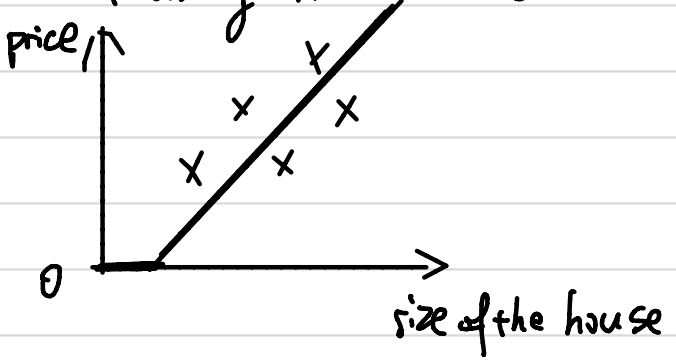
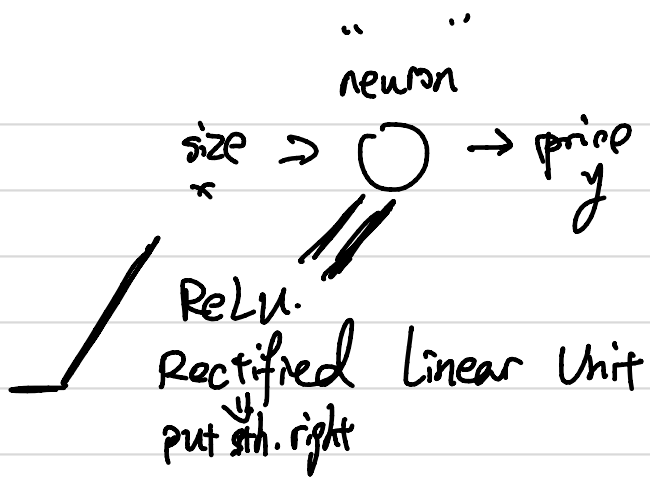
Andrew Ng.

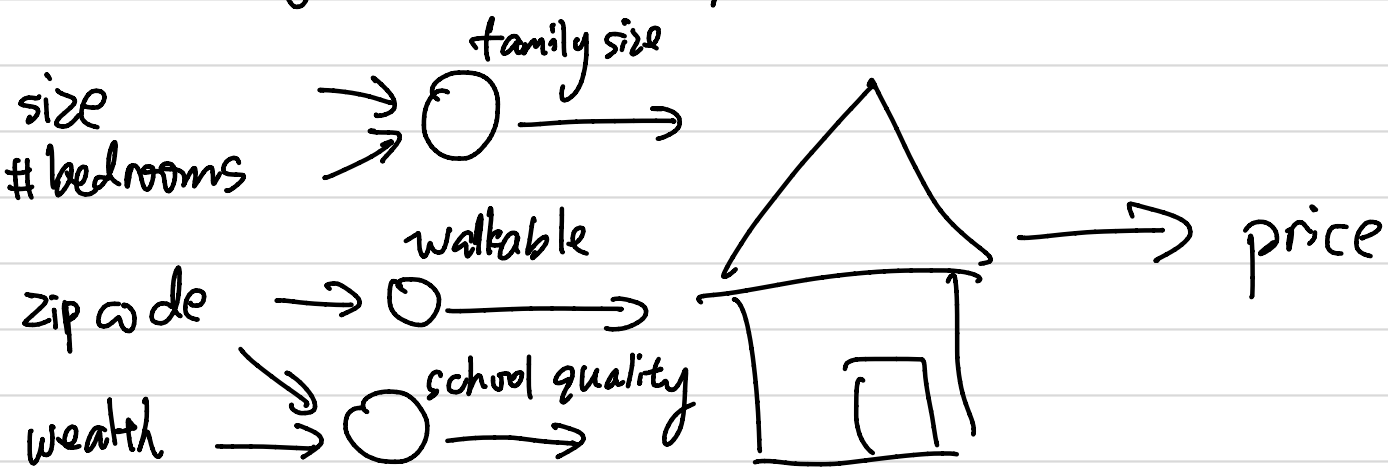
Deep learning class on Coursera

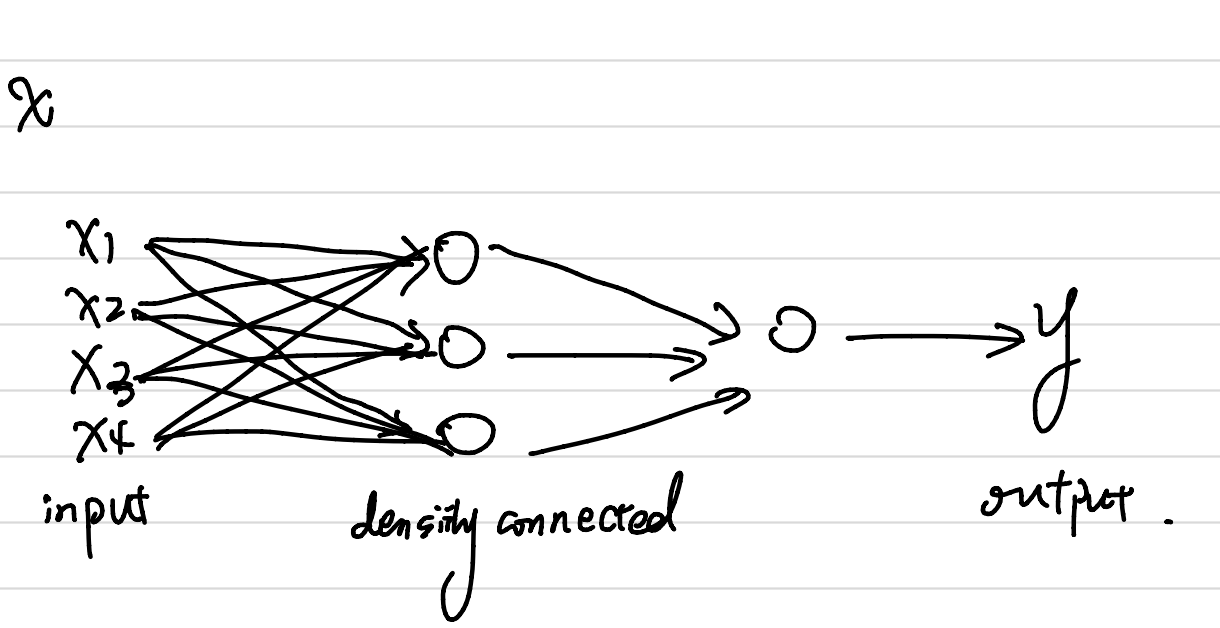
Housing Price Prediction





Here neuron just this linear regression. we have thousands of neurons.





Supervised Learning

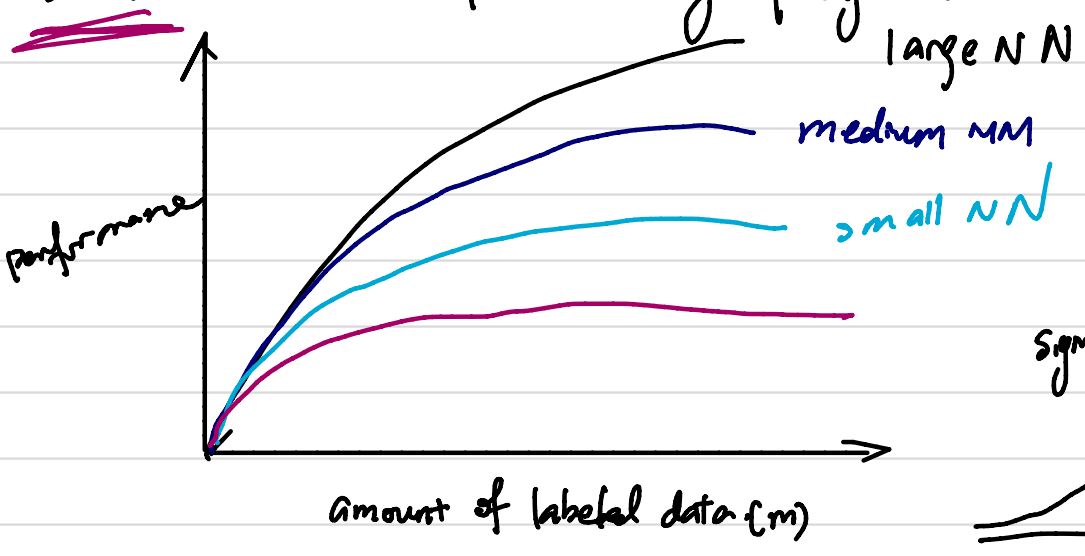
convolutional neural network for images.

machine learning.

structured Data
feature has defined meanings.
ie. number. price. size

unstructured data
audio. image. text

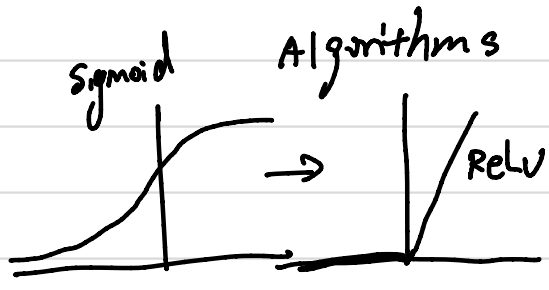
scale drives deep learning progress



Data

Computation

Algorithms



logistic regression

- Binary classification 1 cat vs 0 not cat

* Training data is implemented to build up a model, while a test data set is to validate the model built.

Logistic Regression

Given x. y: be a cat.

