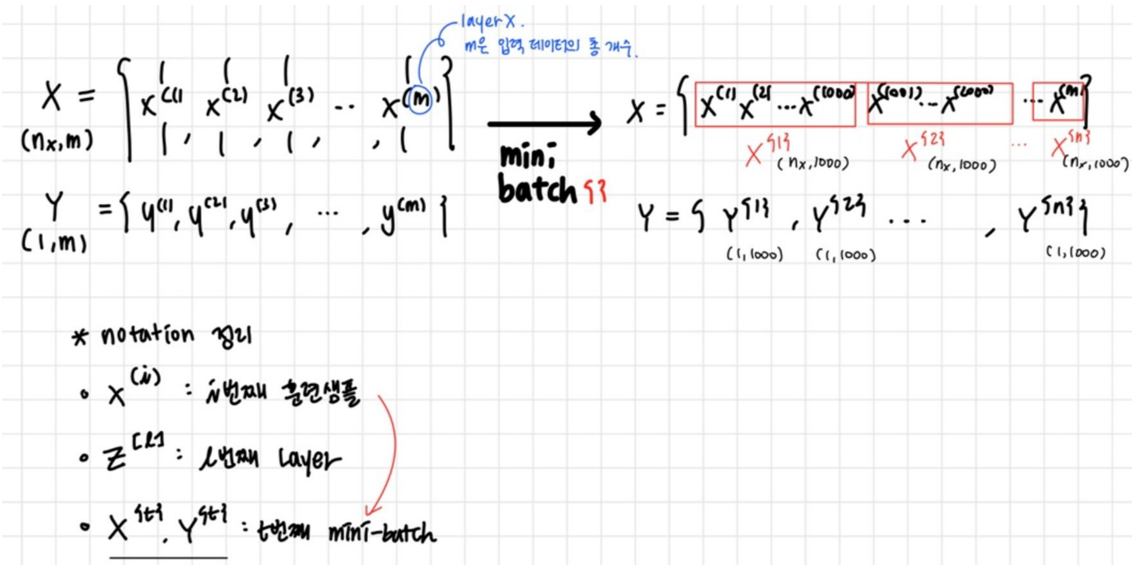
[Week 10] 1. Mini-batch gradient descent

신경망의 학습 속도를 향상시키는 최적화 알고리즘에 대해 배워보자.

Batch vs Mini-batch

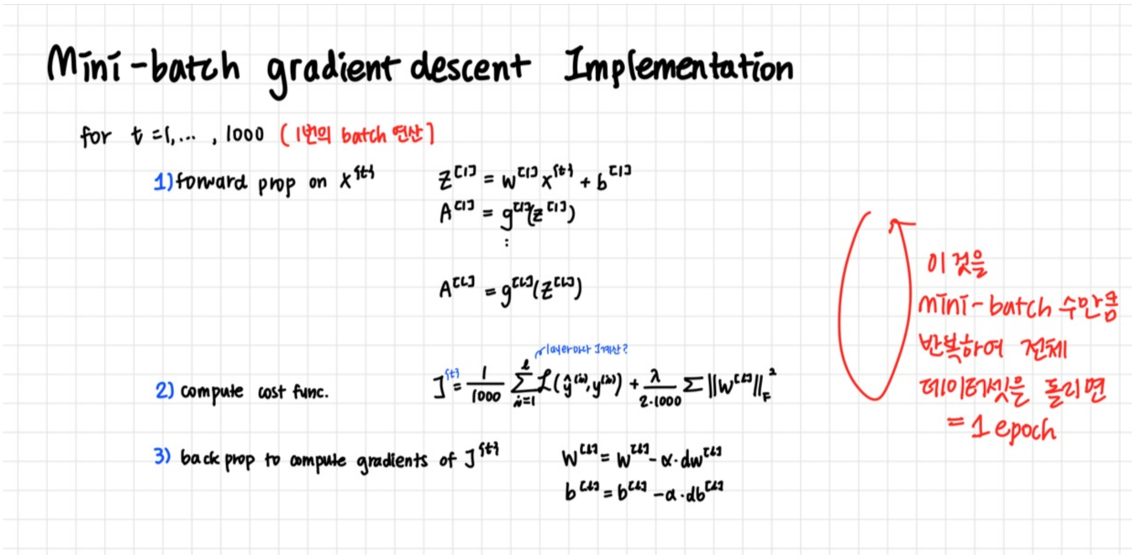
벡터화를 통해 어느정도 m개의 데이터를 훈련하는 시간을 감소시킬 수는 있지만 m이 크다면 여전히 느릴 수 있다. 만약 m이 500,000개라면? 이 m개에 대해 gradient descent를 수행하려고 하면, 경사하강법을 통해 1 step의 가중치 변경을 하려면 모든 m개의 훈련데이터를 처리해야 한다. 또 다음 가중치 변경을 위해서 모든 500000개의 데이터를 계산해야 됨.



