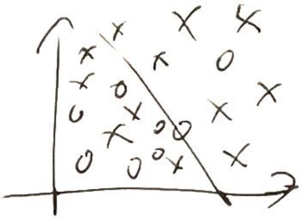
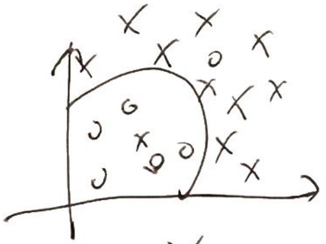
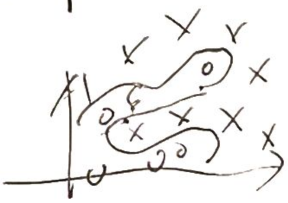
Hyperparameter tuning



high bias.



just right



high variance.
— overfitting.

Recipe:

High bias? → Bigger network.
(traing data performance)
Y Train longer
(NN architecture search)

↓ N

High variance → More data
Y Regularization
(NN arch. search)

Train set error 1%
Dev set error 11%
high varian

not doing very well on traing set. ← { TE 15% / DE 16% } high bias.

" Bias variance tradeoff "
↑
mondern DNN can reduces both when bigger network & more data.

Regularization:

$$\min_{a,b} J(a,b) \qquad J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

single number b ↓ omit

$w \in \mathbb{R}^{n_x}$ $\quad L_2$ Regular.... : $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

$b \in \mathbb{R}$ $\quad \mathcal{L}_1$ Regu: $\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$ norm

l_1 Regn → sparse w (a lot of 0 in)
easy to store.

l_2 Regn used much more often.

λ → Regularization parameter.
lambd – python

Neural network:

$$\boxed{\begin{aligned} J(w^{[1]}, b^{[1]}, \dots, w^{[l]}, b^{[l]}) &= \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{l} \|w^{[l]}\|^2 \\ \|w^{[l]}\|^2 &= \sum_{i=1}^{n^{[l-1]}}\sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2 \end{aligned}}$$

$w \to \in \mathbb{R}^{(n^{[l]}, n^{[l-1]})}$

"Frobenius norm"

dw = (from back prop)

$w^{[l]} := w^{[l]} - \alpha dw^{[l]}$

$dw = (\dots) + \frac{\lambda}{m} w^{[l]}$

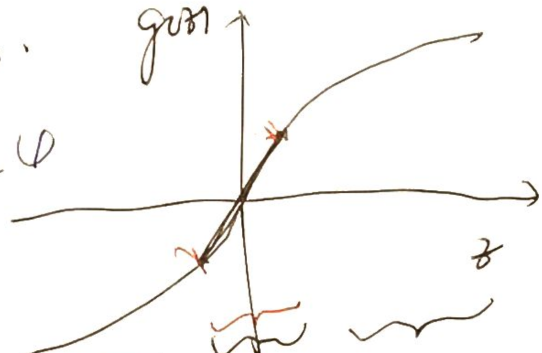
weight decay

$w^{[l]} := w^{[l]} - \alpha \left[(\dots) + \frac{\lambda}{m} w^{[l]} \right]$

$= w^{[l]}\left(1 - \frac{\alpha\lambda}{m}\right) - \alpha(\dots)$

Reasons for using Regularisati

①



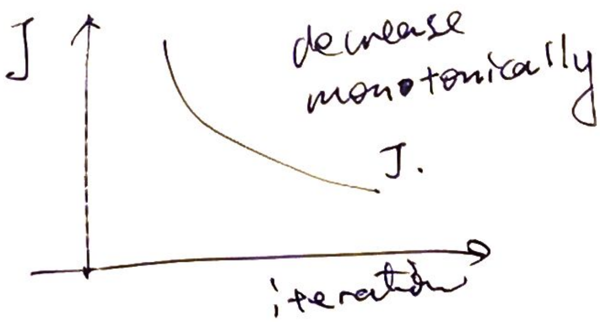
less linear   more linear   less linear

$\lambda \uparrow \to w^{[l]} \downarrow$
penalised.

