Optimization Algorithms

Most of this material is from Prof. Andrew Ng'and Chang's slides.

Introduction

- We will learn the optimization algorithms enable you to train your neural network much faster -> training efficiency!
- Applying machine learning is a highly empirical process (i.e., highly iterative process)
  - You should train a lot of models to find one that works really well
- Deep learning works best in a regime of big data
  - We should train our model on a huge dataset
- So, having fast optimization algorithms can really help the above problems

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = \left[x^{(1)}\ x^{(2)}\ x^{(3)} \dots \qquad \dots\ x^{(m)}\right]$$

$$\underset{(1,m)}{Y} = \left[y^{(1)}\ y^{(2)}\ y^{(3)} \dots \qquad \dots\ y^{(m)}\right]$$

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = \left[x^{(1)}\ x^{(2)}\ x^{(3)} \dots \qquad \dots\ x^{(m)}\right]$$

$$\underset{(1,m)}{Y} = \left[y^{(1)}\ y^{(2)}\ y^{(3)} \dots \qquad \dots\ y^{(m)}\right]$$

What if m = 5,000,000?

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5000\}}}]$$

$$\underset{(1,m)}{Y} = [y^{(1)}\ y^{(2)}\ y^{(3)} \dots \qquad \dots y^{(m)}]$$

What if m = 5,000,000?

5,000 mini-batches of 1,000 samples

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n,1000)} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}\ (n,1000)} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5000\}}}]$$

$$\underset{(1,m)}{Y} = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5000\}}}]$$

What if m = 5,000,000?

5,000 mini-batches of 1,000 samples

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

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = [\underbrace{x^{(1)}\ x^{(2)}\ x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n,1000)} \mid \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}\ (n,1000)} \mid \dots \mid \underbrace{\dots x^{(m)}}_{X^{\{5000\}}}]$$

$$\underset{(1,m)}{Y} = [\underbrace{y^{(1)}\ y^{(2)}\ y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \mid \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \mid \dots \mid \underbrace{\dots y^{(m)}}_{Y^{\{5000\}}}]$$

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

5,000 mini-batches of 1,000 samples

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

$x^{(i)}$?

$z^{[l]}$?

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

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = [x^{(1)}\ x^{(2)}\ x^{(3)} \dots \quad \dots x^{(m)}]$$

$$\underset{(1,m)}{Y} = [y^{(1)}\ y^{(2)}\ y^{(3)} \dots \quad \dots y^{(m)}]$$

$$dW^{[i]} = \frac{\partial J}{\partial W^{[i]}} = \frac{\partial \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y},y)}{\partial W^{[i]}} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}(\hat{y},y)}{\partial W^{[i]}}$$

$$db^{[i]} = \frac{\partial J}{\partial b^{[i]}} = \frac{\partial \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y},y)}{\partial b^{[i]}} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \mathcal{L}(\hat{y},y)}{\partial b^{[i]}}$$

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

5,000 mini-batches of 1,000 samples

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

$x^{(i)}$?

$z^{[l]}$?

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

Batch vs. mini-batch gradient descent

Vectorization allows you to efficiently compute on m examples.

$$\underset{(n,m)}{X} = \Big[\underbrace{x^{(1)}\, x^{(2)}\, x^{(3)} \dots x^{(1000)}}_{X^{\{1\}}\ (n,1000)} \Big| \underbrace{x^{(1001)} \dots x^{(2000)}}_{X^{\{2\}}\ (n,1000)} \Big| \quad \dots \quad \Big| \underbrace{\dots x^{(m)}}_{X^{\{5000\}}}\Big]$$

$$\underset{(1,m)}{Y} = \Big[\underbrace{y^{(1)}\, y^{(2)}\, y^{(3)} \dots y^{(1000)}}_{Y^{\{1\}}\ (1,1000)} \Big| \underbrace{y^{(1001)} \dots y^{(2000)}}_{Y^{\{2\}}\ (1,1000)} \Big| \quad \dots \quad \Big| \underbrace{\dots y^{(m)}}_{Y^{\{5000\}}}\Big]$$

What if m = 5,000,000?

5,000 mini-batches of 1,000 samples

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

$x^{(i)}$?

$z^{[l]}$?

