交通大数据

人工神经网络

- 刘志远教授
- zhiyuanl@seu.edu.cn

人工神经网络

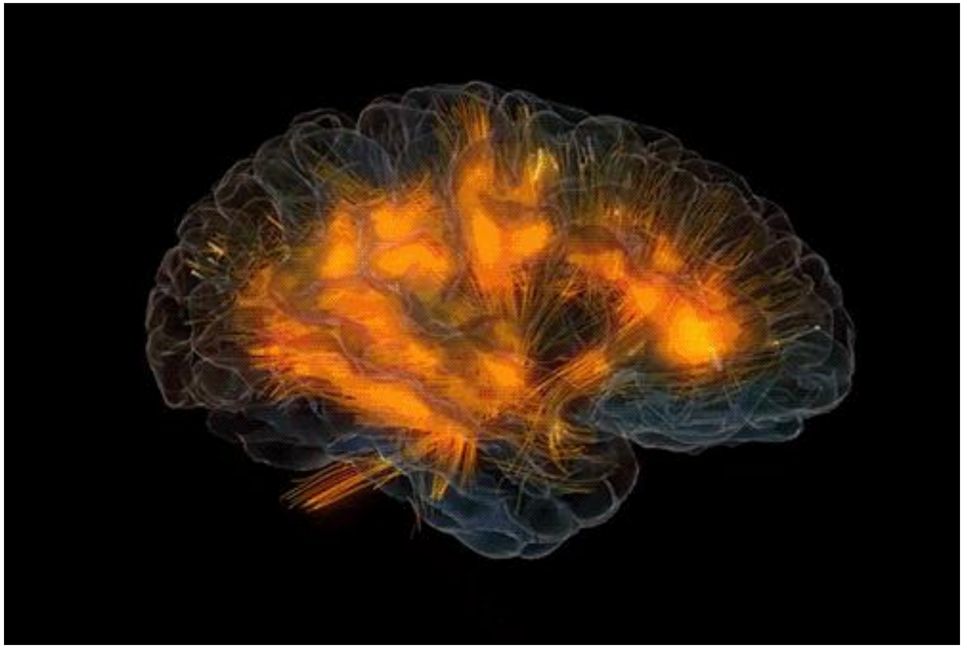
学习目标

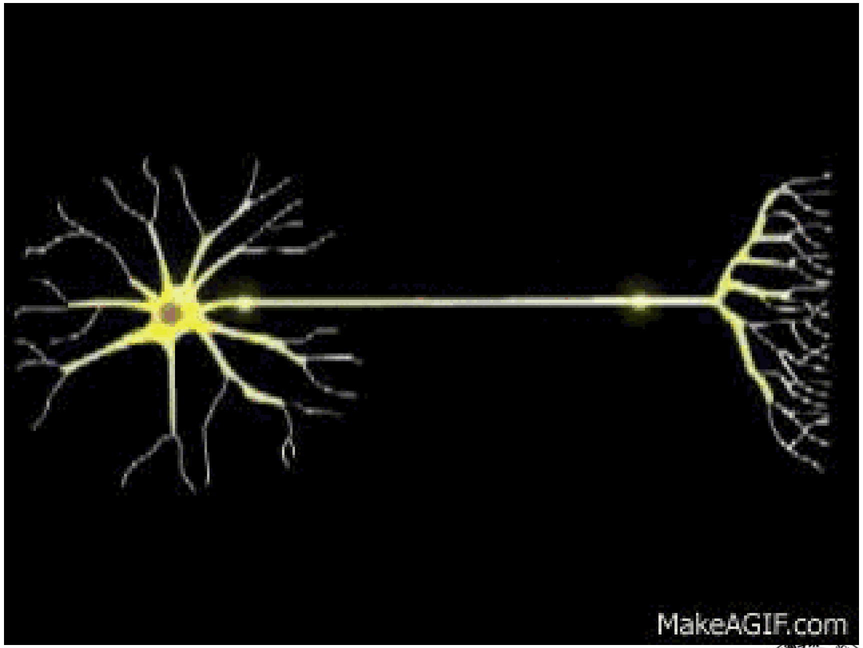
- 学习神经网络的基本结构
  - 神经元模型的基本概念
  - 神经元模型的通用表示形式
  - 多层神经网络模型
  - 激活函数的类型与作用
- 掌握神经网络模型的前向传播与后向传播
- 掌握神经网络中常见问题的解决方法
- 掌握应用简单的人工神经网络模型解决实际问题的方法

11.1 神经网络的基本结构

❑ 生物神经网络

通过突触结构，当由上游神经元输入信号电位超过某个阈值时，神经元才会进入兴奋状态，从而产生输出信号。

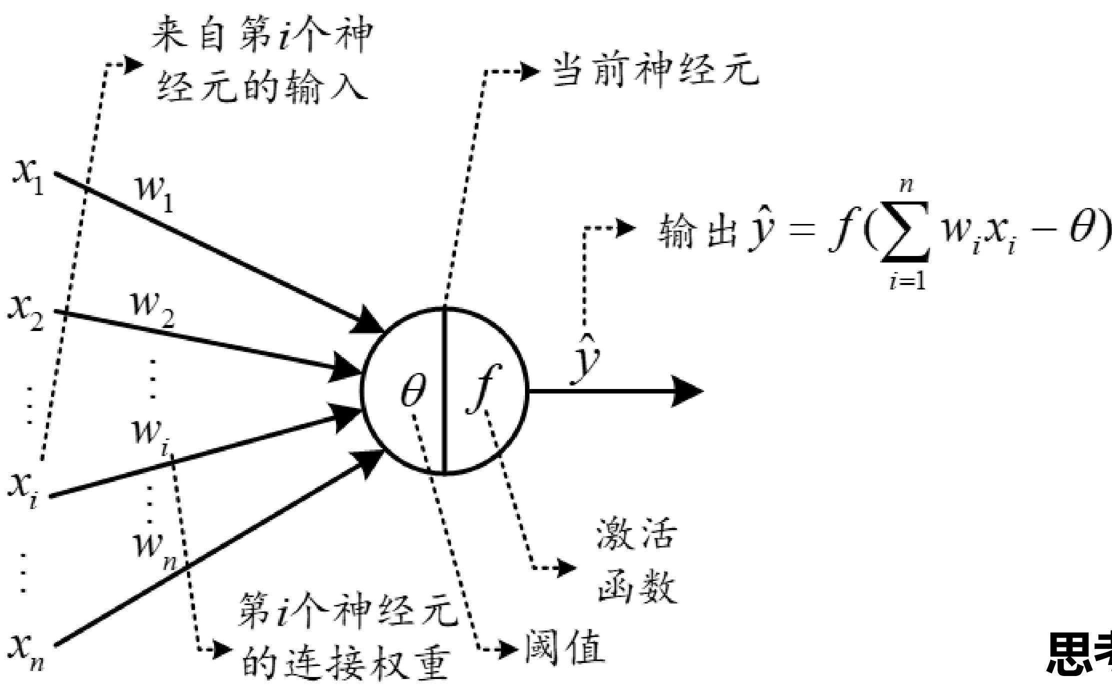




❑ M-P神经元模型（1943）

step 1. 输入信号计算：输入信号用向量x表示，$x=(x_1,x_2,\dots,x_n)$，连接权重用w表示，$w=(w_1,w_2,\dots,w_n)$，则总输入信号为$w^T x$。

step 2. 激活：$w^T x$与阈值θ进行比较，若$w^T x>\theta$，则输出激活信号（神经元**兴奋**），否则不输出信号（神经元**抑制**）。



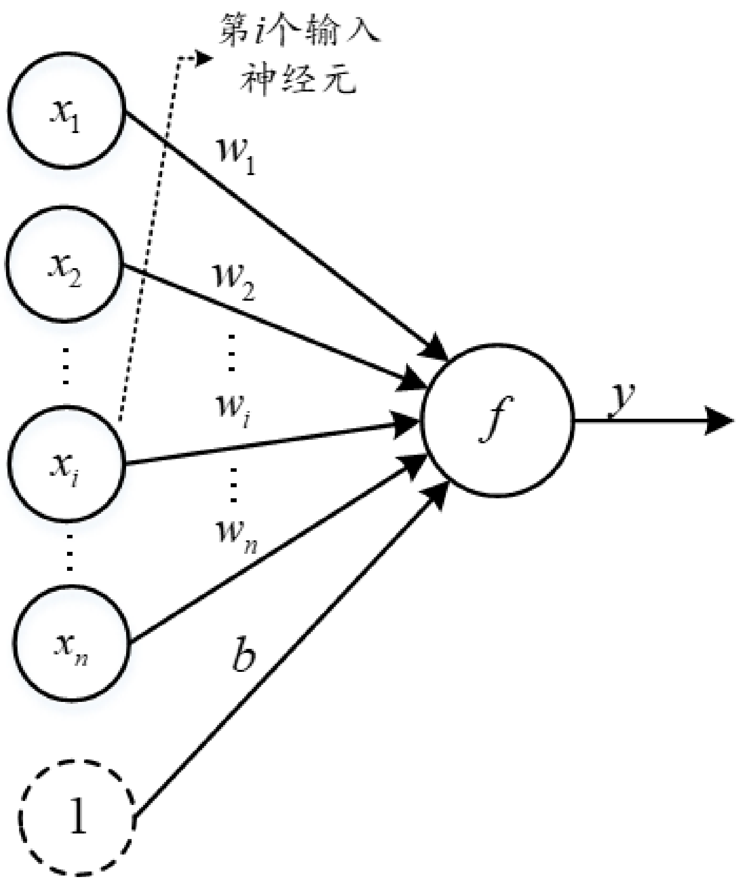
$$f(x)=\begin{cases}0 & x\le 0\\ 1 & x>0\end{cases}$$

阶跃激活函数

思考：阶跃激活函数优缺点

❑ 神经元模型的通用表示

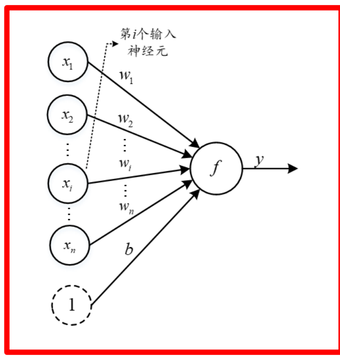
为了方便后续的数学模型分析，我们将阈值θ也作为一个输入值来处理，此时定义符号$b = -\theta$，可以将阈值看作为一个输入恒为1、权重为b的值。此处b也被称作**偏差或偏置**（bias）。

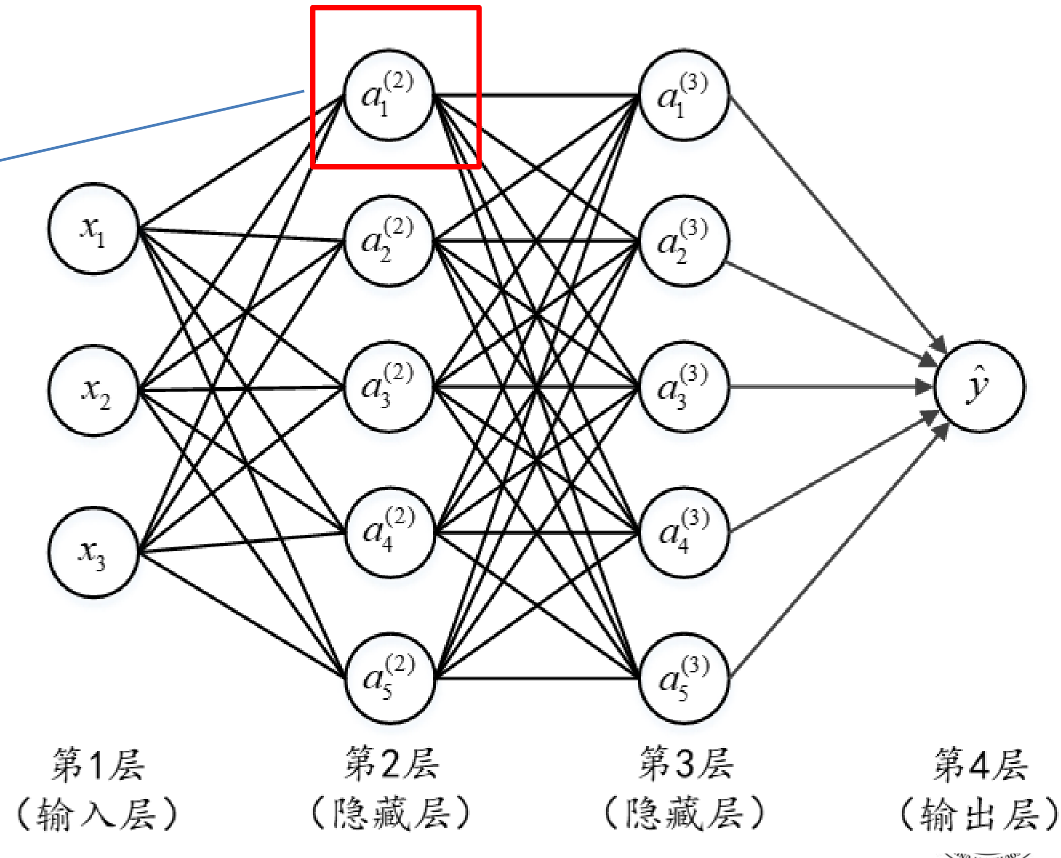


$$y = f\left(\boldsymbol{w}^T\boldsymbol{x} + b\right) = f(z)$$

❑ 多层神经网络模型

现实中使用的复杂神经网络是由许多层神经元模型组成的，每层存在多个神经元（神经元数量不一定相等）。神经网络的结构（architecture）定义了网络中的变量和它们的拓扑关系。





