1/28 Deep Feedforward Networks

1. Deep Feedforward Networks ⇒ 여러 중요 applications의 기본! ex) CNN, RNN, Transformer..

→ 가장 기본적인 neural nets

→ 역할: 크나의 벡터(이미지, scalar)를 다른 벡터로 mapping

→ goal: approximate some function f^*

⇒ $x \longrightarrow y$ 가는 mapping이 무엇인지 가정함을 기반으로 찾는 것!

ex) a classifier: $y = f^*(x)$ maps input x to category y

y = label, x = image

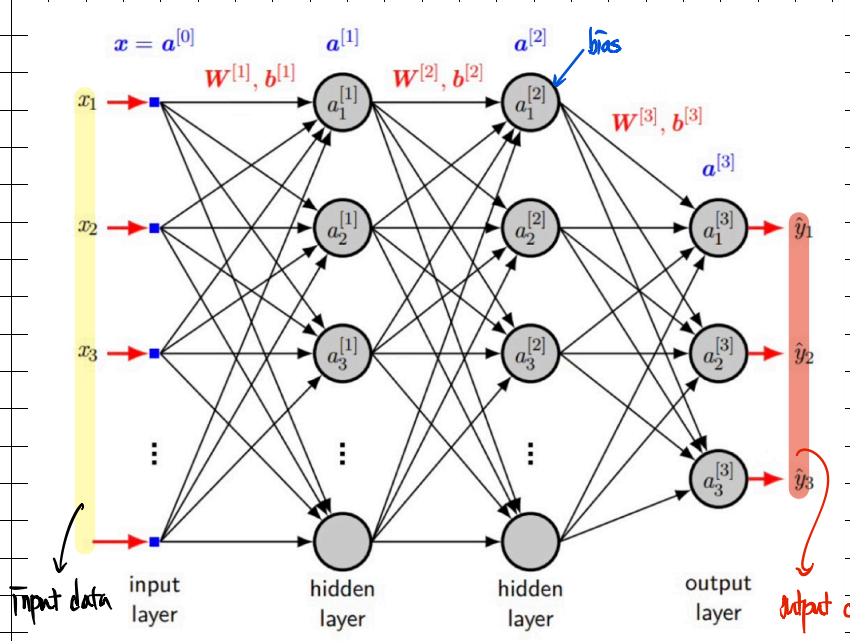
→ how it works? : parameterize + learn

⇒ define a mapping $y = f(x;\theta)$ and

⇒ learn parameter θ that give the best approximation

2. Architecture

a feedforward neural net with two hidden layers.



∴ Input data (vector) $\longrightarrow f^*(\,) \longrightarrow$ output (vector)

→ called feedforward because

⇒ information flows $x \longrightarrow f \longrightarrow y$

⇒ no feedback connection! : 출력 영향 ×

≈ 과거

→ associated with a directed acyclic graph.

⇒ f^1, f^2, f^3 connected in a chain!

$f(x) = f^3(f^2(f^1(x)))$ ⟹ $x \rightarrow f^{[1]} \rightarrow f^{[2]} \rightarrow f^{[3]} \rightarrow f(x)$ ($f(x)$)

- $f^{[l]}$: called l-th layer
- final layer : called output layer
- chain length ⇒ model depth

3. Training neural nets : data를 이용해서 우리가 원하는 $f^*(x)$ approximate 하는 $f(x)$를 유도하는 과정! + 최적 θ를 찾아가는 과정

- training : we drive $f(x)$ to match $f^*(x)$
- training data : noisy / approximate examples of $f^*(x)$ (ε term)

  ⇒ in supervised example (x, y) (x: input, y: label) with $y \approx f^*(x)$

  → output layer 결정 가능 : MNIST, IMAGENET (softmax), BINARY (sigmoid)

- behavior of the other layers : not directly specified by training data

  ⇒ these layers : called hidden layers

  ⇒ feature are distributed over hidden layers

  ↳ ?? : feature 한 층에 대응되지 X!

  여러 hidden layer에 distributed 하게 표현됨!