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

Mini-batch gradient descent

for t = 1, ..., 5000 {

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

}

Mini-batch gradient descent

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

for t = 1, ..., 5000 {

$$\left.\begin{array}{l} \text{Forward prop on } X^{\{t\}} \\ \quad Z^{[1]} = W^{[1]} X^{\{1\}} + b^{[1]} \\ \quad A^{[1]} = g^{[1]}(Z^{[1]}) \\ \qquad \vdots \\ \quad A^{[L]} = g^{[L]}(Z^{[L]}) \end{array}\right\} \text{vectorized implementation (1000 examples)}$$

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

}

Mini-batch gradient descent

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

for t = 1, ..., 5000 {

$$\left.\begin{array}{l} \text{Forward prop on } X^{\{t\}} \\ \quad Z^{[1]} = W^{[1]} X^{\{1\}} + b^{[1]} \\ \quad A^{[1]} = g^{[1]}(Z^{[1]}) \\ \qquad \vdots \\ \quad A^{[L]} = g^{[L]}(Z^{[L]}) \end{array}\right\} \text{vectorized implementation (1000 examples)}$$

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

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

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

}

Mini-batch gradient descent

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

for t = 1, ..., 5000 {

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

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

vectorized implementation (1000 examples)

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

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

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

}

One Epoch

Mini-batch gradient descent

repeat {

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

for t = 1, ..., 5000 {

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

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

vectorized implementation (1000 examples)

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

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

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

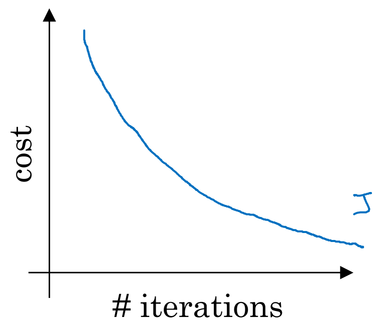
}

One Epoch

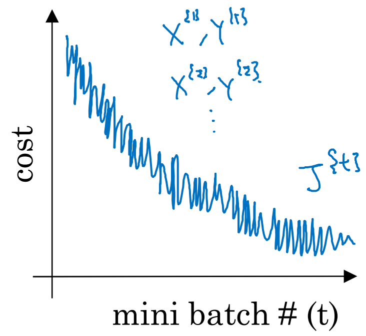
}

Training with mini-batch gradient descent

Batch
gradient descent



Mini-batch
gradient descent



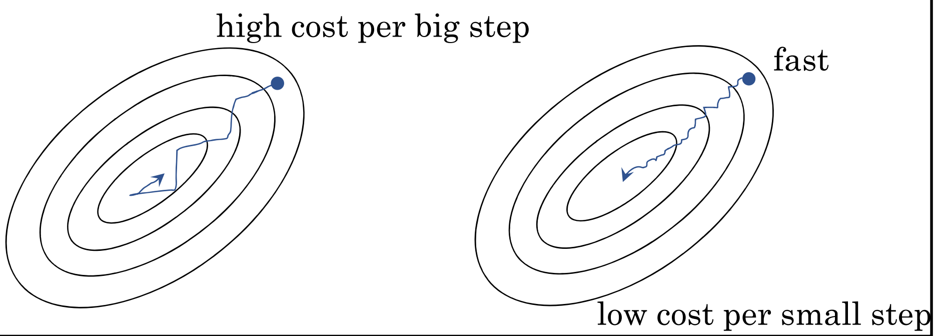
Plot $J^{\{t\}}$ computed using $\underline{X^{\{t\}}, Y^{\{t\}}}$

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:
  $(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)}), \ldots, (X^{\{m\}}, Y^{\{m\}}) = (x^{(m)}, y^{(m)})$
  → Every example is its own mini-batch

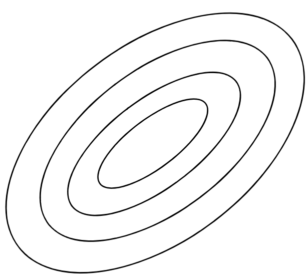
Choosing your mini-batch size

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



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:
  $(X^{\{1\}}, Y^{\{1\}}) = (x^{(1)}, y^{(1)}), \ldots, (X^{\{m\}}, Y^{\{m\}}) = (x^{(m)}, y^{(m)})$
  → Every example is its own mini-batch
- In practice, somewhere in-between 1 and m



stochastic gradient descent (mini-batch size = 1)	In-between (mini-batch size not too big or small)	batch gradient descent (mini-batch size = m)
Lose speedup from vectorization (=parallelization)	Fastest learning - Vectorization - Make progress without processing entire training set	Too long per iteration(=epoch)

Choosing your mini-batch size

- if small training set : use batch gradient descent (e.g., $m \leq 2000$)

- else, typical mini-batch size :
  $\rightarrow \underbrace{\underset{2^6}{64}, \quad \underset{2^7}{128}, \quad \underset{2^8}{256}, \quad \underset{2^9}{512}, \quad \underset{2^{10}}{1024}}_{\text{power of 2}}, \quad \ldots$

