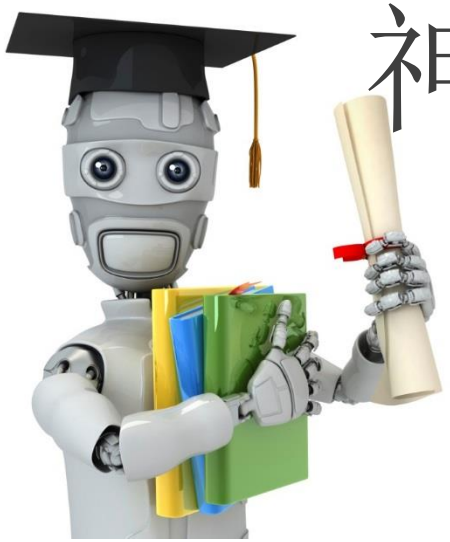


神经网络与深度学习 (Neural Networks & Deep Learning)

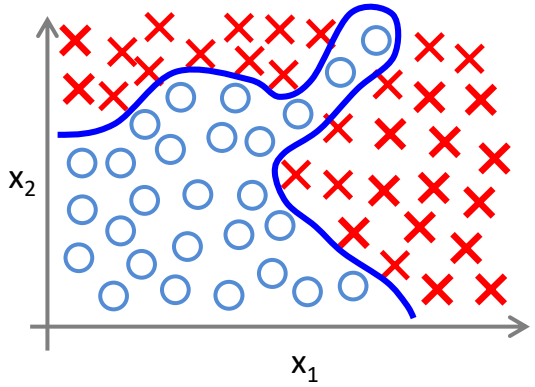
梁毅雄

yxliang@csu.edu.cn

Machine Learning

Some materials from Andrew Ng, Hung-yi Lee, Fei-Fei Li and others

非线性分类



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

x_1 = size
x_2 = # bedrooms
x_3 = # floors
x_4 = age
$\dots$
x_{100}

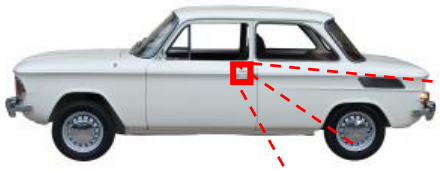
$$x_1^2, x_1 x_2, x_1 x_3, \cdots, x_1 x_{100}, x_2^2, x_2 x_3, \cdots \quad O(n^2)$$

$$x_1^3, x_1^2 x_2, x_1^2 x_3, \cdots x_1 x_2 x_3, \cdots \quad O(n^3)$$

非线性分类

What is this?

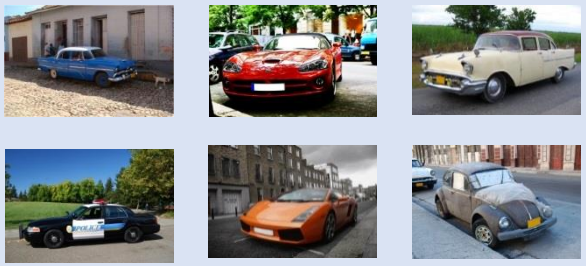
You see this:



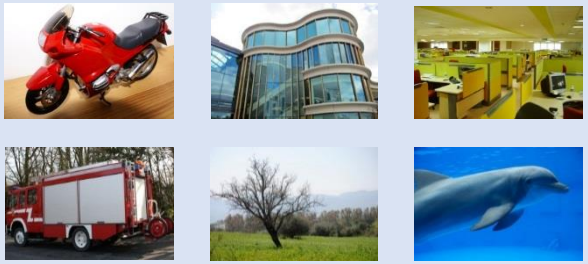
But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Computer Vision: Car detection



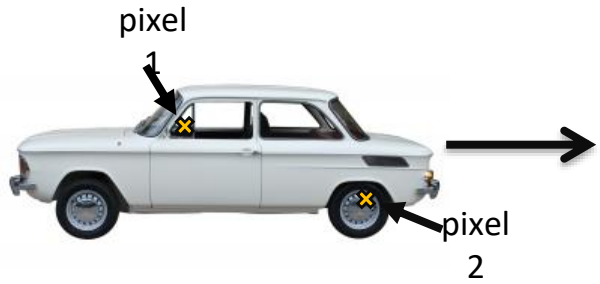
Cars



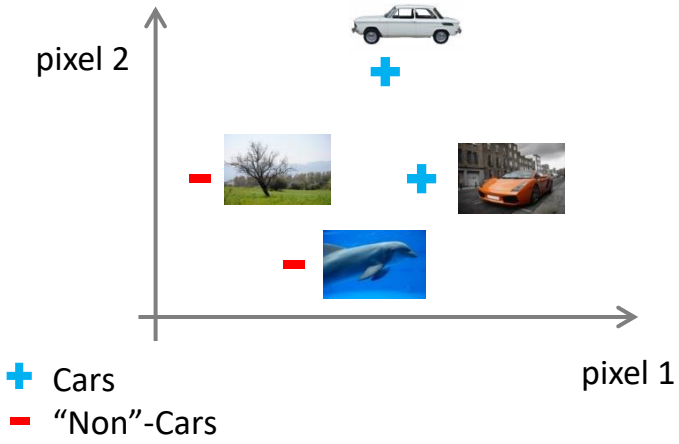
Not a car

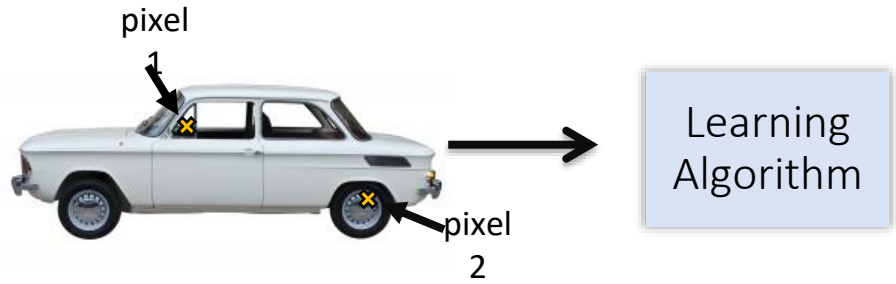
Testing:

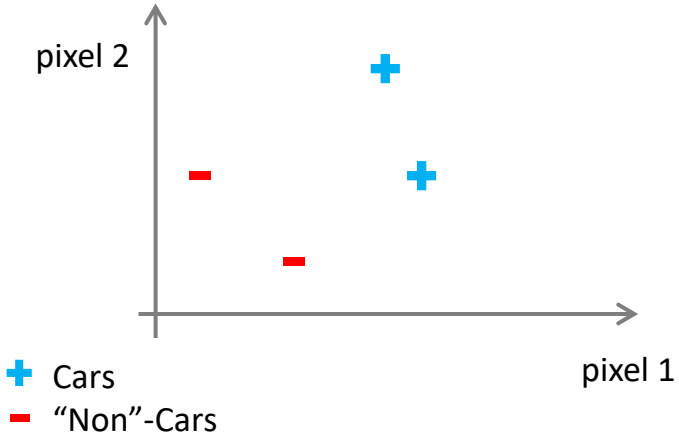
What is this?

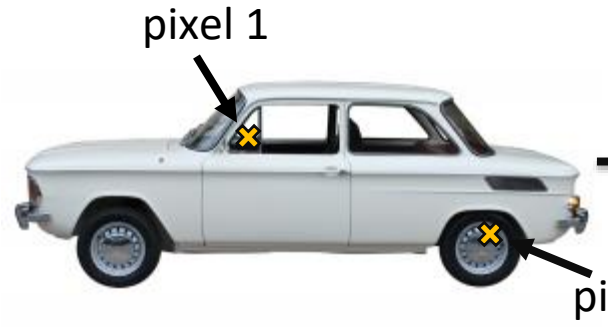


Learning Algorithm

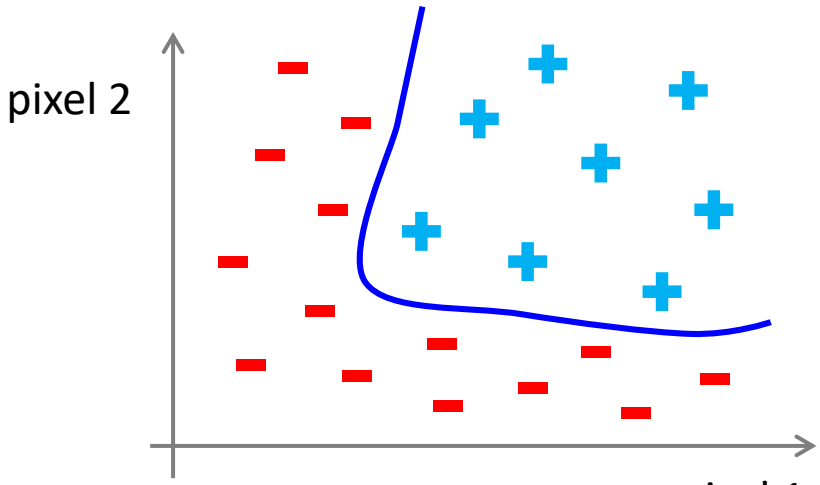



pixel 1
pixel 2
Learning Algorithm


pixel 2
pixel 1
Cars
"Non"-Cars



Learning Algorithm



Cars

"Non"-Cars

50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

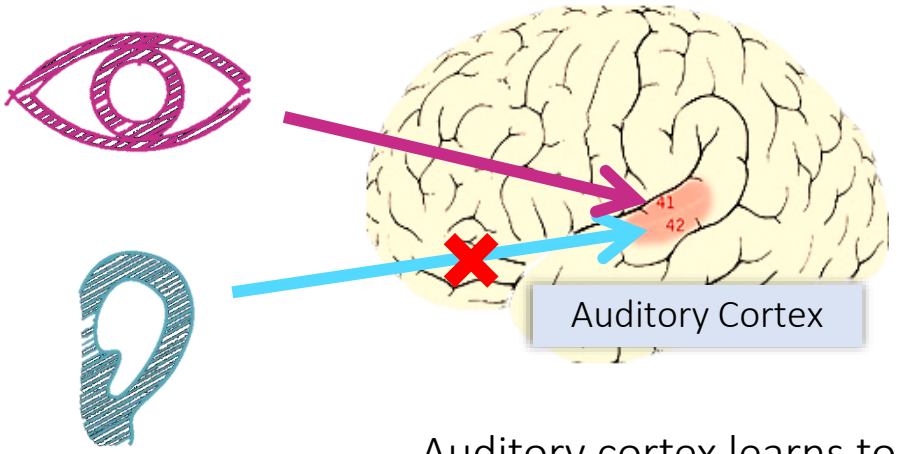
Quadratic features ($x_i \times x_j$): ≈ 3 million features

神经网络发展史

- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is “good enough”, why deep?

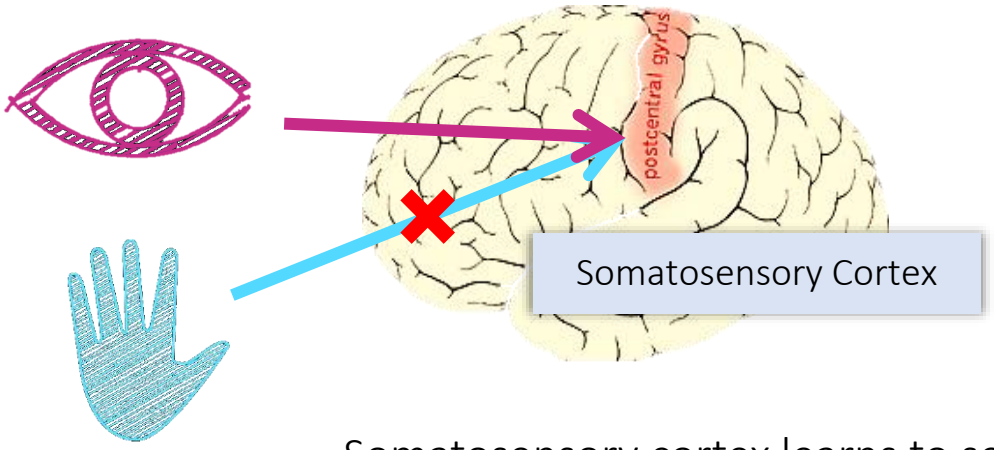
- 2006: RBM initialization
- 2009: GPU
- 2011: Start to be popular in speech recognition
- 2012: win ILSVRC image competition
- 2015.2: Image recognition surpassing human-level performance
- 2016: Alpha GO beats human
- 2016: Speech recognition system as good as humans

The “one learning algorithm” hypothesis



Auditory cortex learns to see

The “one learning algorithm” hypothesis

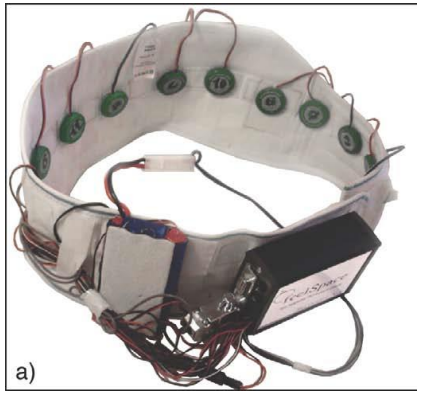


Somatosensory cortex learns to see

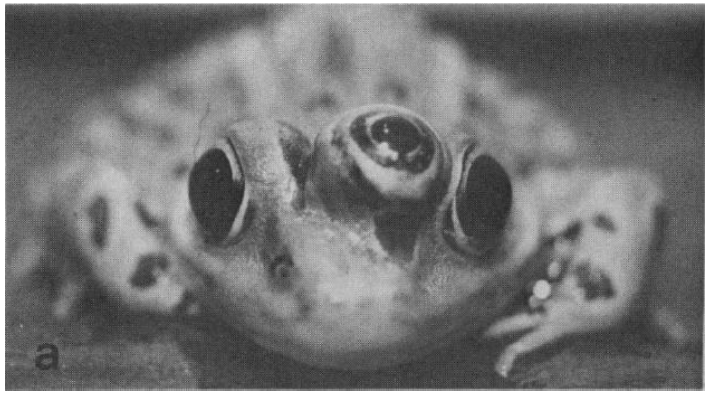
Sensor representations in the brain

Seeing with your tongue

Human echolocation (sonar)

