



deeplearning.ai

# Optimization Algorithms

---

Mini-batch  
gradient descent

# Batch vs. mini-batch gradient descent

$x, y$

$x^{\{t\}}, y^{\{t\}}$

Vectorization allows you to efficiently compute on  $m$  examples.

$$X = \begin{bmatrix} x^{(1)} & x^{(2)} & x^{(3)} & \dots & x^{(1000)} & | & x^{(1001)} & \dots & x^{(2000)} & | & \dots & | & \dots & x^{(m)} \end{bmatrix}$$

$(n_x, m)$        $\underbrace{\hspace{10em}}_{X^{\{1\}} \quad (n_x, 1000)}$        $\underbrace{\hspace{10em}}_{X^{\{2\}} \quad (n_x, 1000)}$        $\dots$        $\underbrace{\hspace{10em}}_{X^{\{5,000\}} \quad (n_x, 1000)}$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)} & \dots & y^{(1000)} & | & y^{(1001)} & \dots & y^{(2000)} & | & \dots & | & \dots & y^{(m)} \end{bmatrix}$$

$(1, m)$        $\underbrace{\hspace{10em}}_{Y^{\{1\}} \quad (1, 1000)}$        $\underbrace{\hspace{10em}}_{Y^{\{2\}} \quad (1, 1000)}$        $\dots$        $\underbrace{\hspace{10em}}_{Y^{\{5,000\}} \quad (1, 1000)}$

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

5,000 mini-batches of 1,000 each

Mini-batch  $t$ :  $x^{\{t\}}, y^{\{t\}}$

$$\left| \begin{array}{l} x^{(i)} \\ z^{[l]} \\ x^{\{t\}}, y^{\{t\}} \end{array} \right.$$

# Mini-batch gradient descent

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

Forward prop on  $X^{\{t\}}$ .

$$Z^{\{t\}} = W^{\{t\}} X^{\{t\}} + b^{\{t\}}$$

$$A^{\{t\}} = g^{\{t\}}(Z^{\{t\}})$$

$$\vdots$$

$$A^{\{t\}} = g^{\{t\}}(Z^{\{t\}})$$

Vectorized implementation  
(1000 examples)

Compute cost  $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^L \ell(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{\mathbf{w}} \|W^{\{t\}}\|_F^2$ .

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

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

$$W^{\{t+1\}} := W^{\{t\}} - \alpha dW^{\{t\}}, \quad b^{\{t+1\}} := b^{\{t\}} - \alpha db^{\{t\}}$$

"1 epoch"

pass through training set.

1 step of gradt desc  
using  $X^{\{t+1\}}, Y^{\{t+1\}}$ .  
(as if  $m=1000$ )