훈련세트를 더 작은 훈련세트인 mini-batch로 만들어 진행할 수 있다.

- batch - 전체 훈련 세트를 한 번에 계산
- mini-batch - 전체 훈련 세트를 한 번에 계산하지 않고 미니배치씩 계산.

Mini-batch Implementation



Epoch

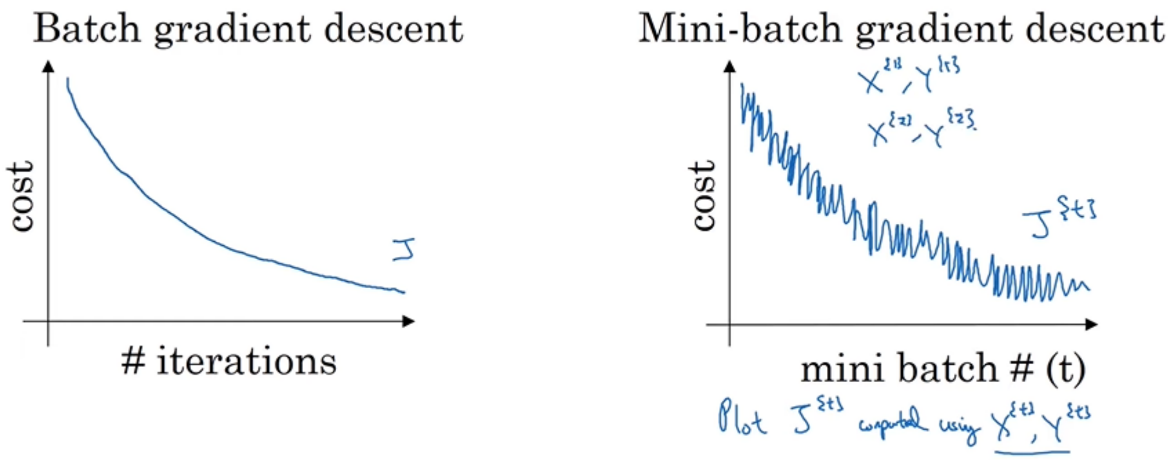
epoch - 전체 데이터셋을 한번 학습하는 주기

예를 들어, 전체 데이터셋이 1000개의 샘플로 구성되어 있고 배치 크기가 100이라면, 한 번의 Epoch는 10개의 배치를 거쳐야 합니다.

즉, 1 epoch를 위해서 batch gradient descent는 한번 수행되고, mini-batch를 사용하여 배치 크기가 100이라면 한번의 에포크 안에 gradient descent는 10번 수행되는 것이다.

??- 궁금한 점. gradient descent는 어

Gradient Descent in batch vs mini-batch



- Batch

  비용함수가 매 반복마다 계속 감소해야 함. 절대 증가하면 안됨.