- Make sure mini-batch fits in CPU/GPU memory

Exponentially weighted averages for understanding momentum

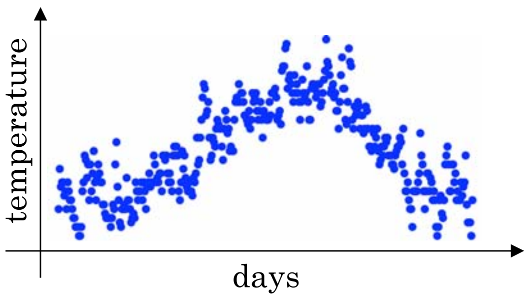
- We need to use exponentially weighted averages to understand more sophisticated optimization algorithms than gradient descent

Temperature in London

$\theta_1 = 40°\text{F}$
$\theta_2 = 49°\text{F}$
$\theta_3 = 45°\text{F}$
$\vdots$
$\theta_{180} = 60°\text{F}$
$\theta_{181} = 56°\text{F}$
$\vdots$

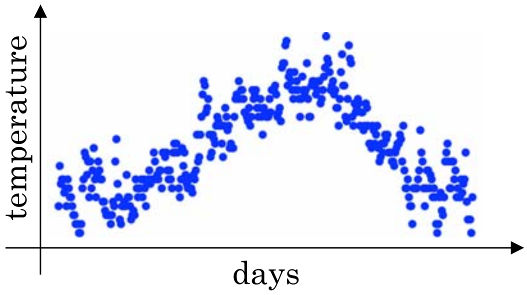
Temperature in London

$\theta_1 = 40°\text{F}$
$\theta_2 = 49°\text{F}$
$\theta_3 = 45°\text{F}$
$\vdots$
$\theta_{180} = 60°\text{F}$
$\theta_{181} = 56°\text{F}$
$\vdots$



Temperature in London

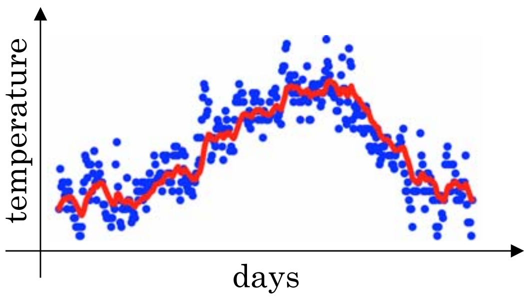
$\theta_1 = 40°F$
$\theta_2 = 49°F$
$\theta_3 = 45°F$
$\vdots$
$\theta_{180} = 60°F$
$\theta_{181} = 56°F$
$\vdots$



$v_0 = 0$
$v_1 = 0.9v_0 + 0.1\theta_1$
$v_2 = 0.9v_1 + 0.1\theta_2$
$v_3 = 0.9v_2 + 0.1\theta_3$
$\vdots$
$\boxed{v_t = 0.9v_{t-1} + 0.1\theta_t}$

Temperature in London

$\theta_1 = 40°F$
$\theta_2 = 49°F$
$\theta_3 = 45°F$
$\vdots$
$\theta_{180} = 60°F$
$\theta_{181} = 56°F$
$\vdots$

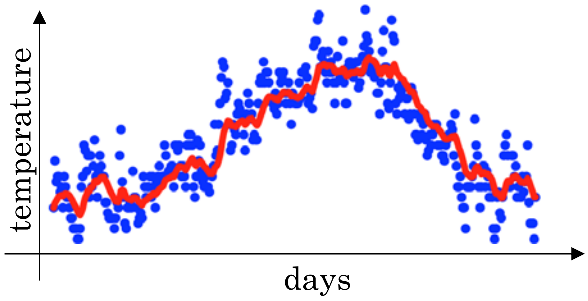


$v_0 = 0$
$v_1 = 0.9v_0 + 0.1\theta_1$
$v_2 = 0.9v_1 + 0.1\theta_2$
$v_3 = 0.9v_2 + 0.1\theta_3$
$\vdots$
$\boxed{v_t = 0.9v_{t-1} + 0.1\theta_t}$

Exponentially weighted averages

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

$\beta = 0.9$

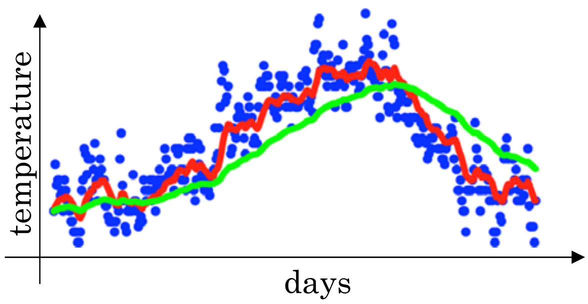


Exponentially weighted averages

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

$\beta = 0.9$

$\beta = 0.98$



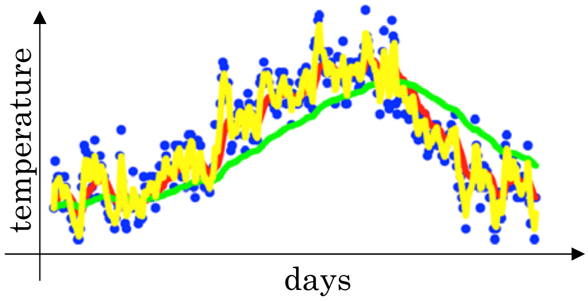
Exponentially weighted averages

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

$\beta = 0.9$

$\beta = 0.98$

$\beta = 0.5$



Exponentially weighted averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$ When $\beta = 0.9$, what is v_{100} ?