  ⇒ distributed representation

4. Understanding feedforward nets

- begin with linear models

  ⇒ pros : efficient / reliable (closed form or convex)

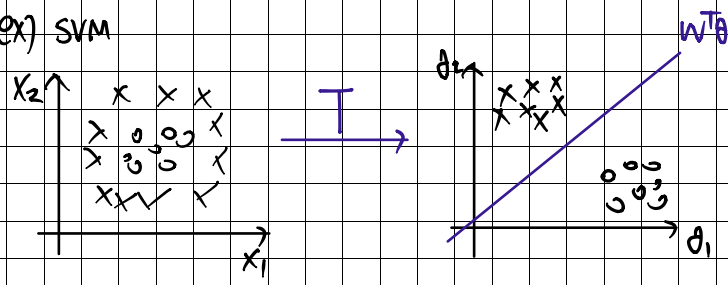
  cons : cannot understand interaction terms like $x_1 \cdot x_2$

- to extend linear models to represent nonlinear function of x

  ⇒ apply linear model transformed input $\phi(x)$

$x \longrightarrow$ [nonlinear transform ϕ] $\longrightarrow \phi(x)$ → new representation for x

(ex) SVM



- how to choose ϕ

  In deep learning ⇒ Learn

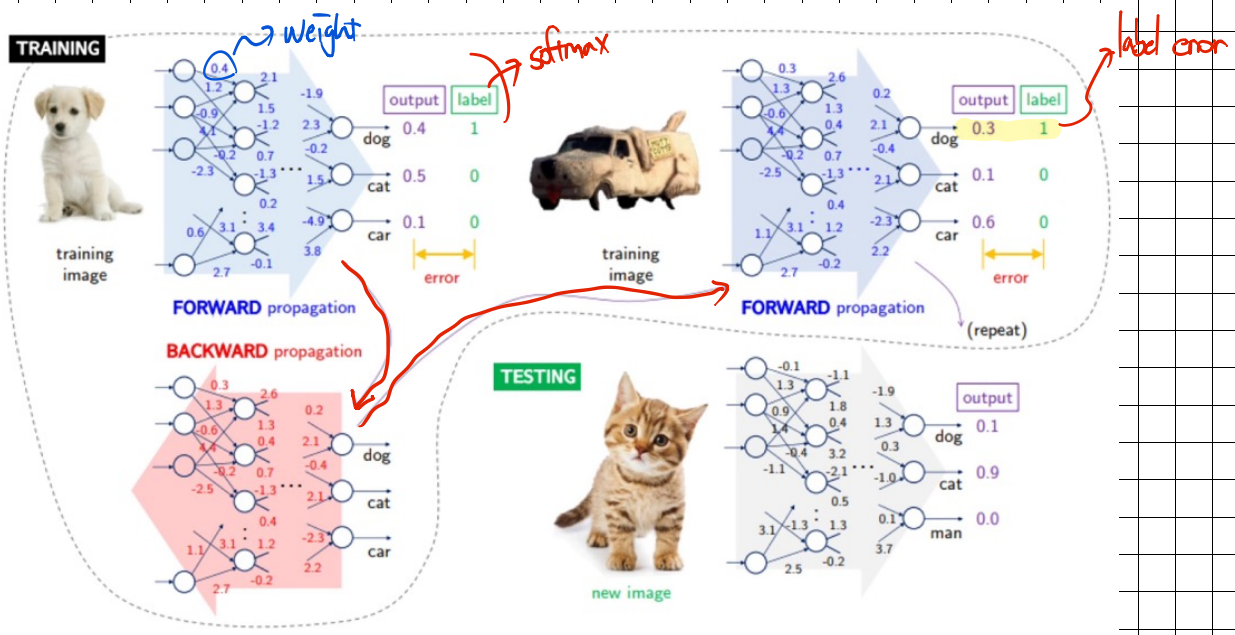
  ⇒ we have a model $y = f(x;\theta,w) = \phi(x;\theta)^T w$

$x \xrightarrow{\phi(\cdot;\theta)}$ feature $\xrightarrow{w} y$

y : output mapping을 위한 parameter : output layer의 weight

transform ϕ의 형태 결정 : hidden layer의 weight

5. Concepts of training & testing a neural net



6. Modern Neural Nets.

⇒ the same backprop / gradient descent : still use....

biggest change ?!