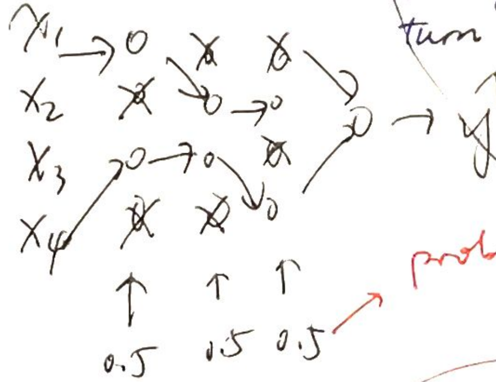
$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$.
small ← small

if z is small then
$g(z)$ could be roughly
linear →

② avoid overfit



Dropout Regularization



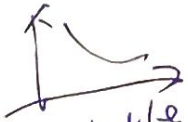
probability

Debug
turn off ..
J-iterat
graph

Drop out node randomly.

Smaller network
→ less overfit.

makes



unstable.

eg: Inverted Dropout.

Expected value of $a^{[3]}$ remains

d3 = np.random.~~randn~~ rand ?

→ layer 3.

(a3.shape[0], a3.shape[1])

< keep-prob. has a 0.8 chance of being True
& a 0.2 chance of being 0.

a3 = a3 * = d_3 → boolean array.

or (np. multiply)

a3 /= keep-prob.

$$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}$$

reduced by 20%

bump back /= 0.8

Making predictions at test.

do not use Dropout

→ variety of results

→ add uncertainty / noise

{ L2 shrink weight
drop out → spread up weight

also similar effect as L2.

any feature could go away
& current node using
dropout can be independent

motivated by spread out the weight.

keep-prob could vary in different layer.

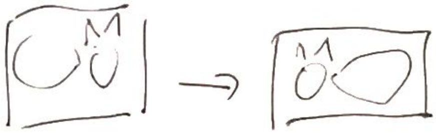
eg: Big layer could cause overfitting → smaller keep-p.

Some layer → 1.0
use very high value ~~for~~ at input layer

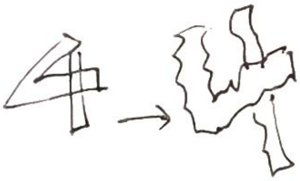
eg: Computer vision → overfitting
large pixel ← successful

Data augmentation

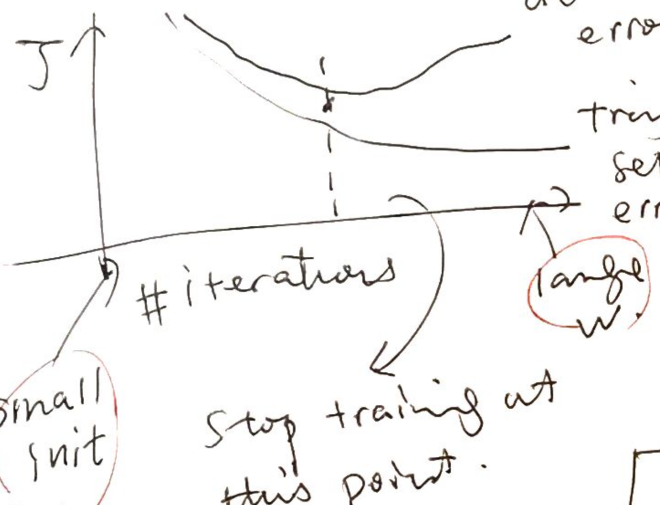
image → flipping, distortion



inexpensive way for extra data.



Early Stopping



Stop training at this point.

W gets bigger overfitting less.

pros: run once & try out the different size of W.

Orthogonalization

Coupled
- Optimize Cost fun J.
  - Grad Des.; momentum
- Not overfit
  - Regu..

