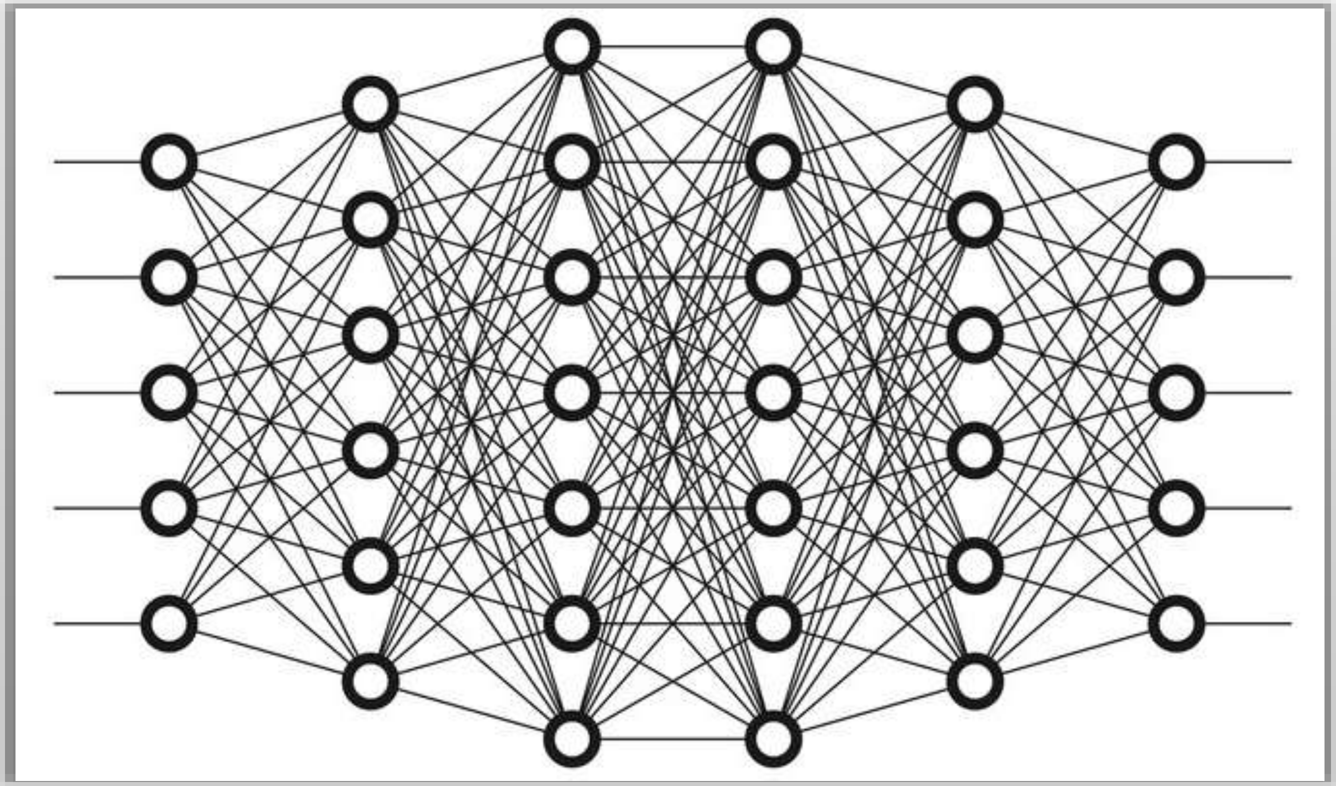
2023 年度云南大学软件学院本科生课程

机器学习

教 师： 李 劲（lijin@ynu.edu.cn）

2023 年 9 月



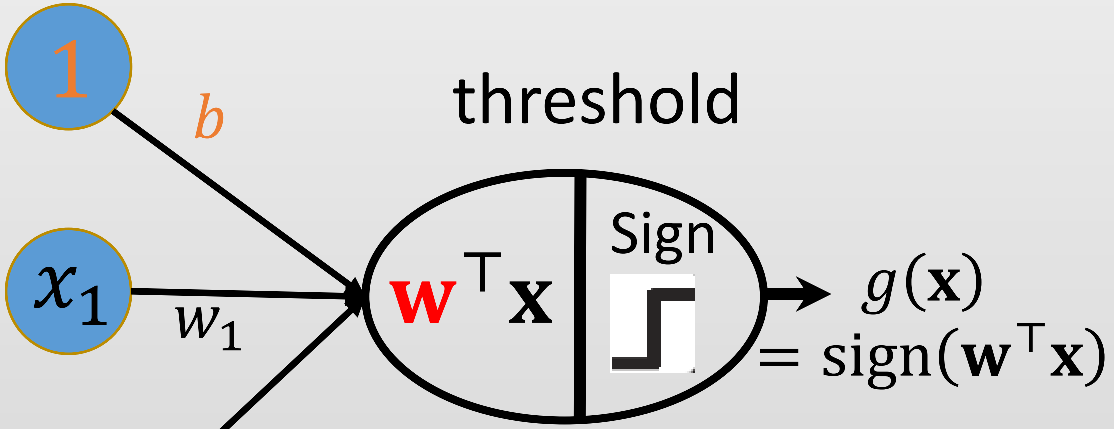
4. Neural Network

4.1 MLP as an Universal Approximator

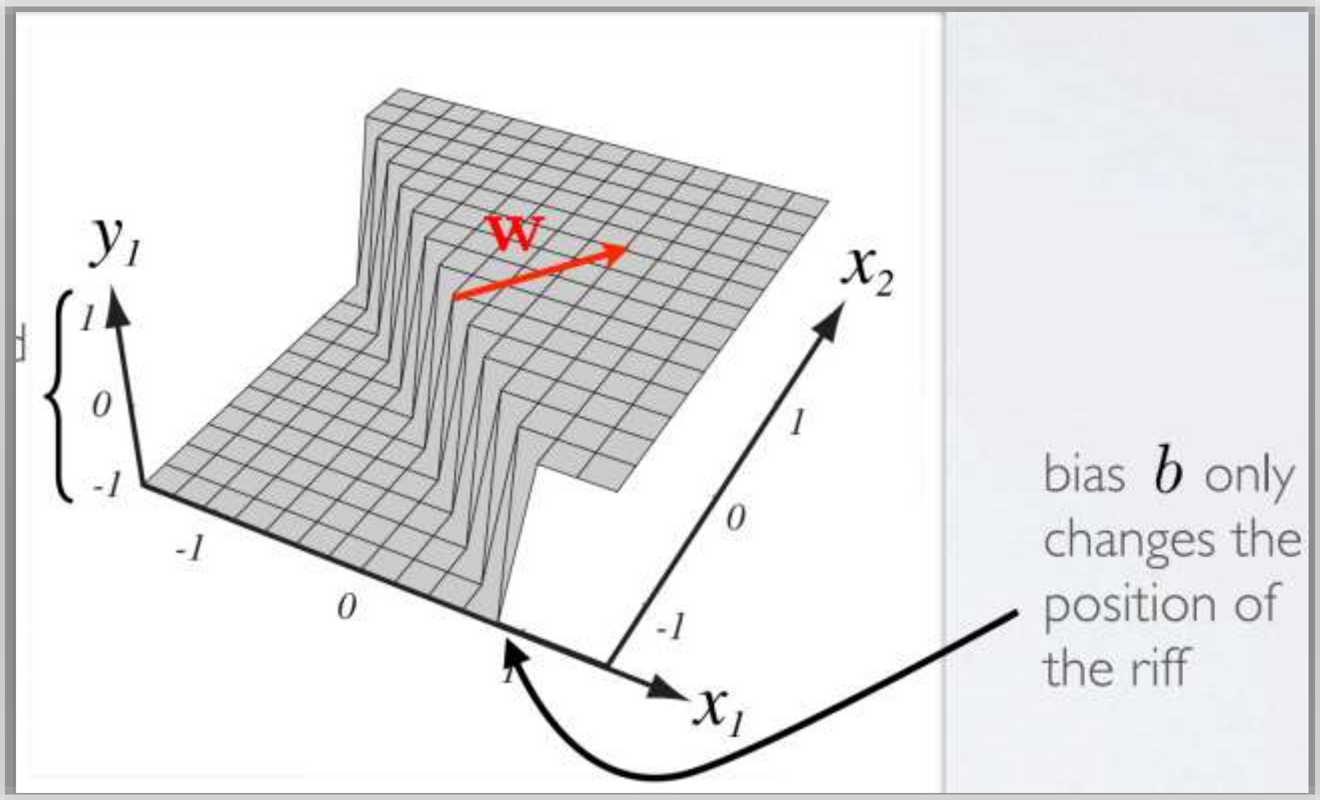
Perceptron

$$\sum_{i=1}^{d} w_i x_i > \text{threshold} \rightarrow g(\mathbf{x}) = +1$$

$$\sum_{i=1}^{d} w_i x_i < \text{threshold} \rightarrow g(\mathbf{x}) = -1$$



$$\mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$$

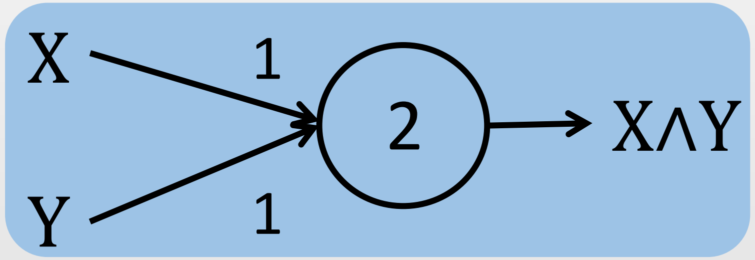


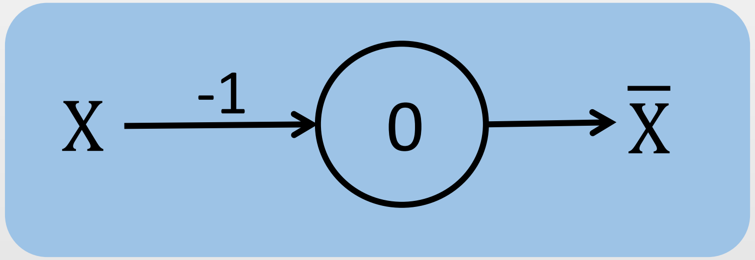
Universal Boolean function approximator

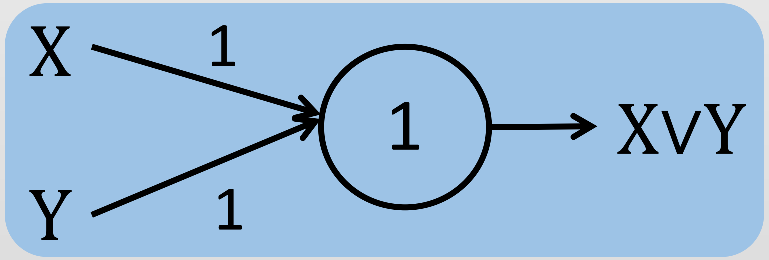
Multi-layer Perceptron model any Boolean function

$X, Y \in \{0,1\}$ are Boolean variables.

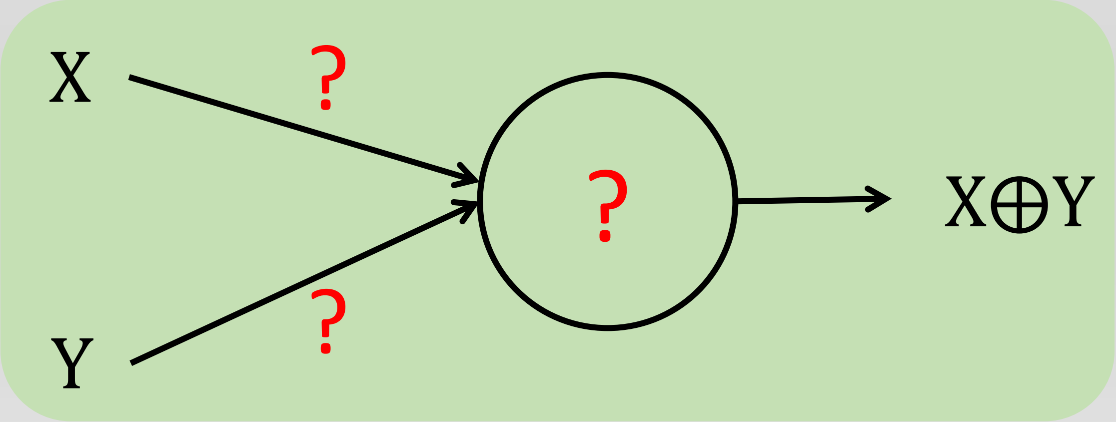
X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1







X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

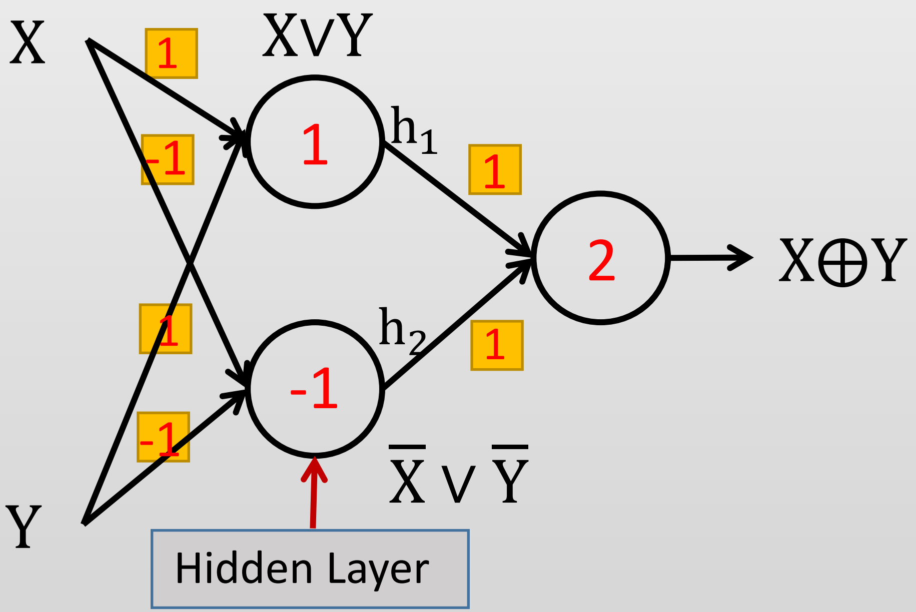


MLP models XOR

$X, Y \in \{0,1\}$
are Boolean variables.

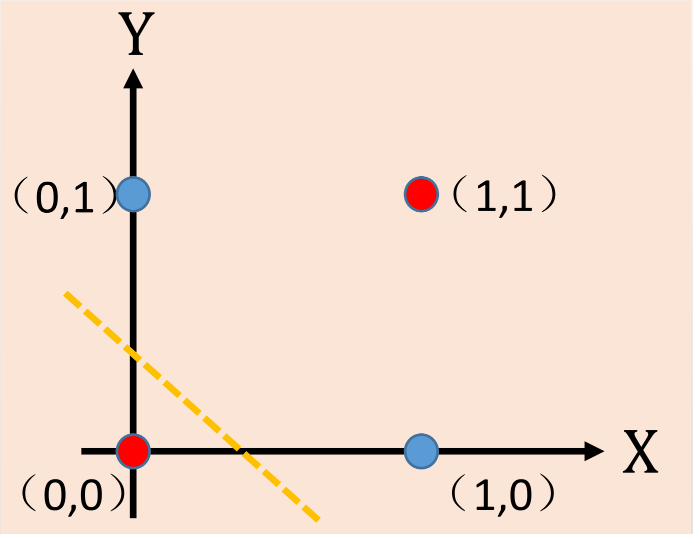
Multi-layer perceptron

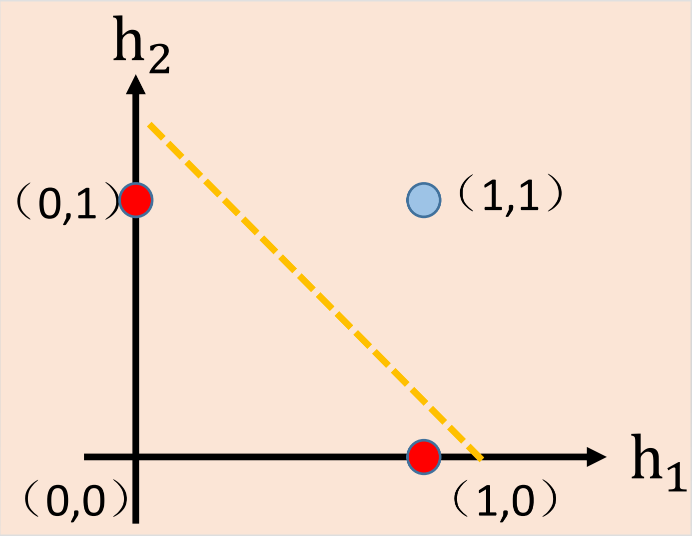
X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0