- $\hat{y} = P(y=1|x)$

  given x. what's the prob of y is cat

n_x

$x \in \mathbb{R}$

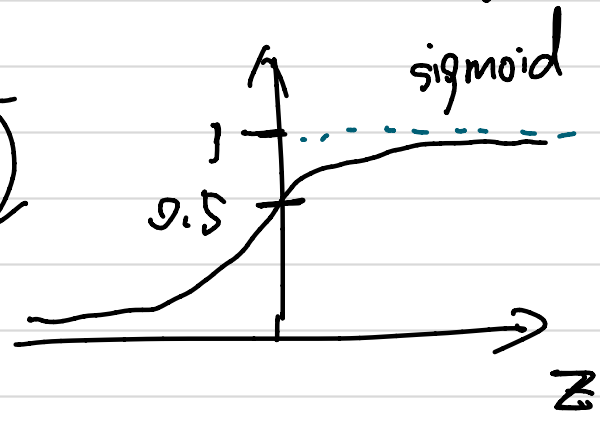
Parameter : $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$. $0 \le \hat{y} \le 1$

Output $\hat{y} = \underbrace{w^T x + b}_{z}$ this value can be negative or very big, which contradicts $0 \le \hat{y} \le 1$

$z = w^T x + b$ 是个酱油. $\hat{y} = \sigma(z)$才是我们要估计的y值.

⇓

$\hat{y} = \sigma(\overbrace{w^T x + b}^{z})$



Formula for sigmoid function:

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

If z large, $\sigma(z) \approx \frac{1}{1+0} = 1$

If z small, $\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{Bignumber}} \approx 0$

Cost function

training data : Given $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

($\hat{y}^{(i)}$: value from the algorithm; $y^{(i)}$: the real training y value)

$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$

Loss function / error function: single training example.

measure how good our output $\hat{y}$ is when the true label is y.

~~$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$~~ 不好. 因为真实情况是 not ∪.

$$\mathcal{L}(\hat{y}, y) = -\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right)$$

want to as small as possible.

If $y=1$. $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ → want $\log \hat{y}$ small, want $\hat{y}$ small (small)

If $y=0$. $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ → want $\log(1-\hat{y})$ small. want $\hat{y}$ large (small)

Cost function

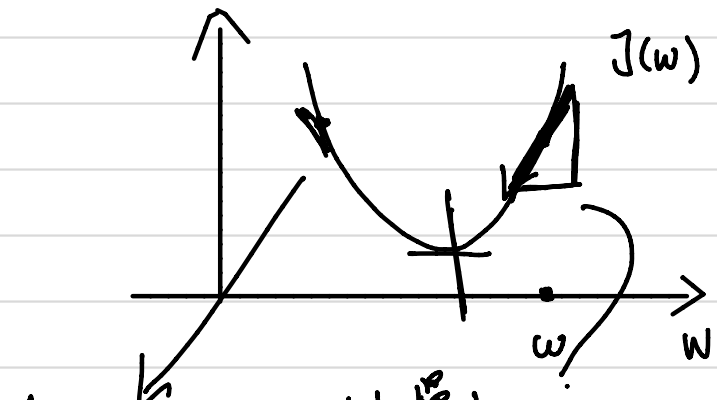
measure how good for entire (training) data set.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

we want to find the w, b to minize the error for the whole training data

⇓ for convex ∪

Gradient Descent



$J(w,b)$

ignore b for now

Gradient Descent

Repeat {

$w := w - \alpha \cdot \frac{dJ(w)}{dw}$ } code: "dw"

learning rate, controls how big each step we take

update to

update change

derivative

what we write in code

$w := w - \alpha\, dw$

$b := b - \alpha \frac{dJ(w,b)}{db}$

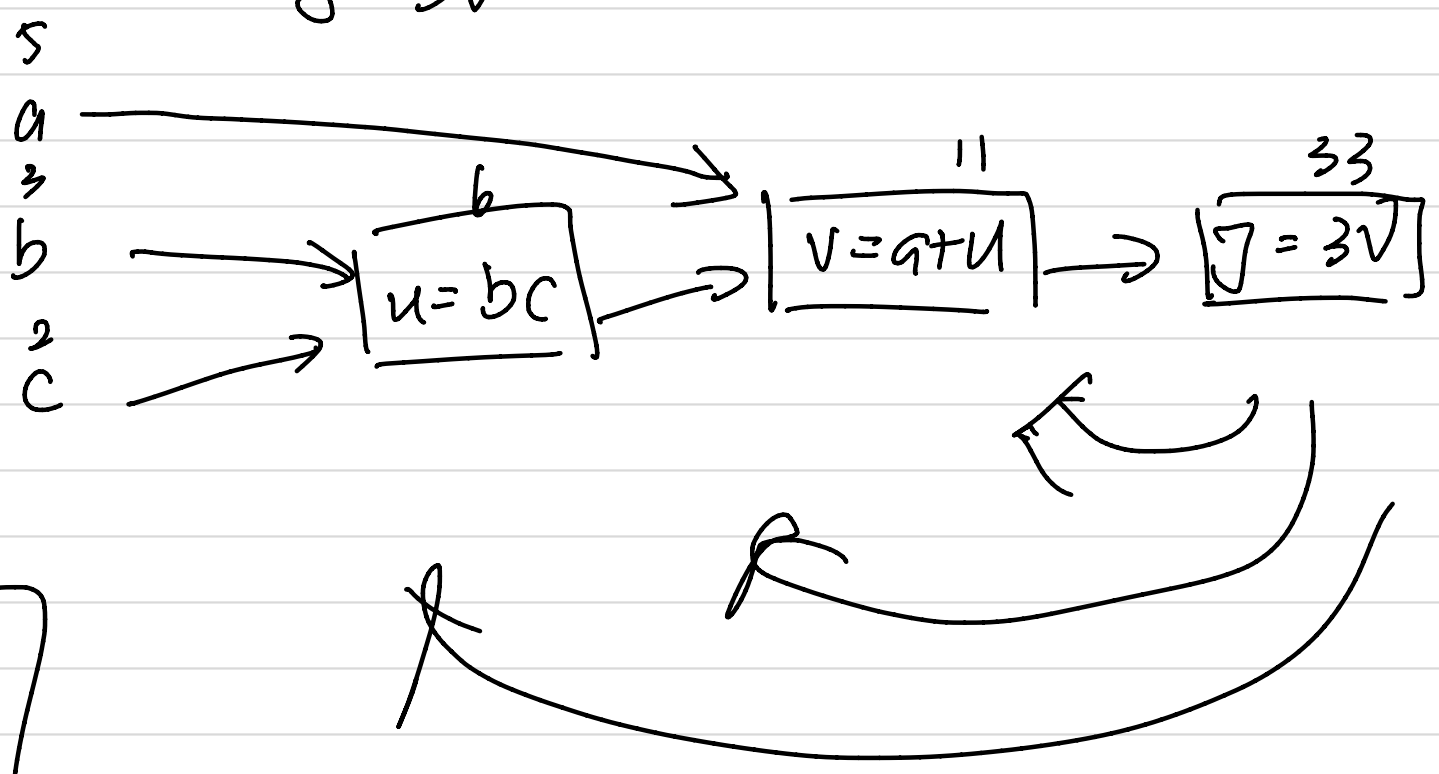
$w := w - \alpha \frac{dJ(w,b)}{dw}$

$\frac{\partial J(w,b)}{\partial w}$ ∂ symbol. J is a function of 2 variables

Computation graph.

$J(a,b,c) = 3(a+bc)$

$u = bc$
$v = a + u$
$J = 3v$



$\frac{dJ}{dv} = 3$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv}$

change a → change v → J

+

$\frac{dv}{da} = 1$

a = 5 → 5.001
v = 11 → 11.001
J = 33 → 33.003

⇒ Chain rule

$$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da}$$

find output variable = coding.

$\frac{d\,\text{FinalOutVar}}{d\text{var}}$ (here is J) in code "dvar"

i.e.

$\frac{dJ}{da}$ in code "da"

$$\frac{dJ}{du} = 3 = \frac{dJ}{dv}\cdot\frac{dv}{du}$$

3 1

$u = 6 \to 6.001$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$$\frac{dJ}{db} = \frac{dJ}{du}\cdot\frac{du}{db} = b$$

3 $=2$

$b = 3 \to 3.001$

$u = b\cdot c \to 6.002$ $(c=2)$

(in code db)

$$\frac{dJ}{dc} = \frac{dJ}{du}\cdot\frac{du}{dc} = 9.$$

3 3

Logistic Regression Recap.

$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

$\mathcal{L}(a,y) = -(y\log(a) + (1-y)\log(1-a))$. loss function

example.

only 2 features x_1, x_2.

w_1, b w_2, b

$z = w_1x_1 + w_2x_2 + b. \to y = a = \sigma(z) \to \mathcal{L}(a,y)$

求导说，给阿狸.

$$\frac{dz}{} = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz}$$

$$= a - y.$$

$$= \frac{d\mathcal{L}}{da}\cdot\frac{da}{dz} \to a(1-a)$$

$$\because da'' = \frac{d\mathcal{L}(a,y)}{da} \leftarrow$$

$$= \frac{-y}{a} + \frac{1-y}{1-a}$$

$$\left(\frac{-y}{a} + \frac{1-y}{1-a}\right)\cdot a(1-a)$$

最后要的是 $dw_1 = x_1\cdot dz$ $dw_2 = x_2\cdot dz$ $db = dz$

$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{da}\cdot\frac{da}{dz}\cdot\frac{dz}{dw_1}$

$$\frac{d\mathcal{L}}{dw_1} = dw_1 = x_1\,dz.$$

$dw_2 = x_2\cdot dz$ $db = dz$

$b := b - \alpha db$

update

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

Logistic on m examples

$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y)$ where $a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$

$dw_1^{(i)}$ $dw_2^{(i)}$

$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w_i} \mathcal{L}(a^{(i)}, y^{(i)})$

$dw_1^{(i)} - (x^{(i)}, y^{(i)})$

example:

$J=0$, $dw_1=0$. $dw_2=0$. $db=0$

For $i=1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 += x_1^{(i)} dz^{(i)}$
$dw_2 += x_2^{(i)} dz^{(i)}$
$db += dz^{(i)}$
$n=2$

$J /= m$ 平均 average

$dw_1 /= m$ $dw_2 /= m$

$db /= m$

$dw_1 = \frac{\partial J}{\partial w_1}$

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

(explicit)

* for loop needs more time to execute (相对 for loop)

Your code needs to be efficient ⇒ vector

① 2 for loops

(i) data 个数. $i = 1 \sim m$

n 是 feature 个数.

