deeplearning.ai

Optimization
Algorithms

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.

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

(n_x, m)

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

$(1, m)$

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

repeat {

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

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

$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} X^{\{t\}} + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ &\vdots \\ A^{[L]} &= g^{[L]}(Z^{[L]}) \end{aligned}\right\}$$

Vectorized implementation (1000 examples)

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

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

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.

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

X, Y

Andrew Ng

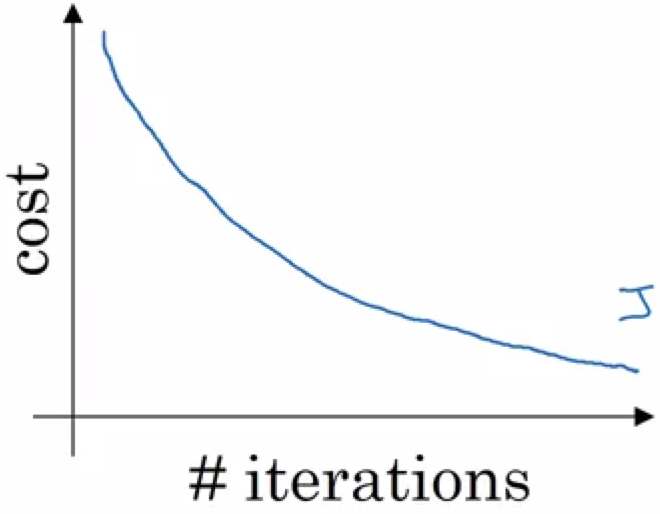
deeplearning.ai

Optimization Algorithms

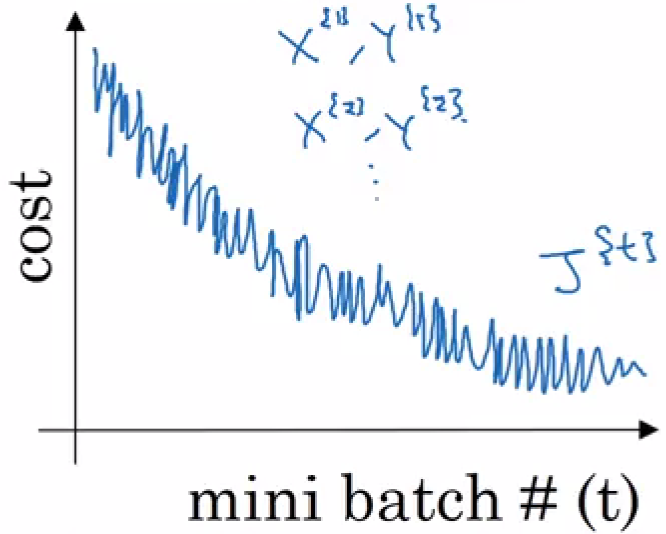
Understanding mini-batch gradient descent

Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent



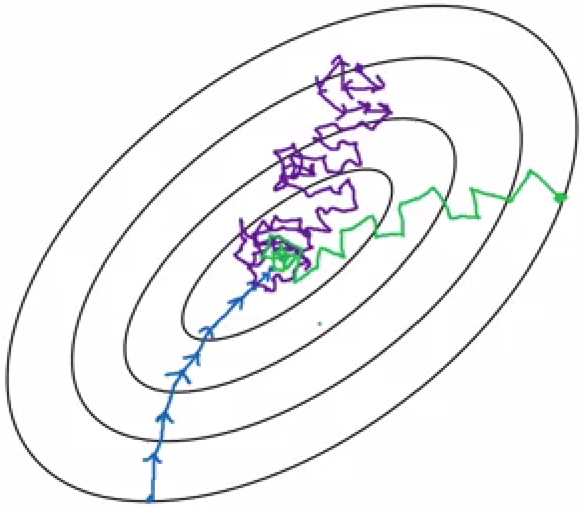
Plot $J^{\{t\}}$ computed using $\underline{X^{\{t\}}, Y^{\{t\}}}$

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
$(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)}) \dots (x^{(2)}, y^{(2)})$ mini-batch.

