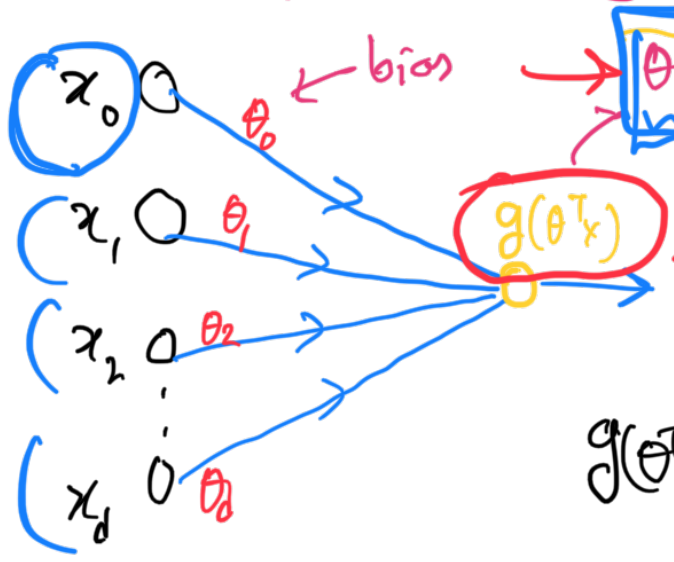
Classification — Logistic Regression

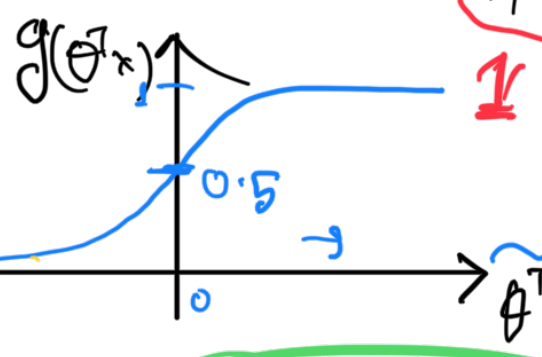
$y = 0, 1$



$\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \cdots \theta_d x_d$

sigmoid

$$P(y=1 \mid x, \theta) = \frac{1}{1 + e^{-\theta^T x}} = h_\theta(x)$$

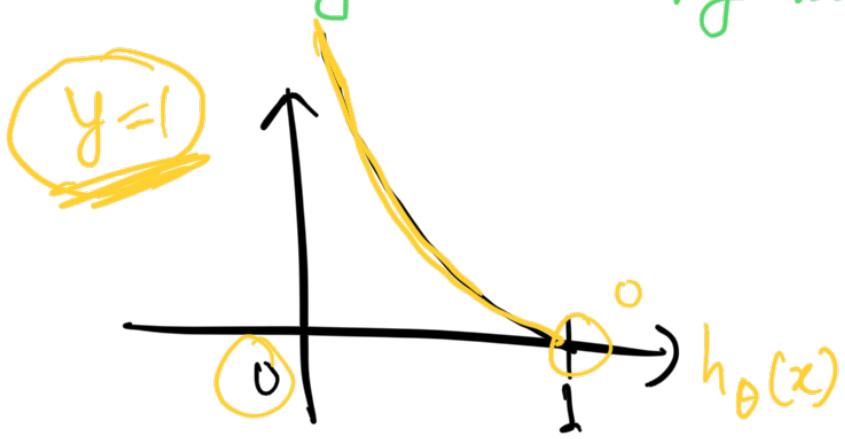


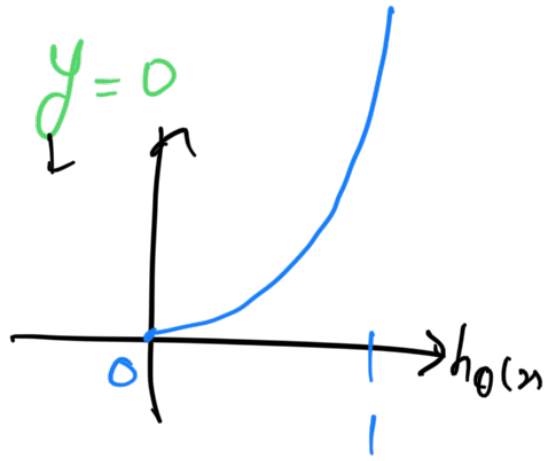
Logistic regression unit

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left[-y^i \log\left(h_\theta(x^i)\right) - \left[1 - y^i\right] \log\left(1 - h_\theta(x^i)\right) \right]$$

$y^i \in \{0, 1\}$

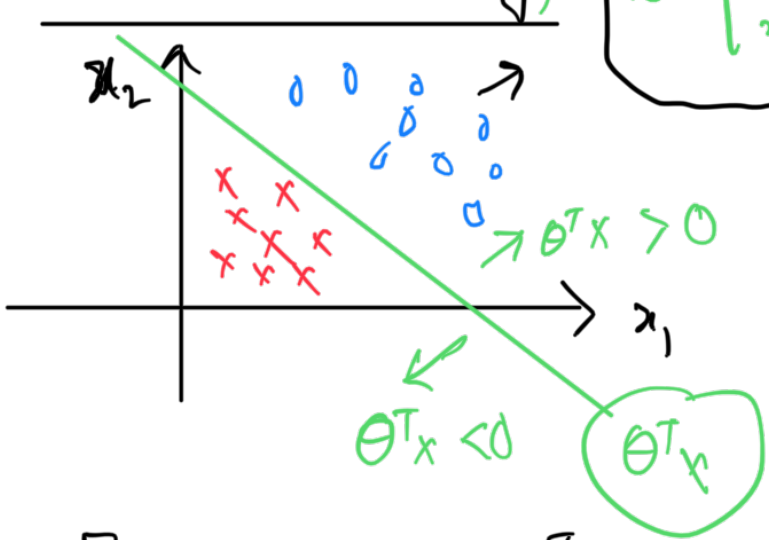
↳ Binary cross entropy loss



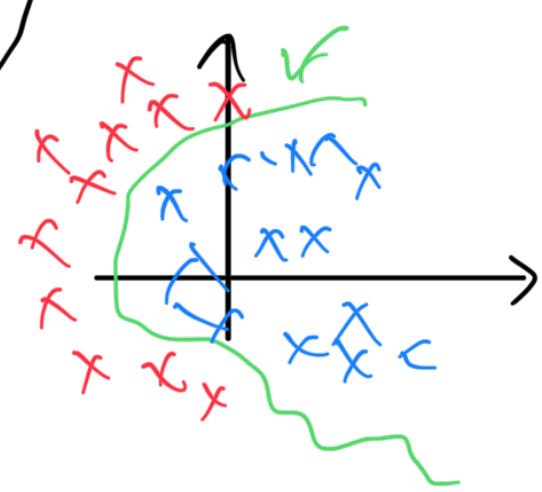


(Decision Boundary)

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \end{bmatrix}$$



$$x = [x_0\ x_1 \cdots x_d]^T$$



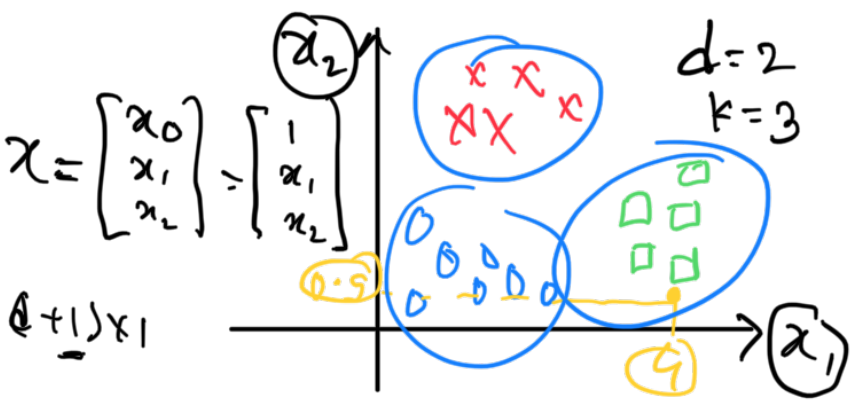
$$x = [x_0, x_1 \cdots x_d, x_0^2, x_1^2, \ldots x_d^2, x_0 x_1, x_0 x_2 \cdots]^T$$