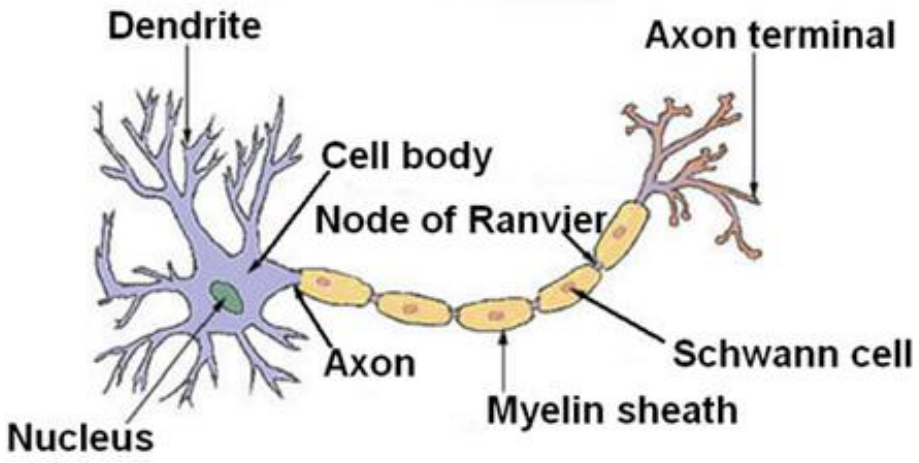


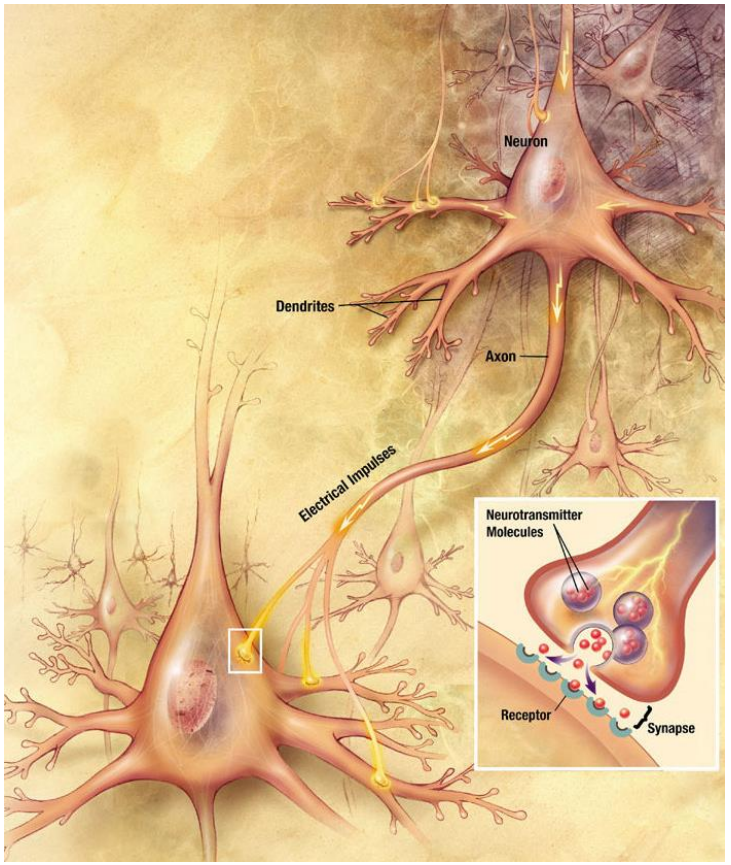
Haptic belt: Direction sense



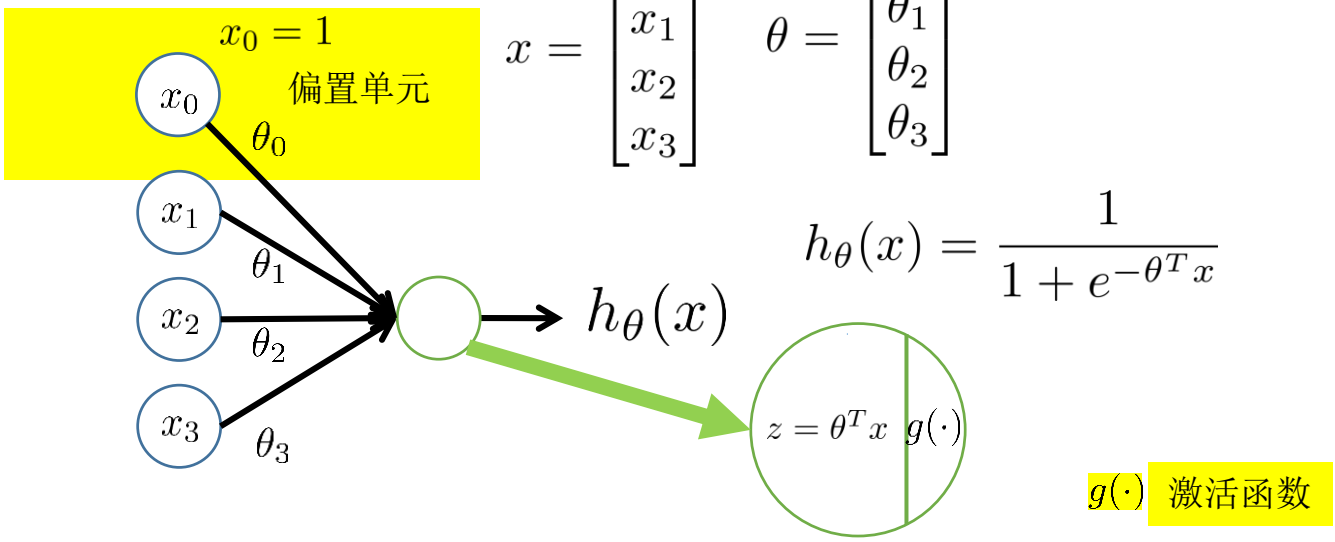
Implanting a 3rd eye

Neuron in the brain





神经元模型: Logistic unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

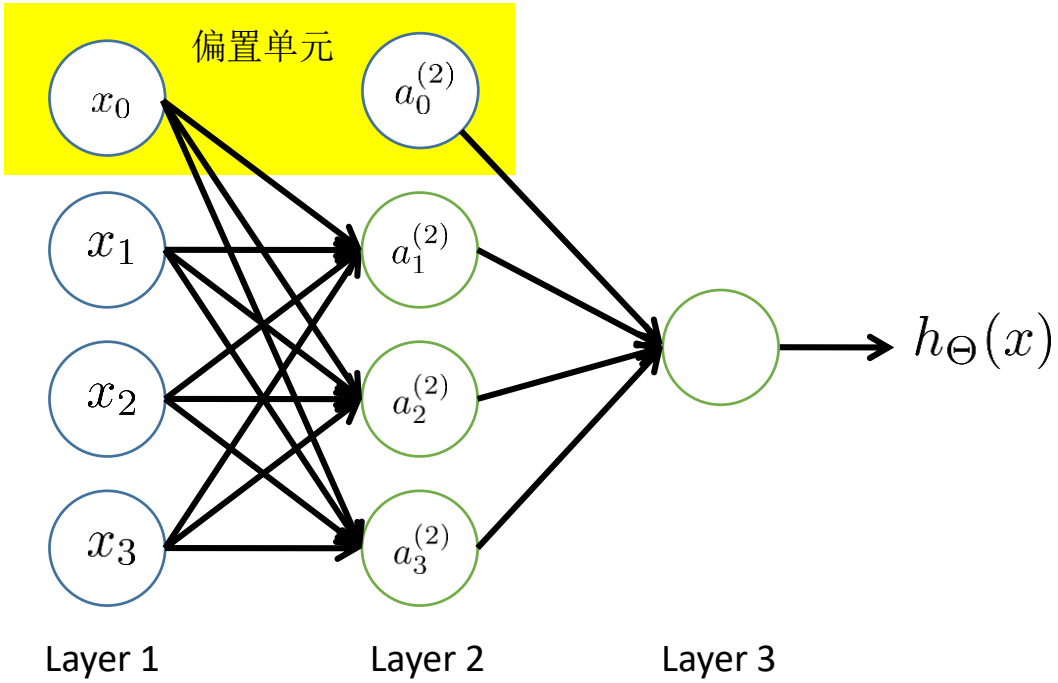
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$g(\cdot)$ 激活函数

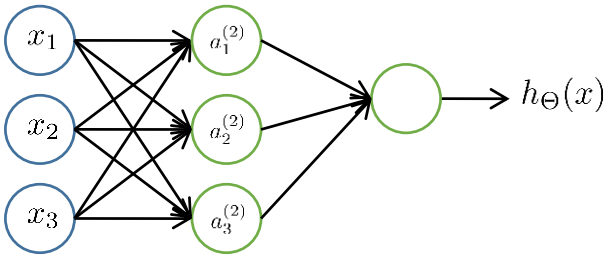
Sigmoid (logistic) activation function.

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

全连接前馈网络 (Fully Connect Feedforward Network)



全连接前馈网络 (Fully Connect Feedforward Network)



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(z_1^{(2)}) = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

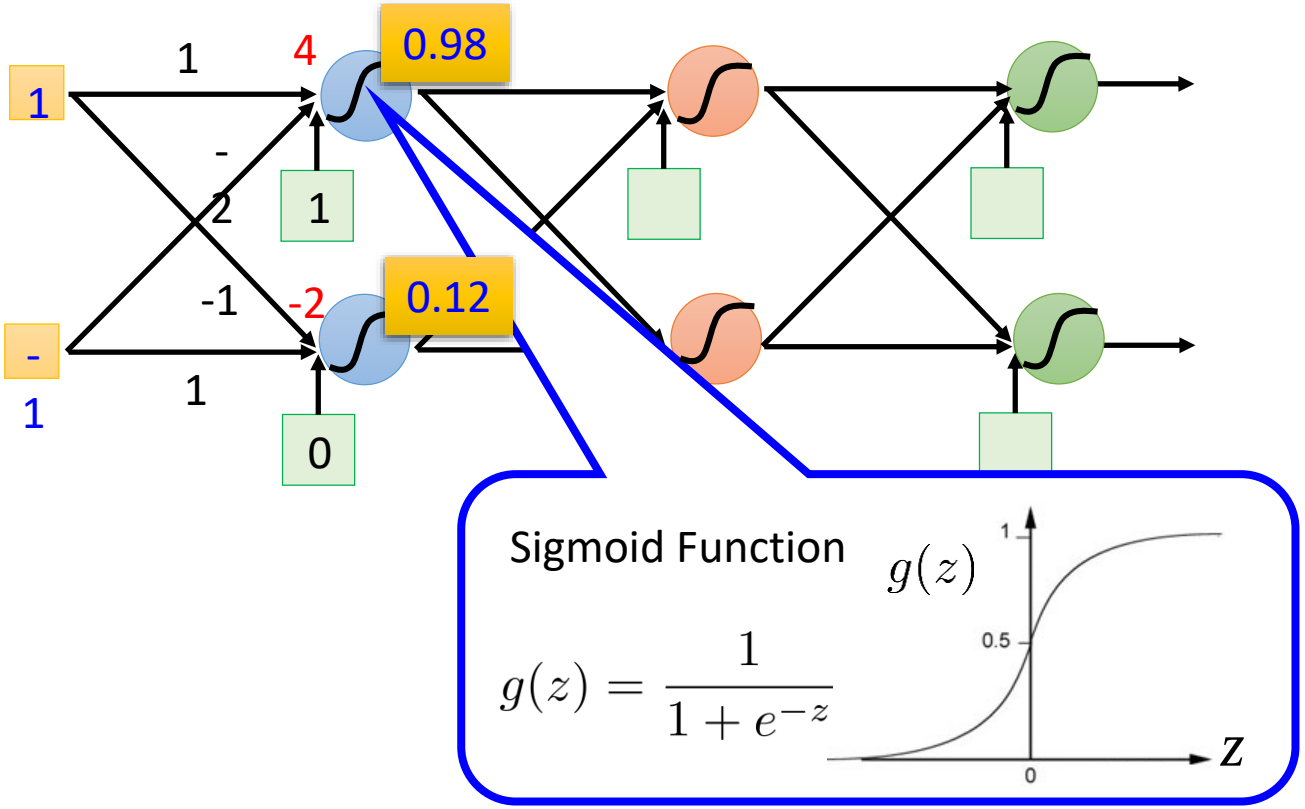
$$a_2^{(2)} = g(z_2^{(2)}) = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(z_3^{(2)}) = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

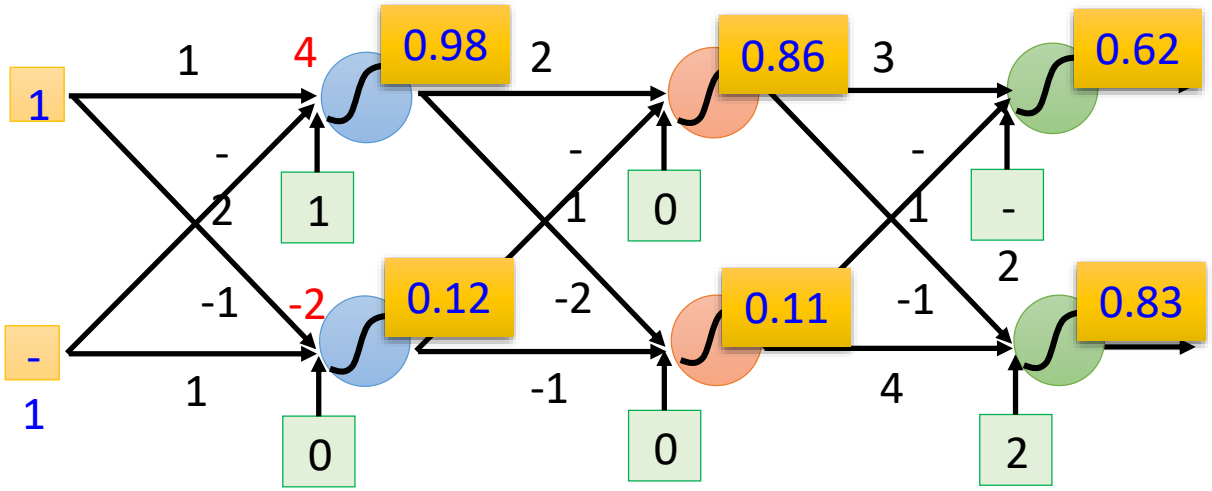
$$h_\Theta(x) = a_1^{(3)} = g(z_1^{(3)}) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$

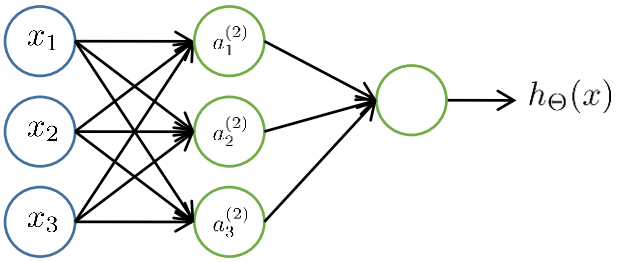
全连接前馈网络 (Fully Connect Feedforward Network)



全连接前馈网络 (Fully Connect Feedforward Network)



前向传播：矩阵表示



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

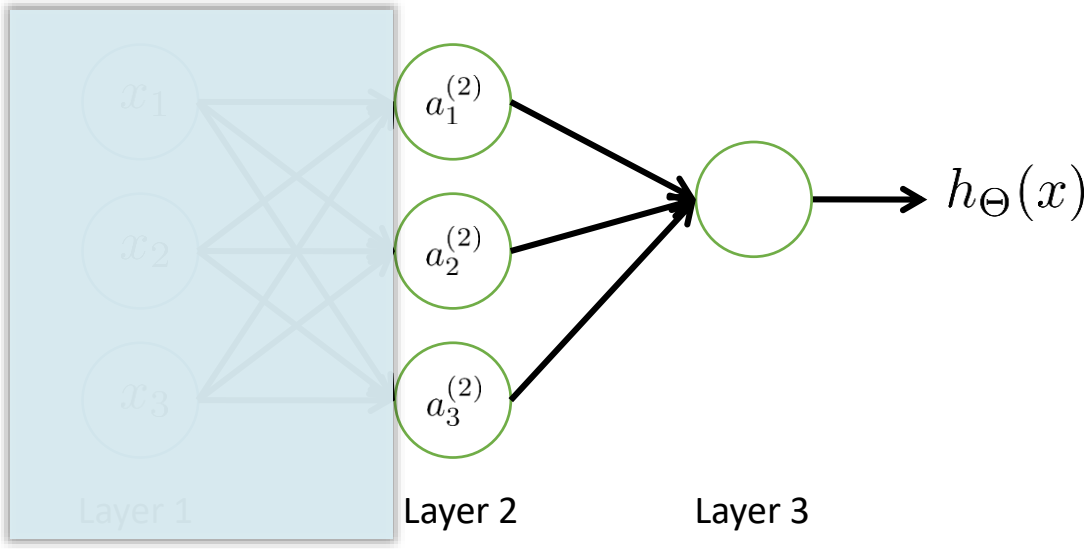
$$z^{(2)} = \Theta^{(1)}x$$
$$a^{(2)} = g(z^{(2)})$$

Add $a_0^{(2)} = 1$.

$$z^{(3)} = \Theta^{(2)}a^{(2)}$$
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

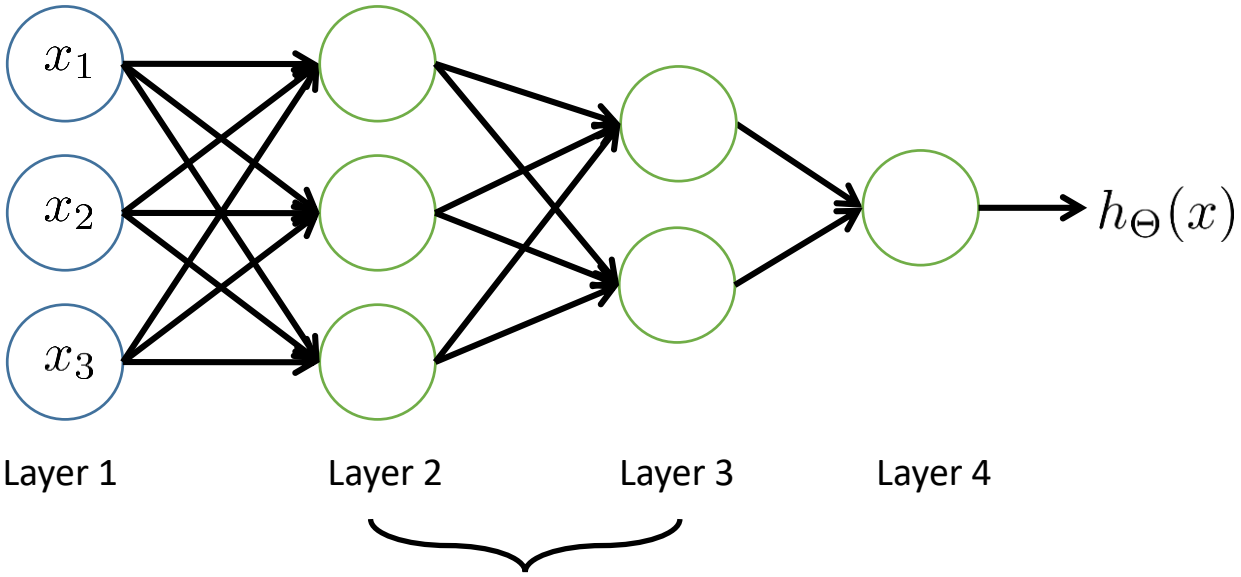
$$h_\Theta(x) = g(\Theta^{(2)}g(\Theta^{(1)}x))$$

特征学习



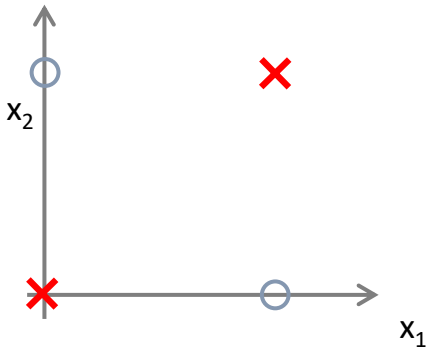
多层神经网络

$$h_\Theta(x) = g(\Theta^{(3)} g(\Theta^{(2)} g(\Theta^{(1)} x)))$$



用神经网络求解XOR/XNOR问题

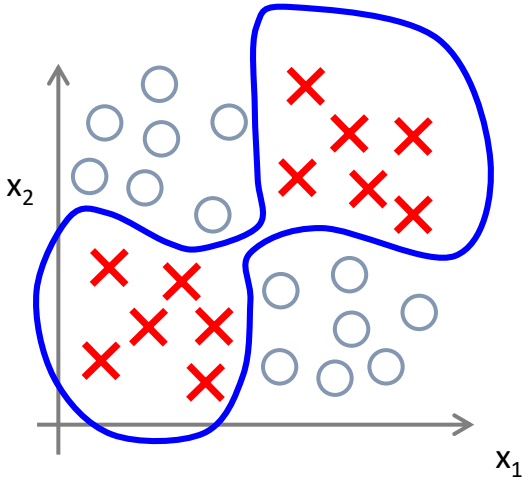
x_1, x_2 are binary (0 or 1).

