실험적이고 반복적인 과정 → 빠르게 훈련!
→ 좋은 최적화 알고리즘!

Optimization Algorithms

deeplearning.ai

Mini-batch gradient descent

Batch vs. mini-batch gradient descent

X, Y $\qquad$ $X^{\{t\}}, Y^{\{t\}}$

Vectorization allows you to efficiently compute on m examples.

$X = [x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)} \mid x^{(1001)} \dots x^{(2000)} \mid \dots \mid \dots x^{(m)}]$ $\quad (n_x, m)$

$X^{\{1\}}\ (n_x, 1000)$ 새로운 표현, $\quad X^{\{2\}}\ (n_x, 1000)\ \dots\ X^{\{5,000\}}\ (n_x, 1000)$

$Y = [y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)} \mid y^{(1001)} \dots y^{(2000)} \mid \dots \mid \dots y^{(m)}]$ $\quad (1, m)$

$Y^{\{1\}}\ (1, 1000)$, $\quad Y^{\{2\}}\ (1, 1000)\ \dots\ Y^{\{5,000\}}\ (1, 1000)$

What if $m = 5{,}000{,}000$? 벡터화 여전히 느림 (경사 하강법 각 단계…)

5,000 mini-batches of 1,000 each

Mini-batch t: $X^{\{t\}}, Y^{\{t\}}$ 작은 훈련세트

$x^{(i)}$ i 번째 훈련샘플

$z^{[l]}$ " 신경망

$X^{\{t\}}, Y^{\{t\}}$

Andrew Ng

Mini-batch gradient descent

한 번에 전체 X (배치 경사하강법)

repeat { for $t = 1, \dots, 5000$ {

① 정방향 전파. Forward prop on $X^{\{t\}}$. 추가!

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

Vectorized implementation (1000 examples) 벡터화.

② Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_l \|W^{[l]}\|_F^2$ (from $X^{\{t\}}, Y^{\{t\}}$; 정규화항; 1000 = size)

③ Backprop to compute gradients wrt $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$) 에 대해서 경사 계산

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

} }

한 번 계산! 1 step of gradient descent using $X^{\{t\}}, Y^{\{t\}}$ (as if $m = 1000$)

X, Y

"1 epoch" single pass through training set.

Andrew Ng

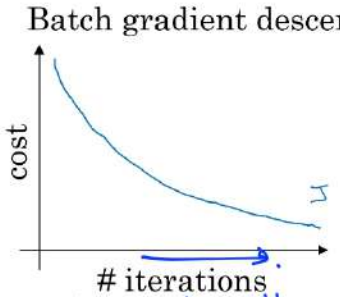
2.

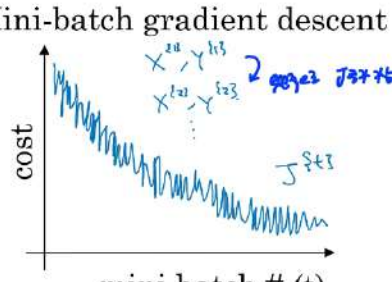
deeplearning.ai

Optimization Algorithms

Understanding mini-batch gradient descent

Training with mini batch gradient descent





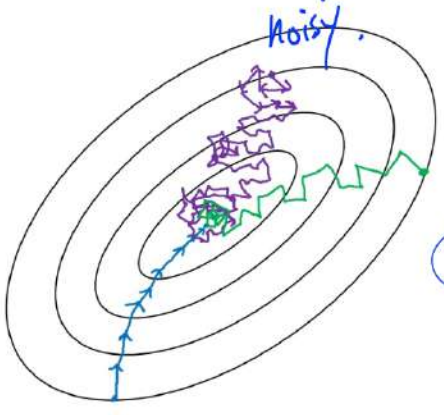
Andrew Ng

Choosing your mini-batch size

→ If mini-batch size = m : Batch gradient descent. $(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own (mini-batch)

In practice: Somewhere in-between 1 and m



Stochastic gradient descent

Too slow

In-between (mini-batch size not too big/small)

Fastest learning.

- Vectorization.
- Make progress without processing entire training set.