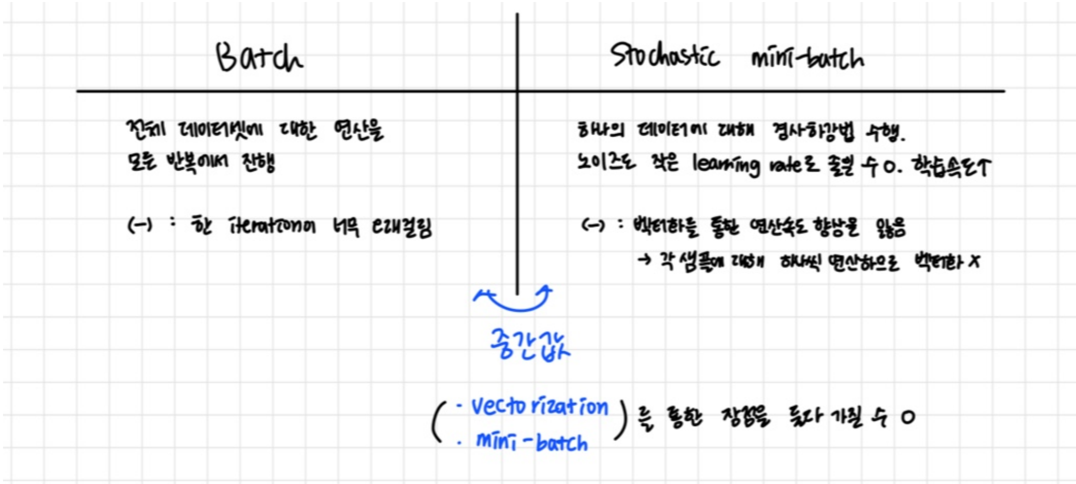
- Mini-batch

  비용함수가 매 반복마다 감소하지는 않을 수 있음. 매 반복마다 다른 미니배치에서 훈련하게 되기 때(=다른 데이터셋으로 훈련하게 됨). 어떻게 보면 batch gradient descent의 비용함수 J를 작게 여러번 붙여넣은 것이라고 볼 수 있겠다.

  전체적인 흐름은 감소하나 노이즈가 발생할 수 있음.

Batch Size

- batch size = m : Batch gradient descent
  - 상대적으로 노이즈가 적고 큰 단계를 가지기 때문에 계속 최솟값으로 나아
- batch size = 1 : Stochastic gradient descent (확률적 경사 하강법)
  - 각각의 샘플 1개가 1개의 미니배치가 됨. 샘플 수만큼 gradient descent를 반복하게 된다.
  - 하나의 훈련 샘플로 경사하강법을 반복하게 되므로, 노이즈가 많을 수 있으나 평균적으로는 최솟값으로 가게 됨. ⇒ 진동할 수 밖에 없다.



◦ batch size = 평균적으로는 저 둘 사이의 값을 사용

How to choose mini-batch size

- small training set(m<2000) → batch grad desc
- typical batch size : 64, 128, 256, 512

  (컴퓨터 메모리의 접근 방식 때문에 2의 제곱수여야 코드가 빠르게 실행됨)
- 내 컴퓨터의 cpu, gpu의 메모리에 맞는 사이즈로 할 것.