Multiclass classification

Email: Friends / work / spam (0 / 1 / 2)

image: human / horse / cat / dog (0 / 1 / 2 / 3)

$y = \{0, 1, 2\}$



Algo-1: One-vs-All

$h_{\theta_0}^{(0)}(x)$, $h_{\theta_1}^{(1)}(x)$, $h_{\theta_2}^{(2)}(x)$

0.3, 0.2, 0.7 $\quad i = 0, 1, 2$

idea: Train a logistic regression classifier $h_\theta^{(i)}$ for each class i to predict the probability of $y=i$

$$y_{pred} = \arg\max_i h^{(i)}(x)$$

index

$$h_{\theta_0}^{(0)}(x) = \frac{1}{1+e^{-\theta_0^T x}} = P(y=0 \mid x, \theta_0), \quad \theta_0 = [\theta_{00}, \theta_{01}, \theta_{02}, \cdots \theta_{0d}]^T$$

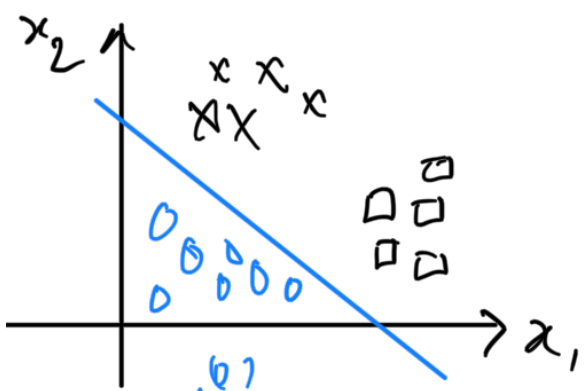
bias, x_1, x_2, x_d

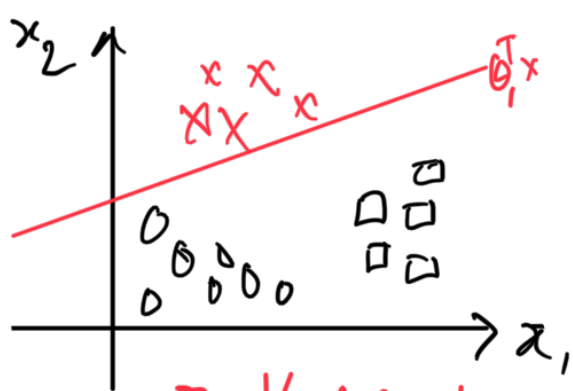
$$h_{\theta_1}^{(1)}(x) = \frac{1}{1+e^{-\theta_1^T x}} = P(y=1 \mid x, \theta_1), \quad \theta_1 = [\theta_{10}, \theta_{11}, \theta_{12}, \cdots \theta_{1d}]^T$$

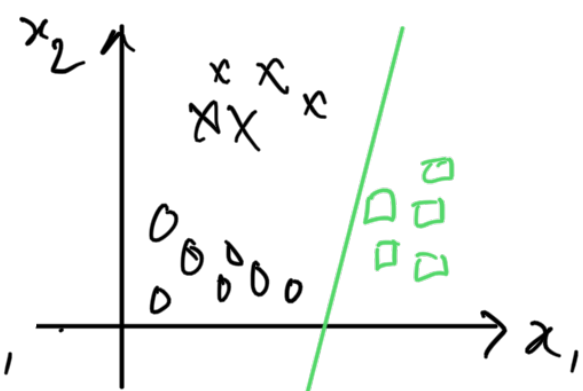
$p(y=2)$

bias, x_1, x_2

$$h_{\theta_2}^{(2)}(x) = \frac{1}{1+e^{-\theta_2^T x}} = P(y=2 \mid x, \theta_2), \quad \theta_2 = [\theta_{20}, \theta_{21}, \theta_{22}, \cdots \theta_{2d}]^T$$







$3 \times (d+1)$ $\quad = K$

$$W = \begin{bmatrix} -\theta_0^T- \\ -\theta_1^T- \\ \theta^T \end{bmatrix} = \begin{bmatrix} \theta_{00} & \theta_{01} & \cdots & \theta_{0d} \\ \theta_{10} & \theta_{11} & \cdots & \theta_{1d} \end{bmatrix}$$

weights/params for $h^{(1)}$

params for $h^{(1)}$

Weight

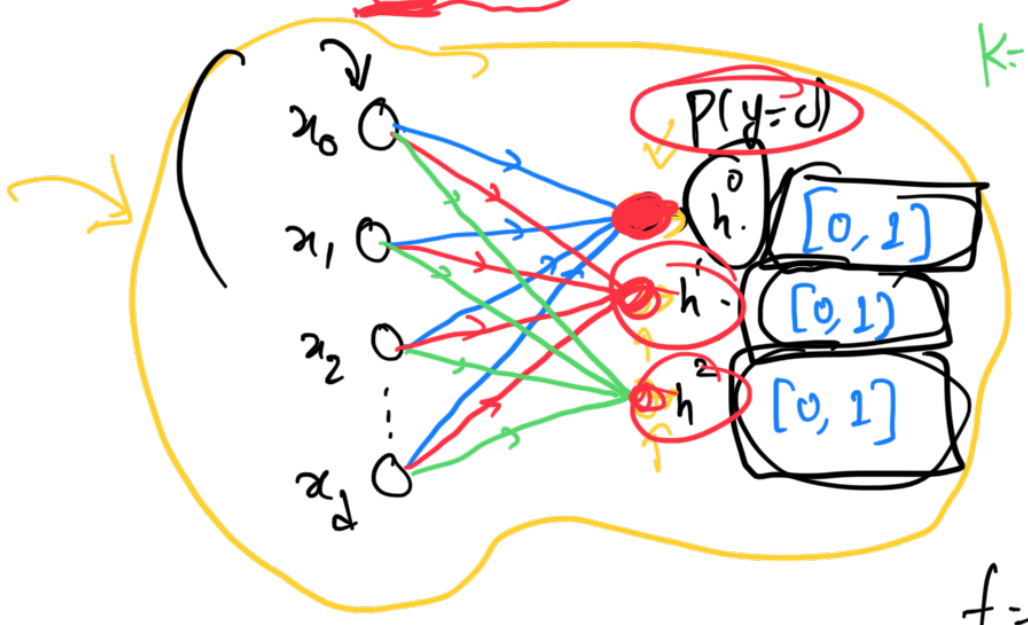
$[\theta_{20} \quad \theta_{21} \quad \dots \quad \theta_2]$ → params for $h_1^{(2)}$

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = h = f(Wx) \quad (k \times 1)$$

$$= f\begin{pmatrix} \theta_0^T x \\ \theta_1^T x \\ \theta_2^T x \end{pmatrix} = \begin{bmatrix} f(\theta_0^T x) \\ f(\theta_1^T x) \\ f(\theta_2^T x) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-\theta_0^T x}} \\ \frac{1}{1+e^{-\theta_1^T x}} \\ \frac{1}{1+e^{-\theta_2^T x}} \end{bmatrix}$$

W: $k \times (d+1)$, x: $(d+1) \times 1$

# of class



k = number of class

$h = f(Wx) = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix}$ $k \times (d+1)$ — $P(y=0)$, $P(y=1)$, $P(y=2)$

$f = \left[\ \right]$ → Prob. dist of y.