1. MSE ⇒ cross entropy loss → loss function 종류 여러가지 있다!!

2. Sigmoid ⇒ ReLu

gradient vanishing

7. Cost function for Neural nets.

- total cost function

  → primary cost function + regularization term

  $\Rightarrow \min E_{avg} = \min \left(E_{train} + \frac{\lambda}{n} \Omega(w) \right)$ (regularization term)

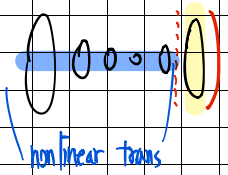
- most modern neural nets : trained using maximum likelihood

  i.e cost function = -NLL (min)

  = cross entropy (min)

- a recurring theme : gradient of cost function must be large ≈ 1 / predictable ?

8. Output units ⇒



transform ϕ의 형태 결정
: hidden layer의 weight

- suppose : a feedforward net provides hidden features $h = f(x;\theta)$

참고!

distribution	# classes	# trials (samples)
Bernoulli	2	1
multinoulli	k	1
binomial	2	n
multinomial	k	n

- Output layer : provides "additional" transformation from h

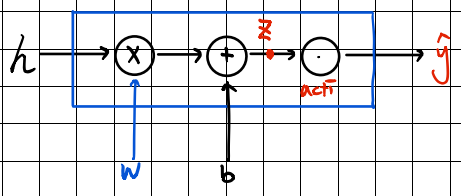
⇒ h (nonlinear trans한 x의 새로운 representation) → Output Unit (끝에서 두번째 layer) → $\hat{y}$

↳ linear / sigmoid (binary) / softmax (multi)