②

Vectorization

$z = w^T x + b$ $\quad w = \begin{bmatrix} \vdots \end{bmatrix}$ $\quad x = \begin{bmatrix} \vdots \end{bmatrix}$ $\quad x \in \mathbb{R}^n$ $\quad x \in \mathbb{R}^n$

Non-vectorize

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

```
z = np.dot(w,x) + b
```

$w^T x$

Code:

```
import numpy as np
a = np.array([1,2,3,4])
print(a)
```

```
import time
a = np.random.rand(1000000)
b = np.random.rand(1000)
tic = time.time()
c = np.dot(a,b)
toc = time.time()
print("Vectorized version:" + str(1000*(toc-tic)) + "ms")
```

avoid explicit for loop

$V = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ $\quad u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$

non-vectorization

```
u = np.zeros((n,1))
for i in range(n):
    u[i] = math.exp(v[i])
```

vectorization

```
import numpy as np
u = np.exp(v)
```

* `np.log(v)`
  `np.abs(v)`
  `np.maximum(v, 0)`

Broadcasting.

m 个 data (attributes) $x^{(1)}, x^{(2)}, \cdots x^{(m)}$

$z^{(1)} = w^T x^{(1)} + b, \quad z^{(2)} = w^T x^{(2)} + b, \quad z^{(3)} = w^T x^{(3)} + b.$ (T: transpose)

$y = a^{(1)} = \sigma(z^{(1)}) \quad a^{(2)} = \sigma(z^{(2)}) \quad a^{(3)} = \sigma(z^{(3)})$

$$Z = [z^{(1)}, z^{(2)}, \cdots, z^{(m)}] = W^T X + [b, b, \cdots b]_{1\times m} = [W^T x^{(1)} + b, \cdots, W^T x^{(m)} + b]$$

(transpose; $m\uparrow$)

`Z = np.dot(W.T, X) + b` 这里写 b，自动生成 vector 配 numpy. "Broadcasting"

$$A = [a^{(1)}, a^{(2)}, \cdots, a^{(m)}]$$

Review:

$dz = a - y$

$\therefore\ dz^{(1)} = a^{(1)} - y^{(1)}$ $dz^{(2)} = a^{(2)} - y^{(2)}$ ... ...

$$dz = [dz^{(1)}\ dz^{(2)} \cdots dz^{(m)}]_{1\times m}$$

$$A = [a^{(1)} \cdots a^{(m)}] \qquad Y = [y^{(1)} \cdots y^{(m)}]$$

$$dz = A - Y = [a^{(1)} - y^{(1)},\ a^{(2)} - y^{(2)},\ \cdots].$$

From before =

```
dw = 0
dw += x^(1) dz^(1)
dw += x^(2) dz^(2)
  ⋮
dw /= m
```

```
db = 0
db += dz^(1)
db += dz^(2)
  ⋮
db /= m
```

$\rightarrow\ db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)}$

$= \frac{1}{m}$ np.sum(dz)

$$dw = \frac{1}{m} X\, dz^T$$

$$= \frac{1}{m}\begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix}_{n\times m} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}_{m\times 1}$$

$$= \frac{1}{m}\left[x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)}\right].$$

$$Z = W^T X + b$$

$= $ np.dot(w.T, X) + b

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X\, dZ^T$

$db = \frac{1}{m}$ np.sum(dZ)

$w = w - \alpha\, dw$ (learning rate)

$b = b - \alpha\, db$

A = np.array([[---],
[ ],
[ ]])

cal = A.sum(axis=0). 一列一列相加.

Broadcasting example

$\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}$ + 100 ⟹ in python will be $\begin{bmatrix}100\\100\\100\\100\\100\end{bmatrix}$ ⇒ $\begin{bmatrix}101\\102\\103\\104\\105\end{bmatrix}$

$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}_{(m,n)} + \begin{bmatrix}100&200&300\end{bmatrix}_{(1,n)} \Rightarrow \begin{bmatrix}101&202&303\\104&205&306\end{bmatrix}$ python: copy to turn into (m,n)

$\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}_{(m,n)} + \begin{bmatrix}100\\200\end{bmatrix}_{(m,1)} \Rightarrow \begin{matrix}100&100&100\\200&200&200\end{matrix} \Rightarrow \begin{bmatrix}101&102&103\\204&205&206\end{bmatrix}$

(m,1) 也会 copy 成 (m,n)

(m,n) matrix ± / (1,n) copy*m ⇝ (m,n)

(m,1) copy*n ⇝ (m,n).

(m,1) + 100 copy*m = $\begin{bmatrix}101\\102\\103\end{bmatrix}$

$\begin{bmatrix}1\\2\\3\end{bmatrix}$

$\begin{bmatrix}1&2&3\end{bmatrix} + 100 = \begin{bmatrix}101&102&103\end{bmatrix}$.

More About Cost Function

Here we only talk about logistic regression! So that's why y=1 or y=0 只有两种可能!

$\hat{y} = \sigma(w^T x + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$.

$\hat{y} = P(y=1|x)$

If y=1: $P(y|x) = \hat{y}$

If y=0: $P(y|x) = 1-\hat{y}$ } $P(y|x)$.

$$P(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$$

可看作 Bernoulli (非成功即失败)

If $y=1$. $p(y|x) = \hat{y}^{1}\cdot(1-\hat{y})^{0} = \hat{y}$

If $y=0$. $p(y|x) = \hat{y}^{0}(1-\hat{y})^{1} = 1\times(1-\hat{y}) = 1-\hat{y}$

$$\log P(y|x) = \log \hat{y}^{y}(1-\hat{y})^{(1-y)} = y\log\hat{y} + (1-y)\log(1-\hat{y})$$

$$= -\mathcal{L}(\hat{y}, y)$$

too find parameter to make this as big as possible

maximize likelihood.

$$\log p(\text{labels in training set}) = \log \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$$

$$+\log$$

$$\Rightarrow \sum_{i=1}^{m} \underbrace{\log P(y^{(i)}|x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$$

$$= -\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

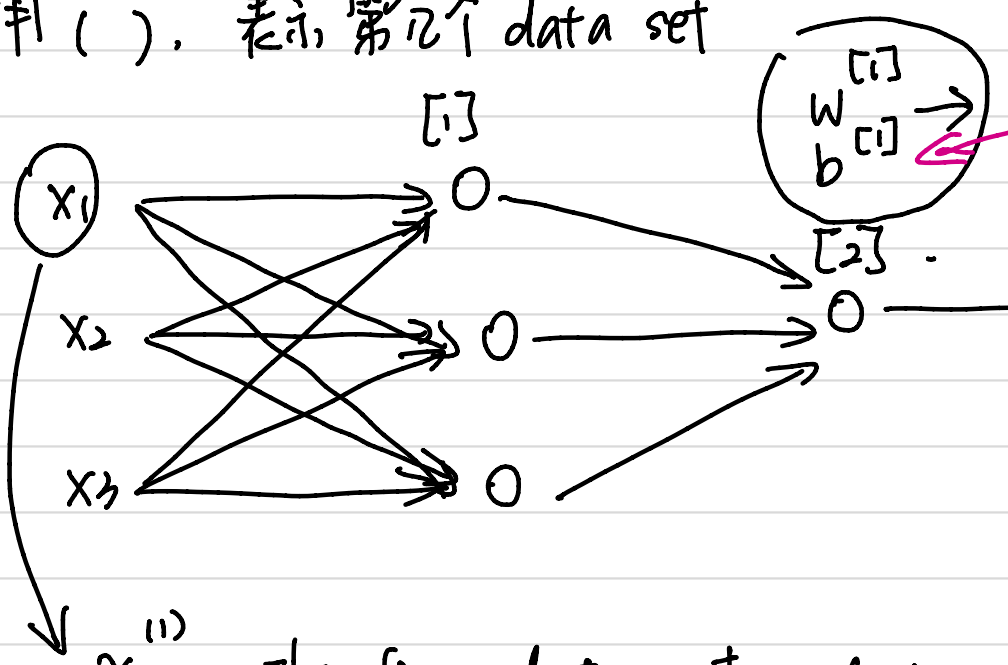
去完负号，要越小越好.

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

$=$ average, scaling

(minimize) 拿负号.

