Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode



deeplearning.ai

Optimization Algorithms

Mini-batch gradient descent

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)}\ \cdots\ x^{(1000)}}_{X^{\{1\}}\ (n_x,1000)} \mid \underbrace{x^{(1001)}\ \cdots\ x^{(2000)}}_{X^{\{2\}}\ (n_x,1000)} \mid \cdots \mid \underbrace{\cdots\ x^{(m)}}_{X^{\{5{,}000\}}\ (n_x,1000)}]$

(n_x, m)

$Y = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)}\ \cdots\ y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)}\ \cdots\ y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \cdots \mid \underbrace{\cdots\ y^{(m)}}_{Y^{\{5{,}000\}}\ (1,1000)}]$

$(1, m)$

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

5,000 mini-batches of 1,000 each

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

$x^{(i)}$

$z^{[l]}$

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

Andrew Ng

Mini-batch gradient descent

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

X, Y

Repeat {

for $t = 1, \ldots, 5000$ {

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

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

} Vectorized implementation (1000 examples)

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

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"
└ pass through training set.

Andrew Ng



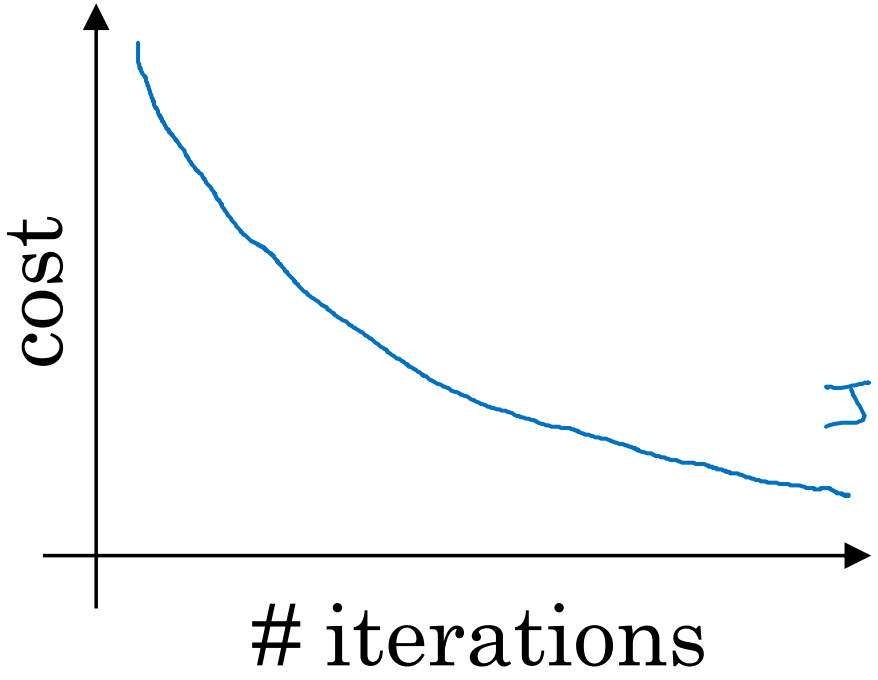
deeplearning.ai

Optimization Algorithms

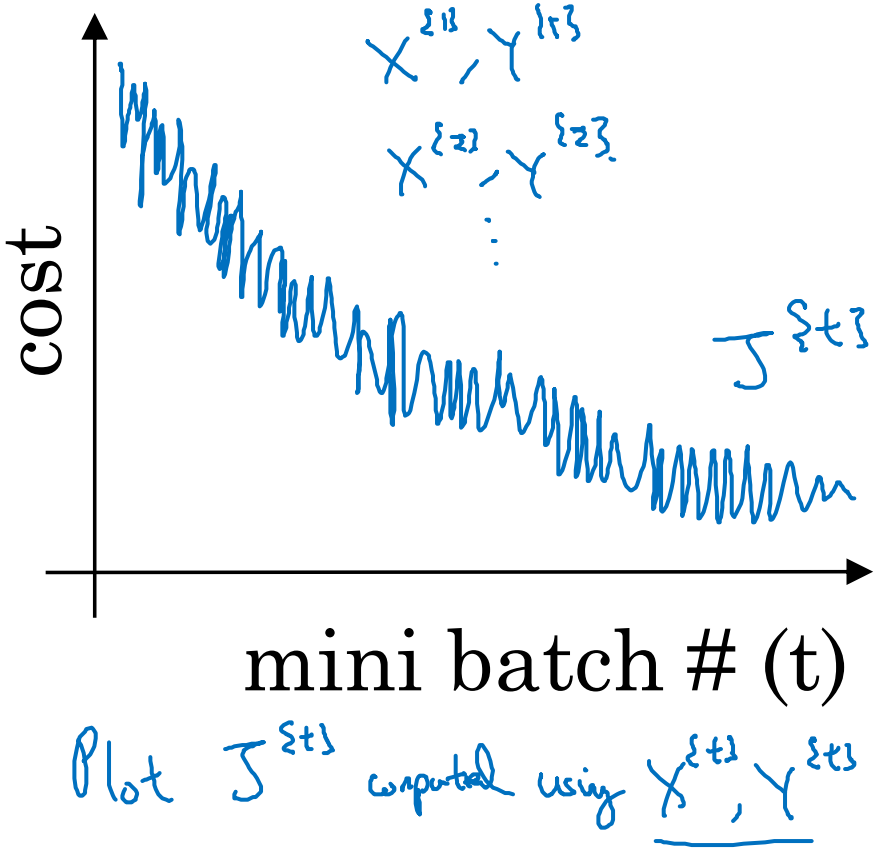
Understanding mini-batch gradient descent

Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent

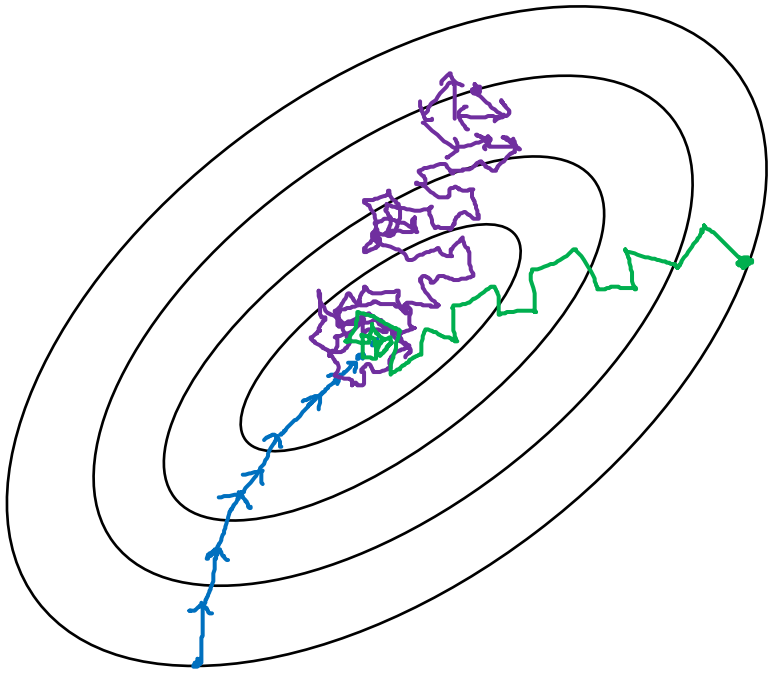


Choosing your mini-batch size

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

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is it own mini-batch. $(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)})$ ... $(x^{(2)}, y^{(1)})$

In practice: Somewhere in-between 1 and m



Stochastic gradient descent

{

Lose speedup from vectorization

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

Fastest learning.

- Vectorization. (n 1000)
- 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 train 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})

Make sure mini-batch fit in CPU/GPU memory.

