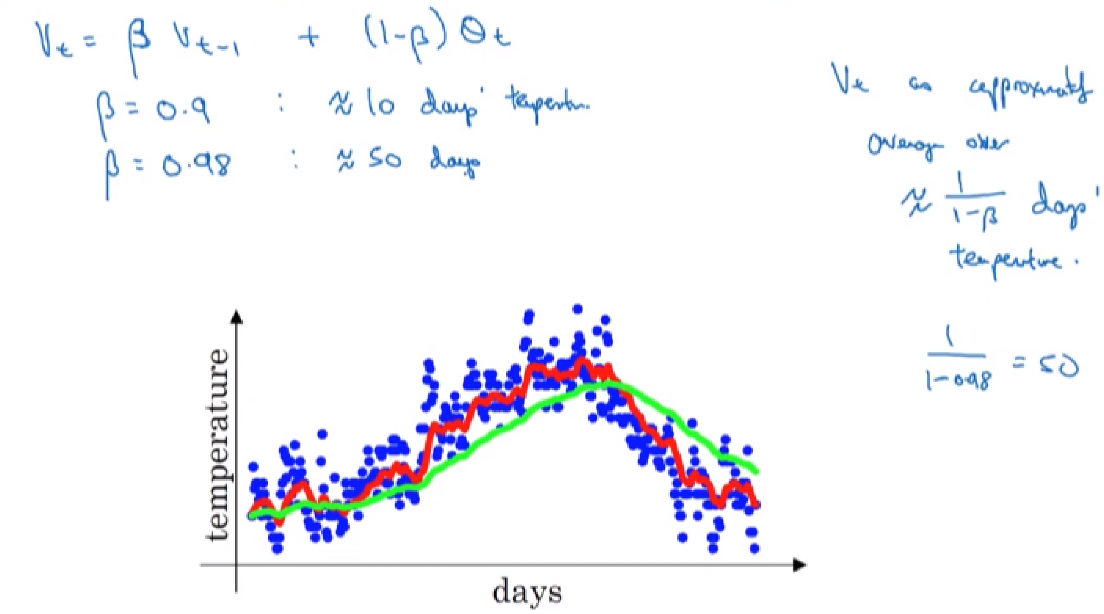
- 10일 동안 기온의 평균과 같다.
- 그래프의 붉은색 선.

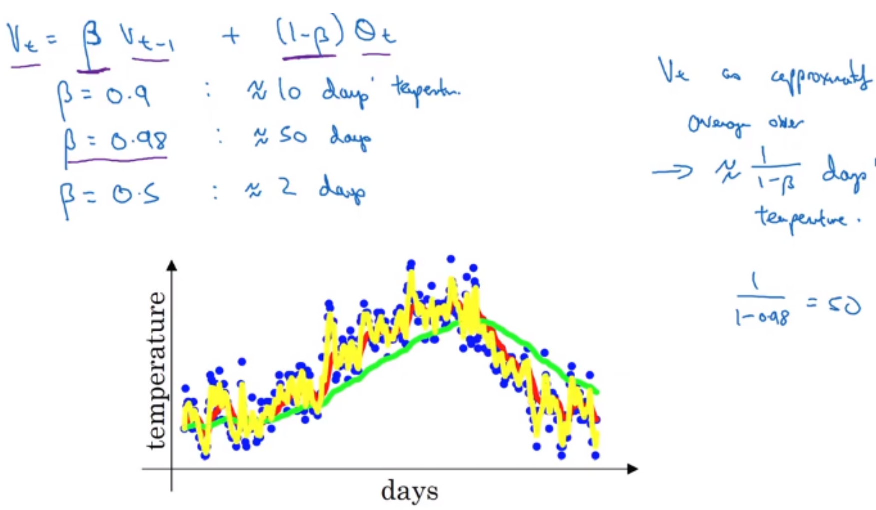
ii) β가 1과 매우 가까운 경우(**β=0.98**)

- 1/(1-0.98)은 50과 비슷하므로 이 값은 기온의 평균과 거의 같다.
- 그래프의 초록색 선.

iii) **β=0.5**

- 공식에 의해서 2일의 기온만 평균하는 것과 같다.
- 오직 2일만의 기온을 사용했기에 더 노이즈가 많고 이상치에 더 민감.
- 그러나 기온 변화에 더 빠르게 적응한다.
- 그래프의 노란색 선.