$X, Y$



deeplearning.ai

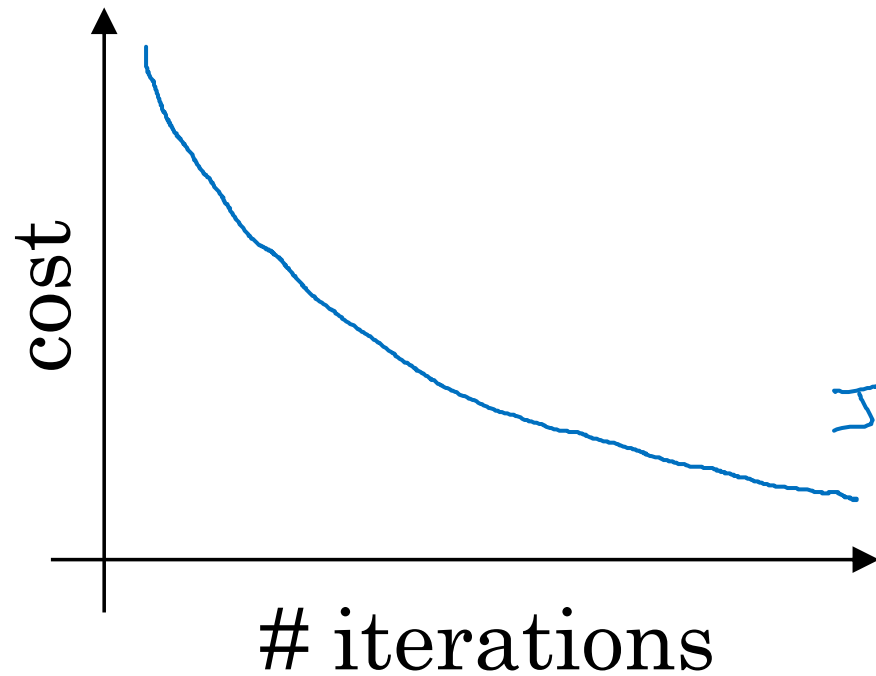
# Optimization Algorithms

---

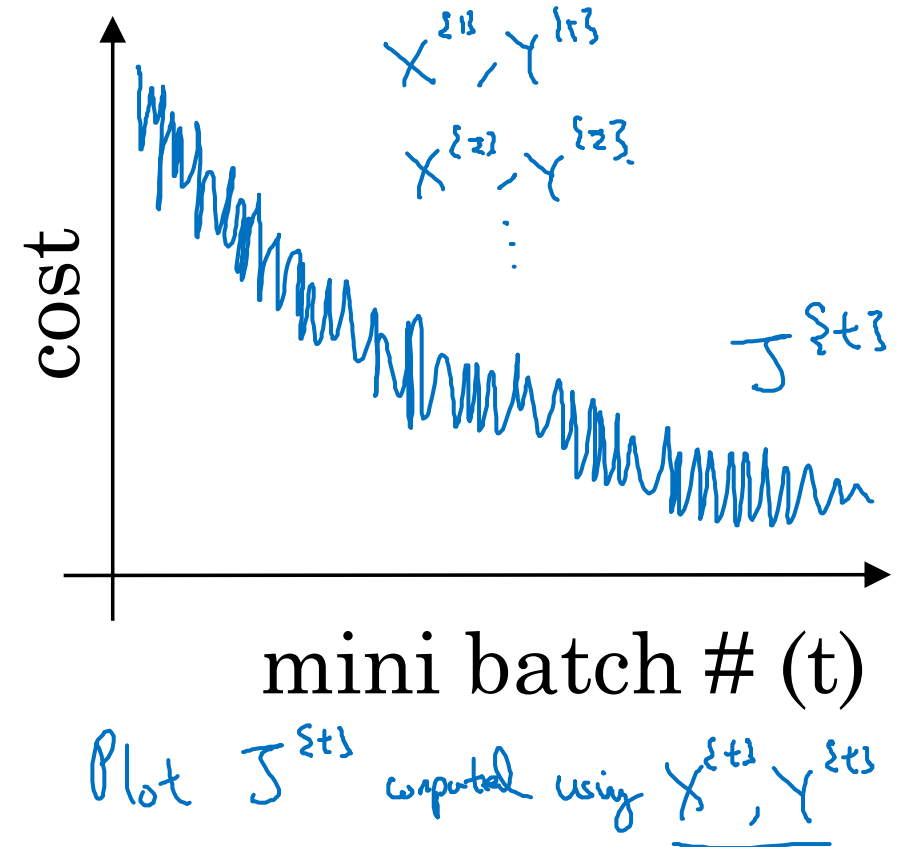
Understanding  
mini-batch  
gradient descent

# Training with mini batch gradient descent

## Batch gradient descent



## Mini-batch gradient descent



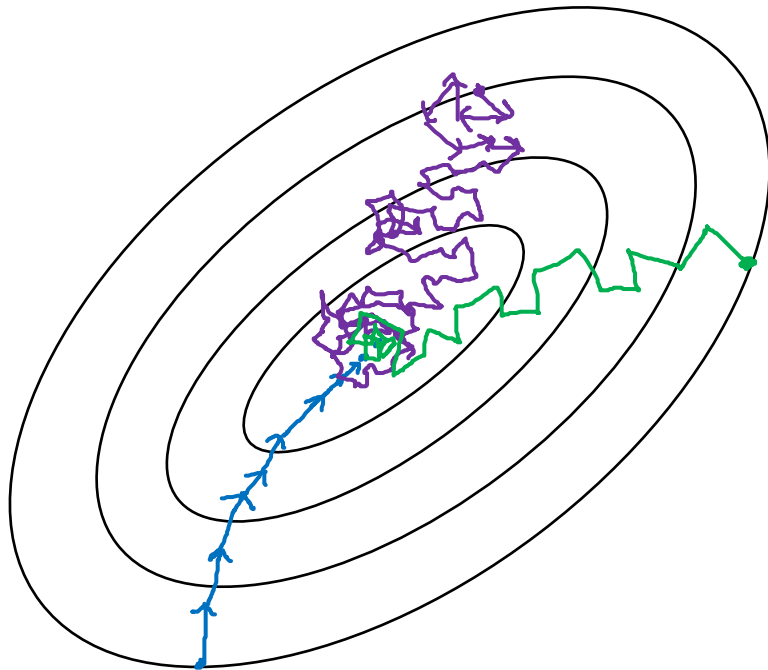
# Choosing your mini-batch size

→ If mini-batch size =  $m$  : Batch gradient descent.

$$(X^{(13)}, Y^{(13)}) = (X, Y).$$

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is its own mini-batch.  
 $(X^{(13)}, Y^{(13)}) = (x^{(1)}, y^{(1)}) \dots (x^{(n)}, y^{(n)})$  mini-batch.

In practice: Somewhere in-between 1 and  $m$



Stochastic  
gradient  
descent

Loss spikes  
from vectorization

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

Fastest learning.

- Vectorization.  
( $n=1000$ )
- Make passes without  
processing entire training set.

Batch  
gradient descent  
(mini-batch size =  $m$ )

Too long  
per iteration

# Choosing your mini-batch size

If small toy set : Use batch gradient descent.  
( $m \leq 2000$ )

Typical mini-batch sizes:

→  $64, 128, 256, 512$   $\frac{1024}{2^{10}}$   
 $\underbrace{2^6 \quad 2^7 \quad 2^8 \quad 2^9}$