Batch gradient descent (mini-batch size = m)

Too long per iteration

Andrew Ng

Choosing your mini-batch size

If small training set: Use batch gradient descent.

$(m \leq 2000)$

Typical mini-batch sizes:

64, 128, 256, 512, 1024

2^6, 2^7, 2^8, 2^9, 2^{10}

Andrew Ng

Optimization Algorithms

경사하강법 보다 빠른 최적화 알고리즘

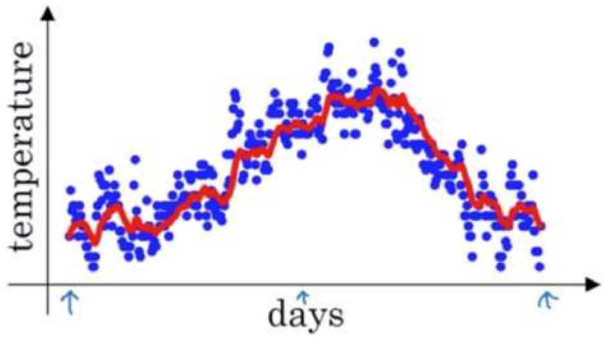
deeplearning.ai

Exponentially weighted averages

지수가중(이동)평균

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
$\theta_{181} = 56°F$ ⋮
⋮



$V_0 = 0$
$V_1 = 0.9 V_0 + 0.1 \theta_1$
$V_2 = 0.9 V_1 + 0.1 \theta_2$
$V_3 = 0.9 V_2 + 0.1 \theta_3$
⋮
$V_t = 0.9 V_{t-1} + 0.1 \theta_t$

Andrew Ng

Exponentially weighted averages (moving)

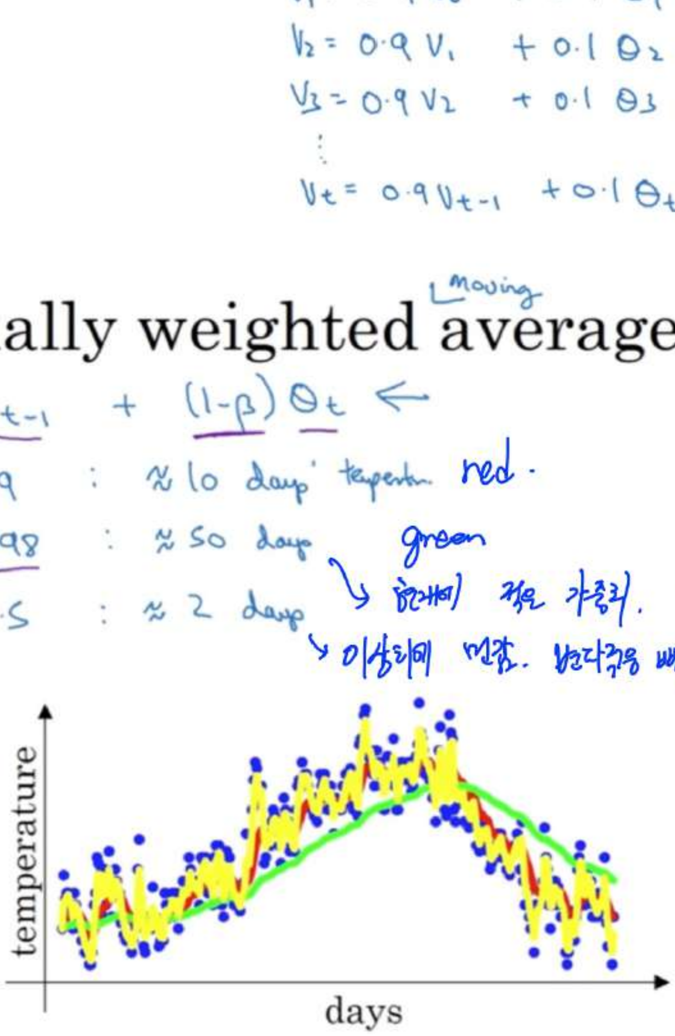
$V_t = \beta V_{t-1} + (1-\beta)\theta_t$ ←

$\beta = 0.9$: ≈ 10 days' temperature. red.
$\beta = 0.98$: ≈ 50 days green
$\beta = 0.5$: ≈ 2 days

부드러운 곡선 지연이 있다.
한개에 적은 가중치. → ≈ $\frac{1}{1-\beta}$ days' temperature.
이상치에 민감. 변화가 더 빠름.

