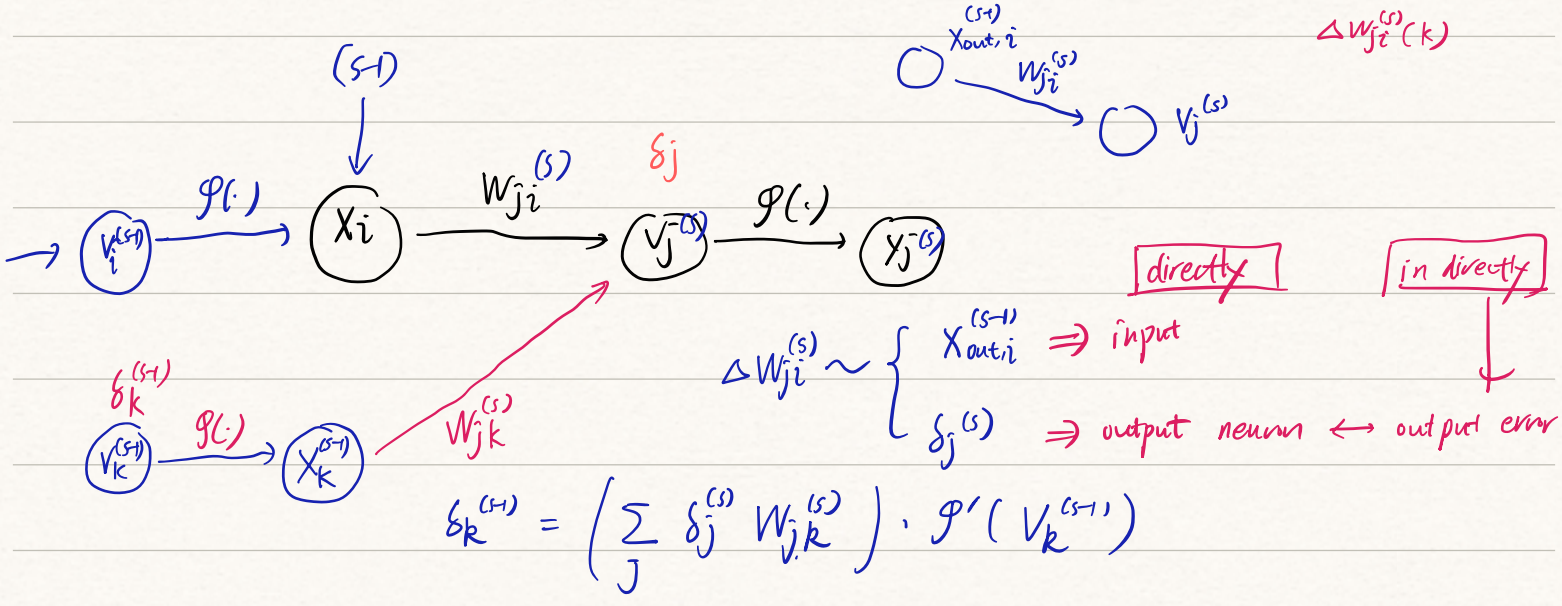
EE5904. lecture 4.

GD Method

$W(k+1) = W(k) - \eta g(k)$

- Single layer perceptron: $W(k+1) = W(k) + \eta\, e(k)\, X(k)$ (output error; input signal)
- MLP: BP $\rightarrow$ $W_{ji}^{(s)}(k+1) = W_{ji}^{(s)}(k) + \eta^{(s)} \delta_j^{(s)} \cdot X_{out,i}^{(s-1)}$

$\Delta W_{ji}^{(s)}(k)$



directly / in directly

$\Delta W_{ji}^{(s)} \sim \begin{cases} X_{out,i}^{(s-1)} \Rightarrow \text{input} \\ \delta_j^{(s)} \Rightarrow \text{output neuron} \leftrightarrow \text{output error} \end{cases}$

$$\delta_k^{(s-1)} = \left(\sum_j \delta_j^{(s)} W_{jk}^{(s)} \right) \cdot \varphi'\left(V_k^{(s-1)}\right)$$

Algorithm Framework

① Initialization $W_{ij}^{(s)}(0)$

② use $W_{ij}^{(s)}(k)$ $k = 0, 1, 2, \ldots$

to compute $X_{out,i}^{(s)} \xrightarrow{W_{ki}^{(s+1)}} V_k^{(s+1)} \xrightarrow{\varphi(\cdot)} X_{out,k}^{(s+1)} \rightarrow \cdots\cdots$