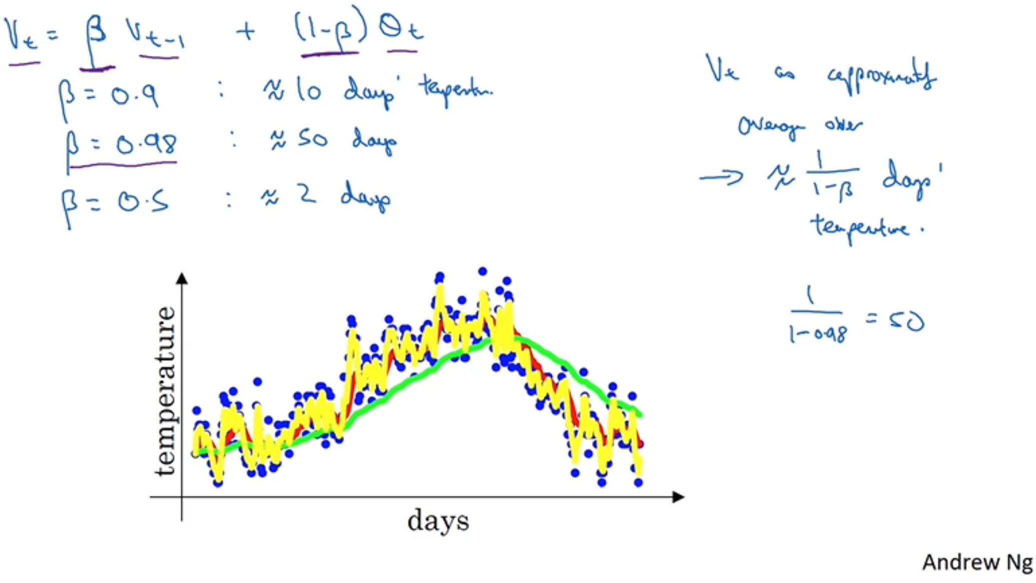
[Week 10] 2. Exponentially Weighted Averages

지금부터 gradient descent보다 빠른 몇가지 최적화 알고리즘에 대해 배워볼 것인데, 그전에 이를 이해하기 위해 ‘지수가중평균’이라는 개념을 알고 있어야 함. 몇몇 최적화 알고리즘의 중요 요소가 되기 때문!

Exponentially Weighted Averages (지수 가중 이동 평균)

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

이는 **시간의 흐름에 따라** 데이터가 어떻게 변하는지 보고 싶을 때 도움이 되는 도구이다. 또한 시간의 흐름에 따라 과거의 값의 반영 비율이 지수(1-β)적으로 감소하게 된다.⇒ 최근의 데이터에 더 많은 영향을 받는 데이터들의 평균 흐름을 계산



v_t 는 $\dfrac{1}{1-\beta}$ 기간 동안 기온의 평균을 의미합니다.

- β = 0.9일 때 10일의 기온 평균

- β = 0.5일 때 2일의 기온 평균

β 의 값이 1에 가까울수록 과거의 값들을 많이 반영하기 때문에 평균적인 값을 나타내지만, 오히려 노이즈가 생기고 딜레이가 생김

β 의 값이 0.5라면 전날과 당일날, 거의 두가지 값을 비교하는 것과 같다. 따라서 기온 변화가 있다면 그값을 빠르게 그래프에서 반영하게 된다. 단, 노이즈가 증가하고 이상치를 반영할 확률이 높아진다.

*β 값은 하이퍼 파라미터로 최적의 값을 찾아야 하는데, 보통 사용하는 값은 0.9

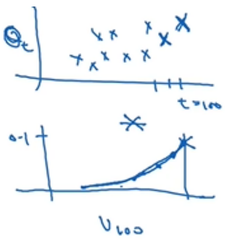
$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

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

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

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

...



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

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

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

V_100의 값을 나타내 보자.

β 값이 클 수록 과거의 값의 반영 비율이 높음을 알 수 있다. 특히 오래된 과거의 값이 꽤나 큰 비중을 차지하게 됨.

*편향 보정 : 세타의 계수를 모두 더하면 1에 매우 가깝다.

그러면 얼만큼의 기간을 기준으로 평균이 구해졌다고 볼 수 있는가?는 아래의 식으로 대략 추정이 가능하다(정확한 식이 아닌, 추정하기 위한 식)

$\beta = (1 - \varepsilon)$ 라고 정의하면

$(1-\varepsilon)^n = \frac{1}{e}$ 를 만족하는 n 이 그 기간이 되는데, 보통 $\frac{1}{\varepsilon}$ 으로 구할 수 있습니다.

