Deep Learning

1. Neural Networks and Deep Learning

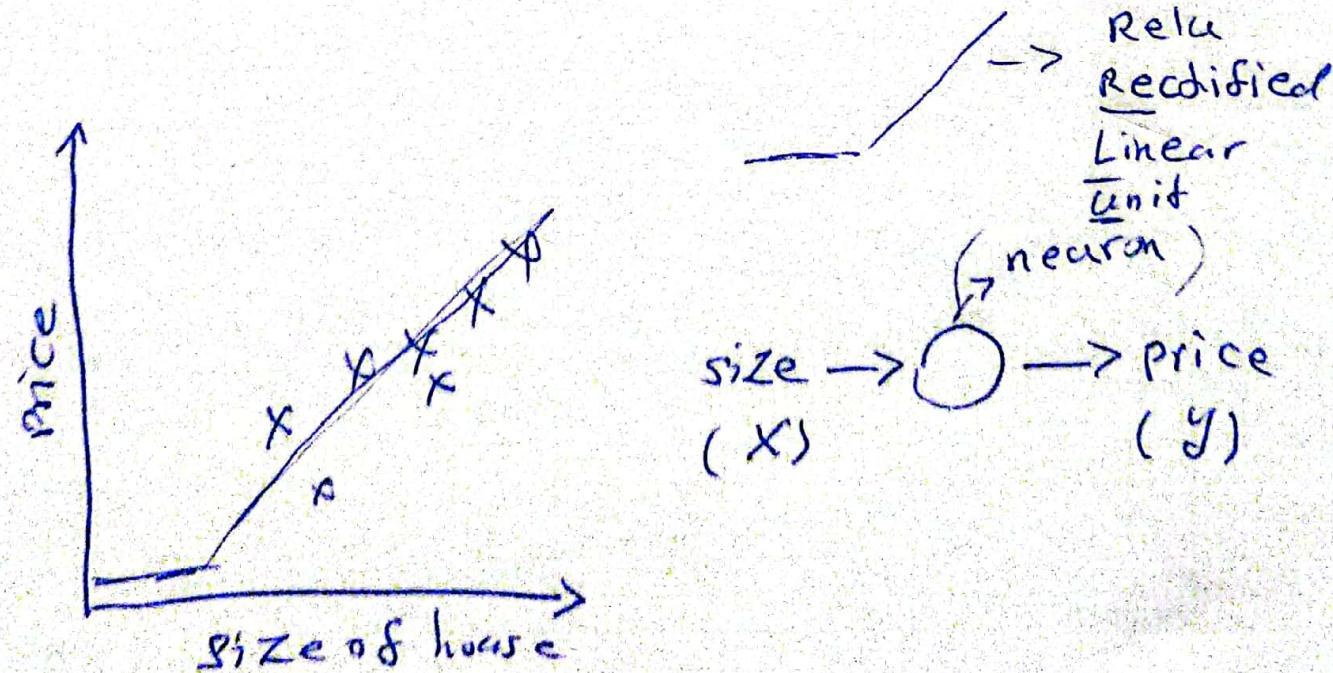
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization.

3. Structuring your Machine Learning project

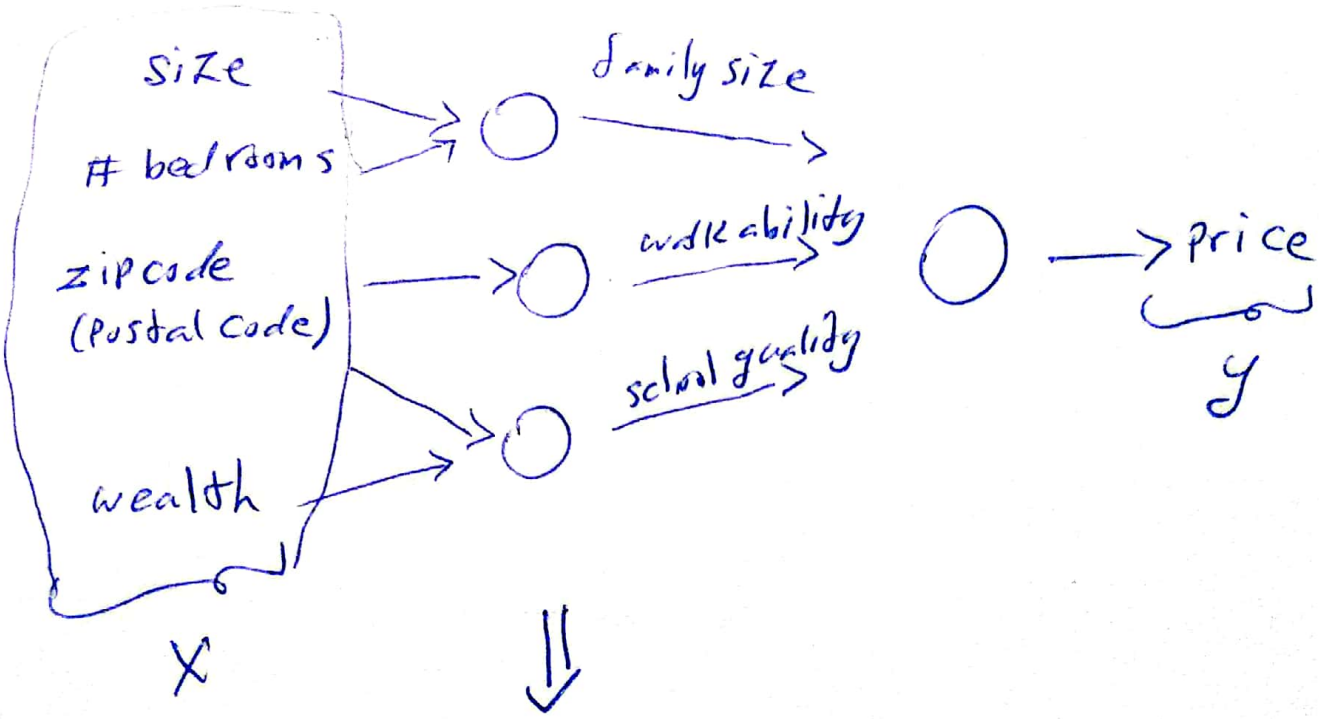
4. Convolutional Neural Networks

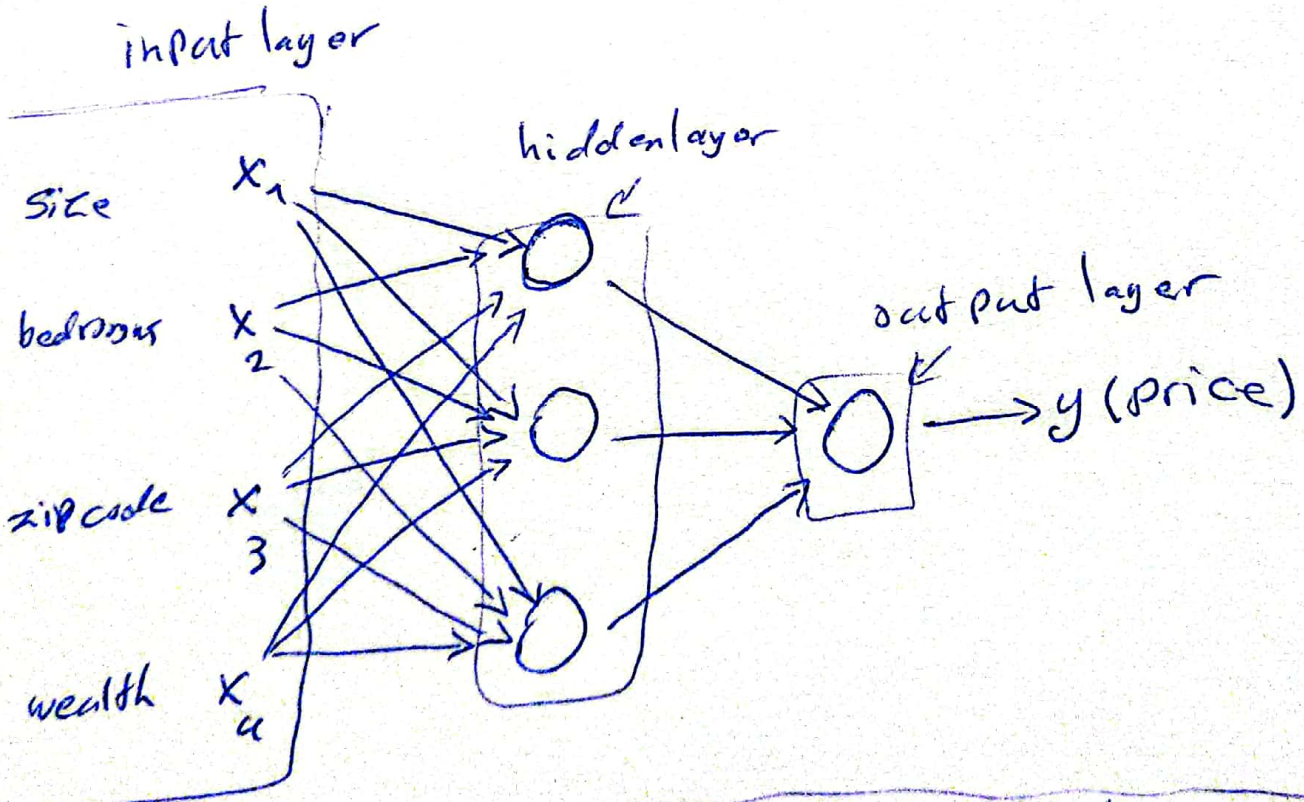
5. Natural Language Processing: Building sequence models (RNNs, LSTM)

what is a neural network?