Numerical example

$\theta_0 = [1, -1, -1]^T$, $\theta_1 = [-1, -0.5, 1]^T$, $\theta_2 = [-4, 2, -1]^T$

→ $x = [1, 4, 0.5]^T$

$$W = \begin{bmatrix} 1 & -1 & -1 \\ -1 & -0.5 & 1 \\ -4 & 2 & -1 \end{bmatrix} \Rightarrow Wx = \begin{bmatrix} 1\times1 + (-1)\times4 + (-1)\times0.5 \\ (-1)\times1 + (-0.5)\times4 + 1\times0.5 \\ (-4)\times1 + 2\times4 + (-1)\times0.5 \end{bmatrix} = \begin{bmatrix} -3.5 \\ -2.5 \\ 3.5 \end{bmatrix}$$ ($\theta^T x$)

W: weight

$$f(Wx) = \begin{bmatrix} \frac{1}{1+e^{-(-3.5)}} \\ \frac{1}{1+e^{-(-2.5)}} \\ \frac{1}{1+e^{-3.5}} \end{bmatrix} \approx \begin{bmatrix} 0.03 \\ 0.08 \\ 0.92 \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix}$$

Activation function

$\therefore y_{pred} = 2$ (□)

$0.92 + 0.08 - 0.03 \approx 1.08$

$\Sigma h = 1$

Learning

$J^{(0)}(\theta_0)$ $J^{(1)}(\theta_1)$ $J^{(2)}(\theta_2)$

Binary Cross Entropy loss

Training/optimizing - separately [G.D. / Adam / RMSprop]

Prob: ① Have to train separately

② No interpretation for h [sum(h) > 1]

ob y = 4 ImageNet → K = 1000

Sol^n: Softmax Classifier - Extension of L.R.

✓ ① We want sum(h) = 1

✓ ② We want end to end learning

y = 0 → 0.1
y = 1 → 0.2
y = 2 → 0.7

→ prob. dist. of y

⇒ modifications

① from f = sigmoid to f = softmax → Activation/non-li. $\sum h = 1$

② Change y to one-hot vector

③ from J(θ) = BCE to CE [Cross entropy]

① ⇐ $W_x = \begin{bmatrix} \theta_0^T x & \theta_1^T x & \cdots & \theta_{k-1}^T x \end{bmatrix}^T \rightarrow f(W_x) = \begin{bmatrix} \frac{1}{1+e^{-z_0}} & \frac{1}{1+e^{-z_1}} & \cdots \end{bmatrix}^T$

$(K) \times (d+1)$; $(d+1) \times 1$

$$f(W_x) = \begin{bmatrix} f_0 & f_1 & \cdots & f_{K-1} \end{bmatrix}^T \quad K \times 1$$

K = 2

$K \times 1$

$f_1 = P(y=1)$

$f_2 = P[y=2]$

$\sum f_i = 1$

$$f_i = \frac{e^{z_i}}{\sum_{i=0}^{k-1} e^{z_j}}$$

$$f_2 = \frac{e^{z_2}}{e^{z_0} + e^{z_1} + e^{z_2} \cdots}$$

① $\sum_{i=0}^{k-1} f_i = 1$

② $0 \le f_i \le 1$

$$P(y=i) = f_i = \frac{e^{z_i}}{\sum_{j=0}^{k-1} e^{z_j}}$$

f → probability distribution of y

② ⇒ $y = [0, 1, 2] \rightarrow y = \left\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix} \right\}$ (0, 1, 2)

* one-hot encoding

③ $J(W) = -\frac{1}{n}\sum_{i=1}^{n}\left[\sum_{k=0}^{k-1} y_k^{(i)} \log f_k\right]$ ← model's output

avg. over all training sample

Cross Entropy loss

$W \rightarrow$ weight matrix $\Rightarrow$ best

Optimizer → same as before (Gradient Descent)

$W^{(0)} = \text{random}$

while not converged:

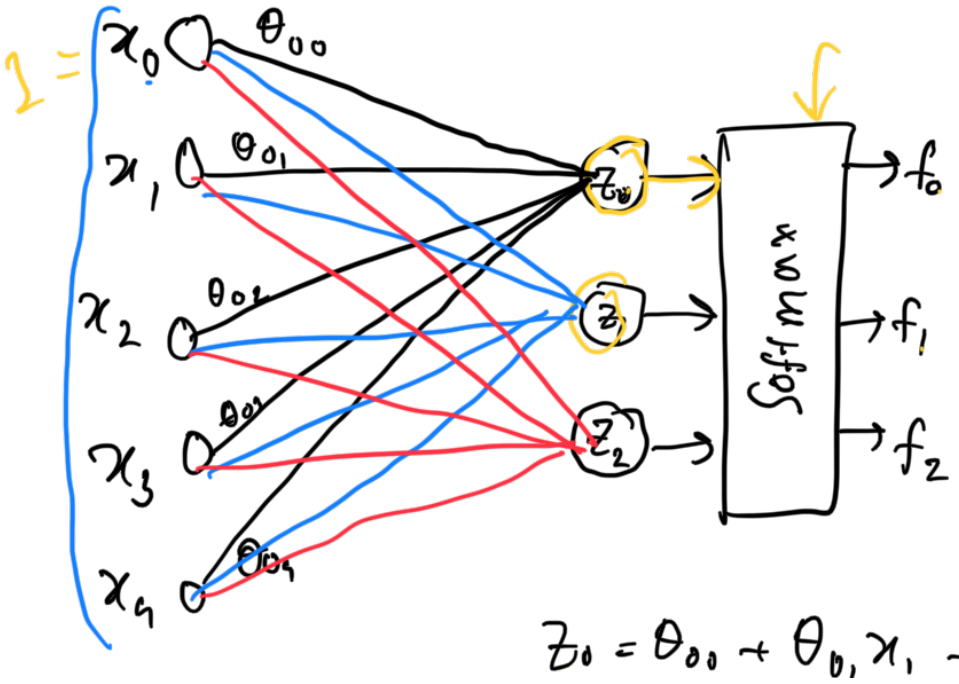
$$W^{(k+1)} = W^{(k)} - \nabla_W J \leftarrow$$

if $\nabla_W J \approx 0$:

converged = True.

$W * z = z = [\ ]$

$f(z) =$



$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

$f = \text{Softmax}(Wx)$

$z_0 = \theta_{00} + \theta_{01}x_1 \text{ --- } \theta_{04}x_4$

$z_1 = \theta_{10} + \theta_1 x_1 \dots \theta_{11} x_5$

for z_0 → $\theta_0 = [0,\ 0.01,\ -0.05,\ 0.1,\ 0.05]^T$ →

for z_1 → $\theta_1 = [0.2,\ 0.7,\ 0.2,\ 0.05,\ 0.16]^T$

for z_2 → $\theta_2 = [-0.3,\ 0.0,\ -0.45,\ -0.2,\ 0.33]^T$

$x^{(i)} = [1,\ -15,\ 22,\ -44,\ 56]$

$y^{(i)} = 2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ ← one-hot

$$W = \begin{bmatrix} 0 & 0.01 & -0.05 & 0.1 & 0.05 \\ 0.2 & 0.2 & 0.2 & 0.05 & 0.16 \\ -0.3 & 0.0 & -0.45 & -0.2 & 0.33 \end{bmatrix}$$

$$z = \begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = Wx = \begin{bmatrix} -2.58 \to z_0 \\ 0.86 \to z_1 \\ 0.28 \to z_2 \end{bmatrix}$$

$$f(wx) = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

$0.016 + 0.631 + 0.353 = 1$

$$y_{pred} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = 1$$

$$f_0 = \frac{e^{z_0}}{e^{z_0} + e^{z_1} + e^{z_2}} = \frac{e^{-2.58}}{e^{-2.58} + e^{0.86} + e^{0.28}}$$