↓ forward computation

③ In step 2, we attain $\begin{bmatrix} \text{output}(k) \\ d(k) \end{bmatrix}$

$\Rightarrow$ error $e(k) = d(k) - \text{output}(k)$

$\|$

optimization objective $E(k) = \frac{1}{2}\|\underline{e}(k)\|_2^2$

update process

$$\underline{W}(k+1) = \underline{W}(k) - \eta \nabla_{\underline{w}} E(k)\Big|_{\underline{w}=\underline{w}(k)}$$

$$W_{ji}^{(s)}(k+1) = W_{ji}^{(s)}(k) - \eta^{(s)} \frac{\partial E(k)}{\partial W_{ji}^{(s)}}\Big|_{W_{ji}^{(s)} = W_{ji}^{(s)}(k)}$$

step 1 the output layer

$$= W_{ji}^{(s)}(k) + \eta^{(s)} \delta_j^{(s)}(k) x_{out,i}^{(s-1)}(k)$$

$$\delta_j^{(e)}(k) = e_j(k) \cdot \varphi'(v_j^{(e)}(k))$$

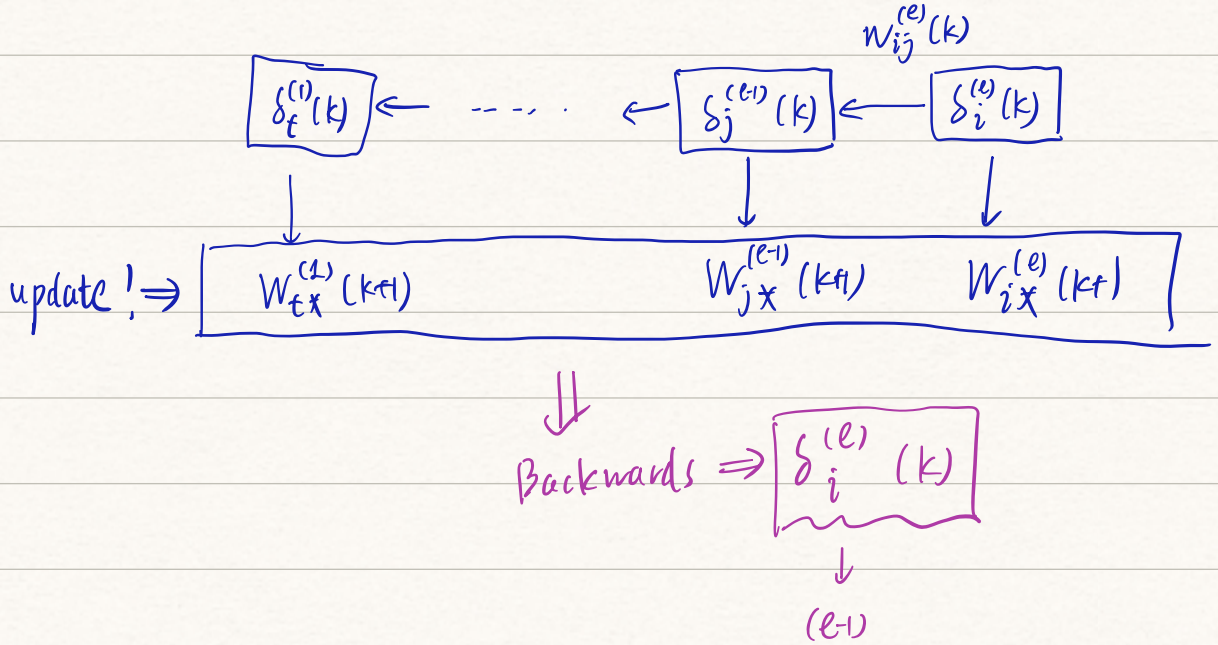
step 2: $\delta_i^{(e-1)}(k) = \left(\sum_j \delta_j^{(e)}(k) W_{ji}^{(e)}(k)\right) \varphi'(v_i^{(e-1)}(k))$

$\vdots$

$\delta_i^{(1)}(k) = \cdots$

step 3: update with

$$W_{ji}^{(s)}(k+1) = W_{ji}^{(s)}(k) + \eta^{(s)} \delta_j^{(s)}(k) x_{out,i}^{(s-1)}(k)$$



↓
⋮
↓
(1)

Some limits!

⇒ the cost surface is typically non-quadratic & non-convex & high-dimensional

→ { convergence slow
  not converge
  converge to a bad sol^n

→ How to construct a NN to avoid so many questions?