"Housing price prediction"





Supervised Learning with neural networks

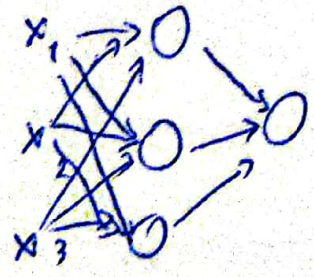
Supervised learning examples (deep learning)

Input (X)	Output (Y)	Application
Home features	Price	Real state ~~Standard~~ NN
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	object (1, ..., 1000)	Photo tagging CNN
Audio	Text transcript	speech recognition RNN
English	Chinese	Machine translation RNN
Image, Radar Info	Position of other cars	Autonomous driving

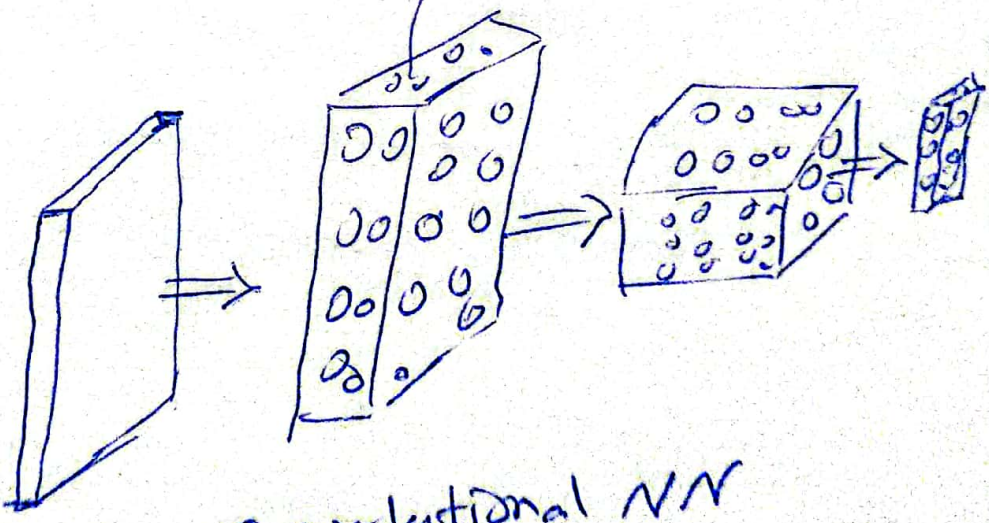
CNN

hybrid neural networks

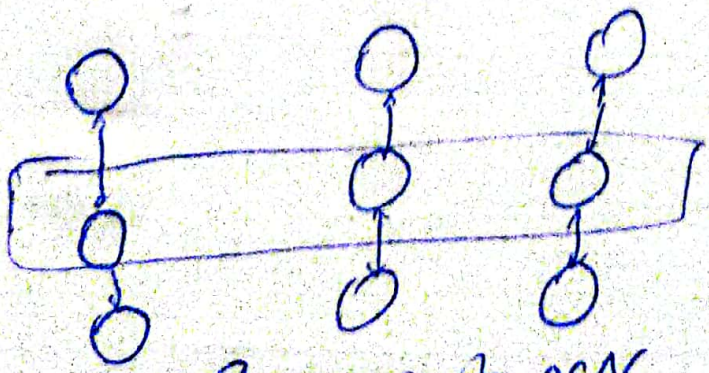
Custom / hybrid



Standard NN



Convolutional NN



Recurrent NN

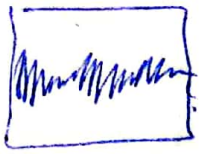
Supervised Learning

Structured data

Size	# bedroom	...	Price
2104	?	⋮	400
1000	3	⋮	300
⋮	⋮	⋮	⋮
3000	4		500

↓ databases

Unstructured data



Audio

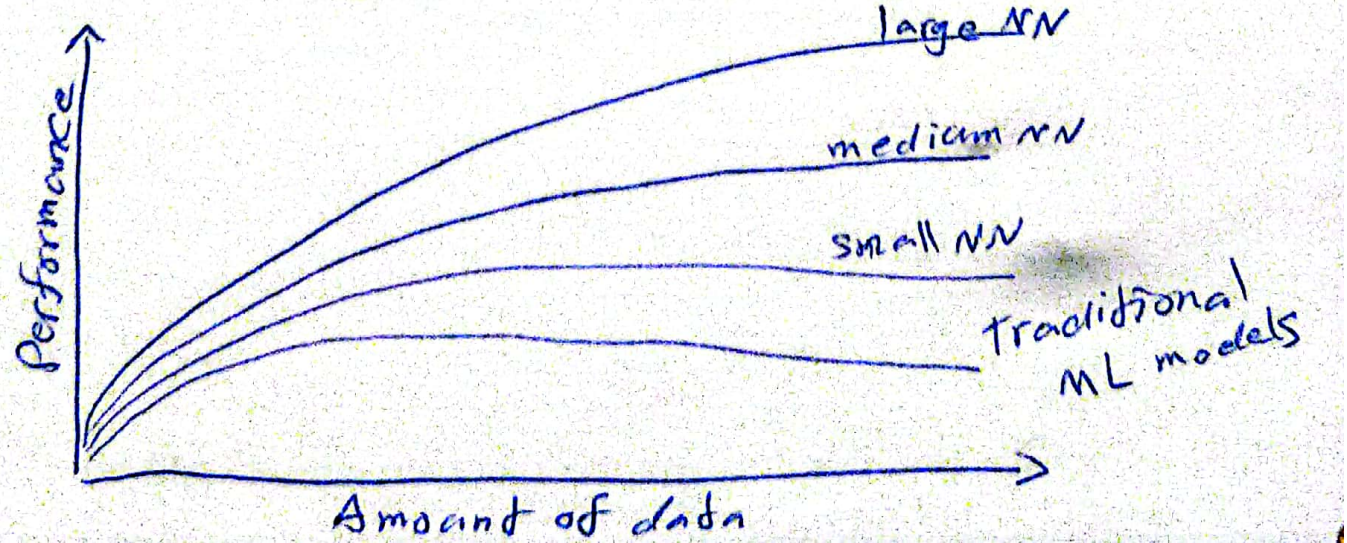
Image

I am so happy, —

Text

historically computers (classical ML) is better in understanding Structured data, but human are really good in understanding unstructured data, thanks deep learning computer nowadays are really better in understanding unstructured data in compare to the past.

why deep learning recently is taking off?

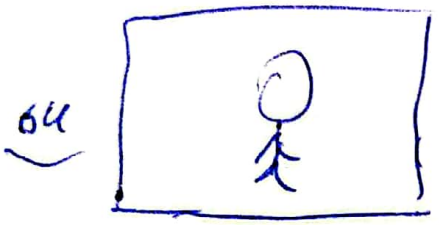


Binary classification

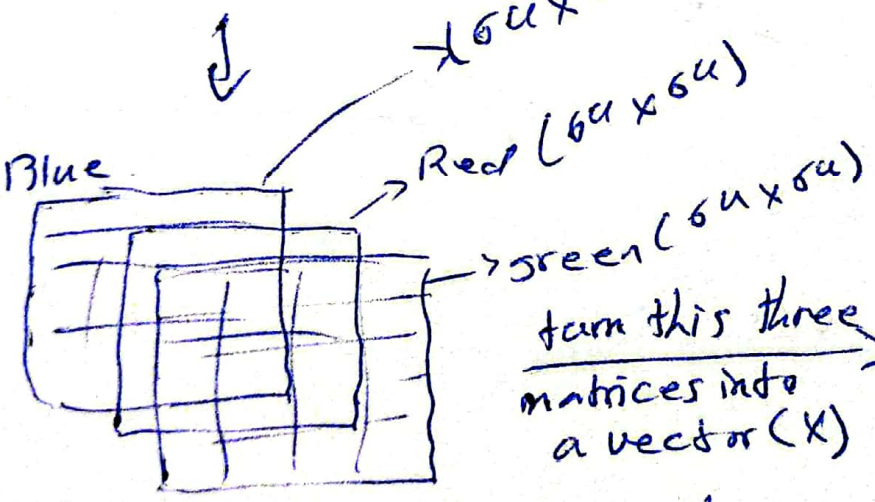
logistic regression: an algorithm ~~to classificate a binary problem~~ for binary classification.

example:

an image



——> human (1), not human (0)



three matrices to represent ~~the~~ the three main colors in the picture.

so the dimension of X is: $n_x = 12288$

notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$m = m_{train}$ $\quad m_{test}$ → number of examples

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \rightarrow \text{number of features}$$

(m) ↓

stacking all m (examples) into ~~the~~ X matrix

So: $X = \mathbb{R}^{(n_x \times m)}$

$X.shape = (n_x, m)$ → number of examples

↓ number of features

$$Y = [y^{(1)}, y^{(2)}, \ldots, y^{(m)}]$$

$Y \in \mathbb{R}^{1 \times m}$

$Y.shape = (1, m)$

logistic regression

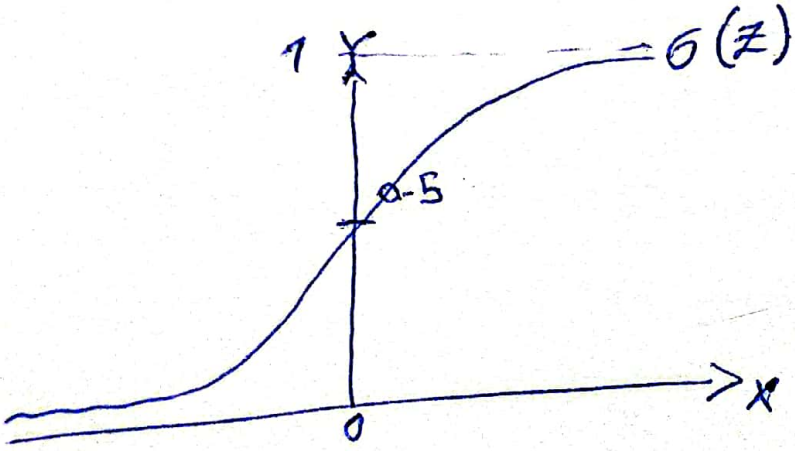
Given x, want $\hat{y} = P(y=1|x)$

$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

$z = w^T x + b$

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



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

if z is large then $\sigma(z) \simeq \frac{1}{1+0} \simeq 1$

if z is small then $\sigma(z) \simeq \frac{1}{1+\infty} \simeq 0$

cost function for logistic regression

loss (error) function

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

if $y=1$ then: $L(\hat{y}, y) = -\log \hat{y}$

if $y=0$ then: $L(\hat{y}, y) = -\log(1-\hat{y})$

cost function (logestic regression)

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

$$= -\frac{1}{2}\sum_{i=1}^{m}\left[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

- what is the difference between the cost function and the loss function?

the loss function computes the error for a single training example, the cost function is the average of the loss functions of the entire training set.

