



太原理工大学  
TAIYUAN UNIVERSITY OF TECHNOLOGY



太原理工大学  
大数据学院  
COLLEGE OF DATA SCIENCE  
TAIYUAN UNIVERSITY OF TECHNOLOGY

## 第六章 深度前馈网络

# 深度前馈网络

## 主要内容

**01** 深度前馈网络

**02** 万能近似性质

**03** TensorFlow

# PART 深度前馈网络 ONE

## 深度前馈网络

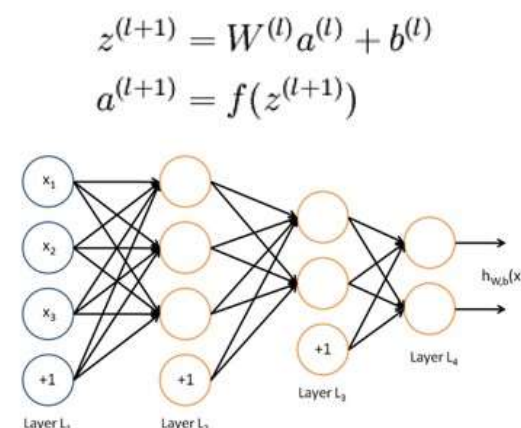
### 前向传播与反向回馈

Square Euclidean Distance (regression)

$$J = \frac{1}{2} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2$$

$$= \left[ \frac{1}{m} \sum_{i=1}^m \left( \frac{1}{2} \|h_{W,b}(x^{(i)}) - y^{(i)}\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ji}^{(l)})^2$$



更新迭代:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) = \left[ \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b) = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(i)}, y^{(i)})$$

单选题 1分

假设输入层中的节点数为8，隐藏层神经元数量为15，且隐藏层每个神经元的偏置不同，请问输入层到隐藏层的参数量是（ ）

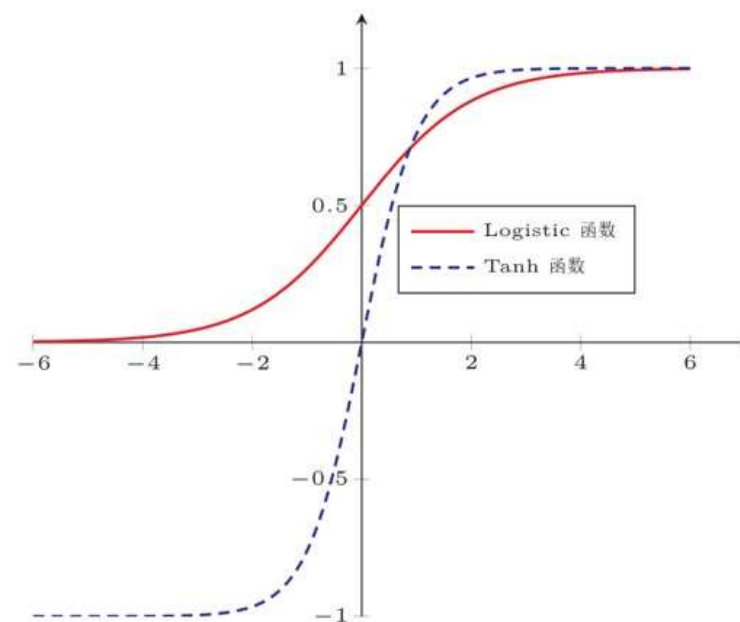
- ☐ A 120
- ☒ B 128
- ☐ C 135
- ☐ D 144

## 常见激活函数

### 激活函数

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$



## 常见激活函数

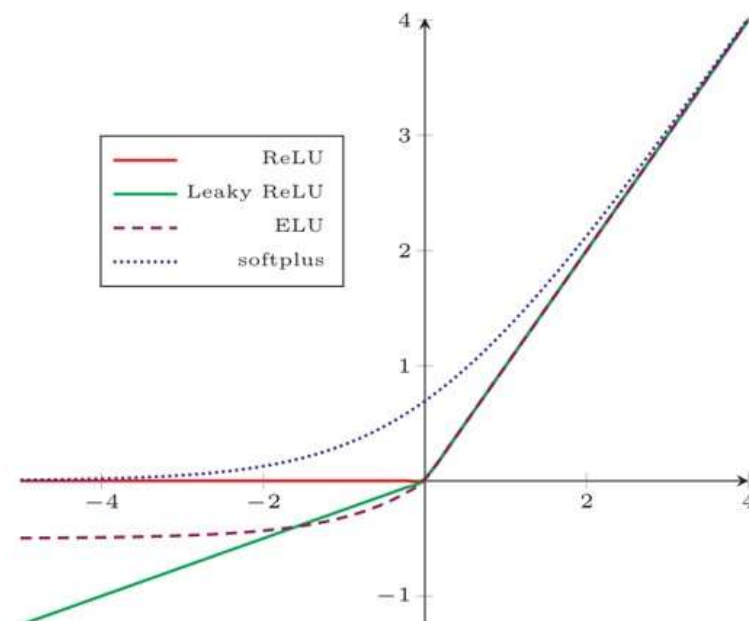
### 激活函数

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$= \max(0, x).$$

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma \min(0, x)$$

$$\text{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma_i \min(0, x)$$

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

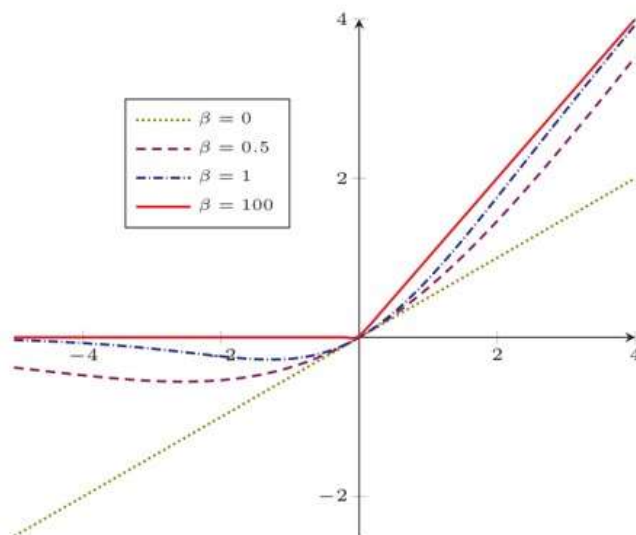


$$\text{softplus}(x) = \log(1 + \exp(x))$$

## 常见激活函数

### 激活函数

Swish函数  $\text{swish}(x) = x\sigma(\beta x)$





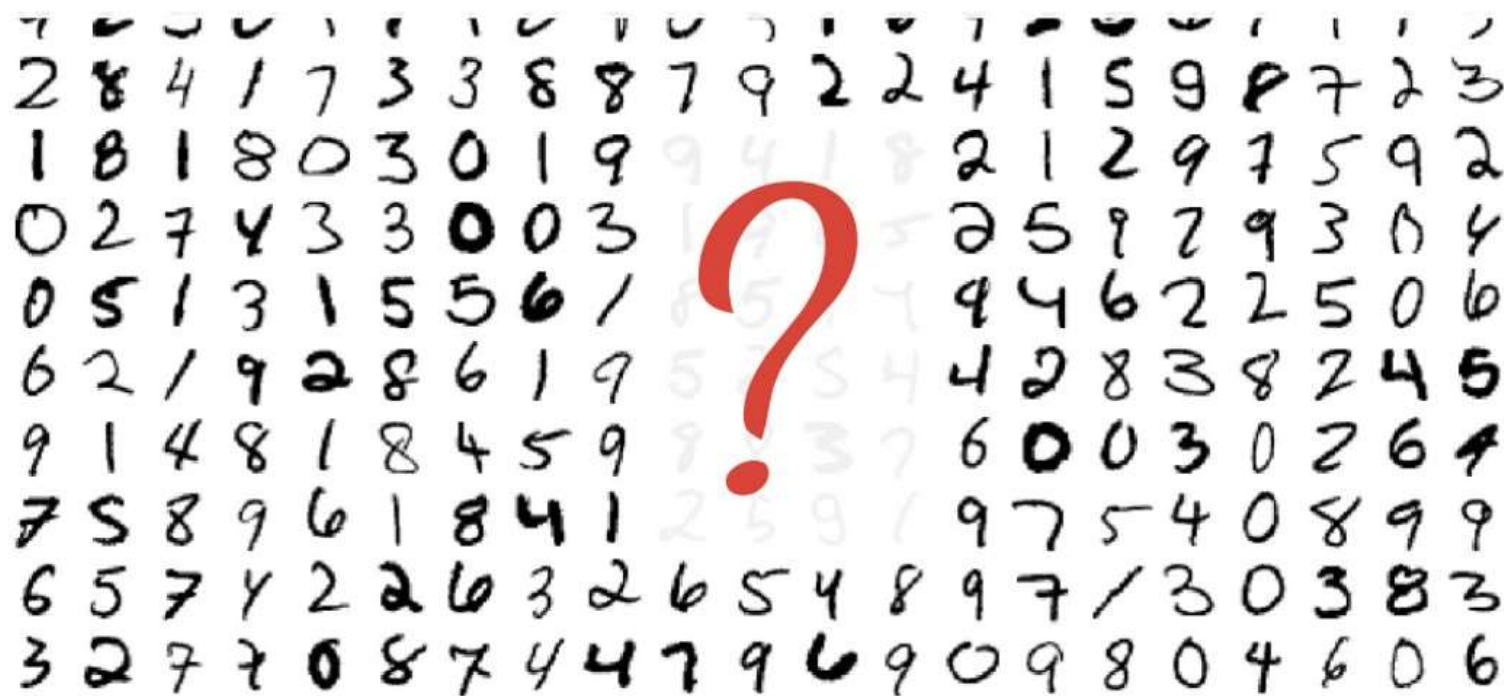
## 常见激活函数

### 激活函数

激活函数	函数	导数
Logistic 函数	$f(x) = \frac{1}{1+\exp(-x)}$	$f'(x) = f(x)(1 - f(x))$
Tanh 函数	$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU	$f(x) = \max(0, x) + \min(0, \gamma(\exp(x) - 1))$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \log(1 + \exp(x))$	$f'(x) = \frac{1}{1+\exp(-x)}$

