Optimization Algorithms

Mini batch Gradient Descent

- $m = 5{,}000{,}000$개의 data가 있다고 가정 하면 이것을 한 번에 처리하기에는 엄청난 처리시간이 필요.
- 5,000개씩 mini-batch 적용하면 아래와 같이 split 가능.

$X^{\{1\}} = [X^{[1]}, X^{[2]}, \dots X^{[1000]}]$

$X^{\{2\}} = [X^{[1001]} \dots X^{[2000]}]$

5000

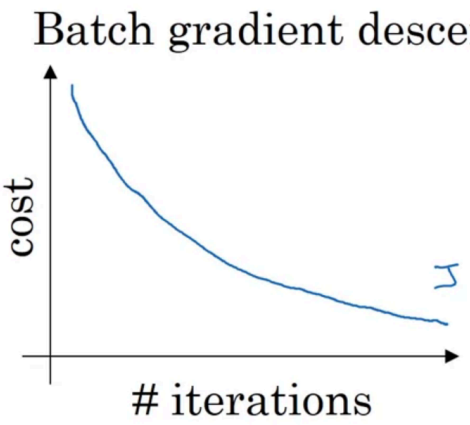
$X^{\{5000\}} = [X^{[4999,999,001]} \dots]$

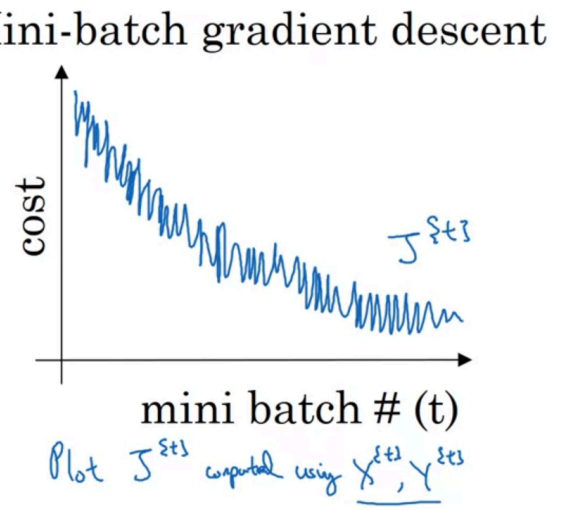
- Y도 이와 같이 1000개씩 잘라준다.
- mini batch 의 t : $X^{\{t\}}, Y^{\{t\}}$

```
for t = 1, ..., 5000
    AL, caches = forward_prop(X{t}, Y{t})
    cost = compute_cost(AL, Y{t})
    grads = backward_prop(AL, caches)
    update_parameters(grads).
```

- mini-batch를 사용할 것이. 속도 측면에서 훨씬 빠르게 작동한다.

Understanding mini batch gradient descent





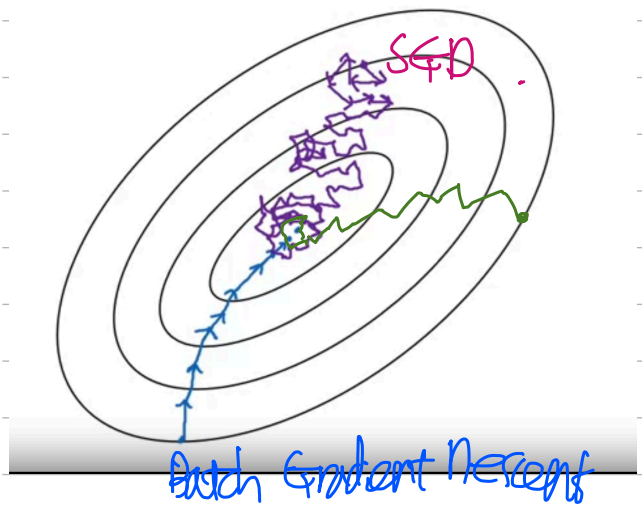
- Gradient Descent에서는 학습할수록 J가 감소.
- Mini-batch gradient descent에서는 각 다른 dataset을 사용하기 때문에 J가 감소하는 경향을 보나 noise 발생

- Mini-batch size

mini-batch size = m ⇒ Batch Gradient Decent

mini-batch size = 1 ⇒ stochastic Gradient Descent.

mini-batch size = between 1 and m ⇒ Mini-batch Gra..



Batch Gradient Descent.