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

...

Exponentially weighted averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$ When $\beta = 0.9,$ what is v_{100} ?

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

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

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

...

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

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

Exponentially weighted averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$ When $\beta = 0.9,$ what is v_{100} ?

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

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

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

...

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

Exponentially weighted averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$ When $\beta = 0.9$, what is v_{100} ?

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

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

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

...

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

Exponentially weighted averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$ When $\beta = 0.9$, what is v_{100} ?

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

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

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

...

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

exponentially decaying weight!

$v_{100} = 0.1\theta_{100} + 0.1 \times 0.9\theta_{99} + 0.1 \times (0.9)^2\theta_{98} + 0.1 \times (0.9)^3\theta_{97} + \cdots + 0.1 \times (0.9)^{100}\theta_0$

Exponentially weighted averages

$v_t = \beta v_{t-1} + (1-\beta)\theta_t$ When $\beta = 0.9$, what is v_n ?

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

...

$$v_{100} = 0.1\theta_{100} + 0.9\big(0.1\theta_{99} + 0.9(0.1\theta_{98} + 0.9v_{97})\big)$$
$$v_{100} = 0.1\theta_{100} + 0.1 \times 0.9\theta_{99} + 0.1 \times (0.9)^2\theta_{98} + 0.1 \times (0.9)^3\theta_{97} + \cdots + 0.1 \times (0.9)^{100}\theta_0$$

$$v_n = 0.1\theta_n + 0.1 \times 0.9\theta_{n-1} + 0.1 \times (0.9)^2\theta_{n-2} + 0.1 \times (0.9)^3\theta_{n-3} + \cdots + 0.1 \times (0.9)^n\theta_0$$

$\therefore$ sum of all weights $\rightarrow 1$ if $n \rightarrow \infty$

Implementing exponentially weighted averages

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

...

Implementing exponentially weighted averages

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

$$v_\theta := 0$$
$$v_\theta := \beta v_\theta + (1-\beta)\,\theta_1$$
$$v_\theta := \beta v_\theta + (1-\beta)\,\theta_2$$
$$\vdots$$

Implementing exponentially weighted averages

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

$$v_\theta := 0$$
$$v_\theta := \beta v_\theta + (1-\beta)\,\theta_1$$
$$v_\theta := \beta v_\theta + (1-\beta)\,\theta_2$$
$$\vdots$$