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



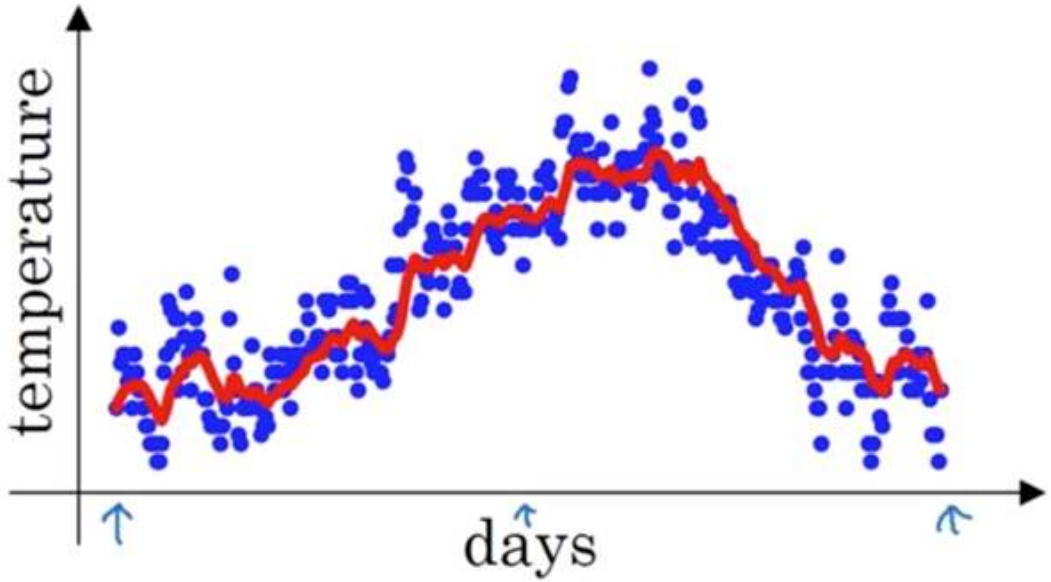
deeplearning.ai

Optimization Algorithms

Exponentially weighted averages

Temperature in London

$\theta_1 = 40°F$ 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

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

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

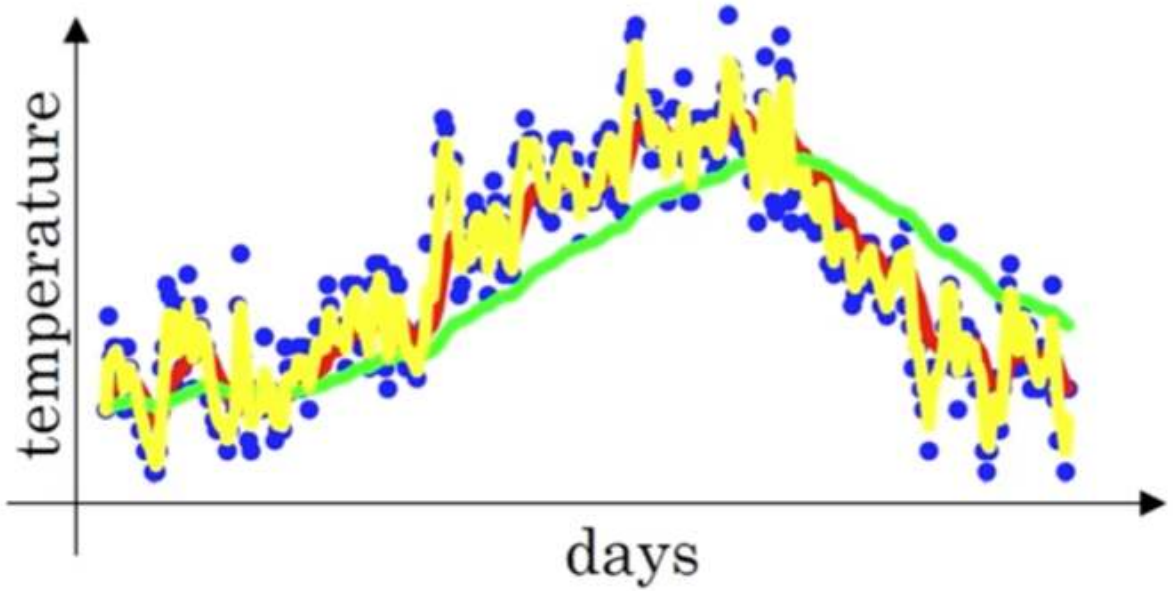
$\beta = 0.98$: $\approx$ 50 days

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

V_t as approximately average over

$\rightarrow \approx \frac{1}{1-\beta}$ days' temperature.