→ Too long per iteration

SGD

→ too noisy regarding cost minimization. (can be reduced by using small α)

- Won't ever converge (reach the minimum cost)
- lose speedup from vectorization

Mini-batch Gradient Descent

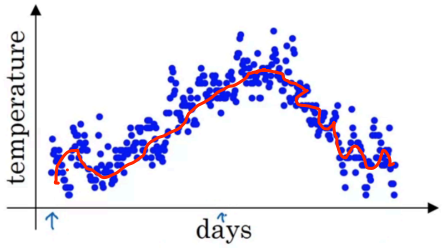
→ • Fastest learning ⇒ vectorization advantage. make process without waiting. to process the entire training-set

* training set이 작을 때는 Batch Gradient Descent, 아니면 mini batch 사용 (2^n, 64, 128, ...)

Exponentially weighted averages

Temperature in London

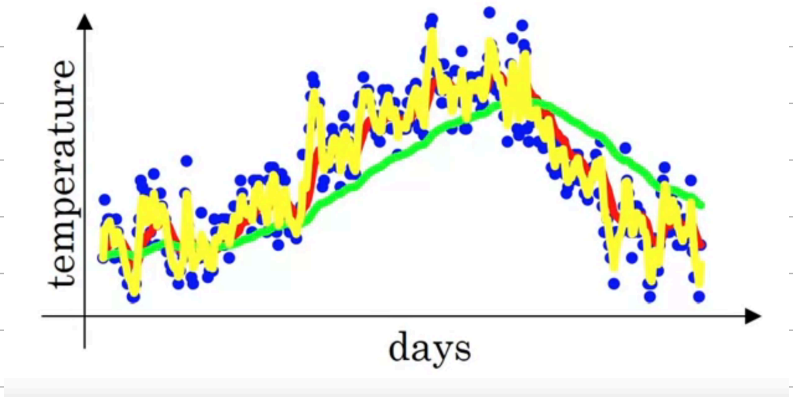
$\theta_1 = 40°F$ 4°C
$\theta_2 = 49°F$ 9°C
$\theta_3 = 45°F$ ⋮
⋮
$\theta_{180} = 60°F$ 15°C
$\theta_{181} = 56°F$ ⋮
⋮



$V_0 = 0$
$V_1 = 0.9V_0 + 0.1\theta_1$
$V_2 = 0.9V_1 + 0.1\theta_2$
⋮
$V_t = 0.9V_{t-1} + 0.1\theta_t$

temperature의 값.
noise가 있다.
noise 없애기 위해 average 사용.

식에 곱해진 0.9는 smoothness의 정도 결정.



V_t can be considered to be approximate average over $\frac{1}{1-\beta}$ days temperature

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

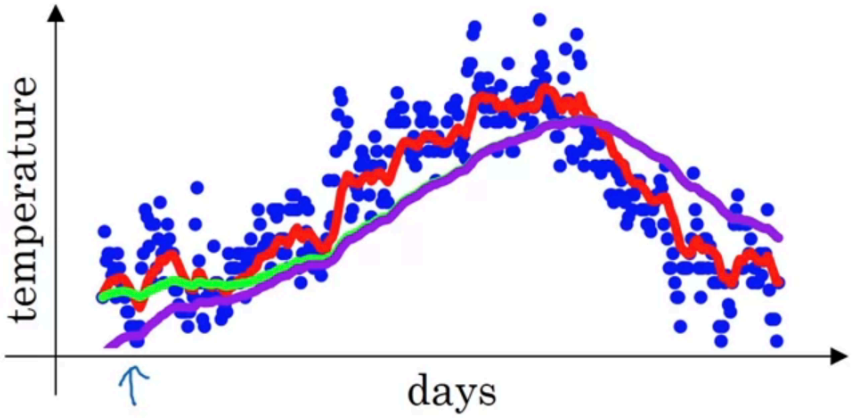
$\beta = 0.9 \rightarrow$ 10 days : 현재 sample을 포함, 과거 9개의 sample을 고려한다는 뜻.
$\beta = 0.98 \rightarrow$ 50 days
$\beta = 0.5 \rightarrow$ 2 days

β를 너무 키워 너무 smooth해지면 기준의 peak가 뒤로 밀리는 delay 발생 가능.
이런 delay를 감내할 수 있는 범위에서 값을 정한다.

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

⇒ β 값이 exponential하게 감소하기에 exponentially weighted 라 한다.