Make sure mini-batch fit in CPU/GPU memory.  
 $X^{(t)}, Y^{(t)}$



deeplearning.ai

# Optimization Algorithms

---

## Exponentially weighted averages



# Temperature in London

$$\theta_1 = 40^\circ\text{F} \quad 4^\circ\text{C} \quad \leftarrow$$

$$\theta_2 = 49^\circ\text{F} \quad 9^\circ\text{C}$$

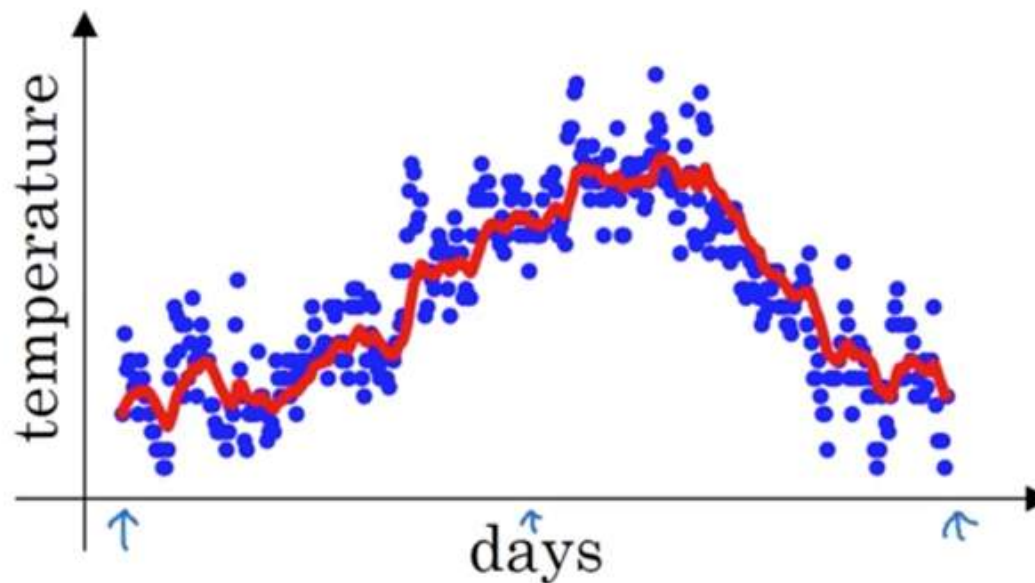
$$\theta_3 = 45^\circ\text{F} \quad \vdots$$

$\vdots$

$$\theta_{180} = 60^\circ\text{F} \quad 15^\circ\text{C}$$

$$\theta_{181} = 56^\circ\text{F} \quad \vdots$$

$\vdots$



$$V_0 = 0$$

$$V_1 = 0.9 V_0 + 0.1 \theta_1$$

$$V_2 = 0.9 V_1 + 0.1 \theta_2$$

$$V_3 = 0.9 V_2 + 0.1 \theta_3$$

$\vdots$

$$V_t = 0.9 V_{t-1} + 0.1 \theta_t$$

# Exponentially weighted averages <sup>moving</sup>

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

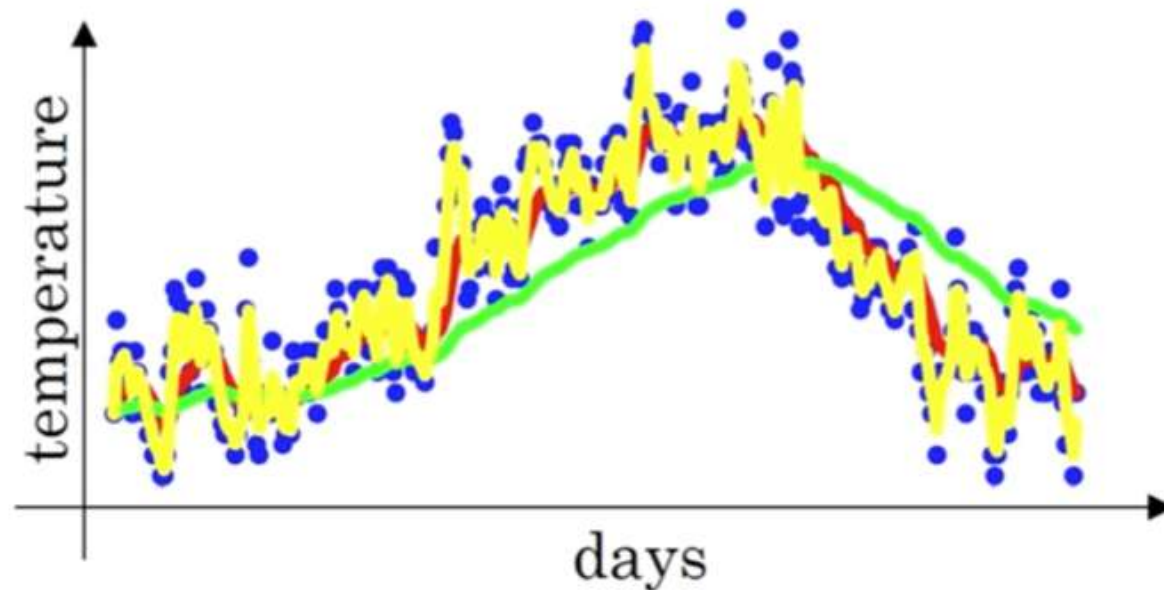
$\beta = 0.9$  :  $\approx 10$  days' temperature.

$\beta = 0.98$  :  $\approx 50$  days

$\beta = 0.5$  :  $\approx 2$  days

$V_t$  is approximately  
average over  
 $\rightarrow \approx \frac{1}{1-\beta}$  days' temperature.