$v_\theta := 0$

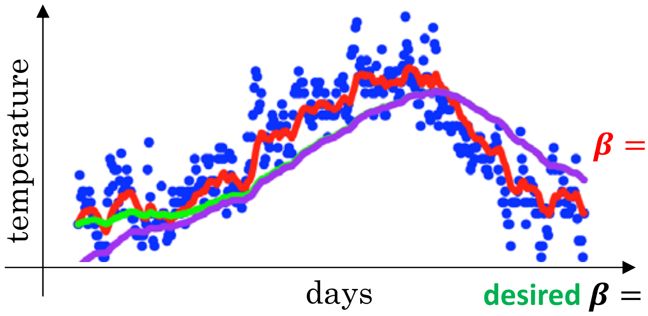
Repeat {

  Get next θ_t

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

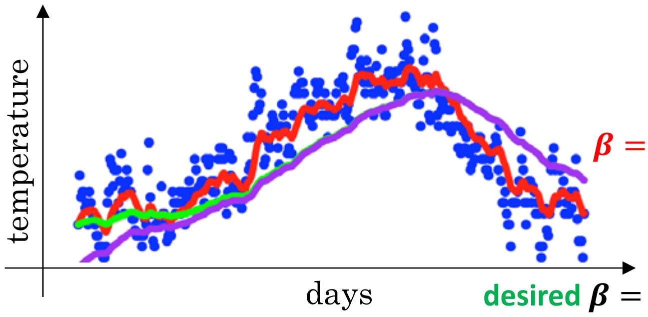
}

Bias correction



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

Bias correction

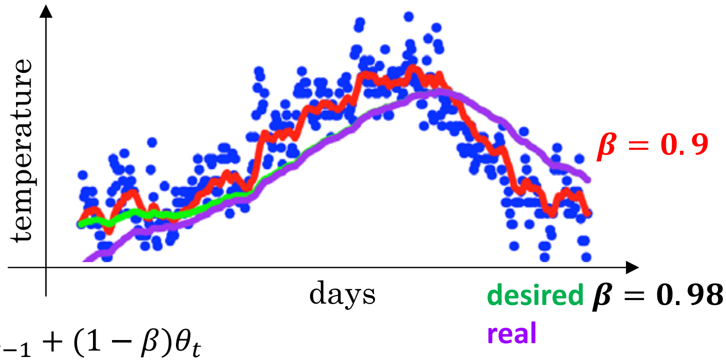


$v_t = \beta v_{t-1} + (1-\beta)\theta_t$

$v_0 = 0$

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

Bias correction



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

$v_0 = 0$

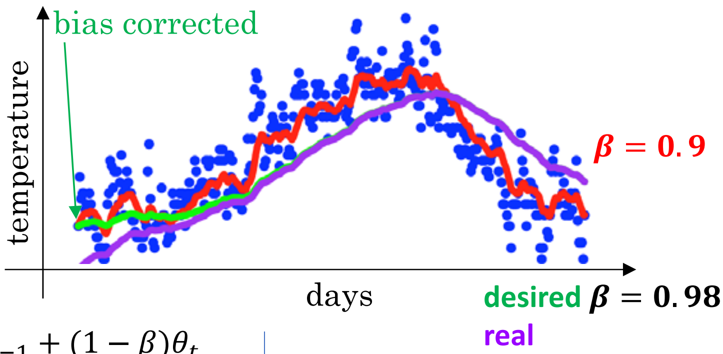
$v_1 = 0.98 v_0 + 0.02\theta_1 = 0.02\theta_1$

$v_2 = 0.98 v_1 + 0.02\theta_2$

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

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

Bias correction



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

$v_0 = 0$

$v_1 = 0.98 v_0 + 0.02\theta_1 = 0.02\theta_1$

$v_2 = 0.98 v_1 + 0.02\theta_2$

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

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

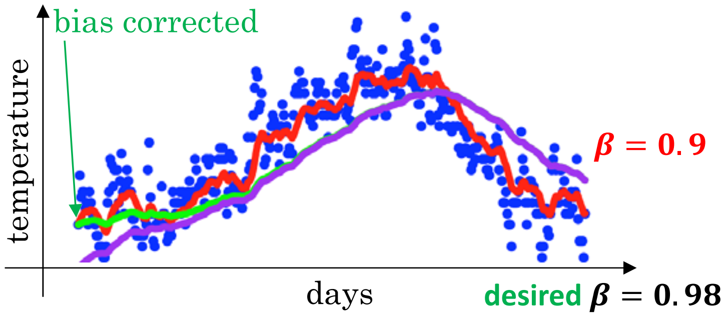
$$\boldsymbol{v_t \leftarrow \frac{v_t}{1-\beta^t}} \quad \leftarrow \text{bias correction!}$$

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

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

Bias correction

exponentially weighted average!



$$v_t = \frac{\beta v_{t-1} + (1-\beta)\theta_t}{\mathbf{1-\beta^t}} \leftarrow \text{bias correction!}$$

$$v_n = \frac{(1-\beta)\theta_n + (1-\beta)\times\beta\,\theta_{n-1} + (1-\beta)\times\beta^2\,\theta_{n-2} + \cdots + (1-\beta)\times\beta^n\,\theta_0}{\mathbf{1-\beta^n}}$$

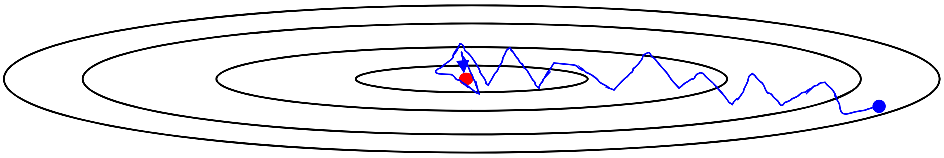
$$v_n = \frac{(1-\beta)\theta_n + (1-\beta)\times\beta\,\theta_{n-1} + (1-\beta)\times\beta^2\,\theta_{n-2} + \cdots + (1-\beta)\times\beta^n\,\theta_0}{\mathbf{(1-\beta)(1+\beta+\beta^2+\cdots+\beta^{n-1})}}$$

$$v_n = \frac{\theta_n + \beta\,\theta_{n-1} + \beta^2\,\theta_{n-2} + \cdots + \beta^n\,\theta_0^{=0}}{1+\beta+\beta^2+\cdots+\beta^{n-1}} = \frac{1\,\theta_n + \beta\,\theta_{n-1} + \beta^2\,\theta_{n-2} + \cdots + \beta^{n-1}\,\theta_1}{1+\beta+\beta^2+\cdots+\beta^{n-1}}$$