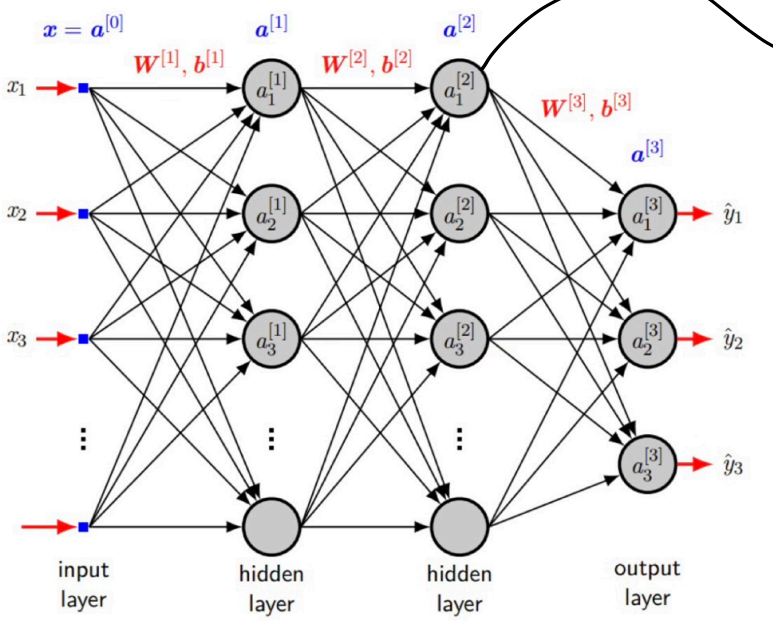
9. Type of Output Units

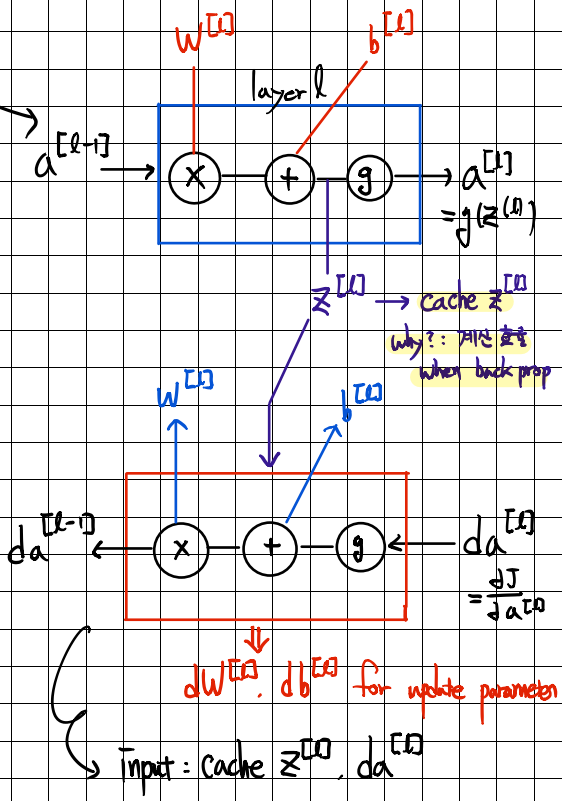
type	output	formula	output distribution
linear	vector	$\hat{y} = W^\top h + b$	Gaussian
sigmoid	scalar	$\hat{y} = \sigma(w^\top h + b)$	Bernoulli
softmax	vector	$\hat{y} = \mathrm{softmax}(W^\top h + b)$	multinoulli
ex) exp		$\hat{y} = \exp(w^\top h + b)$	Poisson

GLM에서 Link function.

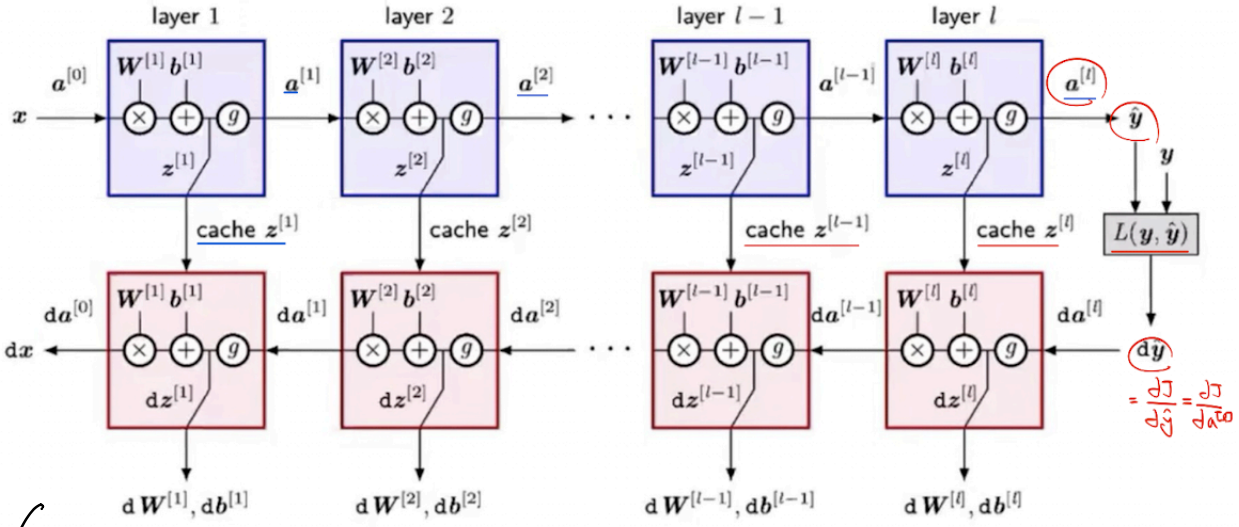


10. Architecture





Overall architecture



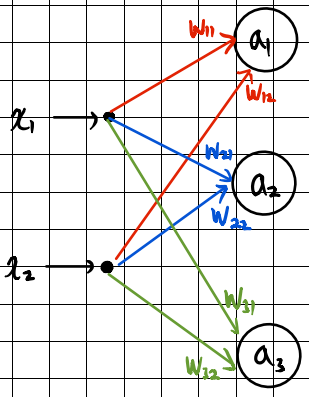
↳ by gradient we get ⇒ parameter update!