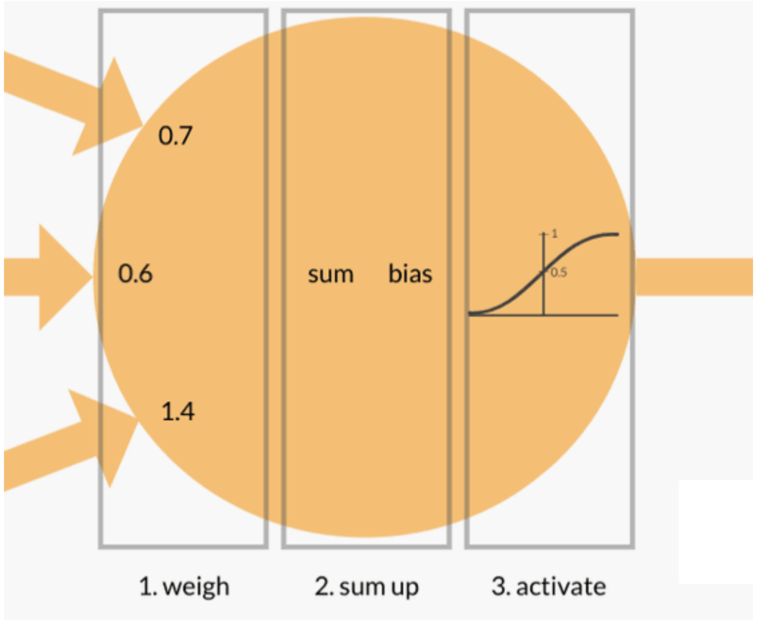
前馈神经网络：

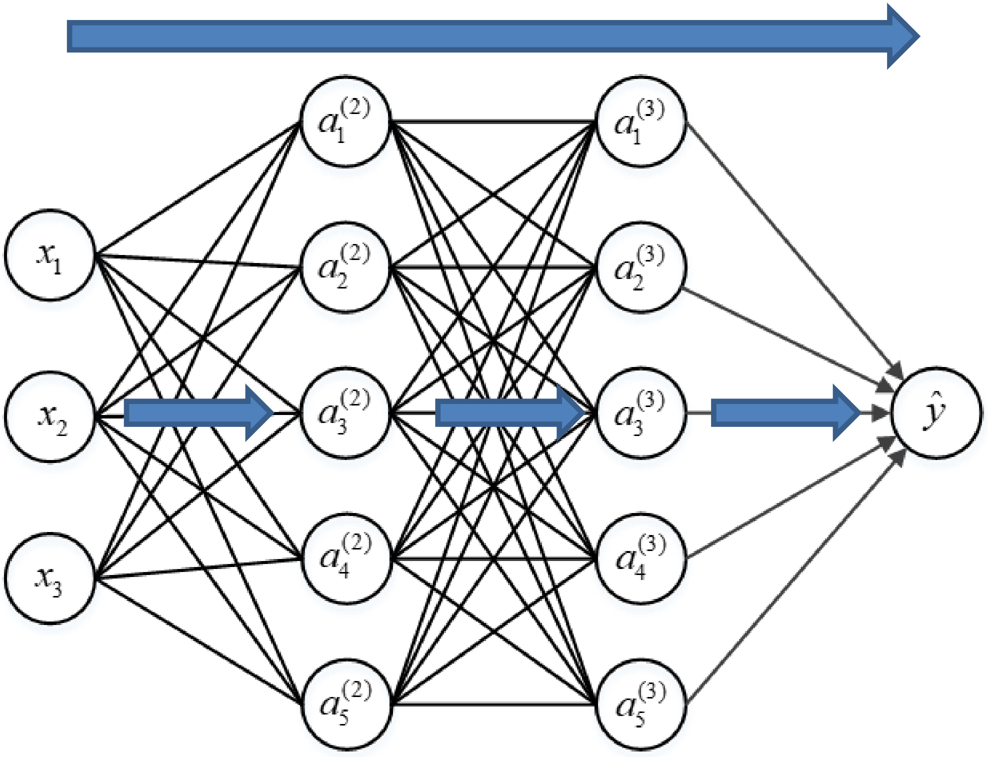
每层神经元与下一层神经元**全连接**

不存在同层连接、不存在跨层连接

❑ 神经网络模型的计算过程

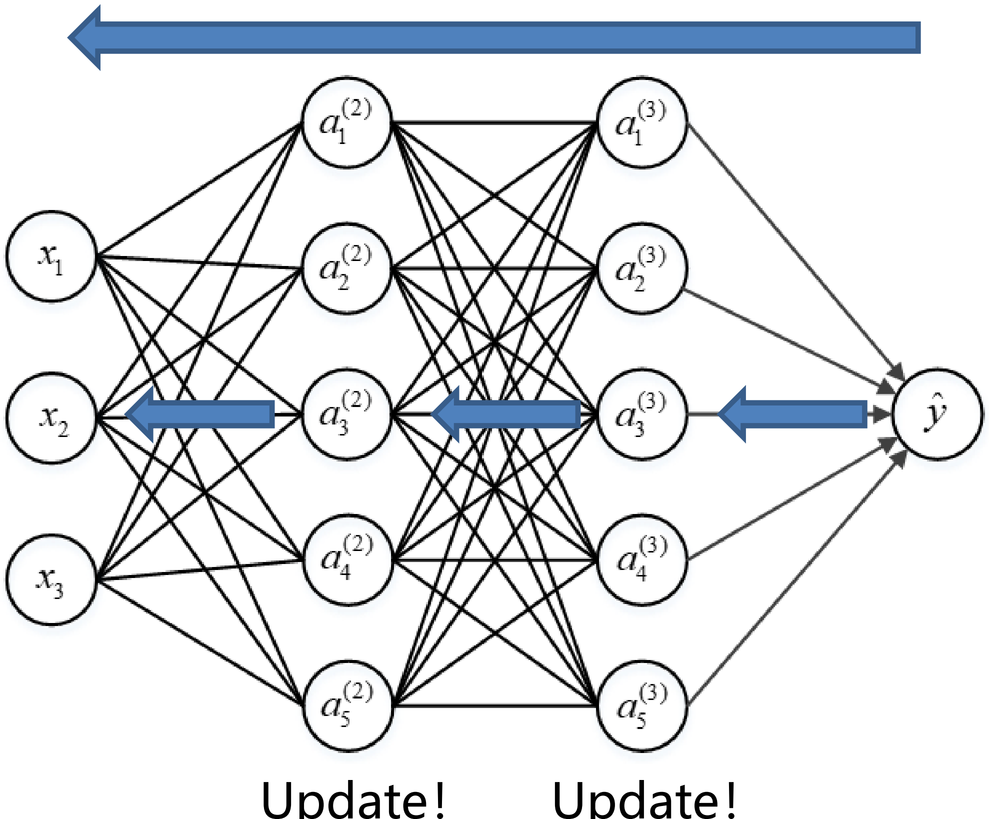
输入数据由输入层进入隐藏层，隐藏层的神经元通过权重和偏差对数据进行线性计算，又通过激活函数完成非线性计算，通过多个隐藏层的处理，将最终的结果传递到输出层，这个计算过程被称为“**前向传播**”。





❑ 神经网络模型的训练过程

模型的训练过程就是寻找神经网络中最优的模型参数（权重和偏差）的过程，被称为“**反向传播**”。其基本原理是通过从输出层到输入层，逐层反向计算参数的梯度，对参数进行更新，直到获得最优值。



11.2 激活函数与前向传递

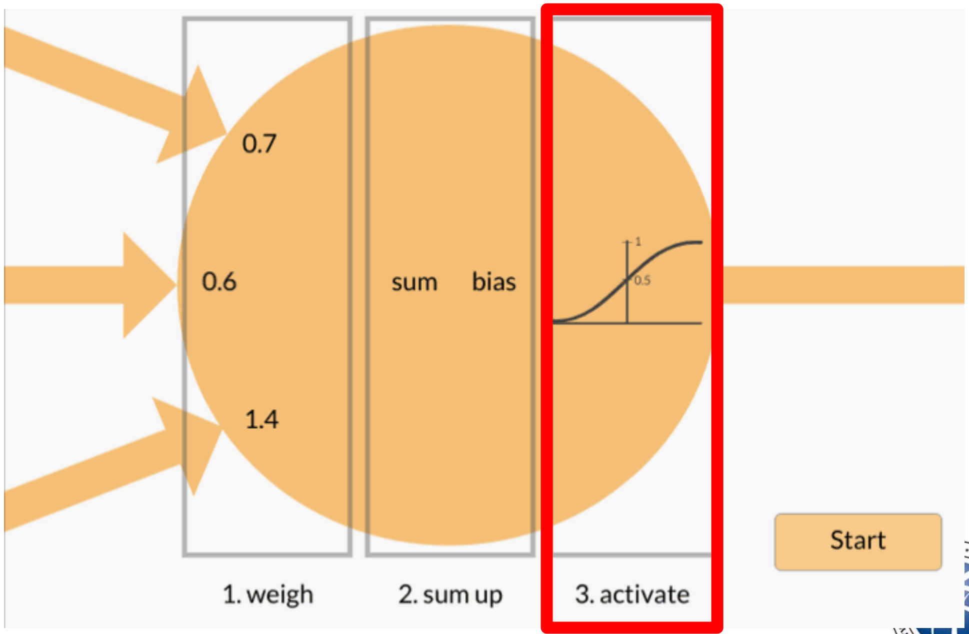
❑ 激活函数

激活函数对输入信息进行**非线性计算**，然后将计算后的值传递至下一层神经元。没有激活函数的神经元模型**等价于线性模型**，激活函数对数据进行非线性计算，是神经网络能够拟合各类复杂关系，具有**强大表征能力的重要基础**。

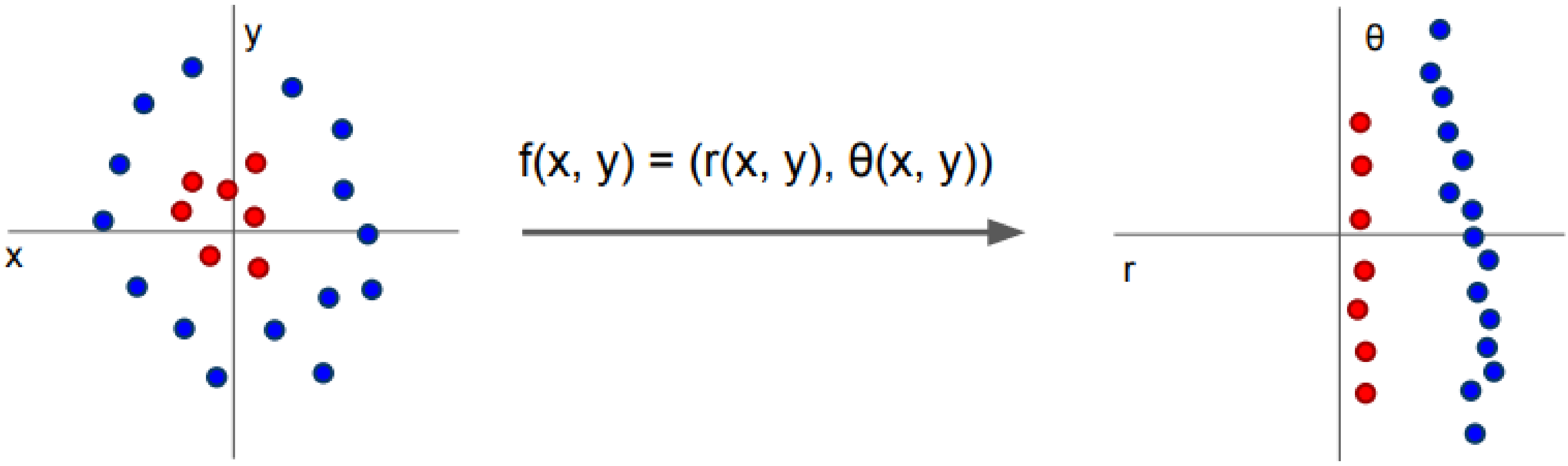
思考：

激活函数为什么是非线性的?

如果没有激活函数f会发生什么?



❑ 为什么激活函数是非线性的?

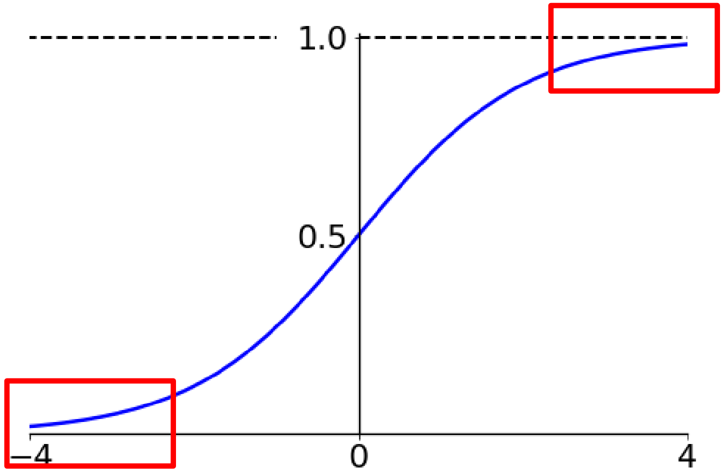


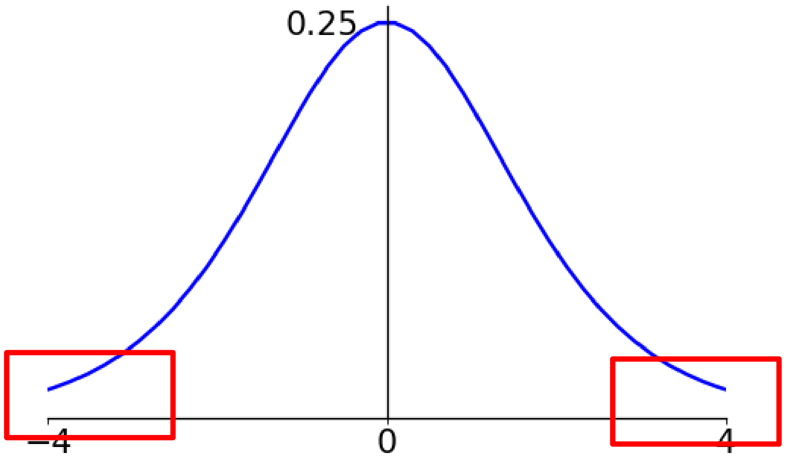
线性分类模型无法区分红色点与蓝色点

通过特征转换，线性分类器具有了分类能力

常用激活函数

1. Sigmoid 函数





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

$$\sigma'(x)=\sigma(x)[1-\sigma(x)]$$

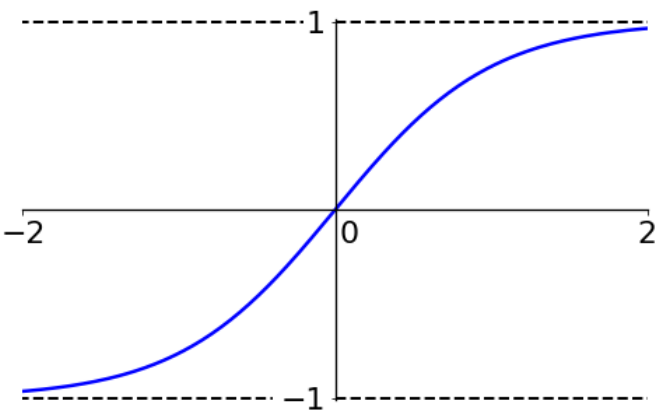
存在问题：

1. Sigmoid函数的输出恒为正值，不是以零为中心的，这会导致权值更新时只能朝一个方向更新，从而影响收敛速度。

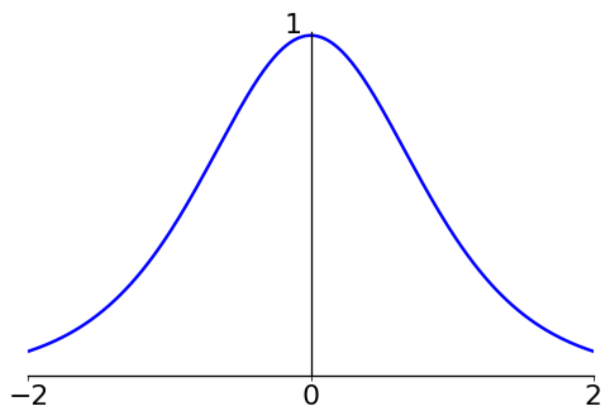
2. 输入非常大或非常小时，函数导数接近于0，易造成梯度值过小（梯度消失），进一步导致参数训练算法的收敛速度过慢，甚至无法收敛。

2. Tanh（双曲正切）函数

Tanh与Sigmoid的形状相似，但输出值域为[-1,1]，均值为0；Tanh函数的梯度变化更陡；但仍存在梯度消失的问题。



$$\tanh(x)=\frac{1-e^{-2x}}{1+e^{-2x}}$$

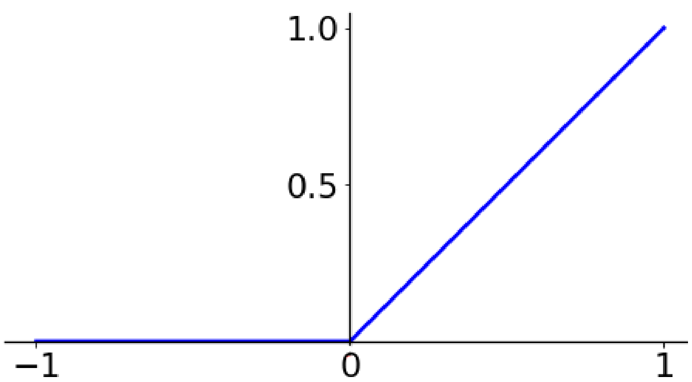


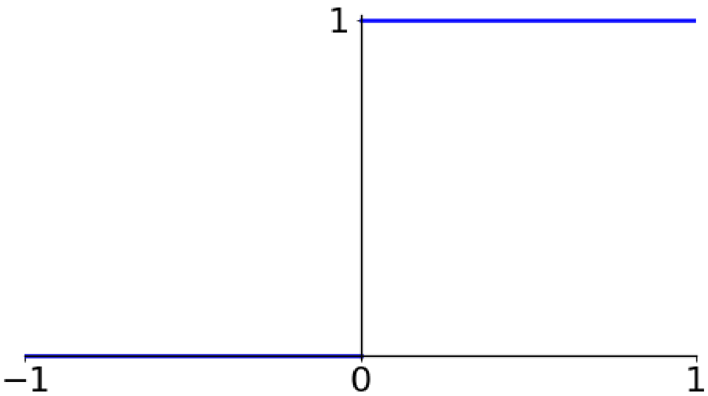
$$\tanh'(x)=1-[\tanh(x)]^2$$

3. ReLU 函数

ReLU （Rectified Linear Unit）函数能够克服梯度消失问题，ReLU是目前深度神经网络中使用最为广泛的激活函数，之后有学者在其基础上提出了新的激活函数，如Leaky ReLU、ELU等。