Gradient descent

logestic regression algorithm

Recap: $\hat{y} = \sigma(w^T \cdot X + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

cost function

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

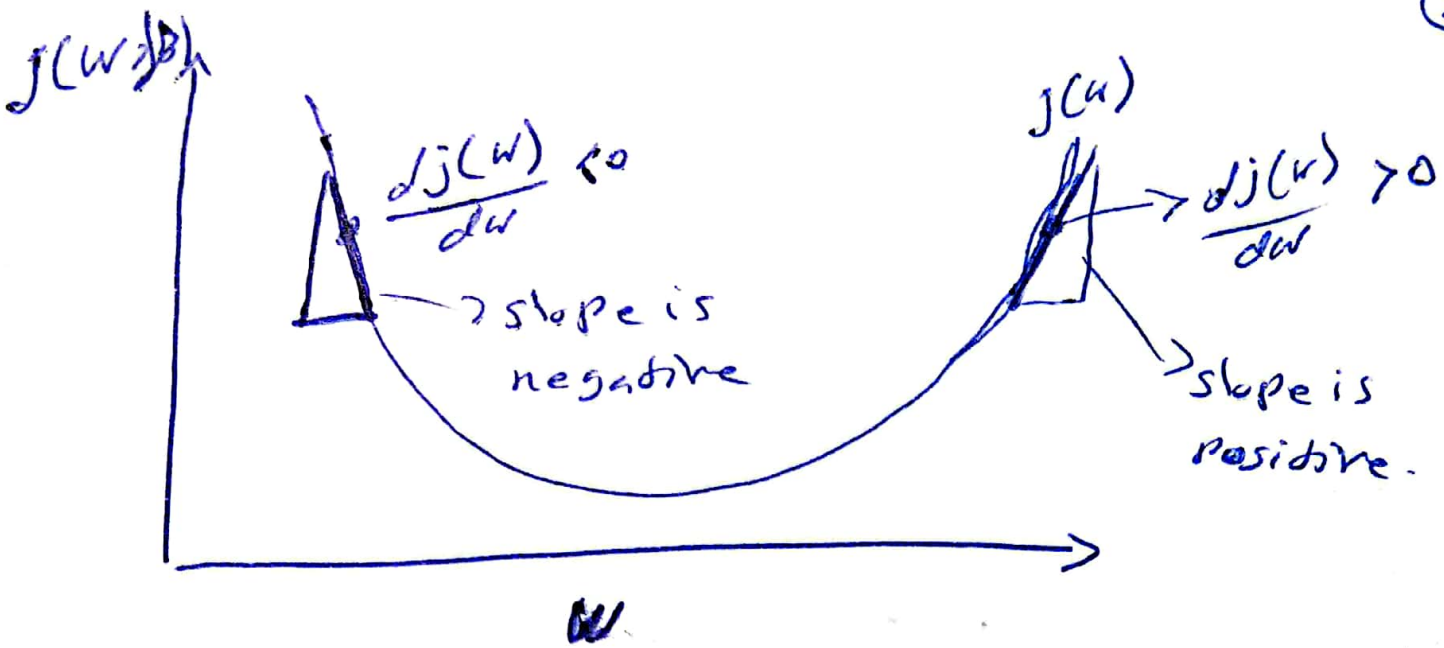
want to find w, b that minimize $J(w,b)$

Repeat {

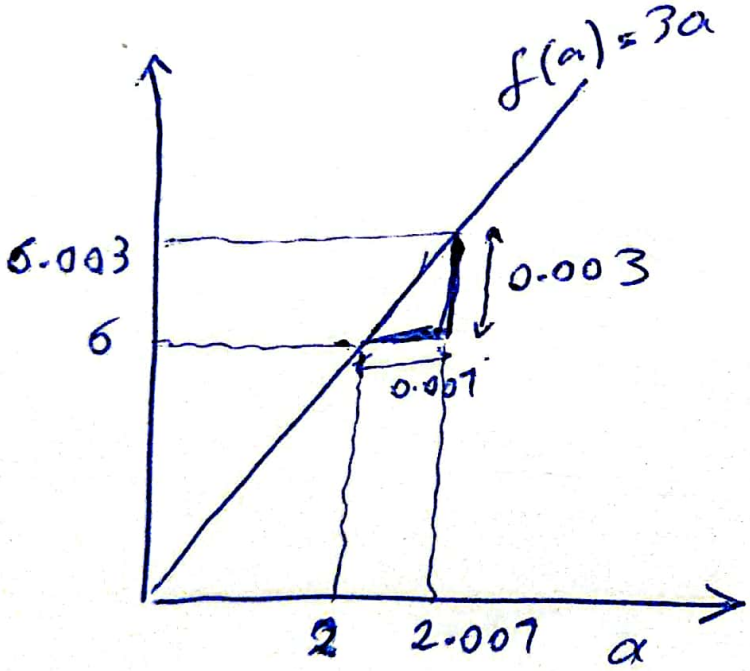
$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

}



derivative



$a = 2 \quad f(a) = 6$

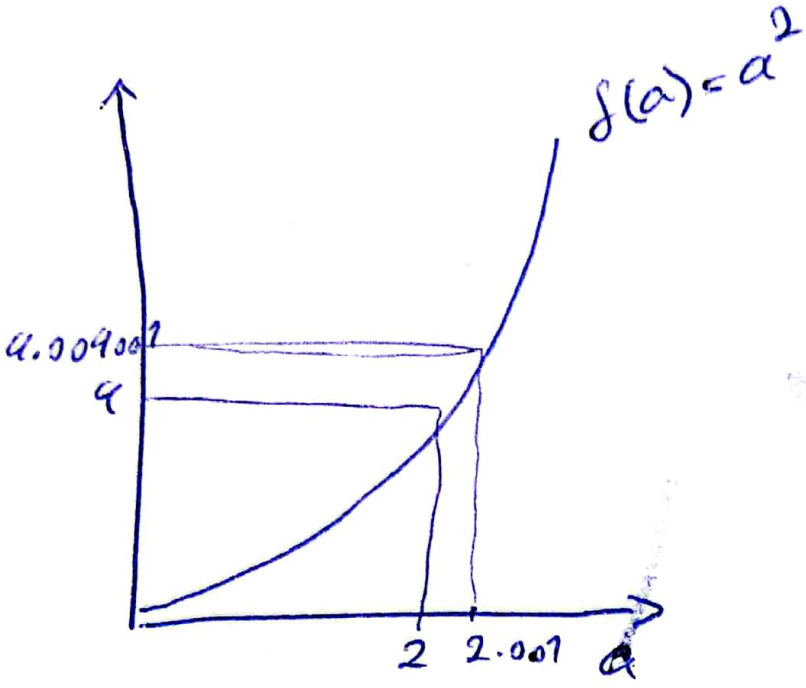
$a = 2.007 \quad f(a) = 6.003$

slope (derivative) of $f(a)$ at $a = 2 = 3$

$$\text{slope} = \frac{\text{height}}{\text{width}} = \frac{0.003}{0.007}$$

$$= 3$$

$$\boxed{\frac{df(a)}{da} = 3}$$



$a = 2 \quad f(a) = 4$

$a = 2.001 \quad f(a) \simeq 4.004001$

slope (derivative) of $f(a)$ at $a = 2 = 4$

$\frac{d f(a)}{da} = 4$ when $a = 2$

$a = 5 \quad f(a) = 25$

$a = 5.001 \quad f(a) \simeq 25.010$

$$\frac{d f(a)}{da} = \frac{d}{da} a^2 = 2a$$

$$f'(a) = a^3 \qquad \frac{d}{da} f(a) = 3a^2$$

$$f(a) = \log(a) \qquad \frac{d f(a)}{da} = \frac{1}{a}$$

$\ln(a)$

computation graph

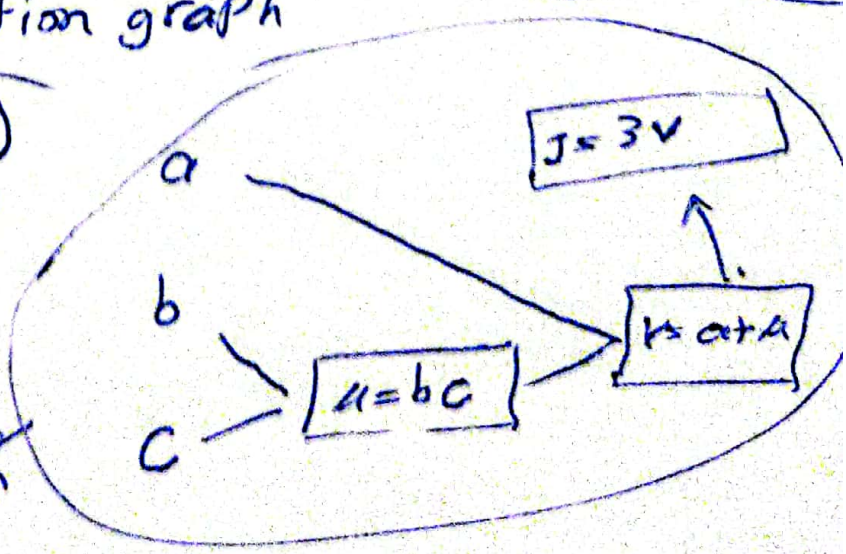
exp:

$J(a,b,c) = 3(a + b \cdot c)$

$u = b \cdot c$

$v = a \cdot u$

$J = 3v$



the computation graph

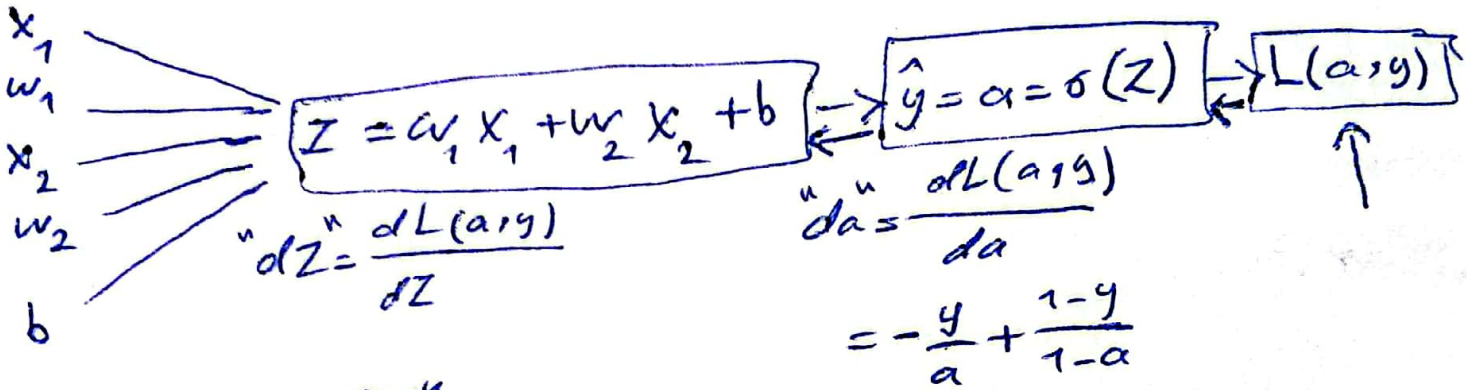
~~derivative with a computation graph~~

Logistic regression Gradient descent

$$z = w^T \cdot x + b$$

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

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



$$= a - y$$

$$\frac{dL}{dw_1} = "dw_1" = x_1 \cdot dz \quad , \quad "dw_2" = x_2 \cdot dz \; , \; db = dz$$

$$w_1 := w_1 - \alpha \, dw_1$$

$$w_2 := w_2 - \alpha \, dw_2$$

$$b := b - \alpha \, db$$

Gradient Descent on m example

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

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

$dw_1^{(i)}$, $dw_2^{(i)}$, $db^{(i)}$ $\quad (x^{(i)}, y^{(i)})$

$$\frac{dJ(w,b)}{dw_1} = \frac{1}{m}\sum_{i=1}^{m} \frac{d}{dw_1} \underbrace{L(a^{(i)}, y^{(i)})}_{dw_1^{(i)} - (x^{(i)}, y^{(i)})}$$

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

for $i=1$ to m:

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

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

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

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} \cdot dz^{(i)}$ } $n=2$

$db \mathrel{+}= dz^{(i)}$

$J /= m$; $dw_1 /= m$; $dw_2 /= m$; $db /= m$

$$dw_1 = \frac{dJ}{dw_1} \quad ; \quad dw_2 = \frac{dJ}{dw_2} \quad ; \quad db = \frac{dJ}{db}$$

$$w_1 := w_1 - \alpha \, dw_1$$

$$w_2 := w_2 - \alpha \, dw_2$$

$$b := b - \alpha \, db$$

---

what is vectorization?