$$W^{[l]} \leftarrow W^{[l]} - \varepsilon dW^{[l]}$$

$$b^{[l]} \leftarrow b^{[l]} - \varepsilon db^{[l]}$$

11. Weight matrix conventions

ex)



- RL (right-left) convention

$$\overleftarrow{W} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \Rightarrow \text{then, } \underset{3\times 2}{\overleftarrow{W}}\underset{2\times 1}{x} \longrightarrow z = \overleftarrow{W}x + b$$

- LR convention

$$\overrightarrow{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \Rightarrow \text{then, } \underset{3\times 2}{\overrightarrow{W}^T}\underset{2\times 1}{x} \longrightarrow z = \overrightarrow{W}^T x + b$$

12. Hidden Units

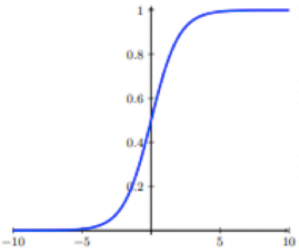
- what they do?!

1. accept a vector of input $\vec{x}$
2. compute affine transformation $z = \vec{W}^T x + b$
   ⇒ linear tranformation + translation
3. apply an element-wise nonlinear function $g \longrightarrow z$
   $g(z) = \begin{bmatrix} g(z_1) \\ g(z_2) \\ g(z_3) \end{bmatrix}$, when $z^T = [z_1, z_2, z_3]$
4. return activation $a = g(z)$

- hidden units differ only in activation function
  ⇒ g 통과 전 affine transf. 다 한다!
  → 요즘은 ReLU (rectified linear units) 사용
  → 새로 제안된 nonlinear function 사용 가능.. ex) sin, cos..
  but published only if clearly significant ↑

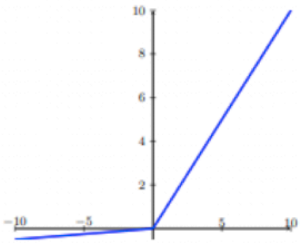
- mixing activation function types in a layer is uncommon

13. Activation functions



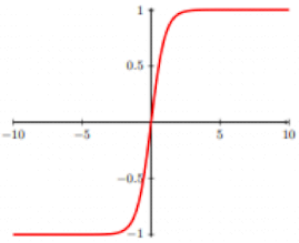
- sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