$$= \frac{0.058}{0.058 + 2.36 + 1.32} = 0.016$$

$$f_1 = \frac{e^{z_1}}{e^{z_0} + e^{z_1} + e^{z_2}} = \frac{2.36}{0.058 + 2.36 + 1.32} = 0.631$$

$$f_2 = \frac{e^{z_2}}{e^{z_0} + e^{z_1} + e^{z_2}} = \frac{1.32}{0.058 + 2.36 + 1.32} = 0.353$$

$$J_i = -\sum_{k=0}^{k-1} y_k \log f_k = -$$

$y = \begin{bmatrix} 0 \to y_0 \\ 0 \to y_1 \\ 1 \to y_2 \end{bmatrix}$

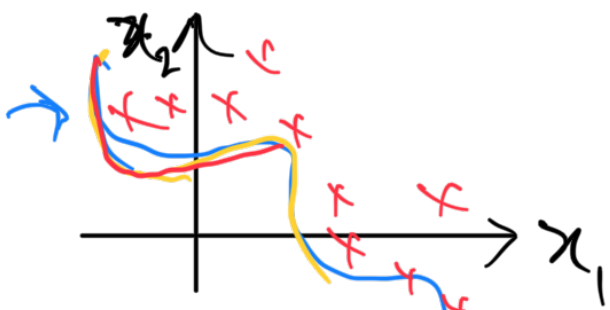
$$= -[y_0 \times \log f_0 + y_1 \times \log f_1 + y_2 \times \log f_2]$$

$$= -[0 \times \log 0.016 + 0 \times \log(0.631) + 1 \times \log(0.353)]$$

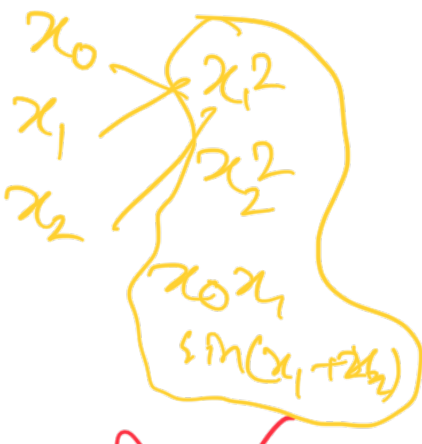
1·34, 2·06, 0·08 ⇒ $J(\vec{w}) = \frac{1}{4}[1.04 + 1.34 + 2.06 + 0.08]$

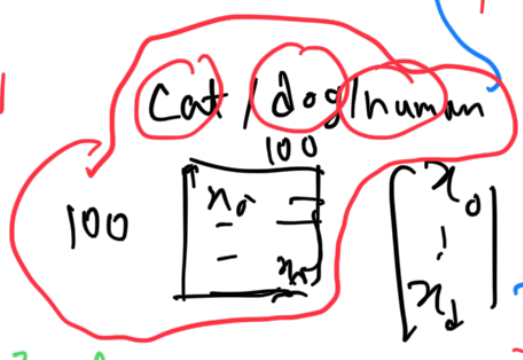
Neural Networks (NN)

Why?



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_1^2 x_2 \\ \sin(x_1 + x_2) \\ \vdots \end{bmatrix}$$





$\begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix}$ → 10000

$x_0^2, x_0 x_1, x_0 x_2, \ldots$

$\left(\frac{n^2}{2}\right)$ → 50×10^6 !!

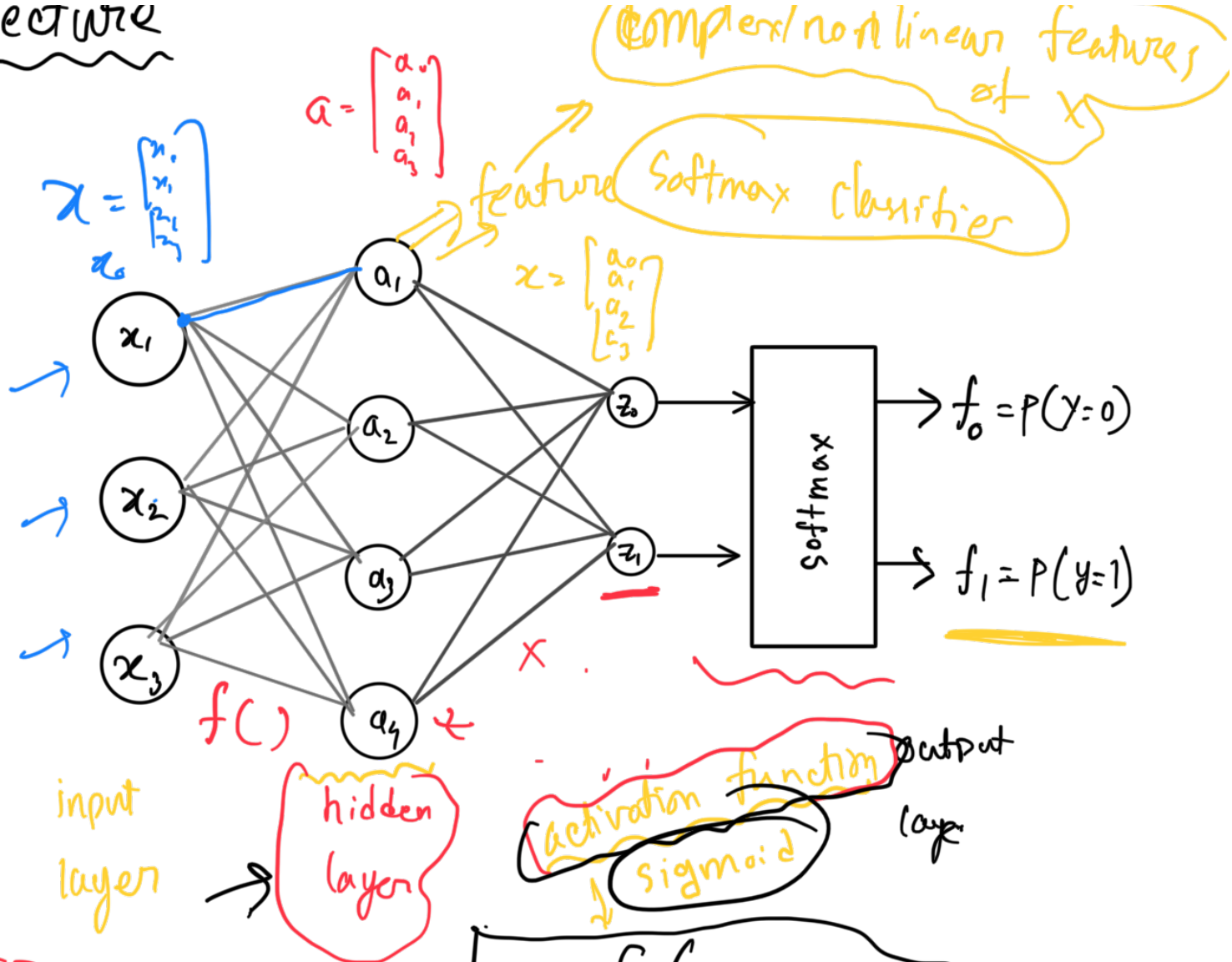
→ NN are universal function approximator

→ instead of increasing length, we add depth

→ Deep neural networks ~ can learn more complex functions, more acc

→ Learn feature themselves!