用 [ ]. 表示 layer
用 ( ). 表示第几个 data set



$$z^{[1]} = W^{[1]} x + b^{[1]} \rightarrow a^{[1]} = \sigma(z^{[1]}) \rightarrow$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]} \rightarrow a^{[2]} = \sigma(z^{[1]}) \rightarrow$$

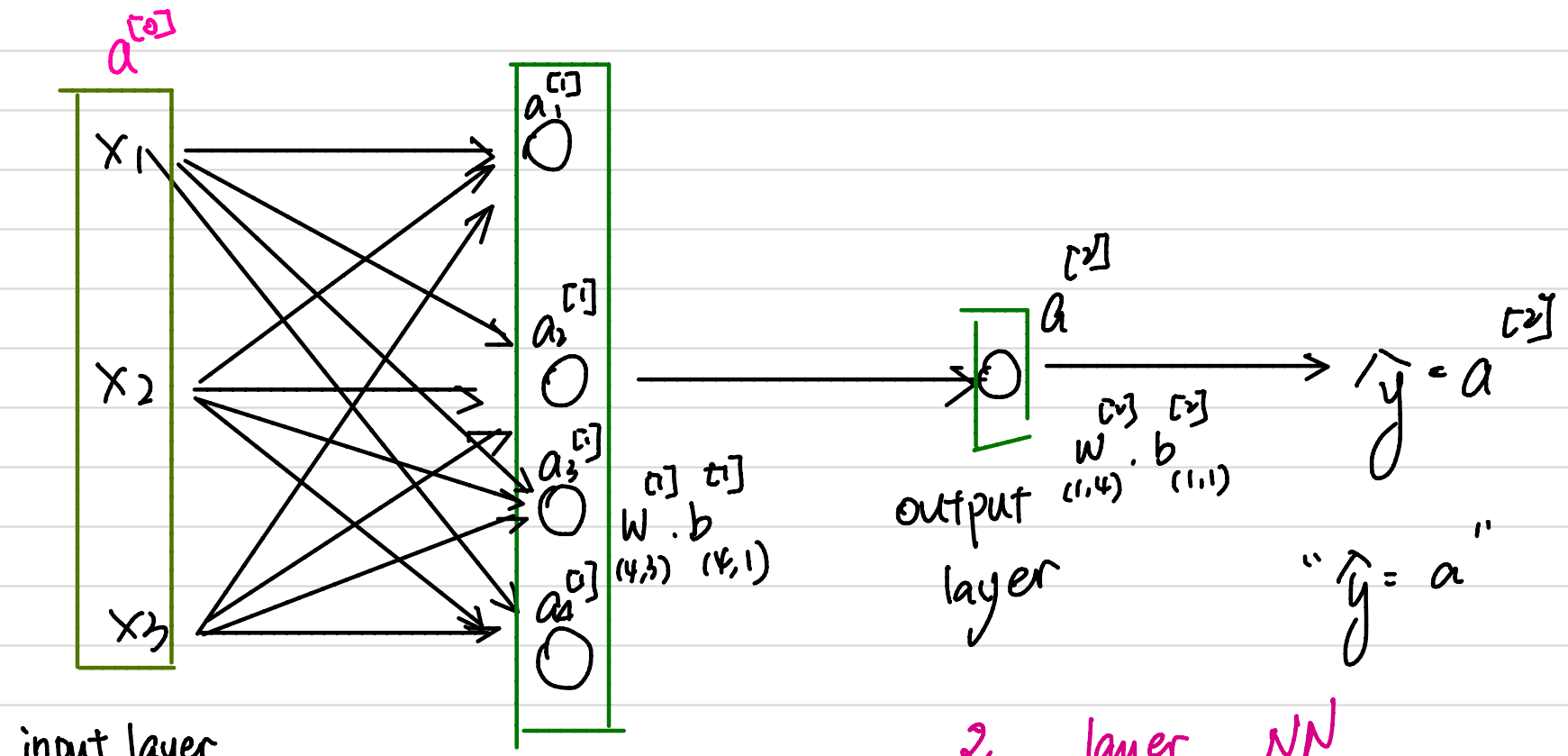
$$\rightarrow L(a^{[2]}, y)$$

$\hat{y}$

$dz^{[2]}$ $da^{[2]}$

$x^{(1)}$: The first data set (The quantity of one data set is the number of data)

backward calculation is derivative

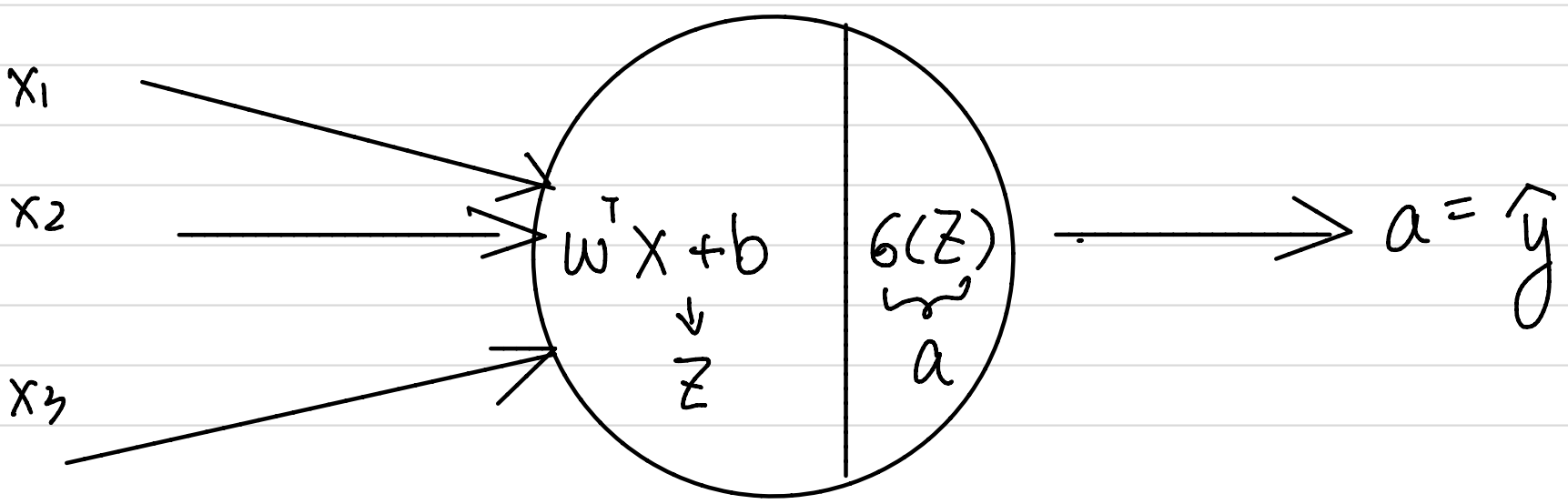


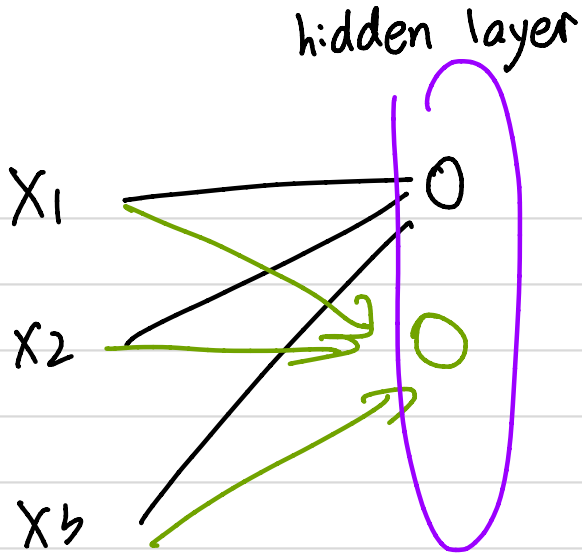
$a^{[0]} = X$

$a^{[1]}$

(activation)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$





$W^T = W^{[1]}$

$b = b^{[1]}$

$W^{[2]}$, $b^{[2]}$

$(1,4)$ $(1,1)$

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$a_i^{[l]}$ ← layer

← node in layer.

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}$$

$$a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}$$

$$a_4^{[1]} = \sigma(z_4^{[1]}).$$

$W^{[1]}$

To Matrix:

$$z^{[1]}_{(4,1)} = \underset{4\times 3}{\begin{bmatrix} \text{---} & w_1^{[1]T} & \text{---} \\ \text{---} & w_2^{[1]T} & \text{---} \\ \text{---} & w_3^{[1]T} & \text{---} \\ \text{---} & w_4^{[1]T} & \text{---} \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}} + \underset{4\times 1}{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}} =$$

$$\begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{(1)} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$$a^{[1]}_{4\times 1} = \begin{bmatrix} a_1^{[1]} \\ \\ \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]}).$$

4×1

$$z^{[2]}_{(1,1)} = \underset{(1,4)}{W^{[2]}}\ \underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$a^{[2]}_{(1,1)} = \sigma(z^{[2]})$$

$x \rightarrow a^{[2]} = \hat{y}$

$x^{(1)} \rightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \rightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots \qquad \vdots \qquad \vdots$

$x^{(m)} \rightarrow a^{[2](m)} = \hat{y}^{(m)}$

$a^{[2](i)}$ ← example i^{th}

a ↓

layer 2

Use matrix broadcasting.

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$a^{[1]} = g(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(Z^{[2]})$$

We define the matrix to be equal to our training examples stacked up in these columns like so. So take the training examples and stack them in columns. So this becomes a n_x × m diminish the matrix

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

(1 2 3 4 ⋮ 10) 身高 体重 肺活量

example: 10个人测身高体重，肺活量。

所以这个matrix是 ~~3×10~~ 10×3

3 ↔ attributes个数　10 ↔ 人个数

For loop: for i=1 to m,

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = g(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = g(z^{[2](i)})$$

↓ Vectorizing.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \begin{bmatrix} x_{11} & x_{21} & \cdots & x_{m1} \\ x_{12} & \vdots & & \vdots \\ \vdots & & & \\ x_{1n_x} & x_{2n_x} & \cdots & x_{mn_x} \end{bmatrix}$$

(n_x, m)

Capital:

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = g(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = g(Z^{[2]})$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

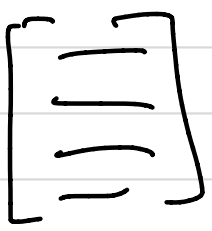
$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

- first hidden unit on the first training example
- second hidden unit on the first training example

hidden unit Number

i^{th} training example.

$z^{[1](1)} = w^{[1]} x^{(1)} + b^{[1]}$, $z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]}$, $z^{[1](3)} = w^{[1]} x^{(3)} + b^{[1]}$

$w^{[1]} =$ 

$w^{[1]} x^{(1)} =$ 

$w^{[1]} x^{[2]} =$ 

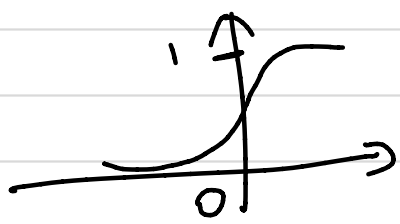
$w^{[1]} x^{(3)} =$ 

$$W^{[1]} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}_{4\times 3} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix}$$

$= Z^{[1]}$

Activation functions.