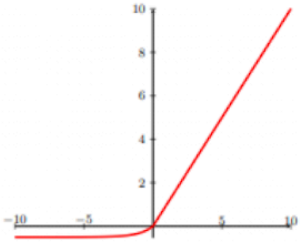
- Leaky ReLU

$$\max\{0.01z, z\}$$



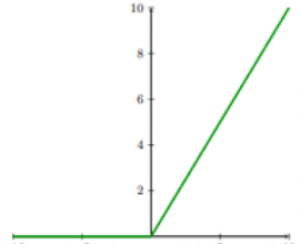
- tanh

$$\tanh(z)$$



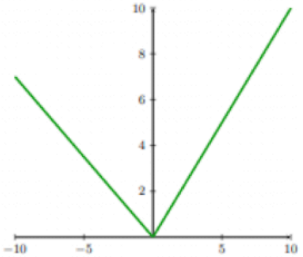
- ELU

$$\begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases}$$



- ReLU

$$\max\{0, z\}$$

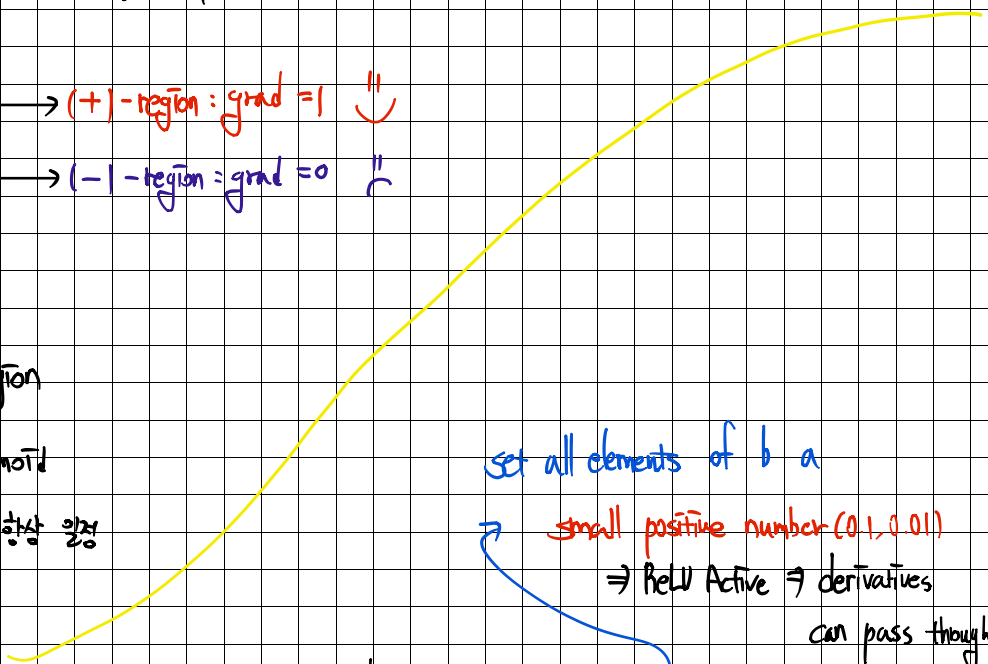


- maxout

$$\max\{z_1, z_2\}$$

$z = \vec{w}^T x + b$, $ReLU(z) = \max\{0, z\} = g(z)$

$$g'(z) = \begin{cases} 1, & z \geq 0 \longrightarrow (+)\text{-region : grad = 1} \\ 0, & z < 0 \longrightarrow (-)\text{-region : grad = 0} \end{cases}$$



- Pros

i) no saturation in (+) region

ii) converges faster than sigmoid

: (+) region에서 gradient가 항상 일정

- Cons

i) not zero-centered output

→ 모델학습이 안정적으로 지속됨...

→ gradient가 한방향으로 변함

→ optimization이 지그재그 (느리게 학습)

ii) zero gradient in (-) region

→ dying ReLU problems...

⇒ 변형 ReLU 등장 시작점!

set all elements of b a small positive number (0.1, 0.01)

⇒ ReLU Active ⇒ derivatives can pass though..

14. ReLU optimization : Why ReLU?

- easy to optimize (∵ so similar to linear units)
  only when ReLU Active

- derivatives through ReLU

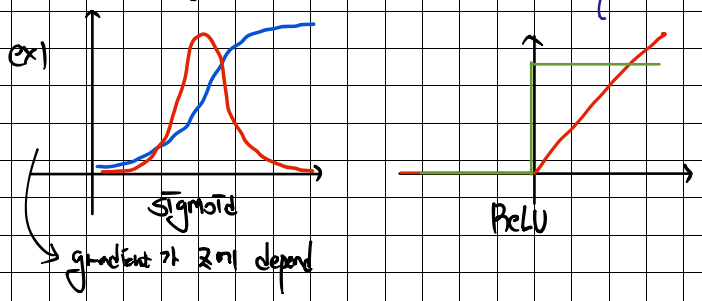
  (1) remain large when ReLU Active

  ⇒ Active 한 애 gradient = 1 이므로.

  ⇒ sigmoid / tanh 계열 $|z|$ ↑ → gradient = 0 에 수렴.

  ⇒ 즉! Vanishing gradient 없다!

  (2) not only large but consistent → second derivation : 0 ?? 이 training에 좋다..?



