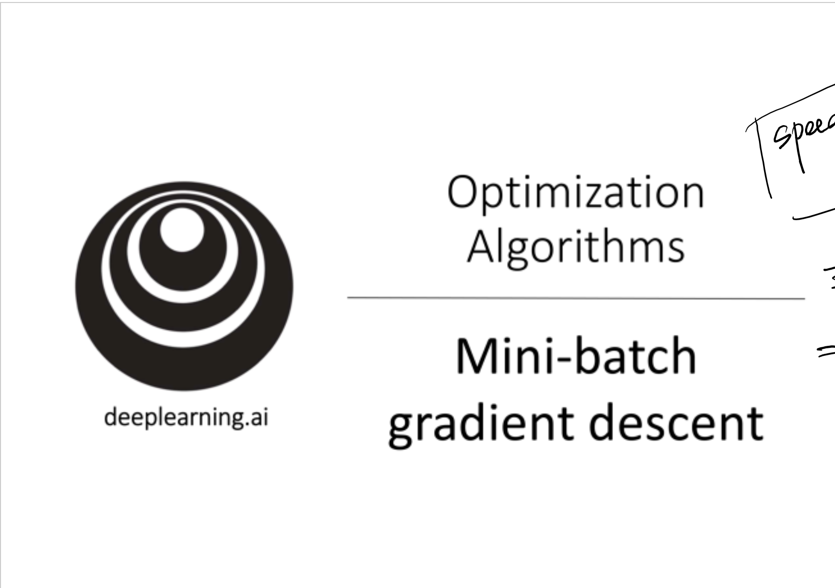
Optimization Algorithm

Monday, June 4, 2018 2:20 PM

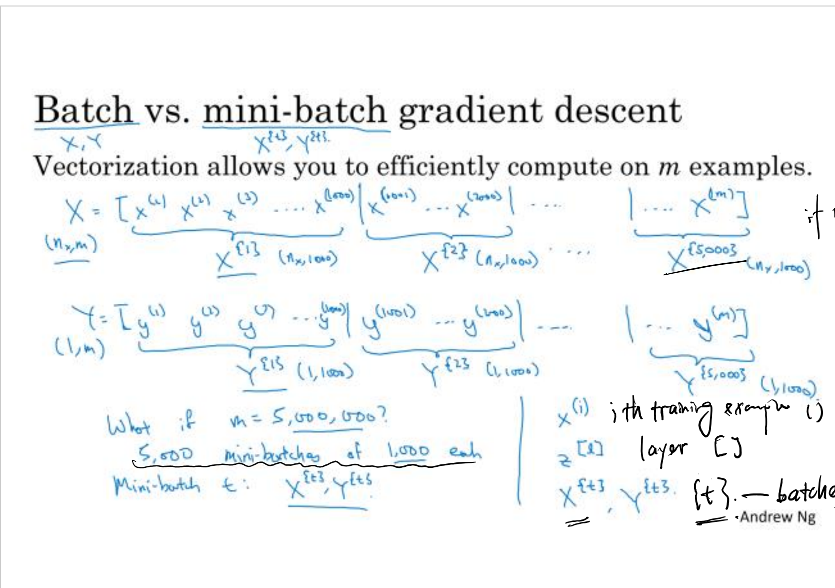
mini batch



speed up model training.

~~The~~ DL is a highly empirical subject

=> skills to speed up training



$X = [x^{(1)}\ x^{(2)}\ x^{(3)} \dots\ x^{(1000)}]$

if m is large, it can be slow.

Need to process the whole training set before gradient descent

5000 mini batches, 1000 each

ith training example ()

layer []

{t} — batches

$(n_x$, size of each batch)

Mini-batch gradient descent

\# of batches

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

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.

one pass through training set "one Epoch"

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

X, Y

vectorization to process all 1000 examples in a certain batch

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

Andrew Ng

runs much faster. Everyone will use, with large datasets

minibatch2

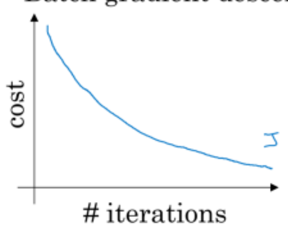
Optimization Algorithms

deeplearning.ai

Understanding mini-batch gradient descent

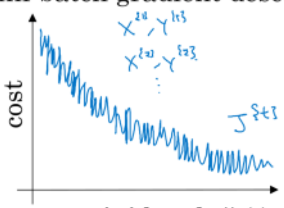
Training with mini batch gradient descent

Batch gradient descent



Always decreases (one every iteration).

Mini-batch gradient descent



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

Andrew Ng

Need not always decrease
should trend down.
but noisier!