- **β값이 클수록** 더 많은 날짜의 기온의 평균을 이용하기 때문에 **곡선이 더 부드러워짐**을 알 수 있다.
- 지만 더 큰 범위에서 기온을 평균하기 때문에 **곡선이 올바른 값에서 더 멀어진다.**

→ 기온이 바뀔 경우에 지수가중편균 공식은 더 느리게 적응한다. 따라서 지연되는 시간이 더 크다.

- **빨간색 곡선이 초록색이나 노란색 곡선보다는 더 나은 평균을 제공한다.**

4. 지수 가중 이동 평균 이해하기

1. **Exponentially weighted averages**

- 지수가중평균: **신경망 훈련에 사용하는 몇 가지 최적화 알고리즘의 주요 구성요소**
- β=0.9 로 설정하고 앞서 나온 지수가중이동평균 식을 하나의 값으로 표현:

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

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

$$\boxed{v_{99} = 0.9v_{98} + 0.1\theta_{99}}$$

$$\boxed{v_{98} = 0.9v_{97} + 0.1\theta_{98}}$$

$$\cdots$$

$$0.1\theta_{98} + 0.9v_{97}$$

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

$$= 0.1\theta_{100} + 0.1 \times 0.9 \cdot \theta_{99} + 0.1(0.9)^2\theta_{98} + 0.1(0.9)^3\theta_{97} + 0.1(0.9)^4\theta_{96} + \cdots$$

- **$v_{100}=0.1\theta_{100}+0.1\times0.9\theta_{99}+0.1\times(0.9)^2\theta_{98}+\cdots$**

→ 가중치의 합(즉, θ_100의 가중치의 평균)

- 그림으로 표현하면 지수적으로 감소하는 그래프**: v100을 구하는 과정에서 각 요소별 곱셉(0.1 x (0.9)^n)을 해서 더하기 때문**

- **편향 보정**: 위의 식들에서 앞에 곱해지는 계수들을 모두 더하면 1 또는 1에 가까운 값이 되는

2. **얼마나 많은 날들이 평균적인 온도가 될까?**

- β가 0.9와 같을 때, 지난 10일간의 온도에만 초첨을 맞춰 가중평균을 계산

→ 10일 뒤에는 현재 날짜의 가중치의 1/3으로 줄어들게 된다. (0.9^10 ~ 0.35 ~ 1/e)

- β가 0.98이라면 0.98^50이 대략적으로 1/e와 같다.

→ 처음 50일 동안의 1/e보다 가중치는 더 커질 것, 더 가파르게 감소

→ 직관적으로 50일의 온도의 평균은 더 급격히 빠르다. 왜냐하면 ε은 0.02이기 때문

따라서 1/ε 은 50과 같다(1/(1-β)와도 대략적으로 같다.)

- **β=(1−ε) 라고 정의 하면, (1−ε)^n = 1/e 를 만족하는 n 이 그 기간이 되는데, 보통 1/ε 으로 구할 수 있다.**

3. **Implementing exponentially weighter averages**

- v를 0으로 초기화(v에 아래 첨자 θ를 한 표기법을 사용하기도 함)
- 반복문으로 나타내기
  - **v_θ를 0**으로 설정
  - 각각의 날짜마다 **다음 θ_t**를 얻는다.