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

$$z = \underbrace{np.dot(w, x)}_{w^T \cdot x} + b$$

Parallelization: both CPU and GPU have Parallelization in structure called SIMD - stand for Single Instruction multiple Data.

- In neural network implementation, whenever possible, avoid explicit for-loops.

logistic regression derivatives with for loops:

$J = 0 \; ; \; dw_1 = 0 \; ; \; dw_2 = 0 \; ; \; db = 0$

for $i = 1$ to m:

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

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

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

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

$dw_1 \mathrel{+}= x_1^{(i)} \cdot dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} \cdot dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m \; ; \; dw_1 = dw_1/m \; ; \; dw_2 = dw_2/m \; ; \; db = db/m$

vectorization of logestic regression

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

$$[z^{(1)}, z^{(2)}, \ldots, z^{(m)}] = w^T \cdot X + \underbrace{[b, b, \ldots, b]}_{(1 \times m)}$$

$$= [w^{(1)}, w^{(2)}, \ldots, w^{(n)}] \cdot \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}_{(n \times m)} + b$$

$$= [w^T \cdot x^{(1)} + b,\ w^T \cdot x^{(2)} + b,\ w^T \cdot x^{(3)} + b \ldots,\ w^T \cdot x^{(m)} + b]$$

$$\Rightarrow Z = w^T \cdot X + b$$

$$Z = np.dot(w.T, X) + \underbrace{b}_{(1,1)}$$

$$A = \sigma(Z)$$

$$A = 1 / 1 + np.exp(-Z) \; ; \; \boxed{dZ = A - Y}$$

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} \Rightarrow \frac{1}{m} \times np.sum(dZ)$$

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

$$= \frac{1}{m} \begin{bmatrix} x^{(1)}, \ldots, x^{(m)} \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)}, \ldots, x^{(m)} dz^{(m)} \right]$$

$(n_x, 1)$

Implementing Logistic Regression (completely)

$Z = w^T \cdot x + b$; $Z = np.dot(w.T, X) + b$

$A = \sigma(Z)$; $A = \frac{1}{1 + np.exp(Z)}$

$dZ = A - Y$

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

$db = \frac{1}{m} \cdot \text{Scum}(dZ)$

$w := w - \alpha . dw$

$b := b - \alpha . db$

a single iteration of logistic regression algorithm.

"Broadcasting in python"

- broadcasting is a technique, which can be used to speed up the code.

"example of broadcasting in Python"

```
A = np.array([
    [1  2  5  2]
    [2  3  5  3]
    [3  4  5  1]
])
```

A.shape = (3 x 4)

```
cal = A.sum(axis=0)  # means: sum up the entire values in each column.
```

↓

[6 9 15 6]

percentage = A / cal

↓

percentage.shape = (3 x 4) # percentage of each value in the corresponding column.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$$

broadcasting ↑

General Principle of broadcasting

Matrix (m,n) $\begin{matrix}+\\-\\\times\\/\end{matrix}$ $(1,n) \rightsquigarrow (m,n)$

$(m,1) \rightsquigarrow (m,n)$

Matrix $(m,1)$ $+$ $R_{(100)} \rightsquigarrow (m,1)$

Matrix $(1,m)$ $+$ $R_{(100)} \rightsquigarrow (1,m)$

broadcasting in numpy

```
np.sum(axis=0)  # sum up vertically
np.sum(axis=1)  # sum up horizontally
```

Logistic regression cost function

$\hat{y} = \sigma(w^T \cdot X + b)$ where $\sigma(Z) = \frac{1}{1+e^{-Z}}$

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

if $y=1$: $P(y|X) = \hat{y}$

if $y=0$: $P(y|X) = 1-\hat{y}$

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

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

$\hat{y}$; $(1-\hat{y})^0 = 1$

if $y=1$: $\hat{y}$

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

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

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

$$= y \log \hat{y} + (1-y) \log (1-\hat{y})$$

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

SOTT

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

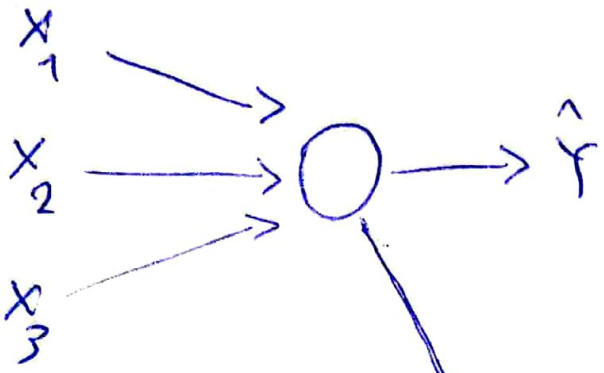
$$\log P(\dots) = \sum_{i=1}^{m} \underbrace{\log P(y^{(i)}|x^{(i)})}_{-L(\hat{y}^{(i)}, y^{(i)})}$$

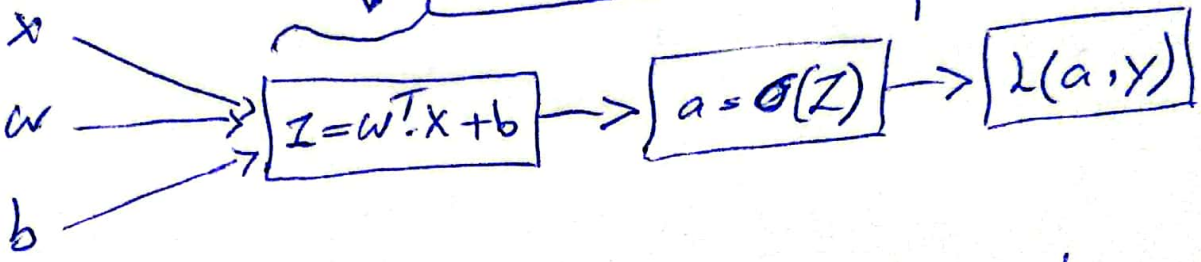
$$= -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

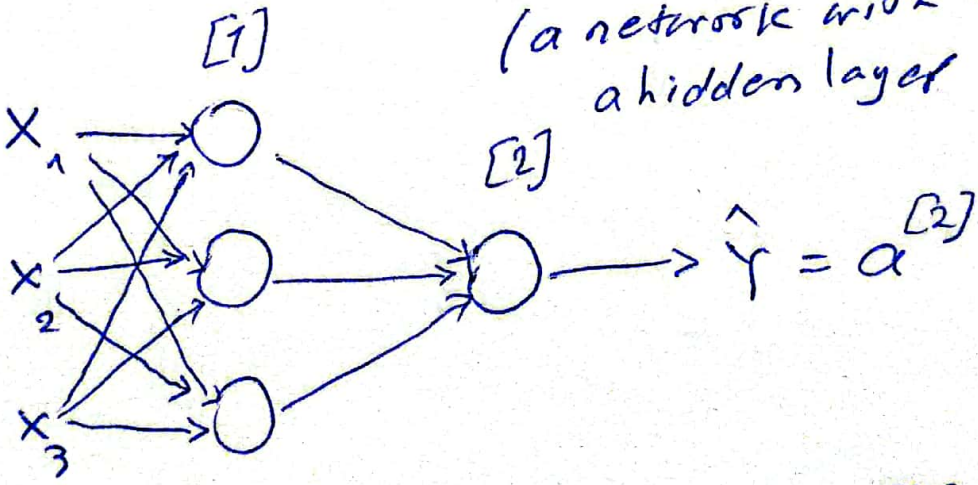
cost $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

Cminimize)

"Neural network overview"