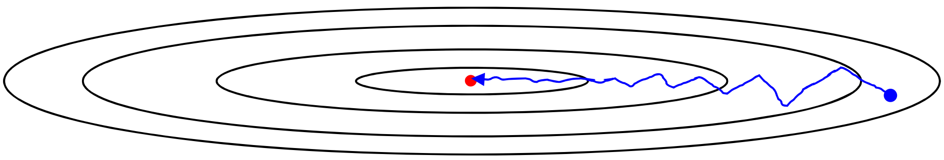
Optimization algorithms

- Gradient descent with momentum
  - almost always works faster than the standard gradient descent
  - use **exponentially weighted average of gradients**
- RMSProp
- ADAM

Gradient descent example



Gradient descent example



On iteration t :

Compute dW, db on current minibatch

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

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

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

Implementation details

On iteration t:

Compute dW, db on the current mini-batch

$v_{dW} \coloneqq \beta v_{dW} + (1-\beta)dW$ initial v_{dW}=0

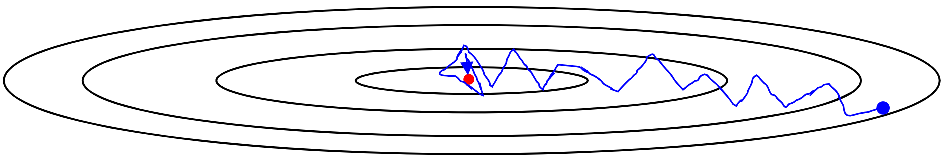
$v_{db} \coloneqq \beta v_{db} + (1-\beta)db$ initial v_{db} =0

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

In this case, usually **not use bias correction**, because initial gradients are not important.

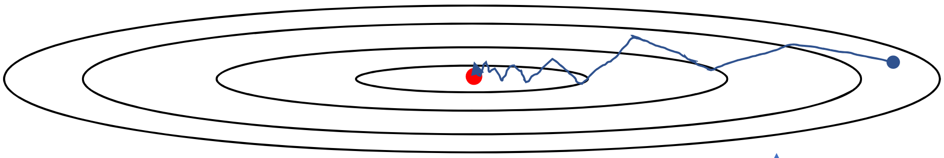
Hyperparameters: α, β $\beta = 0.9$

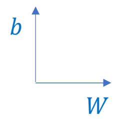
RMSProp
: root mean square propagation



oscillation of gradient descent

RMSProp





On iteration t :

Compute dW, db on current minibatch

$S_{dw} = \beta S_{dw} + (1-\beta) dW^2$ element-wise squaring

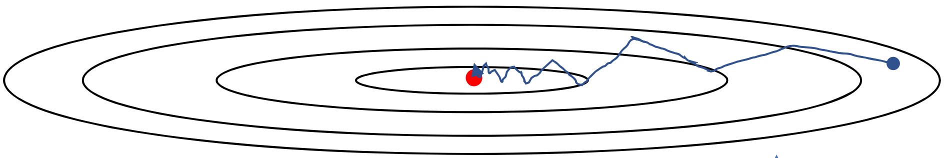
initialS_{dw}=0, initialS_{db}=0

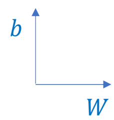
$S_{db} = \beta S_{db} + (1-\beta) db^2$

element-wise division by squared root

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

RMSProp





On iteration t :

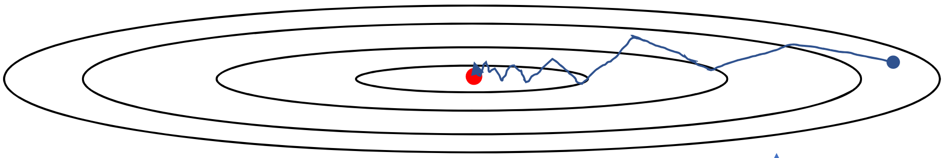
Compute dW, db on current minibatch

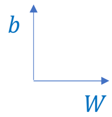
$\boxed{S_{dw} = \beta S_{dw} + (1-\beta) dW^2}$ Suppose it is 'small‘,

$\boxed{S_{db} = \beta S_{db} + (1-\beta) db^2}$ Suppose it is 'large‘,

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

RMSProp





On iteration t :

Compute dW, db on current minibatch

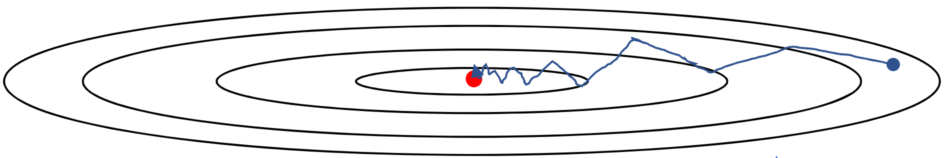
$\boxed{S_{dW} = \beta S_{dW} + (1-\beta) dW^2}$ Suppose it is 'small',