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





deeplearning.ai

# Optimization Algorithms

---

Understanding  
exponentially  
weighted averages

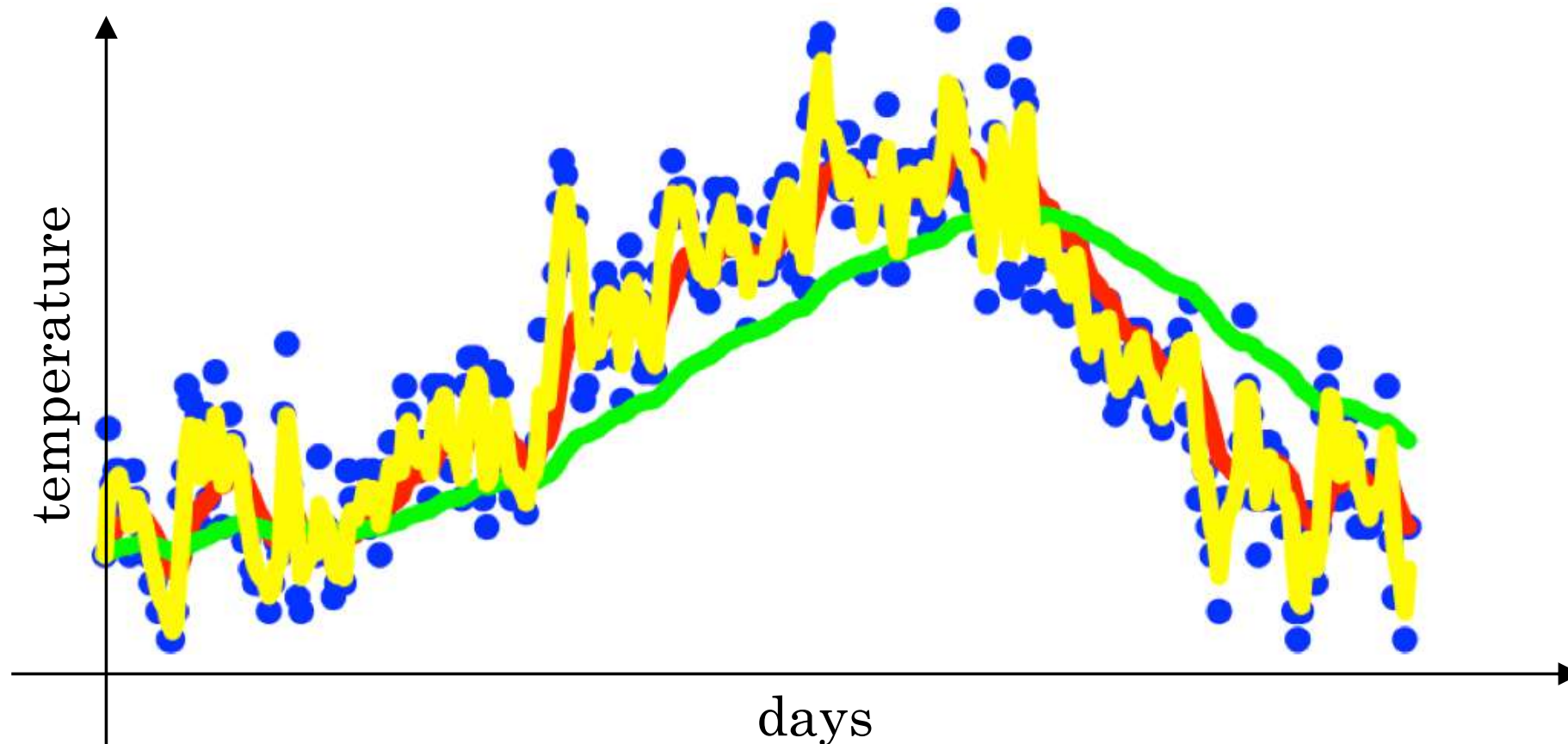
# Exponentially weighted averages

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

$$\beta = 0.9$$

$$0.98$$

$$0.5$$



# Exponentially weighted averages

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

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

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

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

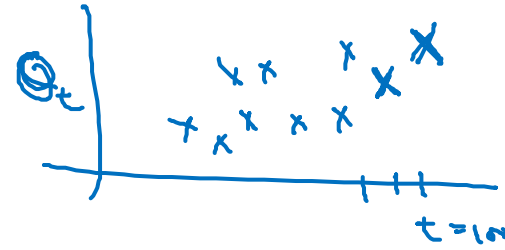
...

$$\begin{aligned} \rightarrow v_{100} &= 0.1\theta_{100} + 0.9 \cancel{v_{99}} (0.1\theta_{99} + 0.9 \cancel{v_{98}}) \\ &= \underbrace{0.1\theta_{100}} + \underbrace{0.1 \times 0.9 \cdot \theta_{99}} + \underbrace{0.1 (0.9)^2 \theta_{98}} + \underbrace{0.1 (0.9)^3 \theta_{97}} + \underbrace{0.1 (0.9)^4 \theta_{96}} + \dots \end{aligned}$$

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

$$\frac{(1-\epsilon)^{1/\epsilon}}{0.9} \approx \frac{1}{e}$$

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



$$\approx \frac{1}{1-\beta}$$

$$\epsilon = 1 - \beta$$

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

# Implementing exponentially weighted averages

$$v_0 = 0$$

$$v_1 = \beta v_0 + (1 - \beta) \theta_1$$

$$v_2 = \beta v_1 + (1 - \beta) \theta_2$$

$$v_3 = \beta v_2 + (1 - \beta) \theta_3$$

...