X	Y	h_1	h_2
0	0	0	1
0	1	1	1
1	0	1	1
1	1	1	0

$$X \oplus Y \Leftrightarrow (X \vee Y) \wedge (\neg X \vee \neg Y)$$





MLP models any Boolean function

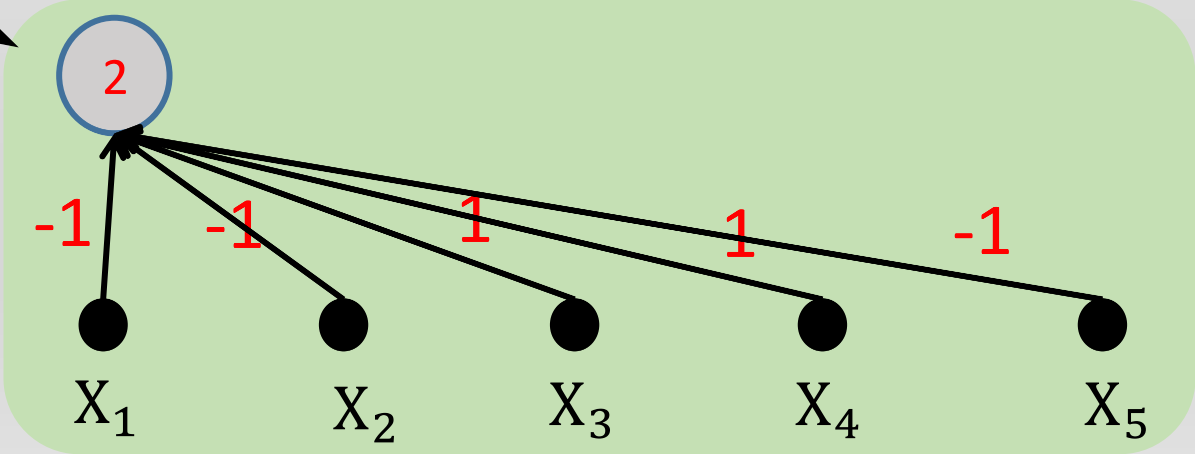
Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows all input combinations for which output is 1

$$Y = \overline{X_1}\,\overline{X_2}\,X_3\,X_4\,\overline{X_5} + \overline{X_1}\,X_2\overline{X_3}\,X_4\,X_5 + \overline{X_1}\,X_2\overline{X_3}\,X_4\,\overline{X_5} + X_1\overline{X_2}\,\overline{X_3}\,\overline{X_4}\,X_5 + X_1\overline{X_2}\,X_3\,X_4\,X_5 + X_1\,X_2\overline{X_3}\,\overline{X_4}\,X_5$$

主析取范式：所有小项的析取



Two-layer MLP is a Universal Boolean Function

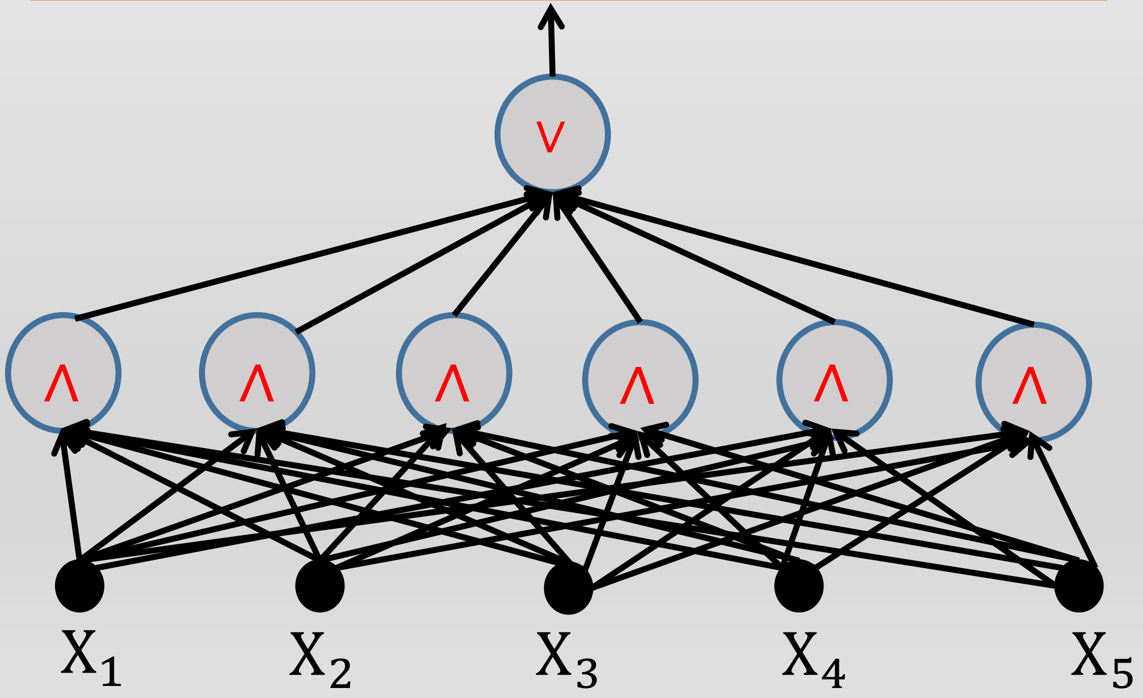
Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

主析取范式：所有小项的析取

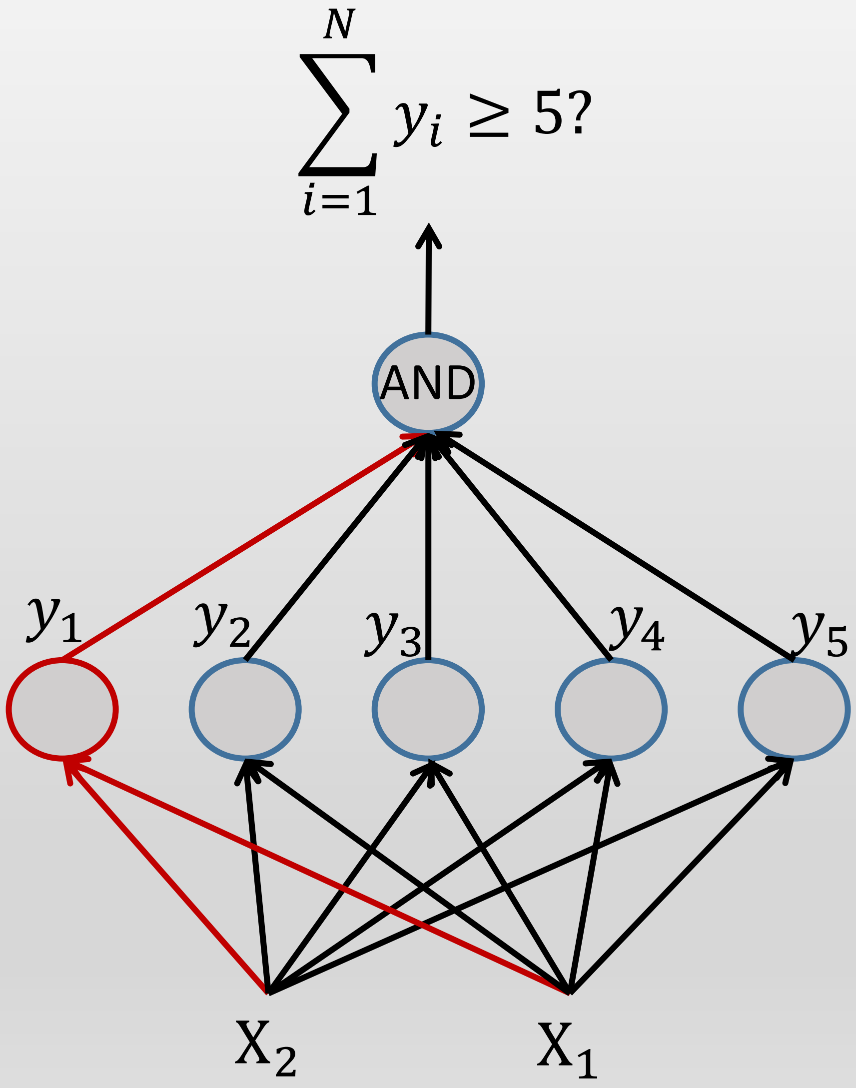
$$Y = \overline{X_1}\,\overline{X_2}\,X_3\,X_4\,\overline{X_5} + \overline{X_1}\,X_2\overline{X_3}\,X_4\,X_5 + \overline{X_1}\,X_2\overline{X_3}\,X_4\,\overline{X_5} + X_1\overline{X_2}\,\overline{X_3}\,\overline{X_4}\,X_5 + X_1\overline{X_2}\,X_3\,X_4\,X_5 + X_1\,X_2\overline{X_3}\,\overline{X_4}\,X_5$$

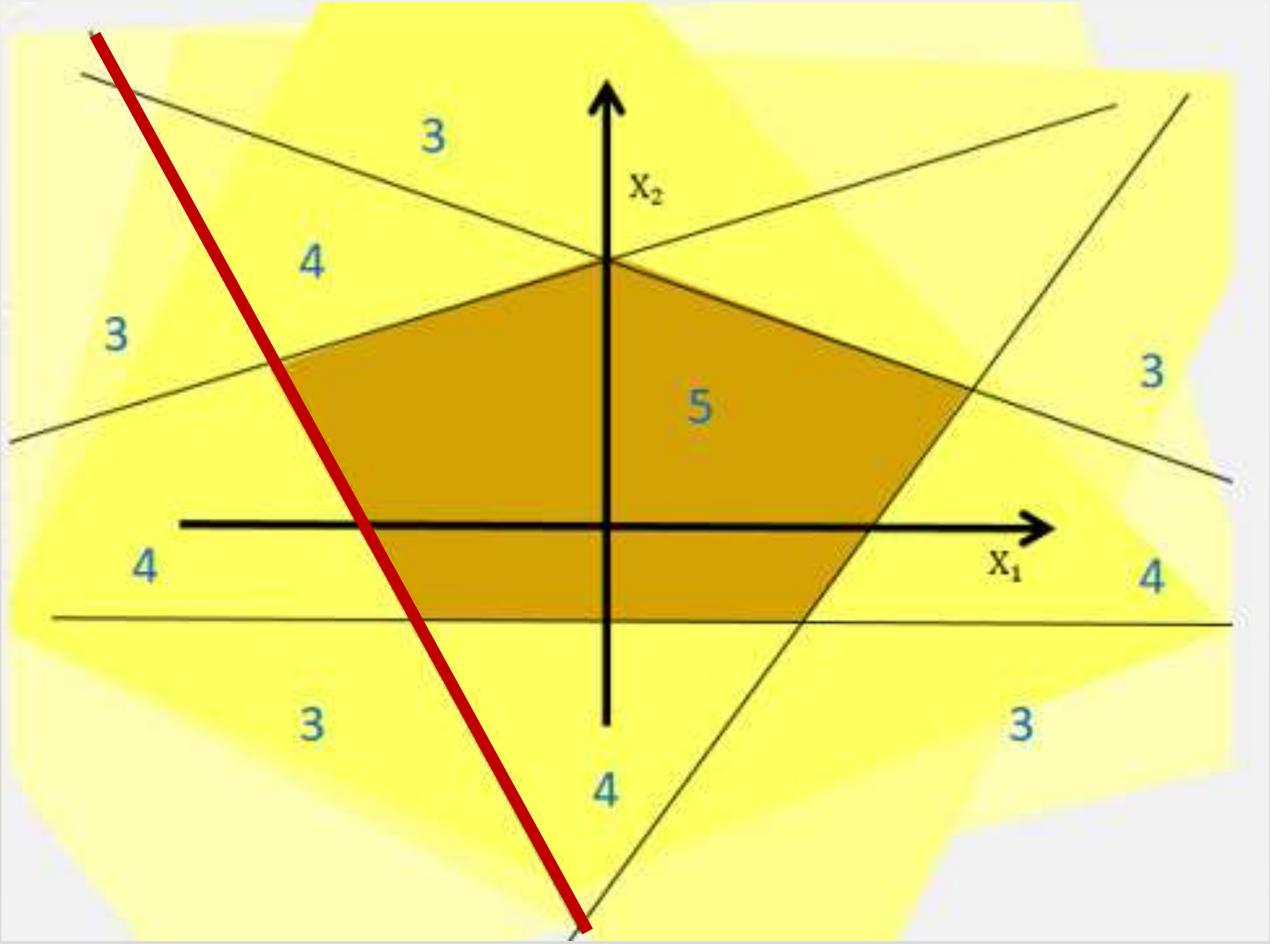
Two-layer MLP is a Universal Boolean Function



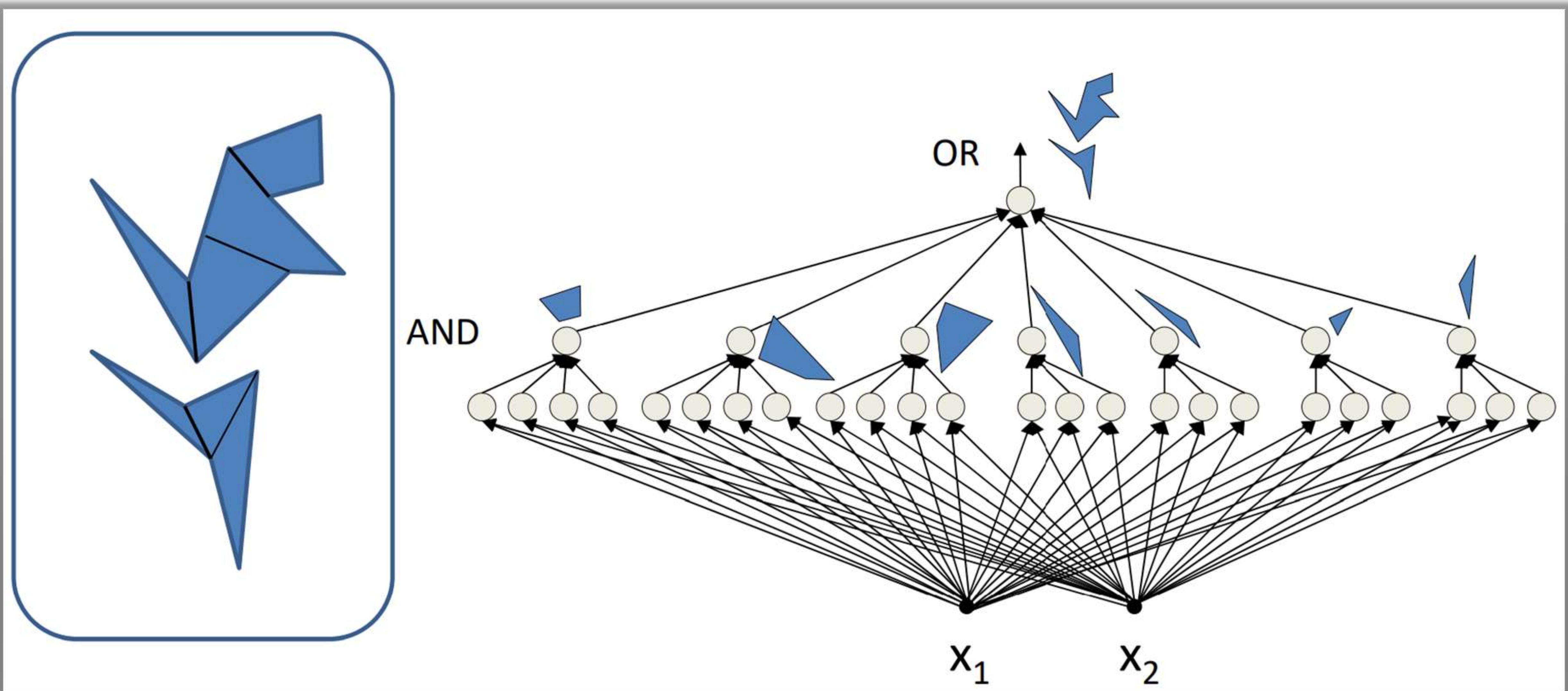
Universal classifiers

Booleans over the reals



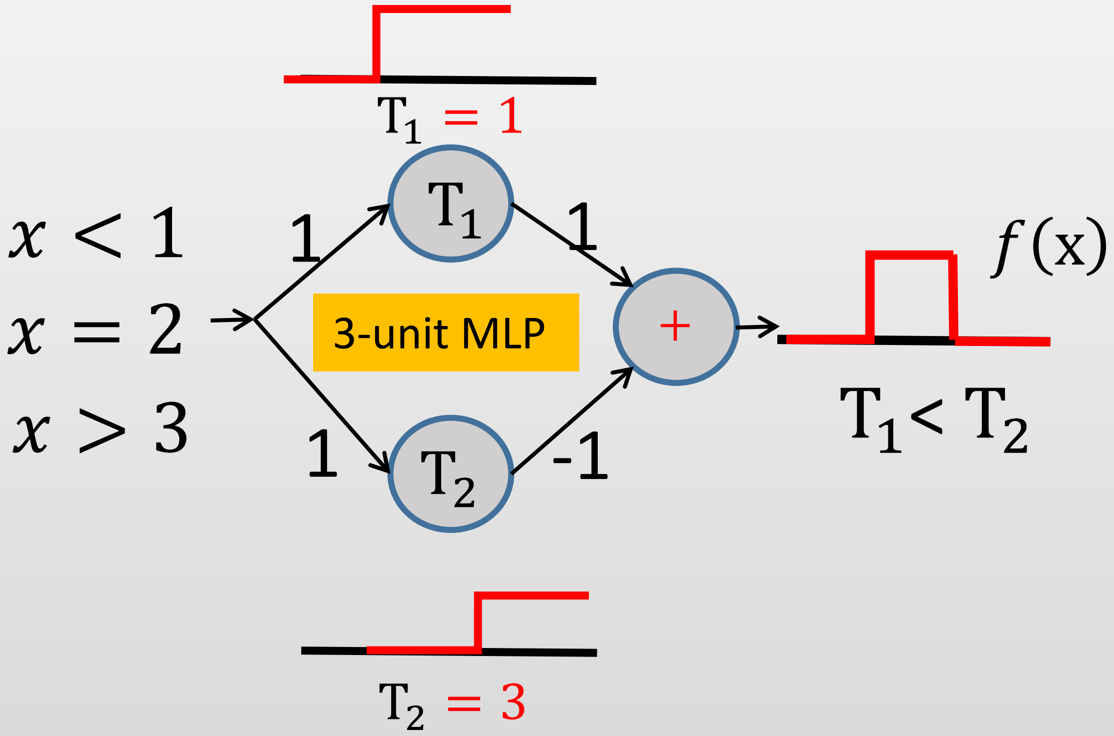


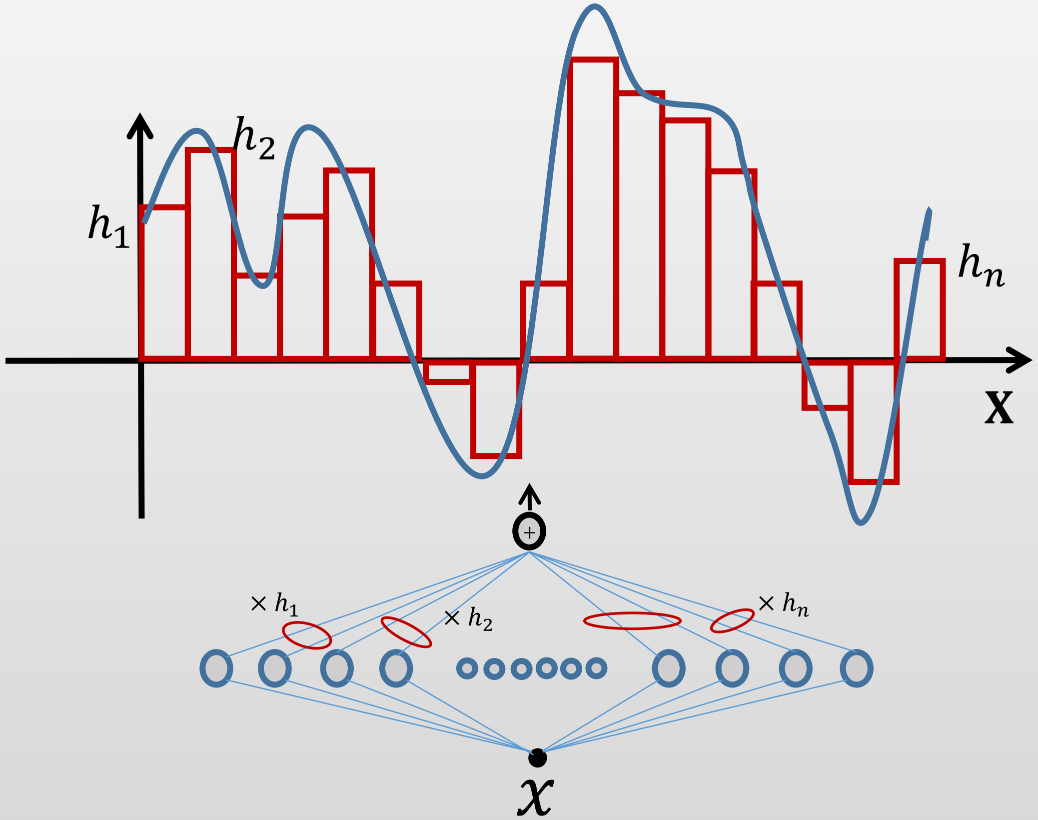
Booleans over the reals



Universal Continuous Value Approximator

MLP as a continuous-valued regression





- A simple 3-unit MLP can generate a "square pulse" over an input
- An MLP with many units can model an arbitrary function over an input
  - To arbitrary precision Simply make the individual pulses narrower
- A one-layer MLP can model an arbitrary function of a single input