ReLU函数有两个**优势**：① ReLU函数形式更加简单，易于计算 ② 采用ReLU函数训练得到的网络具有一定的稀疏性，只对少量的输入值（正值）产生响应。





$$relu(x)=\begin{cases}0 & x<0\\ x & x\geq 0\end{cases}$$

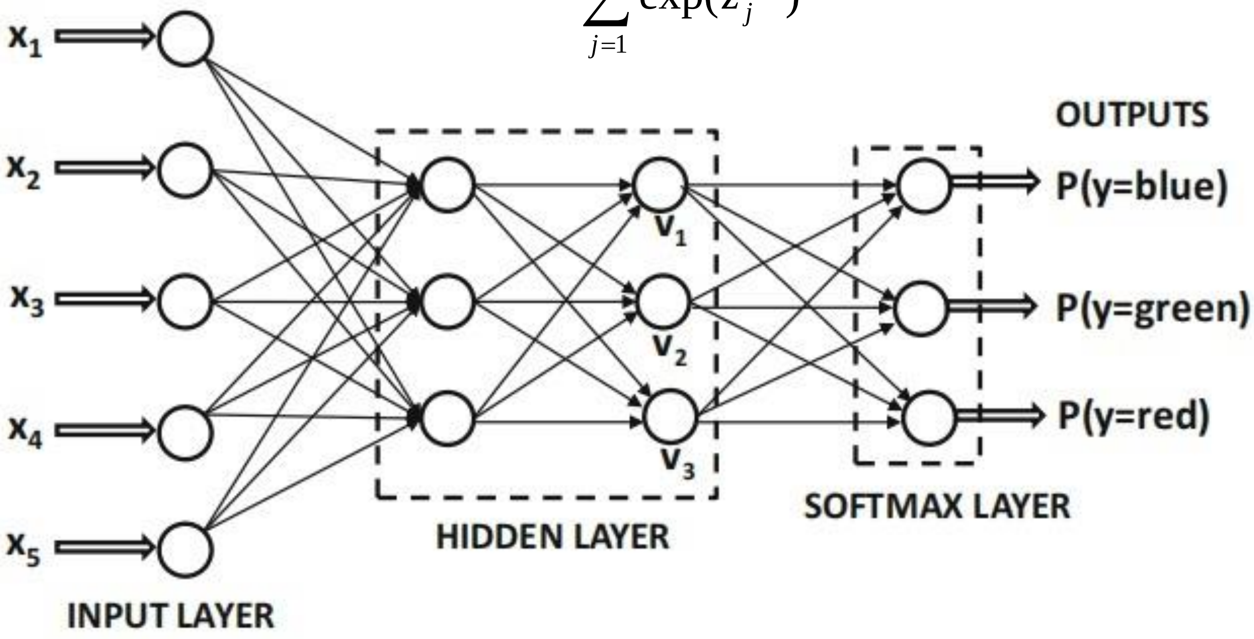
$$relu'(x)=\begin{cases}0 & x<0\\ 1 & x\geq 0\end{cases}$$

❑ 常用激活函数

4. Softmax 函数

Softmax函数为分类问题的每一个类别都提供一个输出，得到一个输出向量；同时将输出向量进行归一化，从而获得各类的相对概率，因此Softmax函数也称作归一化指数函数。

$$\hat{y}_i = softmax_i(\mathbf{z}^{(L)}) = \frac{\exp(z_i^{(L)})}{\sum_{j=1}^{N^{(L)}} \exp(z_j^{(L)})}, i = 1, 2, \ldots N^{(L)}$$



以拥堵分类问题为例，存在畅通、缓行、拥堵3类，那么相应的神经网络结构中，输出层应包含3个神经元，对应输出一个3维向量，分别为样本属于畅通、缓行、拥堵的相对概率。假设在进入Softmax layer之前，畅通、缓行、拥堵三类对应的输出得分分别为10, 6, 8，那么Softmax函数的计算结果如下：

试一试

计算每类的相对概率

$$\hat{y}_i = softmax_i(\mathbf{z}^{(L)}) = \frac{\exp(z_i^{(L)})}{\sum_{j=1}^{N^{(L)}} \exp(z_j^{(L)})}, i = 1, 2, \ldots N^{(L)}$$

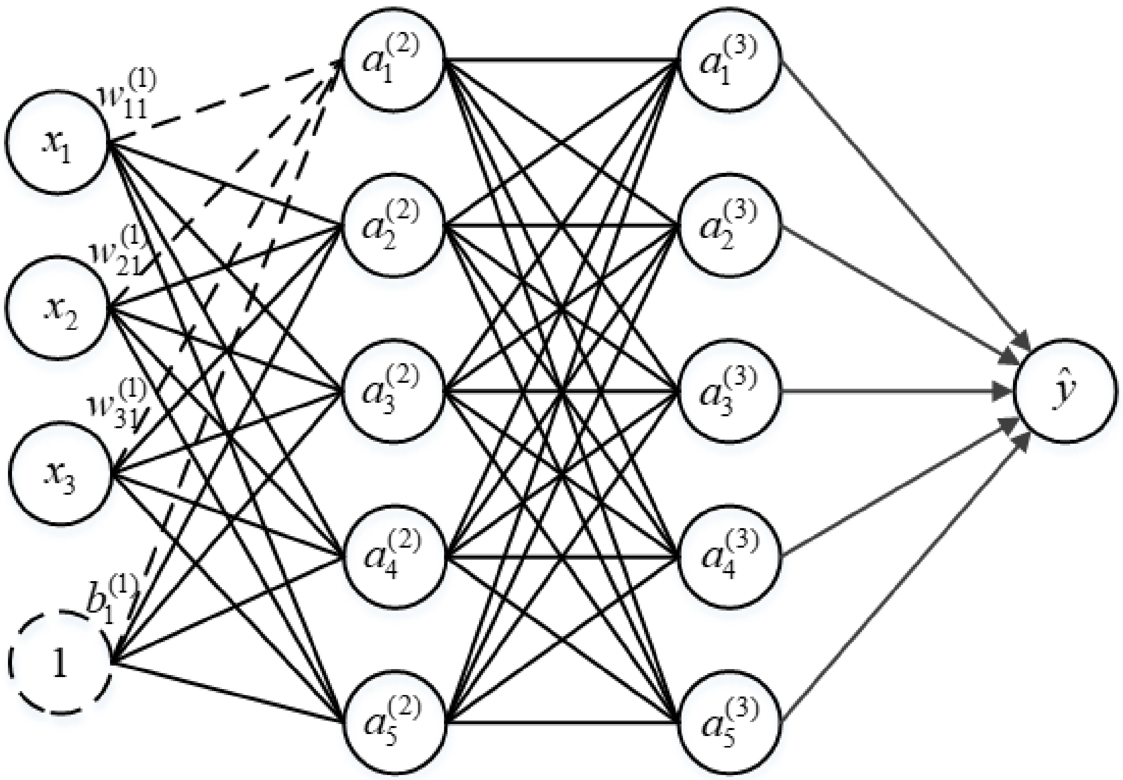
$$y_{畅通} = \frac{e^{10}}{e^{10} + e^8 + e^6} = 0.867$$

$$y_{缓行} = \frac{e^6}{e^{10} + e^8 + e^6} = 0.016$$

$$y_{拥堵} = \frac{e^8}{e^{10} + e^8 + e^6} = 0.117$$

前向传播

前向传播本质上是从输入层开始，逐渐计算每层每个神经元的输出值，直至输出层的过程。以$a_1^{(2)}$这单个神经元为例，展示其前向传播过程。



1. 确定广义输入$z_1^{(2)}$

$$z_1^{(2)} = \boldsymbol{w}_1^{(1),T}\boldsymbol{a}^{(1)} + b_1^{(1)}$$

$$\boldsymbol{w}_1^{(1)} = \left(w_{11}^{(1)}, w_{21}^{(1)}, w_{31}^{(1)}\right)$$

$$\boldsymbol{a}^{(1)} = (x_1, x_2, x_3)$$

2. 计算输出

$$a_1^{(2)} = f\left(z_1^{(2)}\right)$$

$$= f(\boldsymbol{w}_1^{(1),T}\boldsymbol{a}^{(1)} + b_1^{(1)})$$

前向传播

1. 重复上述计算过程，我们可以逐一计算第2层的其他四个神经元，及第三层、第四层的每个神经元的输出值。
2. 全连接层之间的计算能够转化为“矩阵”之间的运算：其中$f(x)$代表激活函数

$$\boldsymbol{z}^{(l+1)}=\boldsymbol{w}^{(l),T}\boldsymbol{a}^{(l)}+\boldsymbol{b}^{(l)},l=1,2,\dots,L-1$$

$$\boldsymbol{a}^{(l+1)}=f\left(\boldsymbol{z}^{(l+1)}\right),l=1,2,\dots,L-1$$

$$w^{(1),T}=\begin{pmatrix} w_{11}^{(1)},w_{21}^{(1)},w_{31}^{(1)} \\ w_{12}^{(1)},w_{22}^{(1)},w_{32}^{(1)} \\ w_{13}^{(1)},w_{23}^{(1)},w_{33}^{(1)} \\ w_{14}^{(1)},w_{24}^{(1)},w_{34}^{(1)} \\ w_{15}^{(1)},w_{25}^{(1)},w_{35}^{(1)} \end{pmatrix} \quad \boldsymbol{a}^{(1)}=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \boldsymbol{b}^{(1)}=\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

试一试

动手试一试展开

❑ 前向传播

$$\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(1),T}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(1)}$$

$$\boldsymbol{z}^{(2)} = \begin{pmatrix} w_{11}^{(1)}, w_{21}^{(1)}, w_{31}^{(1)} \\ w_{12}^{(1)}, w_{22}^{(1)}, w_{32}^{(1)} \\ w_{13}^{(1)}, w_{23}^{(1)}, w_{33}^{(1)} \\ w_{14}^{(1)}, w_{24}^{(1)}, w_{34}^{(1)} \\ w_{15}^{(1)}, w_{25}^{(1)}, w_{35}^{(1)} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} = \begin{pmatrix} w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3 + b_1 \\ w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3 + b_2 \\ w_{13}^{(1)}x_1 + w_{23}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3 \\ w_{14}^{(1)}x_1 + w_{24}^{(1)}x_2 + w_{34}^{(1)}x_3 + b_4 \\ w_{15}^{(1)}x_1 + w_{25}^{(1)}x_2 + w_{35}^{(1)}x_3 + b_5 \end{pmatrix}$$

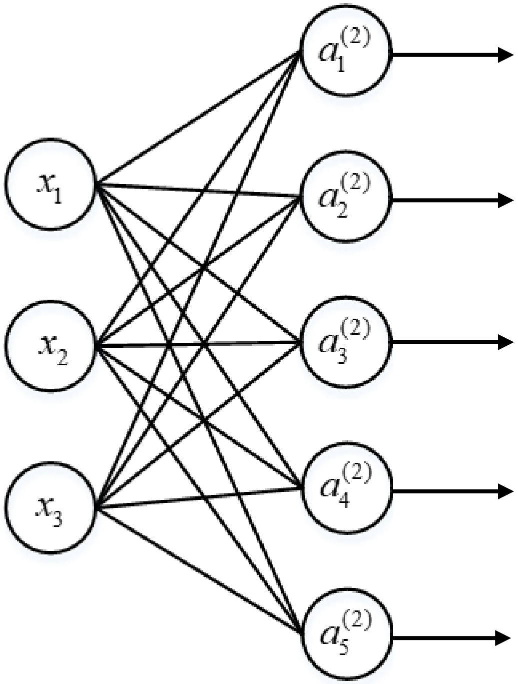
$$\boldsymbol{a}^{(2)} = f\left(\boldsymbol{z}^{(2)}\right)$$

$$\boldsymbol{a}^{(2)} = \begin{pmatrix} f(w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3 + b_1) \\ f(w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3 + b_2) \\ f(w_{13}^{(1)}x_1 + w_{23}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3) \\ f(w_{14}^{(1)}x_1 + w_{24}^{(1)}x_2 + w_{34}^{(1)}x_3 + b_4) \\ f(w_{15}^{(1)}x_1 + w_{25}^{(1)}x_2 + w_{35}^{(1)}x_3 + b_5) \end{pmatrix}$$

试一试

计算将样本$(1,-1,-2)$输入如下网络后产生的输出，其中激活函数采用ReLU

$$relu(x)=\begin{cases}0 & x<0\\ x & x\geq 0\end{cases}$$



$$\boldsymbol{w}^{(1)}=\begin{pmatrix}1&2&3&1&2\\2&3&4&0&1\\3&4&5&2&3\end{pmatrix}$$

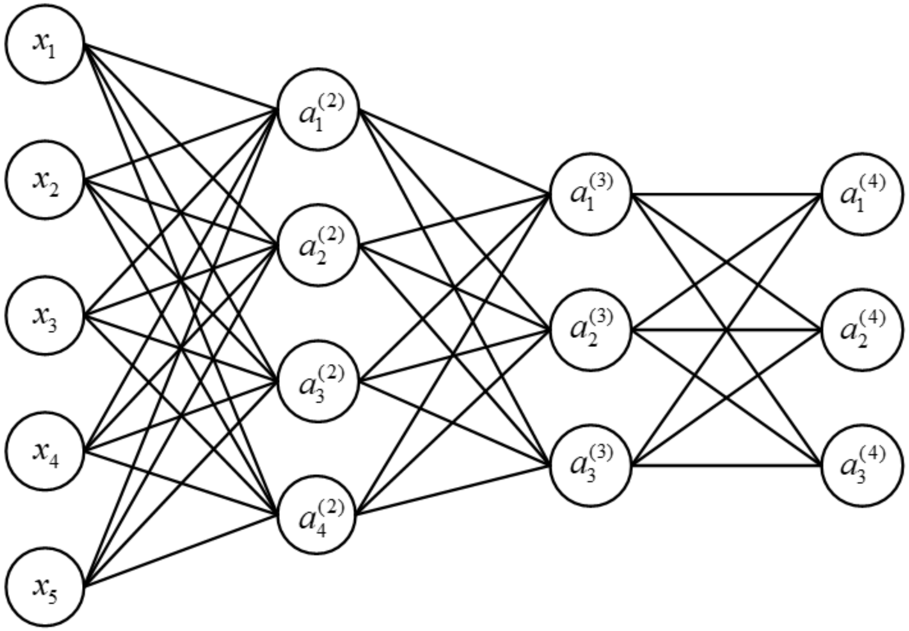
$$\boldsymbol{b}_1^{(1)}=\begin{pmatrix}8\\1\\6\\-1\\7\end{pmatrix}$$

$$\boldsymbol{z}^{(2)}=\boldsymbol{w}^{(1),T}\boldsymbol{a}^{(1)}+\boldsymbol{b}^{(1)}=\begin{pmatrix}1\\-8\\-5\\-4\\2\end{pmatrix}$$

$$\boldsymbol{a}^{(2)}=ReLU\left(\boldsymbol{z}^{(2)}\right)=\begin{pmatrix}1\\0\\0\\0\\2\end{pmatrix}$$