Architecture



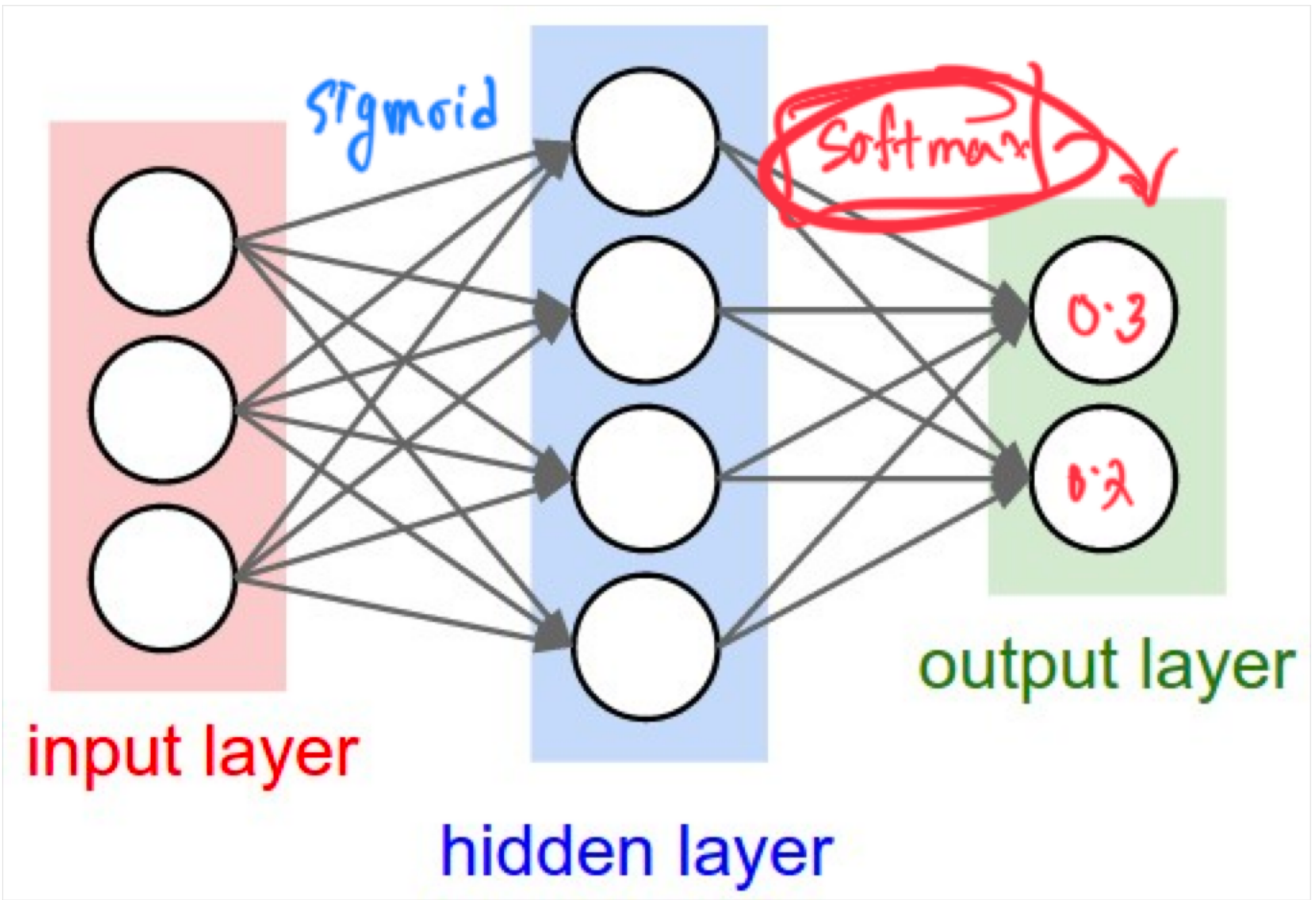
$$a_1 = f(\theta_{00} + \theta_{01} x_1 + \theta_{02} x_2 + \theta_{03} x_3)$$

$$a_2 = f(\theta_{10} + \theta_{11} x_1 + \theta_{12} x_2 + \theta_{13} x_3) = f(\theta_1^T x)$$

$$a = f(W_1 x)$$

$$z = W_2 a$$

$$f = f'(z) = \text{Softmax}(z)$$



More architectures a

$\rightarrow a = g(W_1 x)$

$g(z)$ = non-linearity = activation $= \frac{1}{1+e^{-z}}$

$\rightarrow b = g(W_2 a)$

$F = f(W_3 b)$ $\quad f = \text{Softmax}$

$= \begin{bmatrix} f_0 \\ f_1 \\ \vdots \end{bmatrix} \rightarrow$

$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left[- \sum_{k=0}^{k_1} y_k \log f_k \right]$