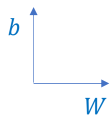
$\boxed{S_{db} = \beta S_{db} + (1-\beta) db^2}$ Suppose it is 'large',

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

then 'large' (speed-up) then 'small' (slow-down)

RMSProp





On iteration t :

Compute dW, db on current minibatch

$\boxed{S_{dW} = \beta S_{dW} + (1-\beta) dW^2}$ Suppose it is 'small',

$\boxed{S_{db} = \beta S_{db} + (1-\beta) db^2}$ Suppose it is 'large', $\epsilon = 10^{-8}$ for numerical stability

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

then 'large' (speed-up) then 'small' (slow-down)

Adam optimization algorithm

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

On iteration t :

Compute dW, db on current minibatch

$v_{dw} = \beta_1 v_{dw} + (1 - \beta_1)dW, \quad v_{db} = \beta_1 v_{db} + (1 - \beta_1)db$

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

Adam optimization algorithm

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

On iteration t :

Compute dW, db on current minibatch

"momentum" with β_1

$\boxed{v_{dw} = \beta_1 v_{dw} + (1 - \beta_1)dW, \quad v_{db} = \beta_1 v_{db} + (1 - \beta_1)db}$

$\boxed{S_{dw} = \beta_2 S_{dw} + (1 - \beta_2)dW^2, \quad S_{db} = \beta_2 S_{db} + (1 - \beta_2)db^2}$

"RMSProp" with β_2

Adam optimization algorithm

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

On iteration t :

Compute dW, db on current minibatch

"momentum" with β_1

$\boxed{v_{dW} = \beta_1 v_{dW} + (1-\beta_1)dW, \quad v_{db} = \beta_1 v_{db} + (1-\beta_1)db}$

$\boxed{S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2}$

"RMSProp" with β_2

$\boxed{v_{dW}^{\text{corrected}} = v_{dW}/(1-\beta_1^t), \quad v_{db}^{\text{corrected}} = v_{db}/(1-\beta_1^t)}$

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

bias correction

Adam optimization algorithm

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

On iteration t :

Compute dW, db on current minibatch

"momentum" with β_1

$\boxed{v_{dW} = \beta_1 v_{dW} + (1-\beta_1)dW, \quad v_{db} = \beta_1 v_{db} + (1-\beta_1)db}$

$\boxed{S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2}$

"RMSProp" with β_2

$\boxed{v_{dW}^{\text{corrected}} = v_{dW}/(1-\beta_1^t), \quad v_{db}^{\text{corrected}} = v_{db}/(1-\beta_1^t)}$

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

$$W := W - \alpha \frac{v_{dW}^{\text{corrected}}}{\sqrt{S_{dW}^{\text{corrected}}} + \epsilon} \qquad b := b - \alpha \frac{v_{db}^{\text{corrected}}}{\sqrt{S_{db}^{\text{corrected}}} + \epsilon}$$

Hyperparameters choice

α: needs to be tuned

β_1: 0.9 (weighted average for dW)

β_2: 0.999 (weighted average for dW^2)

ϵ: 10^{-8}

Adam? Adaptive momentum estimation

Hyperparameters choice

Adam Coates

α: needs to be tuned

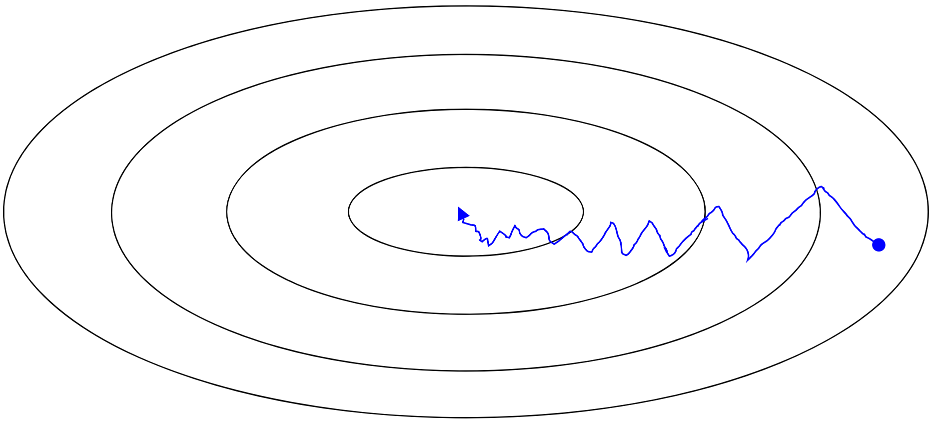
β_1: 0.9 (weighted average for dW)