## 深度前馈网络

### 单层



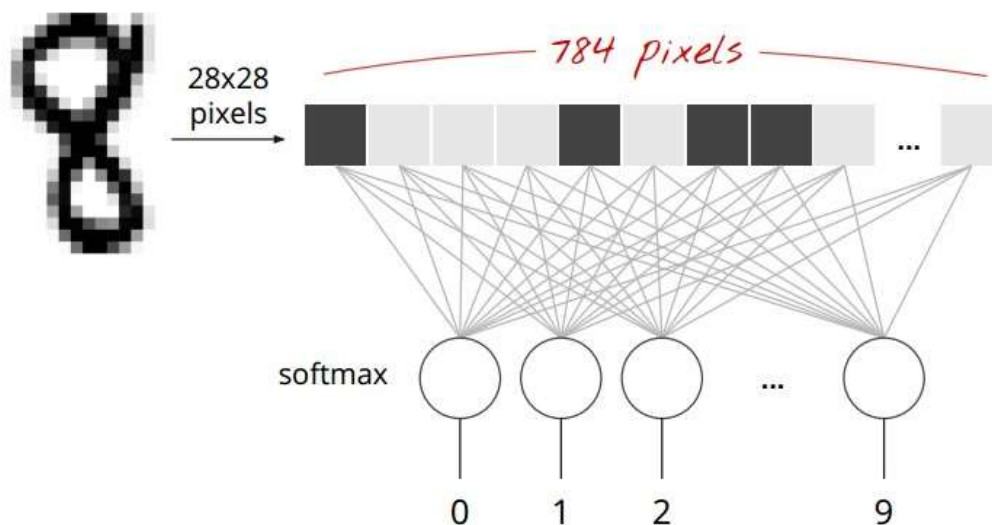
MNIST = Mixed National Institute of Standards and Technology - Download the dataset at <http://yann.lecun.com/exdb/mnist/>

10

# 深度前馈网络

## 单层

$$h_{\theta}(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1 | x^{(i)}; \theta) \\ p(y^{(i)} = 2 | x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k | x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$



*weighted sum of all pixels + bias*

$$\text{softmax}(L_n) = \frac{e^{L_n}}{\|e^L\|}$$

*neuron outputs*

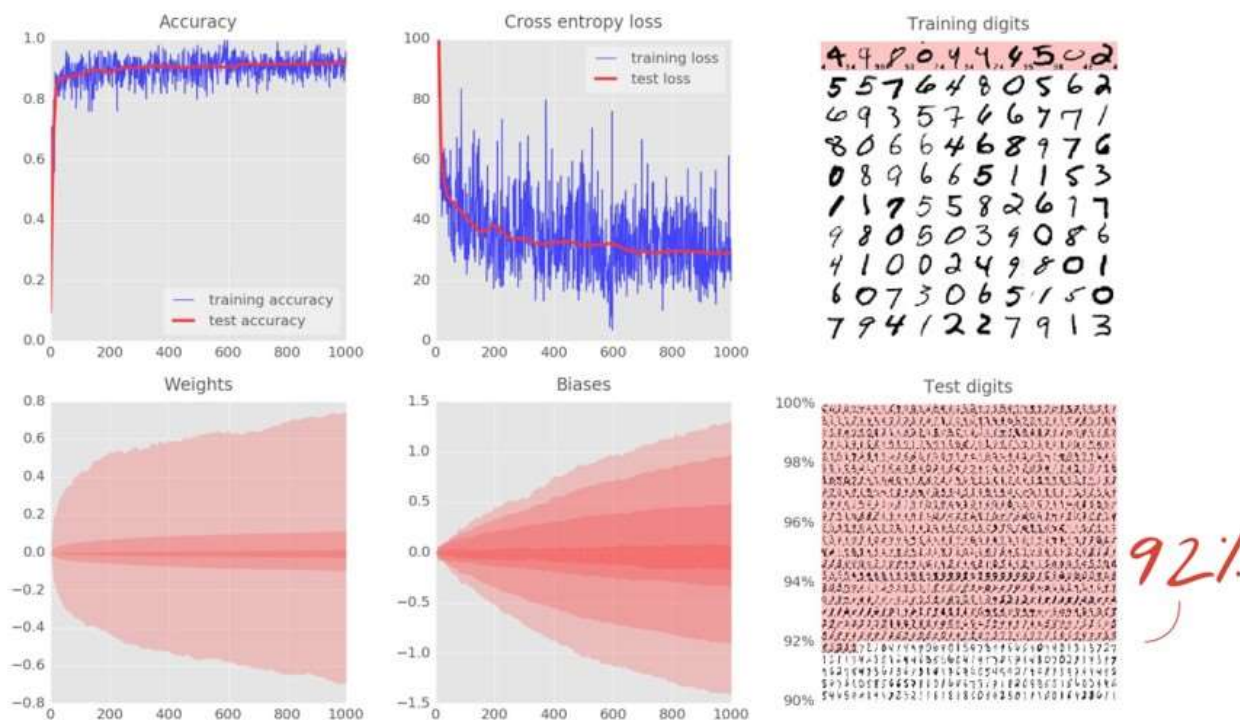
<http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/>

11

# 深度前馈网络

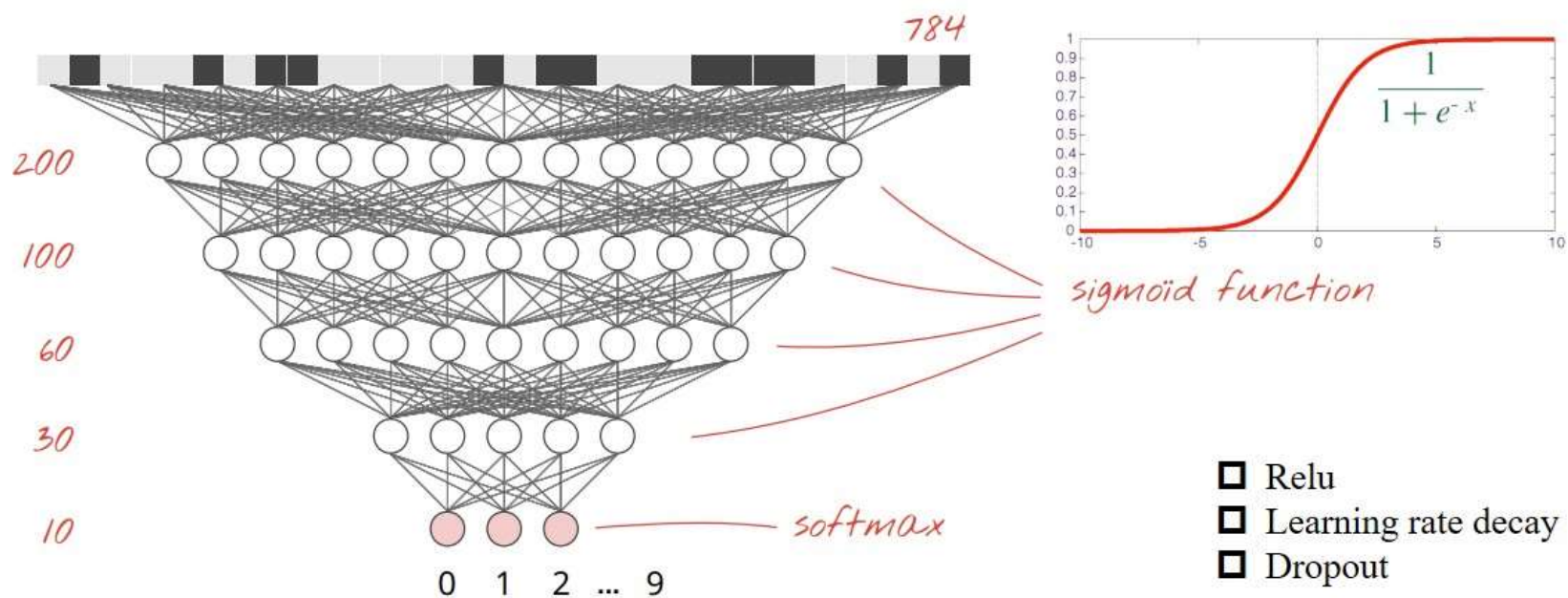
## 单层

<https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0>



# 深度前馈网络

## 多层



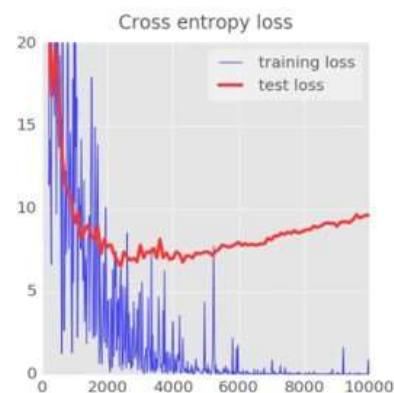
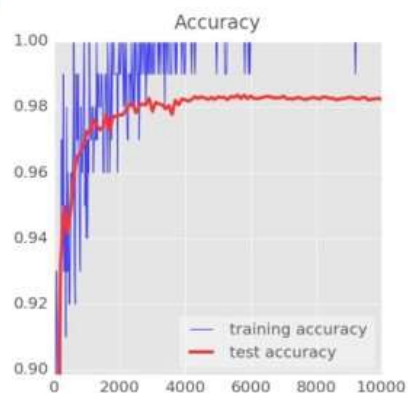
13