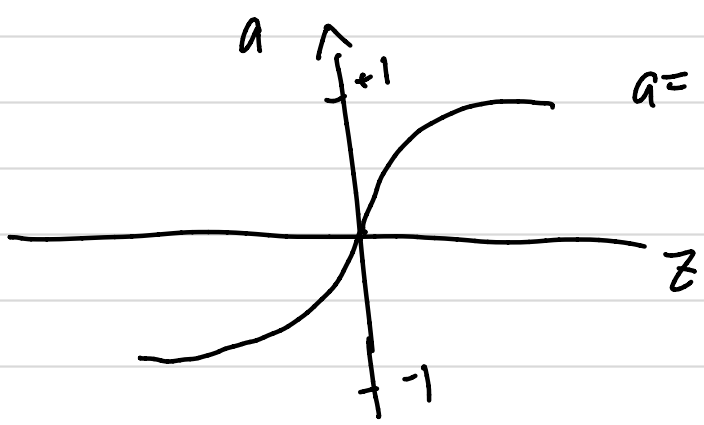
① sigmoid
② tanh
③ ReLU

sigmoid 

$a^{[1]} = g(z^{[1]})$

$a^{[2]} = g(z^{[2]})$.

$g(z^{[1]})$

$\Rightarrow$ 

$a = \tanh(z)$

$$a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

(shifted version of sigmoid).

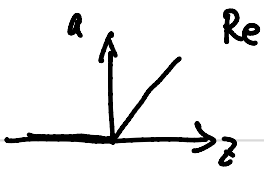
$g(z^{[1]}) = \tanh(z^{[1]})$. so data has a 0 mean.

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

$0 \le \hat{y} \le 1$

hidden layer $g^{[1]}(z^{[1]}) = \tanh(z^{[1]})$. tanh

output layer: $g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$ so use sigmoid for the output layer.



ReLU (Rectified Linear Unit)

$a = \max(0, z)$

leaky Relu

Pros and cons.

sigmoid : never use it (except for output layer sometimes)

we use tanh. $a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

★ the most used is Relu $a = \max(0, z)$

leaky Relu $a = \max(0.01z, z)$

★ Non-linear Regression.

Derivative

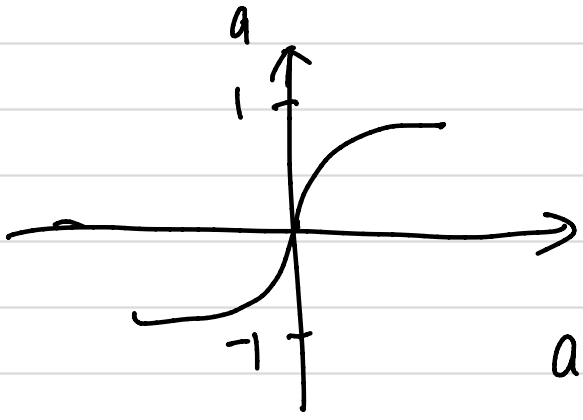
① sigmoid

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

$$z \quad g'(z) = \frac{d}{dz} g(z) = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z)(1 - g(z))$$

$$= a(1-a)$$



② tanh

$$a = g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

$$= 1 - \tanh^2(z)$$

$$= 1 - a^2$$

③ Relu

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0. \\ \text{undefine} & z = 0 \end{cases}$$

④ Leaky Relu

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z > 0. \\ \text{undefine} & z = 0 \end{cases}$$

Your neural network which only has one hidden layer has parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

$(n^{[1]}, n^{[0]})\ (n^{[1]}, 1)\ (n^{[2]}, n^{[1]})\ (n^{[2]}, 1)$

$n_x = n^{[0]}$ input features. $n^{[1]}$ hidden units $n^{[2]}$ output units $= 1$

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{n} L(\hat{y}, y)$

$\hat{y} = a^{[2]}$

Gradient Descent

$\hat{y}^{(i)}$, $i = 1, \cdots, m$

$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$ $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$

$W^{[1]} = W^{[1]} - \alpha dW^{[1]}$ (update)

$b^{[1]} = b^{[1]} - \alpha db^{[1]}$

⋮

I don't know why.

Forward propogation

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$. here we use sigmoid

Backward propogation (derivative)

$dZ^{[2]} = A^{[2]} - Y$ $Y = [y^{(1)}, y^{(2)}, \cdots y^{(m)}]$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

$db^{[2]} = \frac{1}{m}$ np.sum($dZ^{[2]}$, axis=1, keepdims=True)

$(n^{[2]},)$ 默认为 $(n^{[2]}, 1)$.

$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$

$(n^{[1]}, m)$ $(n^{[1]}, m)$

element wise product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

$db^{[1]} = \frac{1}{m}$ np.sum($dZ^{[1]}$, axis=1, keepdims=True)

$(n^{[1]}, 1)$

Derivative. Backward.

x, $W^{[1]}$, $b^{[1]}$ → $z^{[1]} = W^{[1]}x + b^{[1]} \to a^{[1]} = g(z^{[1]}) \to z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \to a^{[2]} = g(z^{[2]}) \to L(a^{[2]}, y)$ ($W^{[2]}, b^{[2]}$)

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$; $da^{[1]}$ column vector

$dz^{[2]} = a^{[2]} - y$

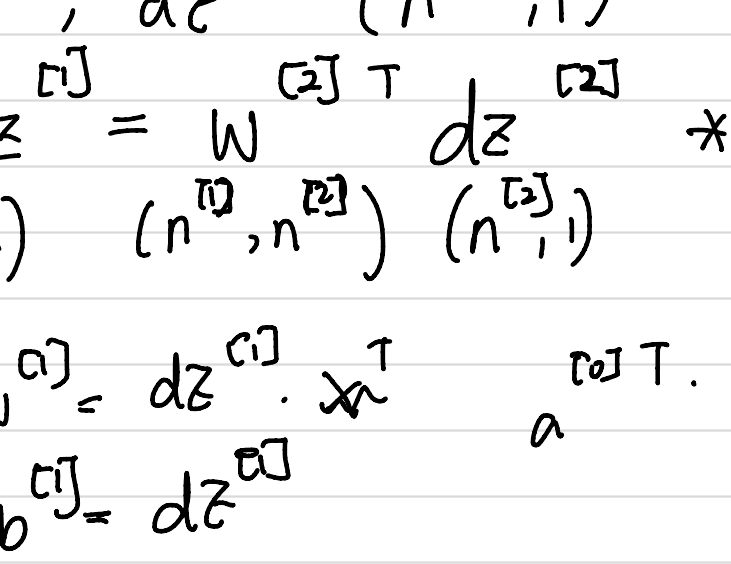
$dW^{[2]} = dz^{[2]} a^{[1]T}$

$db^{[2]} = dz^{[2]}$

$da^{[2]}$

$dW = dz \cdot x^T$

[—] row vector



$n_x = n^{[0]} = 3$ $n^{[1]} = 4$ $n^{[2]} = 1$

($n^{[1]}$: No. of nodes in hidden unit)

$W^{[2]}$ is $n^{[2]} \times n^{[1]}$ dimension

$z^{[2]}, dz^{[2]}$ $(n^{[2]}, 1)$ dimension

$z^{[1]}, dz^{[1]}$ $(n^{[1]}, 1)$

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

$(n^{[1]}, 1)$ $(n^{[1]}, n^{[2]})$ $(n^{[2]}, 1)$ $(n^{[1]}, 1)$.

dimension

$dw^{[1]} = dz^{[1]} \cdot x^T$ $a^{[0]T}$.

$db^{[1]} = dz^{[1]}$

Initialize weights to zero. Weight: W^T.

Random Initialization

建模过程.

每个小module干什么.

小module建大的network.

拆分.

拿一个样本. 画图

一个layer 把 input tenser → output tenser.

样本个数变. 10个data 还是 10个.

breakdown.

CNN image classification

operator.

Question: Does each node (in the same layer) have different w value?

like: $w^{[1]}$ $b^{[1]}$

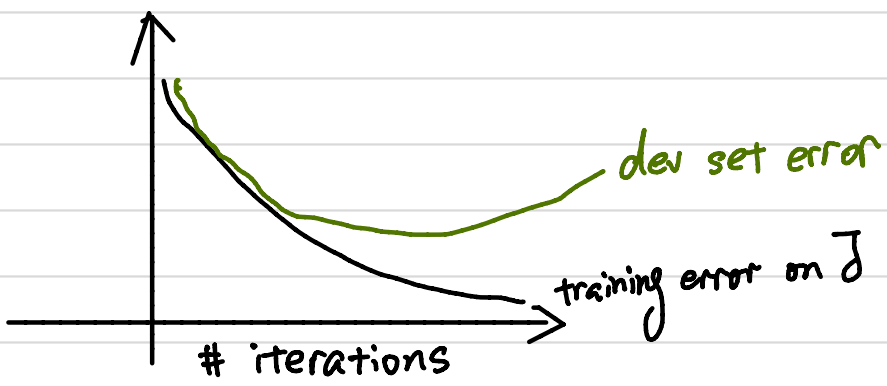
x_1 → $w_1^{[1]}$

x_2 → $w_2^{[1]}$

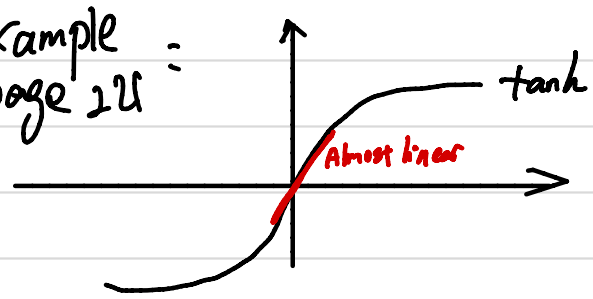
→ $w_3^{[1]}$

x_3 → $w_4^{[1]}$

iteration VS epoch.