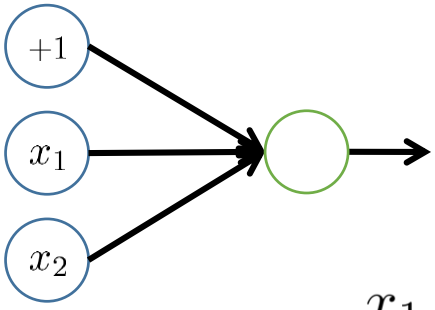


$y = x_1 \text{ XOR } x_2$

$x_1 \text{ XNOR } x_2$

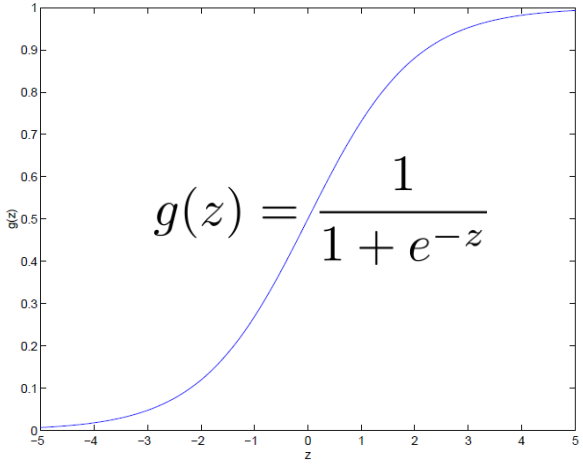
$\text{NOT } (x_1 \text{ XOR } x_2)$





$$h_\Theta(x) = g(\theta^T x)$$

$$x_1, x_2 \in \{0, 1\}$$



$y = x_1$ AND x_2

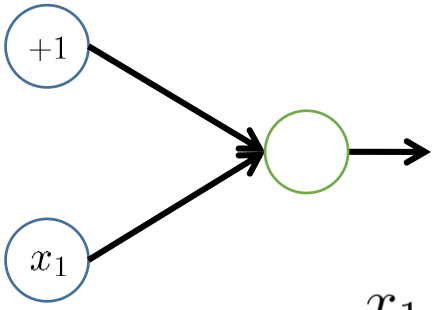
$$\theta = \begin{bmatrix} -30 \\ 20 \\ 20 \end{bmatrix}$$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$y = x_1$ OR x_2

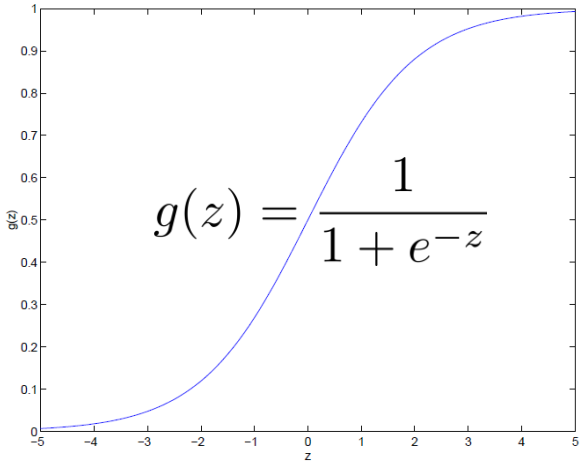
$$\theta = \begin{bmatrix} -10 \\ 20 \\ 20 \end{bmatrix}$$

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$g(10) \approx 1$
1	1	$g(30) \approx 1$



$$h_\Theta(x) = g(\theta^T x)$$

$$x_1, x_2 \in \{0, 1\}$$



NOT x_1

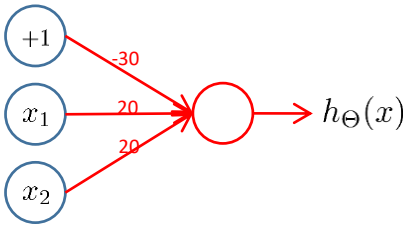
$$\theta = \begin{bmatrix} 10 \\ -20 \end{bmatrix}$$

x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

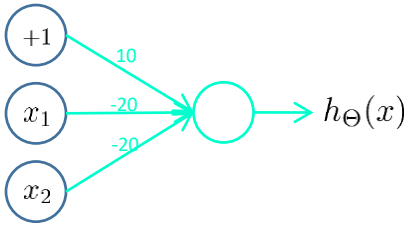
$$h_\Theta(x) = g(10 - 20x_1)$$

(NOT x_1) AND (NOT x_2) $= 1$ if and only if $x_1 = x_2 = 0$

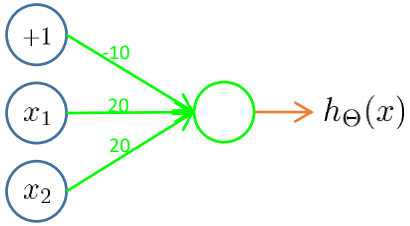
x_1 XNOR x_2



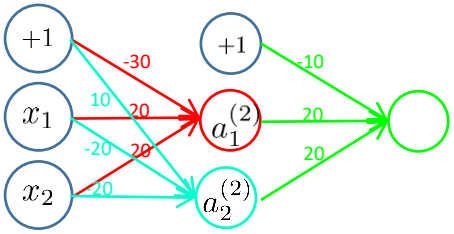
x_1 AND x_2



(NOT x_1) AND (NOT x_2)



x_1 OR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

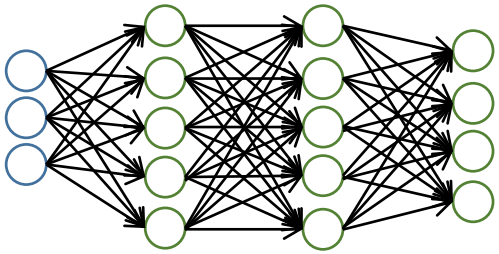
处理多分类问题


Pedestrian


Car


Motorcycle


Truck



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

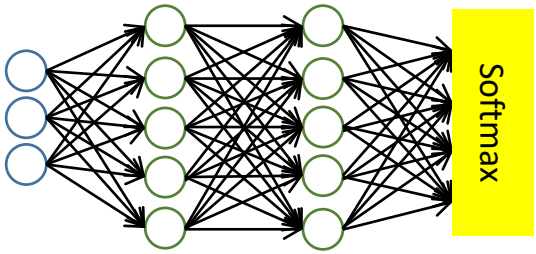
处理多分类问题


Pedestrian


Car


Motorcycle

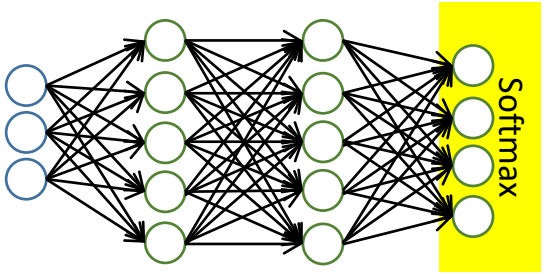

Truck



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

处理多分类问题



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

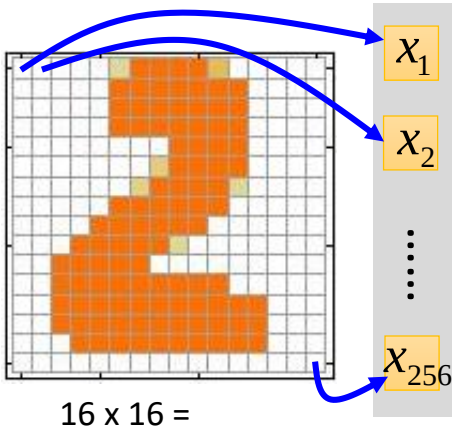
Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

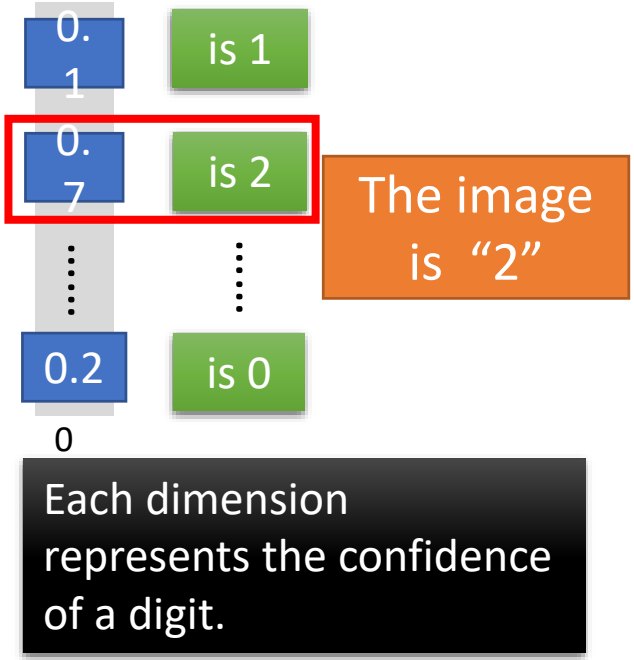
pedestrian car motorcycle truck

手写数字识别

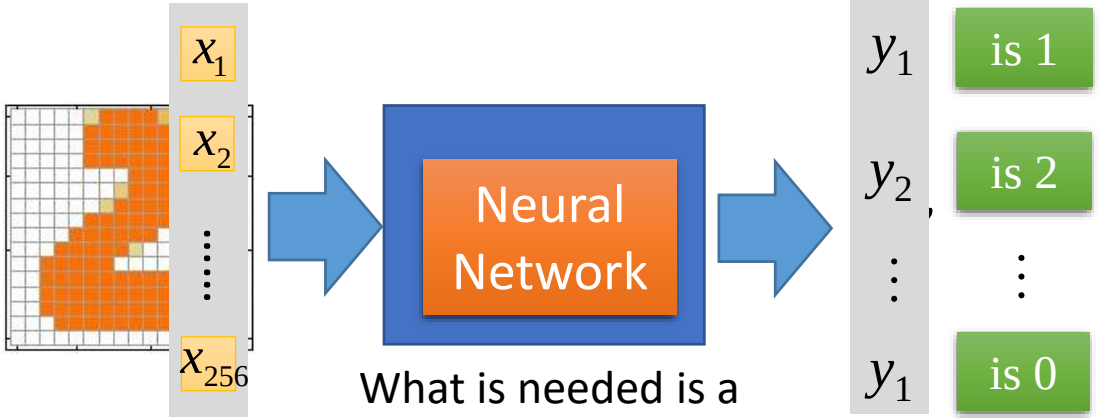
- Input



- Output



手写数字识别

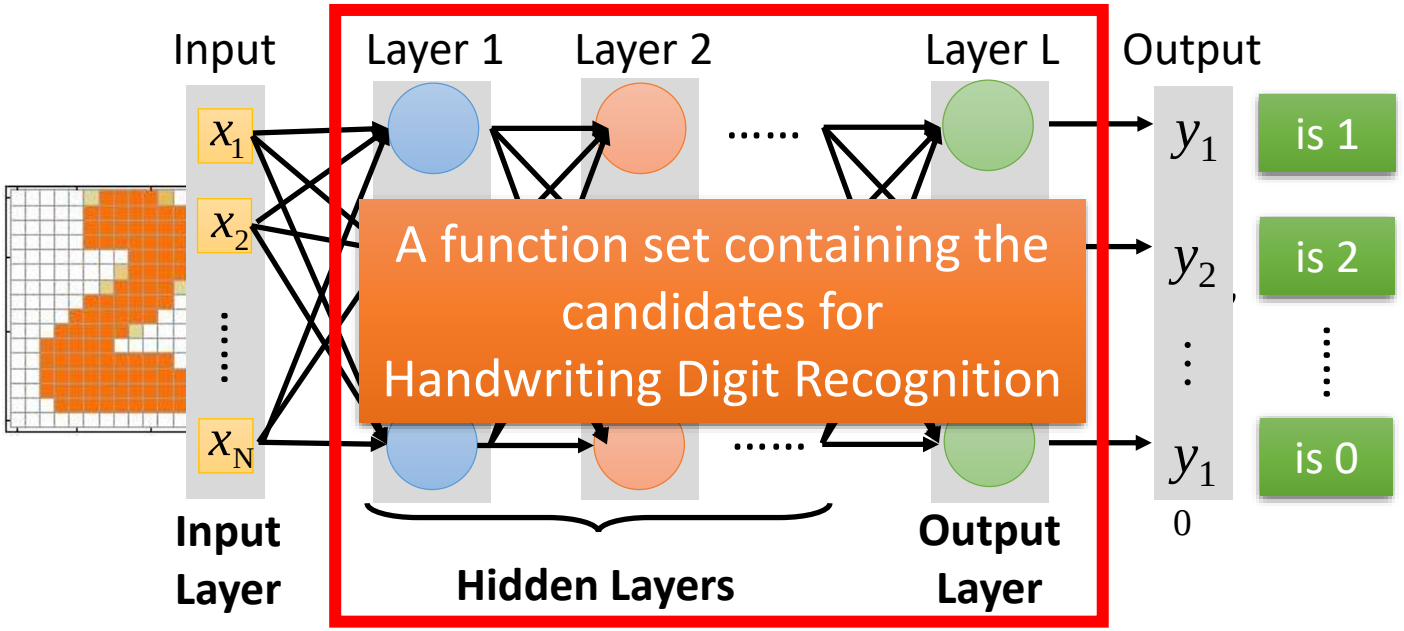


What is needed is a function ……

Input: 256-dim vector

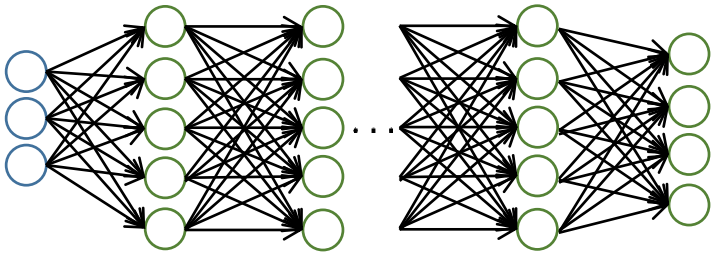
output: 10-dim vector

手写数字识别



You need to decide the network structure to let a good function in your function set.

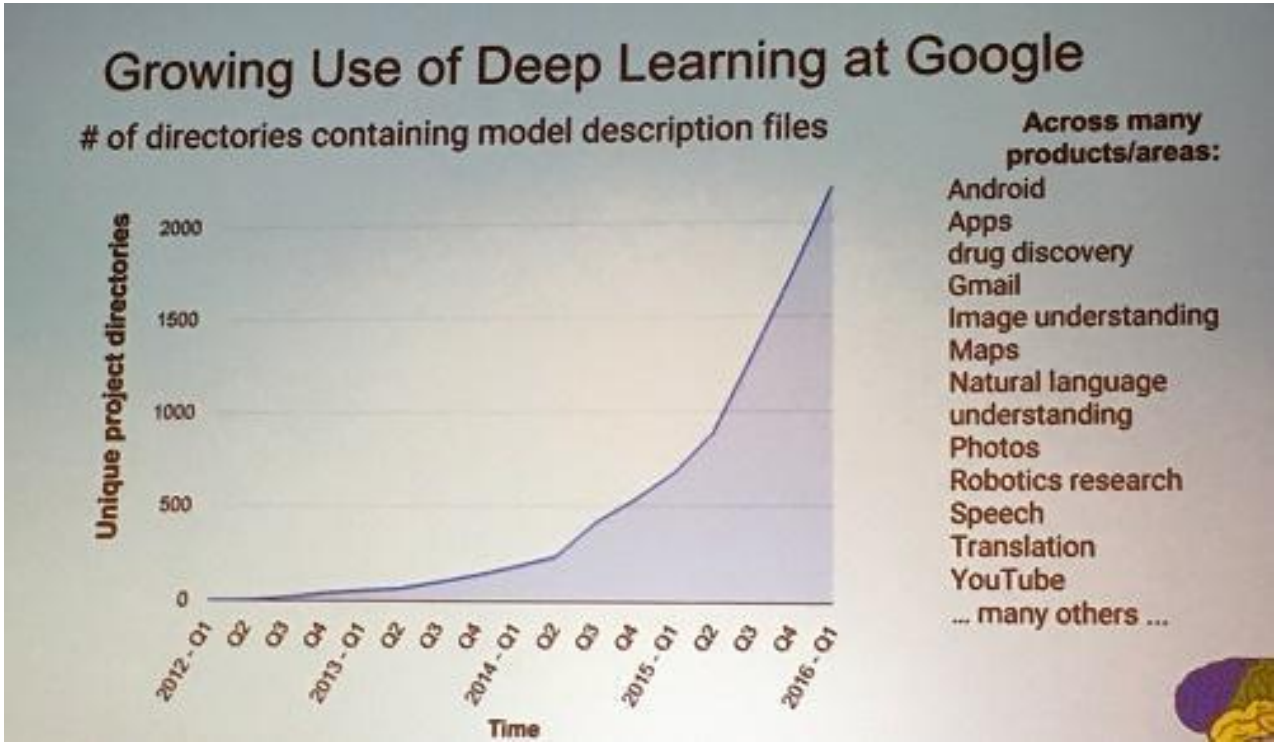
网络结构



$$h_\Theta(x) = g_L(\Theta^{(L-1)} \cdots g_2(\Theta^{(2)} g_1(\Theta^{(1)} x)))$$

- 不同的网络结构对应不同的假设函数
  - 多少层？
  - 每层多少个神经元？
  - 神经元类型？
  - 层之间的连接关系
  - ...

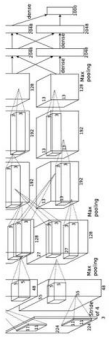
Deep Learning: Many hidden layers



Deep learning trends at Google. Source: SIGMOD 2016/Jeff Dean

Deep: Many hidden layers

8 layers



16.4%

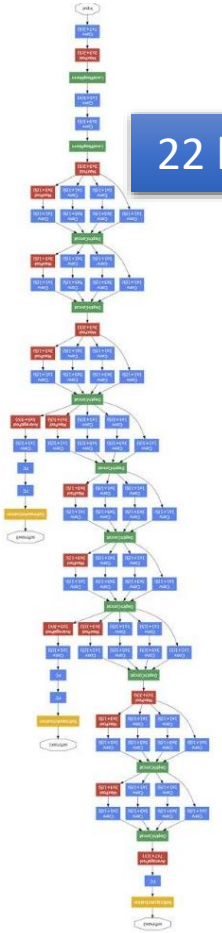
AlexNet (2012)

19 layers

7.3%

VGG (2014)

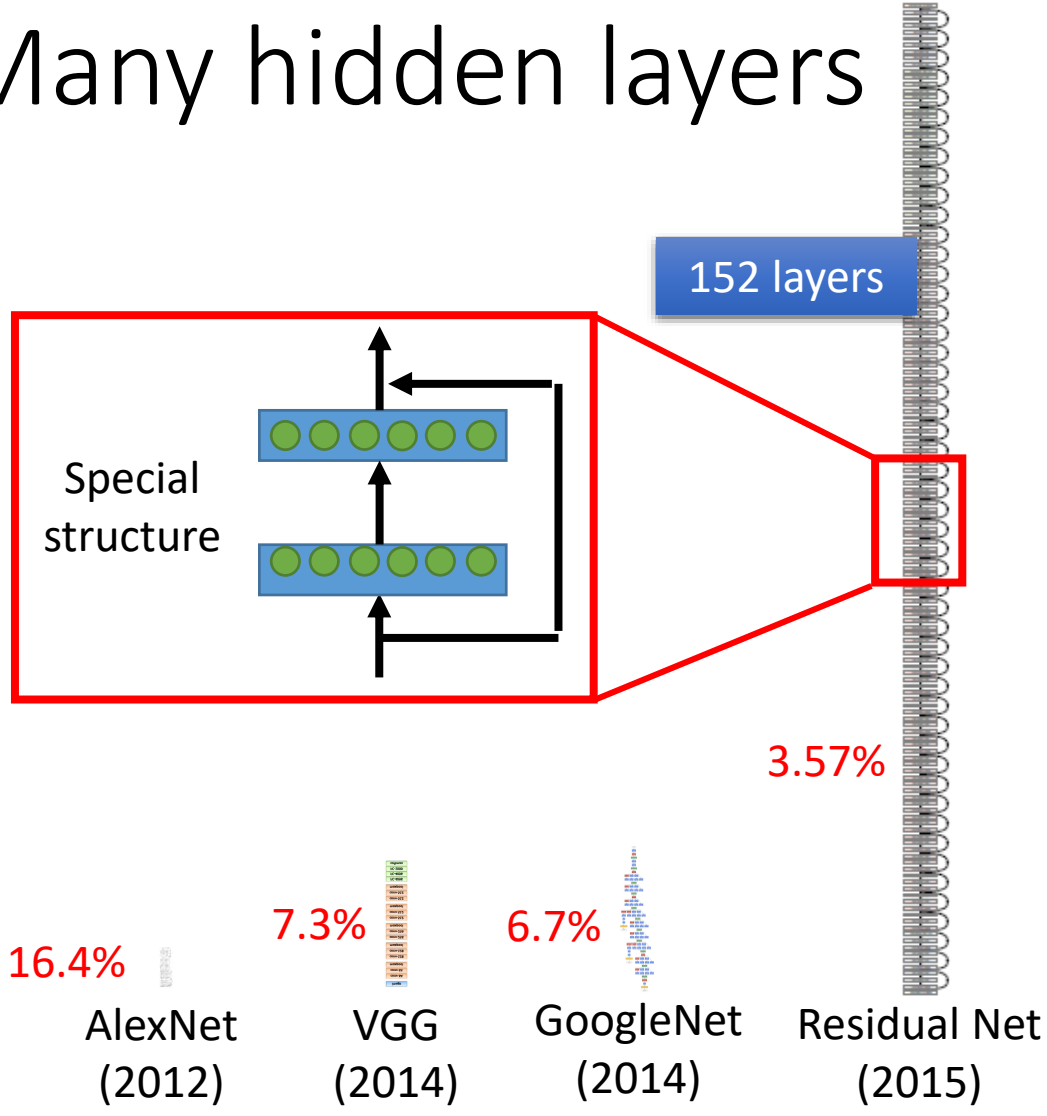
22 layers



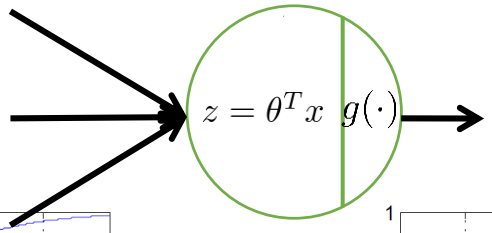
6.7%

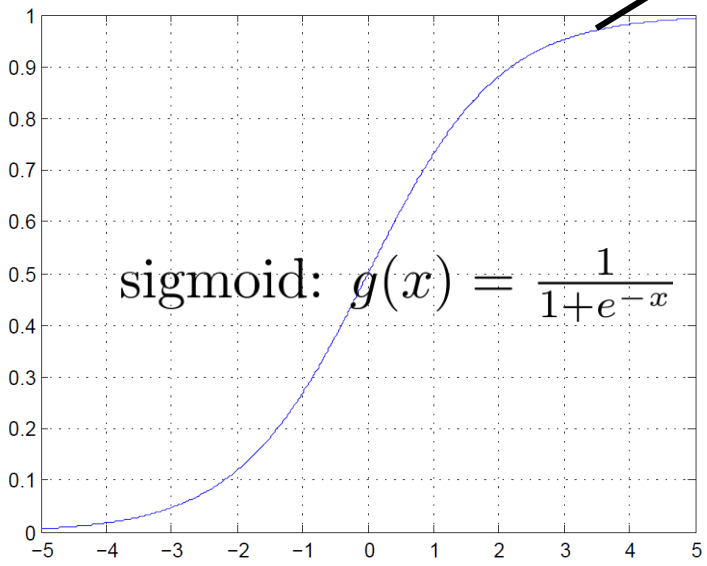
GoogleNet (2014)