前向传播代码实现

- 解决问题：交通状态的预测（三分类）
- 特征工程：
  - aveSpeed平均车速
  - gridAcc平均加速度
  - volume流量
  - speed_std速度标准差
  - stopNum平均停车次数
- 模型结构
  - 输入层：5个神经元
  - 两个隐藏层使用Sigmoid激活函数
  - 输出层使用Softmax激活函数



试一试

试仅基于numpy库，一步一步实现该模型前向传播过程

Step 1. 实现sigmoid和softmax两个激活函数

```
# Sigmoid，z为输入值float
def sigmoid (z):
    pass
# Softmax
def softmax (z):
    pass
```

```
# Sigmoid
def sigmoid (z):
    return 1 / (1 + np.exp(-z))
# Softmax
def softmax (z):
    return np.exp(z) / np.sum(np.exp(z), axis=0)
```

试一试

试仅基于numpy库，一步一步实现该模型前向传播过程

Step 2. 实现前馈神经网络层

```
# 定义前馈神经网络层
def fc_layer(a, w, b, act_func):
    # a为输入值，b为bias，w为权重，act_func为损失函数
```

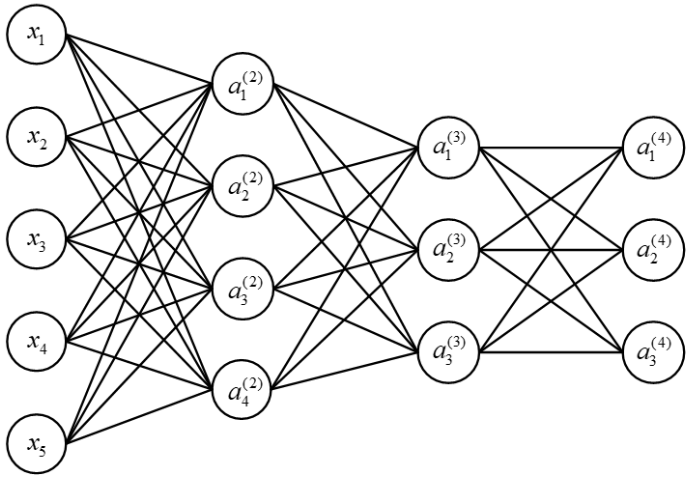
```
import numpy as np
def fc_layer(a, w, b, act_func):
    z = np.dot(w, a) + b
    a_out = act_func(z)
    return z, a_out
```

试一试

试仅基于numpy库，一步一步实现该模型前向传播过程

Step 3. 实现神经网络模型

```python
# 前向传递过程
def forward(x, w, b):
    # x为输入值，b为每层的bias，w为每层的权重
```




```python
# 前向传递过程
def forward(x, w, b):
    # 第2层，使用Sigmoid激活函数
    z2, a2 = fc_layer(x, w[0], b[0], sigmoid)
    # 第3层，使用Sigmoid激活函数
    z3, a3 = fc_layer(a2, w[1], b[1], sigmoid)
    # 第四层，输出层，使用Softmax激活函数
    z4, a4 = fc_layer(a3, w[2], b[2], softmax)
    return z2, a2, z3, a3, z4, a4
```

试一试

试仅基于numpy库，一步一步实现该模型前向传播过程

Step 4. 前向传播

```
# 随机初始化
def init_params(shape_w, shape_b):
    w = [np.random.randn(*x) for x in shape_w]
    b = [np.random.randn(*x) for x in shape_b]
    return w, b
np.random.seed(42)
shape_w = [(4, 5), (3, 4), (3, 3)]
shape_b = [(4, 1), (3, 1), (3, 1)]
w, b = init_params(shape_w, shape_b)
```

```
# 展示前向传播效果
x = np.random.randn(5, 1)
for idx, a in enumerate(forward(x, w, b)[1::2]):
    print(f'Layer {idx + 2}:')
    print(a)
```

试一试

试仅基于numpy库，一步一步实现该模型前向传播过程

Step 4. 输出结果

```
Layer 2:
[[0.86081109]
 [0.35346999]
 [0.03774752]
 [0.06688258]]
Layer 3:
[[0.59127908]
 [0.28733531]
 [0.63138266]]
Layer 4:
[[0.61434866]
 [0.00909127]
 [0.37656007]]
```

11.3 反向传播

❑ 前向传播与反向传播

基于给定的权重w与偏差b，前向传播都可以得到一个样本输入x对应的输出值$\hat{y}$。对于一个带标签的样本(x,y)，为了降低预测值$\hat{y}$与真值y之间的**误差**，提高模型的拟合能力，我们需要**调整/训练**出最优的权重与偏差。**反向传播**就是为了**实现权重与偏差动态调整**的过程。

对于输出值和真值之间的误差可以用第5章中介绍过的损失函数来量化计算，比如对于回归问题常用MSE损失，分类问题常用交叉熵损失。

$$MSEloss=\sum_i(\hat{y}-y)^2$$

$$CrossEntropyLoss=\sum_i y\log(\frac{1}{\hat{y}})$$

11.3 反向传播

❑ 神经网络模型的训练

神经网络训练过程本质上是一个以“损失函数最小化”为目标方程，以权重和偏差为未知变量的**非线性优化问题**。神经网络结构复杂，**无法得到解析解**，因此最优解只能通过优化算法**有限次迭代**来求得最优解。

back-propagation（BP）算法是一种以**负梯度方向**为迭代方向的梯度下降法，从输出层往前传递，传至每一个可学习参数。以权重矩阵的更新为例，每次迭代的更新公式为

$$w = w - \alpha \cdot \Delta \boldsymbol{w}$$

α代表学习率/步长，在神经模型模型的求解中通常取**固定值**

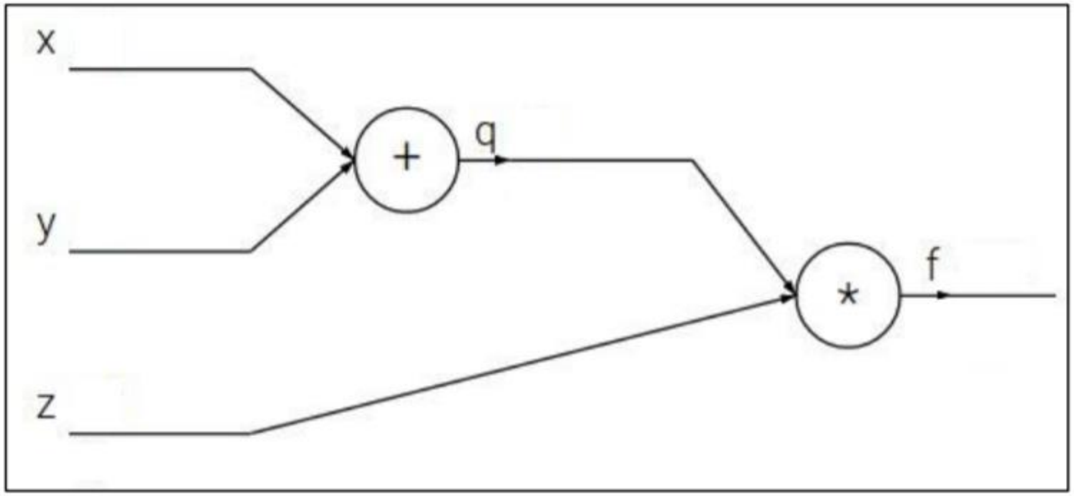
思考：为什么是负梯度方向?

为什么步长通常取固定值?

BP Sample 1

我们先用一个简单的例子说明反向传播算法的数学原理：**求导函数的链式法则**

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

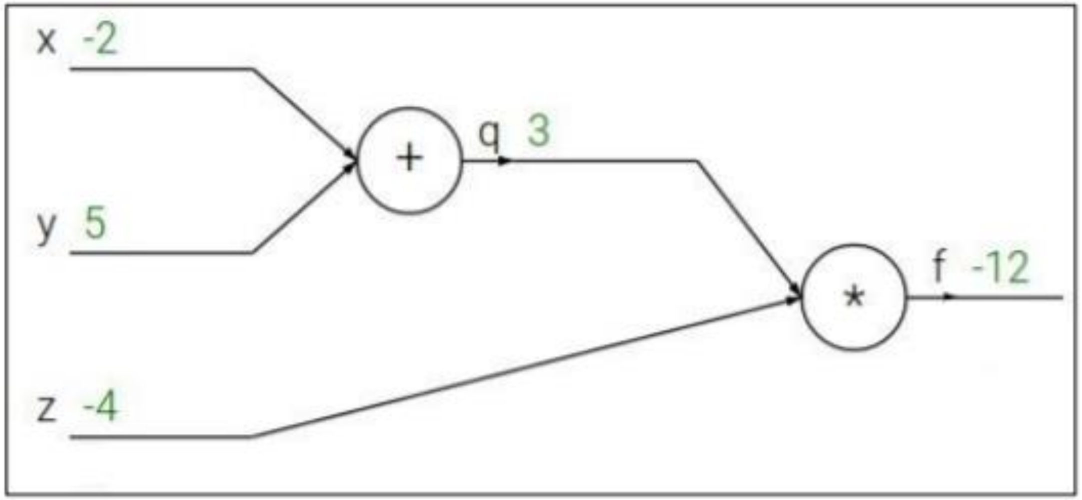


❑ BP Sample 1

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

e.g. 若存在一个样本

$(x = -2, y = 5, z = -4)$



BP Sample 1

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

e.g. 若存在一个样本

$$(x = -2, y = 5, z = -4)$$

$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

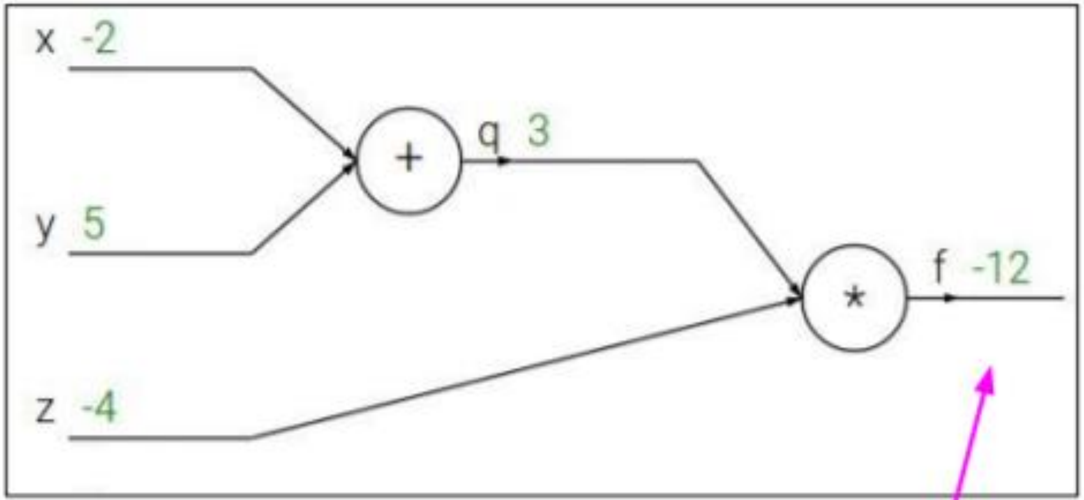
$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

❑ BP Sample 1

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

e.g. 若存在一个样本

$(x=-2, y=5, z=-4)$



$\frac{\partial f}{\partial f}$

$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

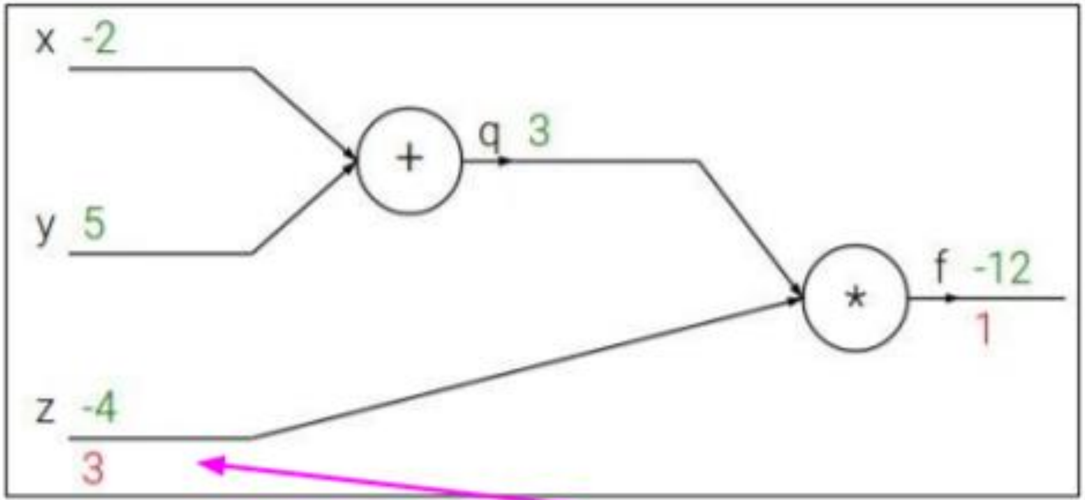
$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

❑ BP Sample 1

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

e.g. 若存在一个样本

$$(x=-2, y=5, z=-4)$$



$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

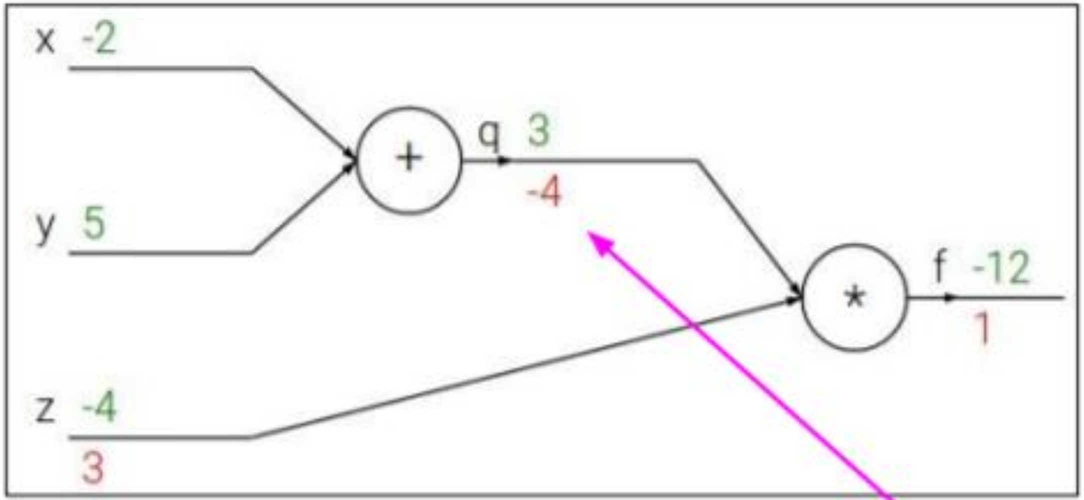
$$\boxed{\frac{\partial f}{\partial z}} = q = 3$$

BP Sample 1

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

e.g. 若存在一个样本

$(x=-2, y=5, z=-4)$



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

$= z = -4$

$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

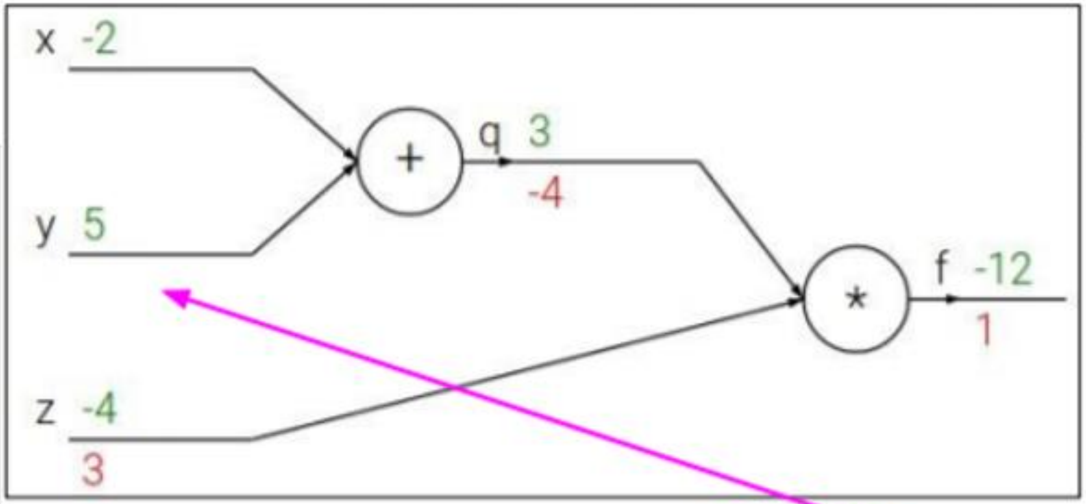
$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

BP Sample 1

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

e.g. 若存在一个样本

$$(x = -2, y = 5, z = -4)$$



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

$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

思考：如何求解f关于y的梯度?