15. One-sided inputs slows down training

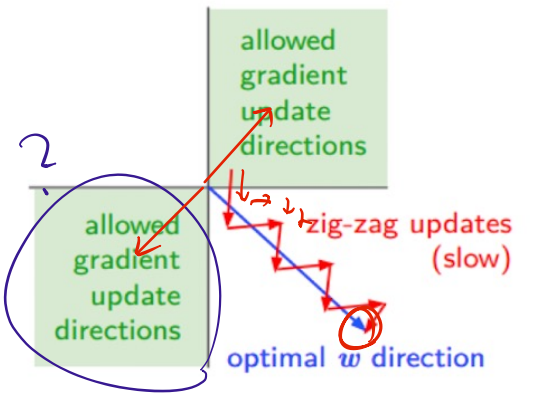
- Recall : $\frac{\partial}{\partial w}(wx) = x$

  ⇒ weight update direction 은 x 의 부호에 depend

  $w \leftarrow w - \varepsilon \delta x$

- One-sided input?? all (+), (-), $EX \neq 0$

  ⇒ not zero-centered



- Then why One-sided inputs are matter?

  ⇒ slow convergence ! ⇒ zig-zag updates

  ⇒ update direction biased

So ! Normalization 필요! (Batch Norm ?)

input (각 layer)

16. Exploiting Linearity : gradient가 선형일 때 학습!

- principle of ReLU

  ⇒ models are easier to optimize if their behavior is close to linear

  ⇒ non-linear ⇒ gradient vanishing problem

  linear: flow of gradient

- cons... ∵ linear boundary

17. Architecture exploration

- main Architecture considerations in chain-based architectures.

  ⇒ Network depth and layer width

  depth: more chain, # of layer

  width: more row? node

- a feedforward net with a single layer

  ⇒ sufficient to represent any function

  ⇒ may fail to learn generalize correctly

  node ↑ → estimate 할 weight & bias ↑ → ∴ parameter ↑

깊게!

- deeper network ⇒ increase depth!

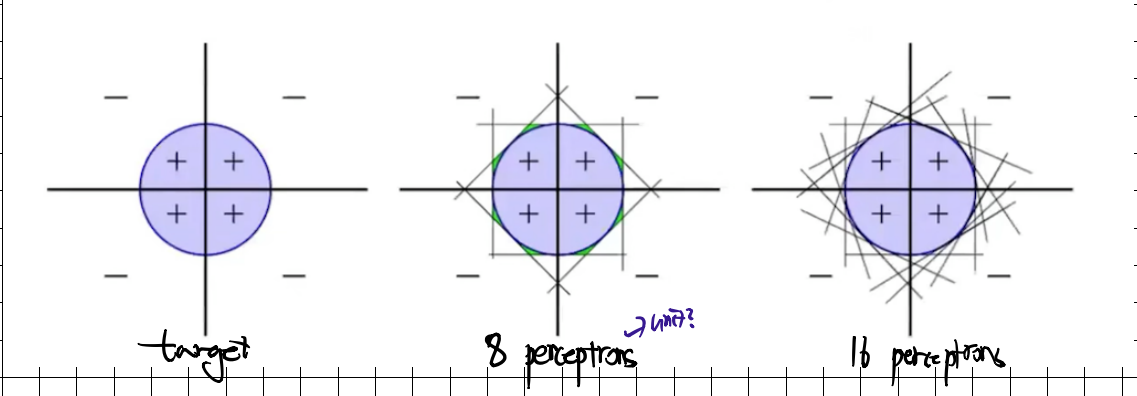
  ⇒ use far fewer units per layer and fewer parameters.

  ⇒ generalize better to test set

  ⇒ but harder to optimize (vanishing, exploring gradient)

18. Universal approximation Theorem

- a feedforward net with linear output layer + hidden layers
  ⇒ can approximate any function (give enough hidden units)



19. Network Size

- UAT says there exists a network large enough to achieve any accuracy
  ⇒ but does not say how large this network will be.
- Unfortunately an exponential number of hidden units may be required in the wose case
- empirically: greater depth ⇒ better generalization