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





deeplearning.ai

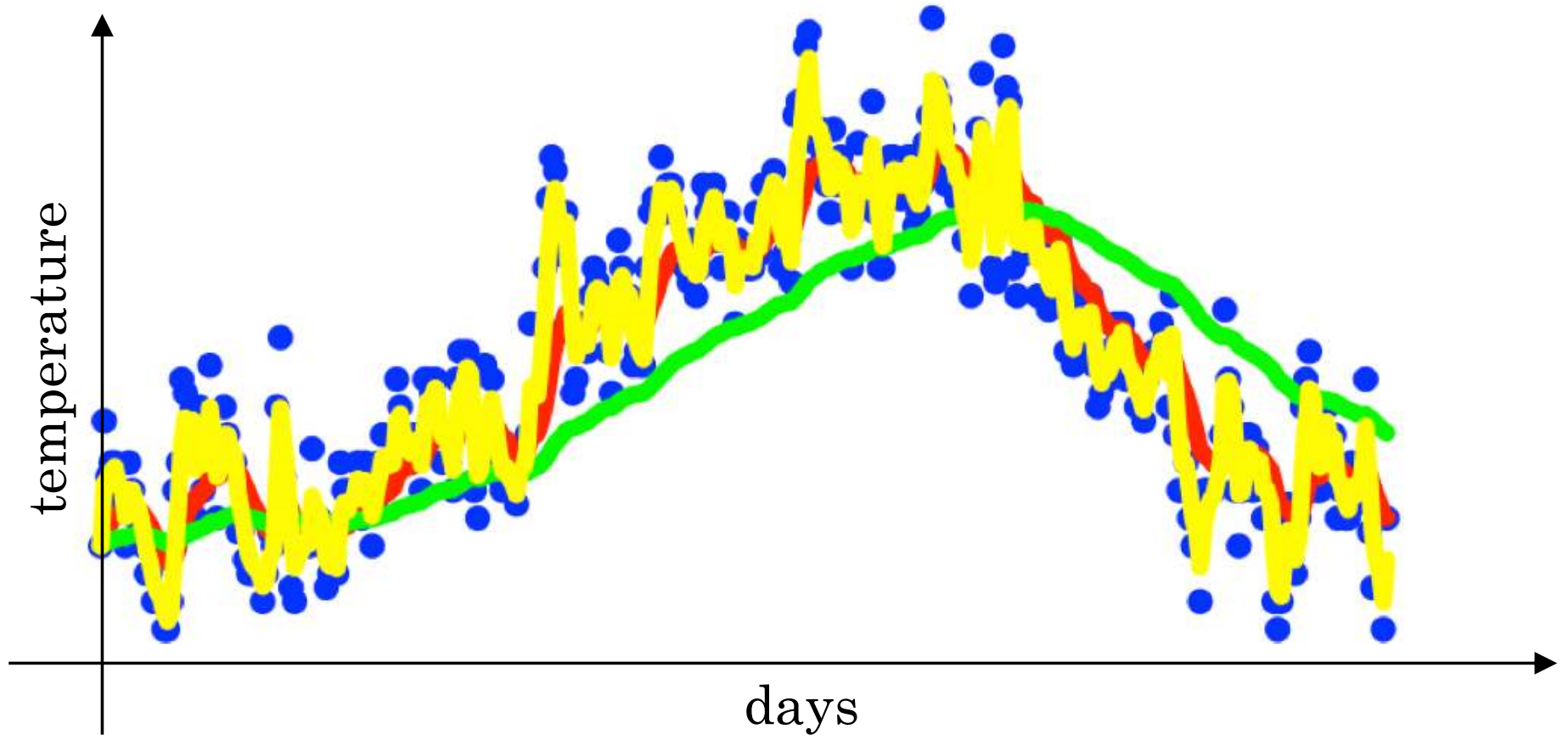
Optimization Algorithms

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

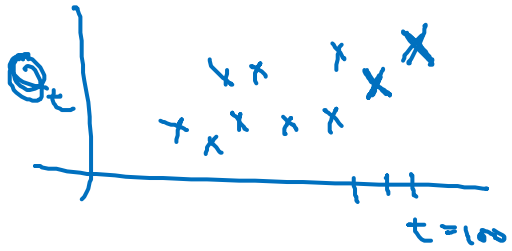
$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

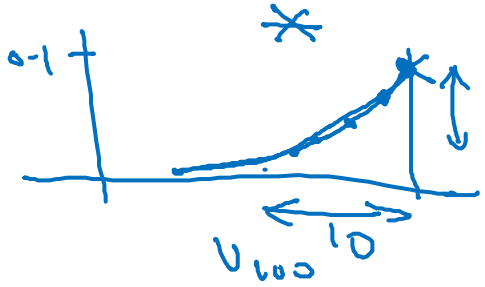
$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$

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

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

...