β_2: 0.999 (weighted average for dW^2)

ϵ: 10^{-8}

Adam? Adaptive momentum estimation

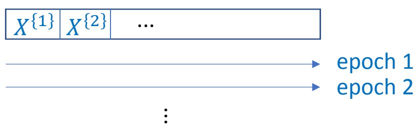
Learning rate decay



slowly reduce the learning rate over time

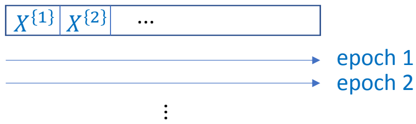
Implementing learning rate decay

1 epoch = 1 pass through data



Implementing learning rate decay

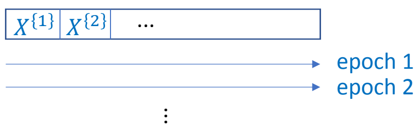
1 epoch = 1 pass through data



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

Implementing learning rate decay

1 epoch = 1 pass through data



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

$$\begin{cases} \alpha_0 = 0.2 \\ \text{decay\_rate} = 1 \end{cases}$$

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04
⋮	⋮

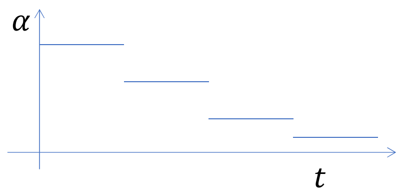
Other learning rate decay methods

Formula based

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

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

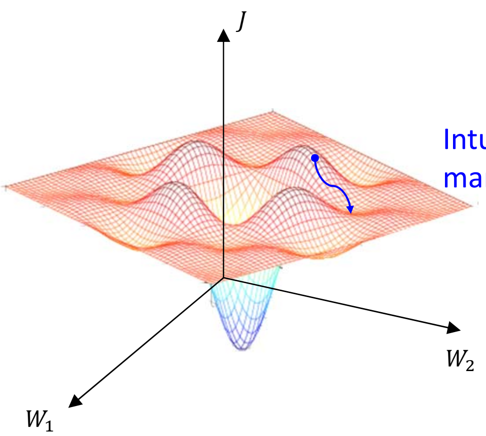
time or iteration



: discrete staircase

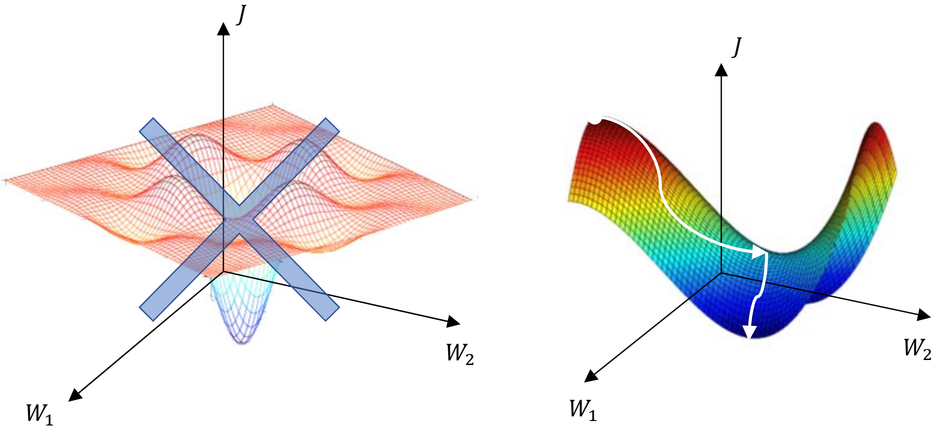
Manual decaying

Local optima in neural networks



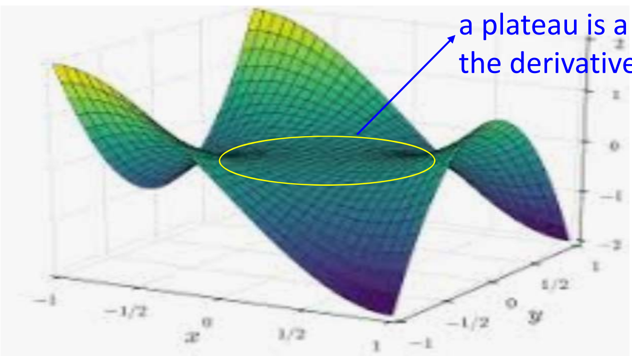
Intuition:
many bad local optima are expected

Local optima in neural networks



Real: many saddle points exists

Problem of plateaus



a plateau is a vast region where the derivative is close to zero

- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow
- momentum, rmsprop, adam optimization algorithm may be very helpful

- End -