复习：高数中的链式法则!

$$\frac{df}{dy} = \frac{df}{dq} * \frac{dq}{dy}$$

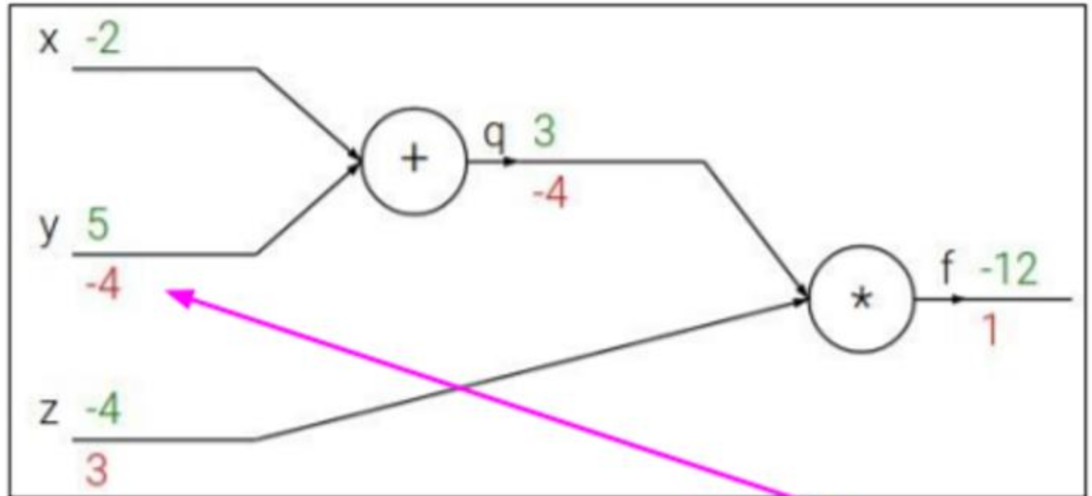
Upstream gradient　Local gradient

BP Sample 1

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

e.g. 若存在一个样本

$$(x=-2, y=5, z=-4)$$



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

$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

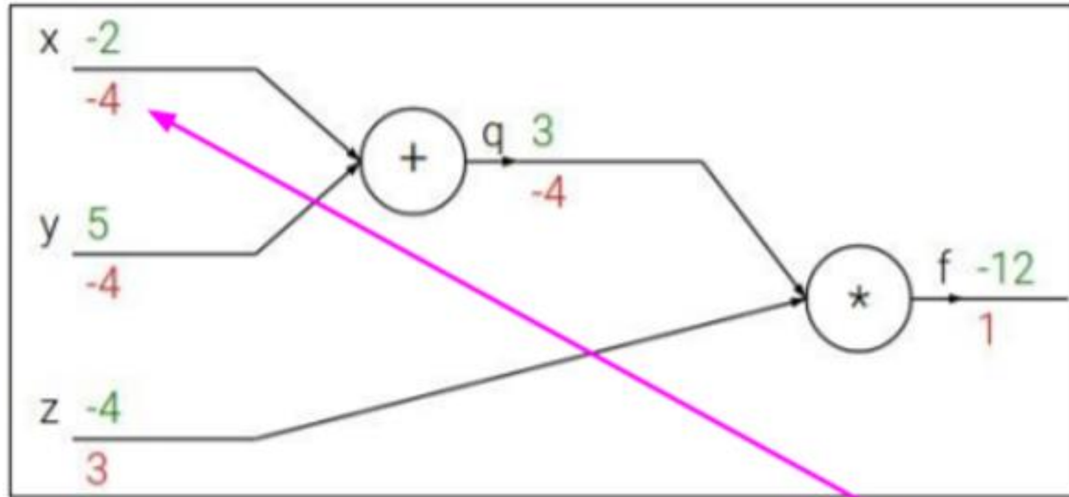
$$\frac{df}{dy} = \frac{df}{dq} * \frac{dq}{dy} = z$$

BP Sample 1

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

e.g. 若存在一个样本

$$(x = -2, y = 5, z = -4)$$



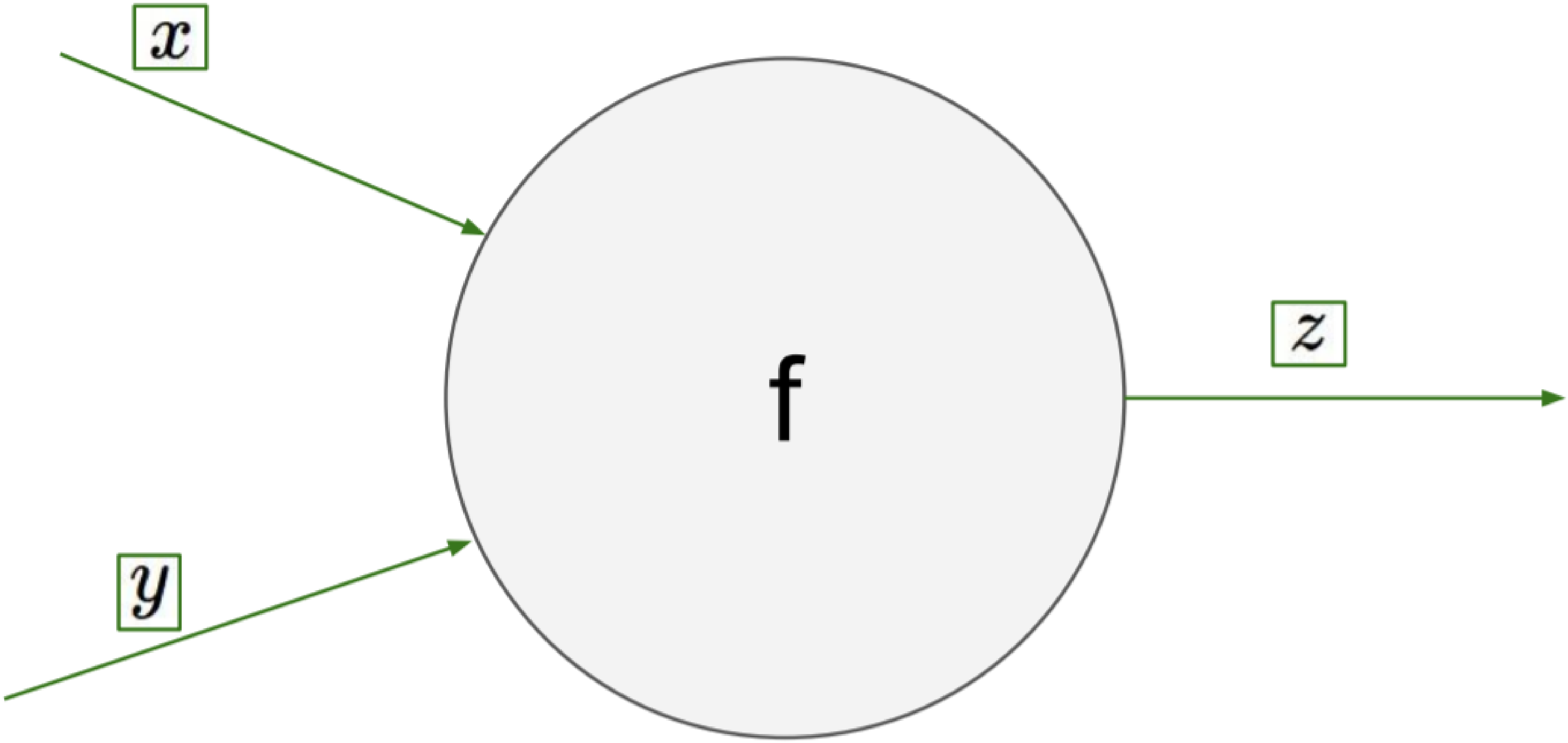
$$\frac{\partial f}{\partial x}$$

$$q = x + y \qquad \frac{dq}{dx} = 1, \frac{dq}{dy} = 1$$

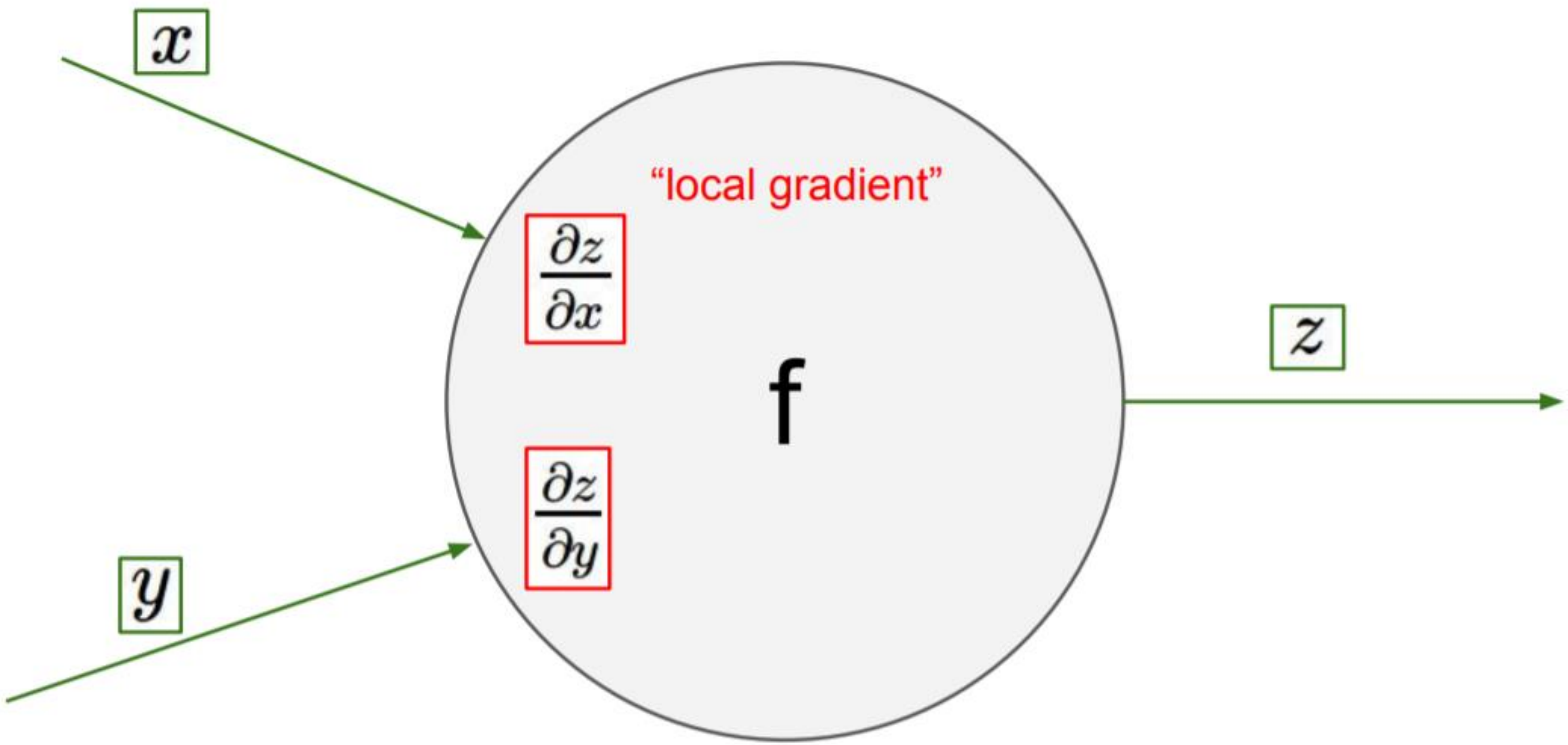
$$f = qz \qquad \frac{df}{dq} = z, \frac{df}{dz} = q$$