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. (~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 trng set: Use batch gradient descent.
$(m \leq 2000)$

Typical mini-batch sizes:

$$\rightarrow \underbrace{\underset{2^6}{64},\ \underset{2^7}{128},\ \underset{2^8}{256},\ \underset{2^9}{512}} \qquad \underset{2^{10}}{1024}$$

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

Andrew Ng

deeplearning.ai

Optimization Algorithms

Exponentially weighted averages

Temperature in London

$\theta_1 = 40°F$ 4°C ←

$\theta_2 = 49°F$ 9°C

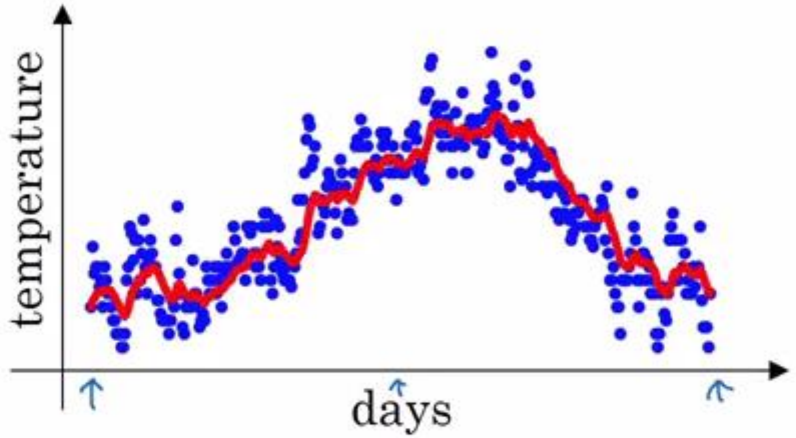
$\theta_3 = 45°F$ ⋮

⋮

$\theta_{180} = 60°F$ 15°C

$\theta_{181} = 56°F$ ⋮

⋮



$V_0 = 0$

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

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

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

⋮

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

2:31 / 5:58

Andrew Ng

Exponentially weighted averages

moving

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

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

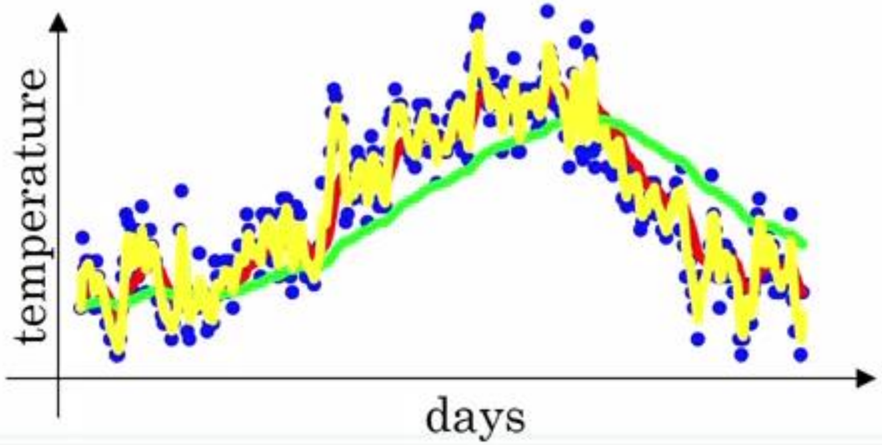
$\beta = 0.98$: ≈ 50 days

$\beta = 0.5$: ≈ 2 days

V_t as approximately average over

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

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



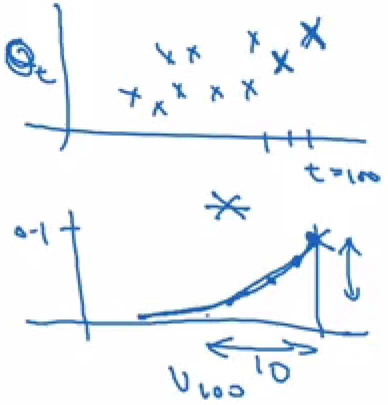
deeplearning.ai

Optimization Algorithms

Understanding exponentially weighted averages

Exponentially weighted averages

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



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

$\varepsilon = 1-\beta$

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

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

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

...