$$V_\theta := 0$$

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

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

$\vdots$

---

$$\rightarrow V_\theta = 0$$

Repeat {

Get next  $\theta_t$

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

}



deeplearning.ai

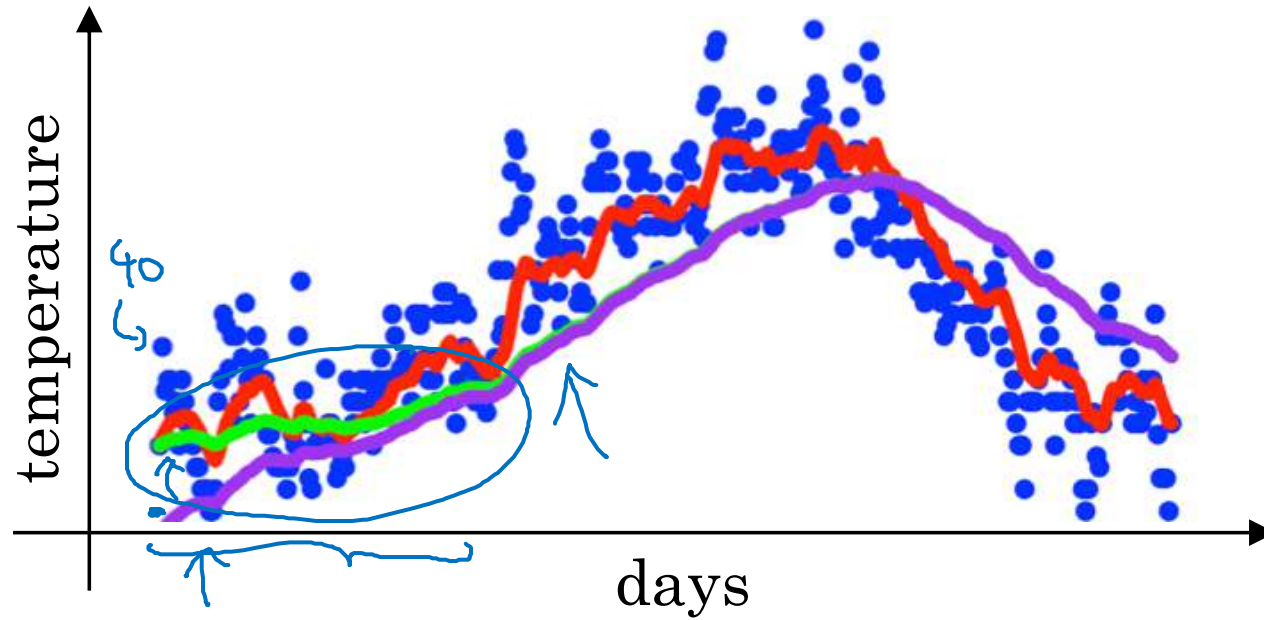
# Optimization Algorithms

---

Bias correction  
in exponentially  
weighted average



# Bias correction



$$\beta = 0.98$$

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

$$v_0 = 0$$

$$v_1 = \cancel{0.98 v_0} + \underbrace{0.02 \theta_1}$$

$$v_2 = 0.98 v_1 + 0.02 \theta_2$$

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

$$= \underline{0.0196 \theta_1} + \underline{0.02 \theta_2}$$

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

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

$$\frac{v_2}{0.0396} = \frac{0.0196 \theta_1 + 0.02 \theta_2}{0.0396}$$





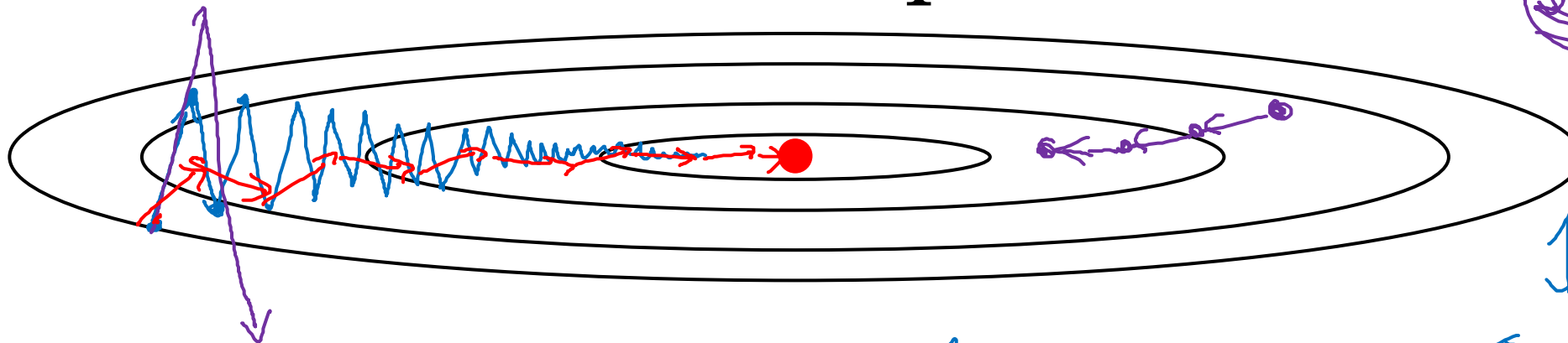
deeplearning.ai

# Optimization Algorithms

---

## Gradient descent with momentum

# Gradient descent example



↑ slower learning  
↔ faster learning.

Momentum:

On iteration  $t$ :

Compute  $\Delta W, \Delta b$  on current mini-batch.