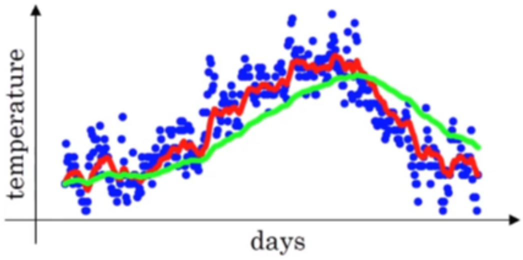
- 그 후, **v_θ := β*v_θ+(1-β)θ_t**로 업데이트

- 이렇게 지수평균을 얻는 식의 장점은 **아주 적은메모리를 사용**한다는 것
    - v_θ 실수 하나만을 컴퓨터 메모리에 저장하고 가장 최근에 얻은 값을 식에 기초에 덮어쓰기만 하면 되기 때문

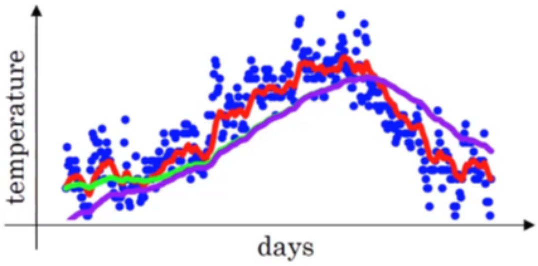
5. 지수 가중 이동 평균의 편향보정

- 편향보정은 평균을 더 정확하게 계산할 수 있게 한다.

1. **편향 보정**



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



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

- 작성된대로 공식을 구현하다면 **β=0.98**일때, 초록색 곡선을 얻지 못하고, 두 번째 그래프의 보라색 곡선을 얻게 된다. 보라색 곡선이 매우 낮은 곳에서 시작함을 알 수 있다.

→ **어떻게 수정해야 할까?**

- 이동평균을 구할 때 v_0 = 0으로 초기화, **v_1 = 0.98v_0 + 0.02θ_1**
    - 그러나 v_0가 0이기 때문에 오른쪽 식의 첫 항은 사라지게 된다.

- 따라서 첫째 날의 온도가 화씨 40도면 v_1의 값은 0.02*40 = 8.

→ 값이 훨씬 낮아져서 첫 번째 날의 온도를 잘 추정할 수 없다.

- **v_2 = 0.98v_1 + 0.02θ_2**
- v_1값을 대입하면 v_2 = 0.98 * 0.02θ_1 + 0.02*θ_2가 된다. 이는 0.0196θ_1 + 0.02θ_2 이다.
    - θ_1과 θ_2가 양수라고 가정한다면 v_2를 계산한 값은 θ_1이나 θ_2보다 훨씬 더 작아질 것이다. 한 해의 첫 두 날짜를 추정한 값이 좋지 않은 추정이 된다.

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

$$V_0 = 0$$

$$V_1 = \cancel{0.98 V_0 +} \; 0.02\,\theta_1$$

$$V_2 = 0.98\,V_1 + 0.02\,\theta_2$$

$$= 0.98 \times 0.02 \times \theta_1 + 0.02\,\theta_2$$

$$= 0.0196\,\theta_1 + 0.02\,\theta_2$$

- 추정의 초기단계에서 더 정확하게 보정이 가능

→ **v_t를 취하는 대신에 v_t/(1-β^t)를 취한다**. (t : 현재의 온도)

ex) t=2인 경우: 1-β^t = 1-(0.98)^2 = 0.0396

둘째 날의 온도를 추정한 값은 v_2를 0.0396으로 나눈 값과 같다.

v2/0.0396 = (0.0196θ_1 + 0.02θ_2) / 0.0396

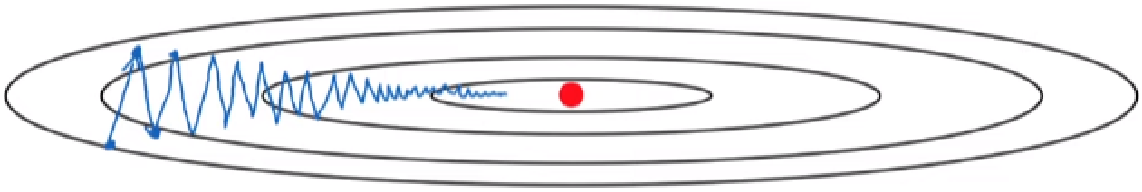
→ 따라서 이는 θ_1과 θ_2의 가중평균에 편향을 없앤 값이 된다. t가 커질수록 β^t는 0에 가까워진다. 그러므로 t가 충분히 커지면 편향보정은 그 효과가 거의 없어진다. t가 커질때 보라색 곡선과 초록색 곡선이 겹치는 이유이다. ??