Regularisation

cons

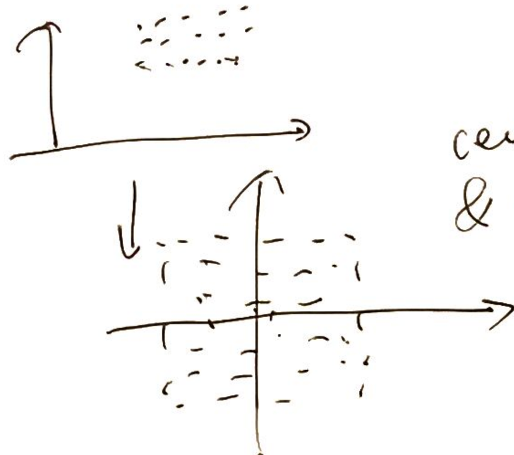
J not optimised. mixed two problems. → still trying to avoid overfit

use L_2 Regu

And train as long as possible → easier to decompose hyper parameters.

cons: try out λ computationaly expensive

Normalizing training data



centralised & augmented set.

zero out mean number.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$X := X - \mu$$

Normalize variance

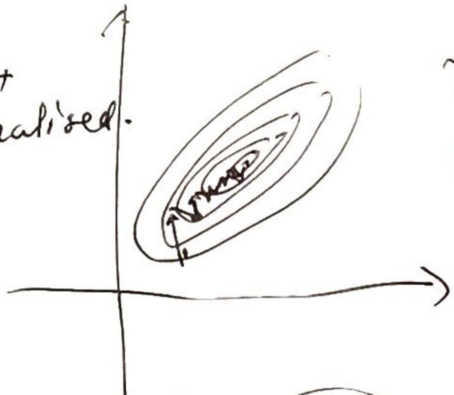
$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} X^{(i)} ** 2.$$

$$X :/= \sigma^2$$

Reasons for normalising

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

not normalised.



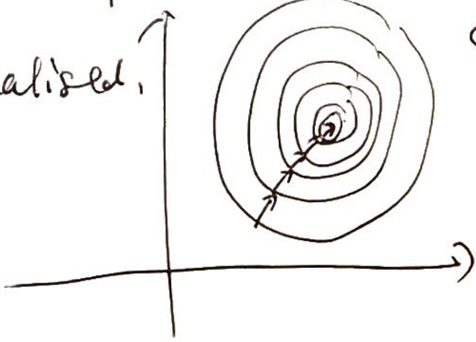
$x_1 : 1 \cdots 1000$

$x_2 : 0 \cdots 1$

range differ

more symmetric

normalised.



normalise test set with same μ, & σ^2 paras.

Vanishing gradient.
exploding

For deep neural network. DNN

$w^{[l]} > I \to$ exponential explodes.

$w^{[l]} < I$

$$\hat{y} = (w^{[l]})^{(L-1)} x$$

Partial solution:

Initialise, assume ReLU

$\frac{2}{n}$ for ReLU

more $n \uparrow \to$ less weight

$w_i \downarrow$ wanted.

$$Var(w_i) = \frac{2}{n}$$

$$W^{[l]} = np.random.randn(shape) * np.sqrt(\frac{1}{n^{[l-1]}})$$

$$\tanh \to \sqrt{\frac{1}{n^{[l-1]}}}$$

? reason

Xavier initialisation

$$\sqrt{\frac{2}{n^{[l-1]} + n^{[l]}}}$$

tuning choices.

Gradient check for NN

Concatenate $W^{[1]}, b^{[1]} \dots W^{[L]}, b^{[L]}$ into single vector θ.

* $J(W^{[1]}, b^{[1]} \dots) = J(\theta)$

Take $dW^{[1]}, db^{[1]} \dots dW^{[l]}, db^{[l]}$ and reshape it into a big vector $d\theta$.