Deep: Many hidden layers

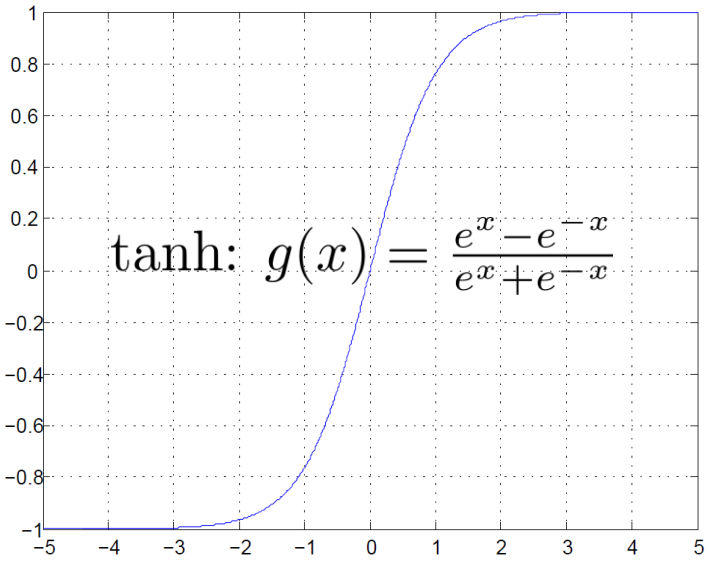


激活函数



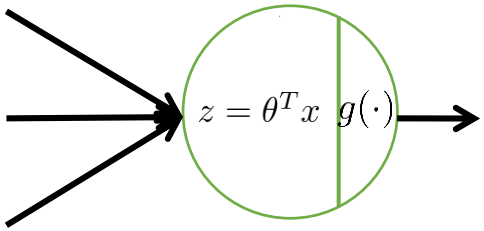


1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive



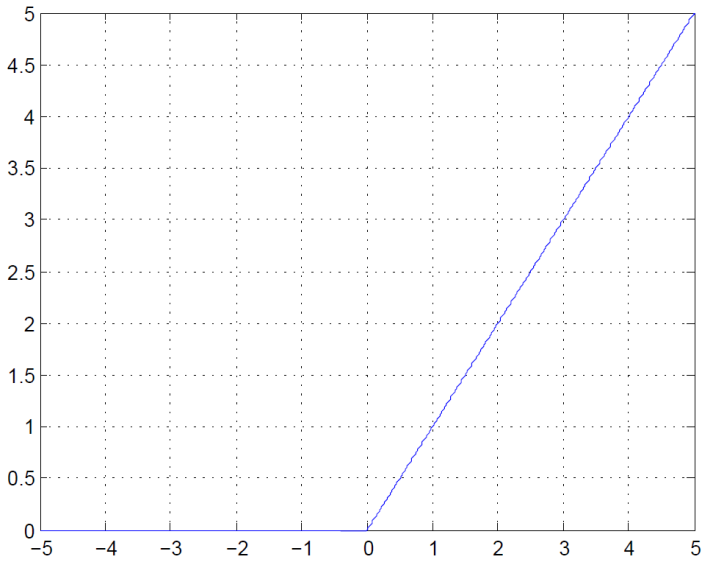
1. Zero centered (nice)
2. “kill” the gradients
3. exp() is a bit compute expensive

激活函数



ReLU (Rectified Linear Unit)

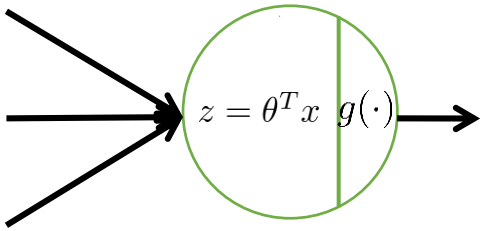
$g(x) = \max(0, x)$



1. Does not saturate (in +region)
2. Very computationally efficient
3. Converges much faster than sigmoid/tanh in practice (e.g. 6x)
4. Actually more biologically plausible than sigmoid

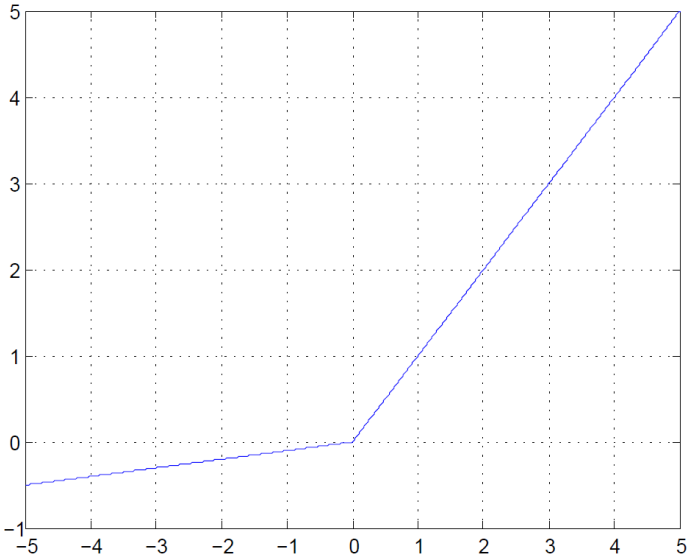
5. Not zero-centered output
6. $x<0$: dead ReLU will never activate => never update

激活函数



Leaky Relu:

$$g(x) = \max(0.1x, x)$$



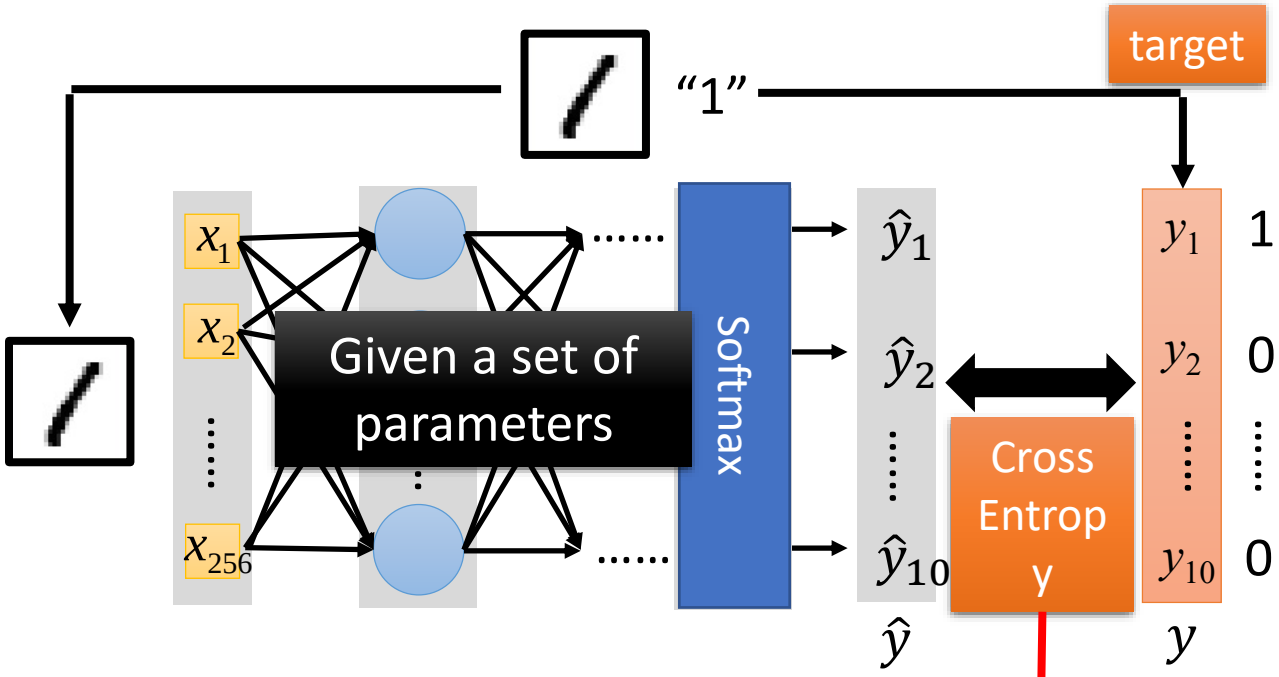
1. Does not saturate (in +region)
2. Very computationally efficient
3. Converges much faster than sigmoid/tanh in practice (e.g. 6x)
4. Actually more biologically plausible than sigmoid

5. Not zero-centered output
6. Will not “die”

Parametric Rectifier (PReLU):

$$g(x) = \max(\alpha x, x)$$

损失函数



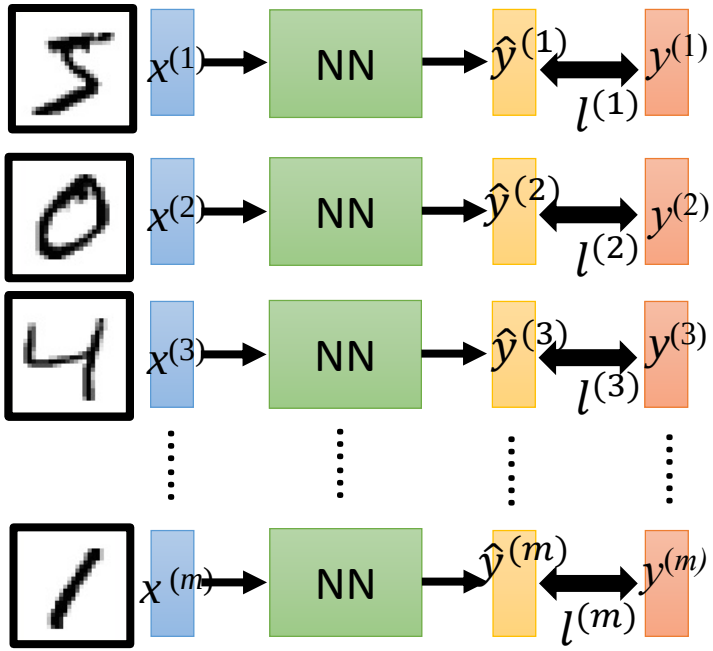
$$l(y, \hat{y}) = -\sum_{i=1}^{10} y_i \log(\hat{y}_i)$$

损失函数

Total Loss:

$$L = \sum_{i=1}^{m} l^{(i)}$$

For all training data …



Find ***a function in function set*** that minimizes total loss L

Find ***the network parameters*** θ^* that minimize total loss L

目标函数

$$J(\Theta) = L(\Theta) + \lambda R(\Theta)$$

Logistic regression:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural network (with softmax loss):

$$h_\Theta(x) \in \mathbb{R}^K \qquad (h_\Theta(x))_k = k^{th} \text{ output}$$

$$\begin{aligned} J(\Theta) &= -\frac{1}{m}\left[\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)};\Theta)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2 \\ &= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2 \end{aligned}$$

Cross Entropy

L2 Regularization (Weight Decay)

梯度下降法

- 仍采用梯度下降法：Alpha Go

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$

- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

梯度计算：反向传播 (Backpropagation, BP)

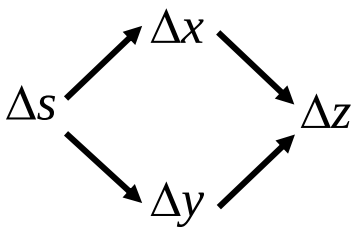
链式法则

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \to \Delta y \to \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

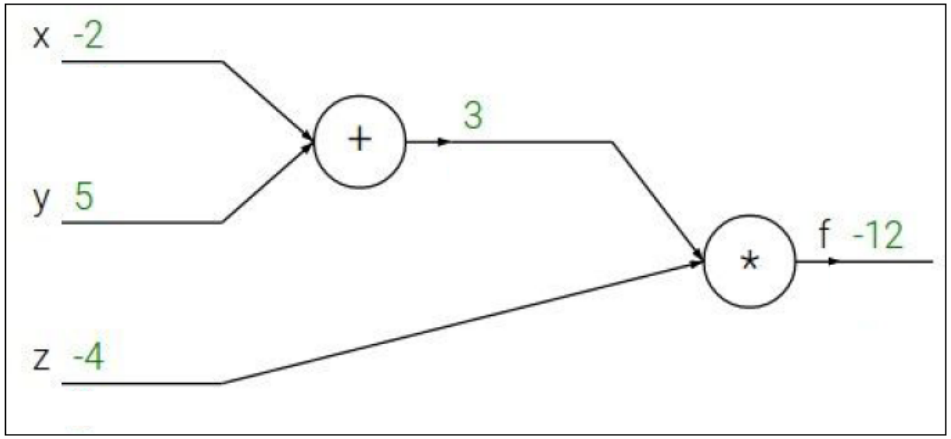


$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

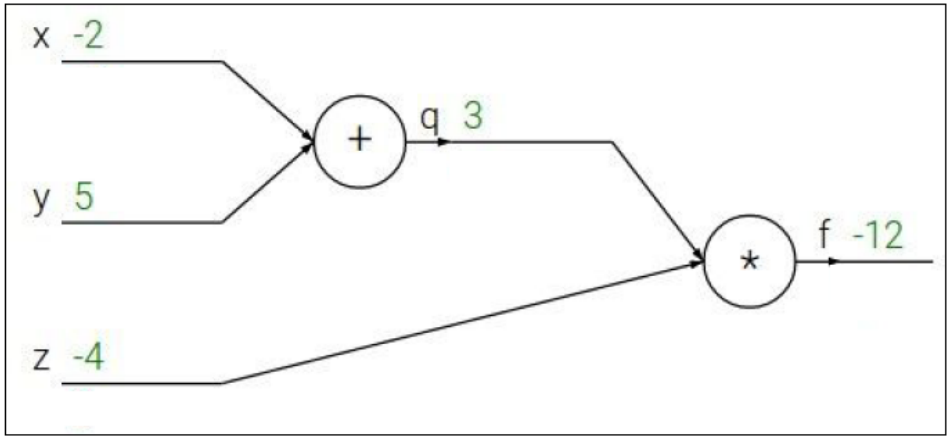
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

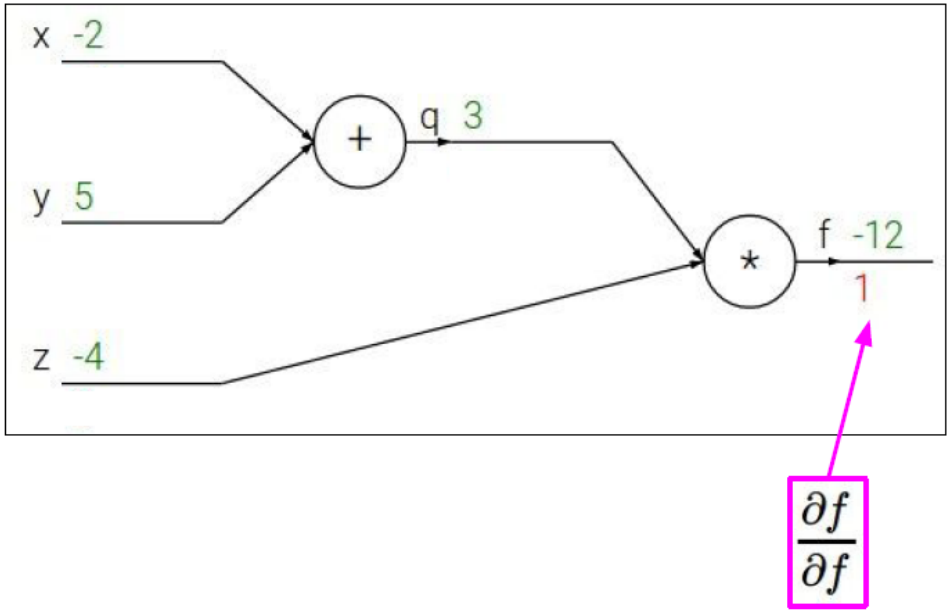
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