① How to use data?

{ Sequential ⇒ online
  Batch

1. sequential

$$\Delta w_{ij}(1) = -\eta \left.\frac{\partial E(1)}{\partial w_{ij}}\right|_{w_{ij}=w_{ij}(1)}$$

→ $w_{ij}(2) = w_{ij}(1) + \Delta w_{ij}(1)$

→ $\Delta w_{ij}(2) = -\eta \frac{\partial E(2)}{\partial w_{ij}(2)}$

→ $w_{ij}(3)$

$\underline{x}(1), \underline{d}(1) \Rightarrow \underline{x}(2), \underline{d}(2) \Rightarrow \cdots \Rightarrow \underline{x}(N), \underline{d}(N)$

one epoch

↓

change the sequence

事实上这不是 GD Method

↓

objective always change

2. Batch

$(\underline{x}(1), \underline{d}(1)), \ldots, (\underline{x}(N), \underline{d}(N))$

↓ ↓

$E(1)$ ... $E(N)$

$$E_{av} = \frac{1}{N}\sum_{i=1}^{N} E(i) \rightarrow \text{one epoch}$$

↓

Steepest descent $\quad \Delta w_{ij} = -\eta \left.\frac{\partial E_{av}}{\partial w_{ij}}\right|_{w_{ij}=w_{ij}(s)}$

⇓

standard form! $\quad = -\eta \frac{1}{N}\sum_{k=1}^{N} \left.\frac{\partial E(k)}{\partial w_{ij}}\right|_{w_{ij}=w_{ij}(s)}$

{ Batch : one epoch → one update
{ Sequential : one epoch → N updates

Which is BETTER? (one output case)

Sequential mode : $E(k) = \frac{1}{2}e(k)^2$

$$\Delta w_{ij}(k) = -\eta \frac{\partial E(k)}{\partial w_{ij}}$$

→ just estimate of the truth gradient

NO!

a good estimation?

Batch mode : $E_{av} = \frac{1}{N}\sum_{k=1}^{N} E(k)$

$$\Delta w_{ij} = -\eta \frac{1}{N}\sum_{k=1}^{N} \frac{\partial E(k)}{\partial w_{ij}}$$

→ theoretically better

However, sequential learning generally provides better result.

↕

local minimum problem!

↳ sequential learning can release this!

⇓

since in sequential learning, the estimate of truth gradient is very

↳ noise!

↓

jump out of local minima

优点!

noisy ! ⇒ advantage!

Amazing !

缺点 ⇒ 优点

give us more chance to have better results !

just chance

缺点

② Batch method advantage ⇒ guarantee to converge

Sequential learning ⇒ no guarantee to converge

⇓

{ ① let $\eta^{(s)} \to \eta^{(s)}(k) \to 0$ as $k \to \infty$

② { mini-batches { firstly sequential, then batch

③ Anther advantages of Batch Method ⇒ the biggest advantage

⇒ use second-order information to increase convergence

{ small # of weights → Newton Method, LM alg.

moderate → quasi-Newton

large → CG method

---

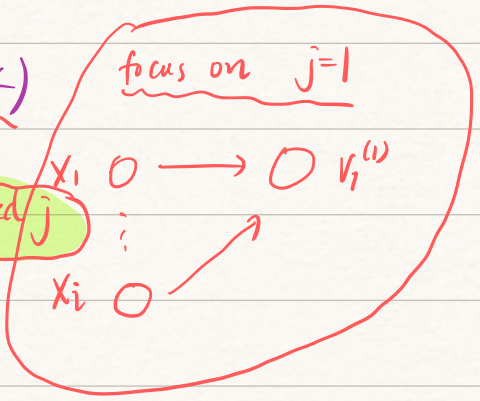
② normalize inputs

reasonable

if $x_i \in [0,1]$ → normalization

$\Delta w_{ji}^{(l)}(k) = \eta\, \delta_j^{(l)}(k)\, x_i(k)$

focus on $j=1$



same for fixed j

Heuristic

→ if $x_i(k) \geq 0$ for all i,

then if ouput ≥ 0, all change $\Delta w_{ji}^{(l)}(k) \geq 0$

不好!

not efficient

[-1,1] likely.

we should normalize the inputs s.t their average ≈ 0

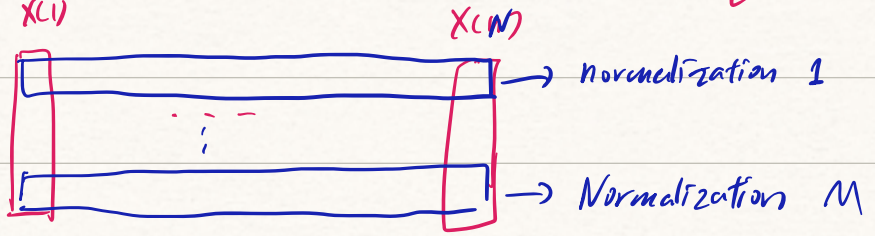
Moreover, the bigger the input, the larger the change!