$$\frac{df}{dx} = \frac{df}{dq} * \frac{dq}{dx} = z$$

BP：单个神经元

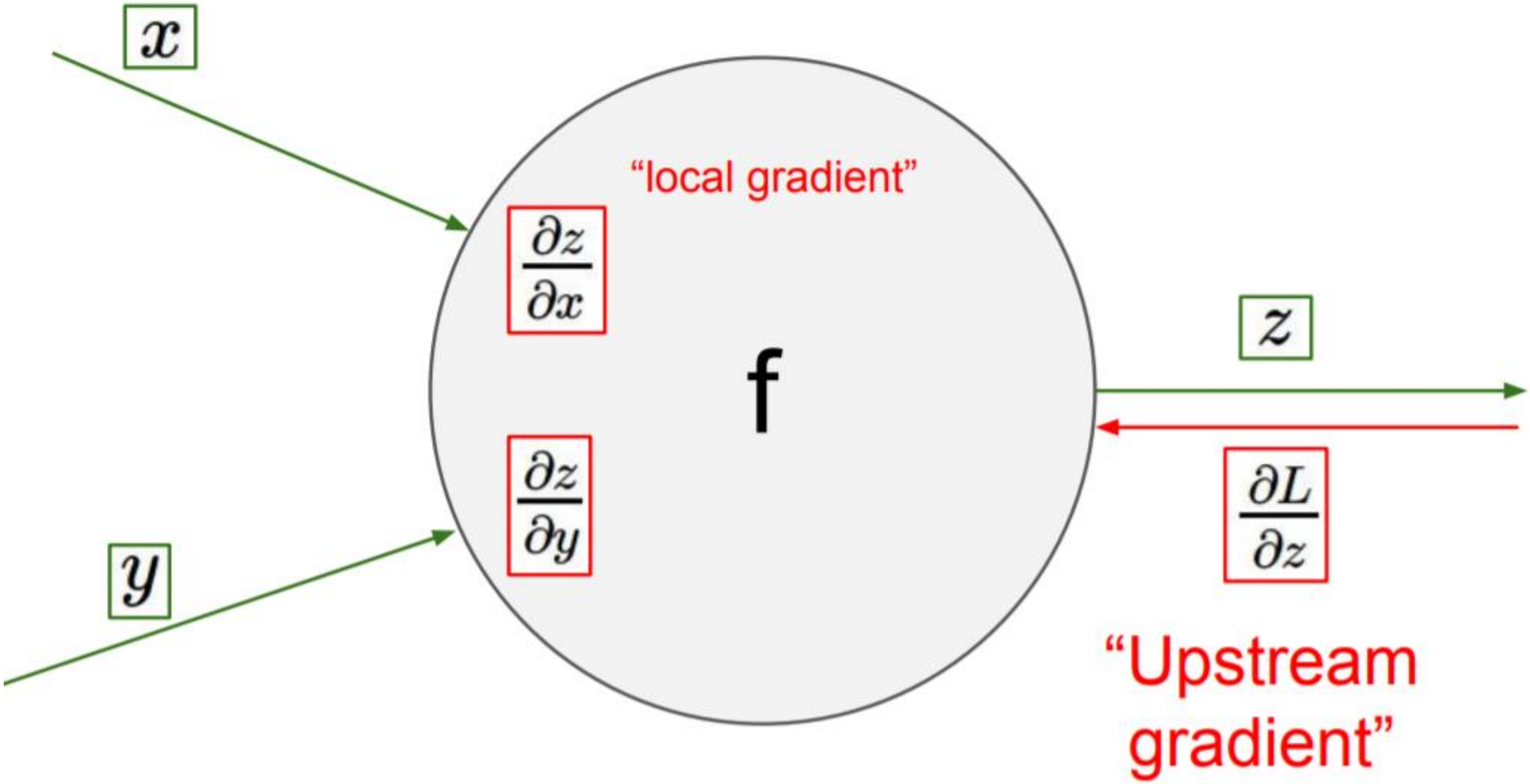


BP：单个神经元

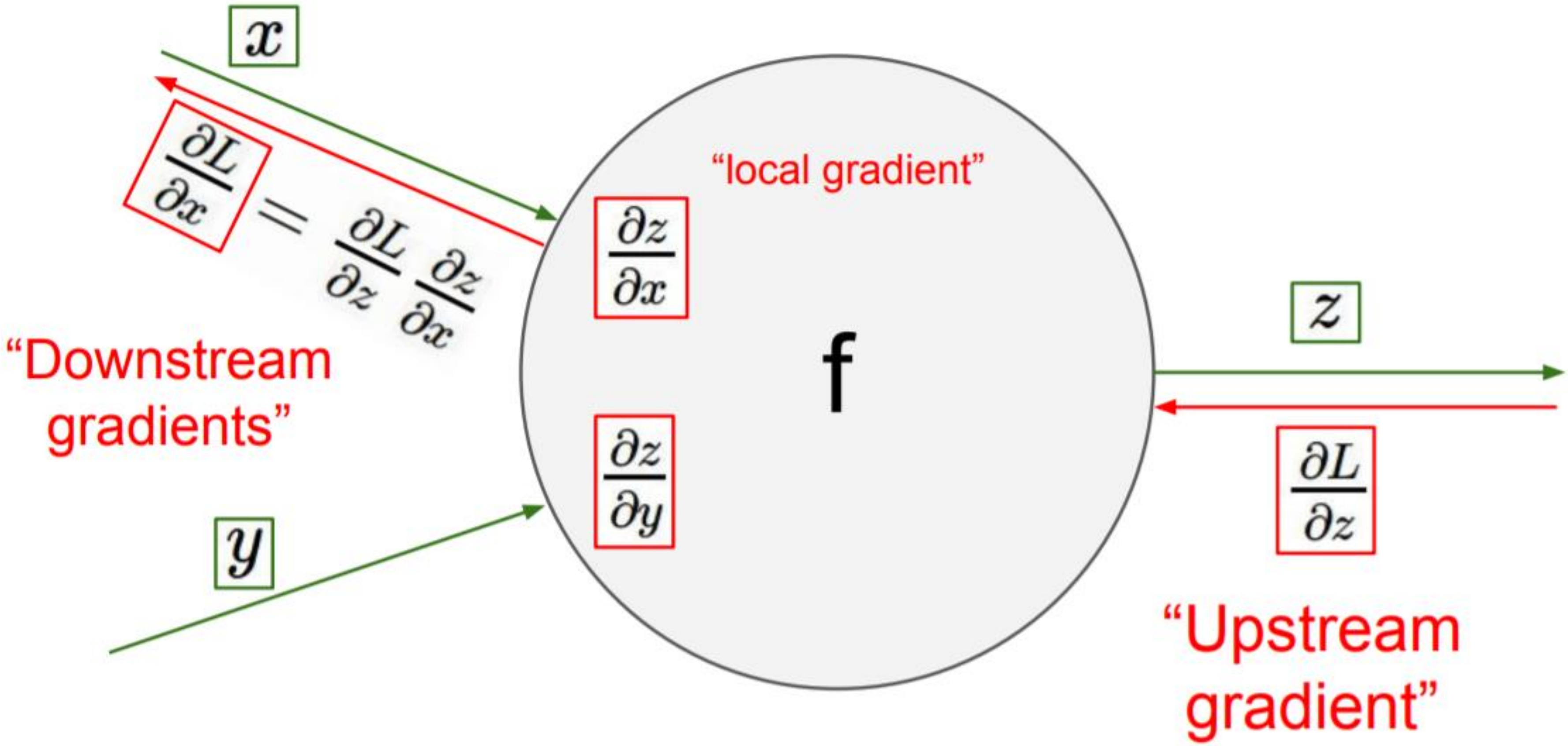


x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z

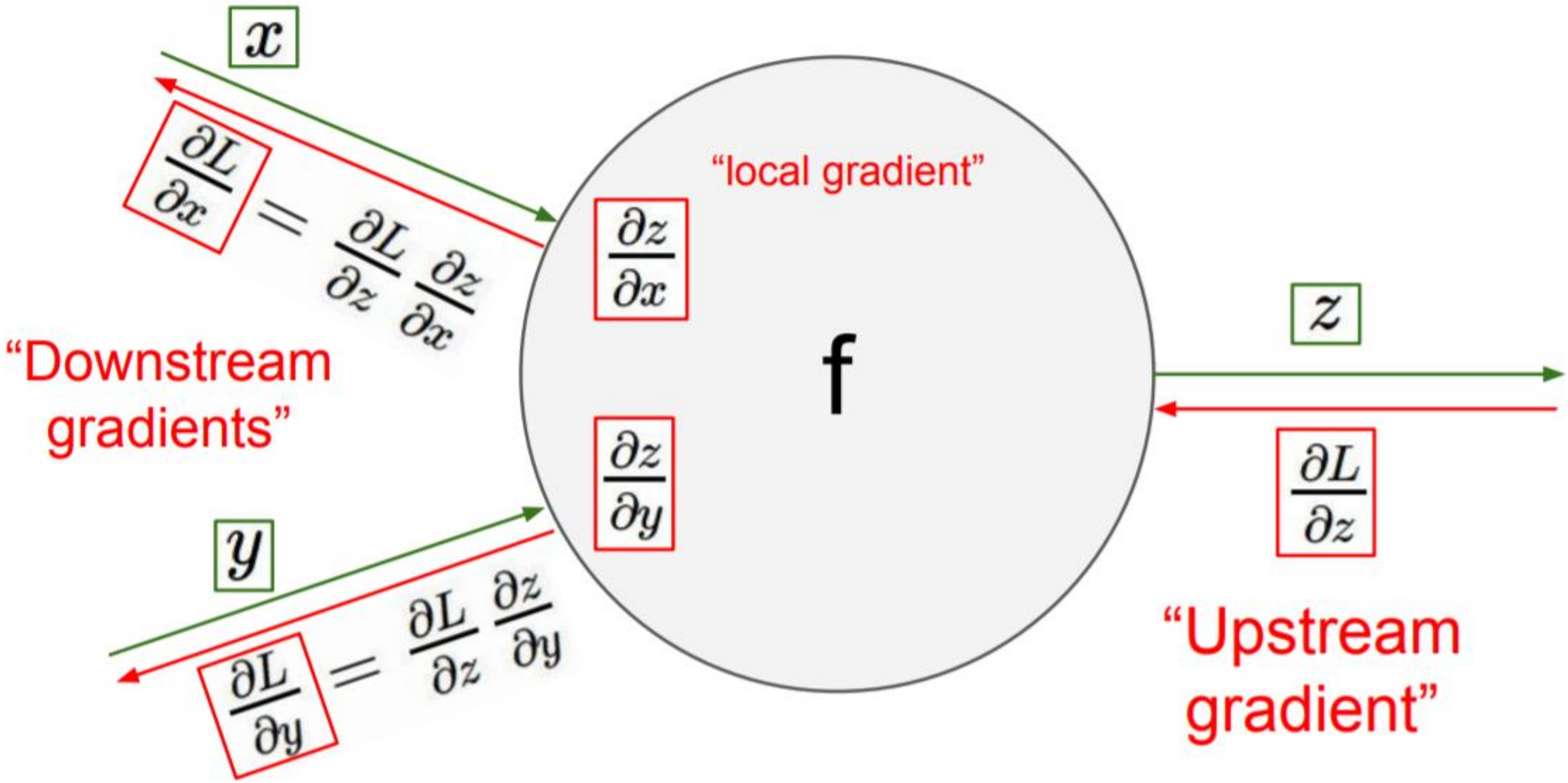
BP：单个神经元


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

- BP：单个神经元


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

BP：单个神经元



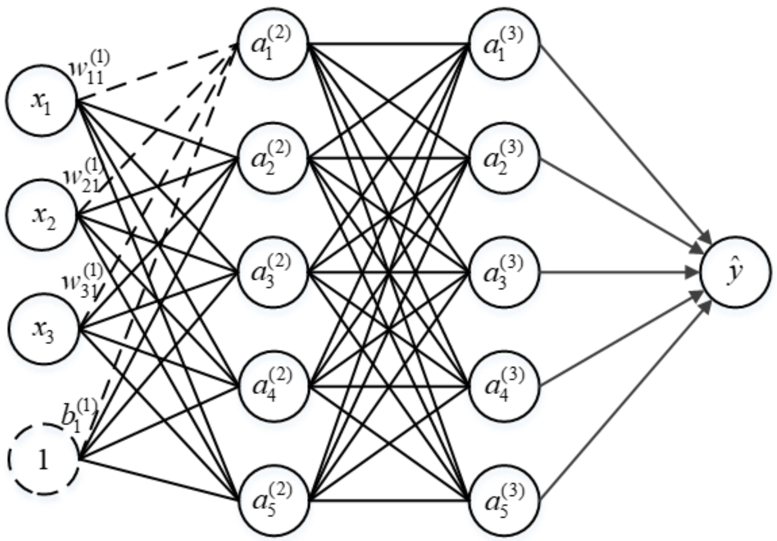
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
"Downstream gradients"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial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

11.3 反向传播

导出负梯度

反向传播主要是为了计算损失函数对于各层权重和偏差的梯度，为其更新做准备。由于损失函数$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$是定义在输出层上的，故各层的梯度需要从最后一层逐渐往前推算。

首先，我们回顾一下前向传播过程：第l层的广义输入为向量$\boldsymbol{z}^{(l)}$，神经元的输出为向量$\boldsymbol{a}^{(l)}$，权重矩阵为$\boldsymbol{w}^{(l)}$，偏差为$\boldsymbol{b}^{(l)}$，前向传播的矩阵形式表达为：



$$\boldsymbol{z}^{(l+1)} = \boldsymbol{w}^{(l),T}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}$$
$$\boldsymbol{a}^{(l+1)} = f\left(\boldsymbol{z}^{(l+1)}\right)$$

11.3 反向传播

导出负梯度

正向传播：

$$\boldsymbol{z}^{(l+1)} = \boldsymbol{w}^{(l),T}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l)}$$

$$\boldsymbol{a}^{(l+1)} = f\left(\boldsymbol{z}^{(l+1)}\right)$$

根据链式法则，损失函数对于该层权重$\boldsymbol{w}^{(l)}$和偏差$\boldsymbol{b}^{(l)}$的梯度为：

$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}^{(l)}} = \frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(l+1)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{w}^{(l)}} = \boldsymbol{a}^{(l)} \cdot \left(\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(l+1)}}\right)^{\cdot}$$

$$\frac{\partial \mathrm{L}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(l+1)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(l+1)}}$$

其中，$\boldsymbol{a}^{(l)}$已知。计算上述梯度的难点在于计算 $\partial \mathrm{L} / \partial \mathbf{z}^{(l+1)}$，这一项也被称为**第$l+1$层的误差项，记为$\boldsymbol{\delta}^{(l+1)}$**

11.3 反向传播

导出负梯度

第l $(l=1,\ldots,L-1)$ 层参数的梯度因此可以写作：

$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}^{(l)}} = \boldsymbol{a}^{(l)} \boldsymbol{\delta}^{(l+1),\bullet}$$

$$\frac{\partial \mathrm{L}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l+1)}$$

由于复杂的层级结构，对于中间的隐层，$\boldsymbol{z}$值（广义输入）与损失函数的关系非常复杂，因此其误差项的值也无法直接求得。而对于输出层，损失函数与其输入$\boldsymbol{z}^{(L)}$之间的函数关系是确定且简单的，如下式：

$$\mathrm{L} = g(\boldsymbol{a}^{(L)}) = g(f(\mathbf{z}^{(L)}))$$

其中，$g(\boldsymbol{a}^{(L)})$为表示损失函数的映射关系，$f(\cdot)$表示激活函数

11.3 反向传播

❑ 导出负梯度

以⊙表示哈达玛积（即逐元素相乘），那么第L层的误差项$\delta(L)$可以写作：

$$\boldsymbol{\delta}^{(L)}=\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L)}}=g'(\boldsymbol{a}^{(L)})\odot f'(\mathbf{z}^{(L)})=\frac{\partial \mathrm{L}}{\partial \boldsymbol{a}^{(L)}}\odot f'(\mathbf{z}^{(L)})$$

将误差项代入下式，可以得到我们能够容易得到最后一个隐层$L-1$层到输出层L层的权重$\boldsymbol{w}^{(l)}$和偏差$\boldsymbol{b}^{(l)}$的梯度。

$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}^{(l)}}=\boldsymbol{a}^{(l)}\boldsymbol{\delta}^{(l+1),\cdot}$$

$$\frac{\partial \mathrm{L}}{\partial \boldsymbol{b}^{(l)}}=\boldsymbol{\delta}^{(l+1)}$$

*哈达玛积（Hadamard Product）：两个结构一样的矩阵，对应元素相乘，构成的一个新矩阵（结构不变）。

11.3 反向传播

导出负梯度

为了继续获得损失函数对于$L-2$层参数的梯度，同样的，我们只需计算$\boldsymbol{\delta}^{(L-1)}$。根据链式法则：

$$\boldsymbol{\delta}^{(L-1)}=\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L-1)}}=\frac{\partial \mathrm{L}}{\partial \boldsymbol{a}^{(L-1)}}\cdot\frac{\partial \boldsymbol{a}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}}$$

上式中 $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L-1)}}$ 可以通过下式计算

$$\frac{\partial \mathrm{L}}{\partial \boldsymbol{a}^{(L-1)}}=\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L)}}\cdot\frac{\partial \mathbf{z}^{(L)}}{\partial \boldsymbol{a}^{(L-1)}}=\mathbf{w}^{(L-1),\mathrm{T}}\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L)}}=\mathbf{w}^{(L-1),\mathrm{T}}\boldsymbol{\delta}^{(L)}$$

其中，$\boldsymbol{w}^{(L-1)}$与$\boldsymbol{\delta}^{(L)}$均为已知

11.3 反向传播

❑ 导出负梯度

为了继续获得损失函数对于$L-2$层参数的梯度，同样的，我们只需计算$\boldsymbol{\delta}^{(L-1)}$。根据链式法则：

$$\boldsymbol{\delta}^{(L-1)}=\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L-1)}}=\frac{\partial \mathrm{L}}{\partial \boldsymbol{a}^{(L-1)}}\cdot\frac{\partial \boldsymbol{a}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}}$$

上式中 $\frac{\partial \boldsymbol{a}^{(L-1)}}{\partial \boldsymbol{z}^{(L-1)}}$ 为激活函数的导数：

$$\frac{\partial \boldsymbol{a}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}}=f'(\mathbf{z}^{(L-1)})$$