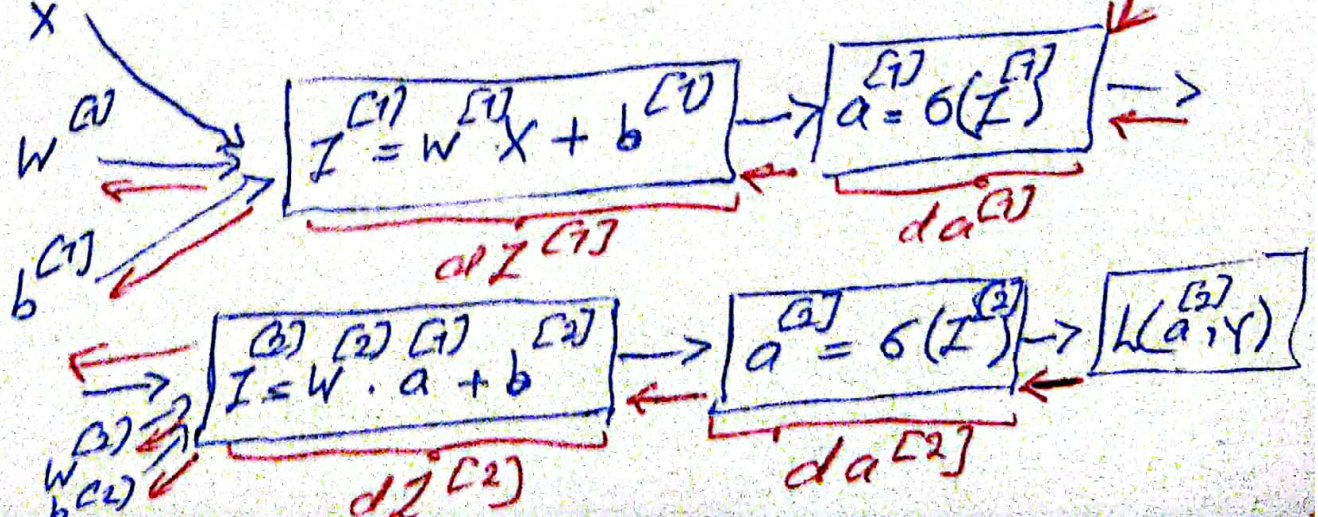


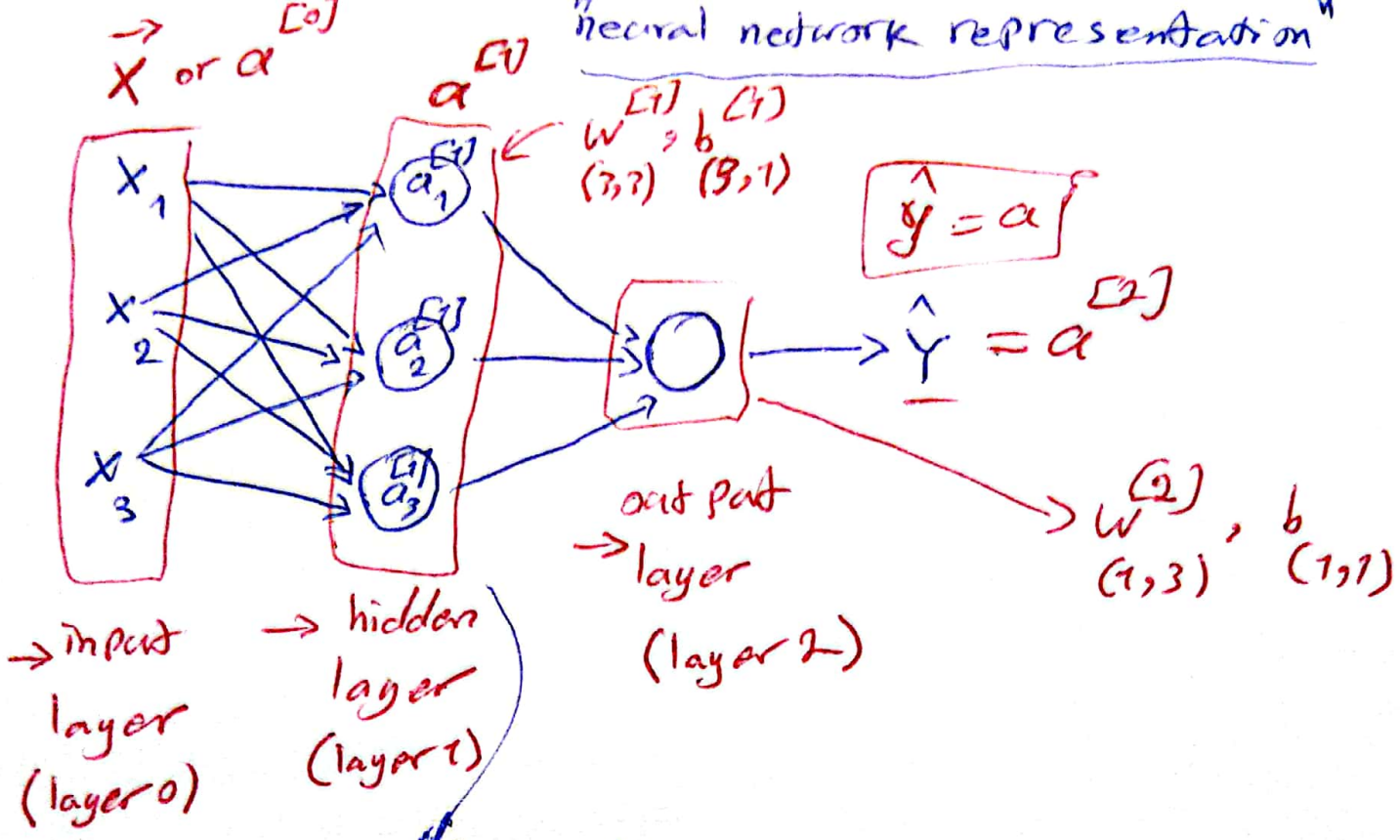
(a network with two layer) a hidden layer and an output layer)

notations:

$[1]$ → layer one

backward propagation

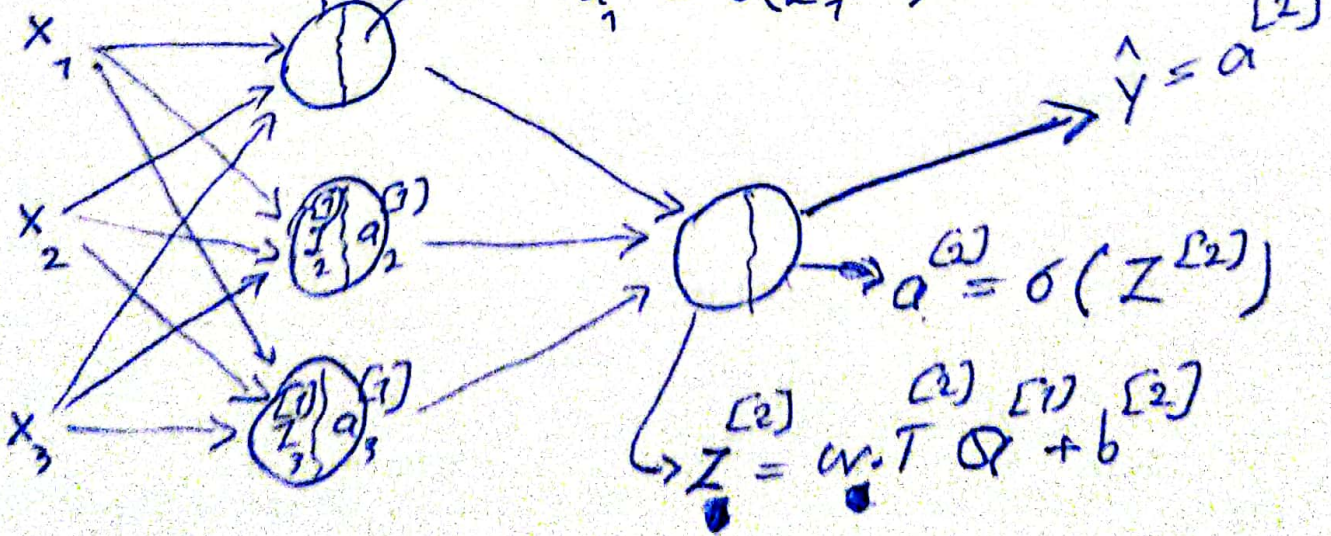




$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix}$$

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

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



hidden layer (1)

$$
\begin{aligned}
Z_1^{[1]} &= W_1^{[1]T} X + b_1^{[1]}, & a_1^{[1]} &= \sigma(Z_1^{[1]}) \\
Z_2^{[1]} &= W_2^{[1]T} X + b_2^{[1]}, & a_2^{[1]} &= \sigma(Z_2^{[1]}) \\
Z_3^{[1]} &= W_3^{[1]T} X + b_3^{[1]}, & a_3^{[1]} &= \sigma(Z_3^{[1]}) \\
Z_4^{[1]} &= W_4^{[1]T} X + b_4^{[1]}, & a_4^{[1]} &= \sigma(Z_4^{[1]})
\end{aligned}
$$

$$
\begin{bmatrix} — W_1^{[1]T} — \\ — W_2^{[1]T} — \\ — W_3^{[1]T} — \\ — W_4^{[1]T} — \end{bmatrix}_{(4\times 3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}_{(4,1)} = \begin{bmatrix} W_1^{[1]T} X + b_1^{[1]} \\ W_2^{[1]T} X + b_2^{[1]} \\ W_3^{[1]T} X + b_3^{[1]} \\ W_4^{[1]T} X + b_4^{[1]} \end{bmatrix}
$$

$$
\Leftrightarrow = \begin{bmatrix} Z_1^{[1]} \\ Z_2^{[1]} \\ Z_3^{[1]} \\ Z_4^{[1]} \end{bmatrix} \rightarrow a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(Z^{[1]})
$$

Given input X: → num units

$$Z^{[1]}_{(4,1)} = W^{[1]}_{(4,3)} X_{(3,1)} + b^{[1]}_{(4,1)}$$

$$a^{[1]}_{(4,1)} = \sigma(Z^{[1]})_{(4,1)}$$

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

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

Vectorizing across multiple training examples

~~layer~~ → ~~example ith~~.

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

→ num examples

↓ num features

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

→ num units

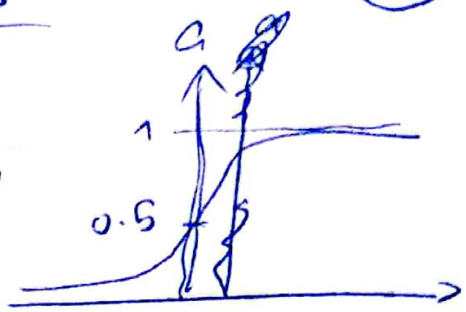
$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1][1]} & a^{[1][2]} & \cdots & a^{[1][m]} \\ | & | & & | \end{bmatrix}_{(\#, m)}$$

→ num units

"Activation functions"

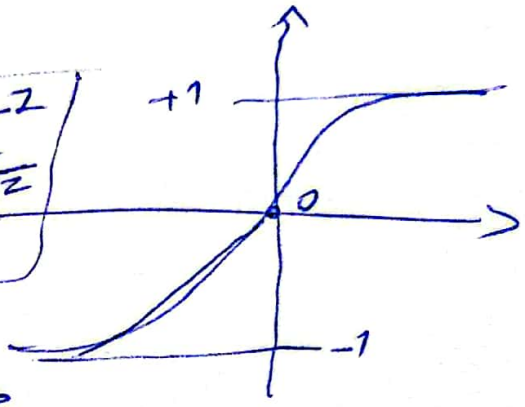
1. Sigmoid function

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



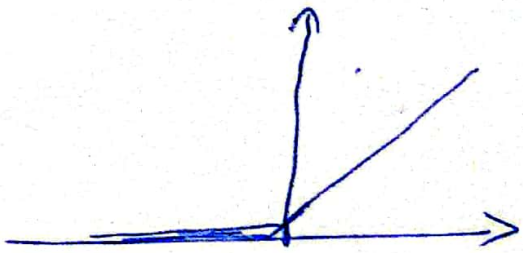
2. tanh function

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



almost always is a better choice than sigmoid function

3. rectified linear unit (Relu)

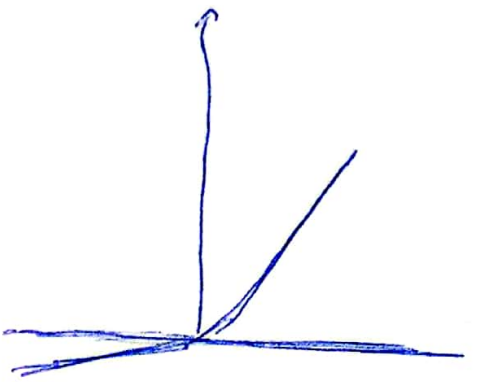


$$a = \max(0, z)$$

- if the output of NN is binary, the output layer should use sigmoid activation function and other layers (hidden layers), should use Relu or sometimes tanh

4. leaky Relu

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



---

why should we use non-linear activation functions?

if does not matter how many hidden layer one uses in a NN, if ~~the~~ they don't have an activation function, the result is just the same as a regression model.