# 深度前馈网络

## 多层

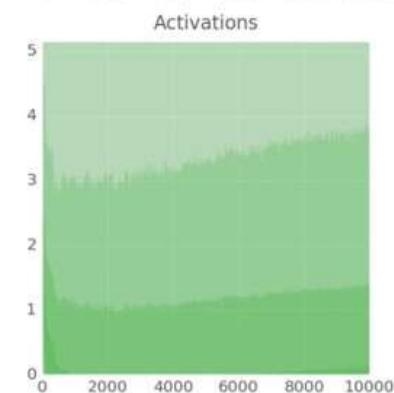
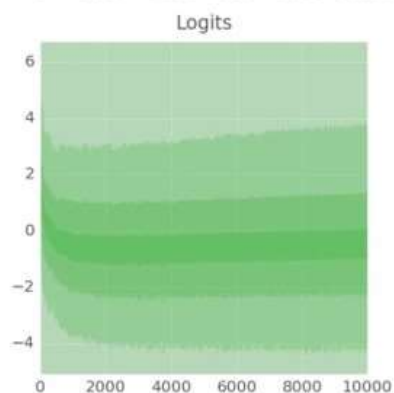
Too many neurons



Training digits

```

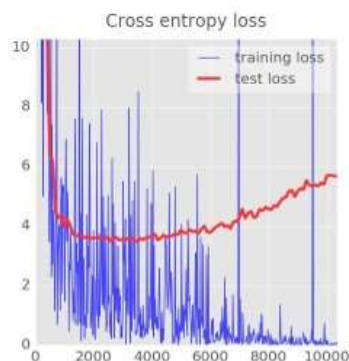
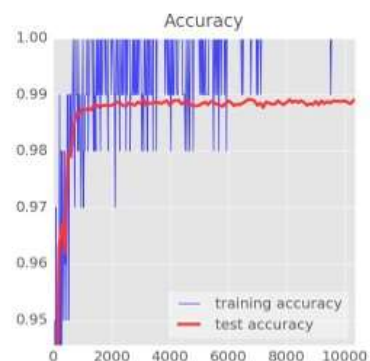
4071034180
5008360183
6003185349
1925802010
8483984258
0614538551
3276015854
8662091045
9964825151
0581127990
    
```



# 深度前馈网络

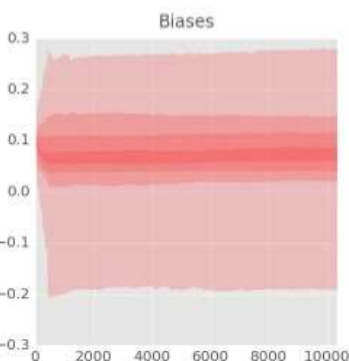
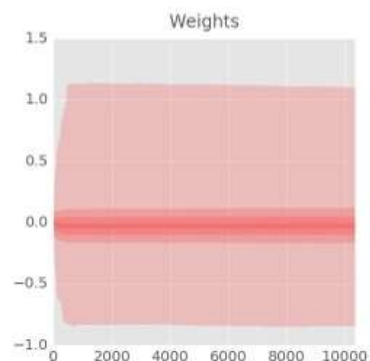
## 卷积层

Still Too many neurons



Training digits

7 4 9 6 2 9 1 6 1 1  
1 0 6 6 6 6 1 3 3 3  
7 0 7 2 6 1 9 5 4 6  
4 4 7 8 9 5 2 7 5 2  
7 0 9 0 4 5 0 6 5 4  
3 1 5 2 6 2 5 2 7 7  
1 2 1 7 0 6 8 0 3 9  
5 9 9 9 7 2 7 7 6 5  
9 9 2 8 7 1 3 7 4 7  
1 0 1 6 5 7 8 7 8 6

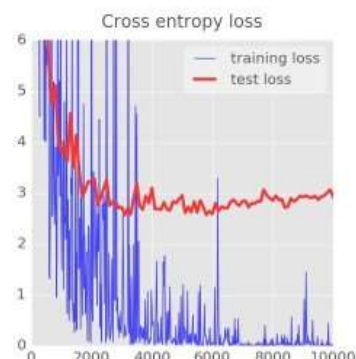
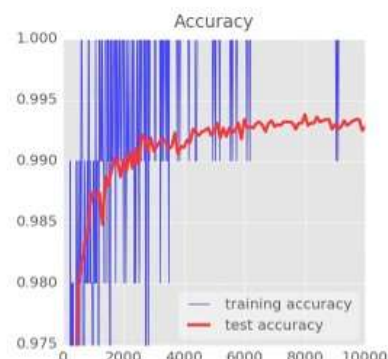


Test digits

100%  
98%  
96%  
94%  
92%  
90%

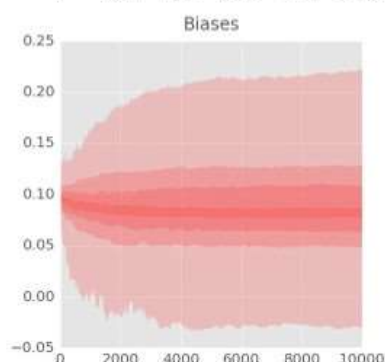
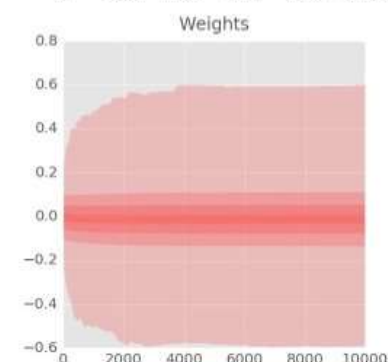
98.9%

## Bigger卷积+ dropout



Training digits

9	4	1	9	2	1	6	9	4	9
0	1	6	1	5	3	8	1	9	3
9	6	8	9	9	0	1	6	2	8
0	1	2	6	3	3	1	1	5	8
9	3	2	4	5	2	0	0	0	9
3	3	4	9	5	0	5	6	1	4
7	9	7	3	6	4	1	9	5	2
9	0	1	8	0	4	5	6	8	2
6	3	0	1	0	0	5	7	2	5
1	2	4	4	2	8	5	6	7	0



Test digit	Percentage
0	98.5%
1	98.5%
2	98.5%
3	98.5%
4	98.5%
5	98.5%
6	98.5%
7	98.5%
8	98.5%
9	98.5%

99.31

Excellent



# 深度前馈网络

## 推导

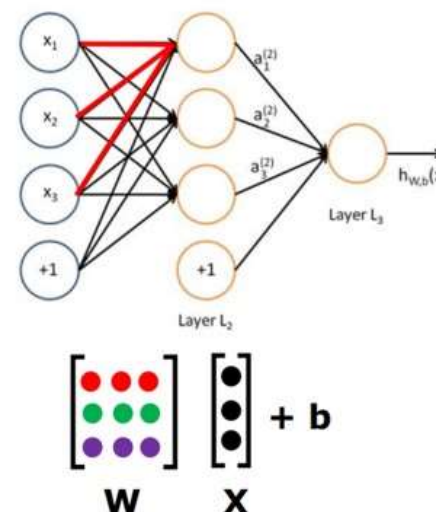
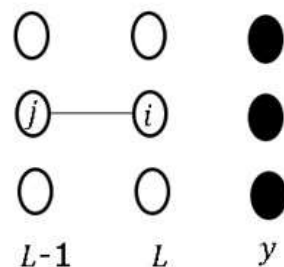
假设神经网络(NN)总共有  $L$  层

当第  $L-1$  层时, 权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-1}} = \frac{\partial J}{\partial z_i^L} \frac{\partial z_i^L}{\partial W_{ij}^{L-1}} = \delta_i^L a_j^{L-1}$$

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = \frac{\partial}{\partial z_i^L} \sum_{i=1}^{s_L} \frac{1}{2} \|y_i - f(z_i^L)\|^2 = -(y_i - f(z_i^L)) f'(z_i^L)$$

$$z_i^L = W_{ij}^{L-1} a_j^{L-1} + b_i^{L-1}$$



17

## 深度前馈网络

### 推导

假设神经网络(NN)总共有  $L$  层

当第  $L-1$  层时, 权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-1}} = \frac{\partial J}{\partial z_i^L} \frac{\partial z_i^L}{\partial W_{ij}^{L-1}} = \delta_i^L a_j^{L-1}$$

当第  $L-2$  层时, 权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-2}} = \frac{\partial J}{\partial z_i^{L-1}} \frac{\partial z_i^{L-1}}{\partial W_{ij}^{L-2}} = \delta_i^{L-1} a_j^{L-2}$$

sigmoid函数  $f'(z) = f(z)(1 - f(z))$

tanh函数  $f'(z) = 1 - (f(z))^2$

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = \frac{\partial}{\partial z_i^L} \sum_{i=1}^{s_L} \frac{1}{2} \|y_i - f(z_i^L)\|^2 = -(y_i - f(z_i^L))f'(z_i^L)$$

$$z_i^{L-1} = W_{ij}^{L-2} a_j^{L-2} + b_i^{L-2}$$

## 深度前馈网络

### 推导

假设神经网络(NN)总共有  $L$  层

当第  $L-1$  层时, 权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-1}} = \frac{\partial J}{\partial z_i^L} \frac{\partial z_i^L}{\partial W_{ij}^{L-1}} = \delta_i^L a_j^{L-1}$$

sigmoid函数  $f'(z) = f(z)(1 - f(z))$

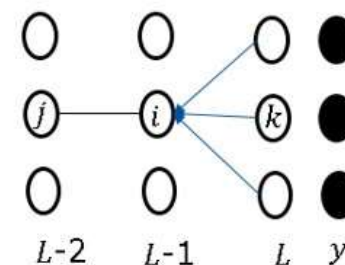
tanh函数  $f'(z) = 1 - (f(z))^2$

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = \frac{\partial}{\partial z_i^L} \sum_{i=1}^{s_L} \frac{1}{2} \|y_i - f(z_i^L)\|^2 = -(y_i - f(z_i^L)) f'(z_i^L)$$