Bias correction in exponentially weighted averages



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

$V_0 = 0$

$V_1 = 0.98 V_0 + 0.02\theta_1 = 0.02\theta_1$

$V_2 = 0.98 V_1 + 0.02\theta_2 = 0.98 \times 0.02\theta_1 + 0.02\theta_2$

$= 0.0196\theta_1 + 0.02\theta_2$

$V_t \Rightarrow \frac{V_t}{1-\beta^t}$ $\quad t=2 \Rightarrow 1-\beta^2 = 1-0.98^2 = 0.0396$

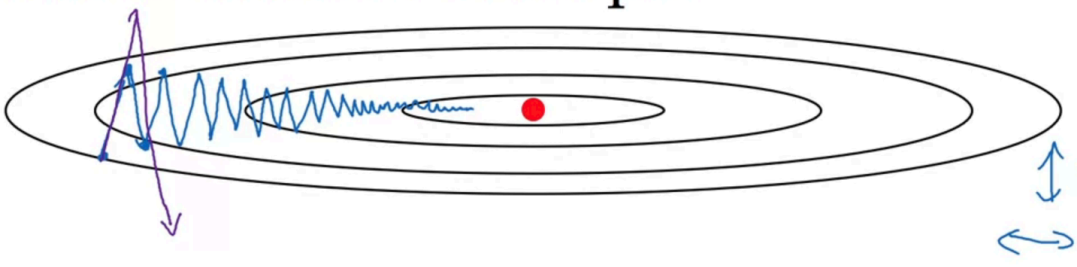
$\frac{V_2}{0.0396} = 0.49\theta_1 + 0.51\theta_2$

- V_1을 구하기 위한 V_0이 없기 때문에 임의로 정의해야함.
하지만 초기값 V_0은 0 이기 때문에 V_1을 구할때 $V_0=0$을 사용해버리면
V_1의 값이 0으로 계산되는 문제점 발생.

- Bias Correction $V_t \to \frac{V_t}{1-\beta^t}$ 는 starting point에서
초록색선과 보라색선을 일치시키기위한 방법.
t가 충분 커지면, β^t가 0에 수렴하게 되면서 분모가 1에
가까워지고 자연스럽게 bias를 고려하지 않게 된다.

Gradient descent with momentum

Gradient descent example



Momentum : On iteration t :

Compute dW and db on current mini-batch.

$V_\theta = \beta V_0 + (1-\beta)\theta_t$.

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

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

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

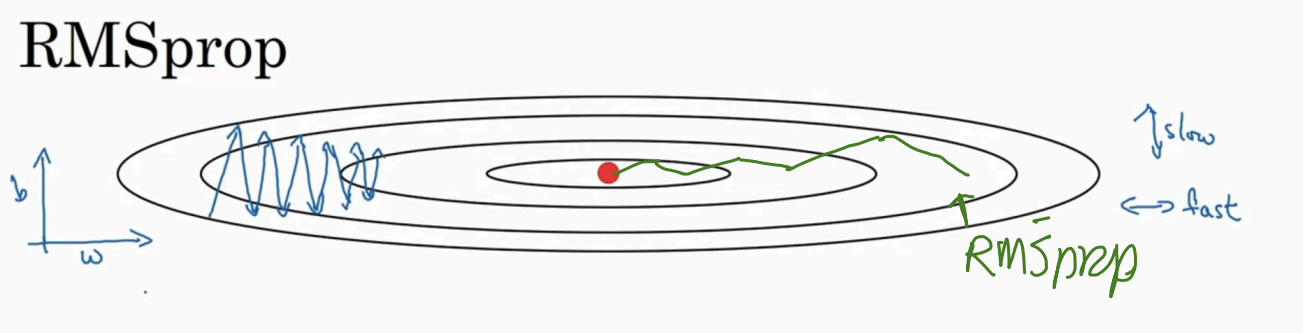
Hyperparameters : α, β $\beta = 0.9$

average over last $\simeq 10$ gradients.

mini-batch를 통해 gradient descent를 사용하면, 발산 정도가 도달하는데 굉장히 noisy하게 움직일 것.

그래서 noise가 덜하도록, $(1-\beta)$로 smooth를 해준다.

RMSprop - Root Mean Square prop.



- RMS prop은 단순히 smooth만 하는게 아니라 minimum으로 가는 속도를 조절한다. (세로방향의 gradient는 크게 흔들면 좋겠고, 가로방향의 gradient는 같이면 좋겠다고 생각)

- On Iteration t:

  Compute dW and db on current minibatch.

  $S_{dW} = \beta S_{dW} + (1-\beta) dW^2$ want small S_{dW}

  $S_{db} = \beta S_{db} + (1-\beta) db^2$ want large S_{db}.

- RMS prop은 W를 키우고, b를 줄이고 싶어함.

  S_{dW}가 1보다 작으면 W의 값이 커지고, S_{db}가 1보다 크면 b의 값이 작아지므로. 작은 S_{dW}와 S_{db}가 필요.

$$W := W - \alpha \frac{dW}{\sqrt{S_{dW}}} \qquad b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

Adam optimization

- Adam은 $V_{dW}=0$ 으로, $S_{dW}=0$, $V_{db}=0$, $S_{db}=0$

On iteration t:

Compute dW, db on current mini-batch.