linear or logistic

Derivatives of activation functions

1. Sigmoid activation function:

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

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

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

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

$g(z) = a$

2. tanh activation function

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

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

$$\frac{d}{dz} g(z) = 1 - (\tanh(z))^2$$

if $g(z) = a$

so: $\frac{d}{dz} g(z) = 1 - a^2$

3. Relu

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

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \\ \text{undefined} & \text{if } z = 0 \text{ (?)} \end{cases}$$

4. Leaky Relu

$$g(z) = \max(0.001z, z)$$

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

Gradient Descent for NNs

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$

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

So: $w^{[1]}$ $(n^{[1]}, n^{[0]})$

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

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

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

cost function: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})$

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

Gradient Descent:

Repeat {

Compute predictions $(\hat{y}^{(1)}, \ldots, \hat{y}^{(m)})$

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ... $\longrightarrow$

$$W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$$

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

$$W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$$

$$b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$$

}

"formulas for computing derivatives"

forward Propagation:

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

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

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

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

Back Propagation

$$dZ^{[2]} = A^{[2]} - Y$$

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

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

$$dZ^{[1]} = W^{[2]T} \cdot dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

↓ elementwise product

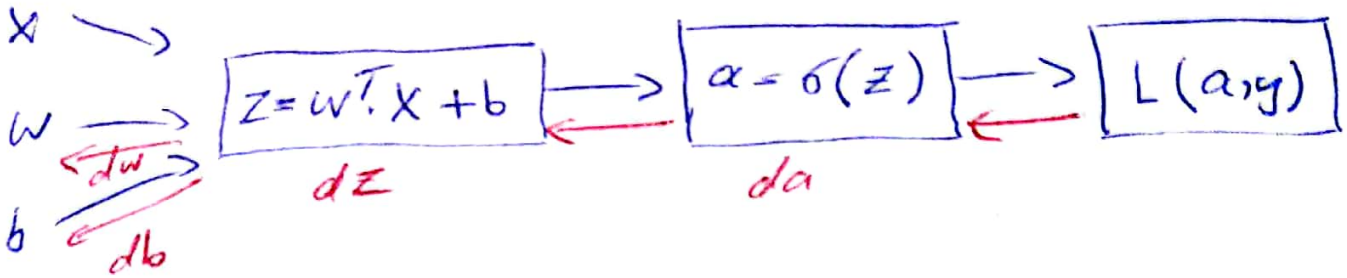
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} \cdot X^{T}$$

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

Prevent (n,)

Same dimension

logistic regression



$$L(a,y) = -y \log a - (1-y)\log(1-a)$$

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

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

$$dz = a - y$$

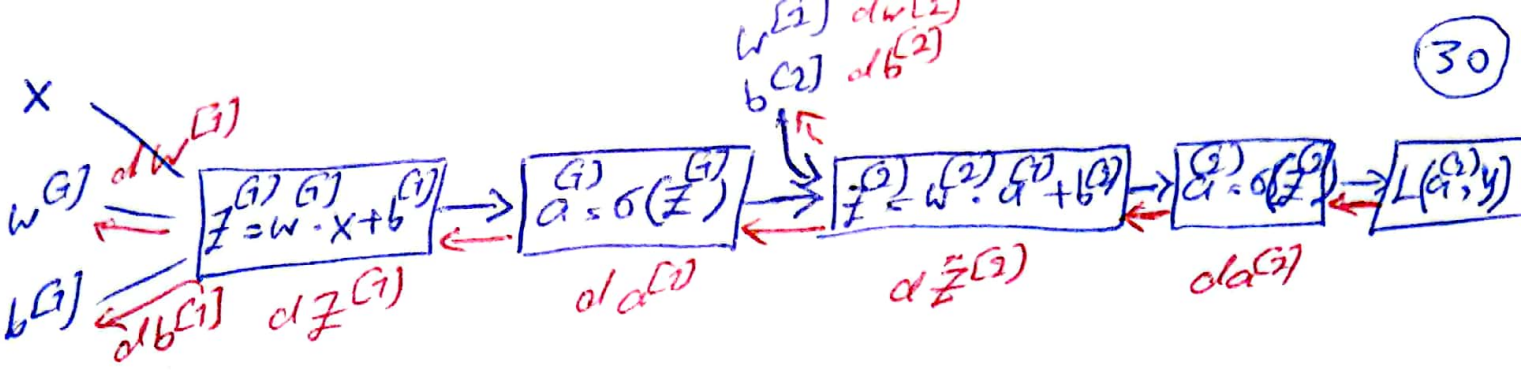
$$= da \cdot g'(z) \quad ; \quad g(z) = \sigma(z)$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \rightarrow \frac{d}{dz} g(z) = g'(z)$$

$$dz = da \cdot g'(z)$$

$$dw = dz \cdot x$$

$$db = dz$$



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

$$dw^{[2]} = dZ^{[2]} \cdot a^{[1]T}$$

~~"$dw = dZ \cdot X$"~~

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

~~$da^{[1]} =$~~

$$dZ^{[1]} = w^{[2]T} \cdot dZ^{[2]} \times g^{[1]\prime}(Z^{[1]})$$

element wise product

~~db~~

$$dw^{[1]} = dZ^{[1]} \cdot X^T$$

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

Summary of gradient descent

Back Propagation for NN (two layers)

for one example

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

$$dW^{[2]} = dZ^{[2]} \cdot a^{[1]T}$$

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

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

↓ elementwise product

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

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

vectorized (for many examples)

$$dZ^{[2]} = A^{[2]} - Y$$

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

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

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

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

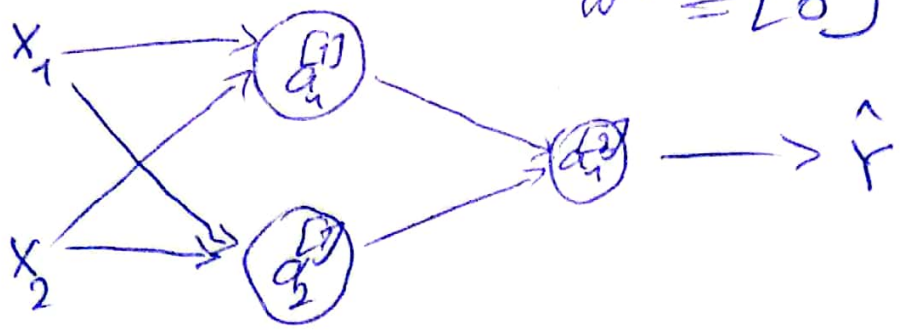
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims=True)$$

"Random initialization"

if the weights ($W^{[1]}$) in a neural network are initialized to zero, the network won't work.

exp: the symmetry breaking problem

$w^{[2]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



if: $w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

then: $a_1^{[1]} = a_2^{[1]}$ $\quad dz_1^{[1]} = dz_2^{[1]}$

So ~~th~~ both units will be exactly the same, means they both calculate the same value and ~~it will after~~ they will compute the same value ~~after it end~~ in each iteration.

the solution: initializing the parameters randomly.

$w^{[1]} = np.random.randn((2,2)) * 0.01$

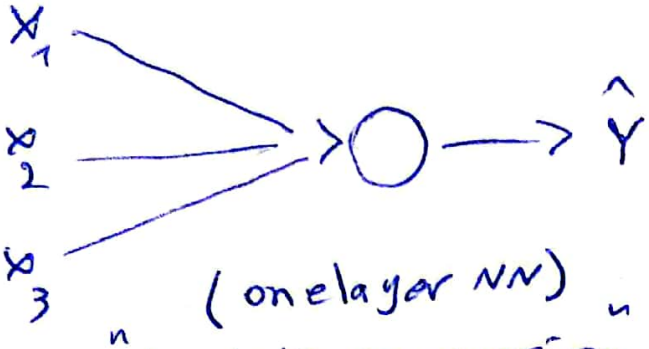
$b^{[1]} = np.zeros((2,1))$

$w^{[2]} = np.random.randn(\cdots) * 0.01$

$b^{[2]} = np.zeros(\cdots)$

Deep L-layer neural network

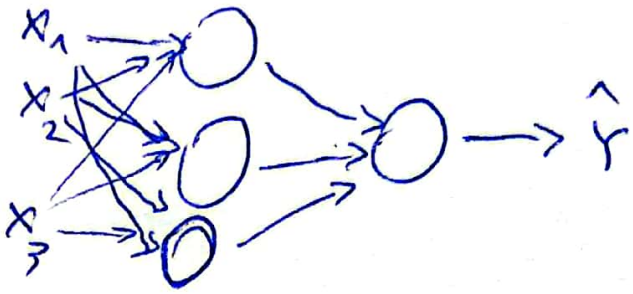
what is a deep neural network ?



(one layer NN)

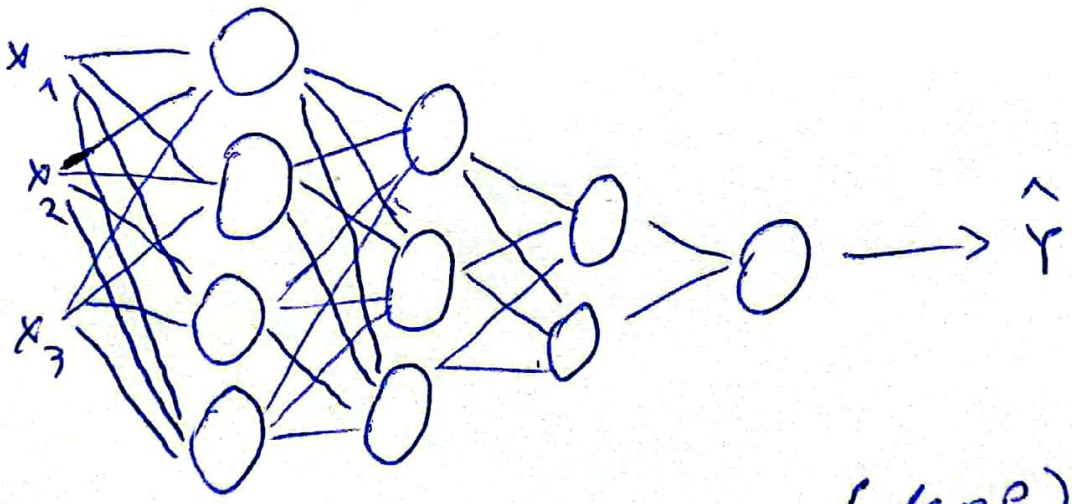
"logistic regression"

(shallow)