4.2 Multilayer Feedforward Neural Networks

损失函数
（优化目标）

模型预测结果　任务模型　数据驱动学习任务模型

$$\hat{y} \leftarrow f(\mathbf{x}', \mathbf{w})$$

‘好’的特征表示 $\mathbf{x}' \leftarrow \phi(\mathbf{x}, \theta)$

监督信息

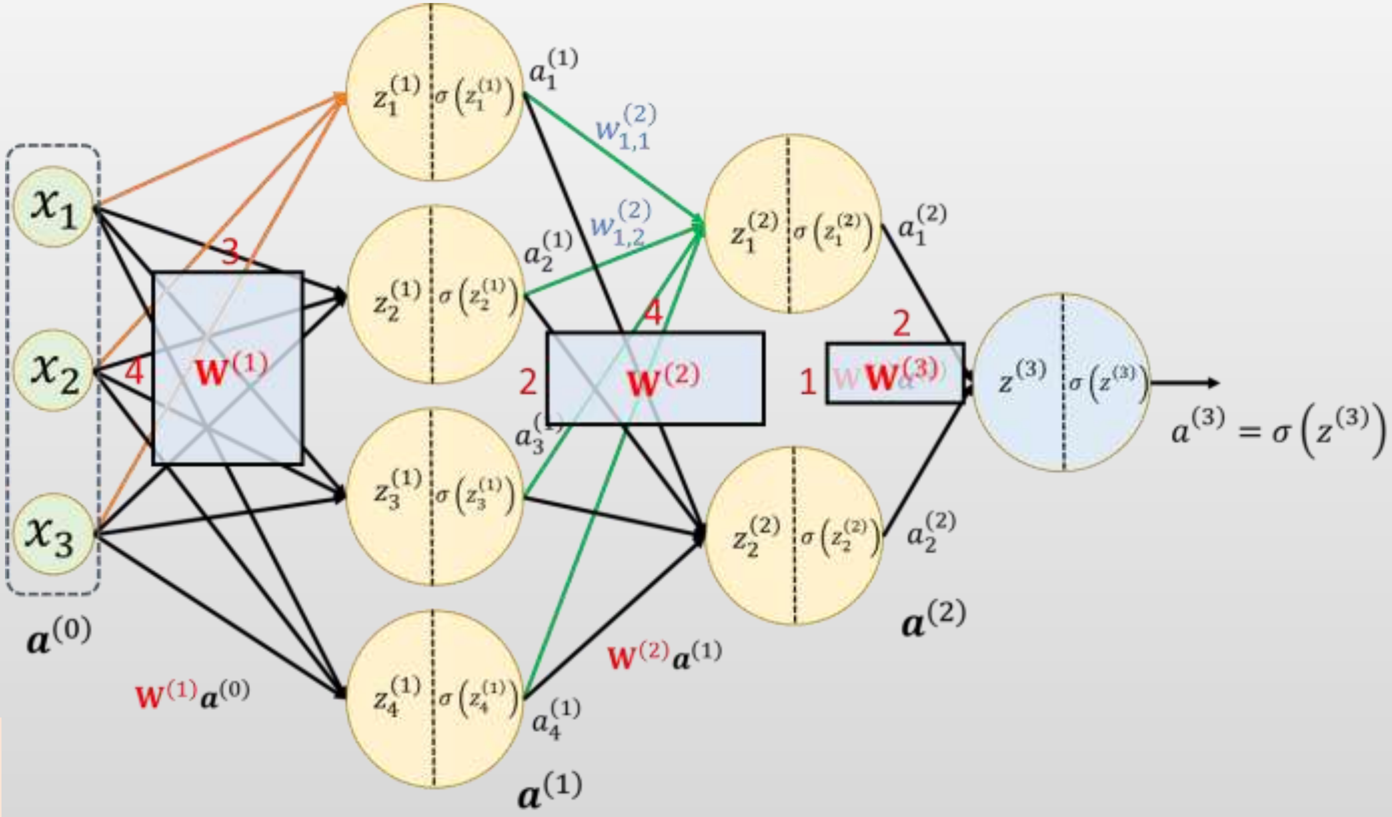
基于NN的特征变换
（表示学习）

数据驱动自动学习特征变换？
（BP反向传播）

Feedforward Calculation

- NN的前向计算过程

$$\boldsymbol{a}^{(0)} = \mathbf{x}$$



线性和: $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\boldsymbol{a}^{(l-1)}$

非线性激活输出:
$\boldsymbol{a}^{(l)} = \sigma\left(\mathbf{z}^{(l)}\right) = \sigma\left(\mathbf{W}^{(l)}\boldsymbol{a}^{(l-1)}\right)$

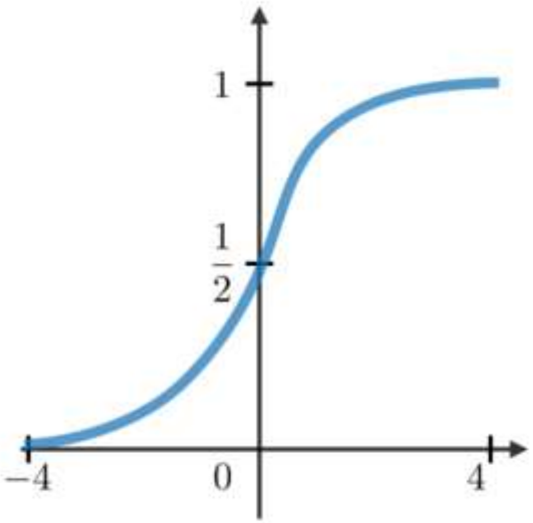
$$\hat{y}_n = f\left(\sigma\left(\mathbf{W}^{(3)}\sigma\left(\mathbf{W}^{(2)}\sigma\left(\mathbf{W}^{(1)}\mathbf{x}_n\right)\right)\right), \phi\right)$$

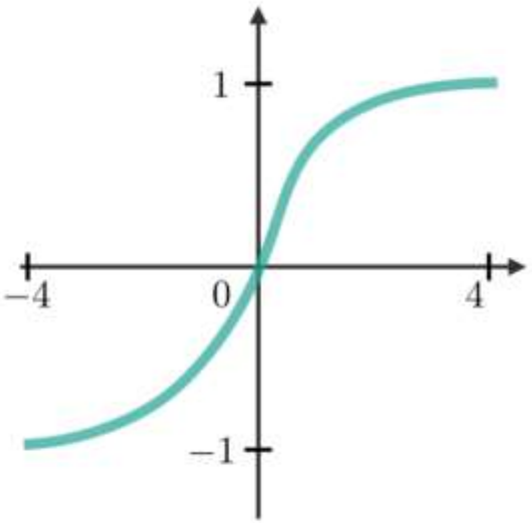
- 损失函数:

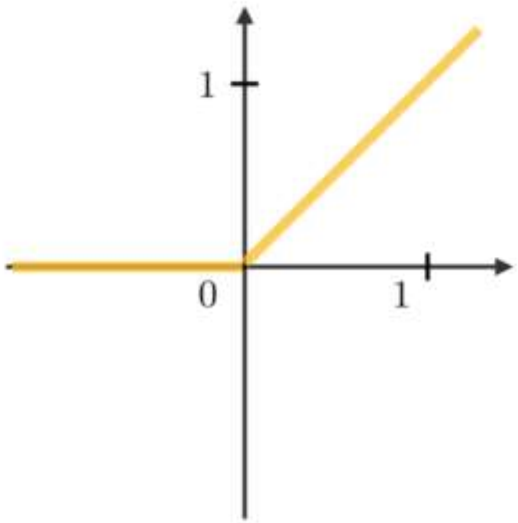
$$\mathcal{L}(\mathbf{W}, \phi) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, \hat{y}_n) + \lambda\|\mathbf{W}\|_F^2$$

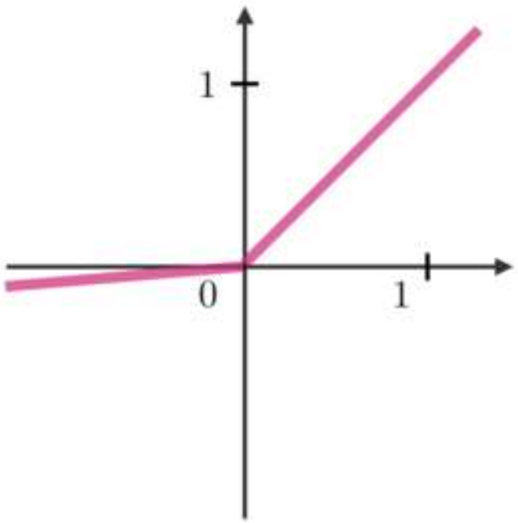
Activation Function

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$









[CS 229 - Deep Learning Cheatsheet (stanford.edu)]

Why are Hidden Layers Nonlinear?

- A multi-layer network that uses only the identity activation function in all its layers reduces to a single-layer network that performs linear regression.

$$\begin{aligned}
&\overline{h}_1 = \Phi(W_1^T \overline{x}) = W_1^T \overline{x} \\
&\overline{h}_{p+1} = \Phi(W_{p+1}^T \overline{h}_p) = W_{p+1}^T \overline{h}_p \quad \forall p \in \{1 \ldots k-1\} \\
&\overline{o} = \Phi(W_{k+1}^T \overline{h}_k) = W_{k+1}^T \overline{h}_k
\end{aligned}$$

- We can eliminate the hidden variable to get a simple linear relationship:

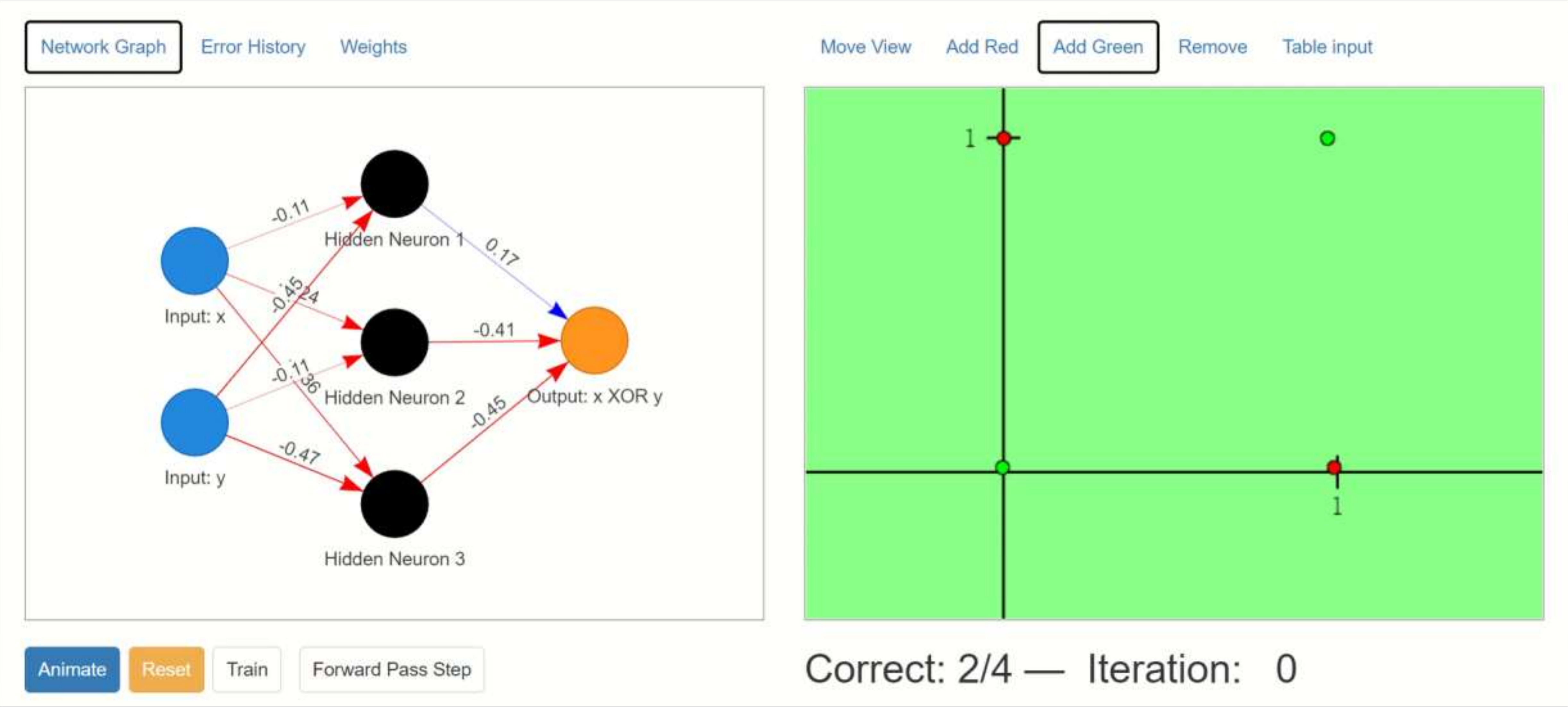
$$\begin{aligned}
\overline{o} &= W_{k+1}^T W_k^T \ldots W_1^T \overline{x} \\
&= \underbrace{(W_1 W_2 \ldots W_{k+1})^T}_{W_{xo}^T} \overline{x}
\end{aligned}$$

- We get a *single-layer* network with matrix W_{xo}.