* $J(dW^{[1]} db^{[1]} \dots dW^{[l]} db^{[l]}) = J(\theta)$

$J(\theta) = J(\theta_1 \theta_2 \dots \theta_m)$

For each i: (component of θ)

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2 \dots \theta_i + \varepsilon \dots) - J(\theta_1, \theta_2 \dots \theta_i - \varepsilon \dots)}{2\varepsilon}$$

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

check euclidean distance

$$\boxed{\frac{\|d\theta_{app} - d\theta\|_2}{\|d\theta_{app}\|_2 + \|d\theta\|_2}} < e^{-7}$$

$\varepsilon = 10^{-7}$ valid.

if $\begin{cases} 10e^{-5} \to \text{maybe components too large.} \\ 10e^{-3} \to \text{highly likely bug.} \end{cases}$

Practical implementation notes:

(1) Do not use in training — only to debug.

$d\theta_{approx}[i]$

→ slow computationally expensive.

only when debug.

(5) Run at random initialization; again after some training

$w, b \approx 0 \to w, b \uparrow$ debug. could error

(2) Turn to different components to identify bug.

$db^{[l]}, dW^{[l]}$.

(3) $J(\theta) = \frac{1}{m} \sum_i L(\hat{y}^{(i)}, y^{(i)})$

Remember Regularization term.

(4) Doesn't work with dropout. (turn off)

Mini-batch gradient descent

Vec → m examples.

$$\begin{cases} X = [x^{(1)} x^{(2)} \cdots x^{(m)}] & (n_x, m) \\ Y = [y^{(1)} y^{(2)} \cdots y^{(m)}] & (1, m). \end{cases}$$

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

have to go through (process) the entire training set before the next step

— slow

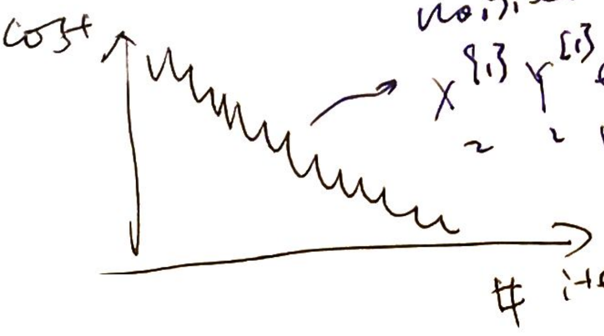
mini batches of 1,000 each.

$$[\underbrace{x^{(1)} \cdots x^{(1000)}}_{\boxed{X^{\{1\}}}} \cdots \cdots \underbrace{| \cdots x^{(m)}}_{\boxed{X^{\{5000\}}}}]$$

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

$(n_x, 1000)$ $(1, 1000)$.

$x^{(i)}$, $z^{[l]}$.



noisier.

$X^{\{1\}} Y^{\{1\}}$ easy mini batch

2 2 harder ....

(mislabled examples)

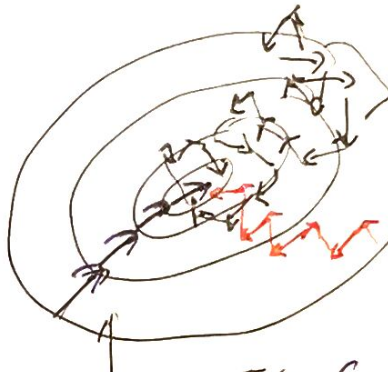
mini-batch size = m :

batch gradient descent

mini-batch size = 1 :

Stochastical GD.

loss full velocity from vectoritization



mini batch → noisy

diverge

~~mini batch~~ GD

Too long per iteration

mini batch trade off

$m \le 2000$

→ BD

Typical mini-batch size

64. 128. 256. 512

2^6 2^7 2^8 2^9

Exponentially Weighted Average (moving).

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

$= (1-\beta)\theta_t + \beta((1-\beta)\theta_{t-1} + \beta v_{t-2})$

$= (1-\beta)\theta_t + \beta(1-\beta)\theta_{t-1} + \beta^2(1-\beta)\theta_{t-2}$

$+ \dots$ takes about 10 days for value to decay to $1/e$.