1 hidden layer

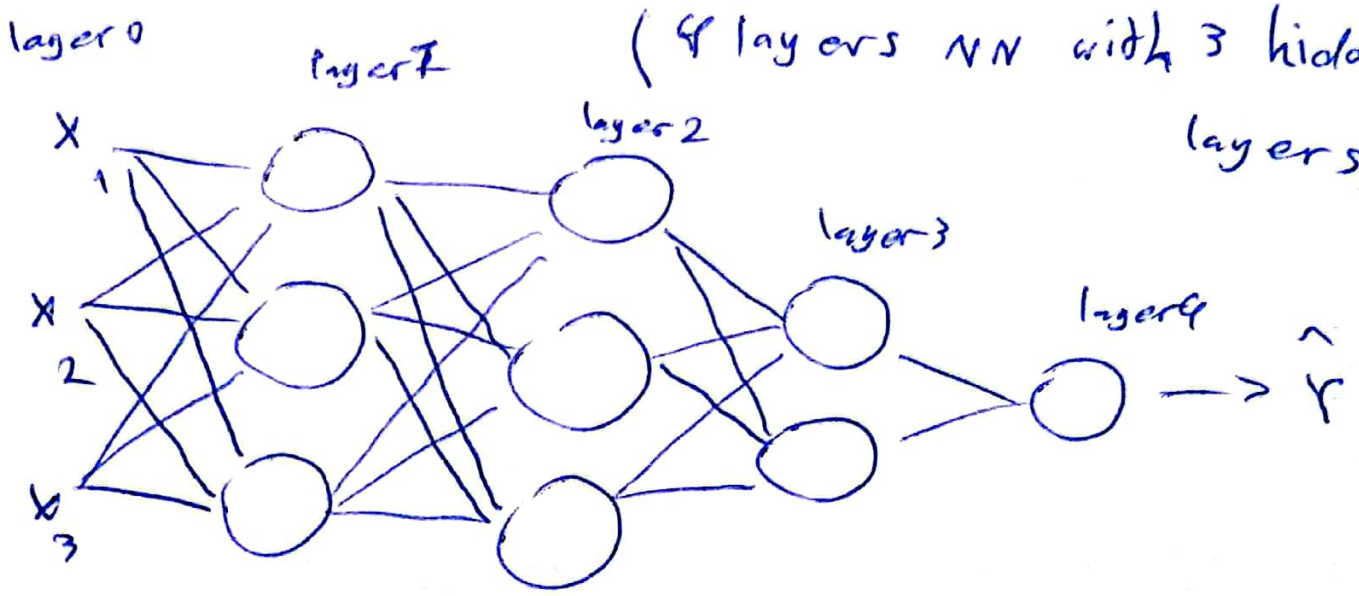
(2 layers NN)



3 hidden layers (deep)

(4 layers NN)

Deep neural network notation



(4 layers NN with 3 hidden layers)

$L = 4$ (# layers)

$n^{[l]}$ = # units in layer l

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

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activation function of layer l

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

$a^{[l]} = \max(z^{[l]}, 0)$

$X = a^{[0]}$; $\hat{y} = a^{[L]}$

"forward propagation in a deep network"

X (a single example): $\rightarrow$ or $a^{[0]}$

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

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

$$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

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

...

$$Z^{[4]} = W^{[4]} \cdot a^{[3]} + b^{[4]}$$

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

$$a^{[4]} = \hat{y}$$

General rule:

$$Z^{[l]} = W^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

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

Vectorized way $\rightarrow$ or $A^{[0]}$

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

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

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

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

...

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

for $L = 1, 2, 3, 4$

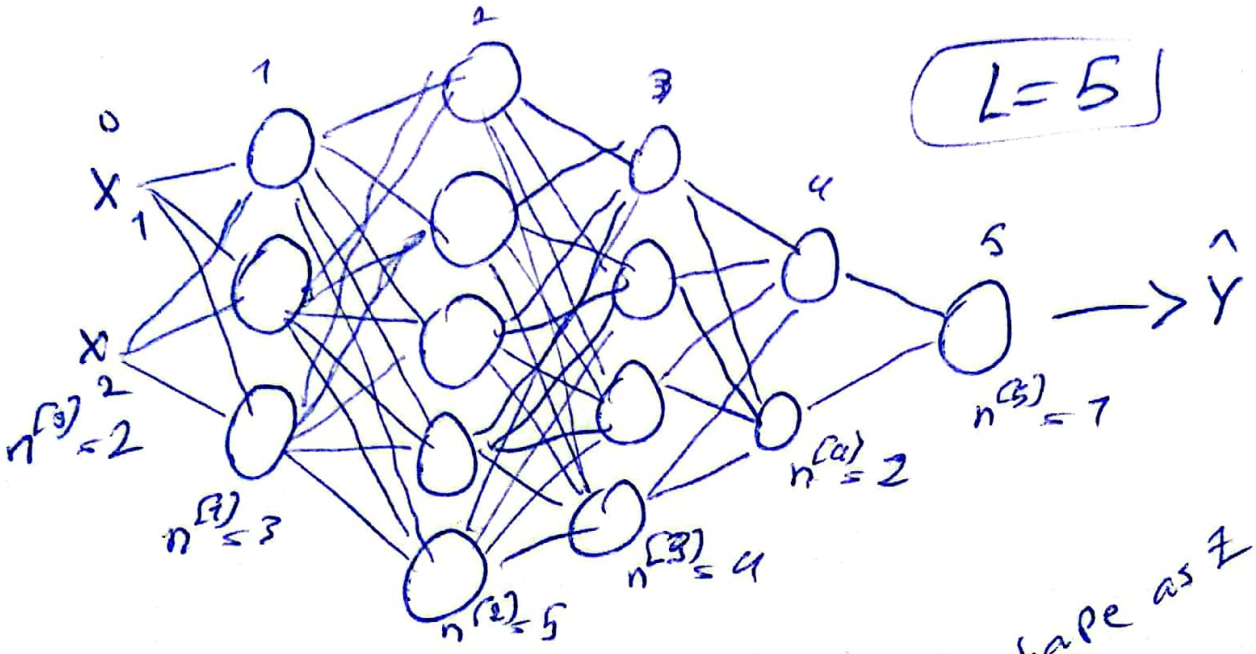
General rule:

$$Z^{[l]} = W^{[l]} \otimes A^{[l-1]} + b^{[l]}$$

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

Getting the matrix dimensions right

Parameters: $W^{[l]}$ and $b^{[l]}$



layer 1 ; $n^{[1]} = 3$

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

$(3,1)$ $(3,2)$ $(2,1)$ $(3,1)$

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

→ same shape as Z

$$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

in General:

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

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

$$dW^{[l]} = (W^{[l]}, W^{[l-1]})$$

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

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

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

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

vectorized

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

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

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

python broadcasting

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

number of examples (size of the training set)

$Z^{[L]} = A^{[L]} = (n^{[L]}, m)$

$A^{[0]} = X = (n^{[0]}, m)$

$dZ^{[1]} = dA^{[L]} = (n^{[L]}, m)$

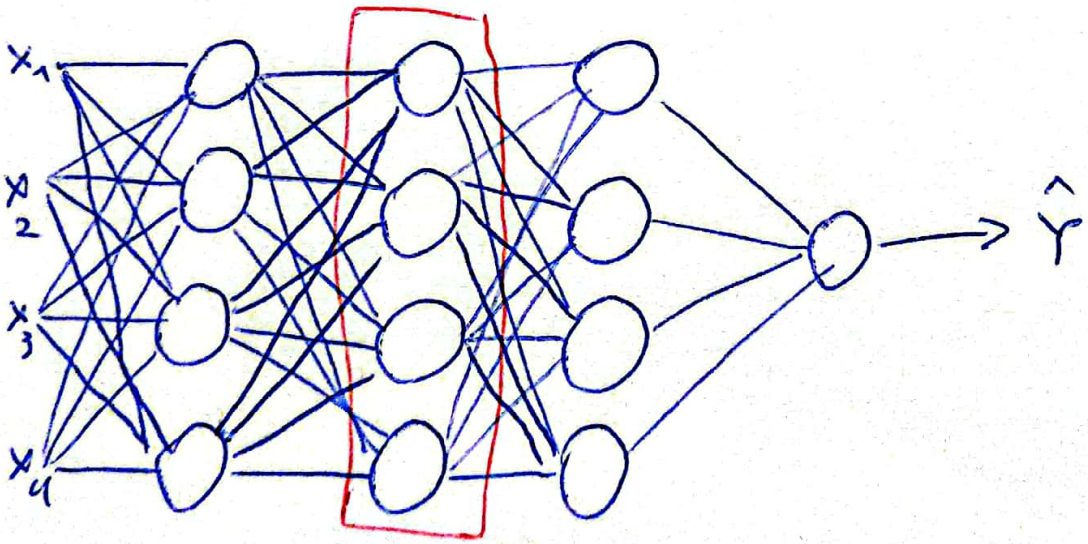
why deep representations?

"Circuit theory" and deep learning

informally: there are functions you can compute with a small L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

(small: number of units is relatively small)

"forward and backward functions"



layer l: $w^{[l]}$; $b^{[l]}$

forward: input $a^{[l-1]}$; output: $a^{[l]}$

$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$; cache $z^{[l]}$

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

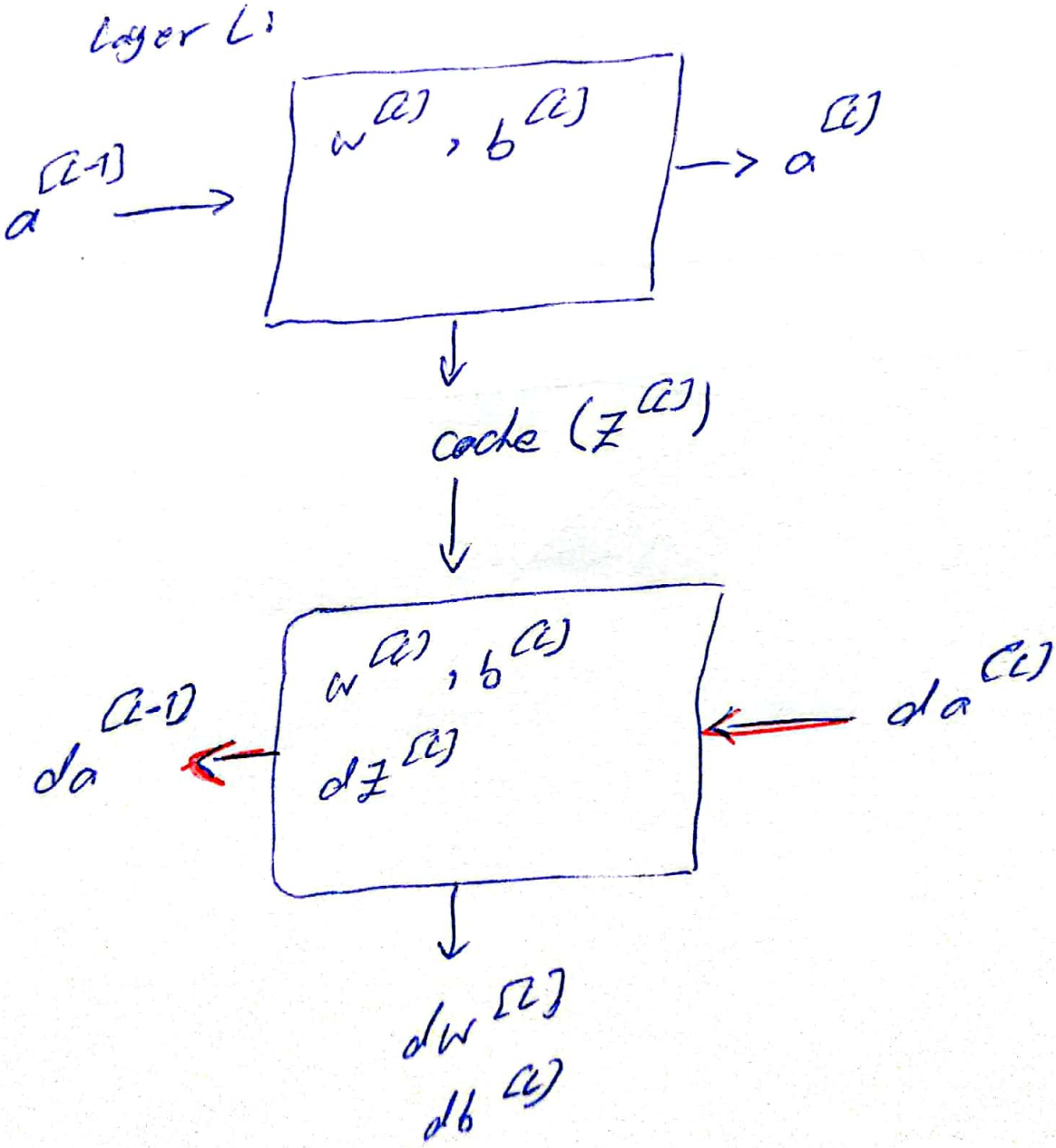
Backward: input: $da^{[l]}$; output: $da^{[l-1]}$