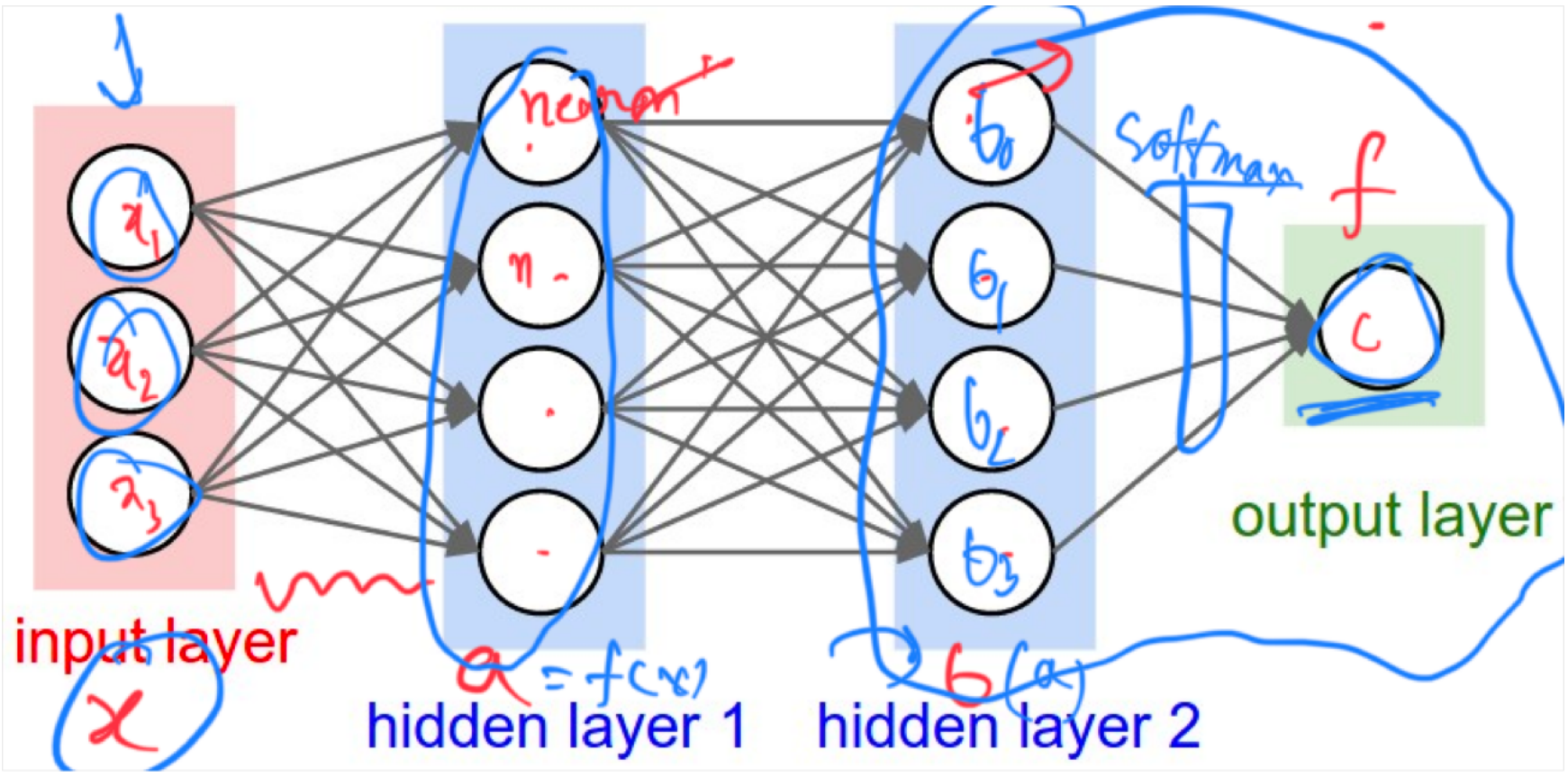
G.D

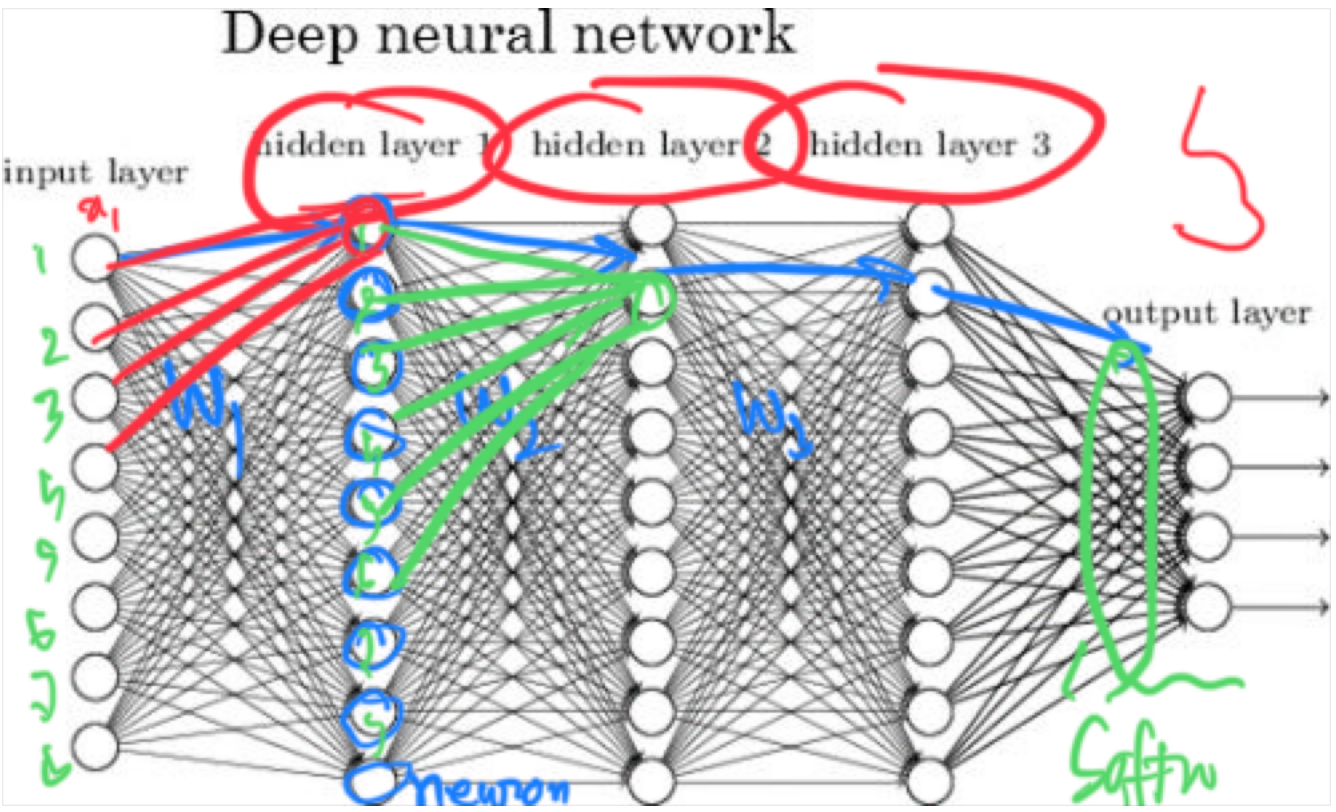
W_1 $\nabla_1 J$

$W_1^{k+1} = W_1^k - \nabla_{W_1} J$

$W_2^{k+1} = W_2^k - \nabla_{W_2} J$

Chain rule


input layer
hidden layer 1
hidden layer 2
output layer
Softmax


Deep neural network
input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

M
N
M
M

Training neural networks

- artificial neural networks
- feed forward " "
- fully connected "
- Dense "

9x1 — 9x8 — 8x1

$$h_1 = g(W_1 x)$$

ReLU

g = ~~sigmoid~~

non-linearity / activation

$$h_2 = g(W_2 h_1)$$

$$h_3 = g(W_3 h_2)$$

final feature vector

$$z = W_4 h_3$$

f = Softmax

$$F = f(z) = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{k-1} \end{bmatrix} \rightarrow P(y=1)$$

z = fun of W_4

$\frac{\partial J}{\partial W_4}$ $\frac{\partial J}{\partial W_1}$ = ~~???~~

$$J(W_1, W_2, W_3, W_4)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left[-\sum_{k=0}^{K-1} y_k \log f_k \right]$$