왜 앞뒤로 10개의 값을 더해서 평균치를 구하고 연산하면 추정치가 더 정확할 수 있는데, 이러한 지수평균을 사용하는 이유가 뭘까?

$V_\theta = 0$

Repeat {

Get next θ_t

$V_\theta := \beta V_\theta + (1-\beta)\theta_t$

⇒ 아주 적은 메모리를 사용하기 때문. 한 줄의 코드로 연산이 된다! 실수 하나만을 변수에 저장하고 해당 값을 식에 대입하여 업데이트만 하면 되기 때문이다.

Bias Correction of Exponentially Weighted Averages

편향 보정(bias correction)으로 더 정확히 계산할 수 있음

초기에 실제 값보다 추정 값이 매우 작아지게 됨.

v_t 대신에 v_t/(1-β)^t 를 사용하여 보정할 수 있다. (t 는 현재 시점을 의미,여기서의 v_dw,v_db(=v_t)값은 기존에 가중치 업데이트에 사용되는 dw,db에 대한 미분계수를 의미)

⇒ 이것은 가중평균의 편향을 없앤 값이 되게 한다.

그러나 t가 충분히 커지면 시간이 지남에 (1–β^t) 는 1에 가까워져서 원하는 값과 일치하게 되기 때문에, 초기값의 보정을 위해서 사용함.

[Week 10] 3. Optimization Algorithm - Momentum

Momentum이 있는 경사하강법은 우리가 이전에 배운 일반적인 경사하강법보다 거의 항상 더 빠르게 작동함.

Momentum

이름이 왜 momentum(관성)이냐??

⇒ Gradient Descent를 통해 이동하는 과정에 일종의 '관성'을 주는 것이다. 현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식

⇒ 일반적인 gradient descent와의 차이점 :

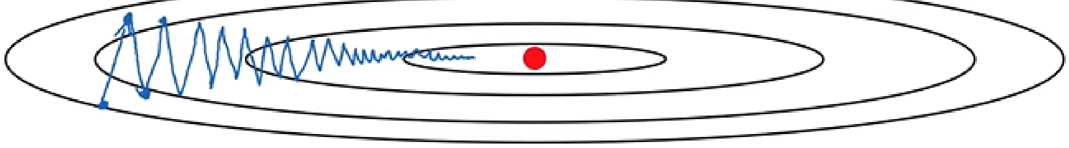
- 일반적 gd: 손실 함수의 기울기(gradient)를 사용하여 매개변수를 업데이트한다. 이 때 전체 데이터 세트의 평균 기울기를 사용

$$\theta = \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla J(\theta; x^{(i)}, y^{(i)})$$

- Momentum : 경사에 대한 지수가중평균을 계산함

$$V_{dW} = \beta_1 V_{dW} + (1 - \beta_1) dW$$
$$w := w - \alpha V_{dW}$$

Why faster in oscillation?



경사하강법을 취하면 한 단계씩 갈 때마다 진동이 발생할 수 있음 → 학습 속도를 느리게 하고, learning rate의 크기에 한계가 생김. 왜냐하면 learning rate가 커지면 진동이 커지고, 이는 발산으로 이어질 수 있기 때문.

빠른 학습을 위해서는 수직축은 작은 변화(slower learning)를 원하고, 수평축은 큰 변화(faster learning)이 있어야 함.

어떤 원리에서 이게 가능한 걸까?

진동하는 그림을 보면, 이 경사(기울기)들의 평균을 구하면 수직축의 기울기의 평균이 0에 가깝게 만들어진다. 그리고 수평축을 보면, 모든 도함수 값이(수직축으로의 변화율) 같은 방향(오른쪽)을 가리키고 있기 때문에 수평방향의 평균은 일반적인 경사하강법에서 사용하는 미분계수보다 오히려 더 큰 값을 가진다.