Advantage: Normalization ⇝ faster convergence

Normalization Method:

$$\bar{x}_i = \frac{\sum_{n=1}^{N} x_i(n)}{N} \qquad \sigma_i = \sqrt{\frac{\sum (x_i(n) - \bar{x}_i)^2}{N}}$$

$$\Rightarrow x_i'(n) = \frac{x_i(n) - \bar{x}_i}{\sigma_i} \sim \begin{cases} \text{mean } 0 \\ \text{var } 1 \end{cases}$$



③ The choice of activation $f^?$

① hidden neuron

- logistic: $\log sig : \mathbb{R} \to [0,1]$, $\log sig(x) = \frac{1}{1+e^{-x}}$
- hyperbolic tangent: $\mathbb{R} \to [-1,1]$, $\tan sig(x) = \frac{2}{1+e^{-2x}} - 1$

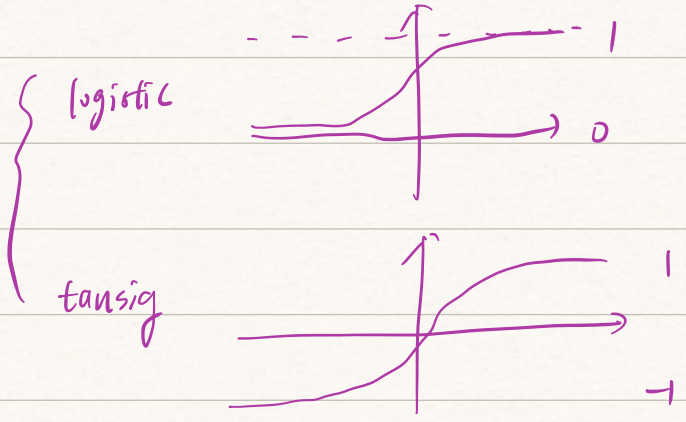
→ preferred

→ similar reason as normalization

② output neuron

- reg → identity mapping (we can still use logsig or tansig to normalize)
- PR → logistic

④ For PR, how to design target value?



if set output $\begin{cases} 0 \\ 1 \end{cases}$ or $\begin{cases} -1 \\ 1 \end{cases}$

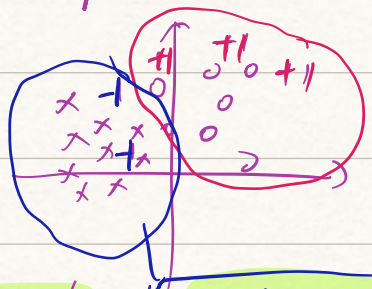
Disadvantage

① then the weights are forced to be AS BIG AS POSSIBLE

② and if induced local field $\to \infty$

$\Rightarrow$ gradient $\to 0$ (update slow !)

③ Large weights will make all the output value close to $\begin{cases} 0 \\ 1 \end{cases}$ or $\begin{cases} -1 \\ 1 \end{cases}$



$\Rightarrow$ Decision Boundary are meaningless !

not what we want

$\to$ As a consequence, we don't set $\begin{cases} 0 \\ 1 \end{cases}$ or $\begin{cases} -1 \\ 1 \end{cases}$

instead. we set the target value to the maxima second derivative on the sigmoid function !

↳ 最陡峭的地方 $\begin{cases} 0.2 \\ 0.8 \end{cases}$ $\begin{cases} -0.6 \\ 0.6 \end{cases}$

recommended !