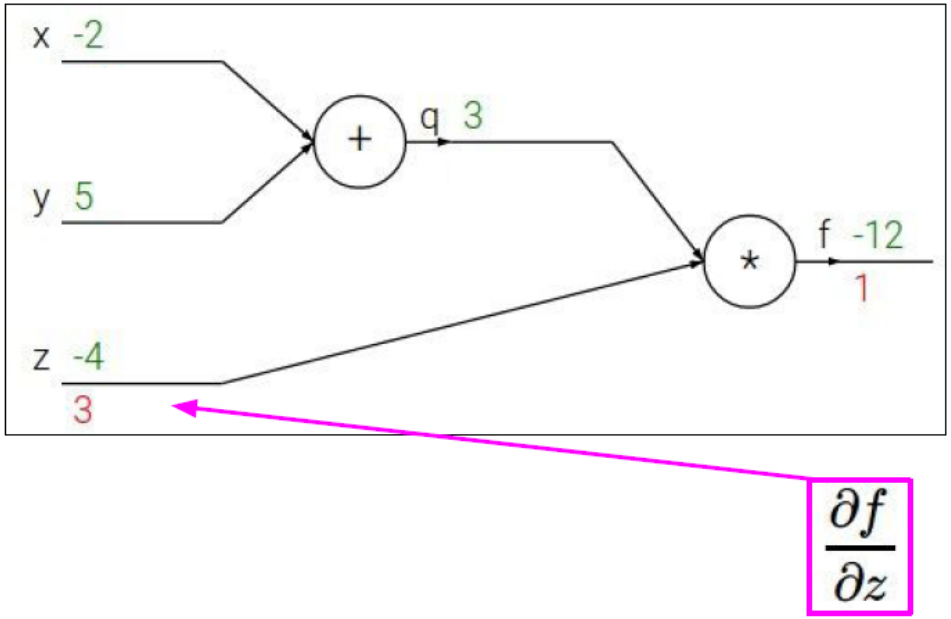
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

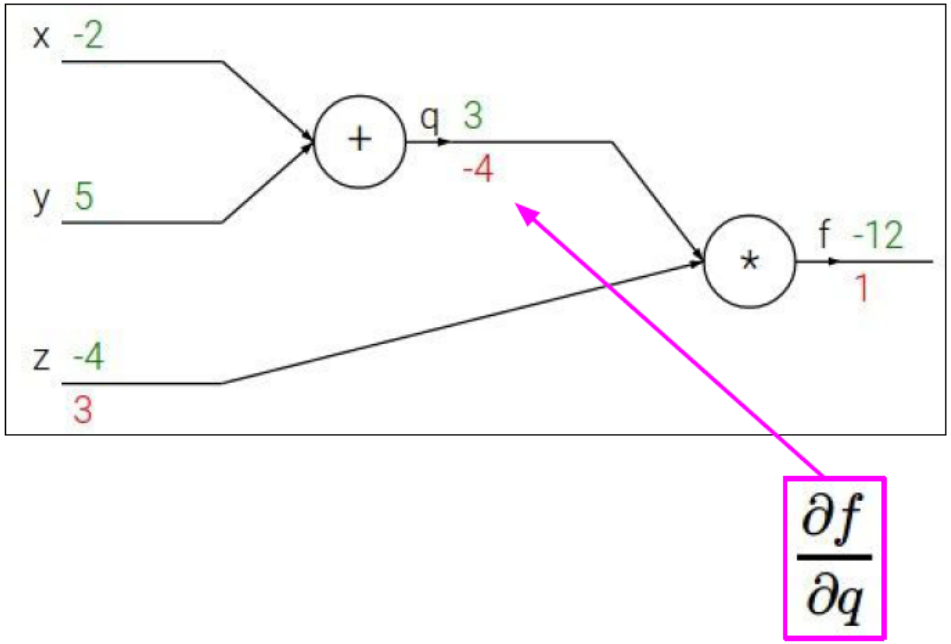
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: a simple example

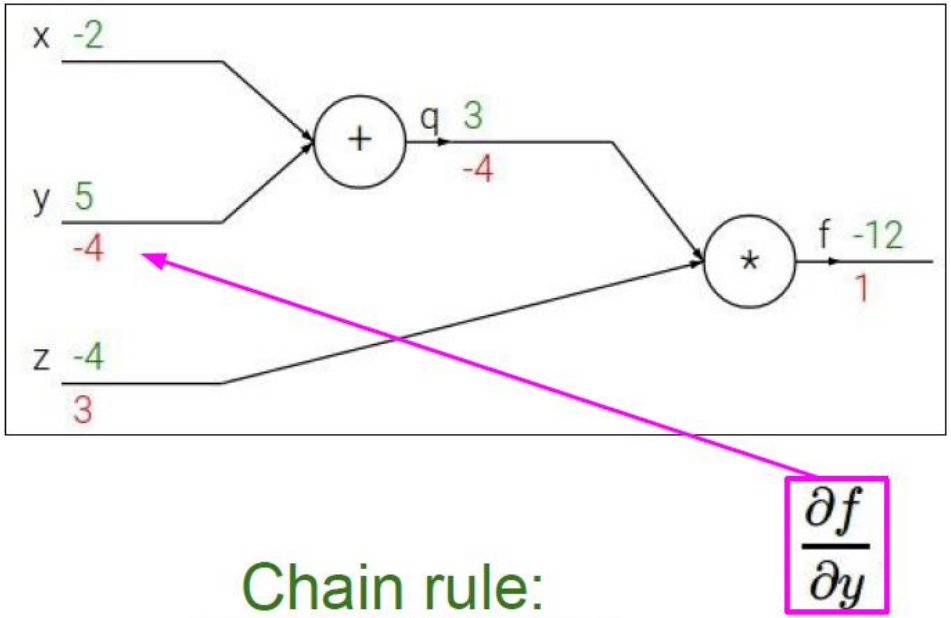
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Backpropagation: a simple example

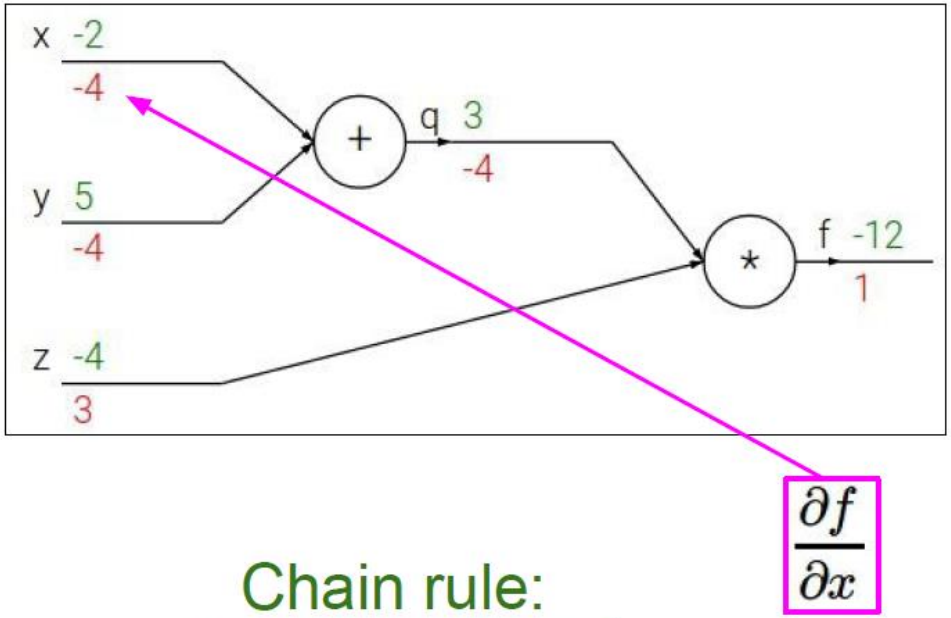
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

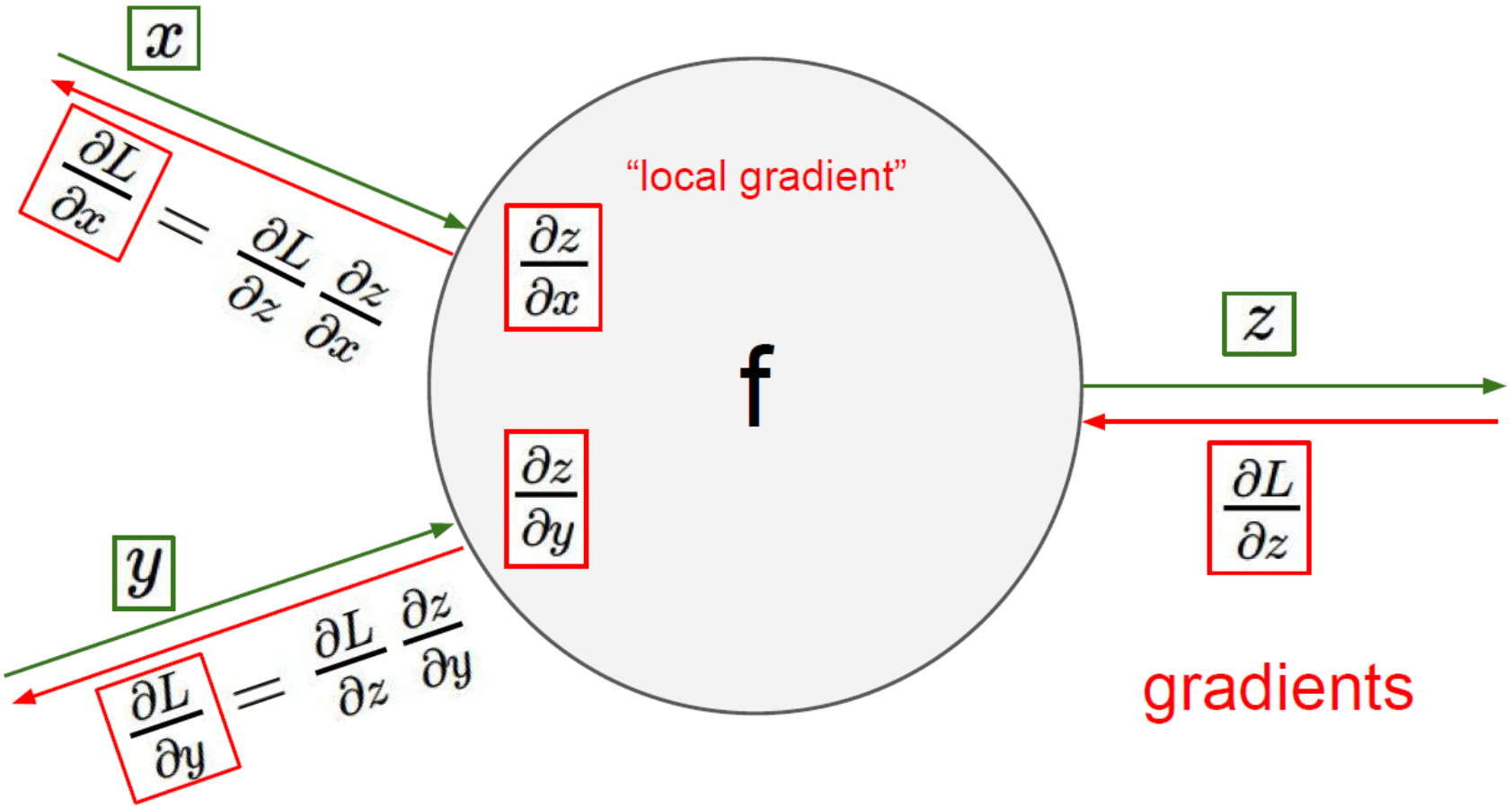
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

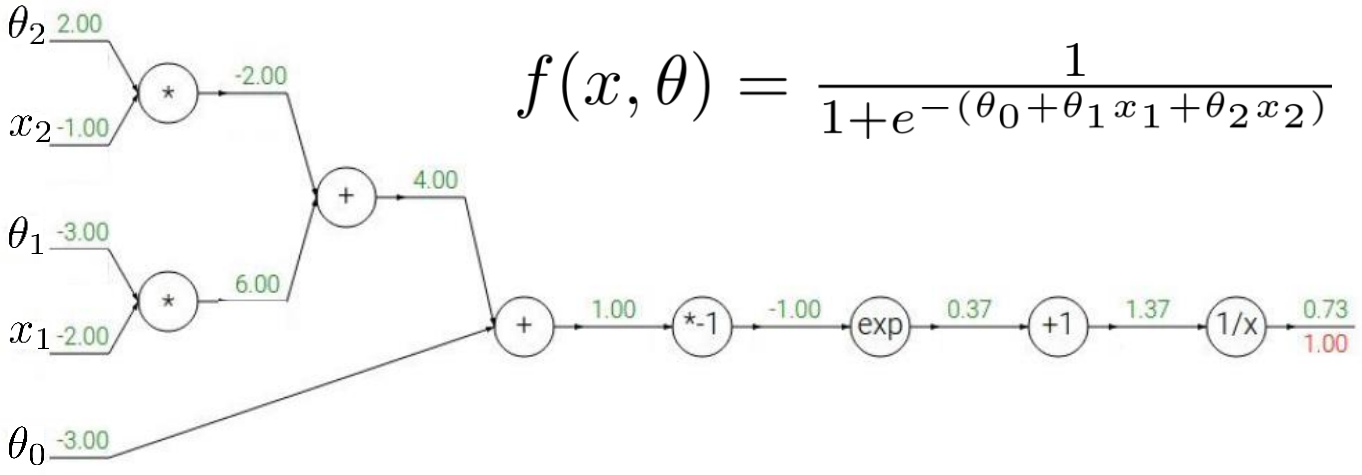


Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Another example:

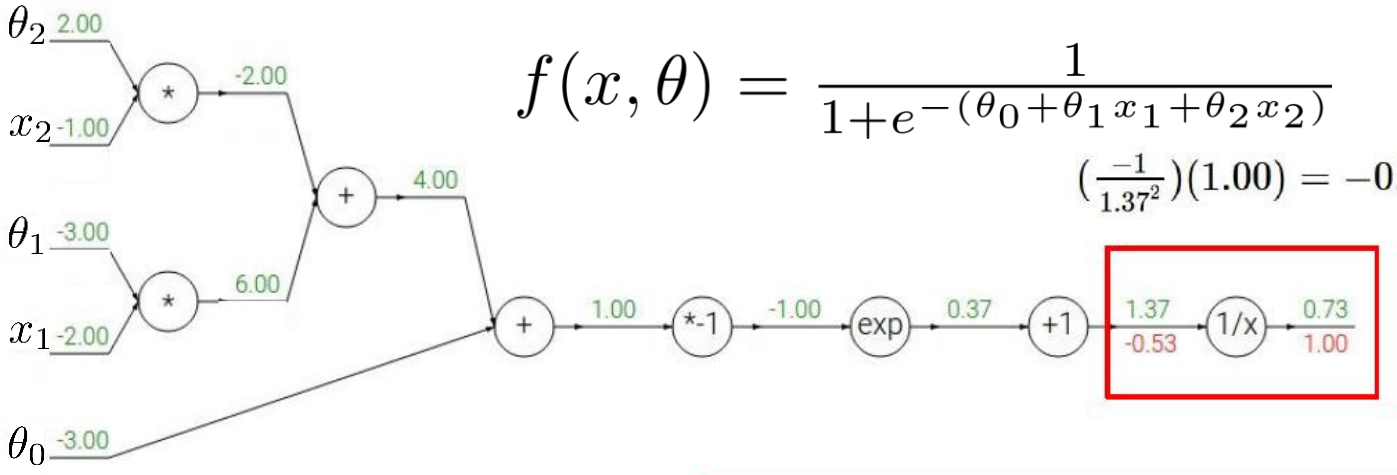


$$f(x,\theta) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \qquad\qquad f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \qquad\qquad f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:



$$f(x,\theta) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2)}}$$

$$(\tfrac{-1}{1.37^2})(1.00) = -0.53$$

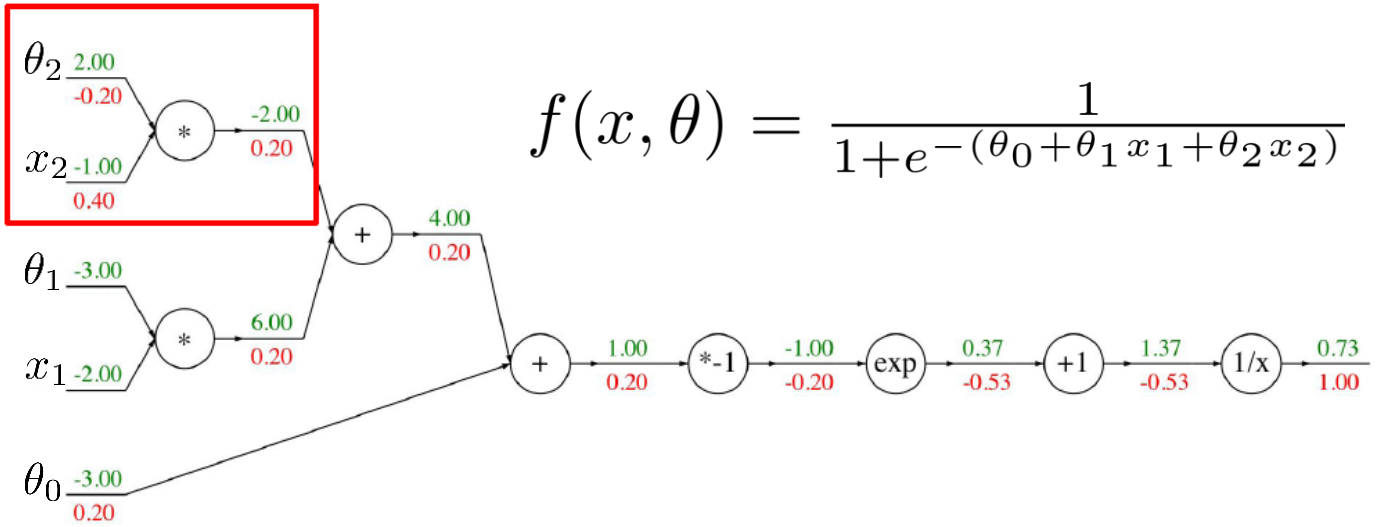
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:



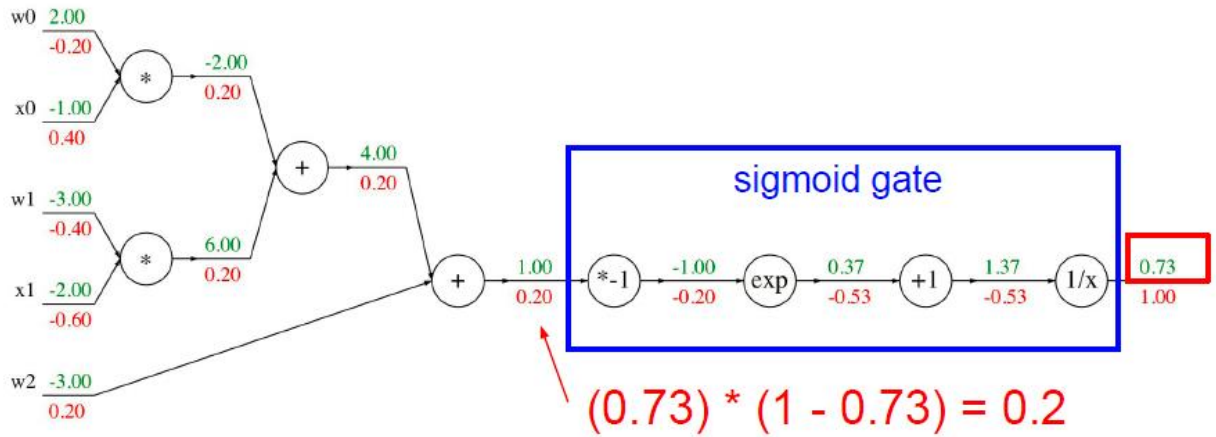
$$f(x,\theta) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2)}}$$