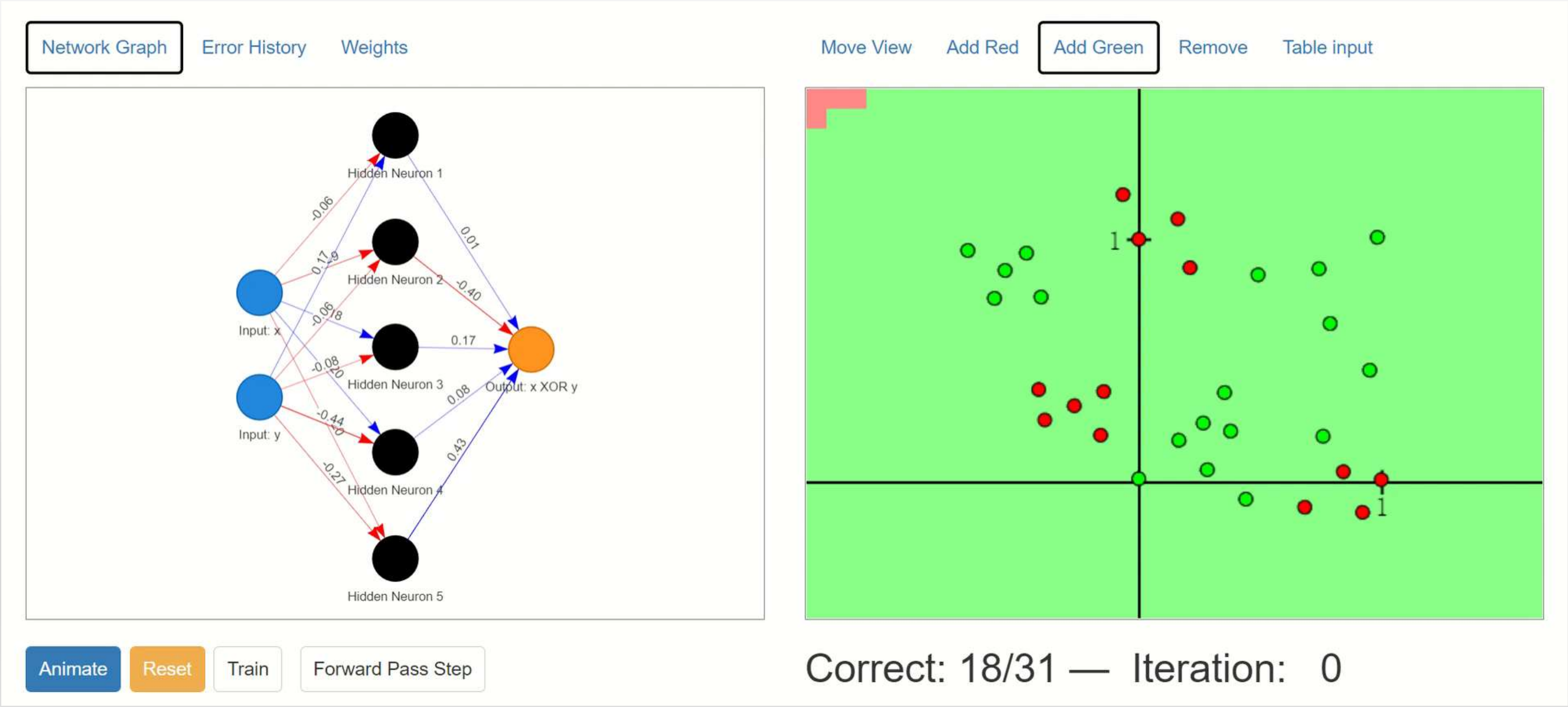
Neural Network Demo

Neural Network Demo

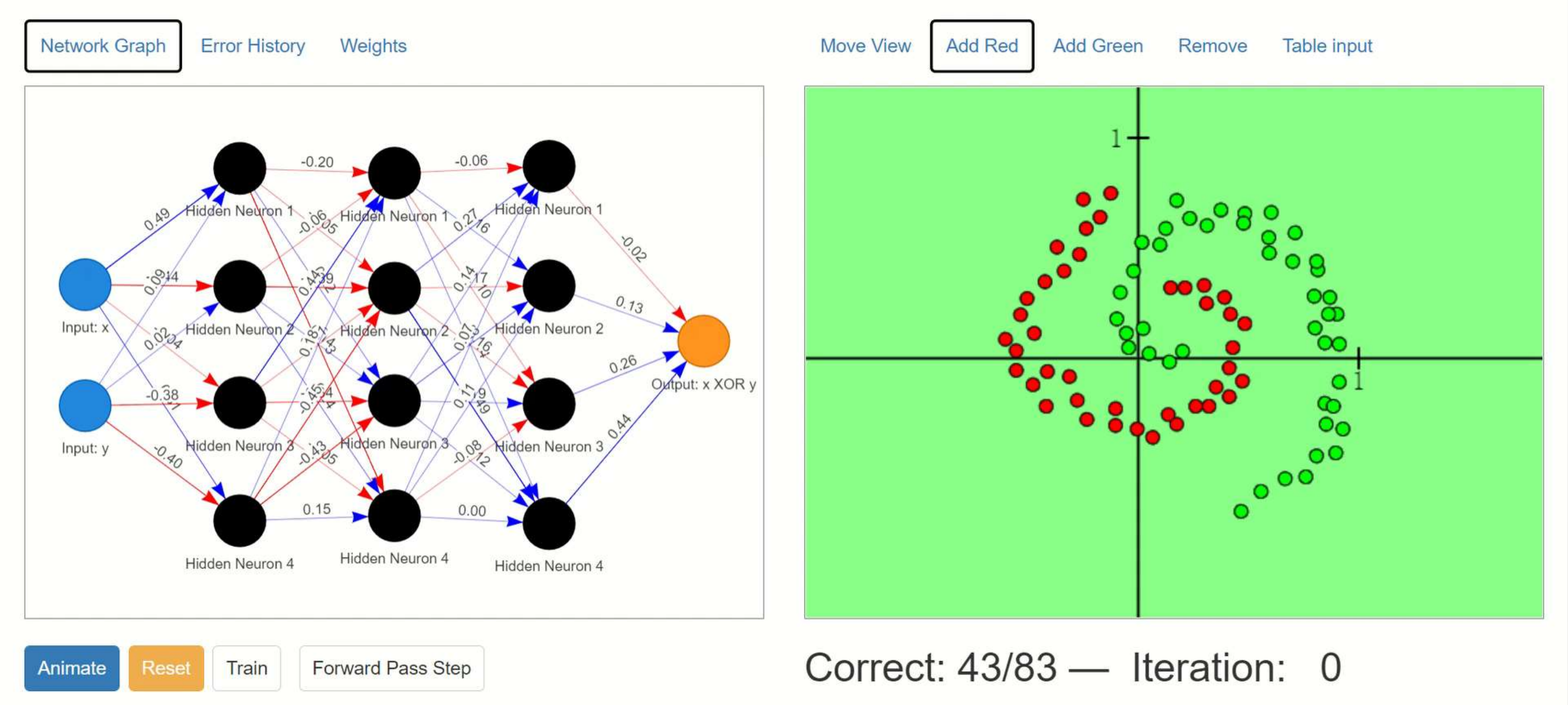
Neural Network Demo



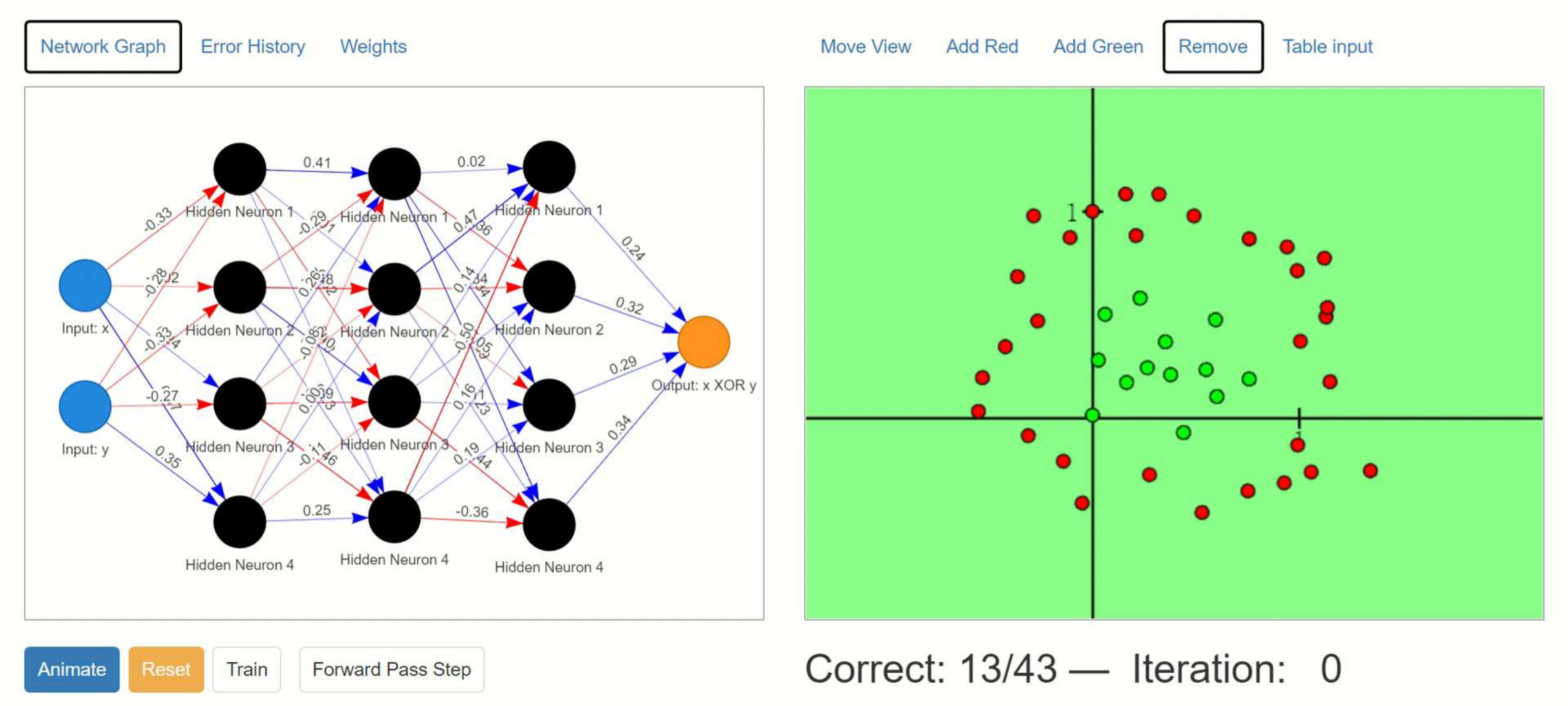
Neural Network Demo



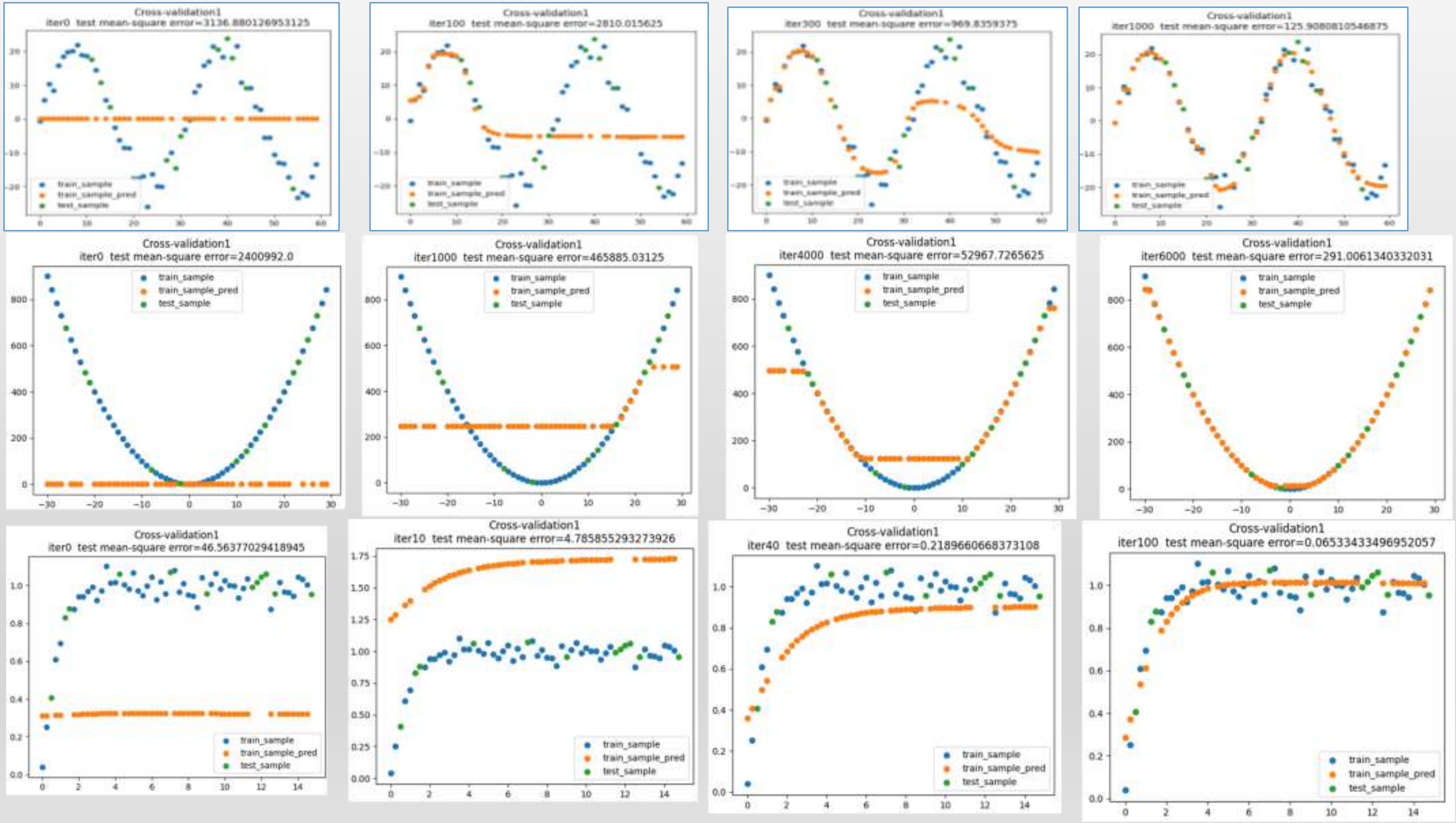
Neural Network Demo



Dot Product

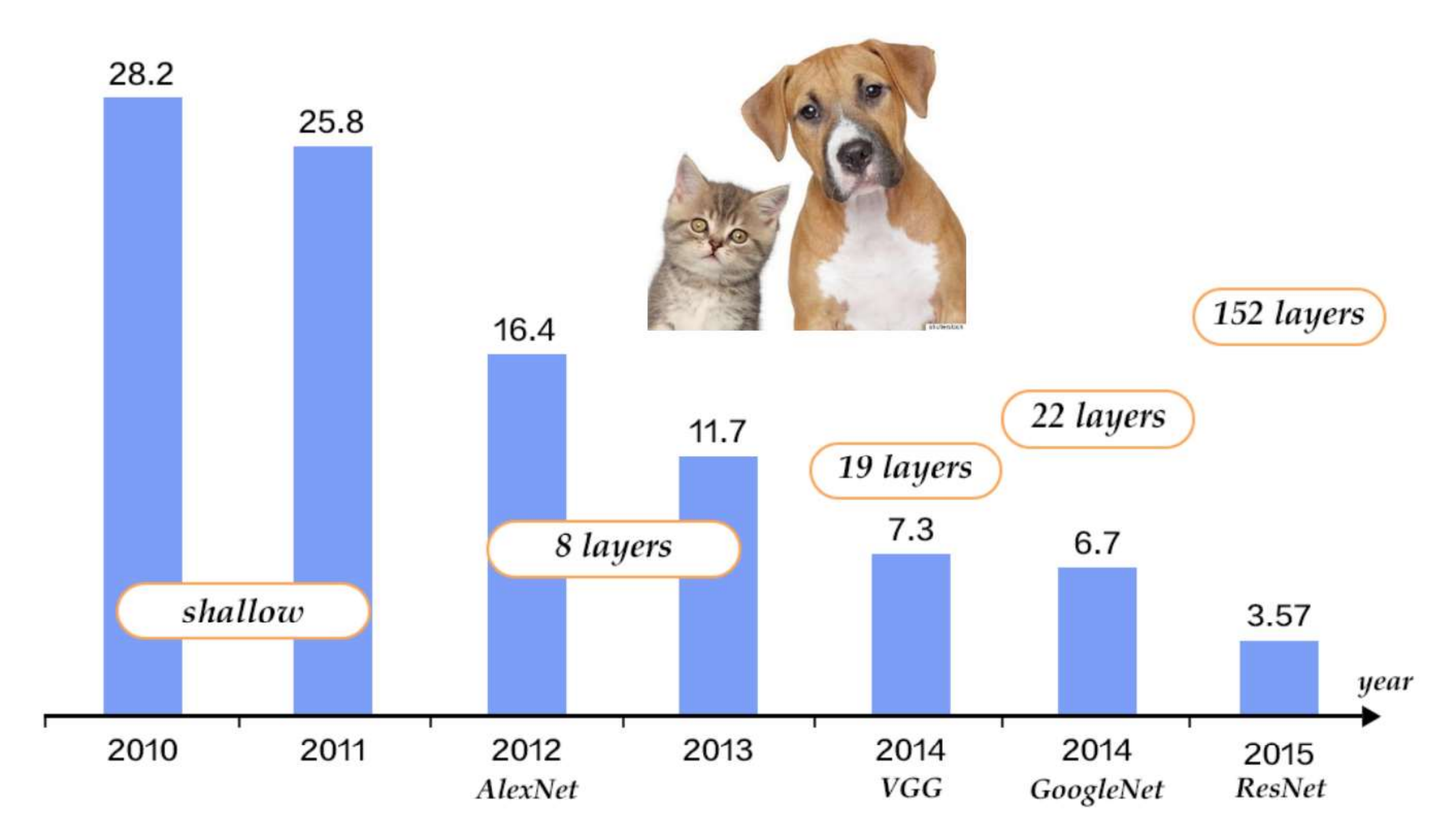

Network Graph
Error History
Weights
Move View
Add Red
Add Green
Remove
Table input
Input: x
Input: y
Hidden Neuron 1
Hidden Neuron 2
Hidden Neuron 3
Hidden Neuron 4
Output: x XOR y
Animate
Reset
Train
Forward Pass Step
Correct: 13/43 — Iteration: 0

Neural Network Regression Demo



Why deeper ?