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

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

friction — ↑ velocity

↑ acceleration

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



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

# Implementation details

$$v_{dW} = 0, \quad v_{db} = 0$$

On iteration  $t$ :

Compute  $dW, db$  on the current mini-batch

$$\left. \begin{aligned} \rightarrow v_{dW} &= \beta v_{dW} + (1 - \beta) dW \\ \rightarrow v_{db} &= \beta v_{db} + (1 - \beta) db \end{aligned} \right\} \quad \left| \quad \underbrace{v_{dW} = \beta v_{dW} + dW}_{\leftarrow}$$

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

$$\frac{\cancel{v_{dW}}}{\cancel{1 - \beta} t}$$

Hyperparameters:  $\alpha, \beta$

$$\underline{\beta = 0.9}$$

average over last  $\approx 10$  gradients



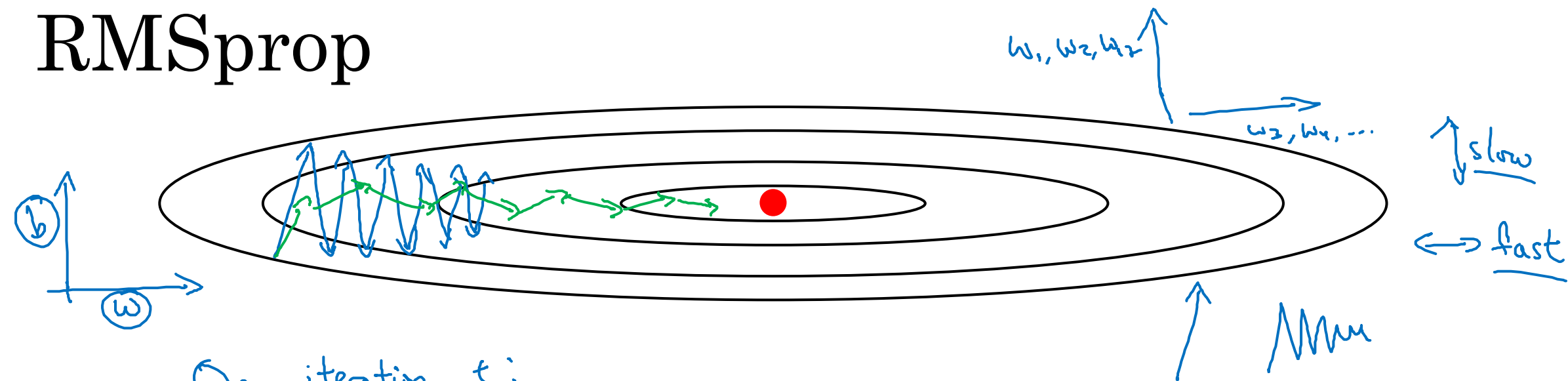
deeplearning.ai

# Optimization Algorithms

---

## RMSprop

# RMSprop



On iteration  $t$ :

Compute  $dw, db$  on current mini-batch

$$\underline{S_{dw}} = \beta_2 \underline{S_{dw}} + (1 - \beta_2) \underline{dw^2} \leftarrow \text{small}$$

element-wise

$$\rightarrow \underline{S_{db}} = \beta_2 \underline{S_{db}} + (1 - \beta_2) \underline{db^2} \leftarrow \text{large}$$

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw} + \epsilon}}$$

$\epsilon = 10^{-8}$

$$b := b - \alpha \frac{db}{\sqrt{S_{db} + \epsilon}}$$



deeplearning.ai

# Optimization Algorithms

---

## Adam optimization algorithm

# 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

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dw, \quad V_{db} = \beta_1 V_{db} + (1 - \beta_1) db \quad \leftarrow \text{"momentum"} \beta_1$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dw^2, \quad S_{db} = \beta_2 S_{db} + (1 - \beta_2) db^2 \quad \leftarrow \text{"RMSprop"} \beta_2$$

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

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

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

$$W := W - \alpha \frac{V_{dw}^{\text{corrected}}}{\sqrt{S_{dw}^{\text{corrected}} + \epsilon}}$$

$$b := b - \alpha \frac{V_{db}^{\text{corrected}}}{\sqrt{S_{db}^{\text{corrected}} + \epsilon}}$$

# Hyperparameters choice:

→  $\alpha$  : needs to be tune

→  $\beta_1$  : 0.9 → ( $dw$ )

→  $\beta_2$  : 0.999 → ( $dw^2$ )