当第  $L-2$  层时, 权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-2}} = \frac{\partial J}{\partial z_i^{L-1}} \frac{\partial z_i^{L-1}}{\partial W_{ij}^{L-2}} = \delta_i^{L-1} a_j^{L-2}$$

$$\begin{aligned} \delta_i^{L-1} &= \frac{\partial J}{\partial z_i^{L-1}} = \frac{\partial}{\partial z_i^{L-1}} \sum_{k=1}^{s_L} \frac{1}{2} \|y_k - f(z_k^L)\|^2 = \sum_{k=1}^{s_L} -(y_k - f(z_k^L)) f'(z_k^L) \frac{\partial z_k^L}{\partial z_i^{L-1}} \\ &= \sum_{k=1}^{s_L} \delta_k^L \cdot w_{ki}^{L-1} f'(z_i^{L-1}) \\ &= \left( \sum_{k=1}^{s_L} \delta_k^L w_{ki}^{L-1} \right) f'(z_i^{L-1}) \end{aligned}$$



$$z_k^L = W_{ki}^{L-1} a_i^{L-1} + b_k^{L-1}$$

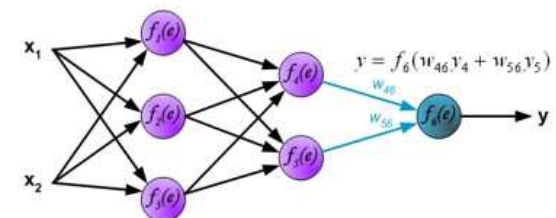
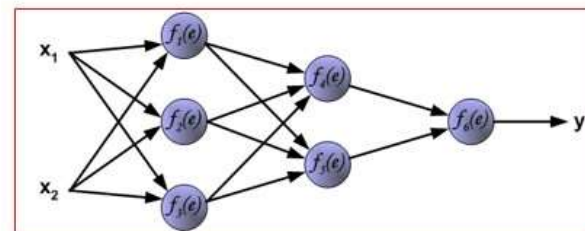
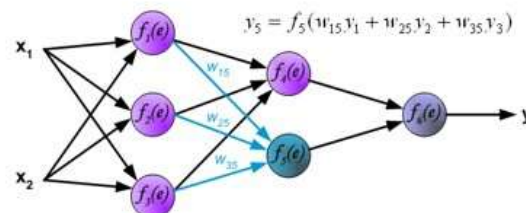
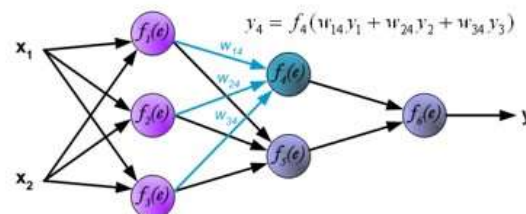
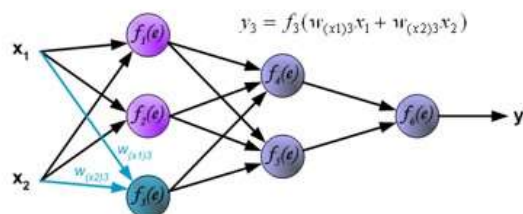
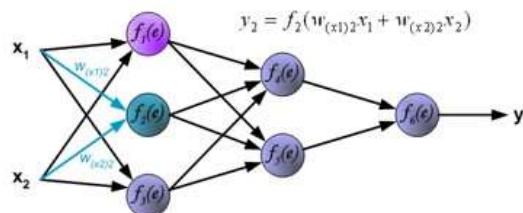
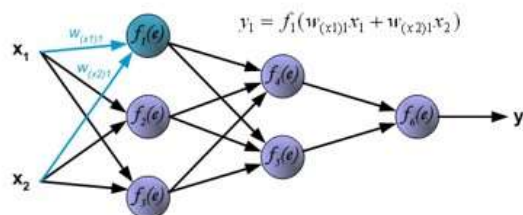
$$a_i^{L-1} = f(z_i^{L-1})$$

19

# 深度前馈网络

## 推导

[http://galaxy.agh.edu.pl/~vlsi/AI/backp\\_t\\_en/backprop.html](http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html)

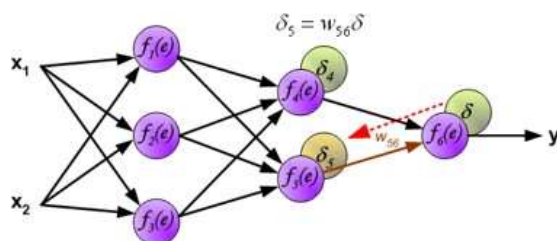
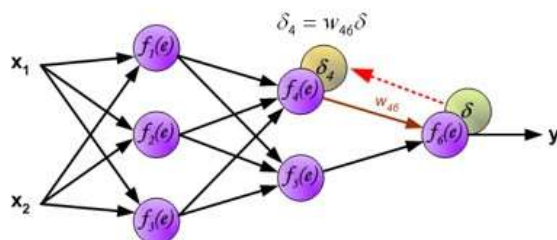
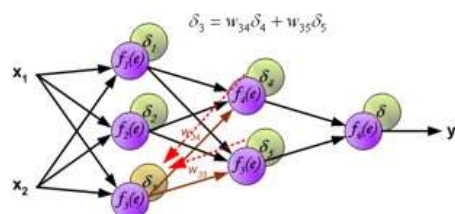
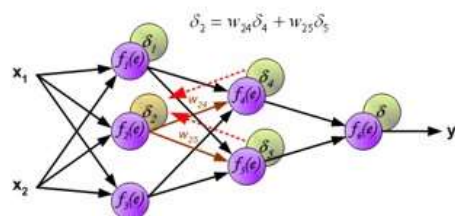
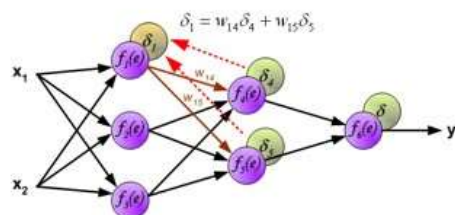


正向

20

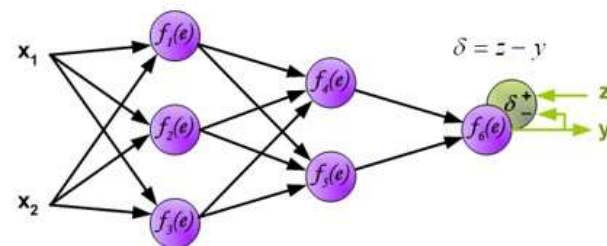
# 深度前馈网络

## 推导



$$\frac{\partial J}{\partial w_{ij}^{L-1}} = -(y_i - f(z_i^L))f'(z_i^L)a_j^{L-1}$$

$$\frac{\partial J}{\partial w_{ij}^{L-2}} = \left( \sum_{k=1}^{S_L} \delta_k^L w_{ki}^{L-1} \right) f'(z_i^{L-1}) a_j^{L-2}$$



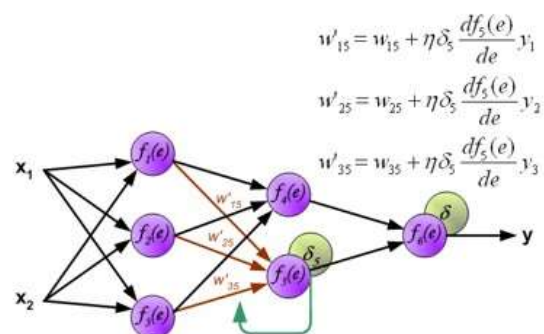
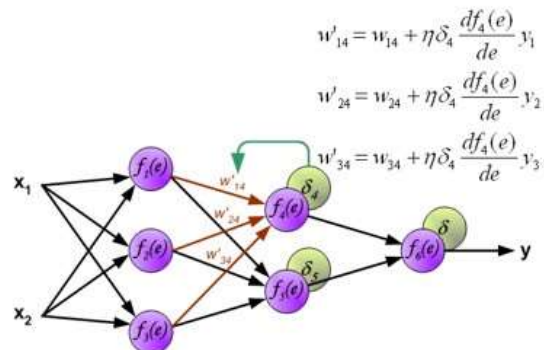
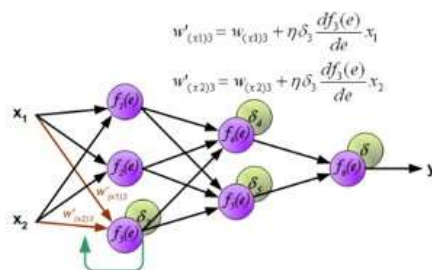
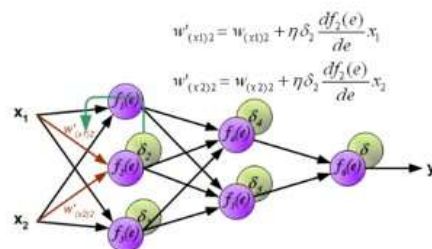
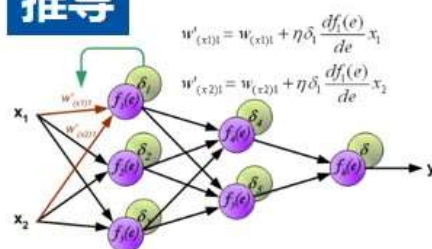
## 误差反向传导