Regularization: example on page 21:



$g(z) = \tanh(z)$

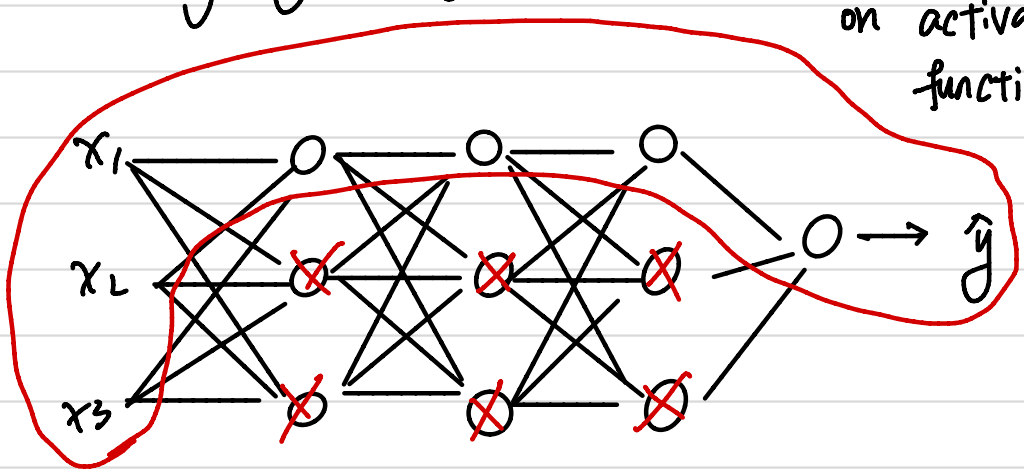
$\uparrow \Rightarrow w^{[l]} \downarrow$ $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$\downarrow$ $\downarrow$

$g^{[l]}(z^{[l]})$ will be roughly linear

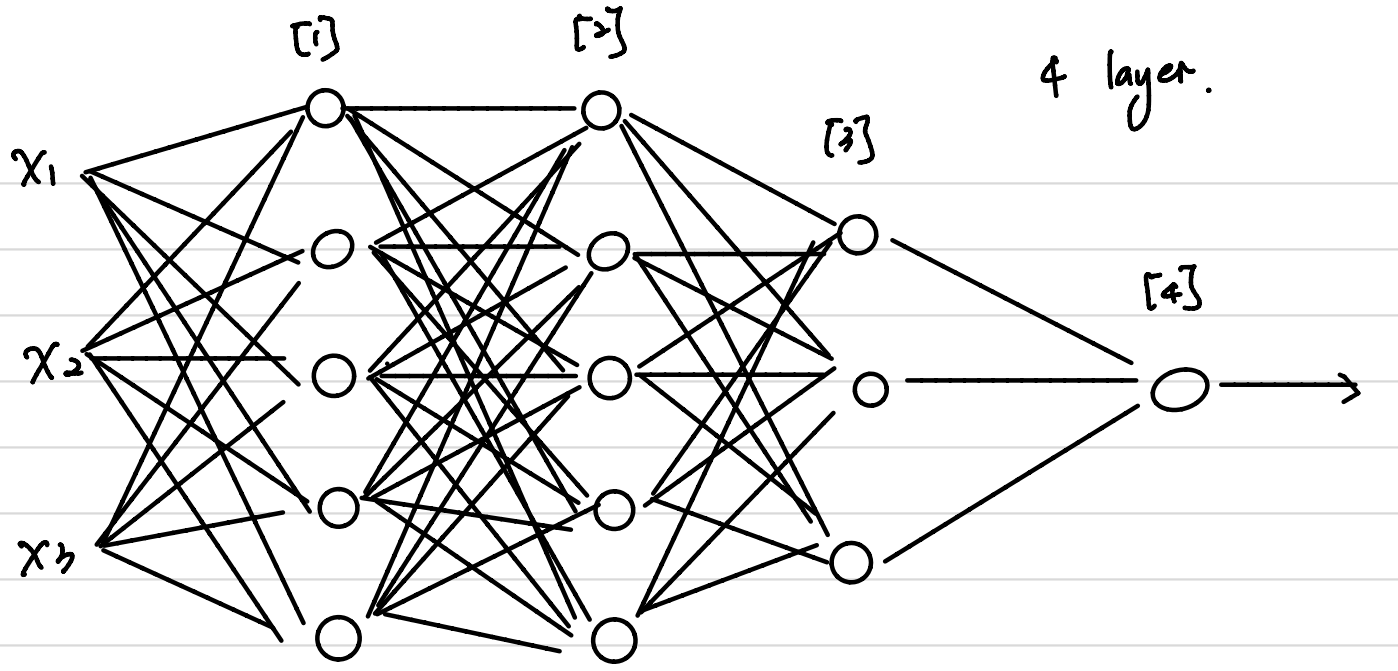
为什么说 $w^{[l]}$ 越小就越趋于 linear.

why say roughly linear, doesn't it need to depend on activation function?



Dev data

原理: zero out units in the layers



4 layer.

$l = 4$

$n^{[l]}$ = # units in layer l $\quad n[0]=3,\ n[1]=5,\ n[2]=5,\ n^{[3]}=3,\ n^{[4]}=1$

$a^{[l]}$ = activations in layer l (即 sigmoid / tan / Relu 公式)

$a^{[l]} = g^{[l]}(z^{[l]})$ (求导. 即 sigmoid 求导 / tanh / Relu 求导)

$w^{[l]}$ = weights for $z^{[l]}$

$b^{[l]}$ = bias for $z^{[l]}$

$[L] \rightarrow$ final layer, use "L"

$a^{[L]} = \hat{y}$

Forward Propogation

X: $a^{[0]} = X$

$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = g^{[2]}(z^{[2]})$

$\Rightarrow$ general $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

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

$a^{[4]} = g^{[4]}(z^{[4]})$

$\Rightarrow$ Vector 用大写

$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[1]}$

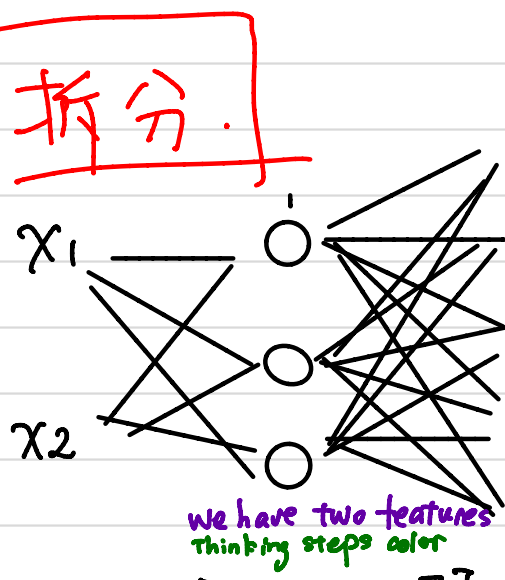
$\rightarrow Z^{[2]} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ z^{[2](1)} & z^{[2](2)} & \cdots & z^{[2](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}$

$\hat{Y} = g(Z^{[4]}) = A^{[4]}$

for loop.

for $l = 1, \ldots, 4$ (This for loop is unavoidable).

拆分.



$Z^{[1]}_{(3,1)} = W^{[1]}_{(3,2)} \cdot X_{(2,1)} + b^{[1]}_{(3,1)}$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]})(n^{[0]}, 1)$

$W^{[1]} = (n^{[1]}, n^{[0]})$

$W^{[2]} = (5,3) \quad (n^{[2]}, n^{[1]})$

$Z^{[2]}_{(5,1)} = W^{[2]}_{(5,3)} a^{[1]}_{(3,1)} + b^{[2]}_{(5,1)}$

$W^{[3]} \quad (4,5)$

$W^{[4]} \quad (2,4)$

$\Rightarrow W^{[l]} = (n^{[l]}, n^{[l-1]})$

$b^{[l]} = (n^{[l]}, 1)$

$dW^{[l]}$: same as $W^{[l]}$

$db^{[l]}$: same as $b^{[l]}$

$Z^{[l]} = g^{[l]}(a^{[l]})$: same as $a^{[l]}$

Then we have more examples. (m examples)

$Z^{[1]}$

$(n^{[1]}, 1) \Rightarrow (n^{[1]}, m)$ $\begin{bmatrix} \vdots & \vdots & & \vdots \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ \vdots & \vdots & & \vdots \end{bmatrix}_{n^{[1]} \times m}$

All of the training sets stack horizontally. $\Rightarrow$ each row represents the number of $X_1, X_2, \cdots, X_k$ (k is the number of features/attributes)

$Z^{[1]} = W^{[1]} \cdot X + b^{[1]} \Rightarrow (n^{[1]}, m)$

$(n^{[1]}, m)\ (n^{[1]}, n^{[0]})\ (n^{[0]}, m)\ (n^{[1]}, 1)$.

* m is the number of samples

Then we have: $Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

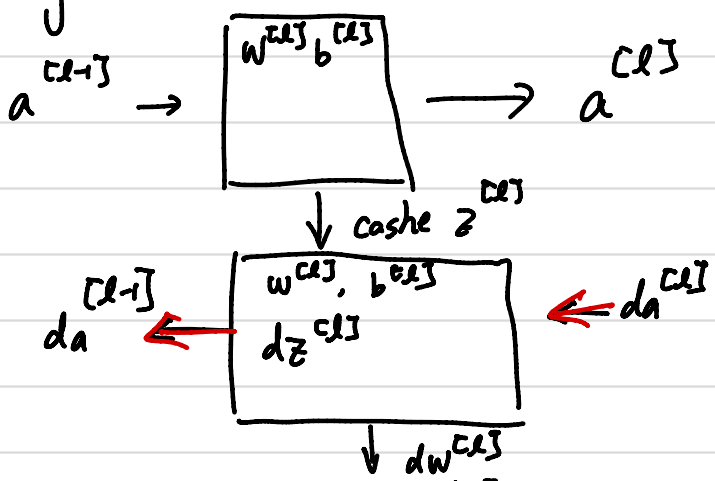
$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$.