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x \qquad f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a \qquad f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

$$f(x,\theta) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2)}}$$

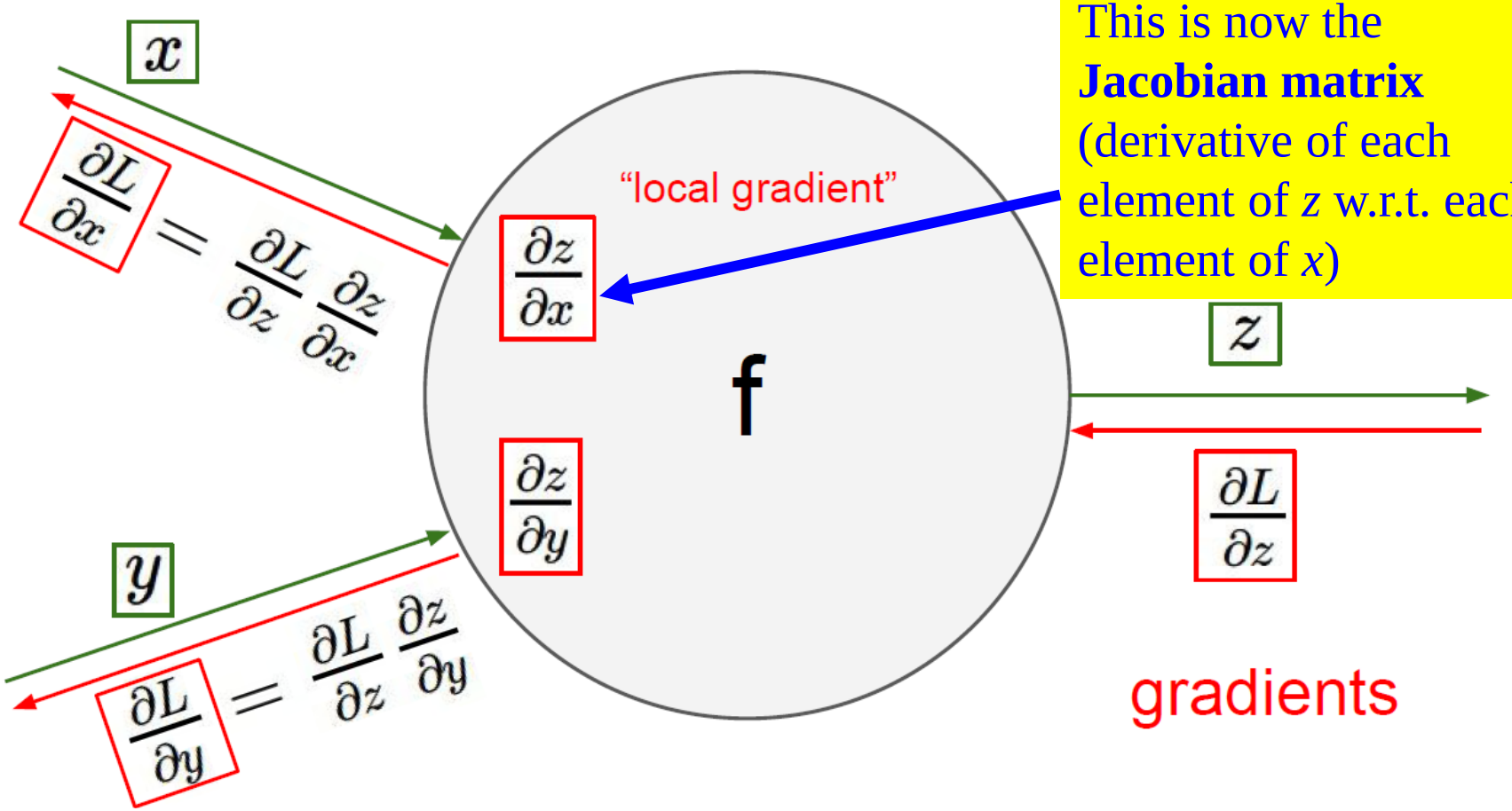
$$g(x) = \frac{1}{1+e^{-x}} \qquad g'(x) = g(x)(1-g(x))$$



梯度矢量化表示

(*x*,*y*,*z* are now vectors)

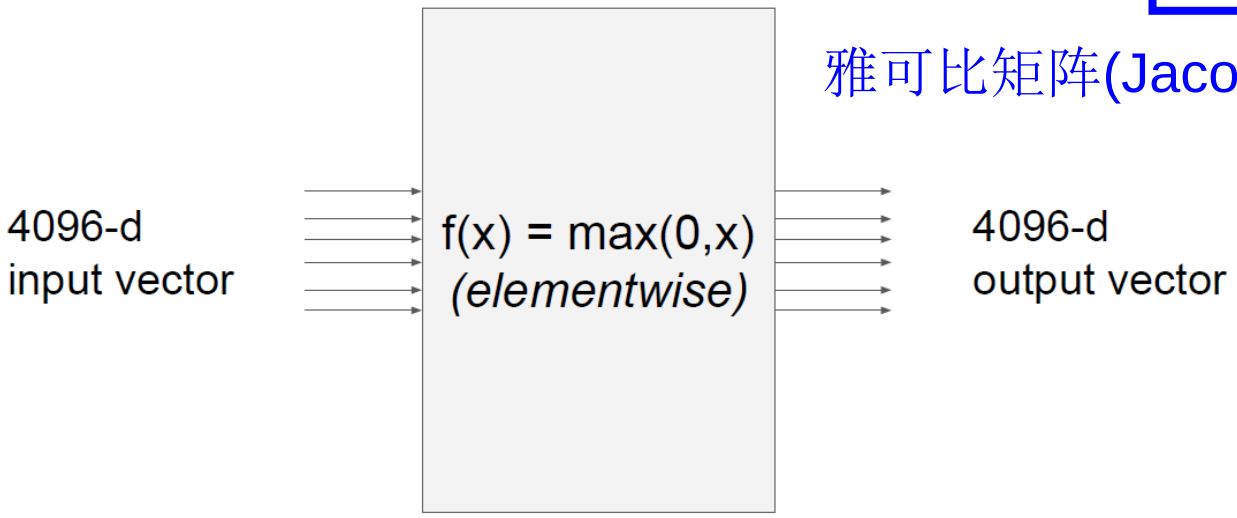
This is now the **Jacobian matrix** (derivative of each element of *z* w.r.t. each element of *x*)



梯度矢量化表示

$$\frac{\partial J}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial J}{\partial f}$$

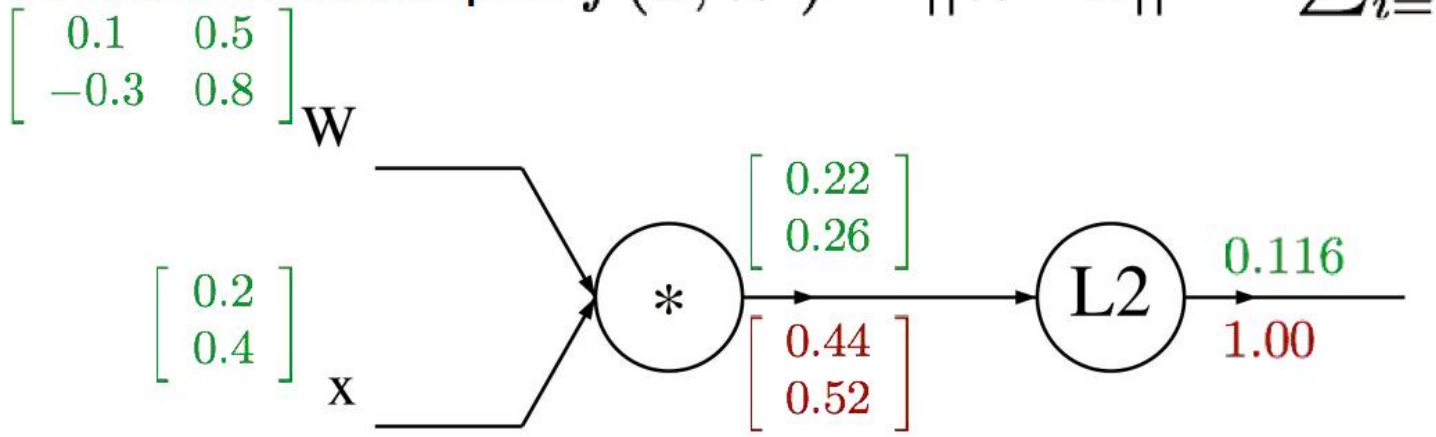
雅可比矩阵(Jacobian matrix)



思考：

1. 雅可比矩阵的大小是？
2. 对于上图Relu激活函数，对应的雅可比矩阵具有什么特点？

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



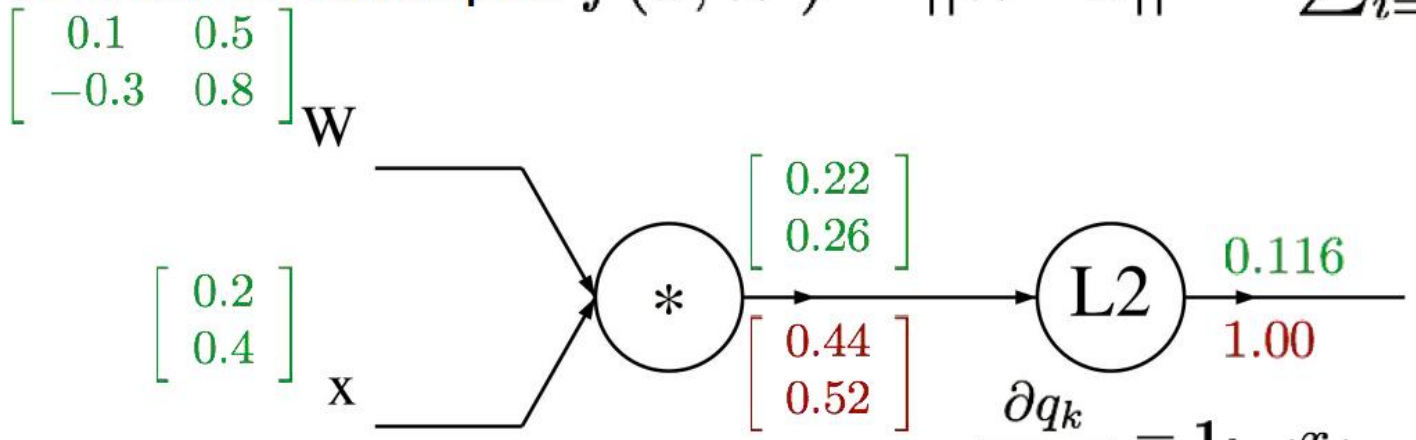
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



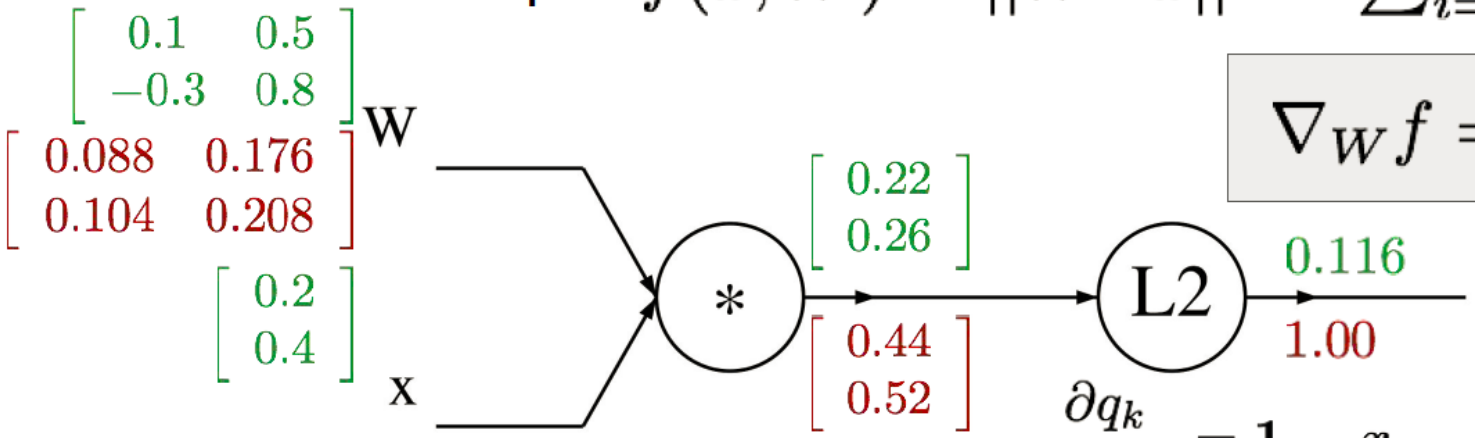
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$\nabla_W f = 2q \cdot x^T$$

Always check: The gradient with respect to a variable should have the same shape as the variable

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

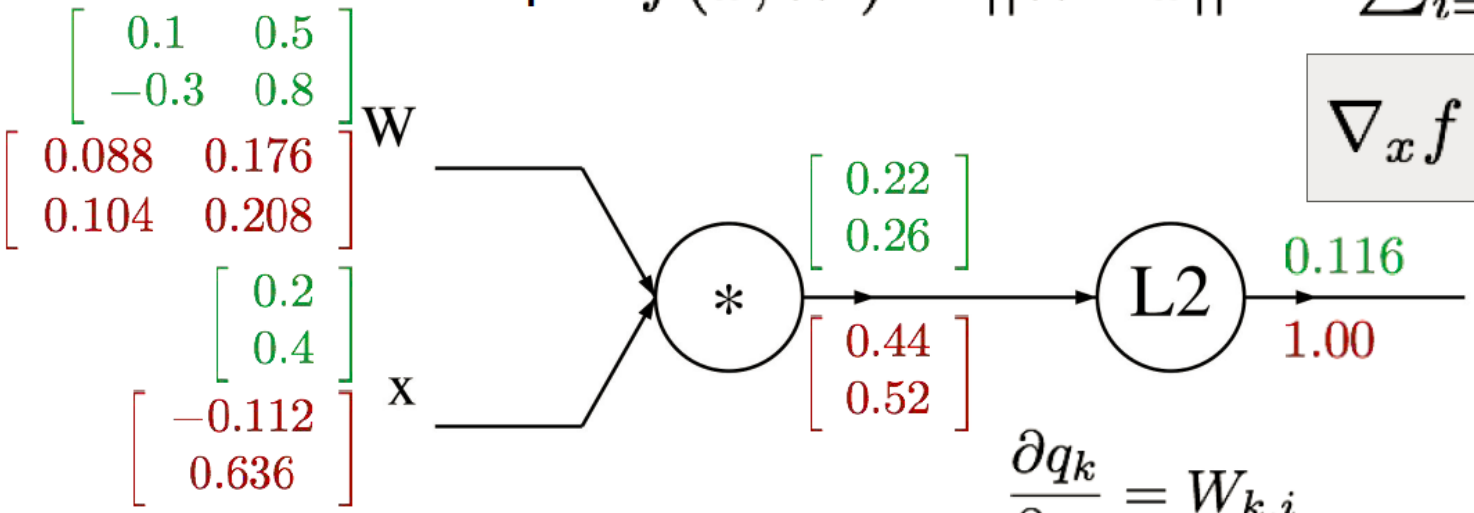
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\begin{aligned} \frac{\partial f}{\partial W_{i,j}} &= \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} \\ &= \sum (2q_k)(\mathbf{1}_{k=i} x_j) \\ &= 2q_i x_j \end{aligned}$$

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\nabla_x f = 2W^T \cdot q$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i}$$

$$= \sum_k 2 q_k W_{k,i}$$

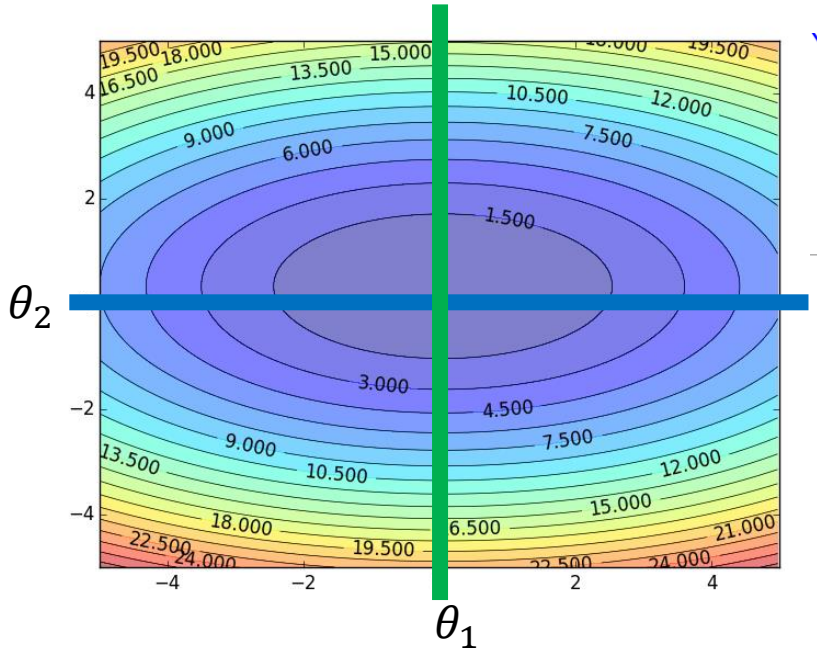
Adaptive 学习率

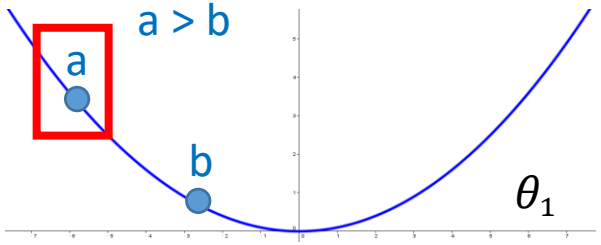
- Idea 1：Reduce the learning rate by some factor every few epochs
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g. 1/t decay: $\alpha^t = \alpha/\sqrt{t+1}$
- Idea 2：Giving different parameters different learning rates
  - E.g. AdaGrad

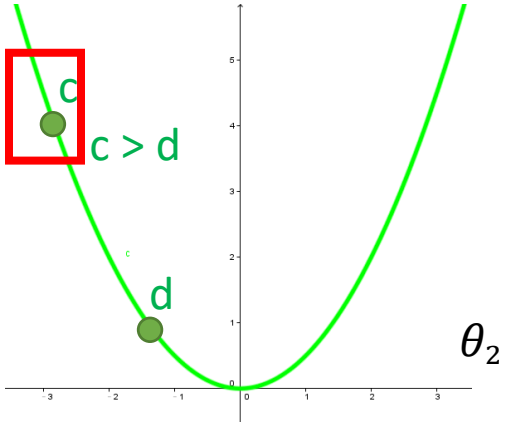
AdaGrad: Motivation

~~Larger 1st order derivative means far from the minima~~

Do not cross parameters







AdaGrad

$$\alpha^t = \frac{\alpha}{\sqrt{t+1}} \qquad g^t = \frac{\partial L(\theta^t)}{\partial \theta}$$