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

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

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

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

$\underbrace{(1-\varepsilon)}_{0.9}{}^{1/\varepsilon} = \frac{1}{e}$

$0.98\,?$

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

Implementing exponentially weighted averages

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

$V_\theta := 0$

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

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

$\vdots$

$\rightarrow V_\theta = 0$

Repeat {

Get next θ_t

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

}

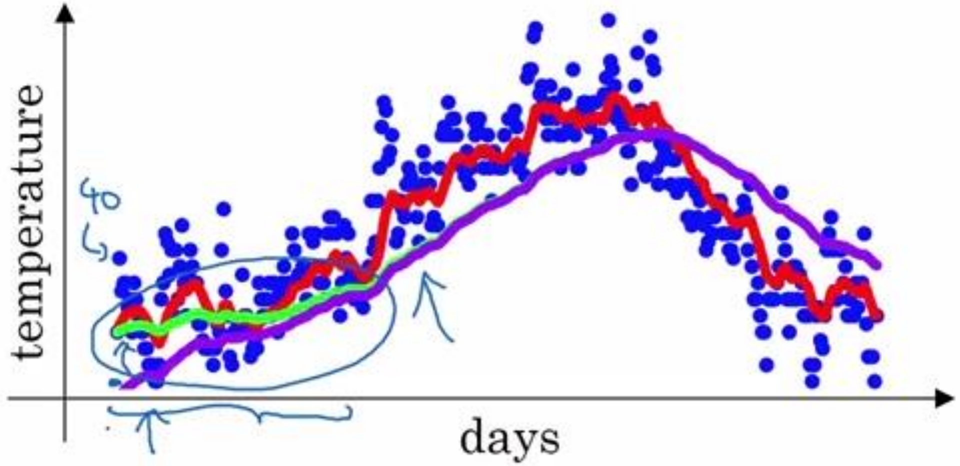
Andrew Ng

deeplearning.ai

Optimization Algorithms

Bias correction in exponentially weighted average

Bias correction



$\beta = 0.98$

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

$V_0 = 0$

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

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

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

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

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

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

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

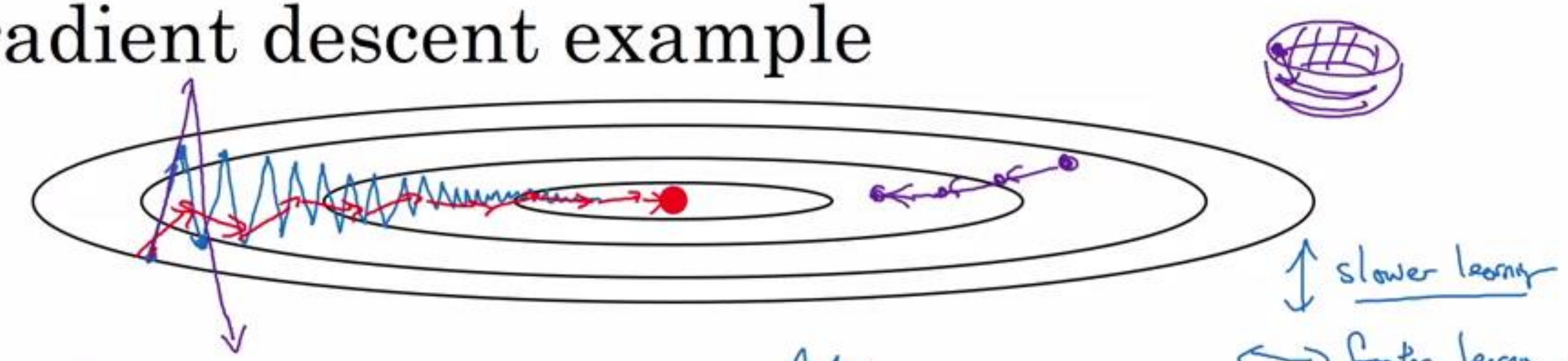
Andrew Ng

deeplearning.ai

Optimization Algorithms

Gradient descent with momentum

Gradient descent example



Momentum:

On iteration t:

Compute dW, db on current mini-batch.

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

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

friction → ↑ velocity ↑ acceleration

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

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

Andrew Ng

Implementation details

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

On iteration t:

Compute dW, db on the current mini-batch

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

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

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

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

$\cancel{\frac{V_{dW}}{1-\beta^t}}$

Hyperparameters: α, β

$\beta = 0.9$

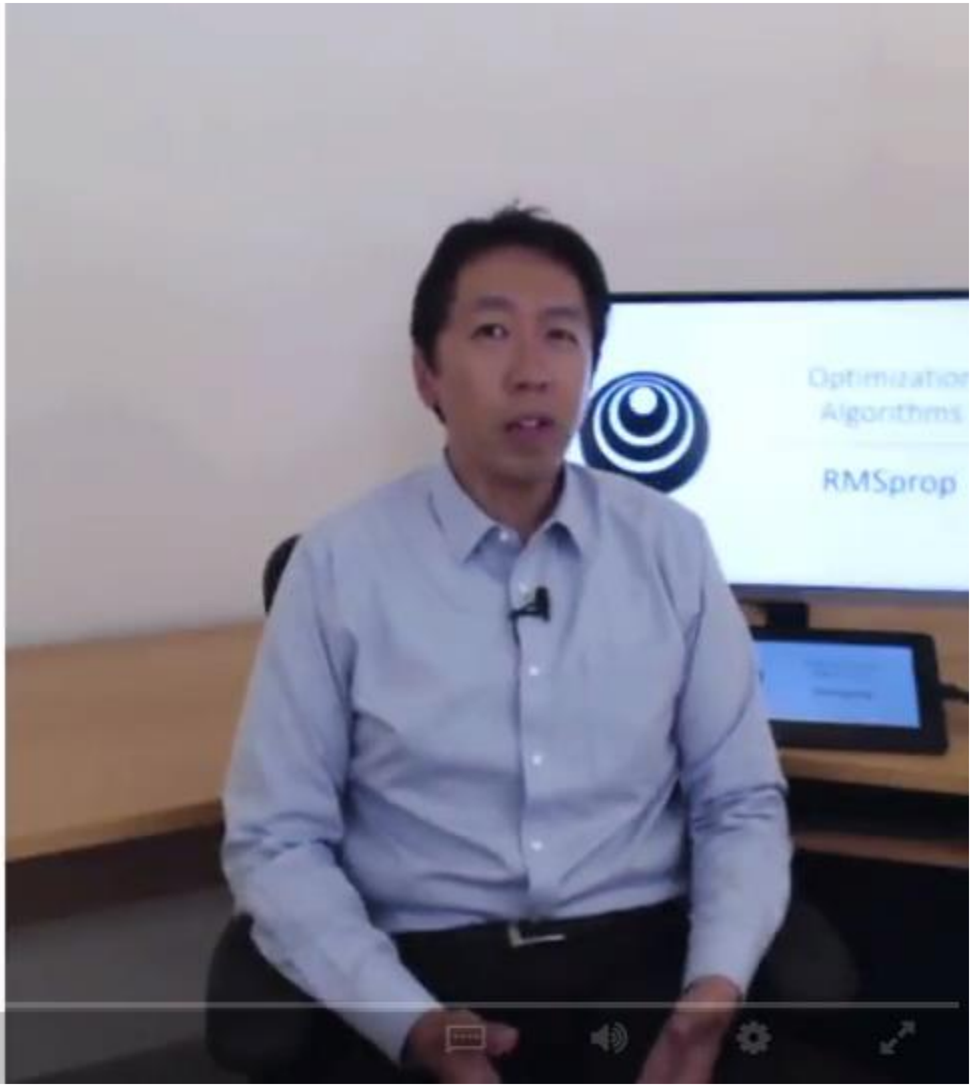
average over last ≈ 10 gradients

Andrew Ng

deeplearning.ai

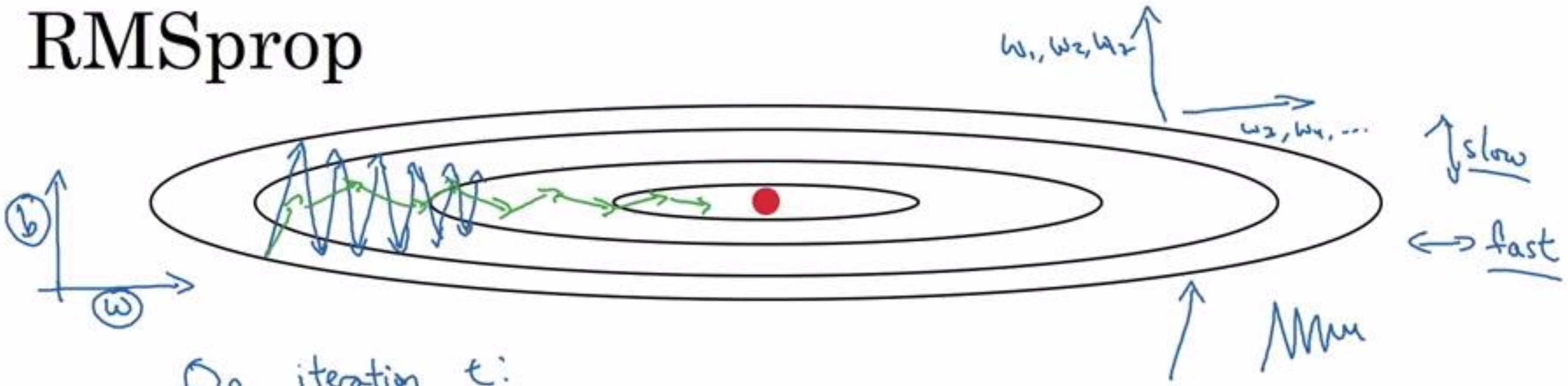
Optimization Algorithms

RMSprop



0:04 / 7:41

RMSprop



On iteration t:

Compute dW, db on current mini-batch

element-wise

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

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

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

$$\varepsilon = 10^{-8}$$

Andrew Ng

deeplearning.ai

Optimization
Algorithms

Adam optimization algorithm

Adam optimization algorithm