ILSVRC (ImageNet Large Scale Visual Recognition Challenge) Winners



4.3 Learning with Backpropagation

The Basic Framework of NN Training

Loss Optimization

We want to find the network weights that achieve the lowest loss

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \mathbf{W}\right), y^{(i)}\right)$$

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \, \mathcal{L}(\mathbf{W})$$

$\uparrow$

Remember: $\mathbf{W} = \left\{\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots\right\}$

Gradient Update Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient $\frac{\partial \mathcal{L}(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights

The Computational Graph: $y = (x_1 + x_2)\max(x_2, x_3)$

$y = fg$

$f = x_1 + x_2$

$g = \max(x_2, x_3)$

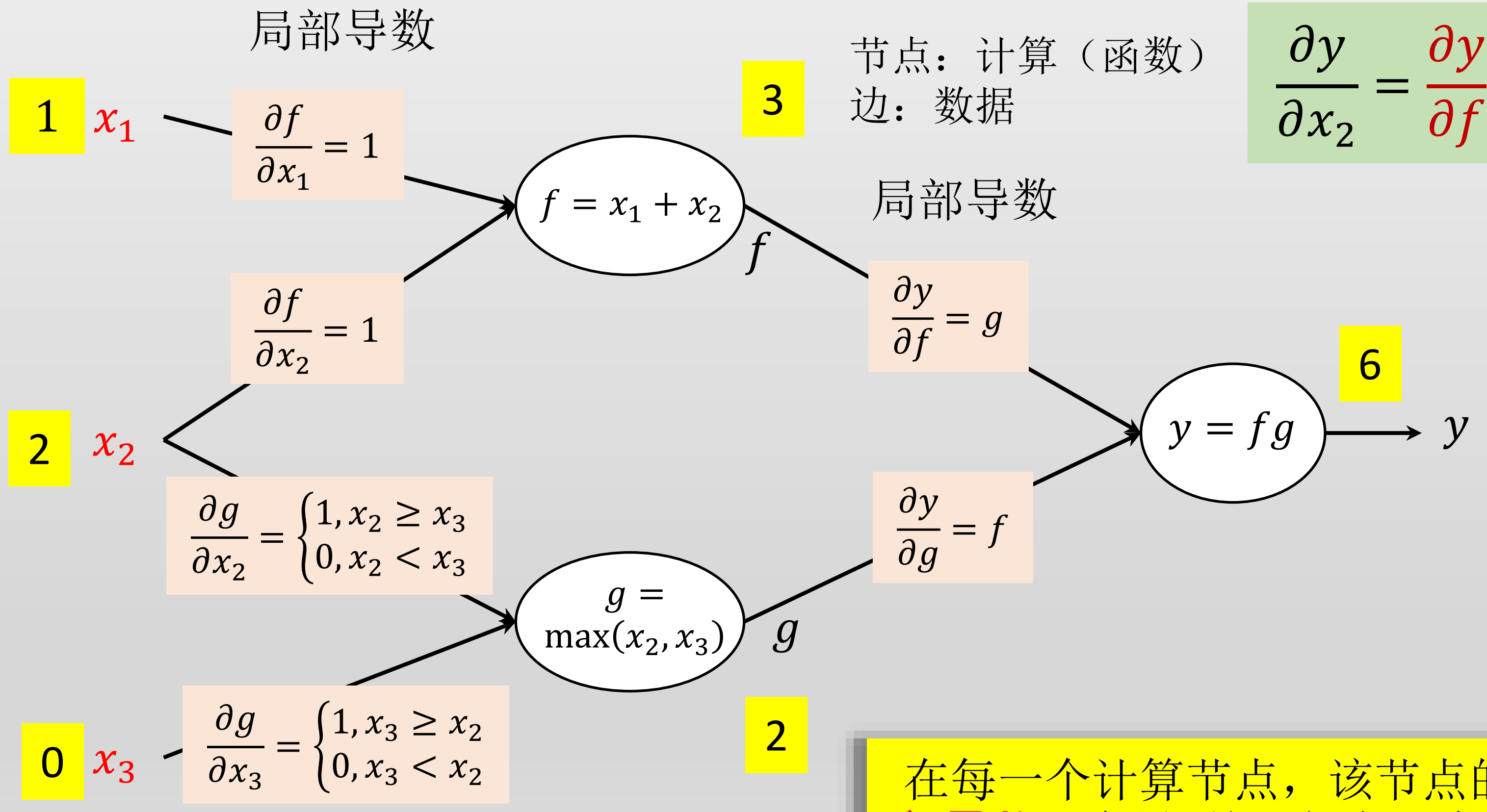
计算图中包括
3个节点

链式公式（沿计算图反向路径）

$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial x_1} = g * 1$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial x_2} + \frac{\partial y}{\partial g}\frac{\partial g}{\partial x_2} = g * 1 + f * \begin{cases} 1, x_2 \geq x_3 \\ 0, x_2 < x_3 \end{cases}$$

$$\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial g}\frac{\partial g}{\partial x_3} = f * \begin{cases} 1, x_3 \geq x_2 \\ 0, x_3 < x_2 \end{cases}$$



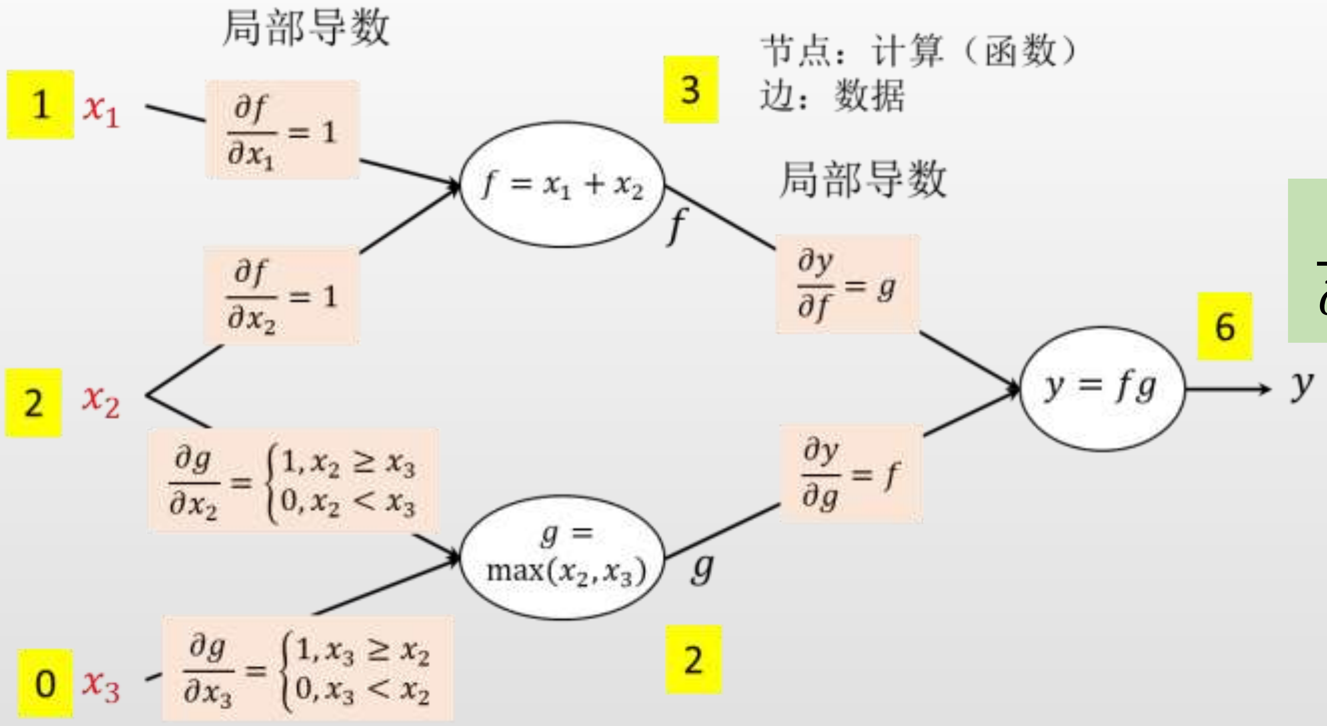
求偏导数的结果（**与当前输入有关**）

$$\frac{\partial y}{\partial x_1} = g1 = 2$$

$$\frac{\partial y}{\partial x_3} = f\begin{cases} 1, x_3 \geq x_2 \\ 0, x_3 < x_2 \end{cases} = 2 * 0 = 0$$

$$\frac{\partial y}{\partial x_2} = g1 + f\begin{cases} 1, x_2 \geq x_3 \\ 0, x_2 < x_3 \end{cases} = 2 + 3 * 1 = 5$$

在每一个计算节点，该节点的输出关于该节点每一个输入均存在一个**局部导数**。变量y关于任意一个变量x的梯度 = 计算图上从y出发，沿计算图反向到达x的所有可能路径（局部导数乘积）之和



$$\frac{\partial y}{\partial x_1}=\frac{\partial y}{\partial f}\frac{\partial f}{\partial x_1}=g*1$$

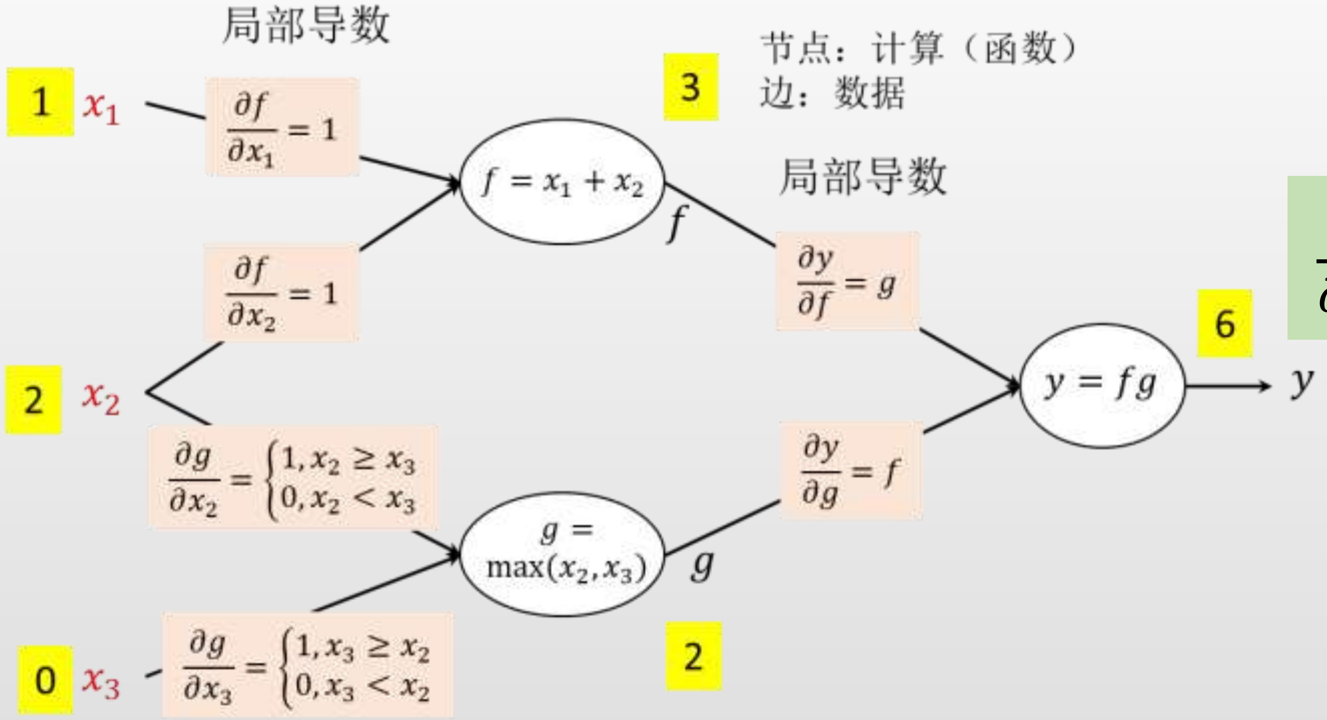
$$\frac{\partial y}{\partial x_2}=\frac{\partial y}{\partial f}\frac{\partial f}{\partial x_2}+\frac{\partial y}{\partial g}\frac{\partial g}{\partial x_2}=g*1+f*\begin{cases}1, x_2 \ge x_3\\0, x_2 < x_3\end{cases}$$

$$\frac{\partial y}{\partial x_3}=\frac{\partial y}{\partial g}\frac{\partial g}{\partial x_3}=f*\begin{cases}1, x_3 \ge x_2\\0, x_3 < x_2\end{cases}$$

（1）**局部导数（简单）**：在每一个计算节点，该节点的输出关于该节点每一个输入均可计算一个简单的**局部导数**。

（2）**链式规则（复合函数）**：变量y关于任意一个变量x的梯度 = 计算图上从y出发，沿计算图反向到达x的所有可能路径（局部导数乘积）之和

（3）**梯度计算依赖于当前输入（需要先计算前向传播）**：每一个计算图中节点的局部导数的计算与该节点当前输入有关，因此，在求梯度前需要先完成前向传播计算，并将计算结果保存下来，例如$\frac{\partial y}{\partial f}=g$，需要知道当前输入下g的值，计算$\frac{\partial y}{\partial x_3}=f*\begin{cases}1, x_3 \ge x_2\\0, x_3 < x_2\end{cases}$需要知道$f$ 以及x_3, x_2的值。



$$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial x_1} = g * 1$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial x_2} + \frac{\partial y}{\partial g}\frac{\partial g}{\partial x_2} = g * 1 + f * \begin{cases} 1, x_2 \geq x_3 \\ 0, x_2 < x_3 \end{cases}$$

$$\frac{\partial y}{\partial x_3} = \frac{\partial y}{\partial g}\frac{\partial g}{\partial x_3} = f * \begin{cases} 1, x_3 \geq x_2 \\ 0, x_3 < x_2 \end{cases}$$

矩阵表示

$$\begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \frac{\partial y}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & 0 \\ \frac{\partial f}{\partial x_2} & \frac{\partial g}{\partial x_2} \\ 0 & \frac{\partial g}{\partial x_3} \end{bmatrix} \begin{bmatrix} \frac{\partial y}{\partial f} \\ \frac{\partial y}{\partial g} \end{bmatrix} = \begin{bmatrix} \frac{\partial y}{\partial f}\frac{\partial f}{\partial x_1} \\ \frac{\partial y}{\partial f}\frac{\partial f}{\partial x_2} + \frac{\partial y}{\partial g}\frac{\partial g}{\partial x_2} \\ \frac{\partial y}{\partial g}\frac{\partial g}{\partial x_3} \end{bmatrix}$$

```python
class Computational_Graph:
    def __init__(self) -> None:
        #初始化x
        self.x1=0
        self.x2=0
        self.x3=0
        #正向传播参数
        self.f=0 #初始化f的值
        self.g=0 #初始化g的值
        self.y=0 #初始化y的值
        #反向传播参数
        self.y_to_f=0 #初始化反向传播中，y关于f的局部导数
        self.y_to_g=0 #初始化反向传播中，y关于g的局部导数
        self.f_to_x1=0 #初始化反向传播中，f关于x1的局部导数
        self.f_to_x2=0 #初始化反向传播中，f关于x2的局部导数
        self.g_to_x2=0 #初始化反向传播中，g关于x2的局部导数
        self.g_to_x3=0 #初始化反向传播中，g关于x3的局部导数
    def function_f(self,x1,x2):
        return x1+x2
    def function_g(self,x2,x3):
        return max(x2,x3)
    def function_y(self,f,g):
        return f*g
```

```python
def forward(self,x1,x2,x3):
    #记录输入的x
    self.x1=x1
    self.x2=x2
    self.x3=x3
    #计算f和g
    self.f=self.function_f(x1,x2)
    print('f : ',self.f)
    self.g=self.function_g(x2,x3)
    print('g : ',self.g)
    #计算y
    self.y=self.function_y(self.f,self.g)
    print('y : ',self.y)
    return self.y
```

```python
x1=1
x2=2
x3=0
net=Computational_Graph()
net.forward(x1,x2,x3)
net.backward()
```

f : 3
g : 2
y : 6
y关于f，g的导数为：2，3
y关于x1，x2，x3的导数为：2，5，0

```python
def backward(self):
    #通过前向传播的记录获取f和g的值
    self.y_to_f=self.g #y关于f的局部导数
    self.y_to_g=self.f #y关于g的局部导数

    #f关于x1的局部导数
    self.f_to_x1=1

    #x2的局部导数
    self.f_to_x2=1 #f关于x2的局部导数
    #g关于x2的局部导数
    if self.x2>=self.x3:
        self.g_to_x2=1
    else:
        self.g_to_x2=0

    #g关于x3的局部导数
    if self.x3>=self.x2:
        self.g_to_x3=1
    else:
        self.g_to_x3=0

    #y关于x1，x2，x3的导数
    self.y_to_x1=self.y_to_f*self.f_to_x1
    self.y_to_x2=self.y_to_f*self.f_to_x2 + self.y_to_g*self.g_to_x2
    self.y_to_x3=self.y_to_g*self.g_to_x3
```

```python
import torch
```

```python
#定义网络结构
class Net(torch.nn.Module):
    def forward(self,x):
        #定义f函数 f=x1+x2
        f=x[0]+x[1]

        #定义g函数 g=max(x1, x2)
        g=max(x[1],x[2]).view(1)

        #计算y y=f×g
        y=f*g

        return y
```

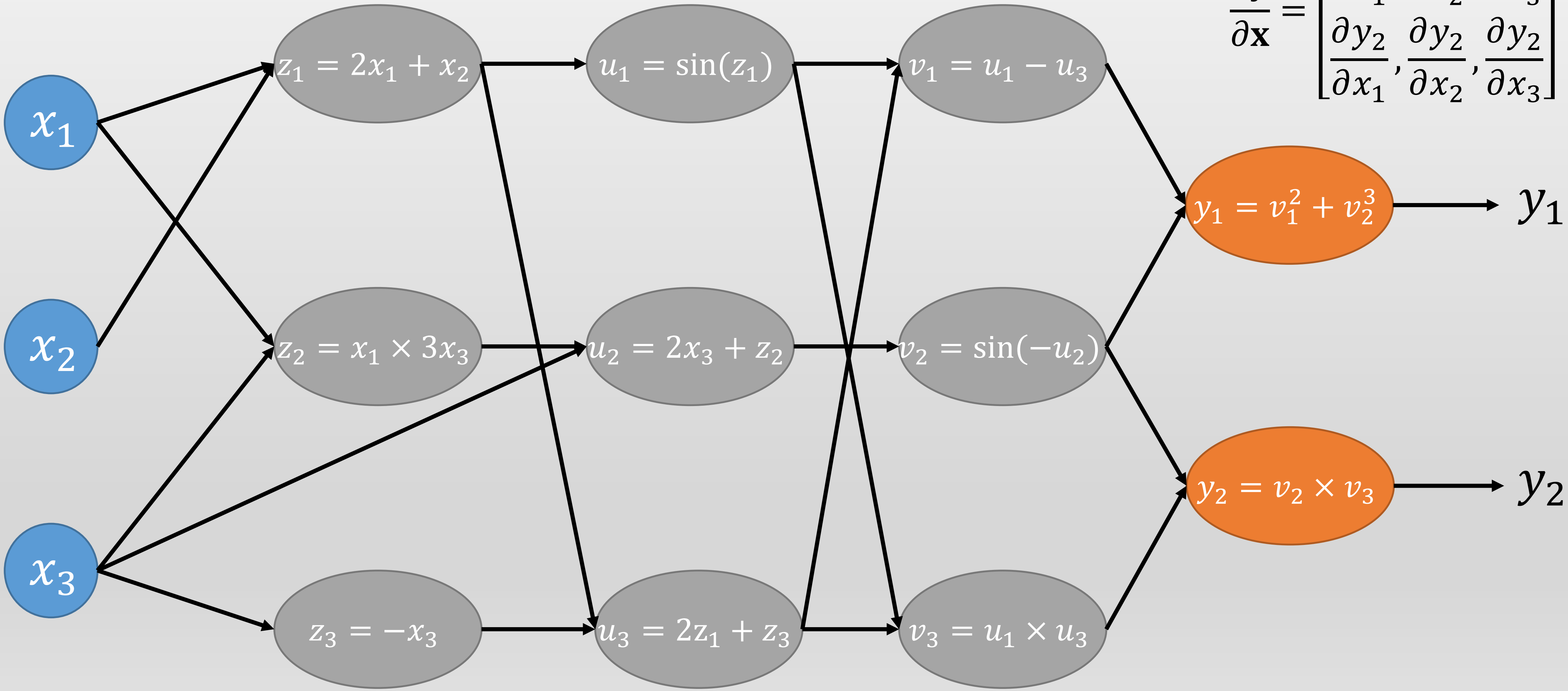
```python
x = torch.tensor([1,2,0],dtype=float,requires_grad=True)
model=Net()
out=model(x)
out.backward()
#y关于x的导数
print(x.grad)
```

```
tensor([2., 5., 0.], dtype=torch.float64)
```