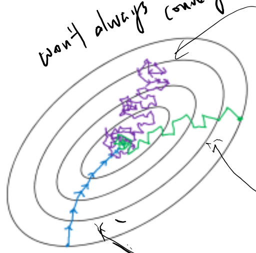
Choosing your mini-batch size

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

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

In practice: Somewhere in-between 1 and m

won't always converge



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

two extremes. 每个 batch → 全部数据
→ 一条数据.

choose something in between, not 1, not m.

Choosing your mini-batch size

If small training set: Use $\boxed{\text{batch}}$ gradient descent.
(m ≤ 2000)

Typical mini-batch sizes:

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

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

Andrew Ng

REMEMBER THESE RULES.

m. ≤ 2000. batch.

> 2000 sizes of mini-batch

因为内存. 2^n 的 batch size 更快.

(不能盲目学习).

$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
最常见.

$2^{10} = 1024$ 有点大.

Don't exceed CPU, GPU memory.

Approach: Try different power to 2. Use the most efficient one (size.)

up next: more efficient than mini-batch gd. "the most" efficient one (size.)

exp weighted average

Optimization Algorithms

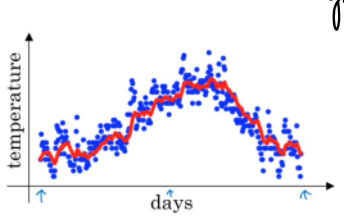
Exponentially weighted averages

deeplearning.ai

Temperature in London

get the trend of the temperature

$\theta_1 = 40°F$ 4°C ←
$\theta_2 = 49°F$ 9°C
$\theta_3 = 45°F$ ⋮
⋮
$\theta_{180} = 60°F$ 15°C
$\theta_{181} = 56°F$ ⋮
⋮



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

ARMA(1,1). ?

non-stationary 时序.

Andrew Ng

Exponentially weighted (moving) averages

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

lots of weight of the past.

$\beta = 0.9$: ≈ 10 days' temperature

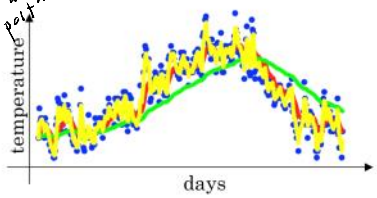
$\beta = 0.98$: ≈ 50 days

$\beta = 0.5$: ≈ 2 days

Smaller weights to the past.

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

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



Andrew Ng

Why? 能解出来的吧?

Averaging over the last $\frac{1}{1-\beta}$ days temperature.

Next what this is doing

ewa2

deeplearning.ai

Optimization Algorithms

Understanding exponentially weighted averages

red. green. — yellow.

"Just a rule of thumb of how to think about it, not mathematical derivation"

?

?

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

怎么推出结果的呢?

$\varepsilon = 1-\beta.$

$\varepsilon = 0.1$

$(1-0.02)^{\frac{1}{0.02}} = 0.98^{50} \approx \frac{1}{e}$ about $\frac{1}{3}$

to talk about later

add up to 1

= bias correction

这就是用来算Average的

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

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

}

repeat

Andrew Ng

Advantage: takes very little memory

efficient. takes one line of code.

Not the most accurate way to compute the average

ewa-bias correction

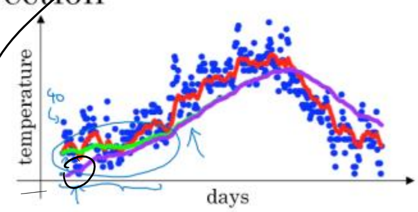
ewa-bias correction

deeplearning.ai

Optimization Algorithms

Bias correction in exponentially weighted average

Bias correction



$\beta = 0.98$

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

$V_0 = 0$

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

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

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

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

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

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

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

Andrew Ng

β large $\Rightarrow$ the initial value $v_0 = 0$ keeps the predicted value v. low for a while, not good ..

to modify this to make it better

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

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

As $t\uparrow$, $\Rightarrow \beta^t \Rightarrow 0$ this correction diminishes

only helpful for the "warm-up" period.

But people don't really bother to do this in ML

gd with momentum

Optimization
Algorithms

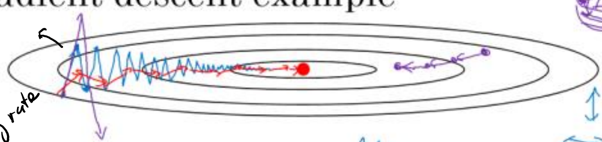
Gradient descent with momentum

→, works faster than gradient descent.

deeplearning.ai

Gradient descent example

traction βV_{dw} + $(1-\beta)\, dW$.
velocity   acceleration



• slow down learning
• can't use larger learning rate

↕ slower learning
⟷ faster learning

"~~bowl~~ ball rolling down a bowl analogy"

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

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

moving average of the derivatives of W.