$V_{dw} = 0,\ S_{dw} = 0.\quad V_{db} = 0,\ S_{db} = 0$

On iteration t:

Compute dW, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dW, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db \quad \leftarrow$ "moment" β_1

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

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

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

$$W := W - \alpha \frac{V^{corrected}_{dw}}{\sqrt{S^{corrected}_{dw}} + \varepsilon} \qquad b := b - \alpha \frac{V^{corrected}_{db}}{\sqrt{S^{corrected}_{db}} + \varepsilon}$$

4:48 / 7:07

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

6:46 / 7:07

Andrew Ng

deeplearning.ai

Optimization Algorithms

Learning rate decay

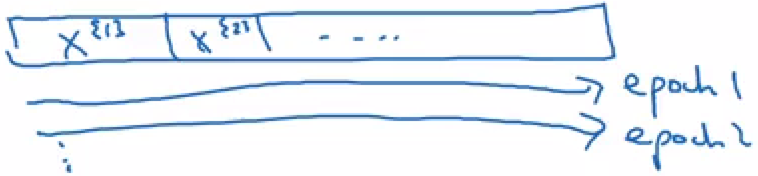
Learning rate decay

Slowly reduce α

Andrew Ng

Learning rate decay

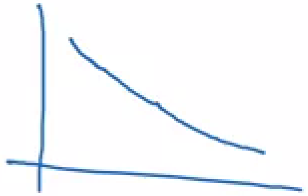
1 epoch = 1 pass through data.