- Divide the learning rate of each parameter by the ***root mean square of its previous derivatives***

Vanilla Gradient descent

$$\theta^{t+1} \leftarrow \theta^t - \alpha^t g^t$$

θ is one parameters

Adagrad

$$\theta^{t+1} \leftarrow \theta^t - \frac{\alpha^t}{\sigma^t} g^t$$

σ^t: ***root mean square*** of the previous derivatives of parameter θ

Parameter dependent

AdaGrad

σ^t: ***root mean square*** of the previous derivatives of parameter θ

$$\theta^1 \leftarrow \theta^0 - \frac{\alpha^0}{\sigma^0} g^0 \qquad \sigma^0 = \sqrt{(g^0)^2}$$

$$\theta^2 \leftarrow \theta^1 - \frac{\alpha^1}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\frac{1}{2}[(g^0)^2 + (g^1)^2]}$$

$$\theta^3 \leftarrow \theta^2 - \frac{\alpha^2}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\frac{1}{3}[(g^0)^2 + (g^1)^2 + (g^2)^2]}$$

$$\vdots$$

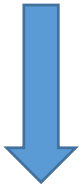
$$\theta^{t+1} \leftarrow \theta^t - \frac{\alpha^t}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$

AdaGrad

$$\alpha^t = \frac{\alpha}{\sqrt{t+1}}$$

$$\theta^{t+1} \leftarrow \theta^t - \frac{\alpha^t}{\sigma^t} g^t$$

$$\sigma^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g^i)^2}$$



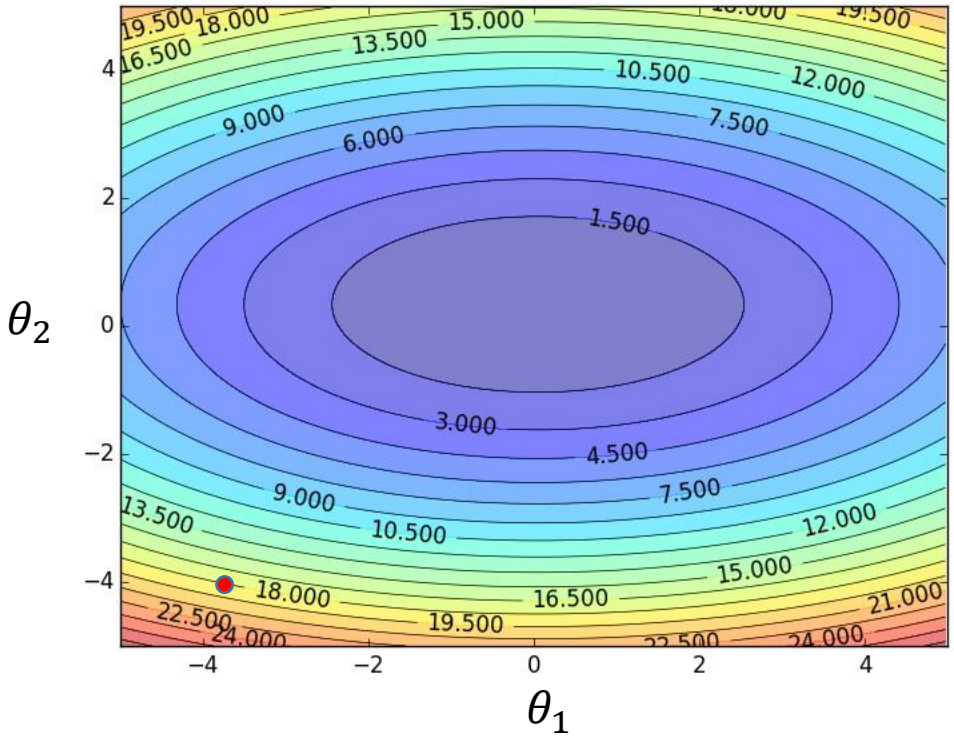
$$\theta^{t+1} \leftarrow \theta^t - \frac{\alpha}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

思考：What happens to the step size over long time?

AdaGrad

AdaGrad:

$$u^0 = 0$$
$$u^{t+1} = u^t + \nabla J(\theta^t)^T \nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - \frac{\alpha}{\sqrt{u^{t+1}}} \nabla J(\theta^t)$$



思考：What happens with AdaGrad?

RMSProp

RMSProp:

$$u^0 = 0$$
$$u^{t+1} = \rho u^t + (1-\rho)\nabla J(\theta^t)^T \nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - \frac{\alpha}{\sqrt{u^{t+1}}} \nabla J(\theta^t)$$

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

思考： What happens to the step size over long time?

Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

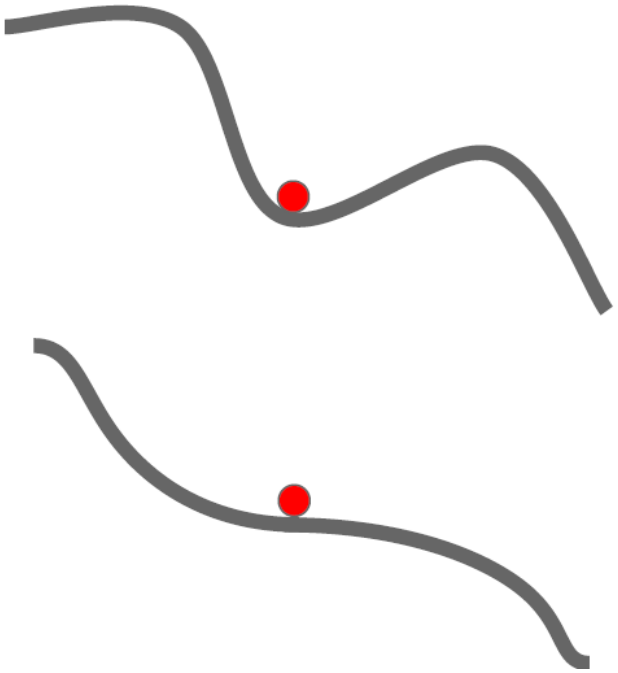
改进的梯度下降法：Momentum

Gradient descent (GD):

$$\theta^{t+1} = \theta^t - \alpha \nabla J(\theta^t)$$

Momentum:

$$v^0 = 0$$
$$v^{t+1} = \rho v^t + \alpha \nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - v^{t+1}$$

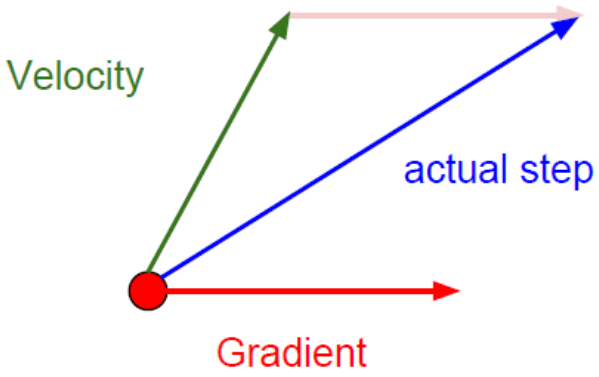


基本思想：引入速度（前面梯度的加权均值）

Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

改进的梯度下降法：Nesterov Momentum (Nesterov accelerated gradient, NAG)

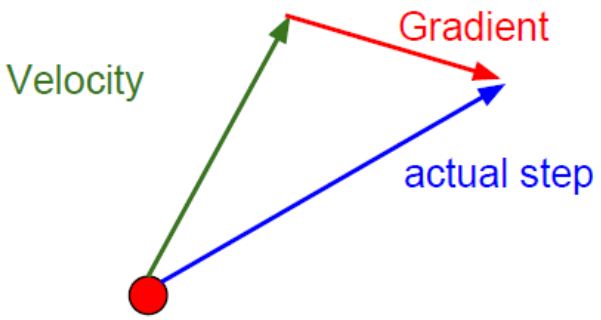
Momentum update:



GD+Momentum:

$$v^{t+1} = \rho v^t + \alpha \nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - v^{t+1}$$

Nesterov Momentum



Nesterov Momentum:

$$v^{t+1} = \rho v^t + \alpha \nabla J(\theta^t + \rho v^t)$$
$$\theta^{t+1} = \theta^t - v^{t+1}$$

Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

改进的梯度下降法：Adam

Adam (biased): Sort of like RMSProp with momentum

$$u^0 = 0$$
$$v^0 = 0$$
$$u^{t+1} = \rho_1 u^t + (1-\rho_1)\nabla J(\theta^t)^T \nabla J(\theta^t)$$
$$v^{t+1} = \rho_2 v^t + (1-\rho_2)\nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - \frac{\alpha}{\sqrt{u^{t+1}}} v^{t+1}$$

思考：What happens at first timestep?

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

改进的梯度下降法：Adam

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

改进的梯度下降法：Adam

GD+Momentum:

$$v^{t+1} = \rho v^t + \alpha \nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - v^{t+1}$$

Nesterov Momentum:

$$v^{t+1} = \rho v^t + \alpha \nabla J(\theta^t + \rho v^t)$$
$$\theta^{t+1} = \theta^t - v^{t+1}$$

Adam (biased): Sort of like RMSProp with momentum

$$u^0 = 0$$
$$v^0 = 0$$
$$u^{t+1} = \rho_1 u^t + (1-\rho_1)\nabla J(\theta^t)^T \nabla J(\theta^t)$$
$$v^{t+1} = \rho_2 v^t + (1-\rho_2)\nabla J(\theta^t)$$
$$\theta^{t+1} = \theta^t - \frac{\alpha}{\sqrt{u^{t+1}}} v^{t+1}$$

思考：Adam可以看成是RMSProp + Momentum, 可否换成Nesterov Momentum?

YES. Nadam

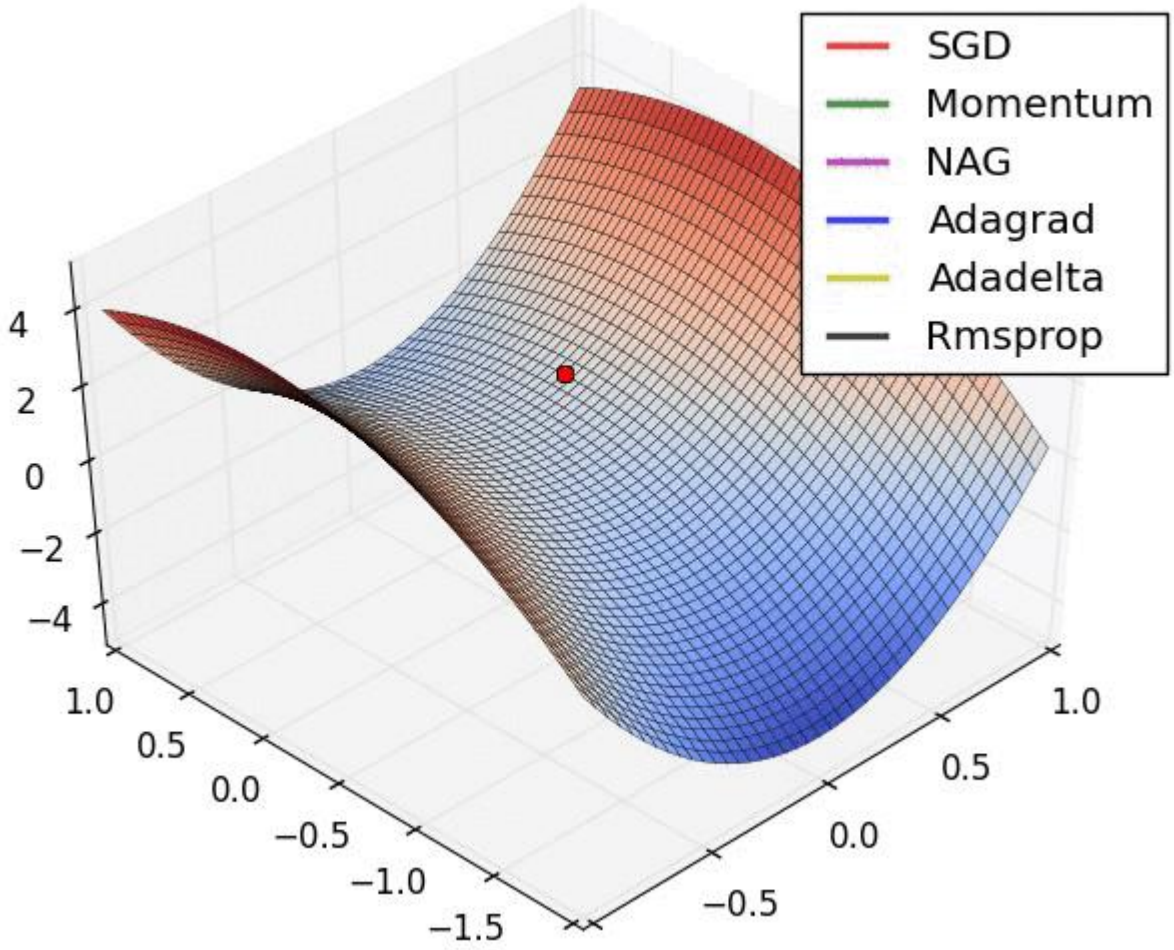
Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

Adam

- It has been typically observed that in these settings some minibatches provide large gradients but only quite rarely, and while these large gradients are quite informative, their influence dies out rather quickly due to the exponential averaging, thus leading to poor convergence.
- In order to have guaranteed convergence the optimization algorithm must have “long-term memory” of past gradients.

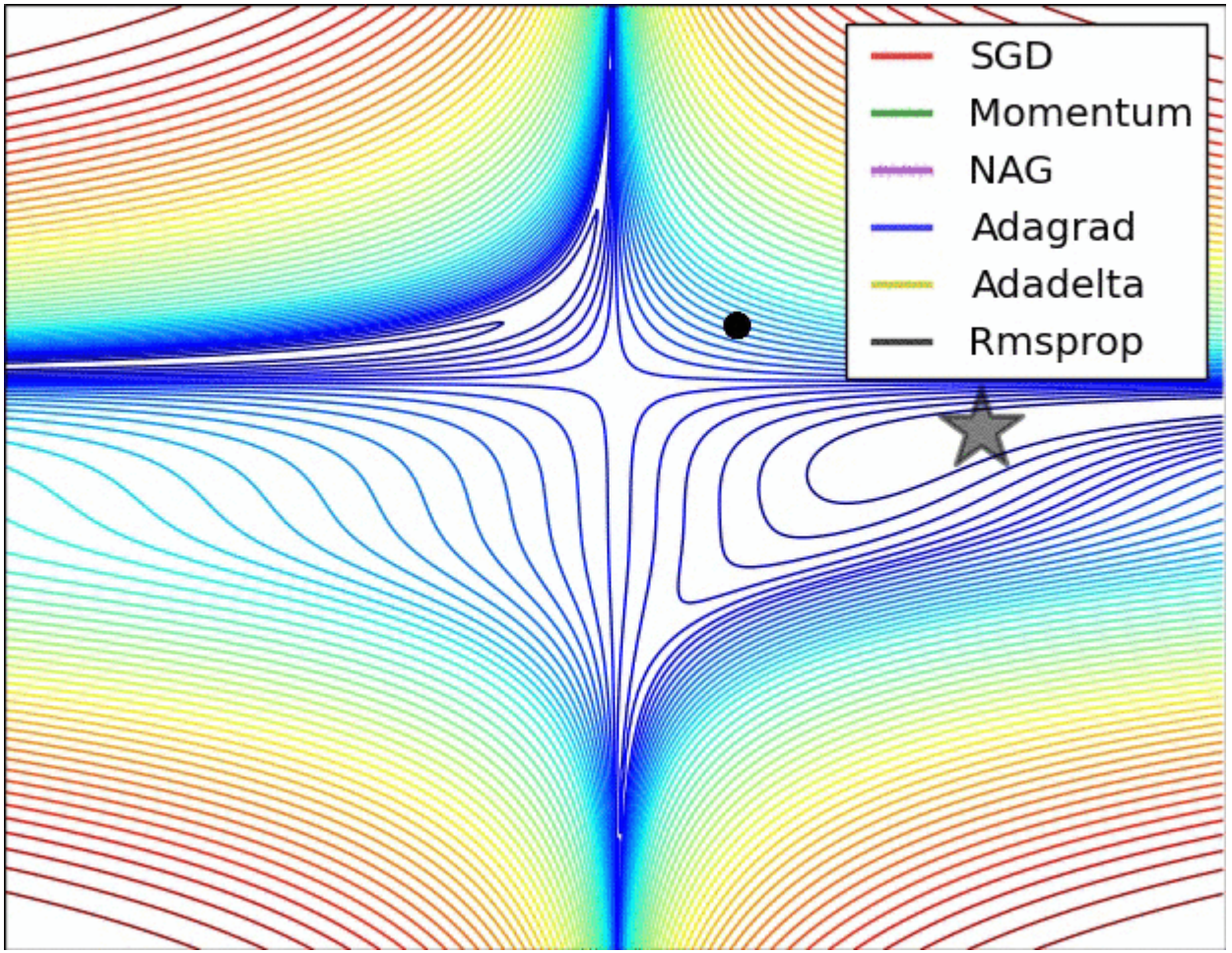
Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. "On the convergence of adam and beyond." *ICLR* (2019) best paper.