⇒**자주 이동하는 방향에 관성이 걸리게 되고**, 진동을 하더라도 중앙으로 가는 방향에 힘을 얻기 때문에 SGD에 비해 상대적으로 빠르게 이동할 수 있다.

+또한 Momentum 방식을 이용할 경우 local minima를 빠져나오는 효과가 있을 것이라고도 기대할 수 있다. 기존의 SGD를 이용할 경우 좌측의 local minima에 빠지면 gradient가 0이 되어 이동할 수가 없지만, momentum 방식의 경우 기존에 이동했던 방향에 관성이 있어 이 local minima를 빠져나오고 더 좋은 minima로 이동할 것을 기대할 수 있게 된다.

Implementation

On iteration t:

Compute dW, db on the current mini-batch

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

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

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

모멘텀 상수인 베타의 값이 클수록 이전의 속도를 더 따르게 된다.

즉, 모멘텀을 사용한다는 것은 학습 속도를 1/1-β 만큼으로 보정하는 것이라고 해석할 수 있다. β를 0.9로 한다는 것은, 기존 대비 약 10배 정도의 속도로 움직이도록 한다고 볼 수 있다.

[Week 10] 4. Optimization Algorithm - RMS prop

RMS prop 역시 경사하강법을 더 빠르게 함.

RMS prop

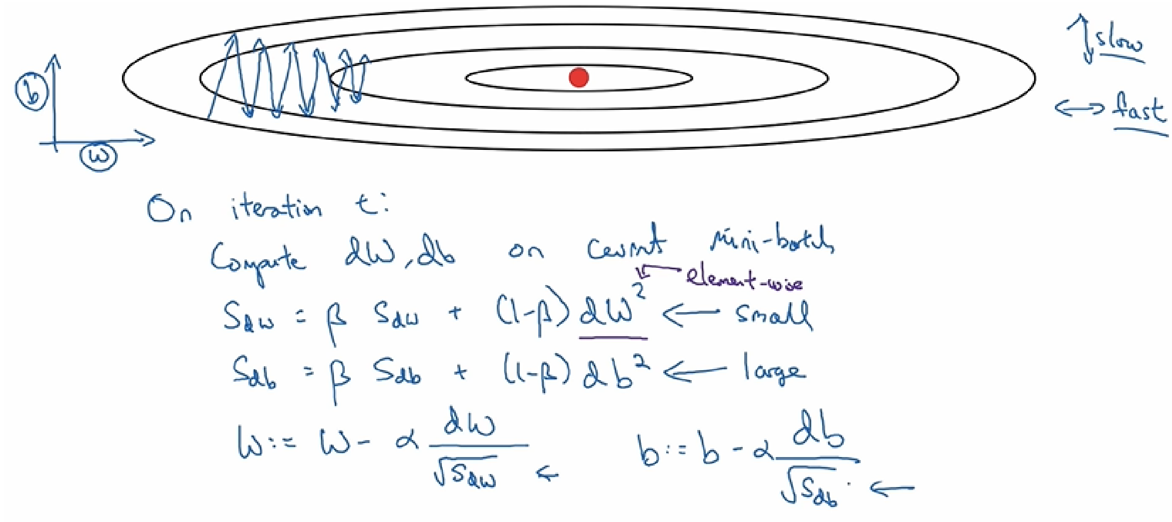
root mean square prop의 약자

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

$$\text{업데이트: } w := w - \alpha \frac{dW}{\sqrt{S_{dW} + \epsilon}}$$

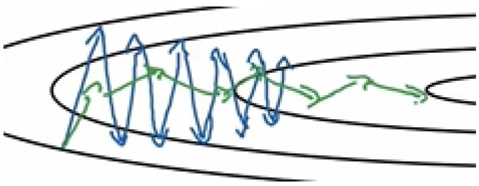
- dW^2은 요소별 제곱(elementwise)
- momentum과 일반적인 gd와의 차이 :
  - 이전의 momentum은 그냥 도함수를 가중 평균하는 것이었다면,도함수의 제곱을 지수가중평균 하는 것.
  - 이전에는 α* dW를 빼주는 방식으로 가중치를 업데이트하였다면, 이제는 거기에 s_dw의 제곱근으로 나눠준 후 뺀다.