What is does average out (cancel out)
vertical 抵消 cancel out
horizontal 同方向. won't cancel out.

Want: larger learning rate horizontally.

Andrew Ng

Implementation details

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

On iteration t:

Compute dW, db on the current mini-batch

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

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

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

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

in the literature. A different type of parametrization.
Difference? scale α. just affect α's choice.
this way of parametrization is less intuitive.

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

Hyperparameters: α, β

$\boxed{\beta = 0.9}$ → appear robust.
average over last ≈10 gradients

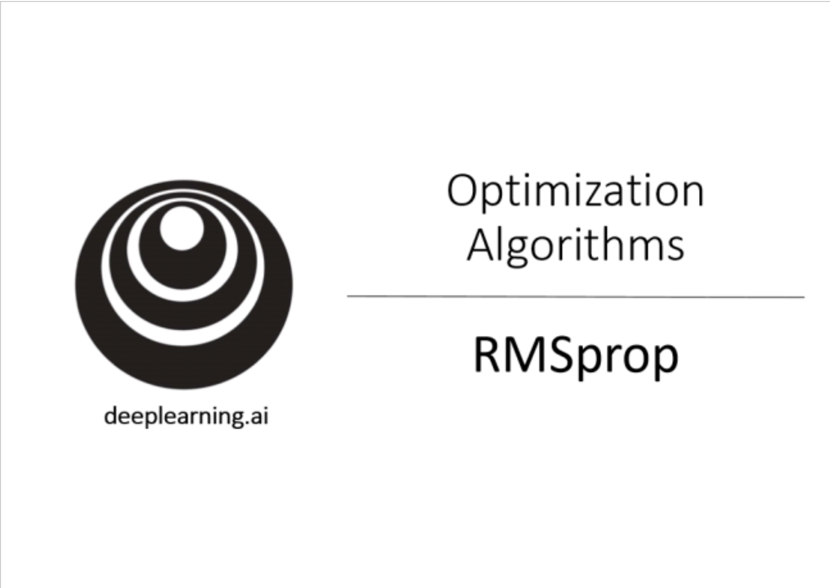
$\beta = 0.9$

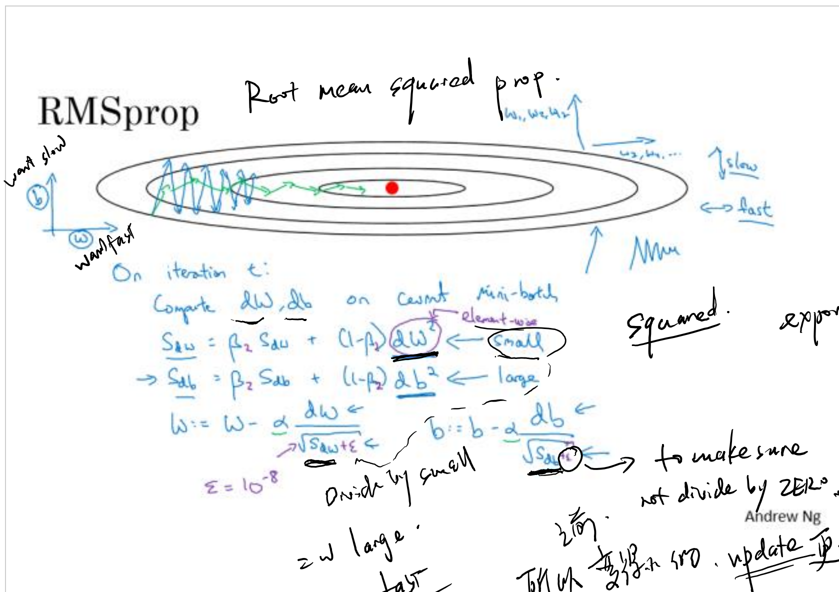
$0.9^{\frac{1}{0.1}} = 0.9^{10} = \frac{1}{e}$

bias correction? Don't bother.
after just $\boxed{10}$ iteration. the moving average would have been warmed up.
the initial value won't cause much trouble.

Andrew Ng

rmsprop





squared. exponentially weighted average of squared derivative.

In practice, high dimension
this is just a 2-dim example

→ to make sure not divide by ZERO → numerical stability

⇒ w large. ⇒ fast

之前. 所以 变得不那么 update 更快
变得不那么. update 慢. 打折.