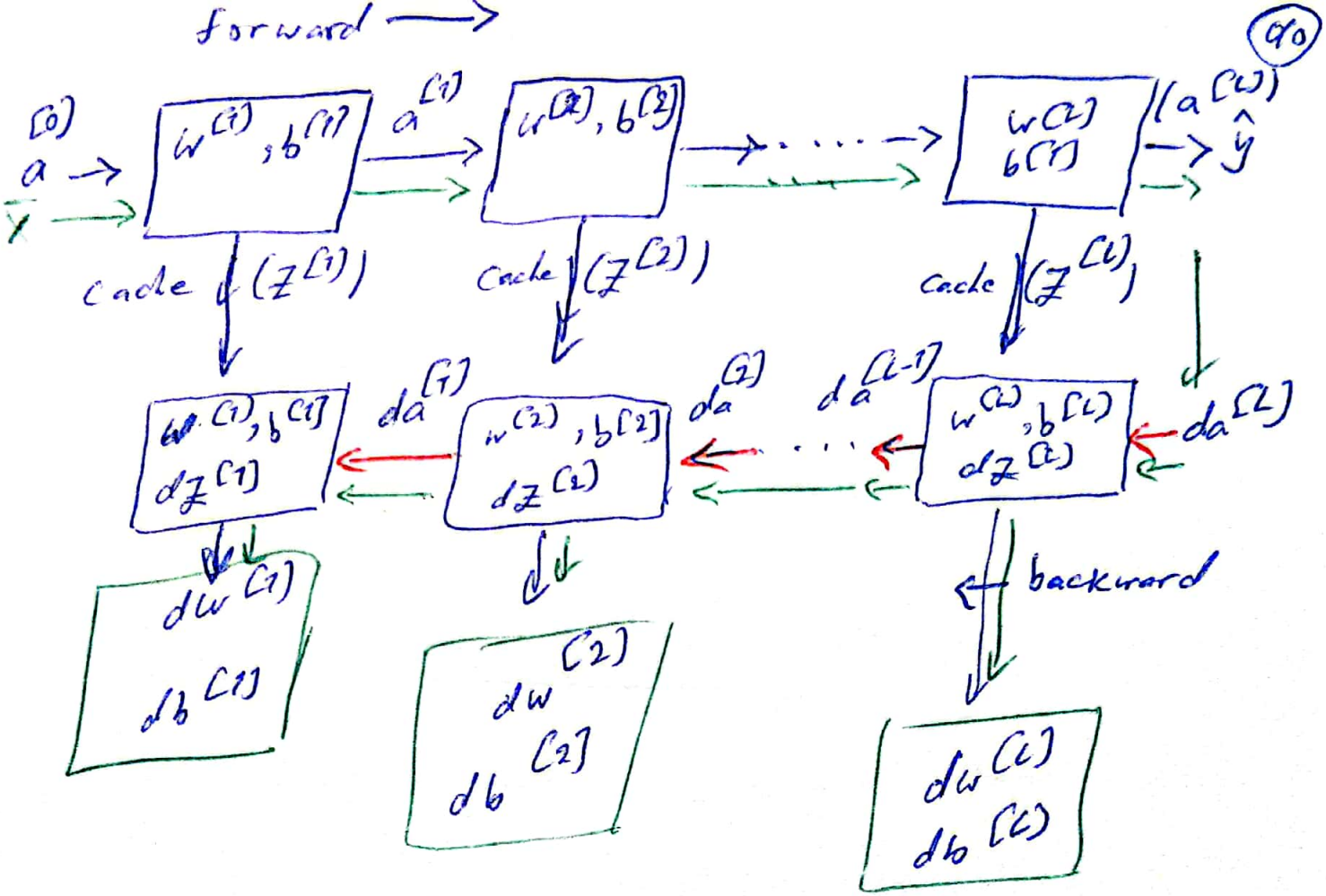
cache ($Z^{[l]}$)

~~cache~~

$dw^{[l]}$

$db^{[l]}$





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

$$b^{[L]} := b^{[L]} - \alpha\, db^{[L]}$$

---

forward propagation for layer L

input: $a^{[L-1]}$

output: $a^{[L]}$; cache($Z^{[L]}$)

$$Z^{[L]} = W^{[L]} \cdot a^{[L-1]} + b^{[L]}$$

$$a^{[L]} = g^{[L]}(Z^{[L]})$$

vectorized

$$Z^{[L]} = W^{[L]} \cdot A^{[L-1]} + b^{[L]}$$

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

$X = A^{[0]} \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow \cdots$

backward propagation for layer l:

Input $da^{[l]}$

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

$$dZ^{[l]} = da^{[l]} \times g^{[l]\prime}(Z^{[l]})$$

$$dW^{[l]} = dZ^{[l]} \cdot a^{[l-1]T}$$

$$db^{[l]} = dZ^{[l]}$$

$$da^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

$$dZ^{[l]} = W^{[l-1]T} \cdot dZ^{[l+1]} \times g^{[l]\prime}(Z^{[l]})$$

---

vectorized version

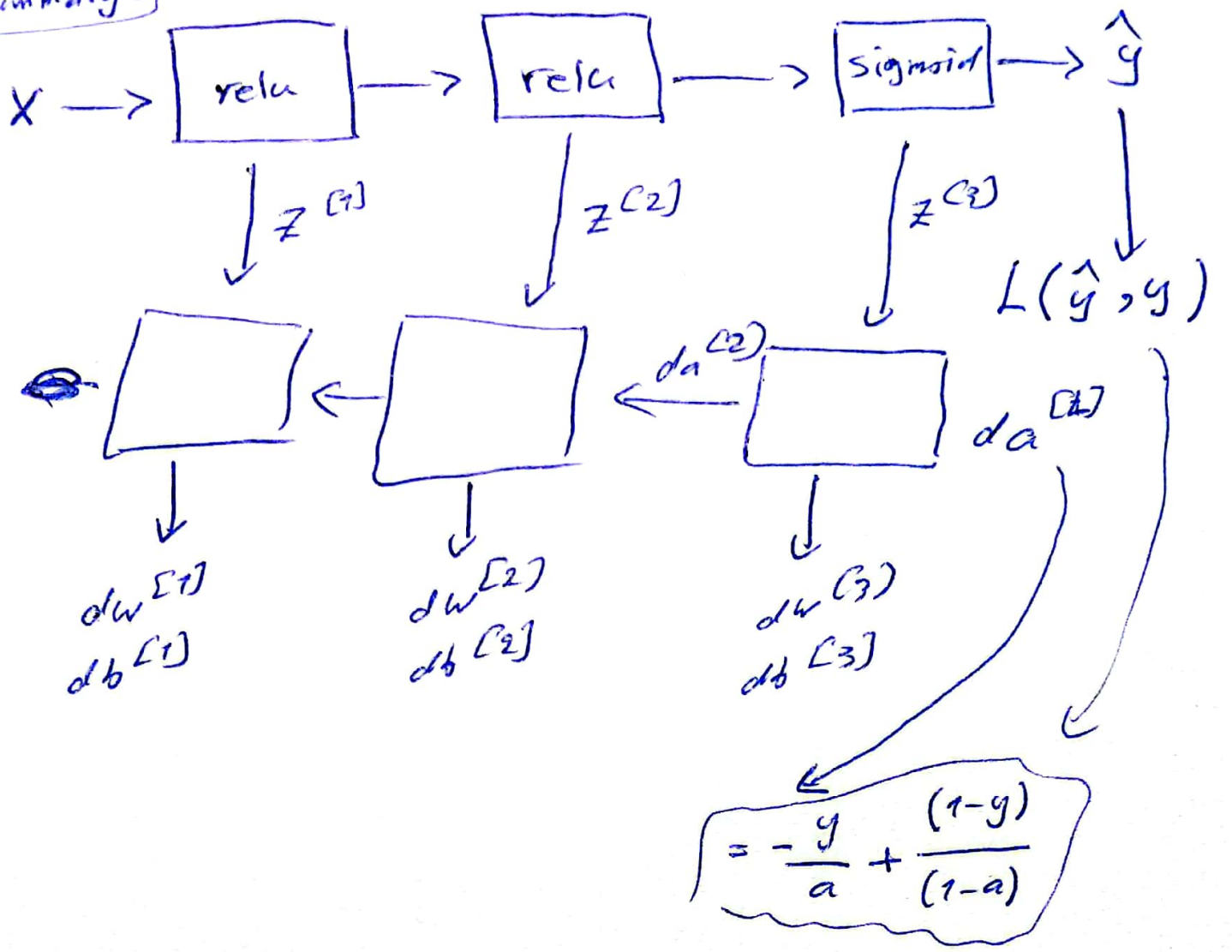
$$dZ^{[l]} = dA^{[l]} \times g^{[l]\prime}(Z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

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

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

Summary:



Parameters vs. hyperParameters

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots, W^{[L]}, b^{[L]}$

HyperParameters:
- learning rate (α)
- # iterations
- # hidden layers (L)
- # hidden units ($n^{[1]}, n^{[2]}, \ldots$)
- choice of activation function
- momentum, mini batch size, regularization

Parameters