---

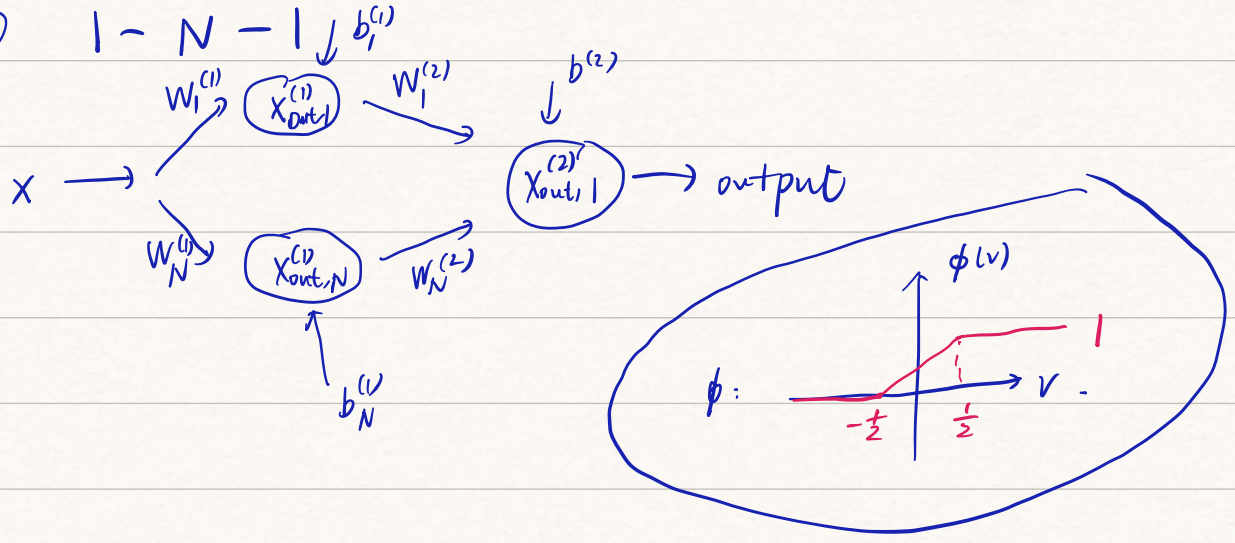
⑤ How many hidden $\begin{cases} \text{layers} \\ \text{neurons} \end{cases}$?

section 4.14

⇒ Geometrical meanings of { # of neurons ? / weights ? / bias ? }

① Hidden Neuron (MLP)

① 1 - N - 1



$$Y(x) = \sum_{i=1}^{N} W_i^{(2)} \phi(W_i^{(1)} x + b_i^{(1)}) + b^{(2)}$$

平移

拉伸 | 拉伸，翻转，平移

$W_i^{(2)} \phi(W_i^{(1)} x + b_i^{(1)}) + b^{(2)}$ 其中一项的图像

一共有N项

height = $W_i^{(2)}$

center = $\frac{-b_i^{(1)}}{W_i^{(1)}}$

width = $\frac{1}{W_i^{(1)}}$

slope = $W_i^{(1)} W_i^{(2)}$

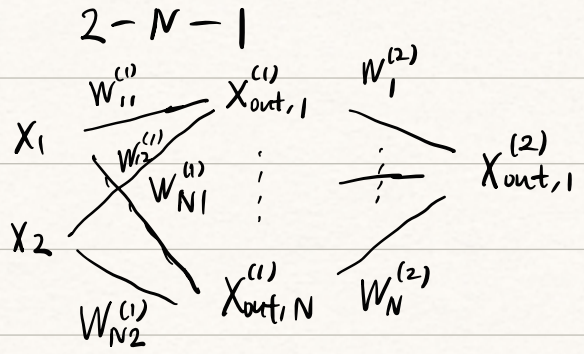
shift = $b^{(2)}$ → offset

starting point = $-\frac{0.5}{W_i^{(1)}} + \frac{-b_i^{(1)}}{W_i^{(1)}}$

Conclusion → guideline! → 1-N-1 MLP

# of neurons (hidden) ≈ # of line segments!

Consider for 2-dimensional case.



$$\Rightarrow \quad y(\underline{x}) = \sum_{i=1}^{N} W_i^{(2)} \phi\left(W_{i1}^{(1)} x_1 + W_{i2}^{(1)} x_2 + b_i^{(1)} \right) + b^{(2)}$$

↓ imagine what this like!

Gaussian Hill ⟶ $\phi(\cdot)$ → logsig ⇒ 2-3-1 can give good approximation

$\phi(\cdot)$ → linear ⇒ 2-20-1.

② More Hidden Layer!

- 1-9-1 ⟶ paras: $18 + 10 \times 1 = 28$
- 1-3-3-1 ⟶ paras: $2\times3 + 4\times3 + 4\times1 = 22$

↓ decrease!

But structure more complicated

↓

more likely to trap into local minima!