"$V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$" ⇒ momentum with β_1

"$S_{dW} = \beta_2 V_{dW} + (1-\beta_2) dW^2$, $S_{db} = \beta_2 V_{db} + (1-\beta_2) db^2$" ⇒ RMSprop with β_2

$$V_{dW}^{corrected} = \frac{V_{dW}}{(1-\beta_1^t)} \qquad V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$$

$$S_{dW}^{corrected} = \frac{S_{dW}}{(1-\beta_2^t)} \qquad S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$$

$$W := W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}} + \varepsilon}, \quad b := b - \alpha \frac{V_{db}^{corr}}{\sqrt{S_{db}^{corr}} + \varepsilon}$$

- Adam에서 V는 과거 gradient 사용, S는 dW^2 사용.
W와 b를 구할 때는 gradient가 반복 값을 유지하도록 분모로 나눠준다.

Hyper parameters α: needs to be tune.

recommended by default:
- $\beta_1 = 0.9$ (weighted average for dW)
- $\beta_2 = 0.999$ (weighted average for dW^2)
- $\varepsilon = 10^{-8}$

Learning rate decay

1 epoch = 1 pass through data.

$$\alpha = \frac{1}{1 + \text{decay\_rate} \times \text{epoch\_num}} \alpha_0$$

$\alpha_0 = 0.2$.

decay_rate = 1

Epoch	α
1	0.1
2	0.067
3	0.5
⋮	⋮

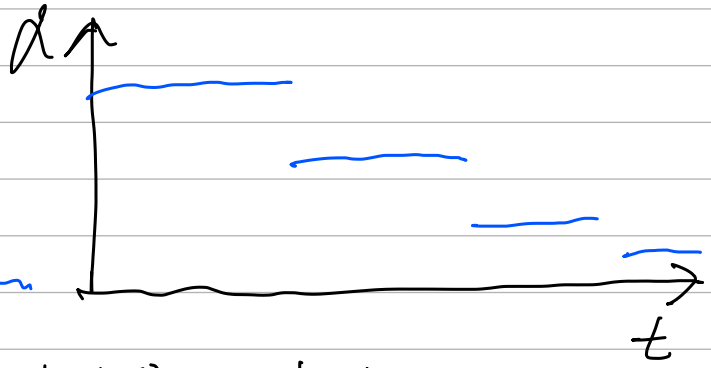
α_0와 decay_rate는 hyper parameter.

iterational 횟수 : learning rate α는 감소.

$\alpha = 0.95^{\text{epoch\_num}} \cdot \alpha_0$: exponentially decaying

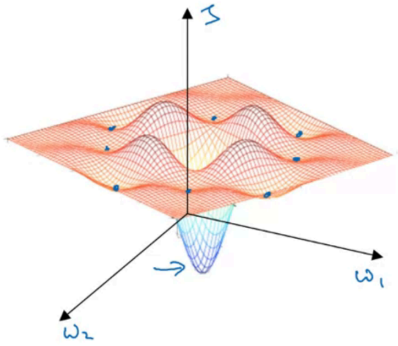
$\alpha = \frac{k}{\sqrt{\text{epoch\_num}}} \cdot \alpha_0$ or $\alpha = \frac{k}{\sqrt{t}} \cdot \alpha_0$

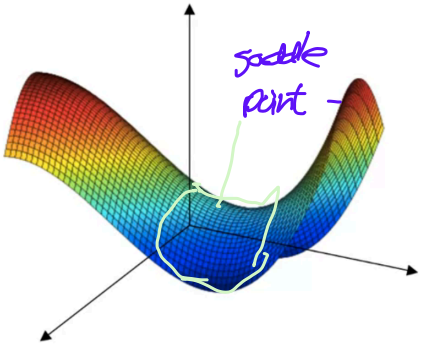
(Formula based.)



α값을 epoch에 따라 임의로 조절하는 방법.

The problem of local optima





과거에는 왼쪽 그림처럼 solution space가 엄청 복잡할 것이라 생각함.
하지만 실제는 오른쪽 그림처럼 말안장처럼 평평하게 생긴 solution space 존재.

- Plateaus can make learning slow
  - Plateau is a regial where the derivative is close to zero for a long time
  - This is where algorithms like momentum, RMSprop or Adam can help.

$0.5V_0 + (0.5)\theta_1$

$V_1 = 15$ $\quad 0.5V_1 + (1-\beta)\theta_2$

$7.5 + 0.5 = 15$

$V_2 = 15$ //