21



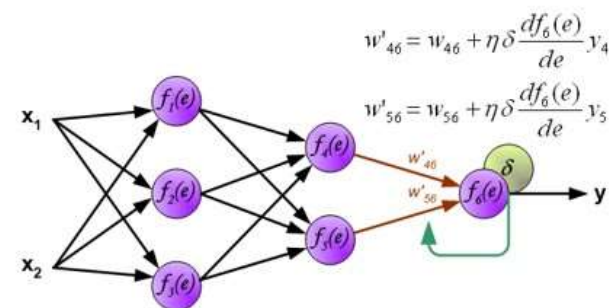
# 深度前馈网络

## 推导



$$\frac{\partial J}{\partial w_{ij}^{L-1}} = -(y_i - f(z_i^L)) f'(z_i^L) a_j^{L-1}$$

$$\frac{\partial J}{\partial w_{ij}^{L-2}} = \left( \sum_{k=1}^{s_L} \delta_k^L w_{ki}^{L-1} \right) f'(z_i^{L-1}) a_j^{L-2}$$



## 梯度更新

22

## 回顾：全连接网络的BP算法

### 反向传播的梯度计算

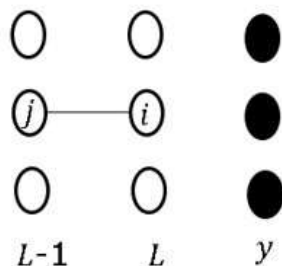
假设神经网络(NN)总共有  $L$  层

当第  $L-1$  层时，权重求导

$$\frac{\partial J}{\partial W_{ij}^{L-1}} = \frac{\partial J}{\partial z_i^L} \frac{\partial z_i^L}{\partial W_{ij}^{L-1}} = \delta_i^L a_j^{L-1}$$

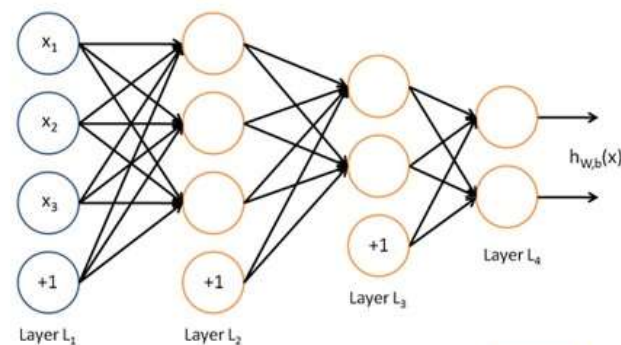
$$\frac{\partial J}{\partial W^{L-1}} = \delta^L * (a^{L-1})^T$$

$$\begin{aligned} \delta_i^L &= \frac{\partial J}{\partial z_i^L} = \frac{\partial J}{\partial f(z_i^L)} \frac{\partial f(z_i^L)}{\partial z_i^L} \\ &= \frac{\partial}{\partial f(z_i^L)} \sum_{i=1}^{s_L} \frac{1}{2} \|y_i - f(z_i^L)\|^2 f'(z_i^L) \\ &= -(y_i - f(z_i^L)) f'(z_i^L) \end{aligned}$$



$$z^L = W^{L-1} a^{L-1} + b^{L-1}$$

$$\begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \text{red} & \text{red} & \text{red} & \text{red} \\ \text{green} & \text{green} & \text{green} & \text{green} \\ \text{purple} & \text{purple} & \text{purple} & \text{purple} \end{bmatrix} \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix} + \begin{bmatrix} \text{yellow} \\ \text{yellow} \\ \text{yellow} \end{bmatrix}$$



23

## 回顾：全连接网络的BP算法

### 反向传播的梯度计算

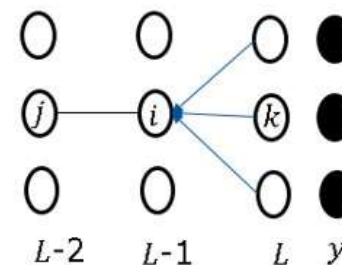
假设神经网络(NN)总共有  $L$  层

当第  $L-1$  层时，权重求导

$$\frac{\partial J}{\partial W^{L-1}} = \delta^L * (a^{L-1})^T$$

当第  $L-2$  层时，权重求导

$$\frac{\partial J}{\partial W^{L-2}} = \delta^{L-1} * (a^{L-2})^T$$



$$\delta_i^{L-1} = \left( \sum_{k=1}^{s_L} \delta_k^L w_{ki}^{L-1} \right) f'(z_i^{L-1})$$

$$\delta^{L-1} = \left[ (w^{L-1})^T * \delta^L \right] \odot f'(z^{L-1})$$



## 回顾：全连接网络的BP算法

### 推导Bias项

假设神经网络(NN)总共有  $L$  层

当第  $L-1$  层时，权重求导

$$\frac{\partial J}{\partial b_i^{L-1}} = \frac{\partial J}{\partial z_i^L} \frac{\partial z_i^L}{\partial b_i^{L-1}} = \delta_i^L$$

$$\frac{\partial J}{\partial b^{L-1}} = \delta^L$$

当第  $L-2$  层时，权重求导

$$\frac{\partial J}{\partial b^{L-2}} = \delta^{L-1}$$

sigmoid函数  $f'(z) = f(z)(1 - f(z))$

tanh函数  $f'(z) = 1 - (f(z))^2$

$$z_i^L = W_{ij}^{L-1} a_j^{L-1} + b_i^{L-1}$$

$$\delta_i^L = \frac{\partial J}{\partial z_i^L} = \frac{\partial}{\partial z_i^L} \sum_{i=1}^{s_L} \frac{1}{2} \|y_i - f(z_i^L)\|^2 = -(y_i - f(z_i^L)) f'(z_i^L)$$

$$z_i^{L-1} = W_{ij}^{L-2} a_j^{L-2} + b_i^{L-2}$$

残差传递和bias项无关

$$\begin{aligned} \delta_i^{L-1} &= \frac{\partial J}{\partial z_i^{L-1}} = \frac{\partial}{\partial z_i^{L-1}} \sum_{k=1}^{s_L} \frac{1}{2} \|y_k - f(z_k^L)\|^2 = \sum_{k=1}^{s_L} -(y_k - f(z_k^L)) f'(z_k^L) \frac{\partial z_k^L}{\partial z_i^{L-1}} \\ &= \sum_{k=1}^{s_L} \delta_k^L \cdot w_{ki}^{L-1} f'(z_i^{L-1}) \\ &= \left( \sum_{k=1}^{s_L} \delta_k^L w_{ki}^{L-1} \right) f'(z_i^{L-1}) \end{aligned}$$

★

$$z_k^L = W_{ki}^{L-1} a_i^{L-1} + b_k^{L-1}$$

$$a_i^{L-1} = f(z_i^{L-1})$$

25

## 深度前馈网络

### 梯度消失和爆炸

Diagram of a neural network with layers  $L-2$ ,  $L-1$ , and  $L$ . Weights  $w^{L-2}$  and  $w^{L-1}$  are indicated. The forward pass is defined as:

$$z^{L+1} = w^{L+1} a^L + b^{L+1}$$

$$a^{L+1} = f(z^{L+1})$$

A graph of the sigmoid function  $f = \text{sigmoid}$  is shown, with the curve approaching 0 and 1 asymptotically.

The derivative of the loss with respect to the weights is given by:

$$\frac{\partial J}{\partial w_{ij}^{L-2}} = \frac{\partial J}{\partial z_i^{L-1}} \frac{\partial z_i^{L-1}}{\partial w_{ij}^{L-2}} = \delta_i^{L-1} a_j^{L-2} = \left( \sum_{k=1}^n \delta_k^L w_{ki}^{L-1} \right) f'(z_i^{L-1}) a_j^{L-2}$$

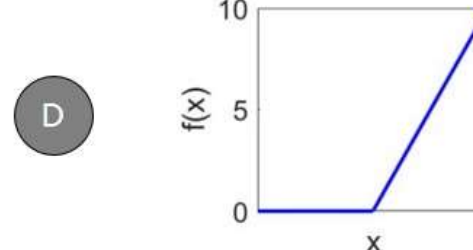
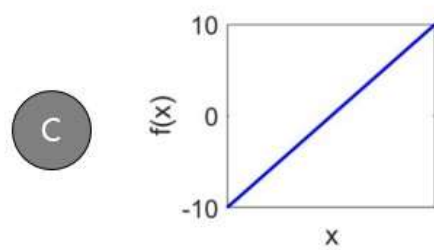
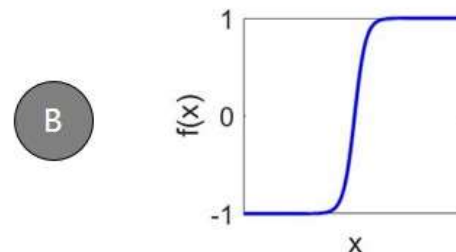
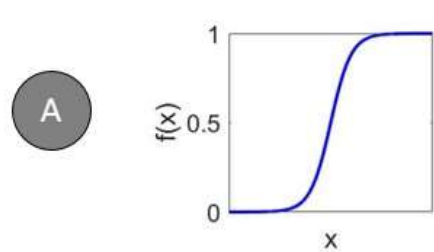
关于梯度消失, 当  $|w| < 1$  时, 则反向传播至  $L-2$  层时, 梯度会越来越小。

- 另一个原因, 若使用 sigmoid, 则  $f$  在很多点处导数为 0, 这也是梯度消失的一个原因。

关于梯度爆炸, 当  $|w| > 1$  时, 反向传播至  $L-2$  层时, 梯度会越来越大。

单选题 1分

下列哪个表示激励函数ReLU的图像( )