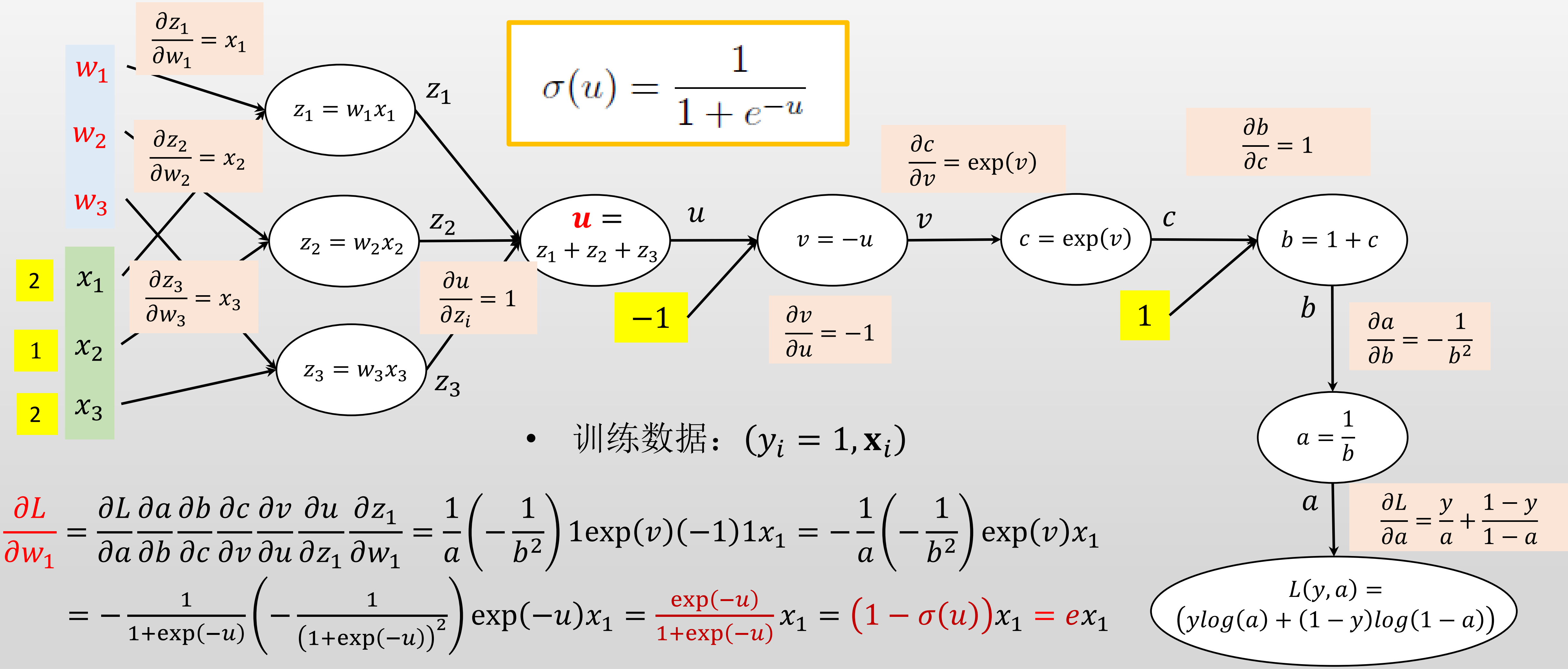
Computational Graph of Function

下面给出了复合函数$f(x_1, x_2, x_3)$的计算图

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \frac{\partial y_1}{\partial x_3} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \frac{\partial y_2}{\partial x_3} \end{bmatrix}$$



Computational Graph of Sigmoid Function



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

- 训练数据：$(y_i = 1, \mathbf{x}_i)$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial b}\frac{\partial b}{\partial c}\frac{\partial c}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial z_1}\frac{\partial z_1}{\partial w_1} = \frac{1}{a}\left(-\frac{1}{b^2}\right)1\exp(v)(-1)1x_1 = -\frac{1}{a}\left(-\frac{1}{b^2}\right)\exp(v)x_1$$

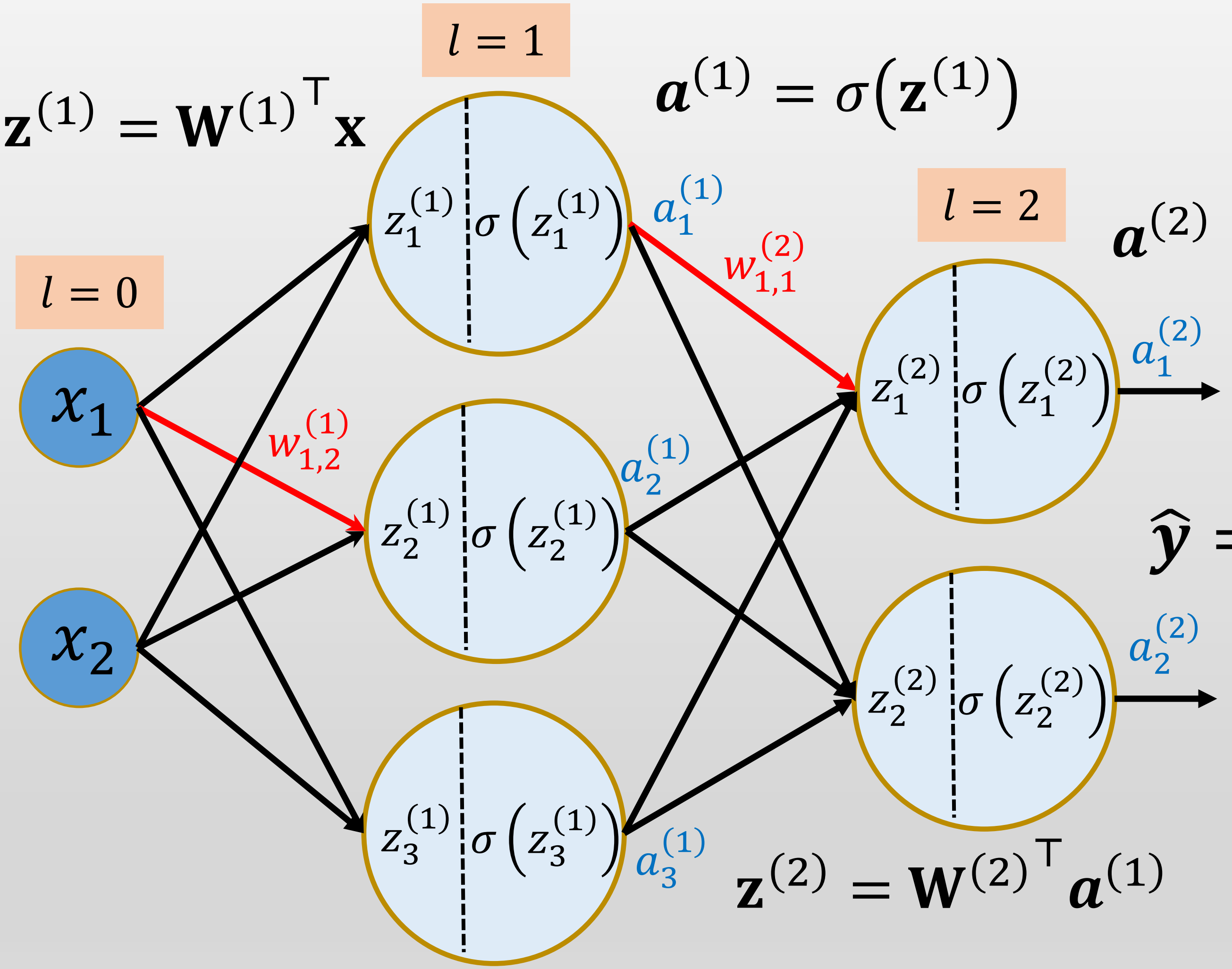
$$= -\frac{1}{1+\exp(-u)}\left(-\frac{1}{(1+\exp(-u))^2}\right)\exp(-u)x_1 = \frac{\exp(-u)}{1+\exp(-u)}x_1 = (1-\sigma(u))x_1 = ex_1$$

$$\frac{\partial L}{\partial w_2} = ex_2;\ \frac{\partial L}{\partial w_3} = ex_3 \Rightarrow \frac{\partial L}{\partial \mathbf{w}} = e\mathbf{x}_i \Rightarrow \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta e\mathbf{x}_i$$

Basic Idea of Backpropagation

$$\mathbf{z}^{(1)} = {\mathbf{W}^{(1)}}^{\top}\mathbf{x}$$

$$\boldsymbol{a}^{(1)} = \sigma\left(\mathbf{z}^{(1)}\right)$$



$$\boldsymbol{a}^{(2)} = \sigma\left(\mathbf{z}^{(2)}\right)$$

$$\hat{\boldsymbol{y}} = \boldsymbol{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix}$$

Loss function:
$L(\boldsymbol{y}, \hat{\boldsymbol{y}})$

$$\hat{\boldsymbol{y}} = \sigma\left({\mathbf{W}^{(2)}}^{\top}\sigma\left({\mathbf{W}^{(1)}}^{\top}\mathbf{x}\mathbf{x}\right)\right)$$

$$\mathbf{z}^{(2)} = {\mathbf{W}^{(2)}}^{\top}\boldsymbol{a}^{(1)}$$

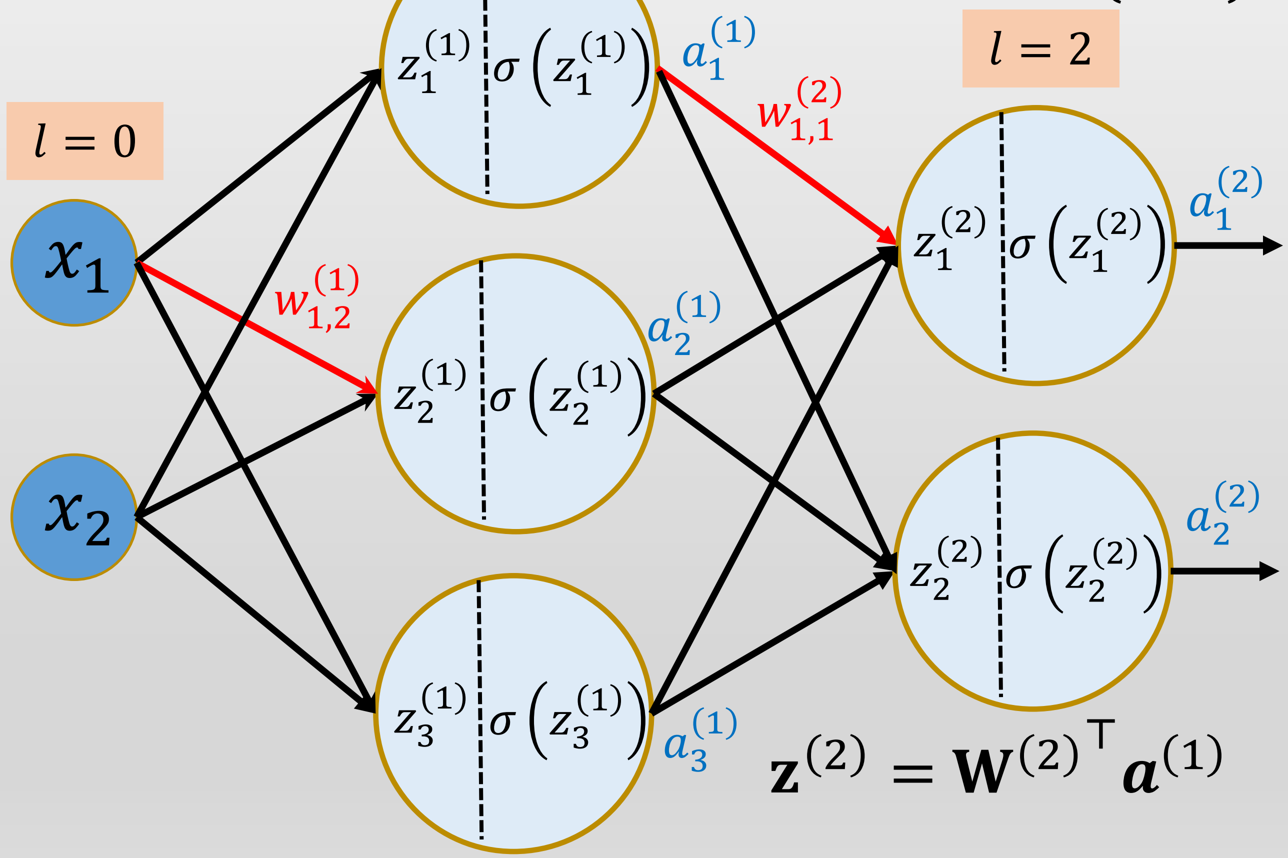
$\mathbf{W}^{(1)} \in \mathbb{R}^{2\times 3}$ $\mathbf{W}^{(2)} \in \mathbb{R}^{3\times 2}$

Basic Idea of Backpropagation

$\boldsymbol{a}^{(1)} = \sigma\left(\mathbf{z}^{(1)}\right)$

$\mathbf{z}^{(1)} = {\mathbf{W}^{(1)}}^{\top}\mathbf{x}$

$\boldsymbol{a}^{(2)} = \sigma\left(\mathbf{z}^{(2)}\right)$



$\mathbf{z}^{(2)} = {\mathbf{W}^{(2)}}^{\top}\boldsymbol{a}^{(1)}$

$\mathbf{W}^{(1)} \in \mathbb{R}^{2\times 3}$ $\mathbf{W}^{(2)} \in \mathbb{R}^{3\times 2}$

Loss function: $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$

目标：对于NN中,任意一条边权重参数$w_{i,j}^{(l)}$，求偏导数$\dfrac{\partial L}{\partial w_{i,j}^{(l)}}$

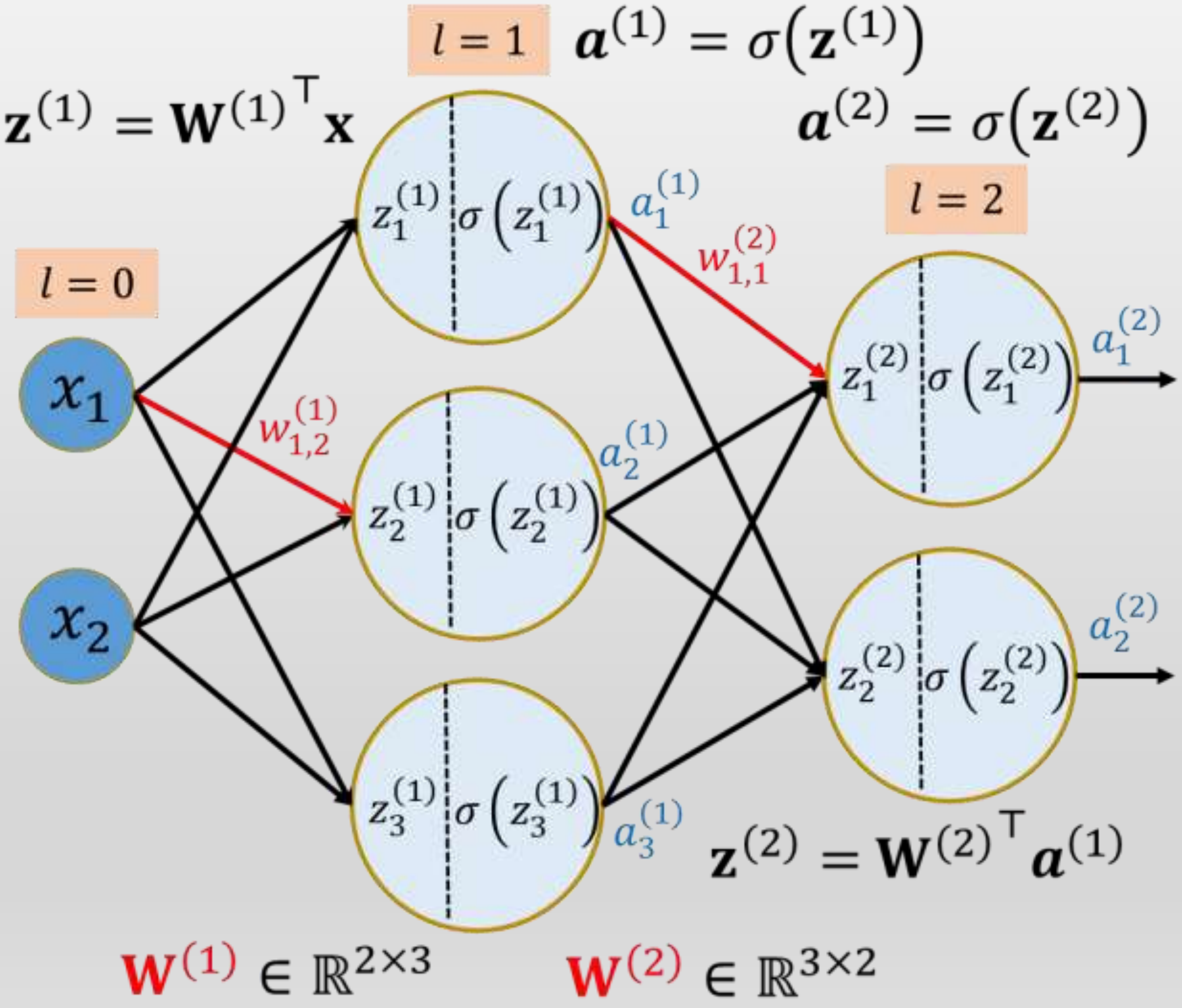
- 记$z_i^{(l)} = {\mathbf{W}_i^{(l)}}^{\top}\mathbf{x}$是NN中第$l$层第$i$个节点的点积。
- 如果知道$\dfrac{\partial L}{\partial z_i^{(l)}}$, 那么