6. Momentum 최적화 알고리즘

- 모멘텀 알고리즘 혹은 모멘텀이 있는 경사하강법은 **일반적인 경사 하강법보다 거의 항상 빠르게 작동**

- 기본적인 아이디어: **경사에 대한 지수가중평균을 계산**하는 것. 그 값으로 가중치를 업데이트 한다.

1. **Gradient descent example**



- 빨간점: 최소값의 위치
- 경사하강법을 시작해서 경사하강법 or 미니배치 경사 하강법의 한 반복을 취하면 그림과 같이 향한다.
- 한단계씩 나아갈 때마다 그림과 같이 나아가게 됨.
- 많은 단계를 취하면 최소값으로 나아가면서 천천히 진동
- 위아래로 일어나는 진동: 경사하강법의 속도를 느리게 하고 더 큰 학습률을 사용하는 것을 막는다.

→ 오버슈팅하게 되어 발산할 수도 있기 때문

- 수직축에서는 진동을 막기위해 학습이 더 느리게 일어나길 바라지만, 수평축에서는 더 빠른 학습을 원한다.

→ 최소값을 향해 왼쪽에서 오른쪽으로 이동하는 것을 처리하고 싶기 때문

2. **구현**

- 각 반복 t에서 현재의 미니배치에 대한 보편적인 도함수인 dw와 db를 계산
- 배치 경사하강법을 사용하는 경우, 현재의 미니배치는 전체 배치와 같다
- V_dw = βV_dw + (1-β)dw 를 계산(이동평균을 w에 대한 도함수로 계산)
- V_db = βV_db + (1-β)db 를 계산
- w를 사용해 가중치를 업데이트
- w := w-aV_dw, b := b-aV_db

Momentum:

On iteration t:

Compute dW, db on current mini-batch.

$$V_{dw} = \beta V_{dw} + (1-\beta) dW \qquad \text{"} V_\theta = \beta V_\theta + (1-\beta)\theta_t \text{"}$$

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

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

3. **모멘텀의 장점**

- 경사하강법의 단계를 부드럽게 만들어 준다.

→ 더 직선의 길을 가거나 진동을 줄일 수 있게 한다.

- 밥그릇 모양의 함수를 최소화하려고 하면 **도함수의 항들(dw, db)**은 아래로 내려갈 때 **가속을 제공**한다고 볼 수 있다.
- 모멘텀 항들(V_dw, V_db)은 속도를 나타낸다

→ 작은 공이 밥그릇의 경사를 내려갈때, 도함수는 가속을 부여하고 더 빠르게 내려가게 만든다

- β는 1보다 조금 작기 때문에 마찰을 제공해서 공이 제한없이 빨라지는 것을 막는다.

4. **구현 세부사항**

- 하이퍼 파라미터: 학습률 a, 지수가중평균을 제어하는 β
  - β의 가장 일반적인 값은 0.9

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + (1 - \beta) dW$$

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

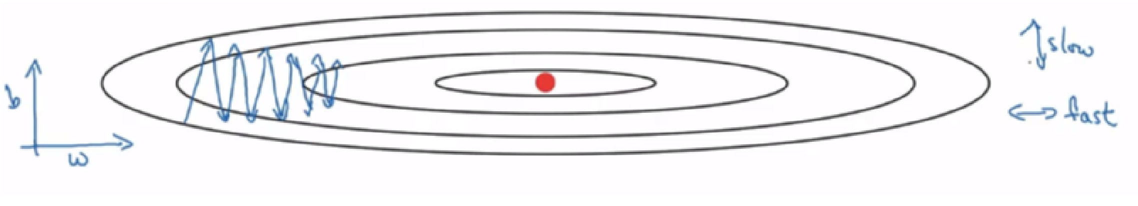
$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