Reason: cost surface is more complicated

Motivation

① If a task can be decomposed into several sub-tasks (simple to solve)

⇓

Multi-layers ≥ 2

② else:

1 hidden layers

---

⑥ Generalization { under-fitting / over-fitting

{ # of samples (traing) small ⇒ over-fitting
# of free parameters
Traing process

Guideline:

$N \approx O(\frac{W}{\varepsilon})$ { W: # of free parameters, ε: error

为了 → more traing samples

☆ { less free parameters, simple structure

GOAL: choose parameters as less as possible!

☆ Method: use line segments to estimate

Network

the least # of parameters

& structure !

---

① ☆☆☆

the value of weights (more important) is more important than the size of network !

↓

can be explained by the geometric meaning of MLP

↓

Strategy : Penalty / Regularization

previously $\lambda = 0$! → no penalty

$F = E_D + \lambda E_W$ Model Modification (objective)

① $E_W = \sum_{i=1}^{N} [w_i^{(1)2} + w_i^{(2)2} + b_i^{(1)}]^2 + b^{(2)2}$

② $E_W = \sum_{i=1}^{N} (w_i^{(2)} w_i^{(1)})^2$ → slope

→ more reasonable !

Occam's Razor

↓

we are interested in minimal structure of NN.

difficult to estimate when dimension is high!

⇒ Our strategy to overcome over-fitting ⇒ ① minimal structure of NN ② Regularization!

SVD! ⇒ $H_{N\times n} = U_{N\times N}\,\Sigma_{N\times n}\,V^T_{n\times n}$

$= \sum_{i=1}^{k} \sigma_i u_i v_i^T$ if rank(H) = k

low rank approximation

numerical rank

numerical rank

actual rank

$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ $B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0.001 \end{pmatrix}$

numerical rank 3 3

actual rank 3 2!

SVD = (2.247, 0.8, 0.55)

SVD = (2.1358, 0.6622, 0.0071)

effective rank = 3

effective rank = 2

⇒ Goal: Define Effective rank of hidden Neurons

⇓

find the redundant hidden neuron

→ N training samples $\in \mathbb{R}^M$ — n hidden neurons

inputs $\begin{cases} x_1(1) \\ \vdots \\ x_M(1) \end{cases}$ $\xrightarrow{w_{i1}, \dots, w_{iM}}$ $v_i := \sum_{j=1}^{M} w_{ij} x_j(k) + b_i \rightarrow f_i(v_i(k)) = h_{ki}$

$$h_{ki} = f_i\left(\sum_{j=1}^{M} w_{ij} x_j(k) + b_i\right) \in H_{N \times n} \rightarrow \text{hidden neuron matrix}$$

$$\begin{pmatrix} h(1)^T \\ \vdots \\ h(N)^T \end{pmatrix}_{N \times n} \qquad h(i)^T = [h_1(i) \cdots h_n(i)]$$

⇓

$$H_{N \times n} = U_{N \times N} \Sigma_{N \times n} V^T_{n \times n}$$

↓ cut-off

define a threshold!

Framework

singular value $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$

↓ normalization

$\tilde{\sigma}_1 \geq \tilde{\sigma}_2 \geq \cdots \geq \tilde{\sigma}_n \geq 0$

① $\tilde{\sigma}_i = \frac{\sigma_i}{\sum \sigma_i}$

② $\tilde{\sigma}_i = \frac{\sigma_i^2}{\sum \sigma_i^2}$

↓

threshold γ (95%, 99%)

$\min_k \sum_{i=1}^{k} \tilde{\sigma}_i \geq \gamma$ → cutoff

① start with n (# of hidden neurons) big enough

↓ cut

② cut off

↓ cut off

⋮

terminate!

Conclusion: use SVD on $H_{N\times n} := \begin{pmatrix} h^T(1) \\ \vdots \\ h^T(N) \end{pmatrix}_{N\times n}$ for large-dimension

to shrink hidden neurons !

Σ值: Determine # of hidden neurons:

① dimension low: minimal # of line segments

② high dimension: start with large number

↓

use SVD to cut!

2:53:00 开始Σ值

Deal with over-fitting
- ① minimal structure
- ② Regularization

main questions:
- ① # of hidden layers?
- ② # of hidden neurons?
- ③ choice of activation f^c
  - hidden → tansig
  - output
- ④ normalization → preprocess
- ⑤ Batch / Sequential
- ⑥ over-fitting
  - Minimal structure
  - Regularization