→  $\epsilon$  :  $10^{-8}$

Adam : Adaptive moment estimation



Adam Coates





deeplearning.ai

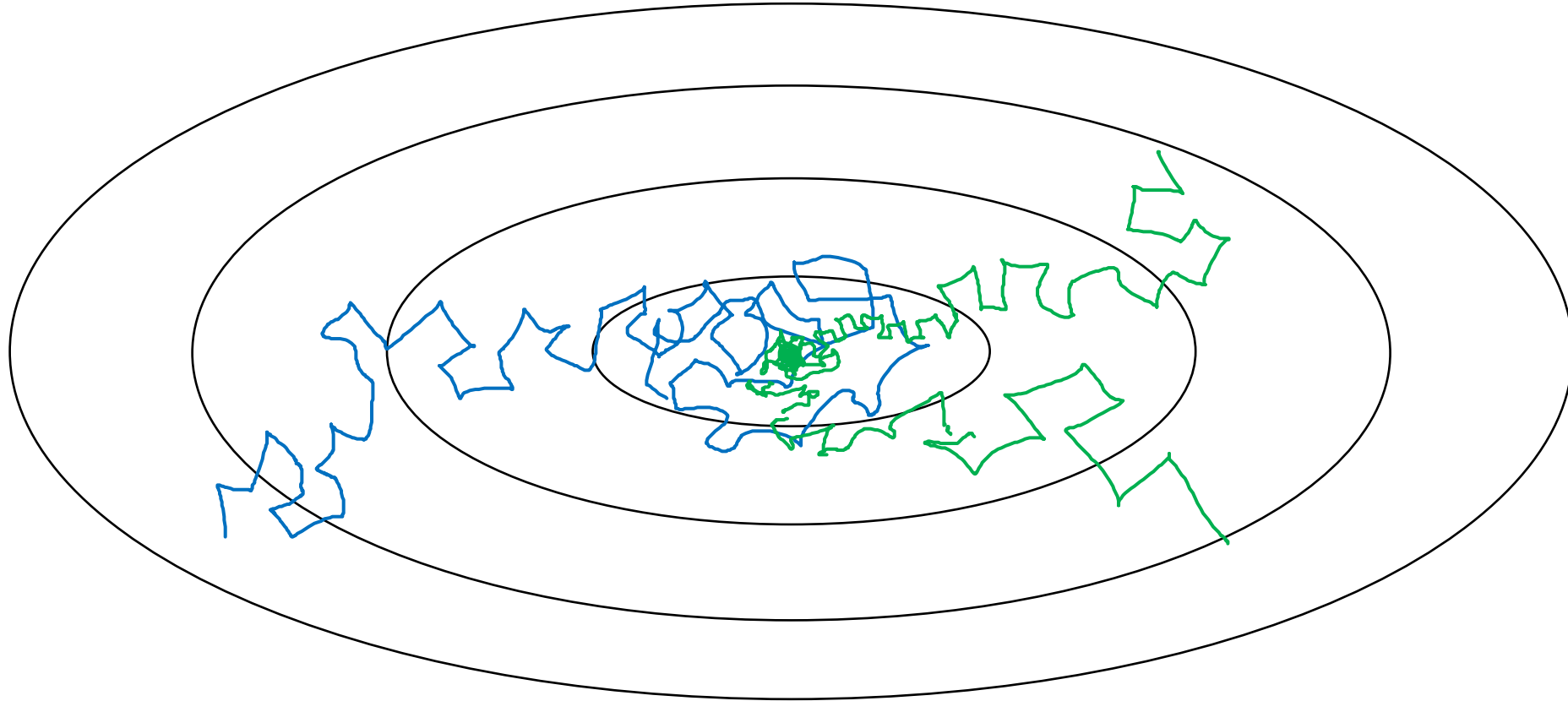
# Optimization Algorithms

---

## Learning rate decay

# Learning rate decay

Slowly reduce  $\alpha$

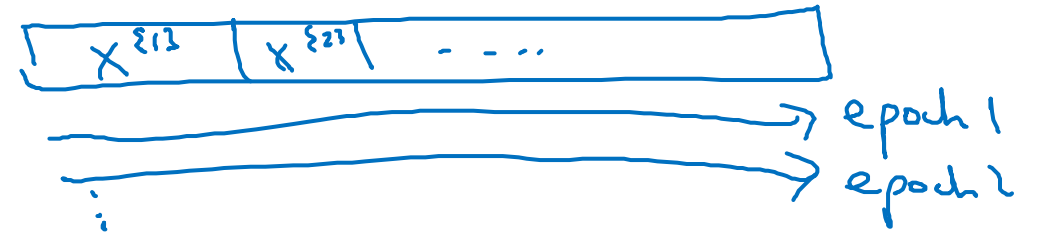


# Learning rate decay

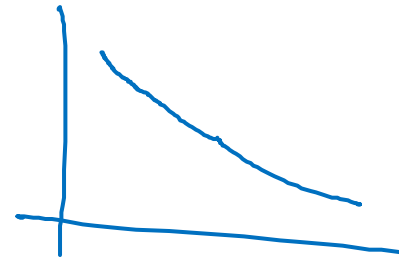
1 epoch = 1 pass through data.