各种梯度下降法可视化



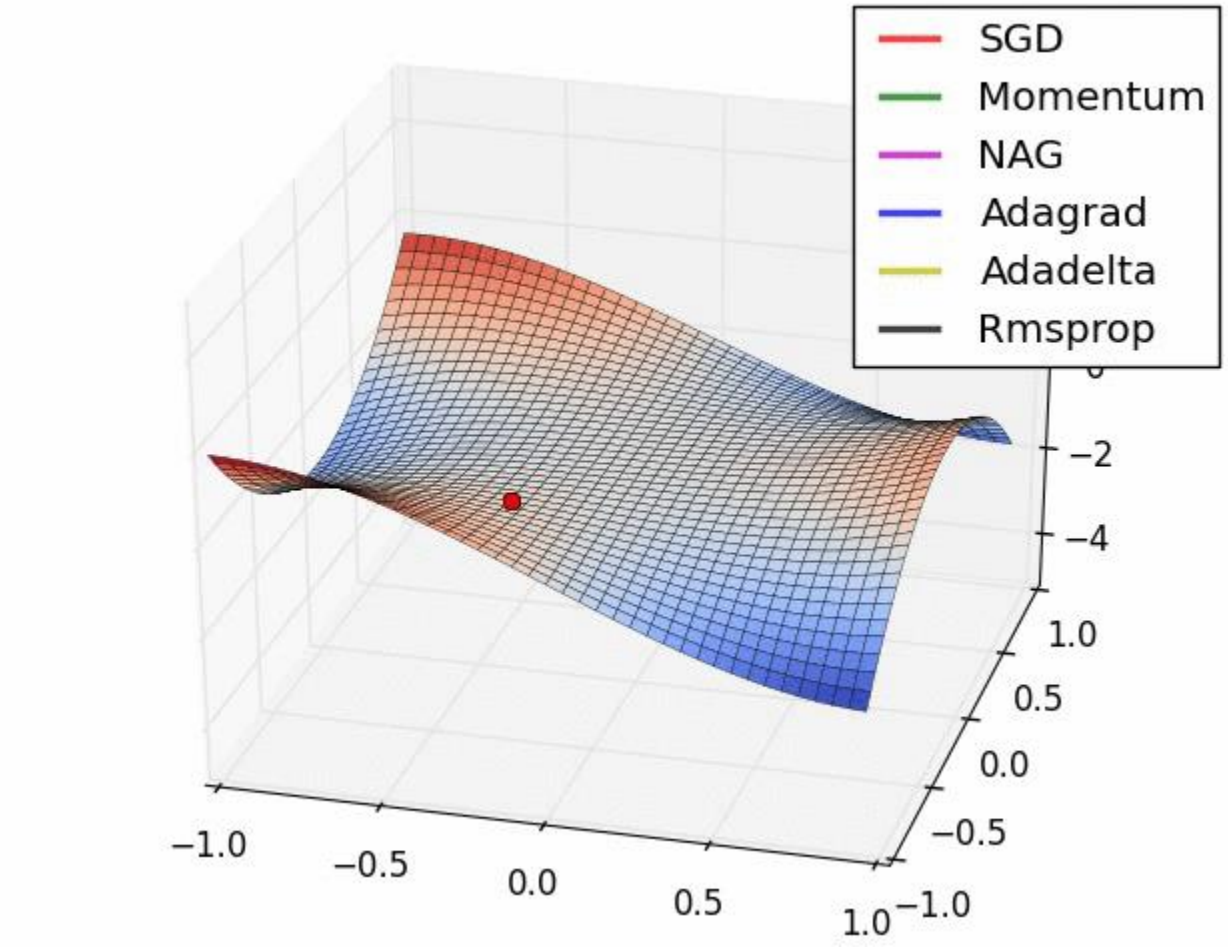
http://imgur.com/a/Hqolp

各种梯度下降法可视化



http://imgur.com/a/Hqolp

各种梯度下降法可视化



http://imgur.com/a/Hqolp

Which optimizer to use?

- Insofar, Adam might be the best overall choice.
- SGD
  - usually achieves to find a minimum, but it might take significantly longer
  - is much more reliant on a robust initialization and annealing schedule
  - may get stuck in saddle points rather than local minima

Additional strategies for optimizing SGD

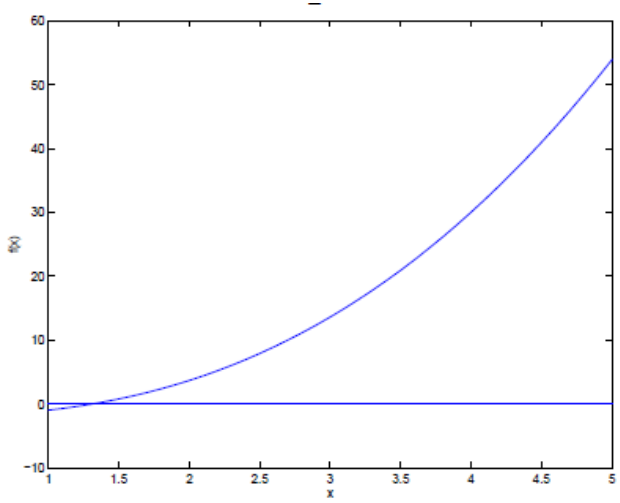
- Shuffling and Curriculum Learning
  - It is often a good idea to shuffle the training data after every epoch
  - On the other hand, for some cases where we aim to solve progressively harder problems, supplying the training examples in a meaningful order may actually lead to improved performance and better convergence
- Early stopping: Early stopping (is) beautiful free lunch (Geoff Hinton)
- Gradient noise: add noise o each gradient update

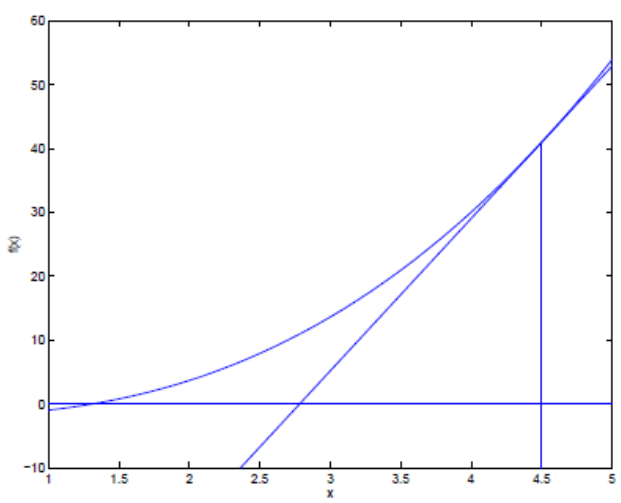
Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:1609.04747*(2016).

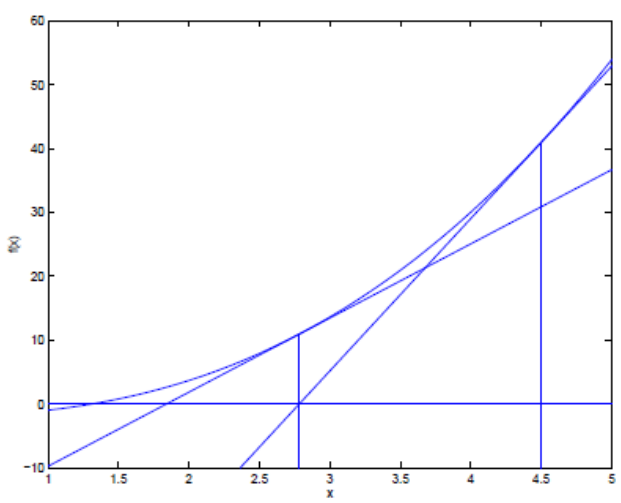
二阶方法：牛顿法

假设采用牛顿法求解方程 $f(\theta)=0$ 的解，迭代规则如下：

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$







牛顿法

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

$$\min_\theta J(\theta) \iff J'(\theta) = 0 \qquad f(\theta) = J'(\theta)$$

$$\theta := \theta - \frac{J'(\theta)}{J''(\theta)}$$

若θ是矢量，则 $\theta := \theta - H^{-1}\nabla_\theta J(\theta).$

Here, $\nabla_\theta \ell(\theta)$ is, as usual, the vector of partial derivatives of $\ell(\theta)$ with respect to the θ_i's; and H is an n-by-n matrix (actually, $n+1$-by-$n+1$, assuming that we include the intercept term) called the **Hessian**, whose entries are given by

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}.$$

与梯度下降法相比：

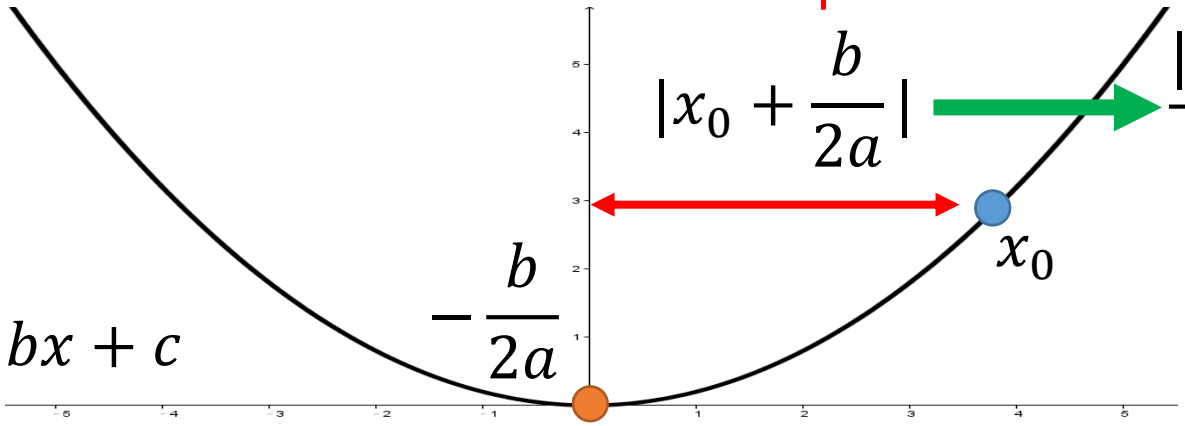
1. Faster convergence
2. More expensive
3. No learning rate!
4. …

牛顿法

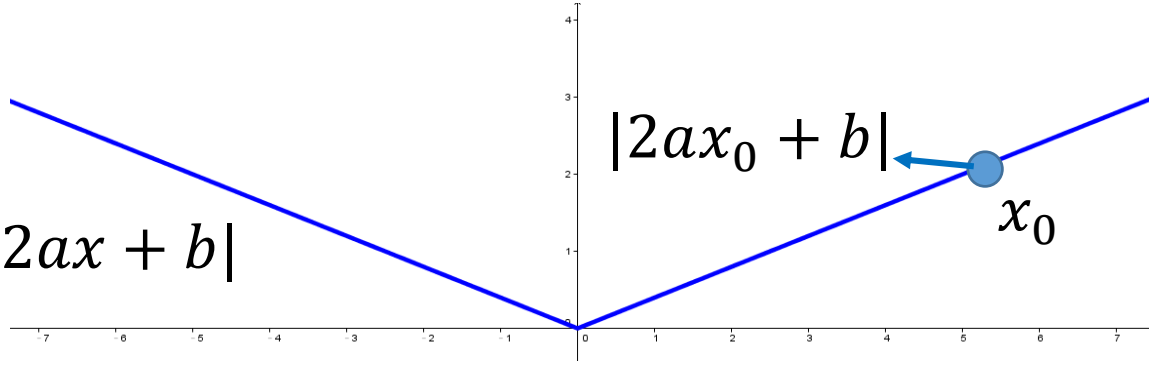
$$\theta := \theta - \frac{J'(\theta)}{J''(\theta)} \qquad \theta := \theta - H^{-1}\nabla_\theta J(\theta).$$

Best step:

$$|x_0 + \frac{b}{2a}| \qquad \frac{|2ax_0 + b|}{2a}$$



$$y = ax^2 + bx + c$$



$$\left|\frac{\partial y}{\partial x}\right| = |2ax + b|$$

$$\frac{\partial^2 y}{\partial x^2} = 2a$$

The best step is $\dfrac{|\text{First derivative}|}{\text{Second derivative}}$

一阶方法

由于二阶方法需要计算Hessian矩阵，在参数数目较多的情况下计算复杂，目前主要仍采用一阶方法，如

- 共轭梯度
- Quasi-Newton methods(BFGS)
  - *Approximate inverse Hessian with rank 1 updates*
- L-BFGS (Limited memory BFGS)
  - *Does not form/store the full inverse Hessian*

优点:

- 不用人为选择学习率
- 比梯度下降算法要快.

缺点:

- 更复杂

Gradient checking

Numerical estimation of gradients: $\frac{\partial J(\theta)}{\partial \theta} \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$

$\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$)

$\theta = \theta_1, \theta_2, \theta_3, \ldots, \theta_n$

$$\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1+\epsilon,\theta_2,\theta_3,\ldots,\theta_n)-J(\theta_1-\epsilon,\theta_2,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1,\theta_2+\epsilon,\theta_3,\ldots,\theta_n)-J(\theta_1,\theta_2-\epsilon,\theta_3,\ldots,\theta_n)}{2\epsilon}$$

$\vdots$

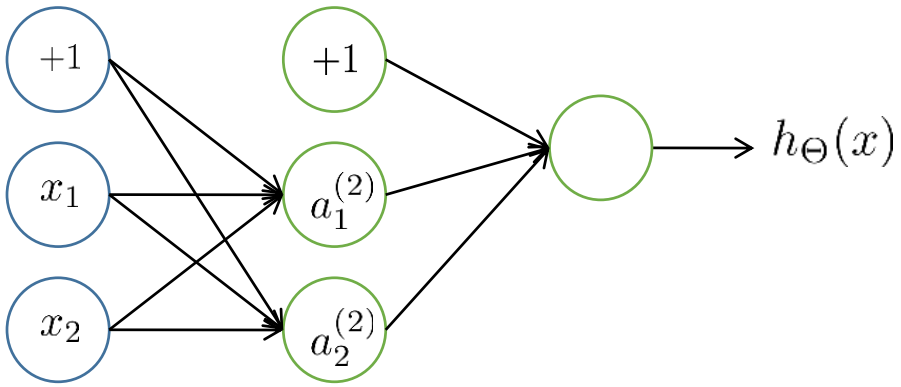
$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n+\epsilon)-J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n-\epsilon)}{2\epsilon}$$

初始化

Θ

Zero initialization?

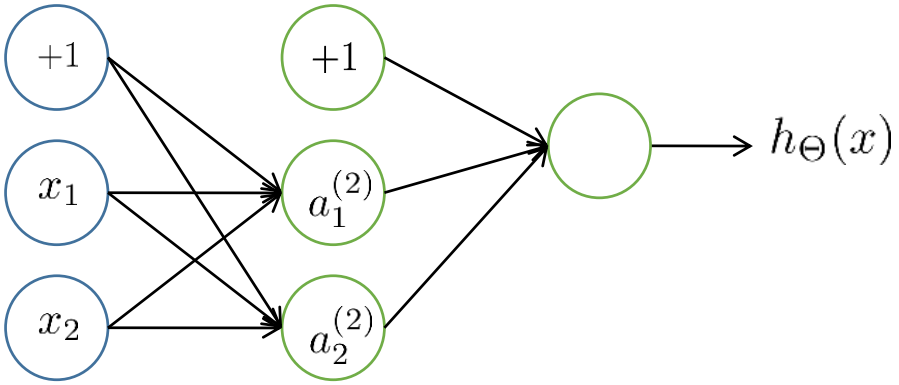
$$\Theta_{ij}^{(l)} = 0 \text{ for all } i, j, l.$$



初始化

Θ

Random initialization (Gaussian with zero mean and 1e-2 standard deviation)?



Works ~okay for small networks, but problems with deeper networks

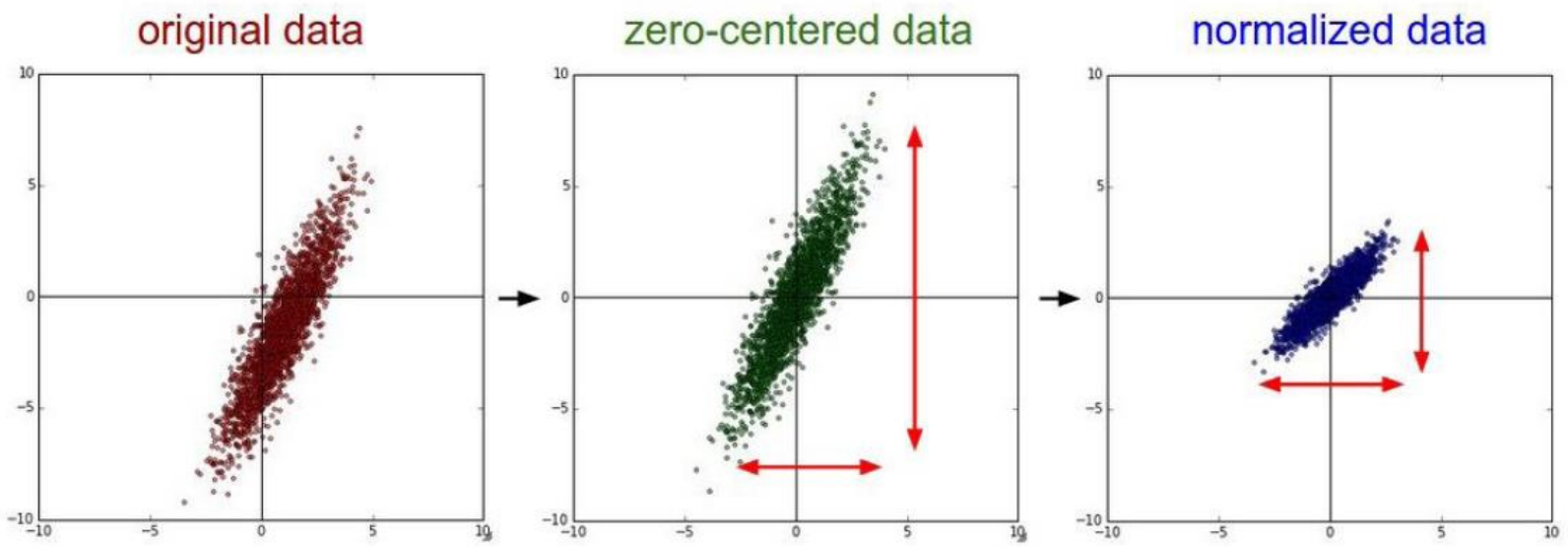
Xavier initialization [Glorot et al., 2010]
He initialization [He et al., 2015]

初始化 Θ

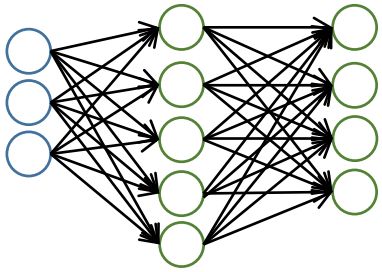
- ***Understanding the difficulty of training deep feedforward neural networks*** by Glorot and Bengio, 2010 (**Xavier initialization**)
- Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013
- Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014
- Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015 (**He initialization**)
- Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015
- All you need is a good init, Mishkin and Matas, 2015
- …

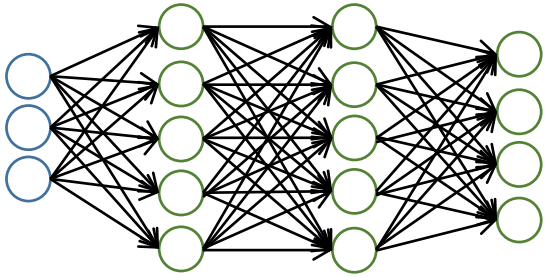
Putting it together

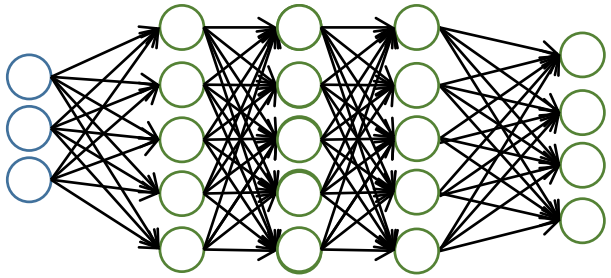
数据预处理



网络结构选择与设计







训练网络

- Initialize weights
- Implement forward propagation
- Implement code to compute cost function
- Implement backpropagation code to compute gradient
- Implement numerical gradient check (disable your gradient checking code before training)
- Double check that the loss is reasonable
- Make sure that you can overfit very small portion of the training data
- Start with small regularization and find learning rate that makes the loss go down

Thanks!

Any questions?