Next video: Combine Momentum & RMSprop. ⇒ Adam optimization

adam

deeplearning.ai

Optimization Algorithms

Adam optimization algorithm

RMSprop & Adam are the two ~~of~~ optim algorithm that stand out among many new optimization algorithms

$\boxed{\text{proven effective}}$

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

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

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

bias correction

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

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

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

一锅炖 ...

Andrew Ng

Adaptive moment estimate.

Hyperparameters choice:

→ α : needs to be tune ← try a range of α to see which works

→ β_1 : 0.9 → (dw)

→ β_2 : 0.999 → (dw^2)

→ ε : 10^{-8} ← Doesn't matter much, no need to ~~tune~~ tune

真的有算 second moment 吗?

Adam: Adaptive moment estimation

Adam Coates

not related to this ~~guy~~ guy.

Andrew Ng

lr decay

lr decay

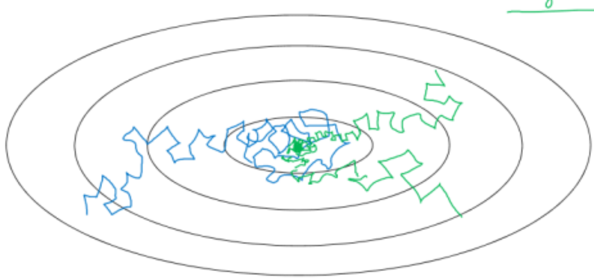
Optimization Algorithms

Learning rate decay

deeplearning.ai

Learning rate decay

Slowly reduce α



Andrew Ng

Learning rate decay

1 epoch = 1 pass through data.

$X^{\{1\}}$ | $X^{\{2\}}$ | ....

→ epoch 1

→ epoch 2

⋮

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

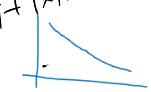
$\alpha_0 = 0.2$

decay.rate = 1

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮

first pass

$0.1 = \frac{1}{1+1 \times 1} \times 0.2$



Andrew Ng

slowly reducing learning rate

start with high LR.

then when it is close to

convergence ⇒ reduce LR.

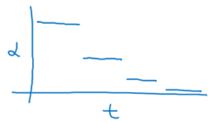
Other learning rate decay methods

看起来都好 ad hoc...

formula

$$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0 \quad \text{– exponentially decay.}$$

$$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0 \quad \text{or} \quad \frac{k}{\sqrt{t}} \cdot \alpha_0$$



discrete staircase

△ interesting. why not continuous?

Manual decay. If training only one model. works only if training only a few number of model.

Andrew Ng

Next week: how to systematically tune param.

△ LR decay is not the top priority for param tuning

不是很重要

AN.

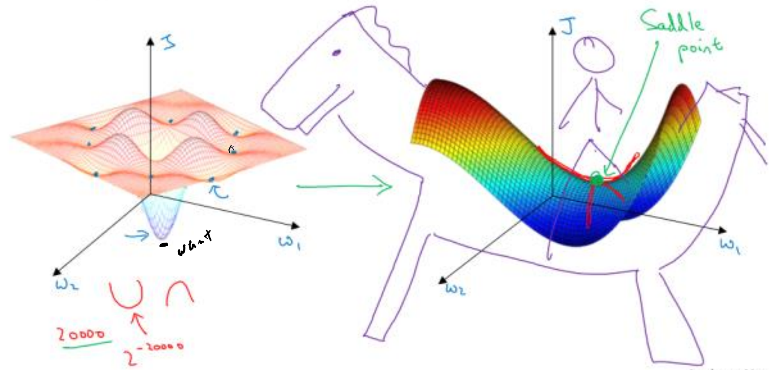
local optimal

Optimization Algorithms

The problem of local optima

deeplearning.ai

Local optima in neural networks



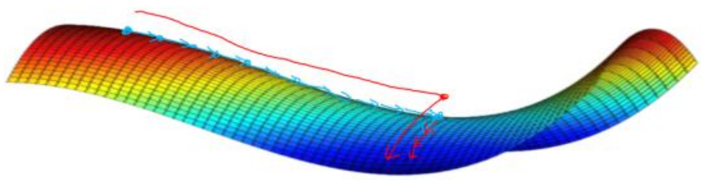
Andrew Ng

Derivative 0.

most points of zero gradient are not optimal.
They are saddle points.

* saddle points problem for high dim data

Problem of plateaus



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

Andrew Ng

$V_1^{cor} =$

$v_1 = 0.5 \times 0 + 0.5 \times 10$
$= 5$
$v_2 = 0.5 \times 5 + 0.5 \times 10 = 7.5$

plateau
slow down training
take a long time to get to optimal.

Take away:

- local optimal not much a problem for DL tasks
- plateaus slow down learning

○ key: high dim spaces.
understanding on high-dim space is still evolving.