Hyperparameters: α, β $\quad\quad$ $\beta = 0.9$

7. RMSProp 최적화 알고리즘

1. **RMSProp**

- 경사하강법에서 수평방향으로 진행을 시도해도 수직방향으로 큰 진동 존재
- 직관을 위해 수직축은 매개변수 b, 수평축은 매개변수 w라고 하면,
  - b방향 또는 수직방향의 학습 속도를 낮추기 위한 것이고 그리고 수평방향의 속도를 빠르게 하기 위한 것



- RMSProp 알고리즘이 하는 일
  - 반복 t에서 현재의 미니배치에 대한 보통의 도함수 dw와 db를 계산할 것
  - 지수가중평균을 유지하기 위해서 새로운 표기법인 s_dw를 사용
  - β*s_dw + (1-β)*dw^2
  - 제곱 표시는 요소별 제곱을 나타냄.

- s_db = β*s_db + (1-β)*db^2
- 매개변수 업데이트
    - w: w에서 학습률 a*dw를 s_dw의 제곱근으로 나눠준 값을 뺀 것
    - b: b-a*db만을 하는 대신에 s_db의 제곱근으로 나눠준 값을 뺀

- 알고리즘은 아래와 같습니다.
    - $S_{dW} = \beta_2 S_{dW} + (1 - \beta_2) dW^2$
    - 업데이트: $w := w - \alpha \dfrac{dW}{\sqrt{S_{dW} + \epsilon}}$
    - dW^2 은 요소별 제곱을 뜻합니다.

2. **작동 방식**

- 원하는 것: s_dw가 상대적으로 작고 s_db가 상대적으로 큰 것

= s_dw로 상대적으로 작게 나누고 s_db로 상대적으로 크게 나눈다는 의미

→ 수직방향에서의 업데이트를 줄이기 위함

- 도함수 db는 매우 크고 dw는 상대적으로 작다. (수직방향 b에서 기울기가 더 가파르기 때문)
- 더 큰 숫자로 나눠서 수직방향에서 업데이트하기 때문에 진동을 줄이는데 도움을 준다.
- 수평방향에서는 작은 숫자로 나눠서 업데이트하기 때문에 RMSProp을 사용한 업데이트는 아래와 같다.
    - 수직방향에서의 업데이트는 감소하지만 수평방향은 계속 나아가게 한다
    - **큰 학습률을 사용해 빠르게 학습하고 수직방향으로 발산하지 않는다.**
- 수직과 수평방향을 b와 w로 나타냈는데 실제로는 매개변수의 고차원 공간에 있기 때문에 진동을 줄이려는 수직차원은 w1, w2, ... w17의 매개변수 집합이고, 수평방향의 차원은 w3, w4.. 처럼 나타날 것

→ RMSProp 알고리즘은 학습 알고리즘의 속도를 올리는 방법. RMSProp와 모멘텀을 함께 사용하면 더나은 최적화 알고리즘을 얻을 수 있다.

8. Adam 최적화 알고리즘

- RMSprop과 모멘텀을 합친 알고리즘

1. **Adam optimization algorithm**

    - 알고리즘은 아래와 같습니다.
        - $V_{dW} = 0, S_{dW} = 0$ 로 초기화 시킵니다.
        - Momentum 항: $V_{dW} = \beta_1 V_{dW} + (1 - \beta_1)dW$
        - RMSProp 항: $S_{dW} = \beta_2 S_{dW} + (1 - \beta_2)dW^2$
        - Bias correction: $V_{dW}^{correct} = \dfrac{V_{dW}}{1 - \beta_1^t}, S_{dW}^{correct} = \dfrac{S_{dW}}{1 - \beta_2^t}$
        - 업데이트: $w := w - \alpha \dfrac{V_{dW}^{correct}}{\sqrt{S_{dW}^{correct}} + \epsilon}$