In general , the deep neural network is to 拆分一个图片 to many pixels, for example first layer is to detect edges, then second hidden layer is to detect the facial features like eyes, noses..... then put the third layer us to put them together,.

Backward Propagation:

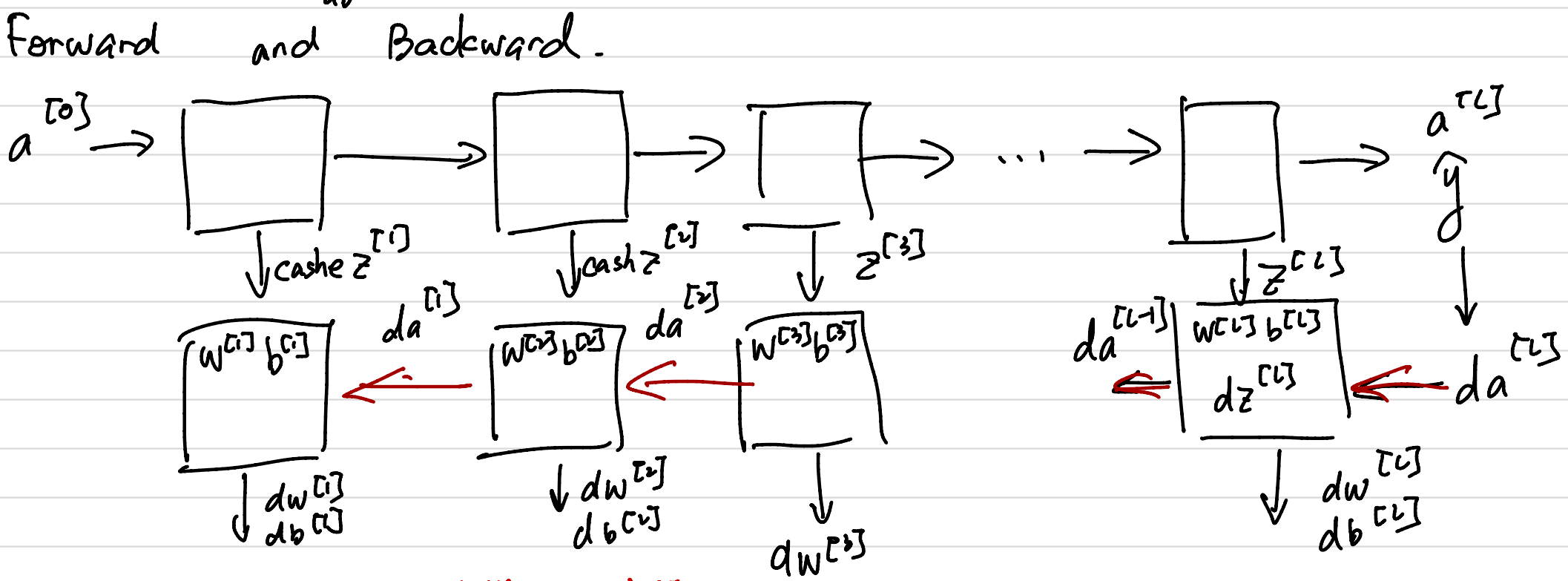
Input $da^{[l]}$, cashe $z^{[l]}$ output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

layer l.



What does cashe mean here: cashe is a widely used method for storing information so that it can be later accessed much more quickly.

Forward and Backward.



$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$

$b^{[l]} := W^{[l]} - \alpha db^{[l]}$

$a=\begin{bmatrix}a_1\\a_2\end{bmatrix}$ $b=\begin{bmatrix}b_1\\b_2\end{bmatrix}$ $a\cdot b = a_1b_1 + a_2b_2$

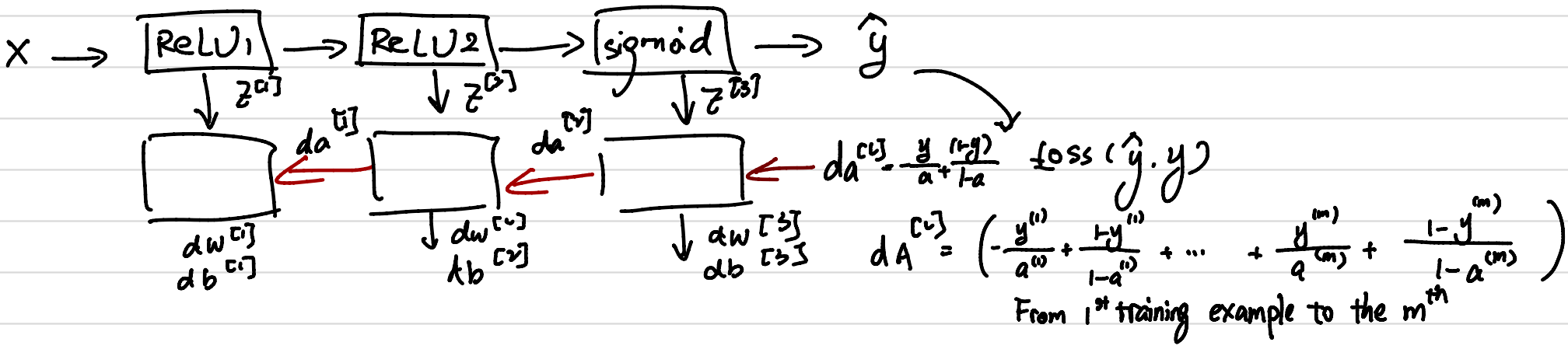
The difference of cross product and dot product
① cross → 结果还是matrix
② dot → 结果为数字. $u\cdot v = u_1v_1 + u_2v_2 + u_3v_3$

Input $da^{[l]}$
Output $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$dz^{[l]} = \frac{da^{[l]}}{dz^{[l]}} * g^{[l]'}(z^{[l]})$ cross product
$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$ dot product
$db^{[l]} = dz^{[l]}$
$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$
反向
$\Rightarrow \frac{da^{[l-1]}}{dz^{[l]}} = w^{[l+1]T} dz^{[l+1]} * g^{[l]'}(z^{[l]})$

$a^{[l]} = g^{[l]}(z^{[l]})$ $z^{[l]} = w^{[l]}A^{[l-1]} + b^{[l]}$

Vectorize :
$$dz^{[l]} = dA^{[l]} \times g^{[l]'}(z^{[l]})$$
$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dz^{[l]}, axis=1, keepdims=True)$$
$$dA^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$



$$dA^{[l]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{1-y^{(1)}}{1-a^{(1)}} + \cdots + \frac{y^{(m)}}{a^{(m)}} + \frac{1-y^{(m)}}{1-a^{(m)}}\right)$$
From 1st training example to the m^{th}

Hyperparameter.
parameters : $W^{[1]}, b^{[1]}, \cdots$

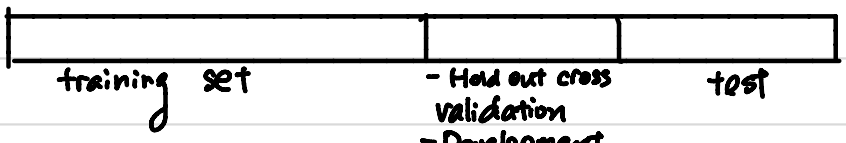
Hyperparameter : learning rate α
iterations
hidden layers L
hidden units $n^{[1]}, n^{[2]} \cdots$
choice of activation function

} parameters control $W^{[1]}, b^{[1]}$.

The Second Course

Train / dev / test sets.

Data



To choose which model is the best

Prev era. 70/30 60/20/20.

Modern data era. 1,000,000 Dev become smaller

98% / 1% / 1%

99.5% / 0.4% / 0.1%

Mismatched train / test distribution

Training set: cat pics from web

Dev/test sets: cat pics from App. 不準. ⇒ Make sure that dev and test sets from same distribution

* Not having a test set might be okay. (only dev set.)

Bias and Variance

cat classification	cat y=1	not cat y=0		
Train set error	1%	15%	15%	0.5%
Dev set error	11%	16%	30%	1%
	high variance overfitting	high bias (can't doing well on training set) underfitting	high bias high variance	low bias low variance.

Human 0%

Optimal error (Bayes): 0%

Blurry pics. ⇒ Baye's error ↑

Bias Problem: How well your training data is

Variance Pro: training → test. how bad

High bias: The model does not fit well

High var: The model overfits the data.

Thinking:

High bias? (training data platform) → · Bigger network. (* Always reduce bias without hurting variance)
· NN larger

↓

High variance? (dev. set performance). → · More Data (* Always reduce var without hurting bias)
· Regularization

↓ N

Done

∴ "Bias-variance tradeoff"

Logistic Regression

λ: regularization parameter (hyperparameter) ← weight 变小, prevent overfitting problem

In python: λ-lambda. So we use lambd for this function (名称区别)

$$\min J(w,b)$$

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

L2 regularization: $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

L1 regularization: $\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$, (w will be spouse).

Neural Network.

$$J(w^{[1]}, b^{[1]}, \cdots, w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|_F^2 \quad \text{— L2 regularization}$$

$w^{[l]}$: $(n^{[l]}, n^{[l-1]})$

$$\|w^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$$

"Frobenius norm" — 跟 w^2 相加.

Gradient, $dw^{[l]} = (\text{from before}) + \frac{\lambda}{m}w^{[l]}$

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

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

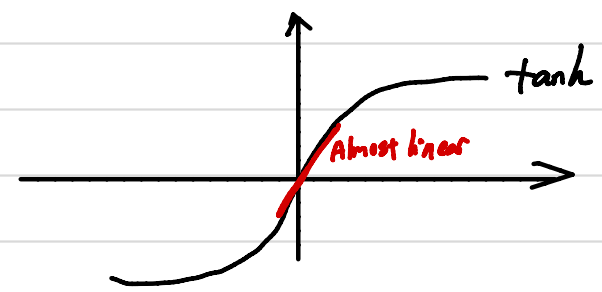
$$= w^{[l]} - \frac{\alpha\lambda}{m}w^{[l]} - \alpha(\text{from before})$$

"weight decay" $= \left(1 - \frac{\alpha\lambda}{m}\right)w^{[l]} - \alpha(\text{from before})$

Why regularization prevent overfitting?