V_t as approximately average over

$\frac{1}{1-0.98} = 50$



Andrew Ng

Optimization Algorithms

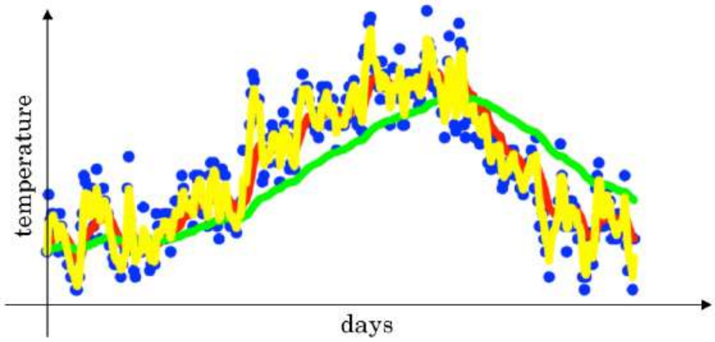
Understanding exponentially weighted averages

deeplearning.ai

Exponentially weighted averages

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

$\beta = 0.9$ 0.98 0.5



Andrew Ng

Exponentially weighted averages

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

$$v_{100} = 0.9 v_{99} + 0.1\theta_{100}$$
$$v_{99} = 0.9 v_{98} + 0.1\theta_{99}$$
$$v_{98} = 0.9 v_{97} + 0.1\theta_{98}$$

...

$$V_{100} = 0.1\,\theta_{100} + 0.9\,(0.1\,\theta_{99} + 0.9\,(0.1\,\theta_{98} + 0.9\,v_{97}))$$

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

$$0.9^{10} \approx 0.35 \approx \frac{1}{e}$$

$$\approx \frac{1}{1-\beta} \qquad \varepsilon = 1-\beta$$

$$(1-\varepsilon)^{1/\varepsilon} = \frac{1}{e}$$

$\varepsilon = 0.02 \rightarrow 0.98^{50} \approx \frac{1}{e}$

Andrew Ng

Implementing exponentially weighted averages

$$v_0 = 0$$
$$v_1 = \beta v_0 + (1-\beta)\,\theta_1$$
$$v_2 = \beta v_1 + (1-\beta)\,\theta_2$$
$$v_3 = \beta v_2 + (1-\beta)\,\theta_3$$

...

$V_\theta := 0$

$V_\theta := \beta v + (1-\beta)\,\theta_1$

$V_\theta := \beta v + (1-\beta)\,\theta_2$

⋮

$V_\theta = 0$

Repeat {

Get next θ_t

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

}

Andrew Ng

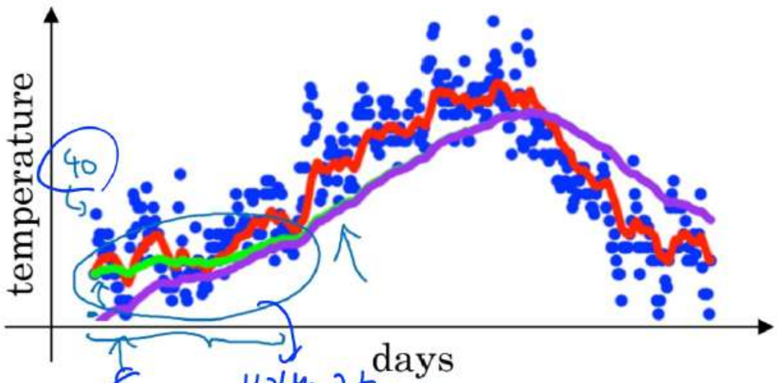
deeplearning.ai

Optimization Algorithms

Bias correction in exponentially weighted average

편향 보정.
→ 평균 더 정확히 계산

Bias correction



$\beta = 0.98$

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

$V_0 = 0$

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

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

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

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

θ1 θ2보다 아주 작은 값

$\frac{V_t}{1-\beta^t}$

initial 단계에 좋는다.

t 커지면 거의 0 .. 효과 보정 줄어짐.