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

$\alpha_0 = 0.2$
decay-rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮

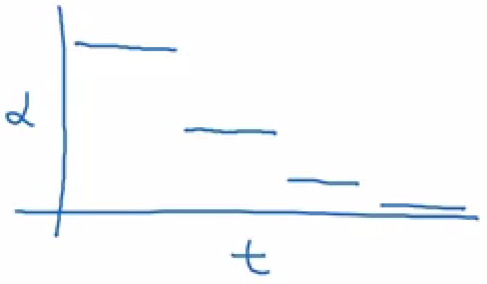


Other learning rate decay methods

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

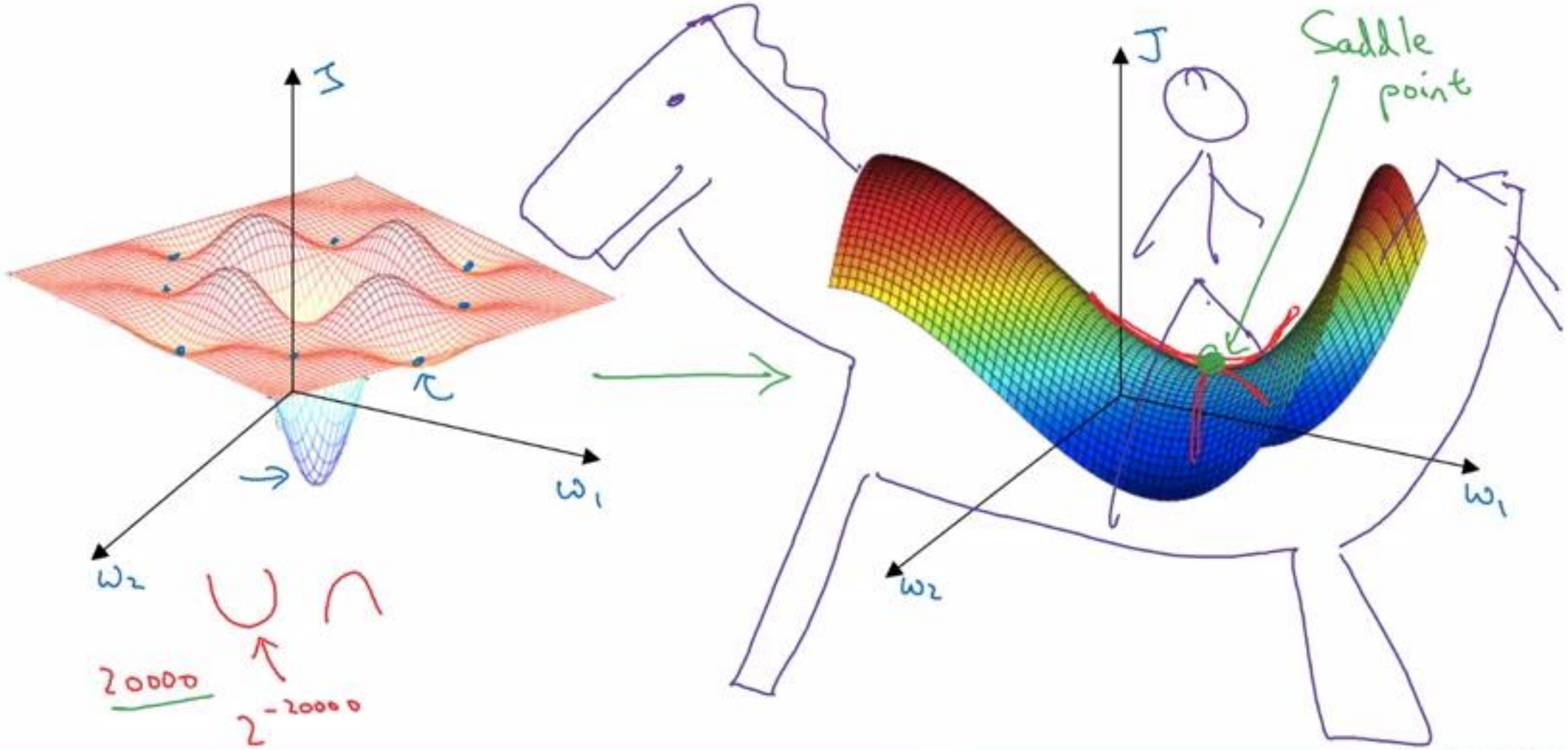
Manual decay.

deeplearning.ai

Optimization Algorithms

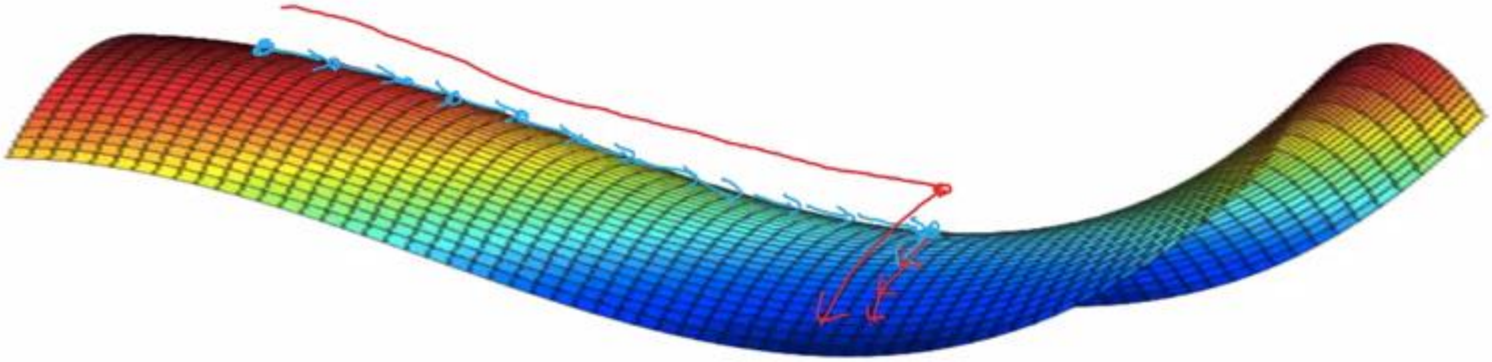
The problem of local optima

Local optima in neural networks



Andrew Ng

Problem of plateaus



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

Andrew Ng