# PART 万能近似性质 TWO

## 深度前馈网络

### 通用近似定理

**定理 4.1 – 通用近似定理 (Universal Approximation Theorem)**

**[Cybenko, 1989, Hornik et al., 1989]:** 令  $\varphi(\cdot)$  是一个非常数、有界、单调递增的连续函数,  $\mathcal{I}_d$  是一个  $d$  维的单位超立方体  $[0, 1]^d$ ,  $C(\mathcal{I}_d)$  是定义在  $\mathcal{I}_d$  上的连续函数集合。对于任何一个函数  $f \in C(\mathcal{I}_d)$ , 存在一个整数  $m$ , 和一组实数  $v_i, b_i \in \mathbb{R}$  以及实数向量  $\mathbf{w}_i \in \mathbb{R}^d, i = 1, \dots, m$ , 以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{i=1}^m v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i), \quad (4.33)$$

作为函数  $f$  的近似实现, 即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_d. \quad (4.34)$$

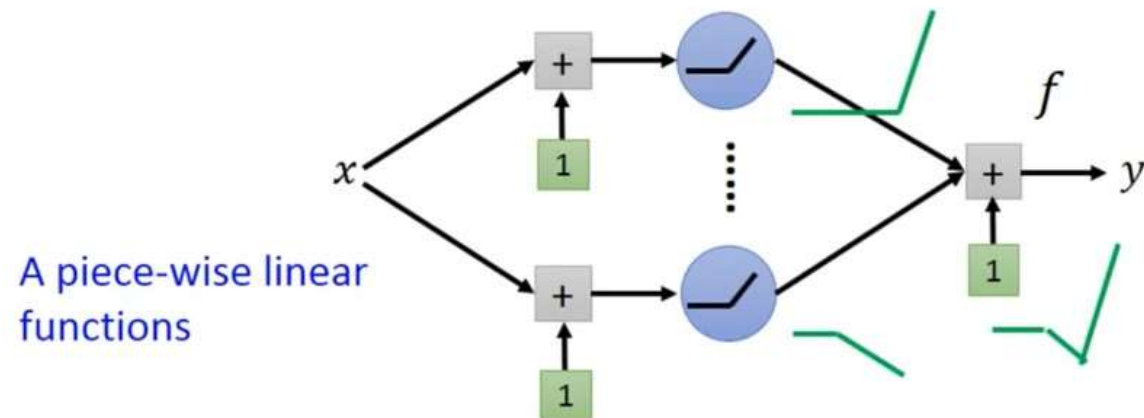
其中  $\epsilon > 0$  是一个很小的正数。

根据通用近似定理, 对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的前馈神经网络, 只要其隐藏层神经元的数量足够, 它可以以任意的精度来近似任何从一个定义在实数空间中的有界闭集函数。

## 深度前馈网络

### 通用近似定理

- Given a **shallow** network structure with one hidden layer with ReLU activation and linear output



- Given a L-Lipschitz function  $f^*$ 
  - How many neurons are needed to approximate  $f^*$ ?

## 深度前馈网络

### 通用近似定理

- Given a  $L$ -Lipschitz function  $f^*$ 
  - How many neurons are needed to approximate  $f^*$ ?

$L$ -Lipschitz Function (smooth)

$$\|f(x_1) - f(x_2)\| \leq L \|x_1 - x_2\|$$

Output  
change

Input  
change

$L=1$  for "1 - Lipschitz"





## 深度前馈网络

### 通用近似定理

$$\begin{aligned} \max_{0 \leq x \leq 1} |f(x) - f^*(x)| &\leq \varepsilon \\ \downarrow \\ \sqrt{\int_0^1 |f(x) - f^*(x)|^2 dx} &\leq \varepsilon \end{aligned}$$

- Given a L-Lipschitz function  $f^*$
- How many neurons are needed to approximate  $f^*$ ?

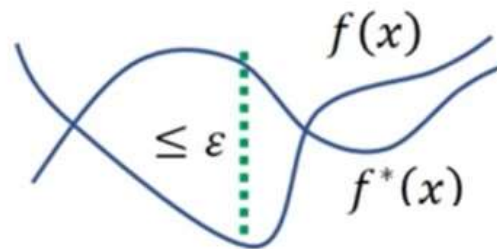
$f \in N(K)$   The function space defined by the network with K neurons.

Given a small number  $\varepsilon > 0$

What is the number of  $K$  such that

$$\text{Exist } f \in N(K), \max_{0 \leq x \leq 1} |f(x) - f^*(x)| \leq \varepsilon$$

The difference between  $f(x)$  and  $f^*(x)$  is smaller than  $\varepsilon$ .



32



## 深度前馈网络

### 通用近似定理

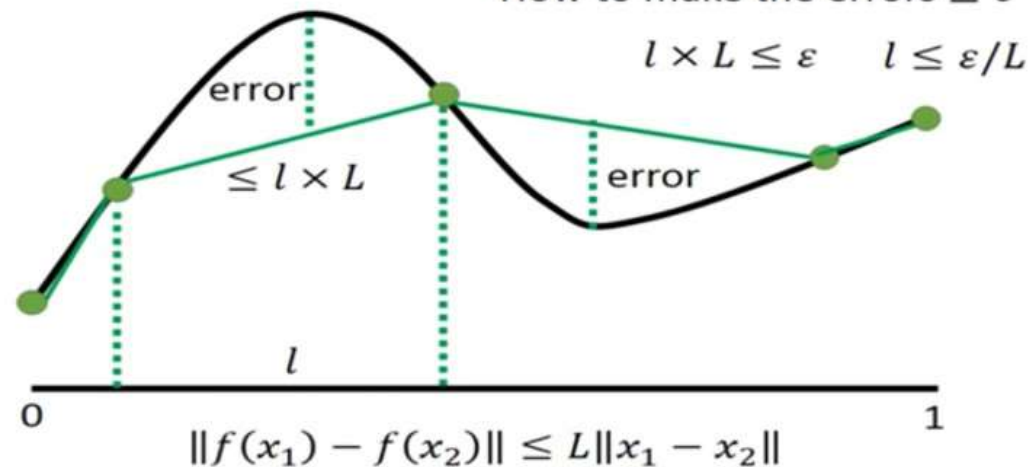
#### Universality

- L-Lipschitz function  $f^*$

All the functions in  $N(K)$  are piecewise linear.

Approximate  $f^*$  by a piecewise linear function  $f$

How to make the errors  $\leq \varepsilon$



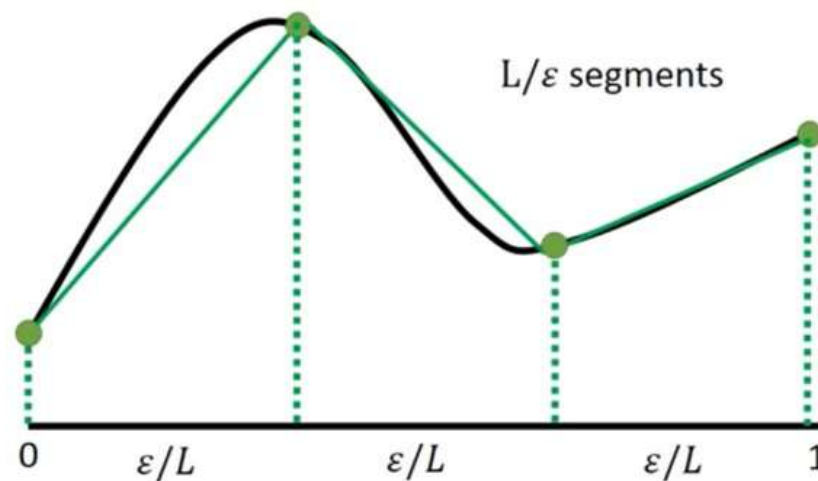
## 深度前馈网络

### 通用近似定理

#### Universality

- L-Lipschitz function  $f^*$

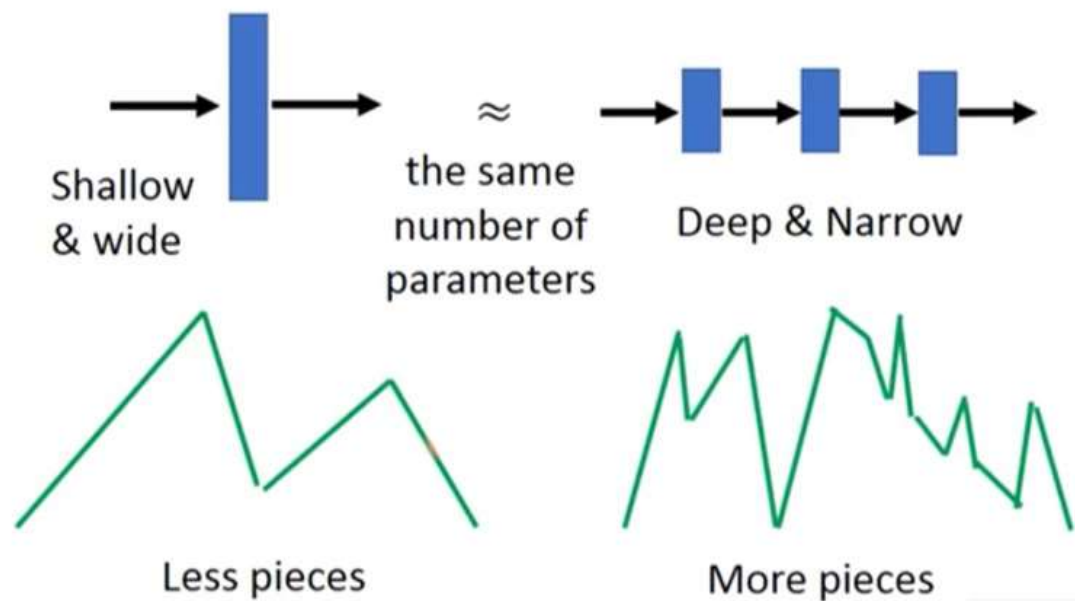
How to make a 1 hidden layer relu network have the output like green curve?