Intuition of RMSprop



*단, 실제로는 w와 b로 나뉘는게 아니라 수직축이 w1,w2,w7,...이고 수평축이 w3,w4,w5...등으로 구성된 매우 고차원의 벡터이다. 직관적으로 이해하기 위해 단일 벡터처럼 나타낸것.

수직 방향에서의 도함수가(도함수의 제곱) 수평방향보다 크다. 수직방향에서 현재 경사가 매우 크기 때문에, 더 큰 숫자로 나뉘어지므로 진동을 줄이는데 도움을 준다. 따라서 RMSprop을 사용하여 업데이트하면 초록색처럼 변할 것이다.



[Week 10] 5. Optimization Algorithm - Adam

Adam optimization algorithm

- Adam = Momentum + RMSprop
- adaptive moment estimation의 약자
- adam에서는 편향 보정을 함 (correct가 그 의미)

$V_{dW} = 0, S_{dW} = 0$ 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: $w := w - \alpha \dfrac{V_{dW}^{correct}}{\sqrt{S_{dW}^{correct}} + \epsilon}$

Parameters

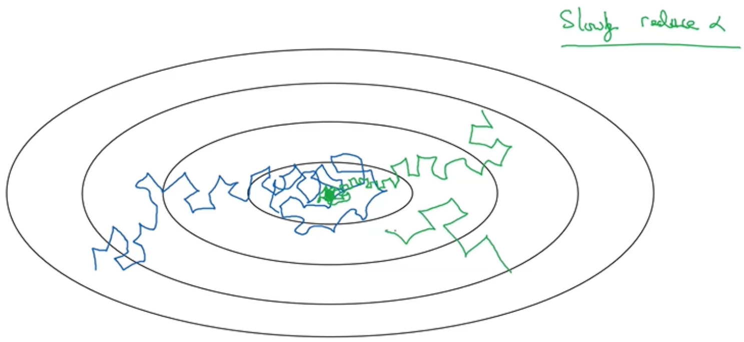
- *α* : needs to be tuned
- *β_1*은 주로 0.9 (dW의 이동 평균)
- *β_2* 는 주로 0.999 (dW^2의 이동 평균)
- *ε* : 10^(-8) 추천.

[Week 10] 6. Learning rate Decay

학습 알고리즘의 속도를 높이는 한가지 방법은 **시간에 따라 learning rate를 천천히 줄이는 것**이다.

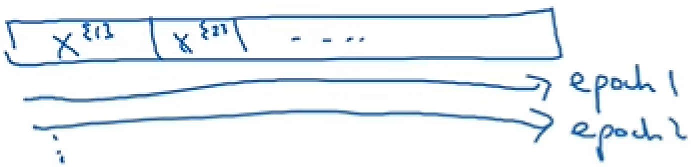
Learning rate decay

작은 배치 크기(ex.64,128)로 학습한다고 해보자, 그러면 노이즈가 있으나 점차 수렴하되, 완전한 최소값에는 도달하지 못한다. 그러나 천천히 a를 줄이면, 점차 최솟값 주변의 지역에서 진동하게 될 것이다.



Implementation

- 1 epoch : 전체 데이터를 훑고 지나감



- decay rate를 적절히 선택해야 함.

$$\alpha = \frac{1}{1 + decay\ rate\ \times epoch\ num}\alpha_0$$

- 여러 deacy methods

$$\alpha = 0.95^{epoch\ num}\alpha_0$$ (exponential decay 라고 부릅니다.)

$$\alpha = \frac{k}{\sqrt{epoch\ num}}\alpha_0$$

$$\alpha = \frac{k}{\sqrt{batch\ num}}\alpha_0$$