$t=2: \quad 1-\beta^t = 1-(0.98)^2 = 0.0396$

$\frac{V_2}{0.0396} = \frac{0.0196\,\theta_1 + 0.02\,\theta_2}{0.0396}$

편향 없앤 값

Andrew Ng

Optimization Algorithms

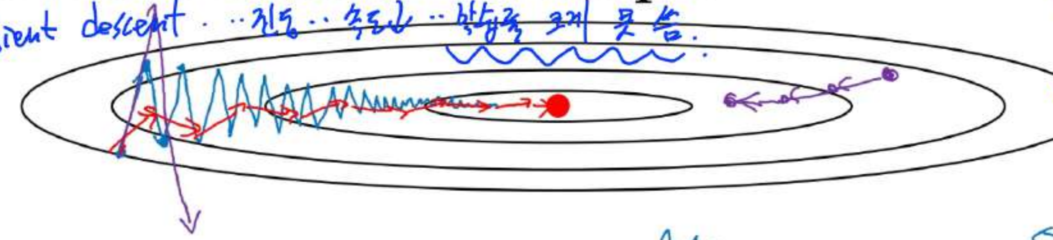
deeplearning.ai

Gradient descent with momentum

모멘텀이 있는 경사 하강법
→ 경사에 대한 지수가중평균 계산 .. 가중치 업데이트
거의 항상 더 잘 작동한다.

Gradient descent example

b: gradient descent .. 진동.. 속도↓.. 학습률 크게 못 씀.



수직·수평축의 다른 요구사항
↕ slower learning
↔ faster learning.

Momentum:
On iteration t:
Compute dW, db on current mini-batch.
$V_{dW} = \beta V_{dW} + (1-\beta) dW$ " $V_\theta = \beta V_\theta + (1-\beta)\theta_t$ "
$V_{db} = \beta V_{db} + (1-\beta) db$
friction → ↑ velocity   ↖ acceleration 가속제공
$W := W - \alpha V_{dW}$, $b := b - \alpha V_{db}$ 빠르게 진동 줄이기!

Andrew Ng

Implementation details

$V_{dW} = 0$, $V_{db} = 0$ β: 지수가중평균의 h.p.

On iteration t:

Compute dW, db on the current mini-batch

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

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

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

(자주 삭제되어 있다) $V_{dW} = \beta V_{dW} + dW$ ← 1/1-β에 대한 계수로 스케일링된다. ??

~~$\frac{V_{dW}}{1-\beta^t}$~~ 10번 후 충분히 편향 보정됨.

Hyperparameters: α, β $\beta = 0.9$ 잘 나오는 값

average over last ≈10 gradients

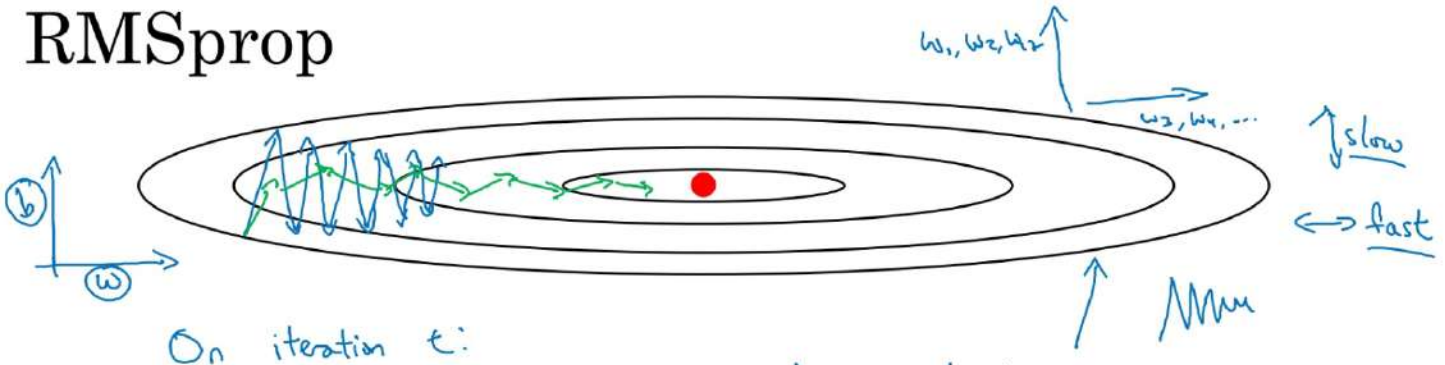
Andrew Ng

deeplearning.ai

Optimization Algorithms

RMSprop

RMSprop



On iteration t:

Compute dW, db on current mini-batch

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) \underbrace{dW^2}_{\text{element-wise}}$ ← small

→ $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$ ← large

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

$\varepsilon = 10^{-8}$

Andrew Ng

장점: 파평값이 큰 곳: 천천히

작은 곳: 더 빠르게 수렴

deeplearning.ai

Optimization Algorithms

(모멘텀+경사하강법)보다 잘 작동하는 최적화 알고리즘 찾기 어렵다

→ RMSprop 와 Adam 잘 연구됨 .. RMSprop + 모멘텀

Adam optimization algorithm

→ Adaptive moment estimation

Adam optimization algorithm

$V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0$

On iteration t:

Compute dw, db using current mini-batch (bet 개념)

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ ← "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db$ ← "RMSprop" β_2

(편향보정)

yhat = np.array([.9, 0.2, 0.1, .4, .9])

$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t)$, $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t)$, $S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$

$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

Hyperparameters choice:

→ α : needs to be tune

→ β_1 : 0.9 → (dw)

→ β_2 : 0.999 → (dw^2)

→ ε : 10^{-8}

Adam : Adaptive moment estimation

Adam Coates

Andrew Ng

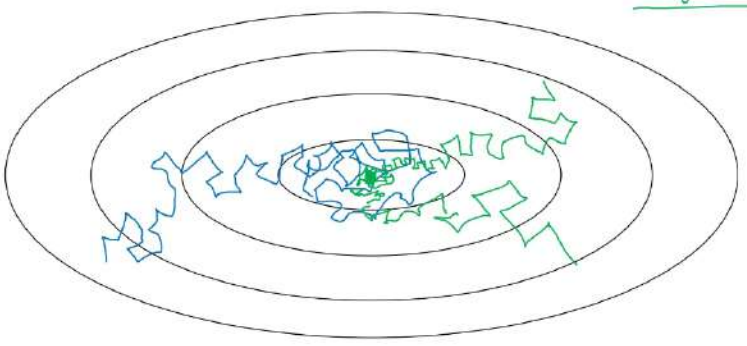
deeplearning.ai

Optimization Algorithms

Learning rate decay

Learning rate decay

Slowly reduce α

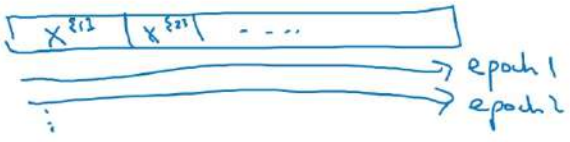


Andrew Ng

Learning rate decay

1 epoch = 1 pass through data.

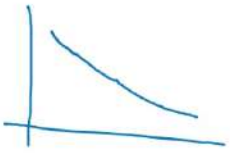
$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$



$\alpha_0 = 0.2$

decay_rate = 1

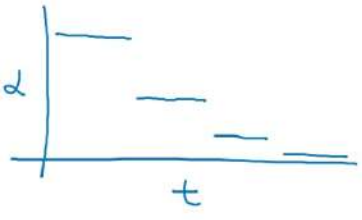
Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮



Andrew Ng

Other learning rate decay methods

formula

- $\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ — exponentially decay.
- $\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$
- 

discrete staircase
- Manual decay. step 별로 α 다르게 설정

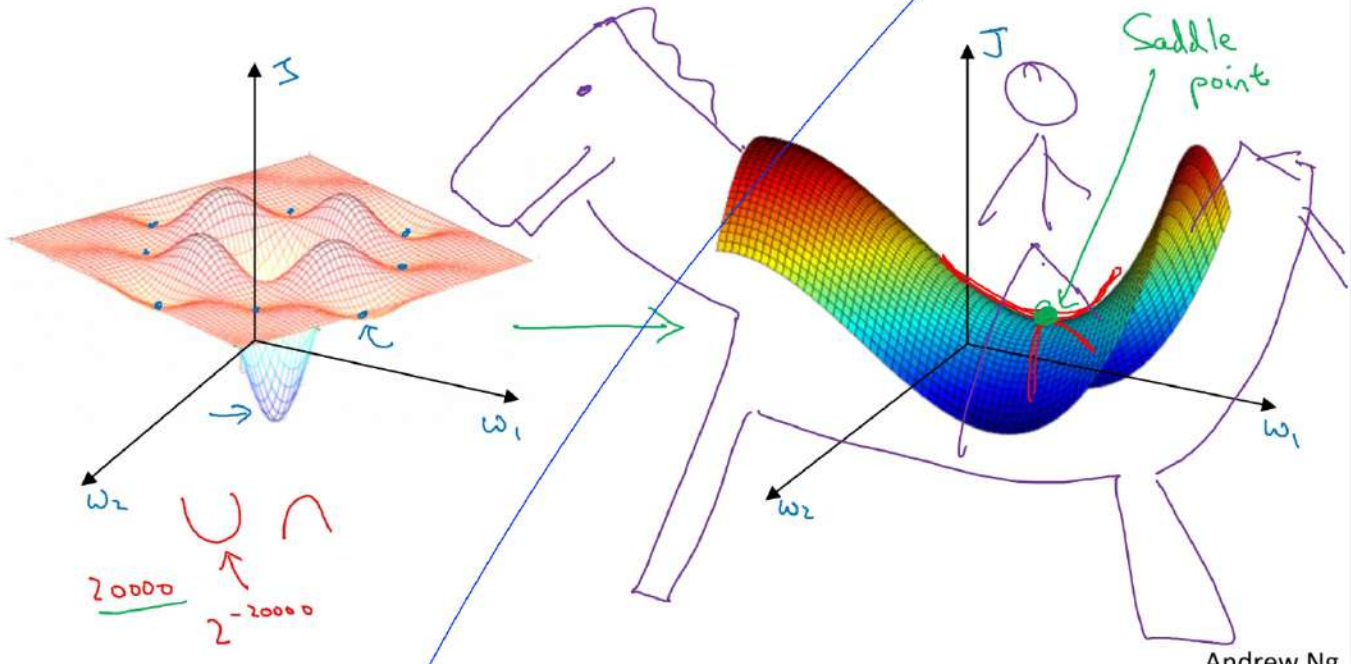
Andrew Ng

deeplearning.ai

Optimization Algorithms

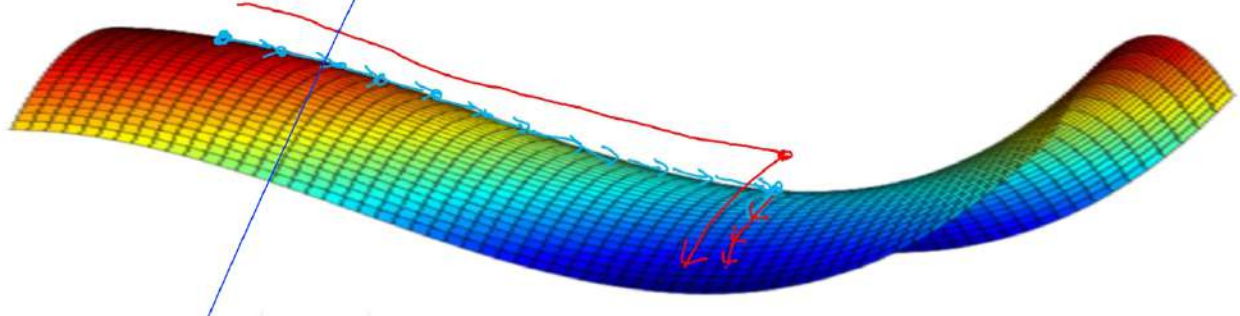
The problem of local optima

Local optima in neural networks



Andrew Ng

Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

Andrew Ng