Cross-entropy

log

Gradient Descent

$$W_1^{(k+1)} = W_1^{(k)} - \nabla_{W_1} J$$

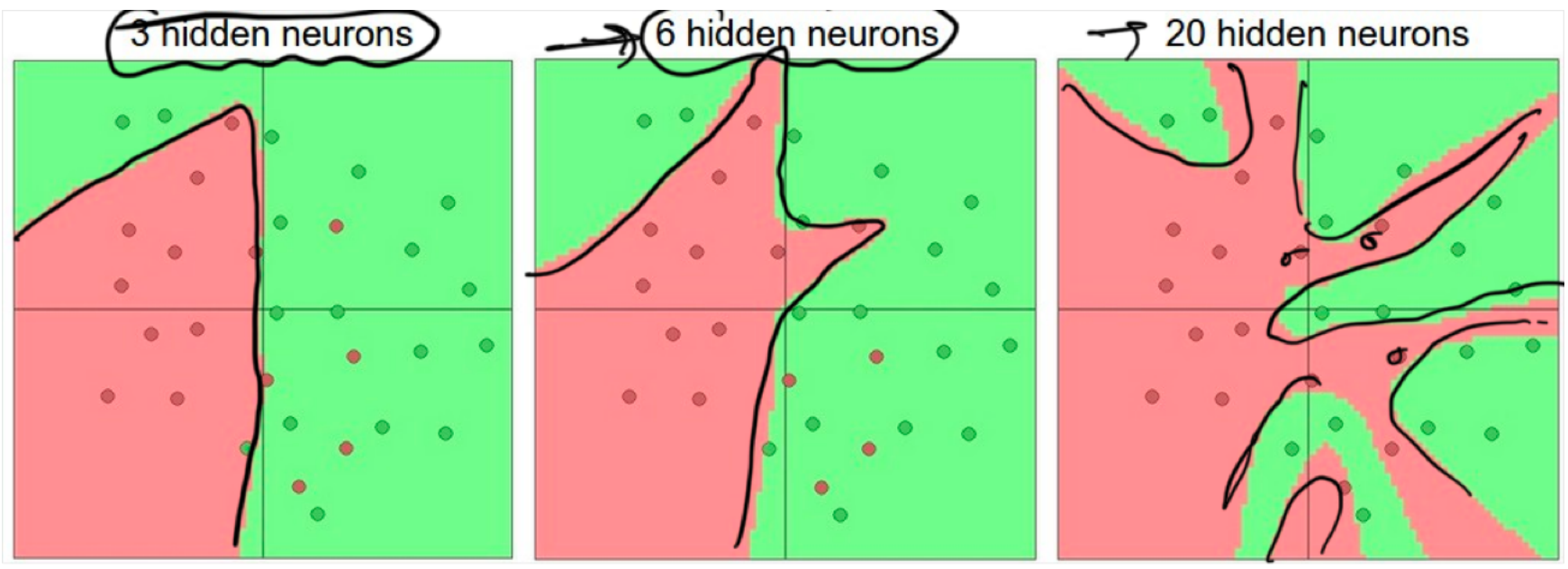
$y = f(g(x))$

$$\frac{dy}{dg} = \frac{dy}{df} \times \frac{df}{dg}$$

$$w_2' = w_2 - \nabla_{w_2} J$$

Practical consideration

① Number of neurons

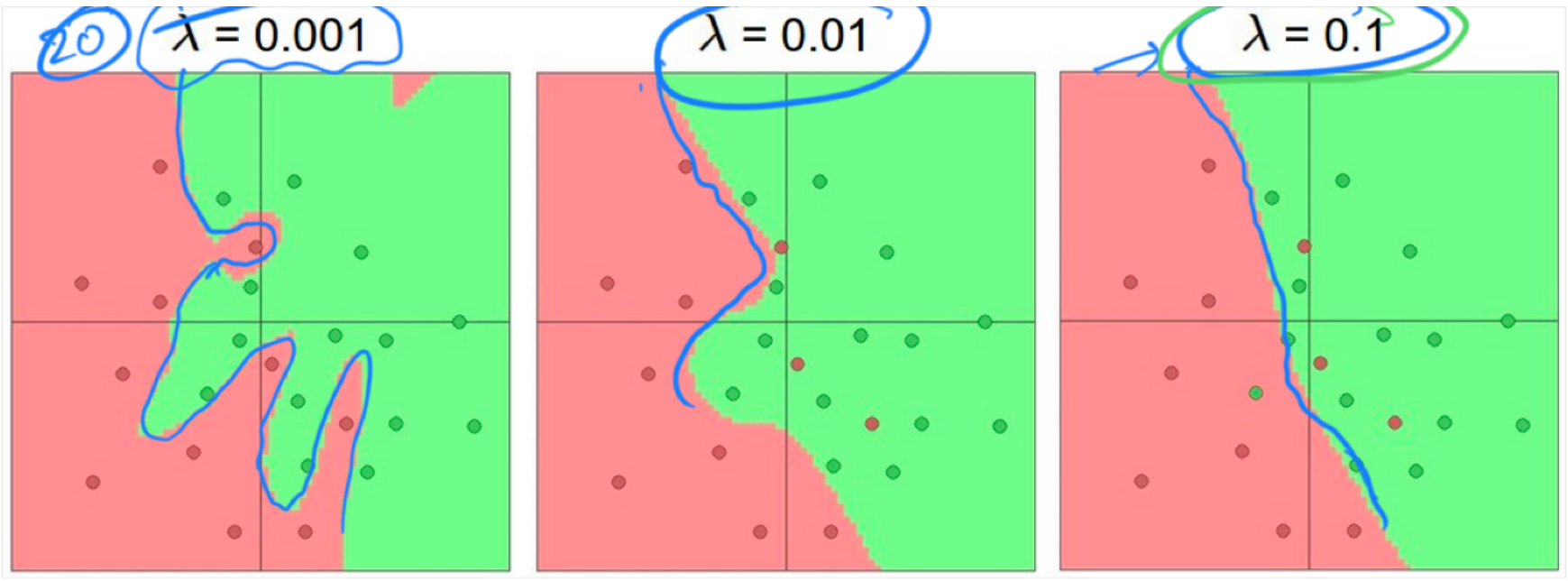


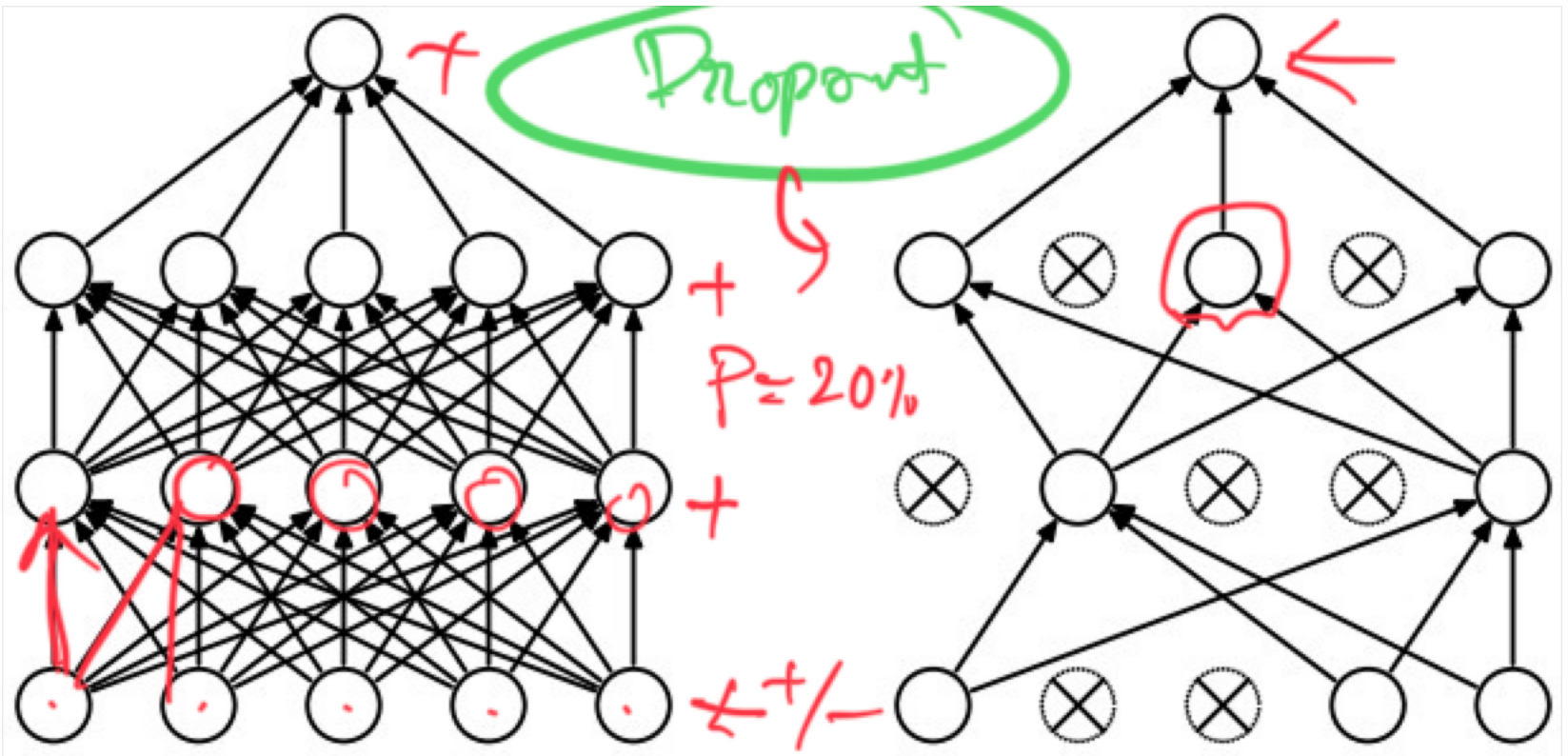
validation set! must

② Overfitting → Regularization

$$J(\theta) + \alpha \sum |\theta|^2$$

α: λ

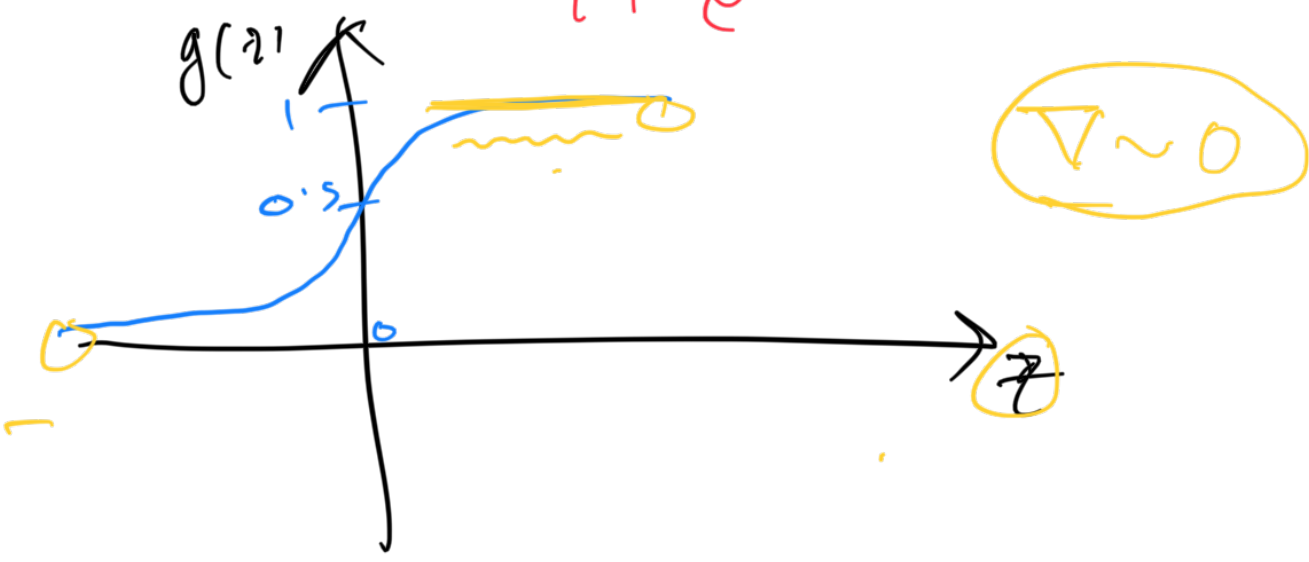




(a) Standard Neural Net (b) After applying dropout.