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

Epoch	$\alpha$
1	0.1
2	0.67
3	0.5
4	0.4
$\vdots$	$\vdots$



$$\alpha_0 = 0.2$$
$$\text{decay-rate} = 1$$



# 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.



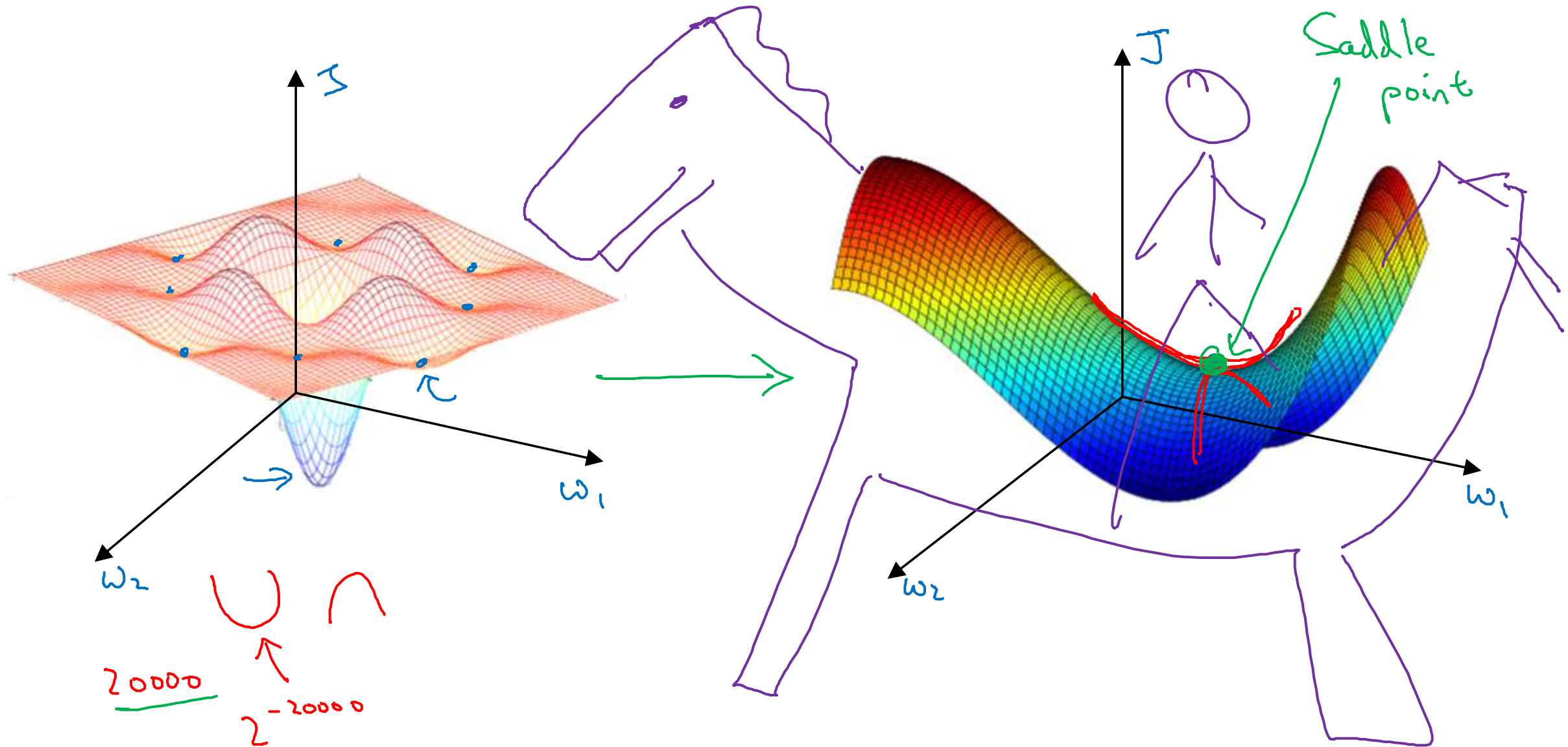
deeplearning.ai

# Optimization Algorithms

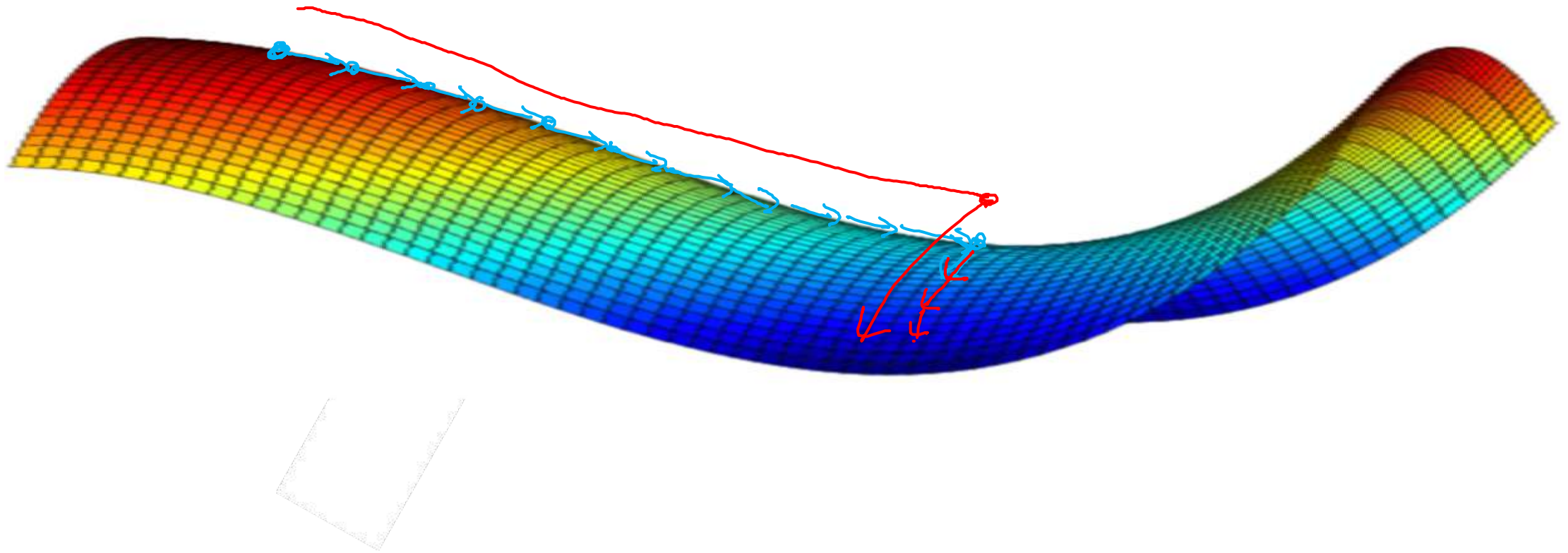
---

## The problem of local optima

# Local optima in neural networks



# Problem of plateaus



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