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

$\varepsilon = 1-\beta$

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

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

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

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

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

$0.98^{?}$

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

Andrew Ng

Implementing exponentially weighted averages

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

$V_\theta := 0$

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

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

$\vdots$

$V_\theta = 0$

Repeat {

Get next θ_t

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

}

Andrew Ng

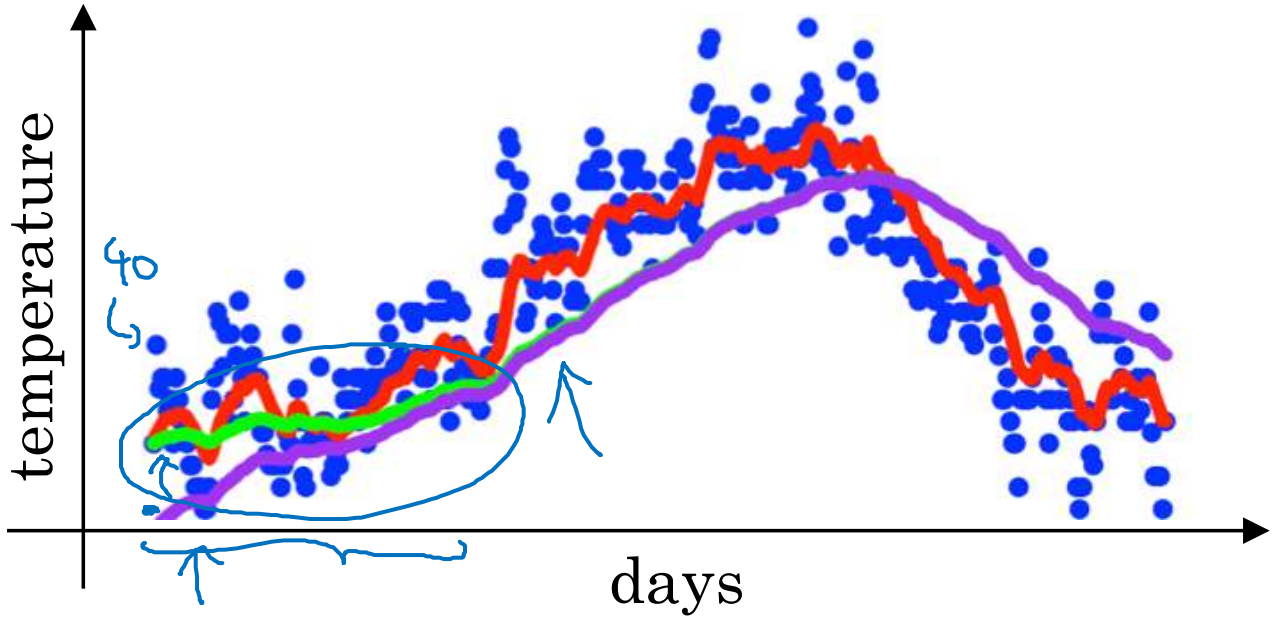


deeplearning.ai

Optimization Algorithms

Bias correction in exponentially weighted average

Bias correction



$\beta = 0.98$

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

$v_0 = 0$

$v_1 = 0.98 v_0 + 0.02 \theta_1$

$v_2 = 0.98 v_1 + 0.02 \theta_2$

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

$= 0.0196 \theta_1 + 0.02 \theta_2$

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

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

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

Andrew Ng

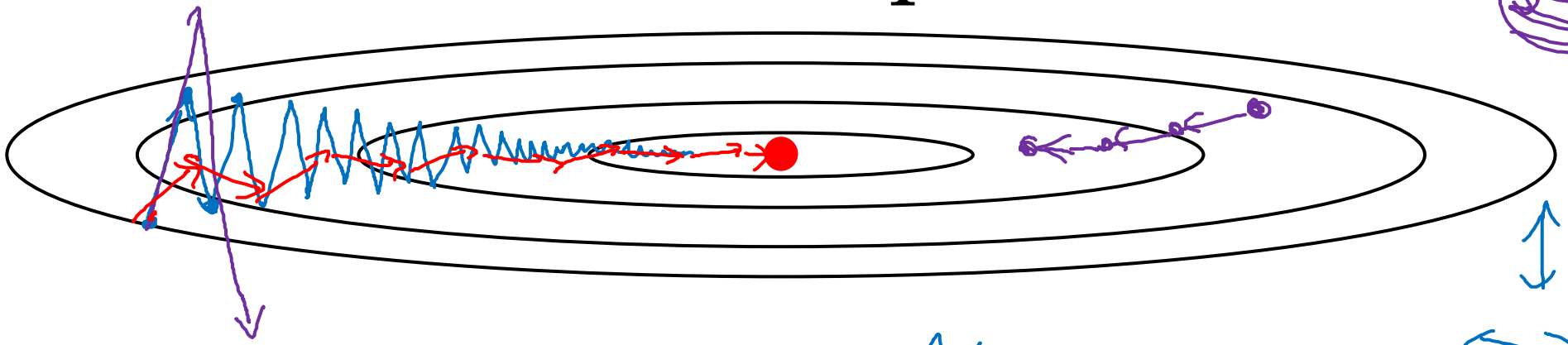


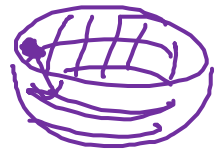
deeplearning.ai

Optimization Algorithms

Gradient descent with momentum

Gradient descent example





slower learning

faster learning.



Momentum:

On iteration t:

Compute dW, db on current mini-batch.

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

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

"$V_\theta = \beta V_{\theta-1} + (1-\beta)\theta_t$"

friction — velocity — acceleration

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

Andrew Ng

Implementation details

$V_{dW} = 0$, $V_{db} = 0$

On iteration t:

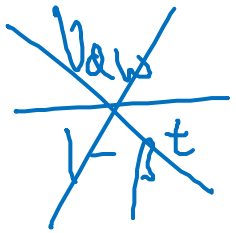
Compute dW, db on the current mini-batch

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

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

$V_{dW} = \beta V_{dW} + dW$

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



Hyperparameters: α, β

$\beta = 0.9$

average over last ≈ 10 gradients

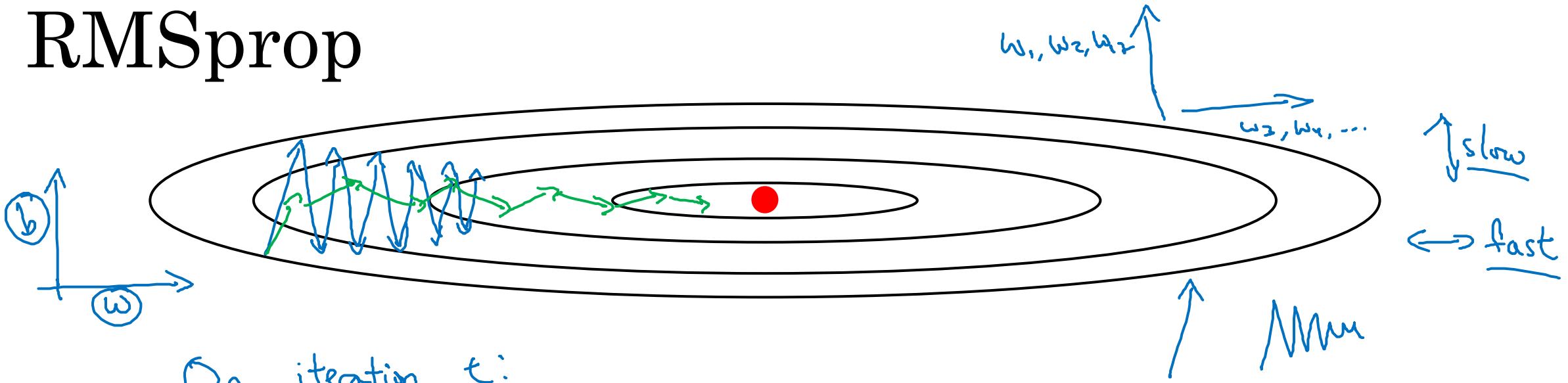


deeplearning.ai

Optimization Algorithms

RMSprop

RMSprop



On iteration t:

Compute dW, db on current mini-batch

element-wise

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

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

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

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

$\varepsilon = 10^{-8}$

Andrew Ng



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

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

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

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

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

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

Andrew Ng

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



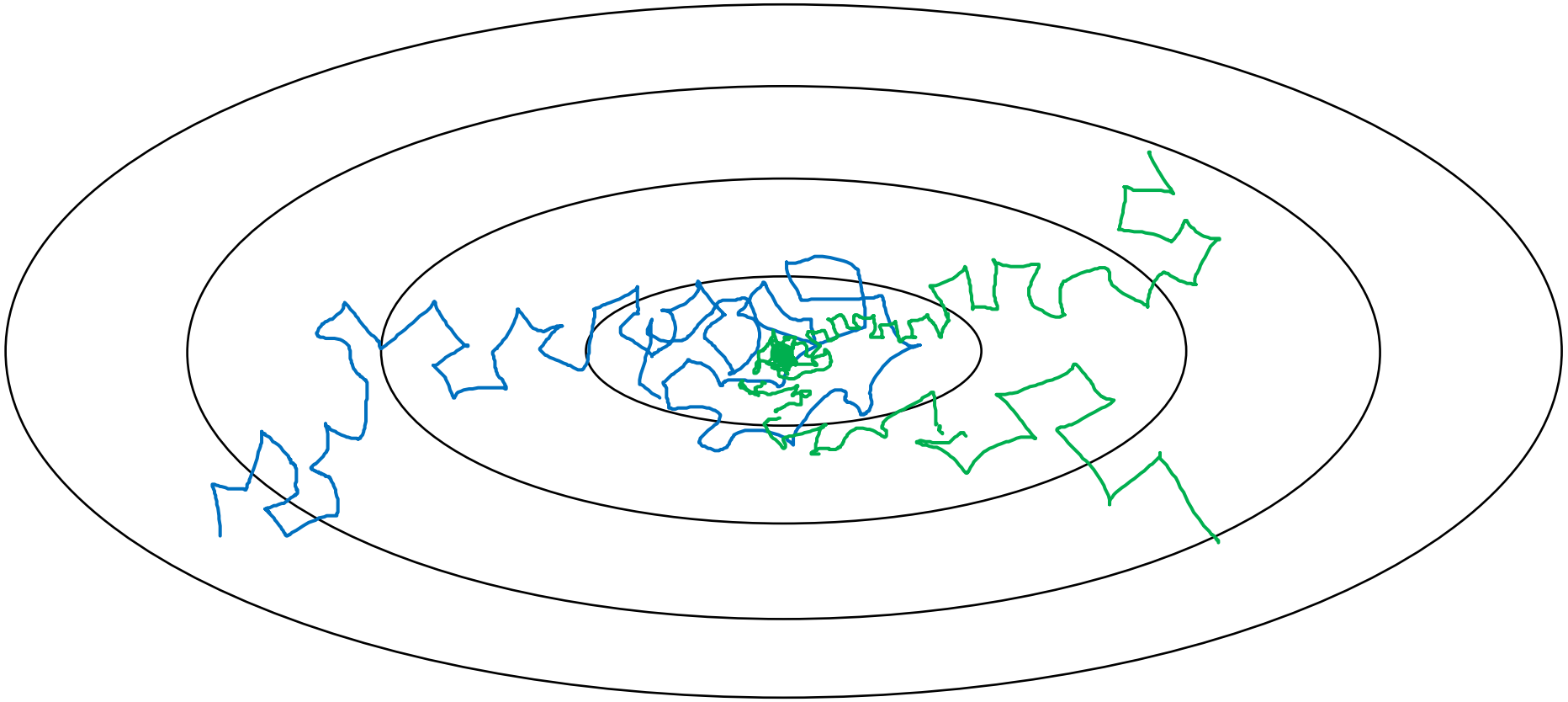
deeplearning.ai

Optimization Algorithms

Learning rate decay

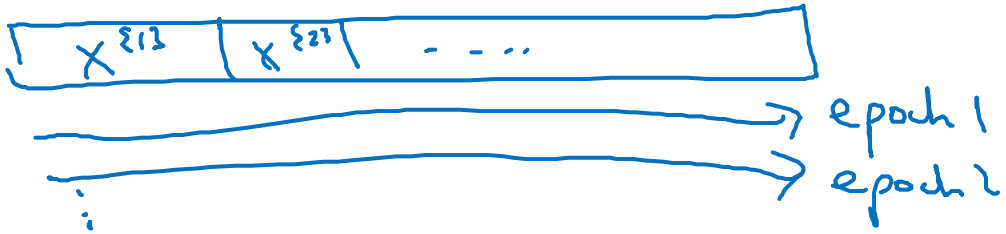
Learning rate decay

Slowly reduce α



Learning rate decay

1 epoch = 1 pass through data.