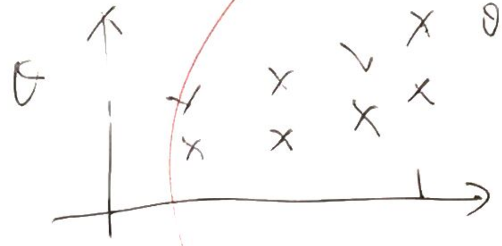
$\beta\uparrow$ plot is much smoother

averaging over more days of temp.

curve shift to right. larger window of temperat

Adapts slowly.

$\theta_t, \theta_{t-1} \dots \theta_1$ *.

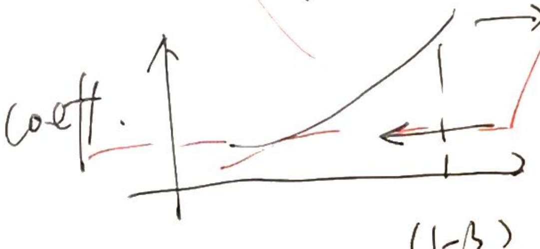


*

This shape. average value (mean) ~~for v~~ is about 1/3

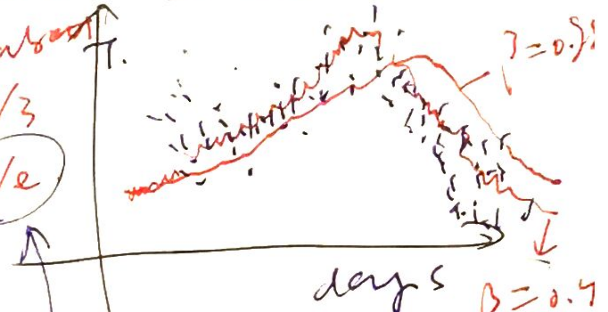
$0 < \beta < 1$

$\beta^{t}(1-\beta)$ $1/e$



exponentially decay

$(1-\beta)$



reason.

$\beta\uparrow \to (1-\beta)\downarrow$.

adapt slowly to current θ_t.

stronger simulation with former v_t.

$\longrightarrow$ smoother

$(1-\varepsilon)^{1/\varepsilon} = \frac{1}{e}$ (0.1)

$\varepsilon = 1-\beta$

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

$0.98^{50} \approx \frac{1}{e}$

$\approx \frac{1}{1-\beta}$ days average.

temperature is v_t.

approximately

$\beta = 0.9 \Rightarrow$ v_t average over the last 10 days of temperature.

$\beta = 0.98 \to 50$. $\beta = 0.5 \to$ average is over 2 days.

more noisy adapt much quicker

To implement.

just iteratively rewrite

the $V_\theta = \beta V + (1-\beta)\theta_t$,

Just a course (rough) estimation

⟶ accurate mean requires storage of previous X days of temperature whereas this method only takes last value.

* { efficiency
  computationally cheap ✓

Bias correction

initialise with $V_0 = 0$.

which influence the performance to produce averaging. unavoidable

During initial phrase: bad start

$$\begin{cases} V_0 = 0 \\ V_1 = 0.98 V_0 + 0.02\theta_1 \\ V_2 = 0.98 V_1 + 0.02\theta_2 \end{cases}$$

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

mini batch

$t=2$: $1-\beta^t = 1-(0.98)^t = 0.396$

$$\frac{V_2}{0.396} = \frac{0.196\theta_1 + 0.02\theta_2}{0.0396} \approx \text{averages}$$

$t\uparrow \longrightarrow$ no effect on $(1-\beta^t) \sim 1$

Smooth out the steps of GD

Gradient descent oscilates a lot & this prevents a large learing rate $-\alpha$. as it may over shoot diveraging.

momentum $= (mV)$.

slower learing ↑

faster lering ⟶

large β. more damp to neutralise ~~the~~ oscillation

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

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

$$\begin{cases} W = W - \alpha V_{dw} \\ b = b - \alpha V_{db} \end{cases}$$

average ↑↓↑↓↑ out the oscillation

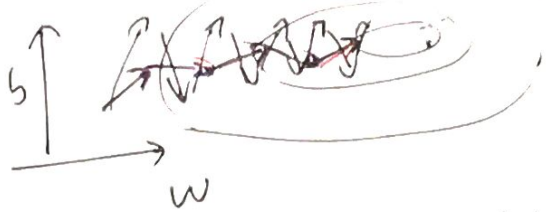
{ vertically slow down
  horizontally — no big influence.

dw → accelaration

W → velocity

β → friction.