## 深度前馈网络

### 通用近似定理

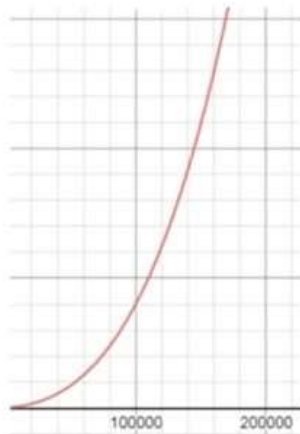
- ReLU networks can represent piecewise linear functions



## 深度前馈网络

### More Theory

- A function expressible by a 3-layer feedforward network cannot be approximated by 2-layer network.
  - Unless the width of 2-layer network is VERY large
  - Applied on activation functions beyond relu



The width of 3-layer network is  $K$ .

The width of 2-layer network  
should be  $Ae^{BK^{4/19}}$

Ronen Eldan, Ohad Shamir, "The Power of Depth for Feedforward Neural Networks", COLT, 2016

## 深度前馈网络

### More Theory

- A function expressible by a deep feedforward network cannot be approximated by a shallow network.
  - Unless the width of the shallow network is VERY large
  - Applied on activation functions beyond relu

Deep Network:

$\Theta(k^3)$  layers,  $\Theta(1)$  nodes per layer,  $\Theta(1)$  distinct parameters

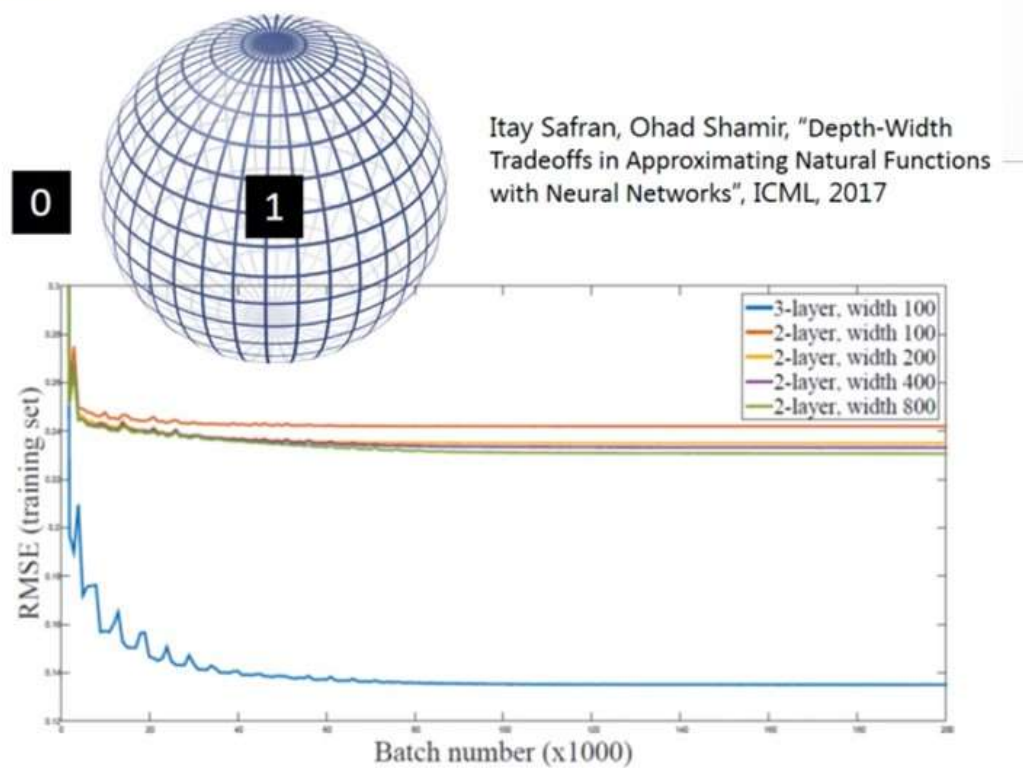
Shallow Network:  $\Theta(k)$  layers   $\Omega(2^k)$  nodes

Matus Telgarsky, "Benefits of depth in neural networks", COLT, 2016

37

## 深度前馈网络

### More Theory



38

# PART **TensorFlow** THREE

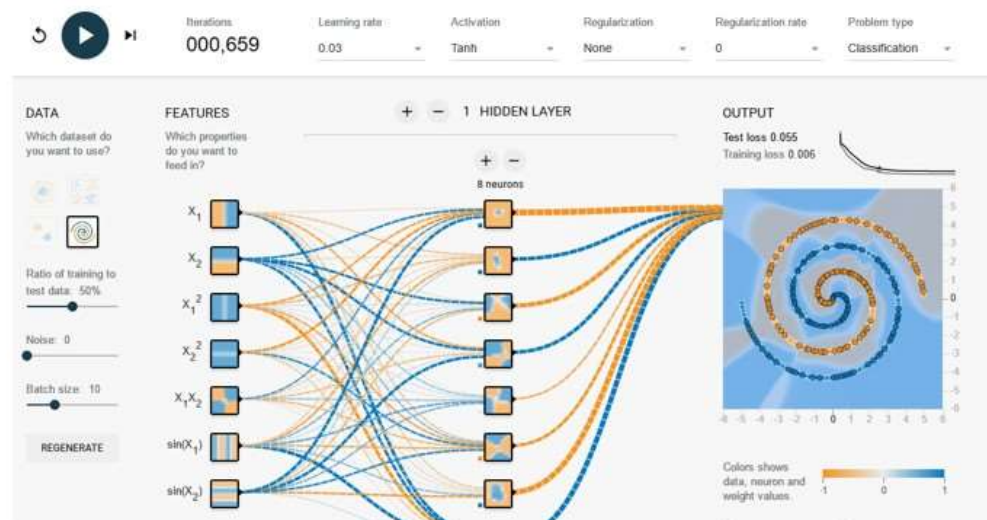
<http://playground.tensorflow.org/>

Tinker With a **Neural Network** Right Here in Your Browser.  
Don't Worry, You Can't Break It. We Promise.





# TensorFlow PlayGround



- ❑ 选择Sigmoid函数作为激活函数，明显能感觉到训练的时间很长，ReLU函数能大大加快收敛速度
- ❑ 当把隐含层数加深后，会发现Sigmoid函数作为激活函数，训练过程loss降不下来
- ❑ 隐含层的数量不是越多越好，层数和特征的个数太多，会造成优化的难度和出现过拟合的现象
- ❑ 只需要输入最基本的特征 $x_1$ ,  $x_2$ ，只要给予足够多层的神经网络和神经元，神经网络会自己组合出最有用的特征

# TensorFlow PlayGround

Epoch  
001,892

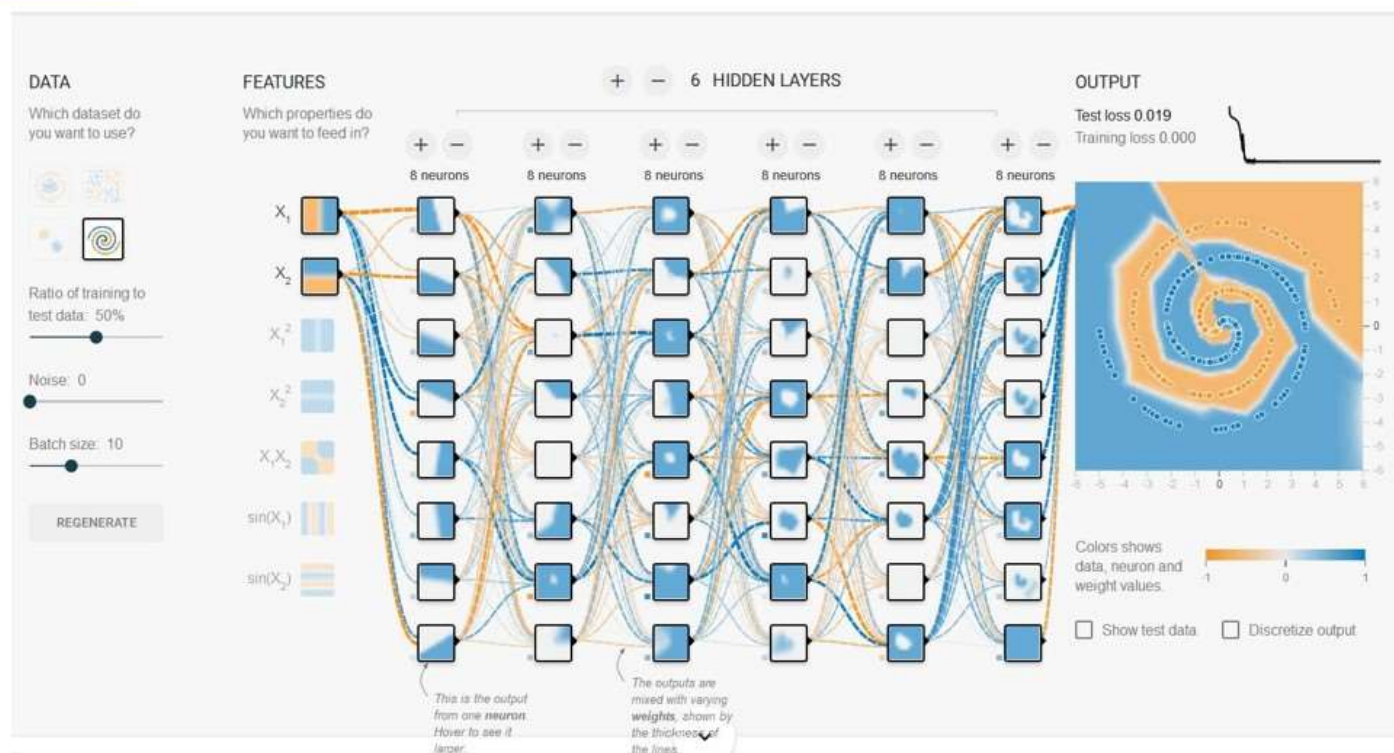
Learning rate  
0.03

Activation  
ReLU

Regularization  
None

Regularization rate  
0

Problem type  
Classification



42

# TensorFlow

## 安装

1. Windows



CPU版:

环境: python 3.5, 3.6(64位)

本地pip安装:

```
pip3 install --upgrade tensorflow
```

Anaconda安装:

(1) 创建一个名为tensorflow的conda环境

```
conda create -n tensorflow pip python = 3.5
```

(2) 激活conda

```
activate tensorflow环境
```

(3) 在conda环境中安装TensorFlow

```
pip install --ignore-installed --upgrade tensorflow
```

# TensorFlow

## 安装

### 1. Windows



#### GPU版:

环境

- (1) python3.5及以上 (64位)
- (2) GPU卡: 计算力不小于3.0的NVIDIA显卡
- (3) CUDA工具包9.0
- (4) cuDNN v7.0

### 2. Ubuntu (Ubuntu 16.04或更高版本)

环境与window要求一致

[https://tensorflow.google.cn/install/install\\_linux](https://tensorflow.google.cn/install/install_linux)



### 3. macOS (macOS X 10.11 (El Capitan) 或更高版本)

[https://tensorflow.google.cn/install/install\\_mac](https://tensorflow.google.cn/install/install_mac)



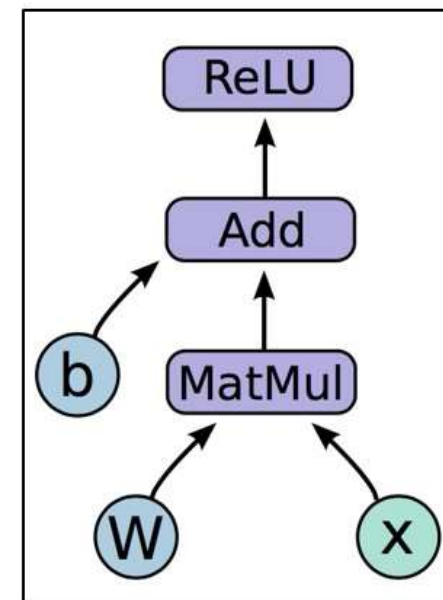
44

# TensorFlow

## Basic concepts

- Express a numeric computation as a **graph**.
- Graph nodes are **operations** which have any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes

$$h_i = \text{ReLU}(Wx + b)$$



45



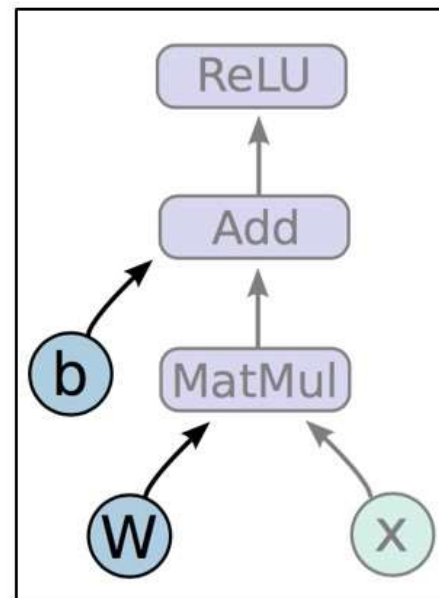
# TensorFlow

## Basic concepts

$$h_i = \text{ReLU}(Wx + b)$$

**Variables** are 0-ary stateful nodes which output their current value. (State is retained across multiple executions of a graph.)

(parameters, gradient stores, eligibility traces, ...)



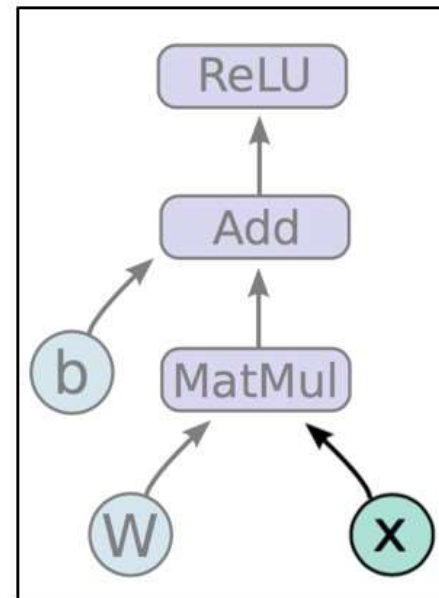
# TensorFlow

## Basic concepts

$$h_i = \text{ReLU}(Wx + b)$$

**Placeholders** are 0-ary nodes whose value is fed in at execution time.

(inputs, variable learning rates, ...)





# TensorFlow

## Basic concepts

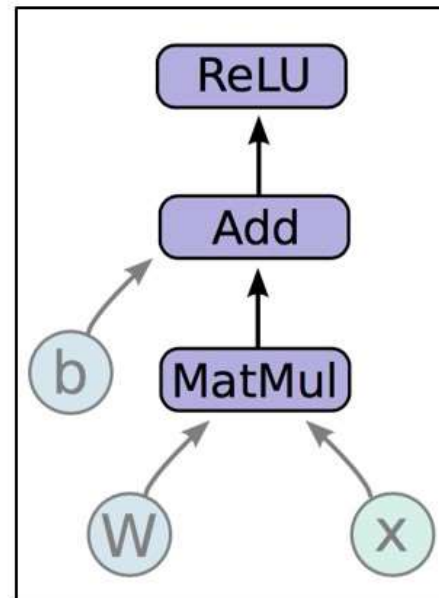
$$h_i = \text{ReLU}(Wx + b)$$

### Mathematical operations:

**MatMul:** Multiply two matrix values.

**Add:** Add elementwise (with broadcasting).

**ReLU:** Activate with elementwise rectified linear function.



# TensorFlow

## Basic concepts

1. Create model weights,  
including initialization

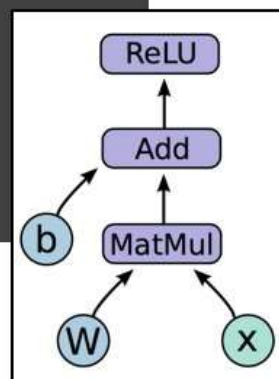
$$a. W \sim \text{Uniform}(-1, 1); b = 0$$

2. Create input placeholder x

$a.m * 784$  input matrix

3. Create computation  
graph

$$h_i = \text{ReLU}(Wx + b)$$



```
import tensorflow as tf
```

```
1  b = tf.Variable(tf.zeros((100,)))  
   W = tf.Variable(tf.random_uniform((784, 100),  
                                     -1, 1))  
2  x = tf.placeholder(tf.float32, (None, 784))  
3  h_i = tf.nn.relu(tf.matmul(x, W) + b)
```

## Just Run It!

```
sess = tf.Session()  
sess.run(tf.initialize_all_variables())  
sess.run(h_i, {x: np.random.random(64, 784)})
```



# TensorFlow

## Basic concepts

### 1. Build a graph

- a. Graph contains parameter specifications, model architecture, optimization process, ...
- b. Somewhere between 5 and 5000 lines

### 2. Initialize a session

### 3. Fetch and feed data with `Session.run`

- a. Compilation, optimization, etc. happens at this step  
— you probably won't notice

## 深度前馈网络

### 单层

tensor shapes:  $X[100, 748]$   $W[748, 10]$   $b[10]$

$Y = \text{tf.nn.softmax}(\text{tf.matmul}(X, W) + b)$

matrix multiply

broadcast on all lines

51

## 深度前馈网络

### 单层

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

Cross entropy:  $-\sum Y_i' \cdot \log(Y_i)$

computed probabilities

0.1	0.2	0.1	0.3	0.2	0.1	0.9	0.2	0.1	0.1
0	1	2	3	4	5	6	7	8	9

this is a "6"

## 深度前馈网络

### 单层

```
import tensorflow as tf
```

*this will become the batch size, 100*

```
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
```

```
W = tf.Variable(tf.zeros([784, 10]))
```

```
b = tf.Variable(tf.zeros([10]))
```

*28 x 28 grayscale images*

```
init = tf.initialize_all_variables()
```

*Training = computing variables W and b*

## 深度前馈网络

### 单层

```
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
# loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))
# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

*flattening images*

*"one-hot" encoded*

*"one-hot" decoding*

54



## 深度前馈网络

### 单层

```
optimizer = tf.train.GradientDescentOptimizer(0.003)  
train_step = optimizer.minimize(cross_entropy)
```

*learning rate*

*loss function*

55

## 深度前馈网络

### 单层

```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # Load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a,c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data={X: mnist.test.images, Y_: mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy, It], feed=test_data)
```

*running a Tensorflow  
computation, feeding  
placeholders*

*Tip:  
do this  
every 100  
iterations*

# 深度前馈网络

## 单层

```
import tensorflow as tf
```

*initialisation*

```
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
init = tf.initialize_all_variables()
```

*model*

```
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
```

```
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

*success metrics*

```
# Loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))
```

```
# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

*training step*

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```

```
sess = tf.Session()
sess.run(init)
```

```
for i in range(10000):
    # Load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data = {X: batch_X, Y_: batch_Y}
```

```
# train
sess.run(train_step, feed_dict=train_data)
```

*Run*

```
# success ? add code to print it
a, c = sess.run([accuracy, cross_entropy], feed=train_data)
```

```
# success on test data ?
test_data = {X: mnist.test.images, Y_: mnist.test.labels}
a, c = sess.run([accuracy, cross_entropy], feed=test_data)
```

57

**THANK YOU**  
**Q&A**