Activation function

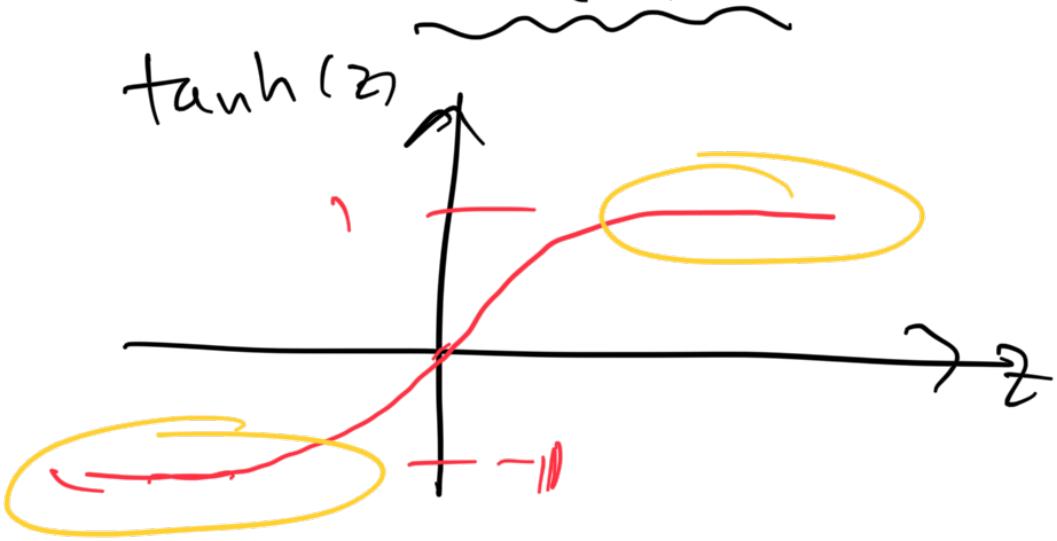
$$g(z) = \frac{1}{1 + e^{-\theta^T x}}$$



$V \sim 0$

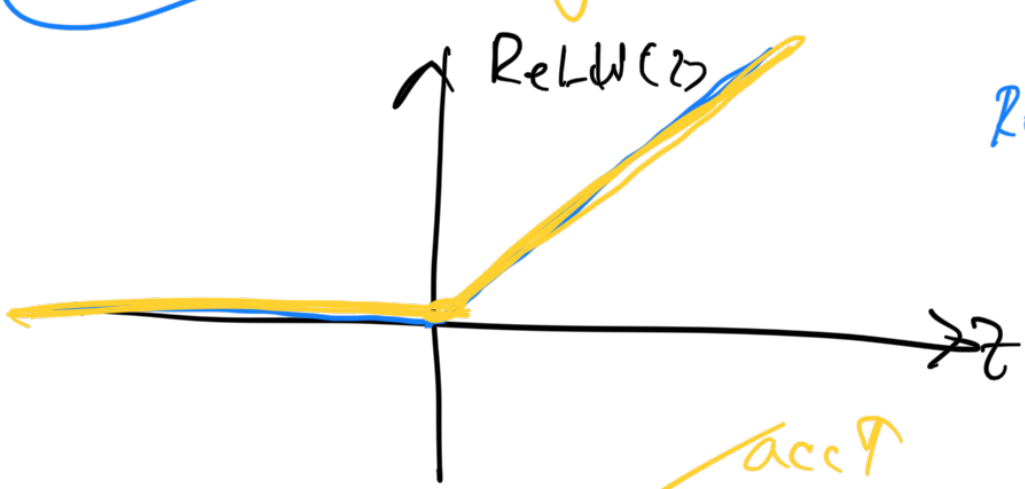
tanh

$4\sigma(2x) - 1$



h

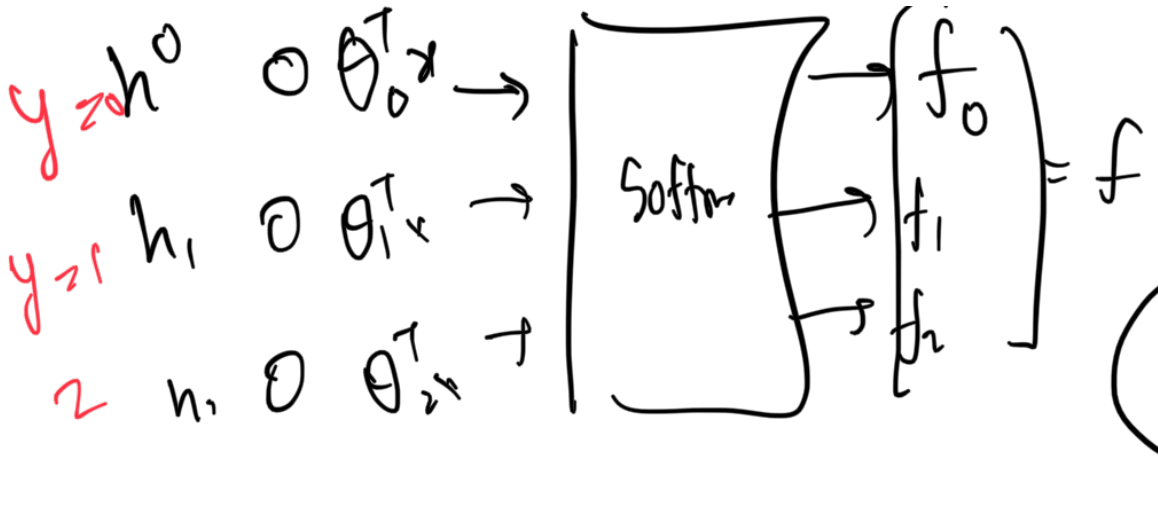
ReLU / Leaky ReLU



Relu(z) → z if z ≥ 0, → 0

acc ↑

speed ↑



$$\mathcal{L}_i = -\sum_{j=0}^{K-1} y_{ij} \log(f_j)$$

i → i'th sample

j → j'th component of y

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

y_i = argmax y

or equivalently

$$L_i = -f_{y_i} + \log \sum_j e^{f_j}$$