- 전형적인 Adam 구현에서는 편향보정을 한다. (V_dw^correctedf는 편향보정을 의미)
- 최종적으로 업데이트를 실행

→ 매우 넓은 범위의 아키텍처를 가진 서로 다른 신경망에서 잘 작동한다는 것이 증명된 일반적으로 많이 쓰이는 학습알고리즘

- 하이퍼파라미터
    - **학습률 하이퍼파라미터 a** : 매우 중요하고 보정될 필요가 있으므로 **다양한 값을 시도**해서 잘 맞는 것을 찾아야 한다.
    - **β_1** : 기본적인 값으로 **0.9**를 보통 선택한다. (dw의 이동평균, 가중평균/ 모멘텀에 관한 항)
    - **β_2** : Adam논문에서 저자가 추천하는 값이 **0.999**이다. (dw^2와 db^2의 이동가중평균을 계산한 것)
    - **ε** : **10^(-8)**의 값을 추천 (이 값을 설정하지 않더라도 전체 성능에 영향은 없음)

- β_1이 도함수의 평균을 계산하므로 이것이 첫번째 모멘트이고, β_2가 지수가중평균의 제곱을 계산하므로 두번째 모먼트

9. 학습률 감쇠

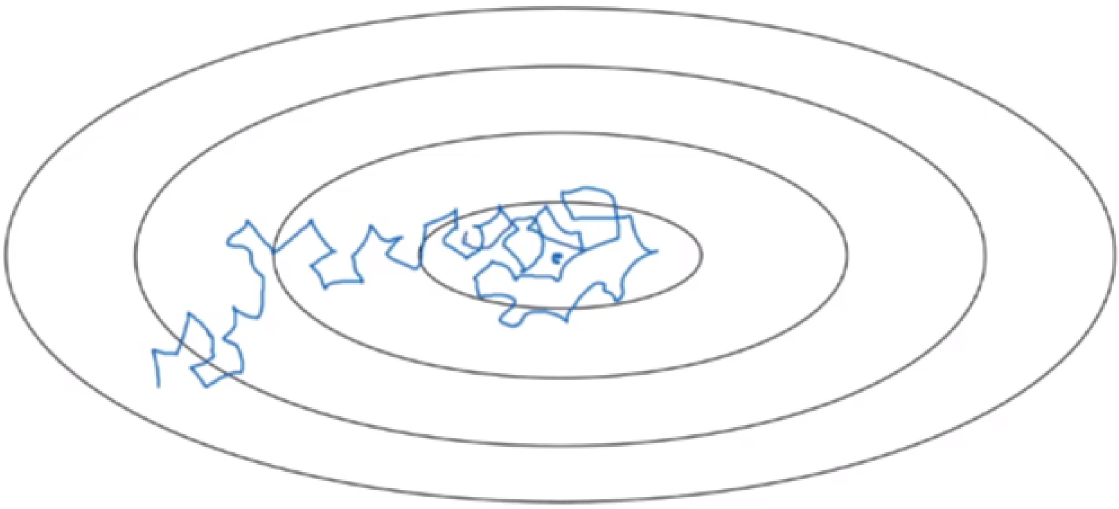
- 학습 알고리즘의 속도를 높이는 한가지 방법: **시간에 따라 학습률을 천천히 줄이는 것**