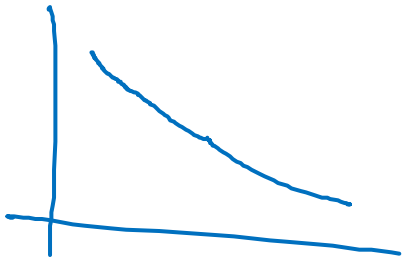


$$\alpha = \frac{1}{1 + \text{decay_rate} * \text{epoch_num}} \alpha_0$$

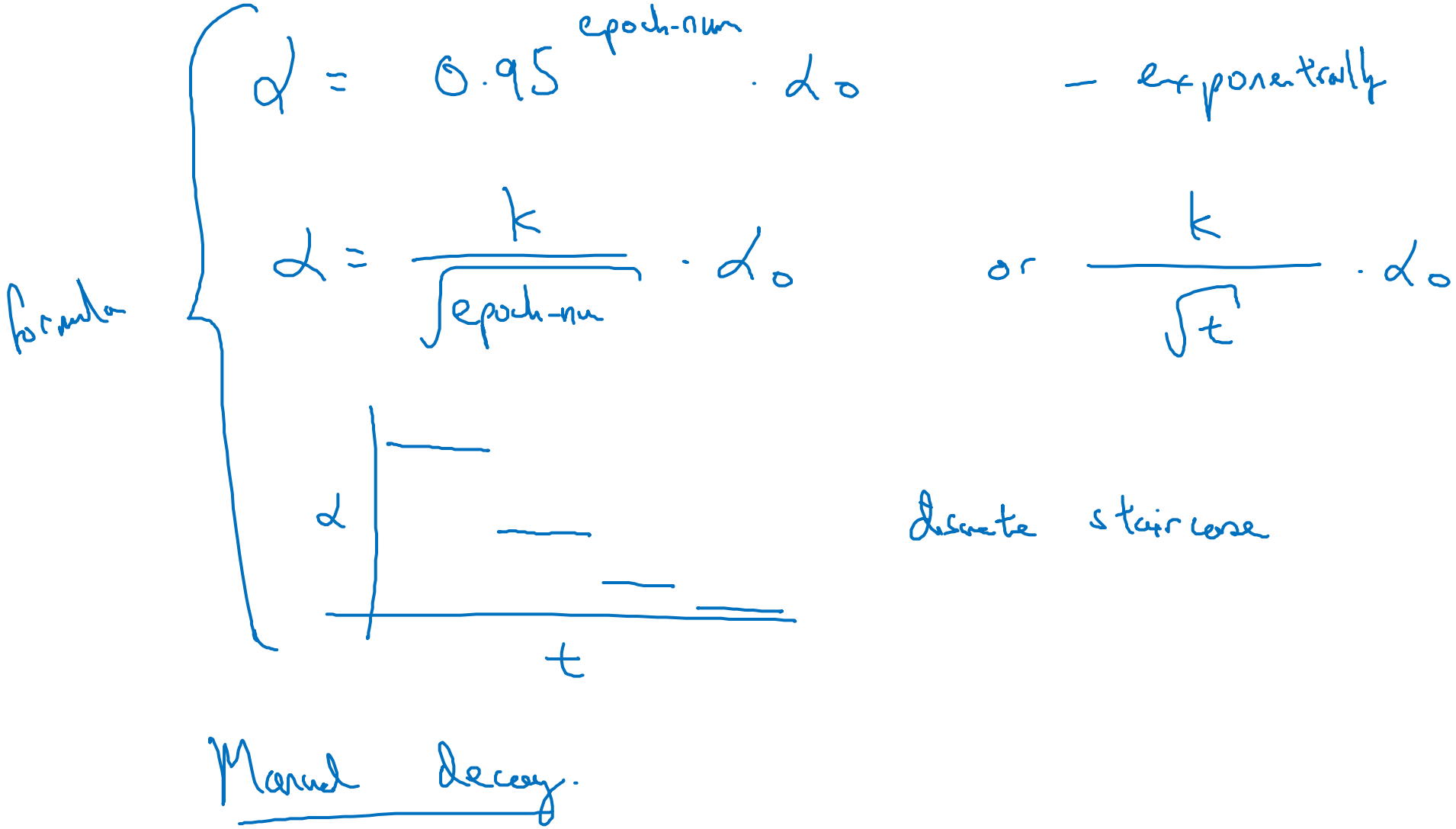
$\alpha_0 = 0.2$

decay_rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮



Other learning rate decay methods



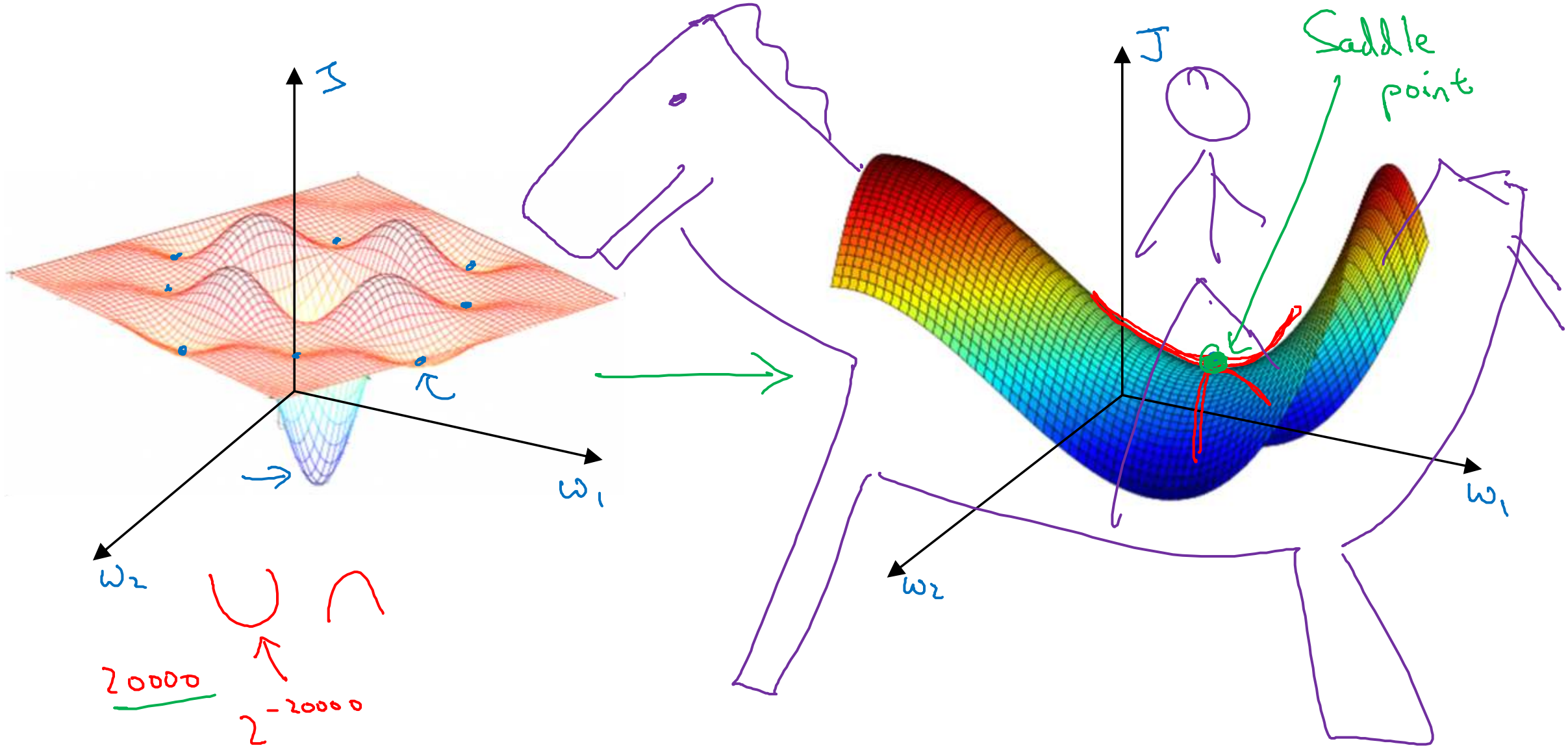


deeplearning.ai

Optimization Algorithms

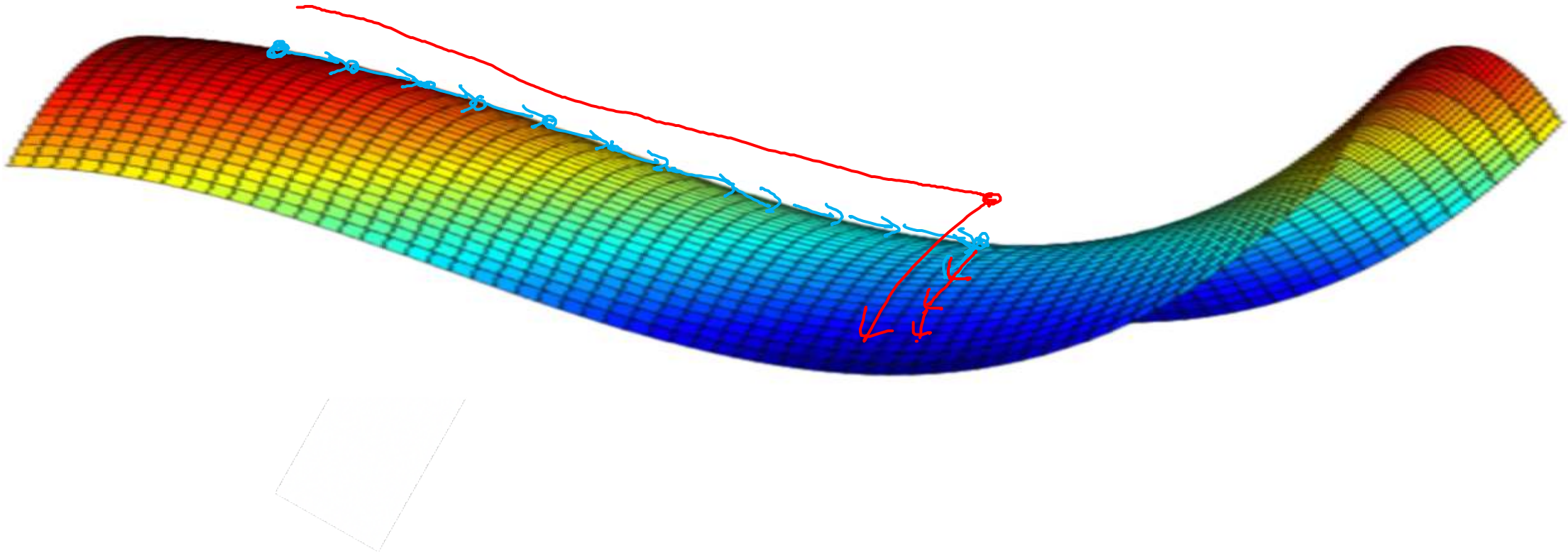
The problem of local optima

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