根据以上推导，得到

$$\boldsymbol{\delta}^{(L-1)}=\mathbf{w}^{(L-1),\mathrm{T}}\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L)}}\odot f'(\mathbf{z}^{(L)})=\mathbf{w}^{(L-1),\mathrm{T}}\boldsymbol{\delta}^{(L)}\odot f'(\mathbf{z}^{(L-1)})$$

11.3 反向传播

❑ 导出负梯度

计算得到$\boldsymbol{\delta}^{(L-1)}$后，将其代入下式，即可获得损失函数对于$L-2$层参数的梯度

$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}^{(l)}} = \boldsymbol{a}^{(l)} \boldsymbol{\delta}^{(l+1),\bullet}$$

$$\frac{\partial \mathrm{L}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l+1)}$$

对$\boldsymbol{\delta}^{(L-1)}$的推导可以拓展至任意相邻两层$l$与$l+1$：

$$\boldsymbol{\delta}^{(l)} = \mathbf{w}^{(l),\mathrm{T}} \boldsymbol{\delta}^{(l+1)} \odot f'(\mathbf{z}^{(l)}),\ l = 1, 2, 3, \cdots, L-1$$

11.3 反向传播

参数更新

在计算得到各层的误差项之后，可以基于误差项和负梯度信息对各层参数进行更新，将学习率（即步长）记为α，参数更新方式如下：

$$\mathbf{w}^{(l)} = \mathbf{w}^{(l)} - \alpha \cdot \boldsymbol{a}^{(l)} \boldsymbol{\delta}^{(l+1),\bullet}$$

$$\boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)} - \alpha \cdot \boldsymbol{\delta}^{(l+1)}$$

11.3 反向传播

BP算法流程

假设神经网络层数为n：

输入：训练集$D=\{x_1,x_2,\ldots,x_m\}$，学习率α。

输出：权重及偏差确定的神经网络

（1）对各层权重矩阵$\boldsymbol{w}^{(l)}$及偏差向量$\boldsymbol{b}^{(l)}$进行初始化；

（2）对样本(x,y)根据下式进行前向传播计算，得到估计值$\hat{y}$；

$$\mathbf{z}^{(l+1)}=\mathbf{w}^{(l),\mathrm{T}}\boldsymbol{a}^{(l)}+\boldsymbol{b}^{(l)},\ l=1,2,\ldots,L-1$$

$$\boldsymbol{a}^{(l+1)}=f(\mathbf{z}^{(l+1)}),\ l=1,2,\ldots,L-1$$

（3）计算损失函数$Loss(y,\hat{y})$，及输出层误差项

$$\boldsymbol{\delta}^{(L)}=\frac{\partial \mathrm{L}}{\partial \mathbf{z}^{(L)}}=g'(\boldsymbol{a}^{(L)})\odot f'(\mathbf{z}^{(L)})=\frac{\partial \mathrm{L}}{\partial \boldsymbol{a}^{(L)}}\odot f'(\mathbf{z}^{(L)})$$

11.3 反向传播

❑ BP算法流程

假设神经网络层数为n：

输入：训练集$D=\{x_1,x_2,\ldots,x_m\}$，学习率α 。

输出：权重及偏差确定的神经网络

（4）逐层计算各层误差项；

$$\boldsymbol{\delta}^{(l)}=\mathbf{w}^{(l),\mathrm{T}}\boldsymbol{\delta}^{(l+1)}\odot f'(\mathbf{z}^{(l)}),\ l=1,2,3,\cdots,L-1$$

对于最后一层即输出层L，误差项为：

$$\boldsymbol{\delta}^{(L)}=\boldsymbol{a}^{(L)}-\boldsymbol{y}$$

其中，$\boldsymbol{y}$为真实标签值。

11.3 反向传播

BP算法流程

（5）逐层计算各层权重及偏差的梯度；

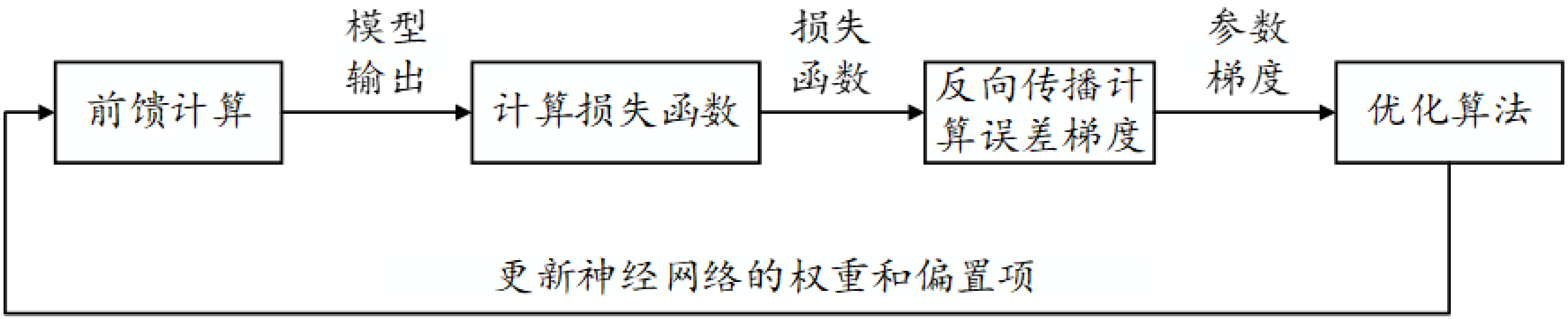
$$\frac{\partial \mathrm{L}}{\partial \mathbf{w}^{(l)}} = \boldsymbol{a}^{(l)} \boldsymbol{\delta}^{(l+1),\bullet} \;, \frac{\partial \mathrm{L}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l+1)}$$

（6）对权重及偏差进行更新；

$$\mathbf{w}^{(l)} = \mathbf{w}^{(l)} - \alpha \cdot \boldsymbol{a}^{(l)} \boldsymbol{\delta}^{(l+1),\bullet} \;, \boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)} - \alpha \cdot \boldsymbol{\delta}^{(l+1)}$$

（7）对新的样本重复上面的第（2）到（6）步。

11.3 反向传播

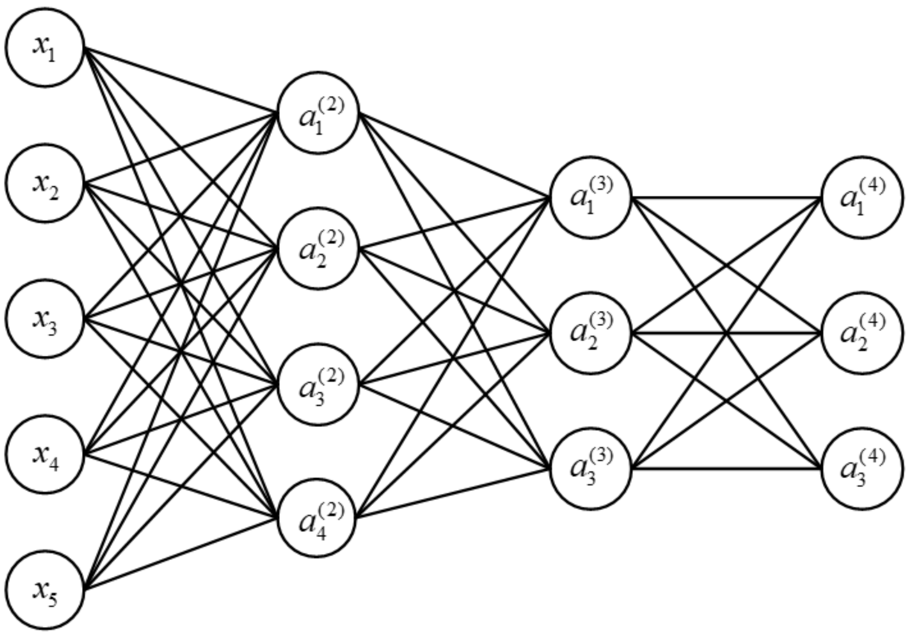


通过**多次**前向传播与反向传播，对参数进行更新，获得最优参数值。

神经网络内参数的更新过程，就是神经网络的“**学习**”过程。神经网络“学习”到的知识，就蕴含在模型训练后得到的所有参数中。一旦模型训练完成，神经网络的**各项参数就固定下来**。在神经网络的预测过程中，权重和偏差项将保持不变，仅通过对输入值的前向传播就可以得到对应的预测值。

反向传播代码实现

- 解决问题：交通状态的预测（三分类），和**正向传播案例一致**
- 特征工程：
  - aveSpeed平均车速
  - gridAcc平均加速度
  - volume流量
  - speed_std速度标准差
  - stopNum平均停车次数
- 模型结构
  - 输入层：5个神经元
  - 两个隐藏层使用Sigmoid激活函数
  - 输出层使用Softmax激活函数



试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 1. 实现sigmoid函数的导数

$$\sigma'(x) = \sigma(x) \cdot \sigma(1-x)$$

```
def sigmoid_grad(z):# sigmoid激活函数的导数
    pass
```

```
def sigmoid_grad(z):# sigmoid激活函数的导数
    return np.multiply(sigmoid(z), (1 - sigmoid(z)))
```

试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 2. 定义计算各层误差项

$$\delta^{(4)} = a^{(4)} - y$$

```
def delta_output(a, y): # 计算输出层误差项
    pass
```

$$\delta^{(l)} = \mathbf{w}^{(l),\mathrm{T}} \delta^{(l+1)} \odot f'(\mathbf{z}^{(l)}),\ l = 1, 2, 3, \cdots, L-1$$

```
def delta_hidden(delta, w, z, act_grad_func):# 计算隐藏层误差项
    # act_grad_func 激活函数的导数
    pass
```

```
def delta_output(a, y):# 计算输出层误差项
    def onehot(y, n_class):
        return np.array([np.eye(n_class)[yi] for yi in y])
    return a - onehot(y, 3).T

def delta_hidden(delta, w, z, act_grad_func):# 计算隐藏层误差项
    act_grad =  act_grad_func(z)    # 激活函数的导数
    return np.multiply(w.T @ delta, act_grad)
```

试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 3. 定义隐藏层梯度

$$\mathbf{w}^{(l)} = \mathbf{w}^{(l)} - \alpha \cdot \boldsymbol{a}^{(l)} \boldsymbol{\delta}^{(l+1),\top}$$

$$\boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)} - \alpha \cdot \boldsymbol{\delta}^{(l+1)}$$

```
def hidden_grad(delta, a, alpha): # 计算隐藏层权重w梯度
    pass

def hidden_bias_grad(delta, alpha):  # 计算隐藏层p偏差b梯度
    pass
```

```
 # 隐藏层梯度
def hidden_grad(delta, a, alpha): # 计算隐藏层权重w梯度
    return alpha * (delta @ a.T)
def hidden_bias_grad(delta, alpha):  # 计算隐藏层p偏差b梯度
    return alpha * np.mean(delta, axis=1, keepdims=True)
```

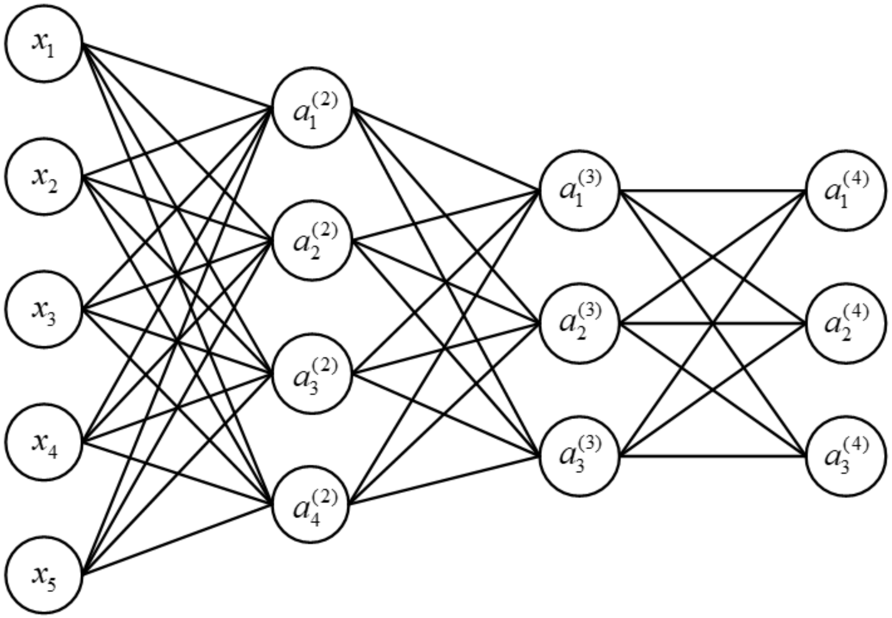
试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 4. 实现反向传播过程

```
def delta_output(a, y)# 计算输出层误差项
def delta_hidden(delta, w, z, act_grad_func)#计算隐藏层误差项
def hidden_grad(delta, a, alpha)# 计算隐藏层权重w梯度
def hidden_bias_grad(delta, alpha)# 计算隐藏层p偏差b梯度
```

```
def backward(x, y, w, a_list,
             z_list, alpha=0.02):
    pass
```



试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 4. 实现反向传播过程

```
def backward(x, y, w, a_list, z_list, alpha=0.02):
    delta4 = delta_Output(a_list[2], y)  # 第4层，输出层误差项
    #第3/2层误差项
    delta3 = delta_hidden(delta4, w[2], z_list[1], sigmoid_grad)
    delta2 = delta_hidden(delta3, w[1], z_list[0], sigmoid_grad)
    # 参数更新量的计算
    w_grad_3 = hidden_grad(delta4, a_list[1], alpha)# 第3层权重梯度*学习率
    b_grad_3 = hidden_bias_grad(delta4, alpha) # 第3层偏置梯度*学习率
    w_grad_2 = hidden_grad(delta3, alist[0], alpha)#第2层权重梯度*学习率
    b_grad_2 = hidden_bias_grad(delta3, alpha) # 第2层偏置梯度*学习率
    w_grad_1 = hidden_grad(delta2, x, alpha) # 第1层权重梯度*学习率
    b_grad_1 = hidden_bias_grad(delta2, alpha) # 第1层偏置梯度*学习率
    return [w_grad_1, w_grad_2, w_grad_3], [b_grad_1, b_grad_2, b_grad_3]
```

试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 5. 优化步骤

```
def optimize(w, b, w_grad, b_grad):
    pass
    return w, b
```

```
def optimize(w, b, w_grad, b_grad):
    for i in range(len(w)):
        w[i] -= w_grad[i]
        b[i] -= b_grad[i]
    return w, b
```

试一试

试仅基于numpy库，一步一步实现该模型反向传播过程

Step 6. 模型优化

```
x = np.array([[4, 0, 2, 3, 3]]).T
y = np.array([1])
# 输入层(5)-隐藏层(4)-输出层(3)
shape_w = [(5, 4), (4, 3), (3, 3)]     # 各层权重矩阵大小
shape_b = [(4, 1), (3, 1), (3, 1)]     # 各层偏差矩阵大小
w, b = init_params(shape_w, shape_b)   #初始化权重和偏差
# 更新50步
for _ in range(50):
    a, z = forward(x, w, b)
    w_grad, b_grad = backward(x, y, w, a, z, 0.01) #反向传播
    w, b = optimize(w, b, w_grad, b_grad)           # 优化参数
```