$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}\|w^{[l]}\|_F^2$$

If λ is really big. and we want $J(w^{[l]}, b^{[l]})$ to be small then $w^{[l]} \approx 0$, which will leads to a simplier NN (neural network) ⇒ overfitting fix

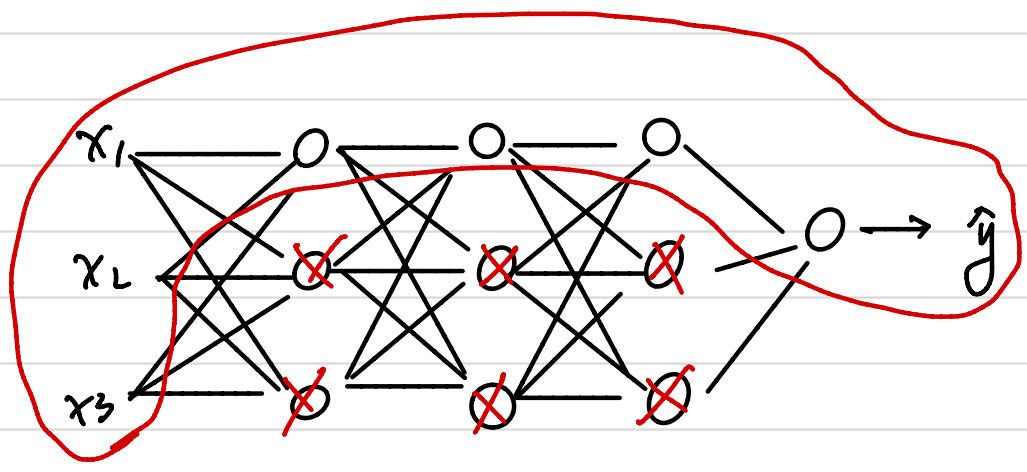


$g(z) = \tanh(z)$

$\lambda \uparrow \quad w^{[l]} \downarrow \quad z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$

z ↓ w ↓

$g^{[l]}(z^{[l]})$ will be roughly linear ↗



Dropout (drop some units in different layers) "Inverted Dropout"

layer: $l = 3$ keep-prob = 0.8 (the prob that nodes will be kept)

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_prob
```

↓ random matrix

activation of layer 3.

```
a3 = np.multiply(a3, d3)      # a3 *= d3
a3 /= keep_prob
```

把 zero out.

e.g. If we have 50 units in 3rd layer ∵ keep_prob = 0.8 ⇒ 20% zero

then 10 units will be zero out

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

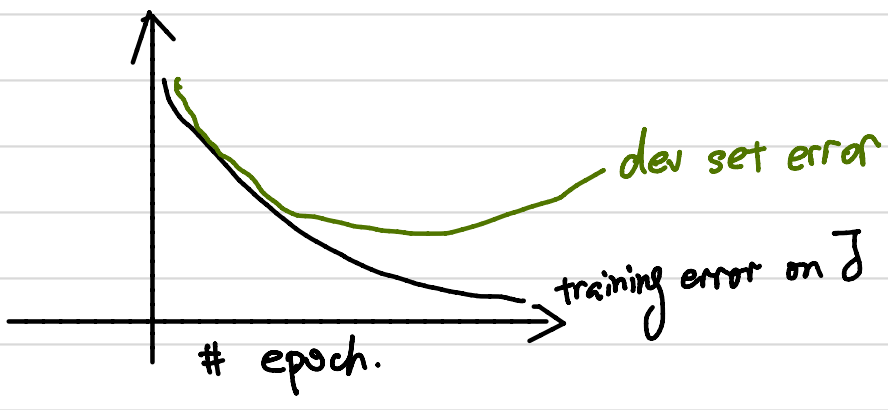
↓

reduced by 20%

Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights.

Each layer can have different keep-prob. you can also set keep-prob=1. means no units need to be get rid of.



Othorganization.

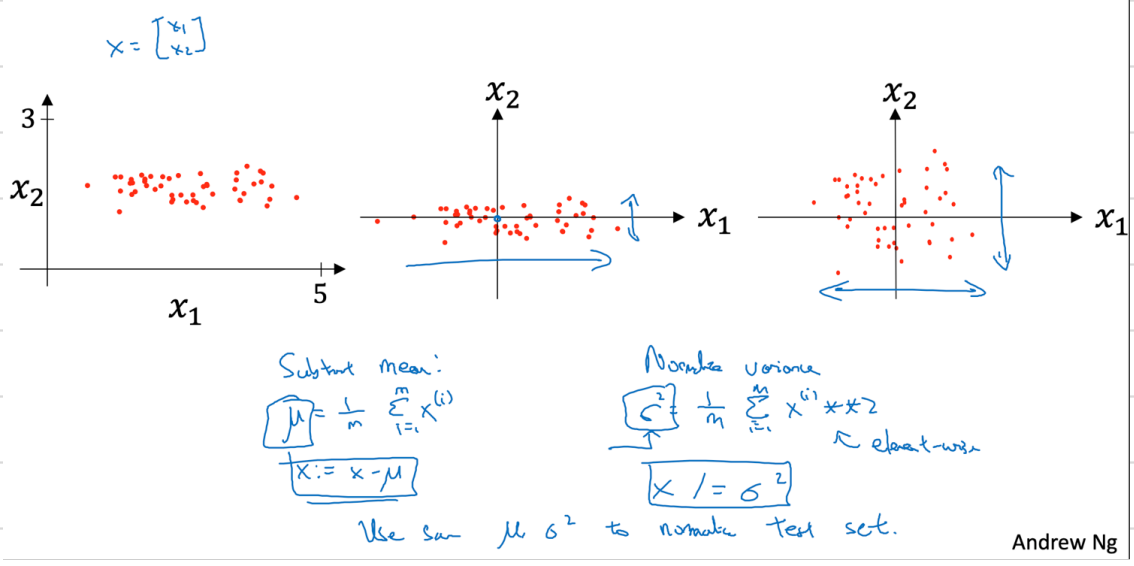
Optimize cost function J: $\Rightarrow J(w,b)$

. Gradient,

Not overfitting:

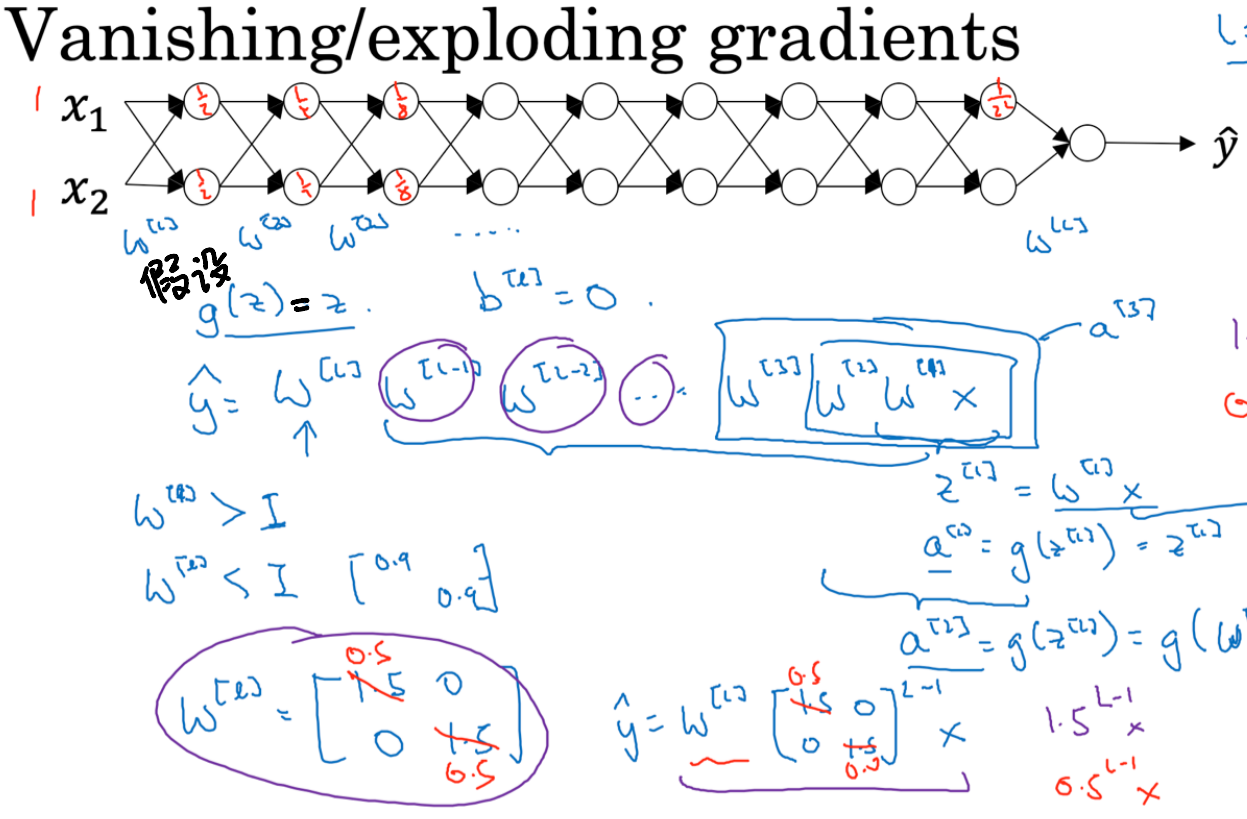
. Regularization $\|w\|_F^2$.

Normalizing training sets



Andrew Ng

Vanishing/exploding gradients



Andrew Ng