1. **학습률 감쇠가 왜 필요?**

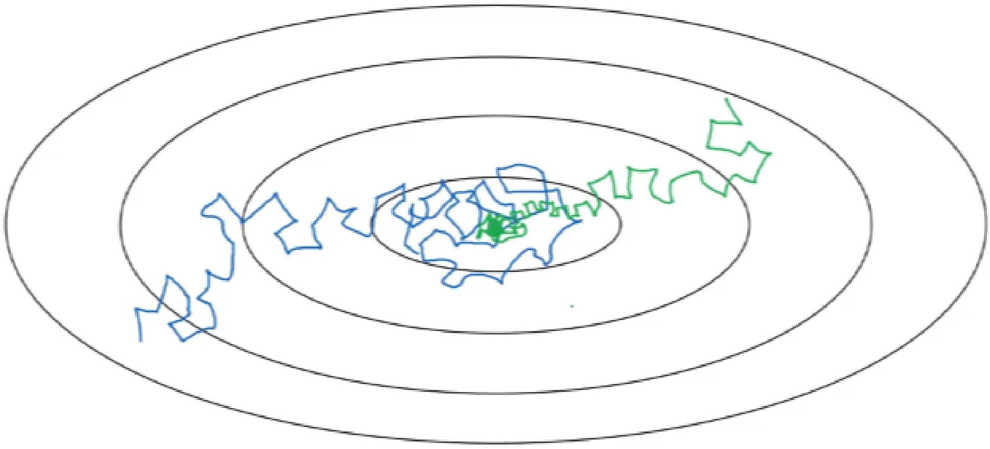
ex) 상당히 작은 미니배치(64, 128)에 대해 미니배치 경사 하강법을 구현한다고 가정

- 단계를 거치면서 **약간의 노이즈가 있지만 최소값으로 향하는 경향**을 보일 것
- 정확하게는 수렴하지 않고 주변을 돌아다니게 된다

→ 어떤 고정된 값인 a를 사용했고, 서로 다른 미니배치에 노이즈가 있기 때문



- **천천히 학습률 a를 줄이면** a가 여전히 **큰 초기 단계에서는 상대적으로 빠른 학습**이 가능
    - a가 작아지면 단계마다 진행정도가 작아지고 **최소값의 밀집된 영역에서 진동**하게 될 것
    - a를 천천히 줄이는 것의 의미: 학습 초기 단계에서는 훨씬 큰 스텝으로 진행하고 학습을 수행할 수록 학습률이 느려저 작은 스텝으로 진행하는 것



초록색 선: 학습률을 줄인 경우

2. **학습률 감쇠를 구현하는 방법**

    - 1 epoch = 전체 데이터를 1번 훑고 지나가는 횟수 입니다.
    - $\alpha = \frac{1}{1 + decay\ rate\ \times epoch\ num}\alpha_0$
    - $\alpha = 0.95^{epoch\ num}\alpha_0$ (exponential decay 라고 부릅니다.)
    - $\alpha = \frac{k}{\sqrt{epoch\ num}}\alpha_0$
    - $\alpha = \frac{k}{\sqrt{batch\ num}}\alpha_0$
    - step 별로 α 다르게 설정

- 하나의 에포크: 데이터를 지나는 하나의 패스

→ 훈련세트를 서로 다른 미니배치로 나눠서 훈련세트를 지나는 첫번째 패스를 첫번째 에포크라고 한다.

- 에포크수에 대한 함수에서 학습률은 점차적으로 감소
- 학습률 감쇠를 사용하고 싶다면 하이퍼파라미터 a_0와 감쇠율에 대해서 다양한 값을 시도하고 잘 작동하는 값을 찾으면 된다.

3. **Other learning rate decay methods**

- 지수적 감쇠
    - a가 1보다 작은 값
    - **a = 0.95^epoch_num*a_0** → 기하급수적으로 빠르게 학습률을 감소시킨다.
    - **a = k(상수)/epoch_num의 제곱근*a_0** or **k(상수)/미니배치의 개수 t의 제곱근 *a_0**

- 이산적 단계로 감소하는 학습률

: 어떤 단계에서는 어떤 학습률 값을 가지고, 그 뒤에는 학습률이 반으로 줄어들고 일정 시간이 지날 때마다 계속 반씩 줄어드는 모습

- 직접 조작하는 감쇠
    - 한 번에 하나의 모델을 훈련하는데 몇시간 혹은 며칠이 걸린다면, 훈련을 거치면서 모델을 정리해 나갈 것이다.
    - **학습률이 느려지고 있는 것처럼 느껴서 데이터의 크기를 줄이는 것**이다.
    - a의 값을 시간이나 날마다 직접 보정하는 것은 훈련이 작은 수의 모델로만 이루어진 경우에 가능