RMSprop: Root Mean Square)



$w_2 w_5 w_{18}\ldots$ high dimensional space tuning $w_3 w_4 \ldots w_{17}$

essentially

want to speed up at w.
slow down at b.

On minibatch t:

Compute dw. db ....

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

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

$$w: w - \alpha \frac{dw}{\sqrt{S_{dw}}}, \quad b: = b - \alpha \frac{db}{\sqrt{S_{db}}}.$$

damp out oscillation

slope is very large on b direction $db \to$ large.
$\to S_{db}$ ~~large~~. $\to$ b updates vertically more slowly.

Damp out ~~those~~ many parameters with large slope.

in case the denometer is too small that leads to exponential explosion: add Epsilon

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

$\boxed{\text{ADAM}}$ *.

α = needs to be tuned

β_1 : 0.9 (dw)

β_2 : 0.999 (dw^2)

$\varepsilon : 10^{-8}$

For iteration t: $\to$ each mini batch.

$$\begin{cases} V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw \\ V_{db} = \beta_1 V_{db} + (1-\beta_1) db \end{cases}$$

$$\begin{cases} S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2 \\ S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2 \end{cases}$$

$\Downarrow$

~~Combine~~. correct (bias correction)

$$V_{dw}^{co} = V_{dw}/(1-\beta_1^t)$$

$$V_{db}^{co} = V_{db}/(1-\beta_1^t)$$

$$S_{dw} = S_{dw}/(1-\beta_2^t)$$

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

Combine momentum & RMSprop.

$$\begin{cases} W: = w - \alpha \dfrac{V_{dw}}{\sqrt{S_{dw}} + \varepsilon} \\ b: = b - \alpha \dfrac{V_{db}}{\sqrt{S_{db}} + \varepsilon} \end{cases}$$

Mini Batches won't converge.

oscillates in a tighter region.

oscillate around the optimised point & may not converge.

Learning rate decay.

1 epoch = 1 pass through data

$X^{\{1\}}$ $X^{\{2\}}$ - - - - - $X^{\{n\}}$

→ epoch 1

→ epoch 2.

set $\alpha = \frac{1}{1 + \text{decayrate} * \text{epoch\_num}} \alpha_0$

2 for tune

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4
⋮	⋮

↓ gradually decrease.

$\alpha = 0.2$

decay_rate = 1

③ local optimal: n dimensional space.

need ∪ ∩

convex concave.

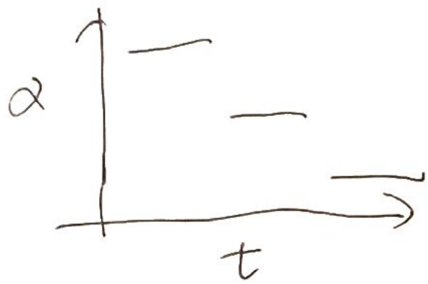
$P(2\cup) = 2^{n}$

2∪ → impossible nearly

$\alpha = 0.95^{ep} \cdot \alpha_0$

$\alpha = \frac{k}{\sqrt{ep}} \cdot \alpha_0$

or $\frac{k}{\sqrt{t}} \cdot \alpha_0$



discreate staircase

OR manual decay.

→ for small number of models. training

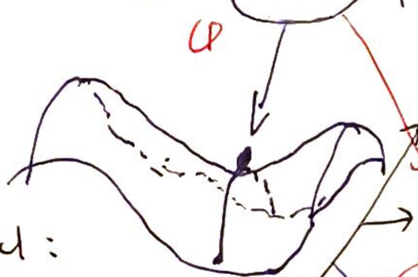
local optimal

advanced perceivation

most zero gradients are saddle points.

LP

n is high then more common



→ ∪+∩

② plateau

around zero gradient for a long time