上一层节点 j 的激活输出

$$\frac{\partial L}{\partial w_{i,j}^{(l)}} = \frac{\partial L}{\partial z_i^{(l)}}\frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial L}{\partial z_i^{(l)}} a_j^{(l-1)}$$

- 如何高效求出$\dfrac{\partial L}{\partial z_i^{(l)}}$？？

Basic Idea of Backpropagation



- $$\delta_1^{(2)} \equiv \frac{\partial L}{\partial z_1^{(2)}} = \frac{\partial L}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = L'\left(a_1^{(2)}\right)\sigma'\left(z_1^{(2)}\right)$$

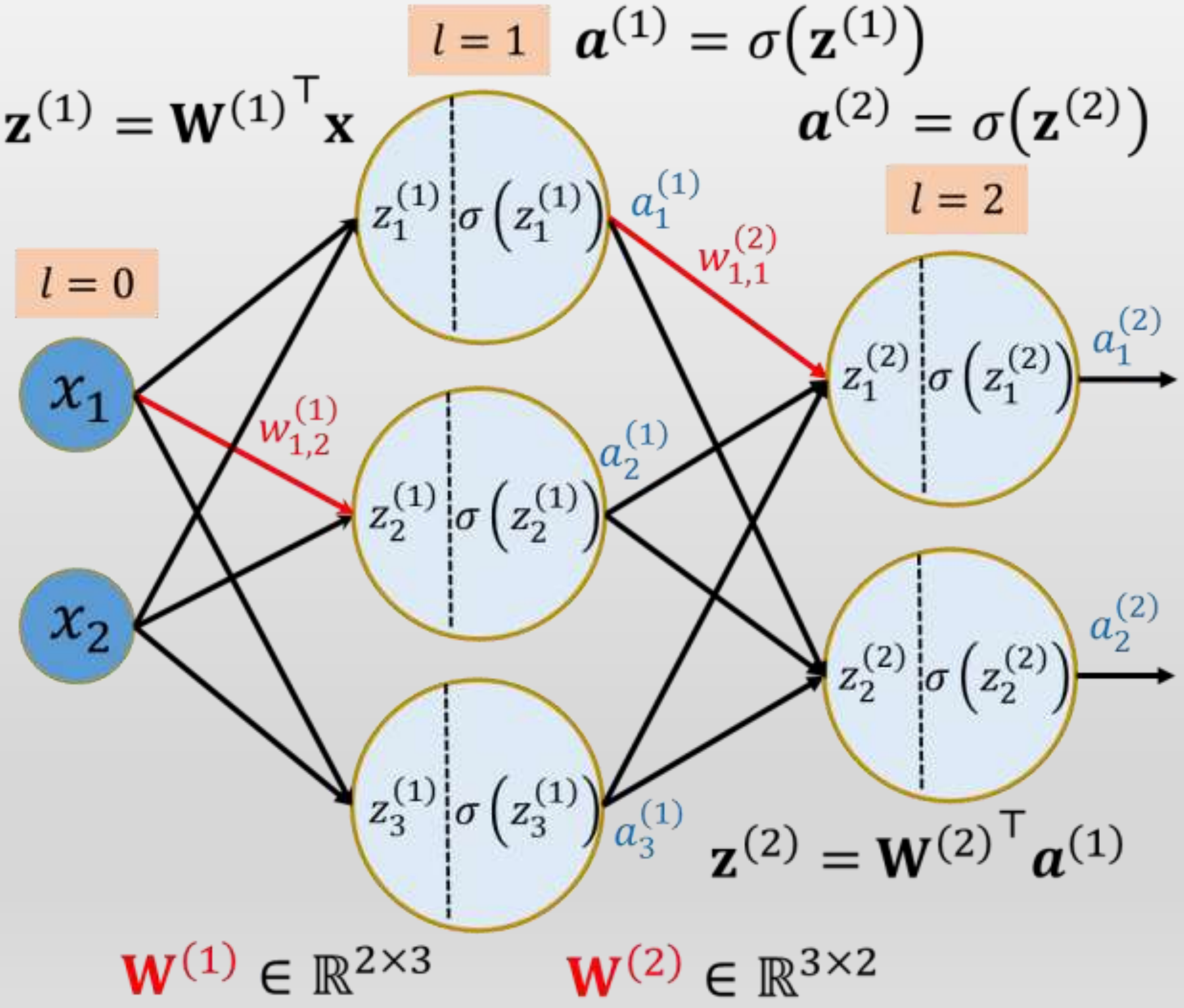
- $$\delta_2^{(2)} \equiv \frac{\partial L}{\partial z_2^{(2)}} = \frac{\partial L}{\partial a_2^{(2)}}\frac{\partial a_2^{(2)}}{\partial z_2^{(2)}} = L'\left(a_1^{(2)}\right)\sigma'\left(z_1^{(2)}\right)$$

当前计算节点输入
（需要通过前向计算获得，并存储）

- $$\delta_i^{(2)} \equiv \frac{\partial L}{\partial z_i^{(2)}} = L'(a_i^{(2)})\sigma'(z_i^{(2)})$$

局部导数
（容易求导）

Basic Idea of Backpropagation

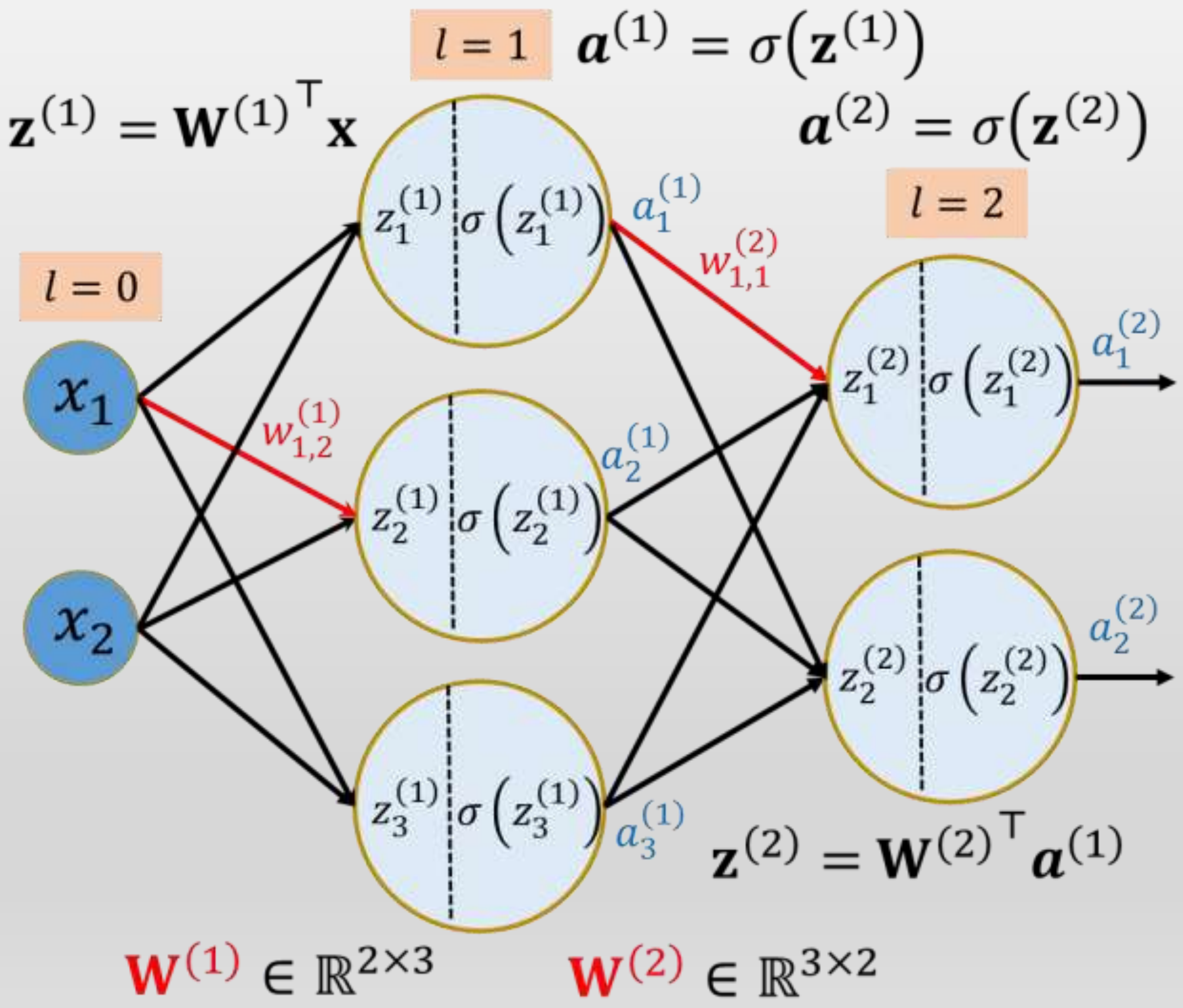


- $$\frac{\partial L}{\partial w_{1,1}^{(2)}} = \frac{\partial L}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(2)}} = \delta_1^{(2)} a_1^{(1)}$$

- $$\frac{\partial L}{\partial w_{i,j}^{(2)}} = \delta_j^{(2)} a_i^{(1)}$$

$$\delta_i^{(2)} = L'\left(a_i^{(2)}\right)\sigma'\left(z_i^{(2)}\right)$$

Basic Idea of Backpropagation



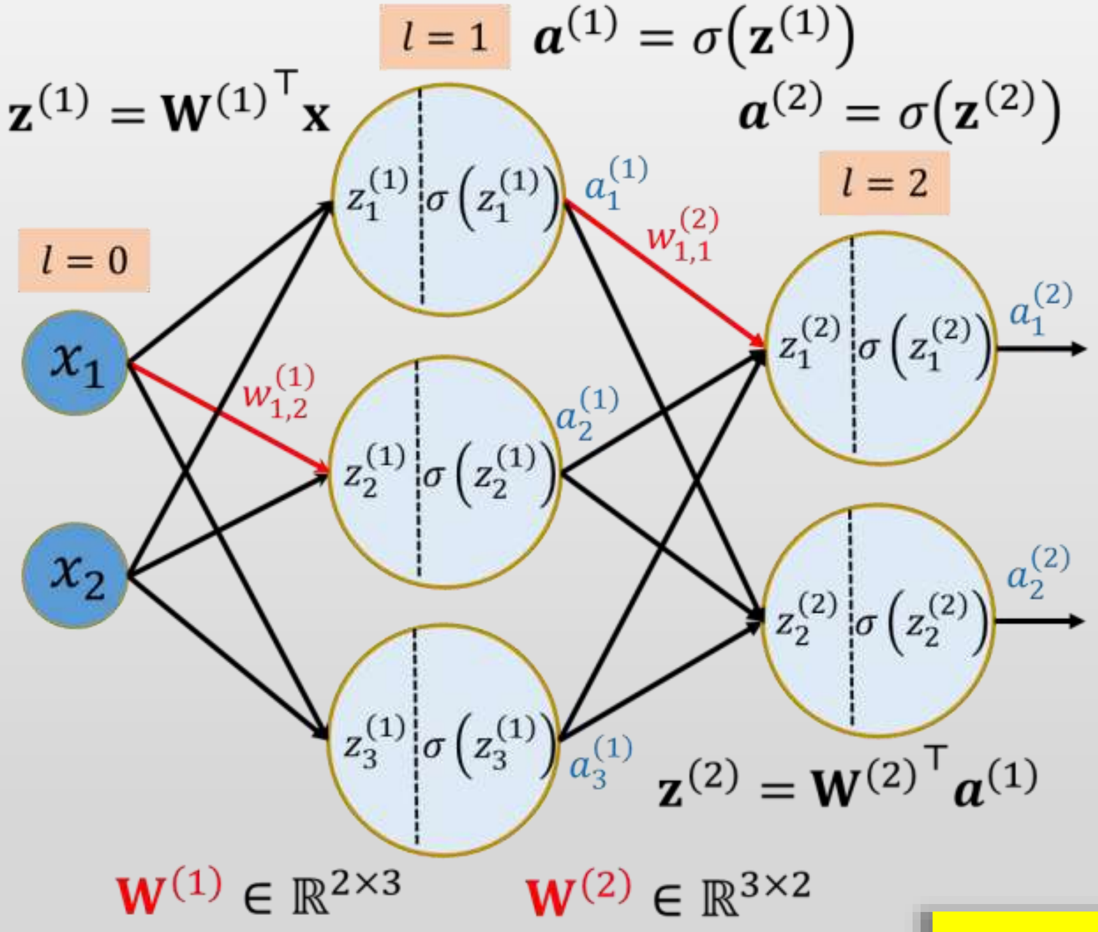
能否用$\delta_1^{(2)}$, $\delta_2^{(2)}$来表示$\delta_1^{(1)}$, $\delta_2^{(1)}$, $\delta_3^{(1)}$ ？

$$\delta_1^{(1)} = \delta_1^{(2)} \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} + \delta_2^{(2)} \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}}$$
$$= \left(\delta_1^{(2)} w_{1,1}^{(2)} + \delta_2^{(2)} w_{1,2}^{(2)}\right) \sigma'\left(z_1^{(1)}\right)$$

$$\delta_2^{(1)} = \left(\delta_1^{(2)} w_{2,1}^{(2)} + \delta_2^{(2)} w_{2,2}^{(2)}\right) \sigma'\left(z_2^{(1)}\right)$$

$$\delta_3^{(1)} = \left(\delta_1^{(2)} w_{3,1}^{(2)} + \delta_2^{(2)} w_{3,2}^{(2)}\right) \sigma'\left(z_3^{(1)}\right)$$

Basic Idea of Backpropagation



能否用$\delta_1^{(2)}$, $\delta_2^{(2)}$来表示$\delta_1^{(1)}$, $\delta_2^{(1)}$, $\delta_3^{(1)}$?

$$\delta_1^{(1)} = \left(\delta_1^{(2)} w_{1,1}^{(2)} + \delta_2^{(2)} w_{1,2}^{(2)}\right) \sigma'\left(z_1^{(1)}\right)$$

$$\delta_2^{(1)} = \left(\delta_1^{(2)} w_{2,1}^{(2)} + \delta_2^{(2)} w_{2,2}^{(2)}\right) \sigma'\left(z_2^{(1)}\right)$$

$$\delta_3^{(1)} = \left(\delta_1^{(2)} w_{3,1}^{(2)} + \delta_2^{(2)} w_{3,2}^{(2)}\right) \sigma'\left(z_3^{(1)}\right)$$

$$\boldsymbol{\delta}^{(1)} = \left(\mathbf{W}^{(2)} \boldsymbol{\delta}^{(2)}\right) \odot \sigma'\left(\mathbf{z}^{(1)}\right)$$

3×1 $\quad$ 3×2 $\quad$ 2×1 $\quad$ 3×1

$$\boldsymbol{\delta}^{(l)} = \left(\mathbf{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}\right) \odot \sigma'\left(\mathbf{z}^{(l)}\right)$$

Basic Idea of Backpropagation

Hadamard 乘积，**s⊙t** 假设**s**和**t**是两个同样维度的向量，那么我们使用**s⊙t**来表示**按元素**的乘积。所以**s⊙t**的元素就是$(\mathbf{s}\odot\mathbf{t})_j = s_j t_j$。

$$\begin{bmatrix}1\\2\end{bmatrix}\odot\begin{bmatrix}3\\4\end{bmatrix}=\begin{bmatrix}1*3\\2*4\end{bmatrix}=\begin{bmatrix}3\\8\end{bmatrix}$$

这种类型的按元素乘法有时候被称为**Hadamard乘积**，或者**Schur乘积**。

- $L=\frac{1}{2}\sum_j\left(y_j-a_j\right)^2$
- $\frac{\partial L}{\partial a_j^L}=\left(a_j-y_j\right)$
- $\delta_j^L=\frac{\partial L}{\partial a_j^L}\sigma'\left(z_j^L\right)$

向量化表示：

$$\boldsymbol{\delta}^L=\nabla_a L\odot\sigma'\left(\mathbf{z}^L\right)$$

$$\boldsymbol{\delta}^L=\left(\boldsymbol{a}^L-\boldsymbol{y}\right)\odot\sigma'\left(\mathbf{z}^L\right)$$

Basic Idea of Backpropagation

总结：反向传播的四个方程式

① $$\boldsymbol{\delta}^L = \nabla_a L \odot \sigma'(\mathbf{z}^L)$$ (BP1)

② $$\boldsymbol{\delta}^l = \left(\mathbf{W}^{l+1}\boldsymbol{\delta}^{l+1}\right) \odot \sigma'(\mathbf{z}^l)$$ (BP2)

③ $$\frac{\partial L}{\partial w_{ij}^l} = \delta_j^l a_i^{l-1}$$ (BP3)

④ $$\frac{\partial L}{\partial b_j^l} = \delta_j^l$$ (BP4)

Basic Idea of Backpropagation

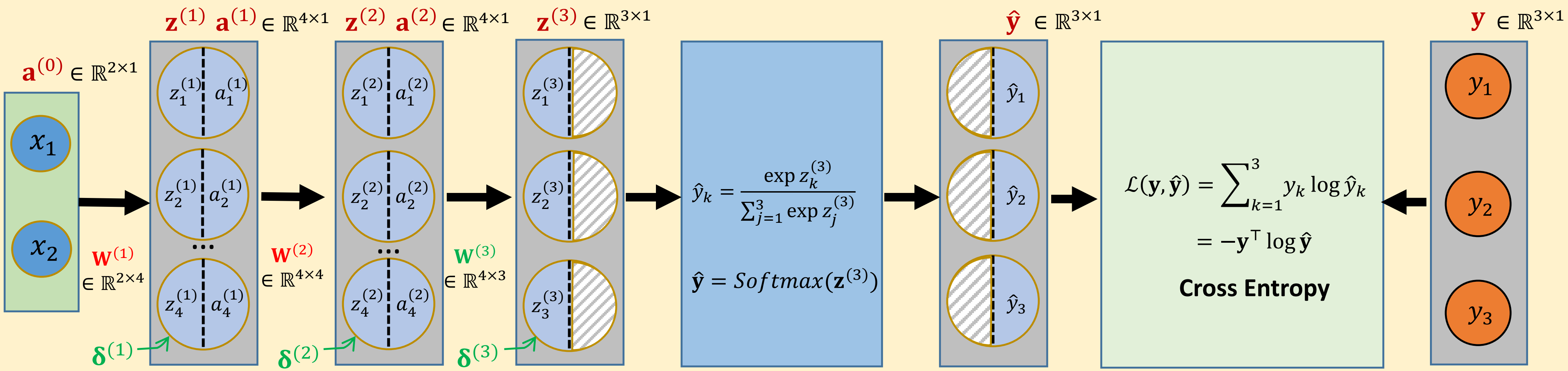
反向传播方程给出了一种计算损失函数**梯度**的方法

1. **输入**x：为输入层设置对应的激活值a^1
2. **前向传播**：对每一个$l = 2, \ldots, L$计算相应的

$$\mathbf{z}^l = \left(\mathbf{W}^l\right)^\top \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \quad \text{和} \quad \boldsymbol{a}^l = \sigma\left(\mathbf{z}^l\right)$$

3. **输出层误差δ^L**：计算向量 $\boldsymbol{\delta}^L = \nabla_a L \odot \sigma'\left(\mathbf{z}^L\right)$
4. **反向误差传播**：对每个$l = L-1, \ldots, 2$, 计算 $\boldsymbol{\delta}^l = \left(\mathbf{W}^{l+1}\boldsymbol{\delta}^{l+1}\right) \odot \sigma'\left(\mathbf{z}^l\right)$
5. **输出**：代价函数的梯度由 $\frac{\partial L}{\partial w_{ji}^l} = a_i^{l-1}\delta_j^l$ 和 $\frac{\partial L}{\partial b_j^l} = \delta_j^l$得出.

FFNN classification with multiple hidden layers



$\mathbf{z}^{(1)} = (\mathbf{W}^{(1)})^\top \mathbf{x};$
$\boldsymbol{a}^{(1)} = \sigma(\mathbf{z}^{(1)})$

$\mathbf{z}^{(2)} = (\mathbf{W}^{(2)})^\top \boldsymbol{a}^{(1)}$
$\boldsymbol{a}^{(2)} = \sigma(\mathbf{z}^{(2)})$

$\mathbf{z}^{(3)} = (\mathbf{W}^{(3)})^\top \boldsymbol{a}^{(2)}$

② $\hat{\mathbf{y}} = \mathrm{Softmax}\,(\mathbf{z}^{(3)})$

① $\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = \sum_{k=1}^{3} y_k \log \hat{y}_k = -\mathbf{y}^\top \log \hat{\mathbf{y}}$

列向量求导“链式法则”

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_r \end{bmatrix}, \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_m \end{bmatrix}$$

$\mathbb{R}^{n\times 1}$ $\mathbb{R}^{r\times 1}$ $\mathbb{R}^{m\times 1}$

- 假定: $\mathbf{z} = g(\mathbf{y}), \mathbf{y} = f(\mathbf{x}),$
  例如: $\mathbf{z} = \mathbf{W}^\top \mathbf{x}, \mathbf{W} \in \mathbb{R}^{n\times m}$
- 那么: $\dfrac{\partial \mathbf{z}}{\partial \mathbf{x}} = \dfrac{\partial \mathbf{y}}{\partial \mathbf{x}} \dfrac{\partial \mathbf{z}}{\partial \mathbf{y}}$

注意, 与“标量”的链式法则的区别是:
向量求导链“从右向左”构造.

② $\hat{\mathbf{y}} = Softmax(\mathbf{z}^{(3)})$ 函数对”向量 $\mathbf{z}^{(3)}$ ”求导的“雅可比矩阵”：

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(3)}} = \begin{bmatrix} \hat{y}_1\,(1-\hat{y}_1) & -\hat{y}_1\hat{y}_2 & -\hat{y}_1\hat{y}_3 \\ -\hat{y}_2\hat{y}_1 & \hat{y}_2\,(1-\hat{y}_2) & -\hat{y}_2\hat{y}_3 \\ -\hat{y}_3\hat{y}_1 & -\hat{y}_3\hat{y}_2 & \hat{y}_3\,(1-\hat{y}_3) \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

① 交叉熵对$\hat{\mathbf{y}}$求导：

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}} = -\mathbf{y} \odot \frac{\mathbf{1}}{\hat{\mathbf{y}}} = \begin{bmatrix} -y_1 \frac{1}{\hat{y}_1} \\ -y_2 \frac{1}{\hat{y}_2} \\ -y_3 \frac{1}{\hat{y}_3} \end{bmatrix} \in \mathbb{R}^{3\times 1}$$

③ 求$\boldsymbol{\delta}^{(3)}$, 即$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}})$对$\mathbf{z}^{(3)}$的导数:

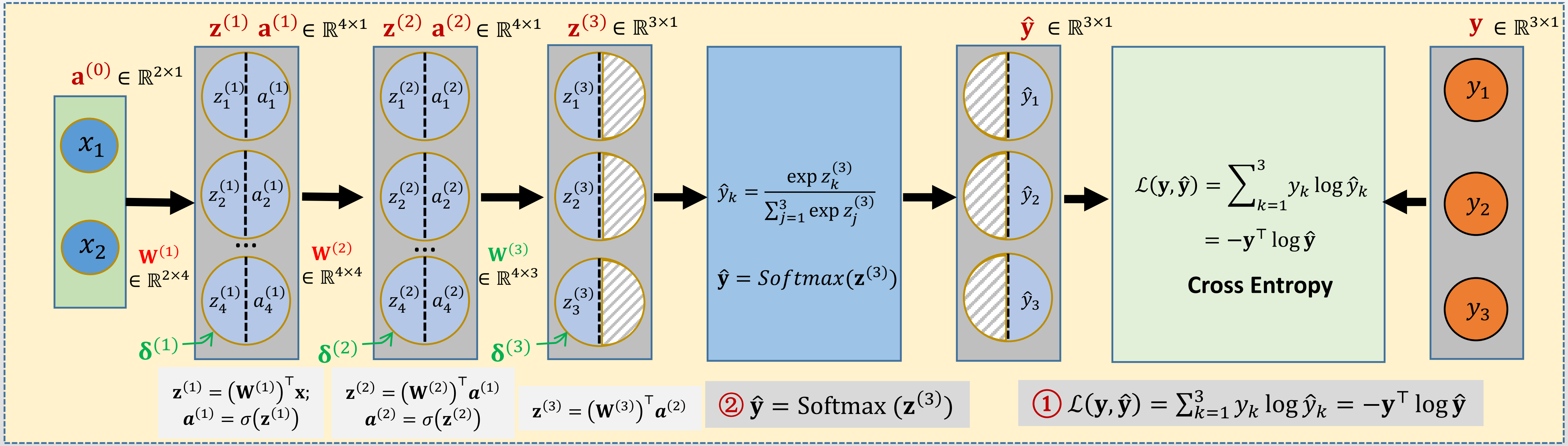
$$\boldsymbol{\delta}^{(3)} = \underset{\mathbb{R}^{3\times 1}}{\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{z}^{(3)}}} = \underset{\mathbb{R}^{3\times 3}}{\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(3)}}} \underset{\mathbb{R}^{3\times 1}}{\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}}} = \begin{bmatrix} (\hat{y}_1 - 1)y_1 + \hat{y}_1 y_2 + \hat{y}_1 y_3 \\ \hat{y}_2 y_1 + (\hat{y}_2 - 1)y_2 + \hat{y}_2 y_3 \\ \hat{y}_3 y_1 + \hat{y}_3 y_2 + (\hat{y}_3 - 1)y_3 \end{bmatrix} \in \mathbb{R}^{3\times 1}$$

举个例子: 当 $\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ 时, $\boldsymbol{\delta}^{(3)} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 - 1 \\ \hat{y}_3 \end{bmatrix} = \begin{bmatrix} \hat{y}_1 - 0 \\ \hat{y}_2 - 1 \\ \hat{y}_3 - 0 \end{bmatrix} = \hat{\mathbf{y}} - \mathbf{y}$

在“Softmax分类任务”中,
反向传播的第一个方程式 (BP1)
(与“任务相关”的梯度$\boldsymbol{\delta}^{(L)}$):

$$\boldsymbol{\delta}^{(3)} = \hat{\mathbf{y}} - \mathbf{y} \in \mathbb{R}^{3\times 1}$$

FFNN classification with multiple hidden layers



反向传播的四个方程式

① $\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{a}^{(L)}} L \odot \sigma'(\mathbf{z}^{(L)})$ (BP1)

② $\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}) \odot \sigma'(\mathbf{z}^{(l)})$ (BP2)

③ $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \mathbf{a}^{(l-1)} (\boldsymbol{\delta}^{(l)})^\top$ (BP3)

④ $\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$ (BP4)

“Softmax 分类任务”反向传播的四个方程式

① $\boldsymbol{\delta}^{(3)} = \hat{\mathbf{y}} - \mathbf{y} \in \mathbb{R}^{3\times1}$ (BP1)

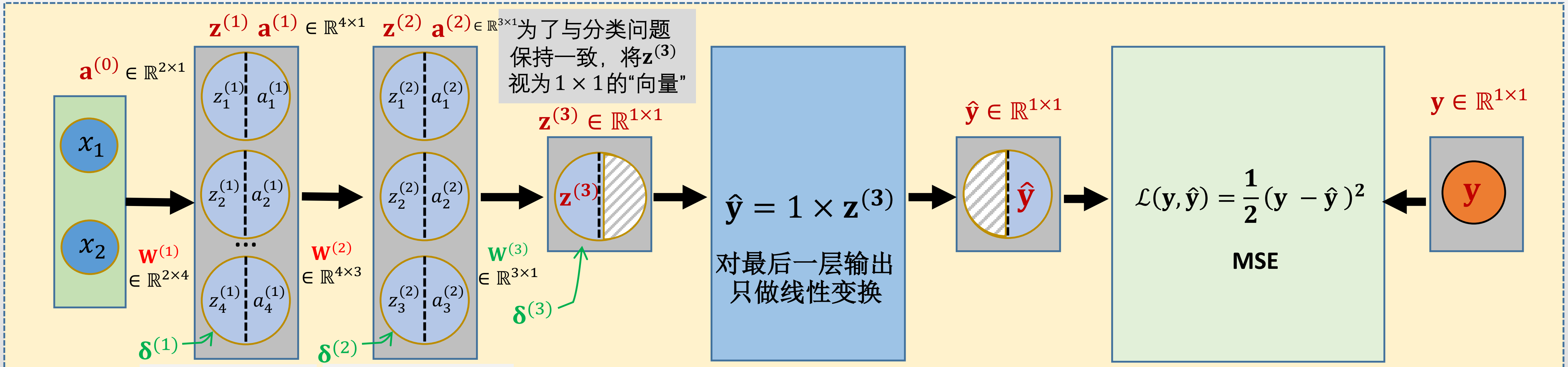
② $\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)} \boldsymbol{\delta}^{(l+1)}) \odot \sigma'(\mathbf{z}^{(l)})$ (BP2)

用于计算 $\boldsymbol{\delta}^{(1)}$和$\boldsymbol{\delta}^{(2)}$

③ $\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \mathbf{a}^{(1)} (\boldsymbol{\delta}^{(2)})^\top$; $\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \mathbf{a}^{(0)} (\boldsymbol{\delta}^{(1)})^\top$ (BP3)

④ $\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \boldsymbol{\delta}^{(2)}$; $\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \boldsymbol{\delta}^{(1)}$ (BP4)

FFNN Regression with multiple hidden layers



$\mathbf{z}^{(1)} = \left(\mathbf{W}^{(1)}\right)^{\mathsf{T}}\mathbf{x}$; $\boldsymbol{a}^{(1)} = \sigma\left(\mathbf{z}^{(1)}\right)$

$\mathbf{z}^{(2)} = \left(\mathbf{W}^{(2)}\right)^{\mathsf{T}}\boldsymbol{a}^{(1)}$; $\boldsymbol{a}^{(2)} = \sigma\left(\mathbf{z}^{(2)}\right)$

$\mathbf{z}^{(3)} = \left(\mathbf{W}^{(3)}\right)^{\mathsf{T}}\boldsymbol{a}^{(2)}$

② $\hat{\mathbf{y}} = \mathbf{1} \times \mathbf{z}^{(3)}$

① $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2}(\mathbf{y} - \hat{\mathbf{y}})^2$

① MSE $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 对$\hat{\mathbf{y}}$求导：$\dfrac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}} = \hat{\mathbf{y}} - \mathbf{y} \in \mathbb{R}^{1\times 1}$

② $\hat{\mathbf{y}} = \mathbf{1} \times \mathbf{z}^{(3)}$线性函数对"向量$\mathbf{z}^{(3)}$"求导：$\dfrac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(3)}} = [1] \in \mathbb{R}^{1\times 1}$

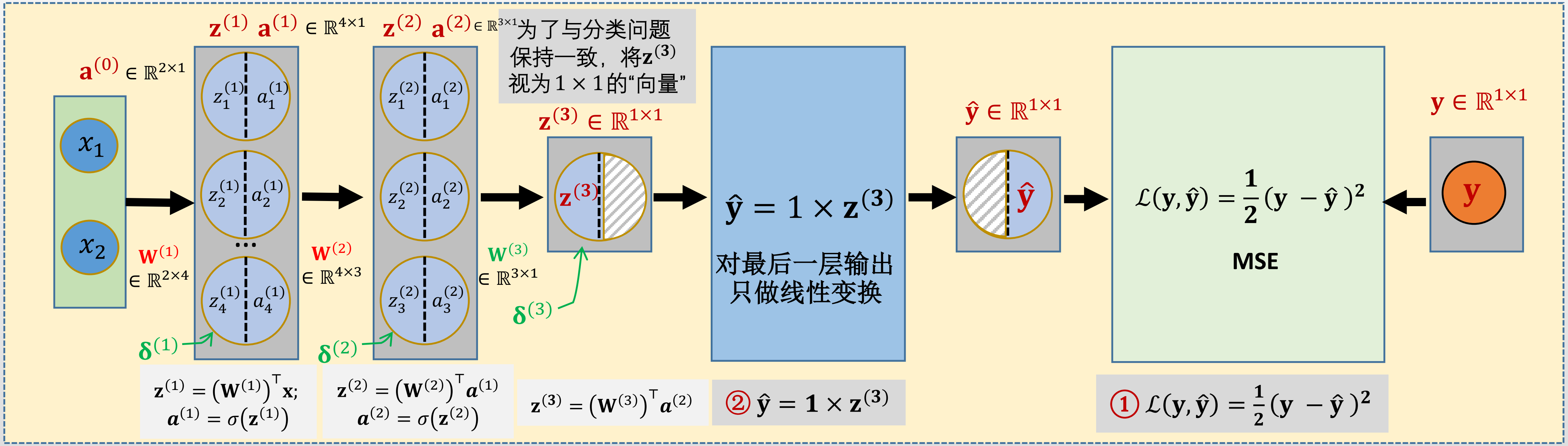
③ 求$\boldsymbol{\delta}^{(3)}$, 即$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$对$\mathbf{z}^{(3)}$的导数：$\boldsymbol{\delta}^{(3)} = \dfrac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(3)}} = \dfrac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}^{(3)}} \dfrac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}} = \hat{\mathbf{y}} - \mathbf{y} \in \mathbb{R}^{1\times 1}$

在“回归任务”中，
反向传播的第一个方程式 (BP1)
(与“任务相关”的梯度$\boldsymbol{\delta}^{(L)}$)：

$$\boldsymbol{\delta}^{(3)} = \hat{\mathbf{y}} - \mathbf{y} \in \mathbb{R}^{1\times 1}$$

与“Softmax”分类的结果一致！
只是维度不同.

FFNN Regression with multiple hidden layers



反向传播的四个方程式

① $\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{a}^{(L)}} L \odot \sigma'(\mathbf{z}^{(L)})$ (BP1)

② $\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)}\boldsymbol{\delta}^{(l+1)}) \odot \sigma'(\mathbf{z}^{(l)})$ (BP2)

③ $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \mathbf{a}^{(l-1)}(\boldsymbol{\delta}^{(l)})^{\mathsf{T}}$ (BP3)

④ $\frac{\partial L}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$ (BP4)

“回归任务”反向传播的四个方程式

① $\boldsymbol{\delta}^{(3)} = \hat{\mathbf{y}} - \mathbf{y} \in \mathbb{R}^{1\times1}$ (BP1)

② $\boldsymbol{\delta}^{(l)} = (\mathbf{W}^{(l+1)}\boldsymbol{\delta}^{(l+1)}) \odot \sigma'(\mathbf{z}^{(l)})$ (BP2)

用于计算$\boldsymbol{\delta}^{(1)}$和$\boldsymbol{\delta}^{(2)}$

③ $\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \mathbf{a}^{(1)}(\boldsymbol{\delta}^{(2)})^{\mathsf{T}}$; $\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \mathbf{a}^{(0)}(\boldsymbol{\delta}^{(1)})^{\mathsf{T}}$ (BP3)

④ $\frac{\partial L}{\partial \boldsymbol{b}^{(2)}} = \boldsymbol{\delta}^{(2)}$; $\frac{\partial L}{\partial \boldsymbol{b}^{(1)}} = \boldsymbol{\delta}^{(1)}$ (BP4)