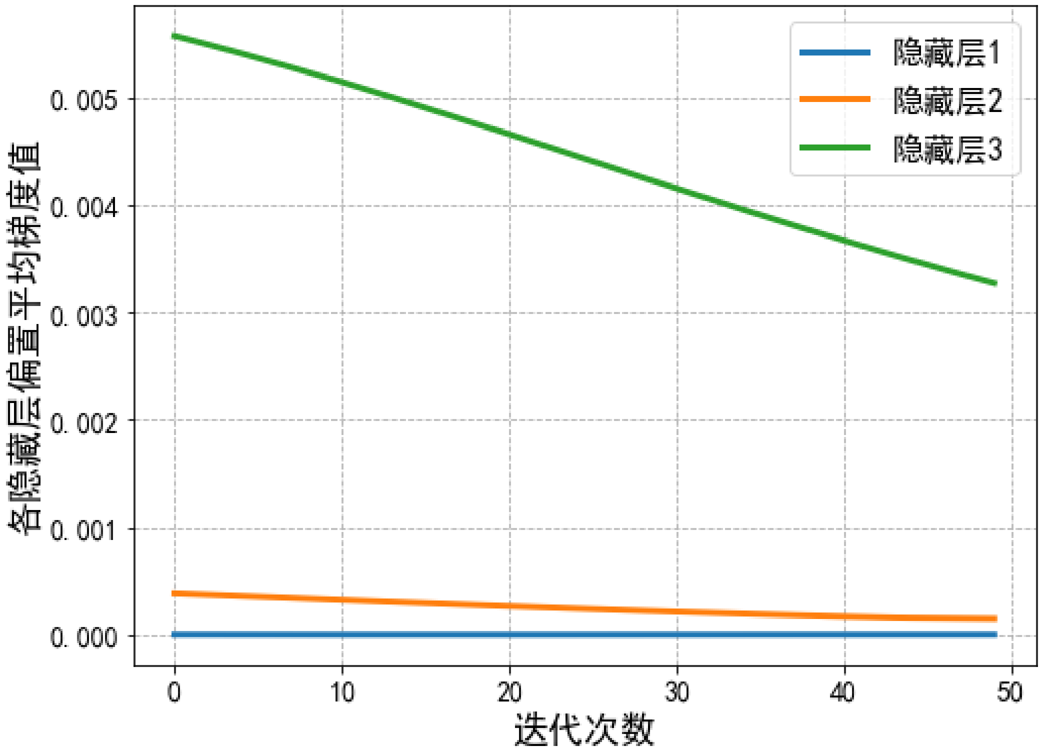
迭代50次后，各类的预测值如下：

[[0.45860661] [0.08641333][0.37609855]]

11.4 神经网络中的常见问题解决方案

梯度消失与梯度爆炸

梯度消失：在反向传播的过程中，参数关于损失函数的梯度，可能会随着神经网络层数增加，而逐渐减少至零。



11.4 神经网络中的常见问题解决方案

❑ 梯度消失与梯度爆炸

梯度爆炸：在实际案例中，反向传播中的梯度变化是不稳定的，在计算过程中，梯度不仅可能会逐渐消失，也有可能会激增，这被称为**梯度爆炸**。

梯度的不稳定变化是深度学习中的重要瓶颈。

思考：

梯度消失对于模型训练的影响是什么？

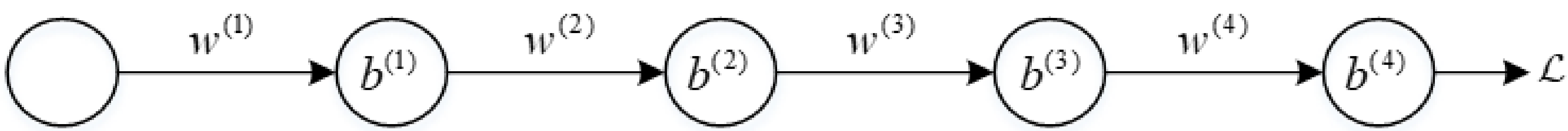
梯度爆炸呢？

$$w = w - \alpha \cdot \Delta \boldsymbol{w}$$

11.4 神经网络中的常见问题解决方案

❑ 梯度消失与梯度爆炸

例子：如下所示的简单神经网络，采用的激活函数为Sigmoid函数(σ)：



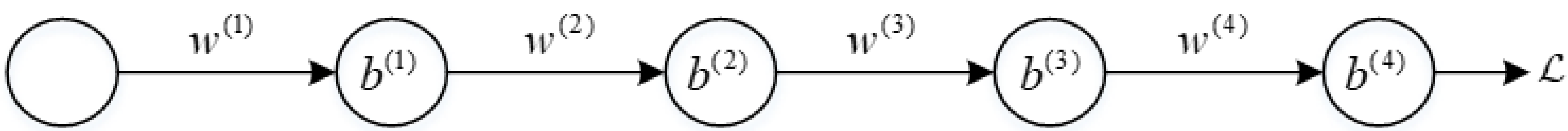
其中$w^{(l)}$和$b^{(l)}$表示第l层的可学习权重与偏差，$z^{(l)}$表示第l层的广义输入，$a^{(l)}$表示第l层的输出，$z^{(l+1)} = w^{(l)}a^{(l)} + b^{(l)}$，且$a^{(l+1)} = \sigma\left(z^{(l+1)}\right)$，则有：

$$\frac{\partial L}{\partial b^{(1)}} = \sigma'(z^{(1)})w^{(2)}\sigma'(z^{(2)})w^{(3)}\sigma'(z^{(3)})w^{(4)}\sigma'(z^{(4)})\frac{\partial L}{\partial a^{(4)}}$$

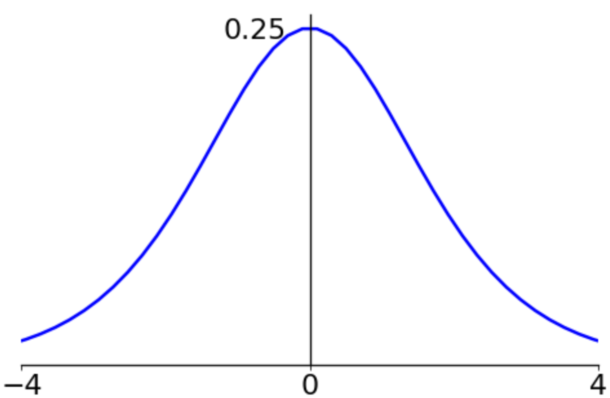
11.4 神经网络中的常见问题解决方案

❑ 梯度消失与梯度爆炸

例子：如下所示的简单神经网络，采用的激活函数为Sigmoid函数：



$$\frac{\partial \mathrm{L}}{\partial b^{(1)}} = \sigma'(z^{(1)})w^{(2)}\sigma'(z^{(2)})w^{(3)}\sigma'(z^{(3)})w^{(4)}\sigma'(z^{(4)})\frac{\partial \mathrm{L}}{\partial a^{(4)}}$$



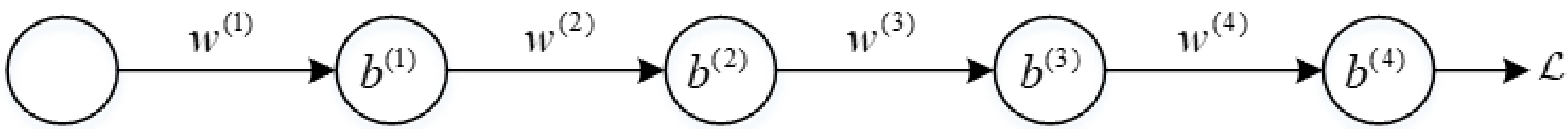
$$\sigma'(x) = \sigma(x)[1-\sigma(x)]$$

如左图所示，Sigmoid函数的导数最大值仅0.25。在初始化参数值时，若服从均值为0且标准差为1的正态分布，则往往$|w^{(l)}\sigma'\left(z^{(l)}\right)| < 1$，随着层数的增加，这些导数乘积会越来越小，最终趋近于0，导致梯度消失。

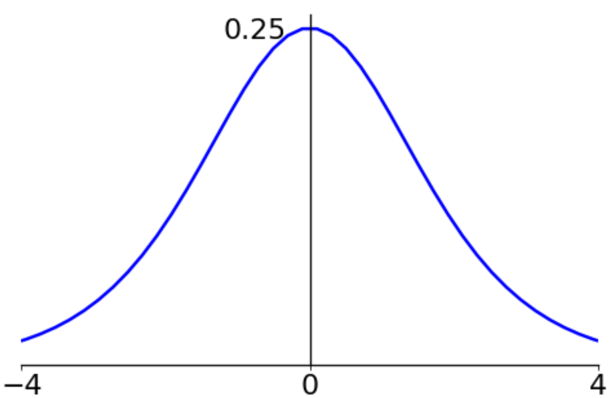
11.4 神经网络中的常见问题解决方案

❑ 梯度消失与梯度爆炸

例子：如下所示的简单神经网络，采用的激活函数为Sigmoid函数：



$$\frac{\partial L}{\partial b^{(1)}} = \sigma'(z^{(1)})w^{(2)}\sigma'(z^{(2)})w^{(3)}\sigma'(z^{(3)})w^{(4)}\sigma'(z^{(4)})\frac{\partial L}{\partial a^{(4)}}$$



$$\sigma'(x) = \sigma(x)[1-\sigma(x)]$$

但权重取值也可能会比较大，即 $|w^{(l)}\sigma'(z^{(l)})| >> 1$ ，如果多个权重较大的数值相乘，会导致导数越来越大，造成梯度爆炸，但（从概率上判断）此种情况较为少见。

11.4 神经网络中的常见问题解决方案

❑ 梯度消失与梯度爆炸的解决方案

- 更换激活函数：比如使用ReLU替代Sigmoid函数。由于ReLU对于输入小于0时输出也为0，会舍弃一部分神经元，因此可以采用改进的激活函数如Leaky ReLU，使得负区间也有梯度传播。
- 梯度剪切：设定一定阈值， 更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在阈值以下
- 权重正则化：正则化可以降低网络中各层权重值。如果发生梯度爆炸，那么权值就会变的非常大，反过来，通过正则化项来限制权重的大小，也可以在一定程度上防止梯度爆炸的发生。

$$L_{sum} = loss + \lambda \sum_i |w_i|^2$$

上式给出的是L2正则化，由两部分构成，$loss$ 是任务原本的损失函数，$\sum_i |w_i|^2$ 是所有可学习参数的模的平方和，λ 是正则化系数

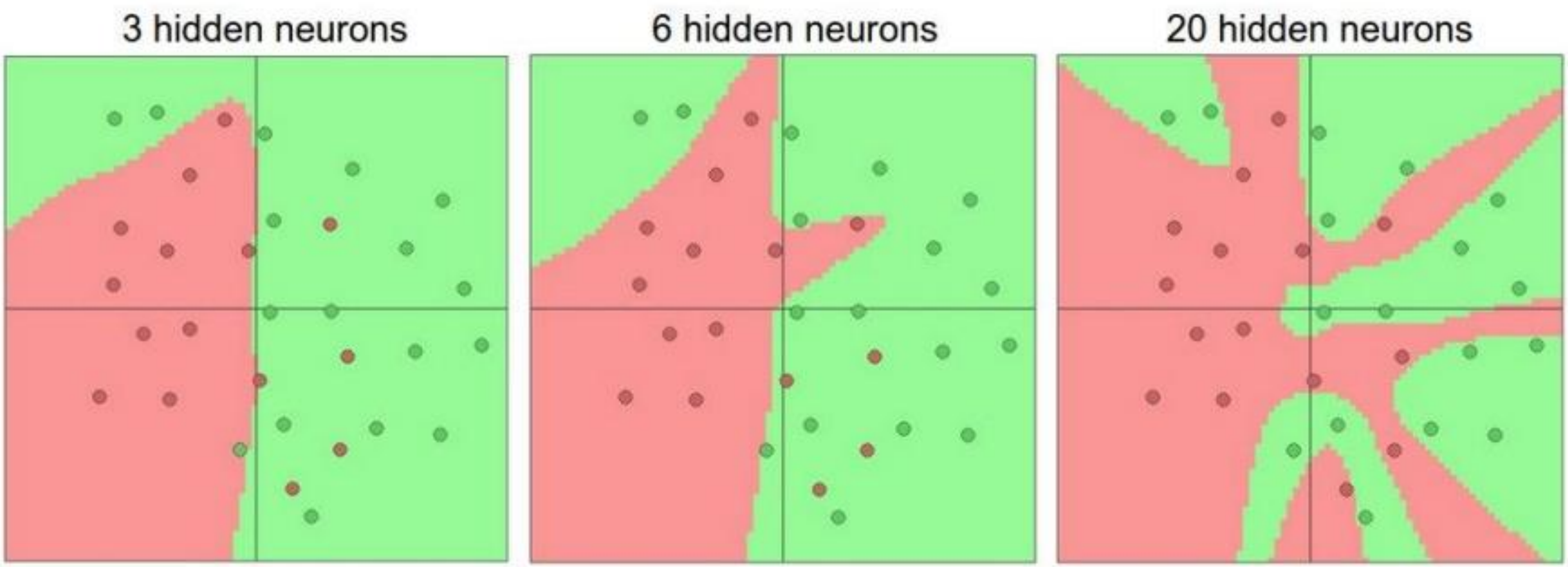
11.4 神经网络中的常见问题解决方案

❑ 过拟合问题

思考：

神经网络模型的泛化能力

是否随着神经元数量的提升而提升？

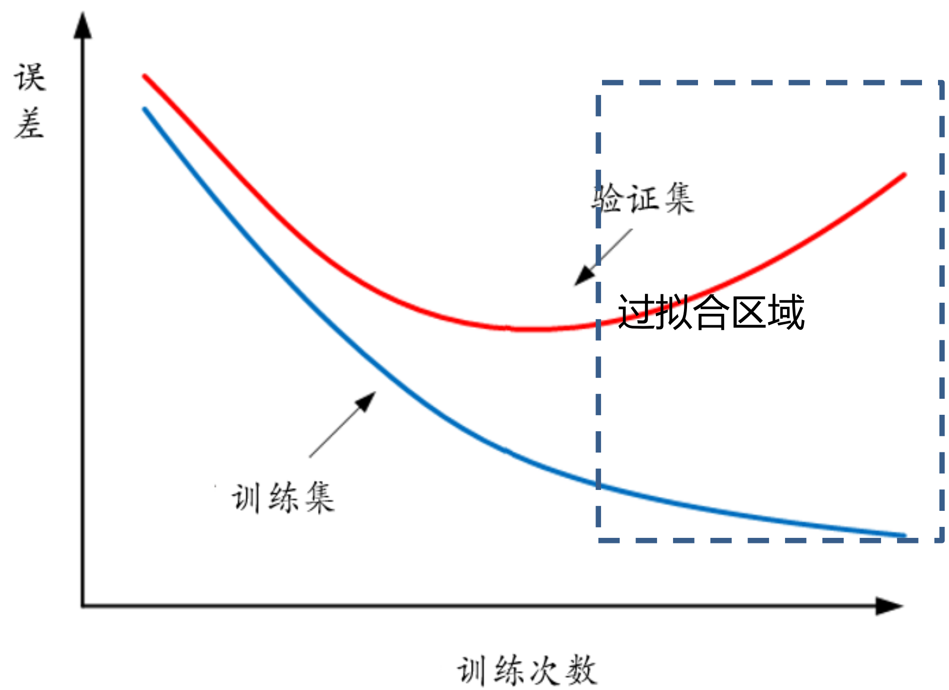


上面三个模型哪个表现最好？

11.4 神经网络中的常见问题解决方案

❑ 过拟合问题

神经网络模型因为其结构的复杂性，能够对训练数据进行很好的拟合，在数据量不足的情况下，模型更倾向于“背”数据，随着训练次数的增长，模型对训练数据的学习效果过好，容易产生过拟合的情况。



11.4 神经网络中的常见问题解决方案

❑ 过拟合问题的解决方法

✓ 增加数据量

高质量数据量提升总是利于模型训练的。参数较多的模型被认为具有较高的容量，为了获得对未知测试数据的泛化能力，模型需要更多的数据。增加数据量的方法包括使用额外引入的数据集、或通过数据增强等方式生成新数据。

✓ 正则化

将模型复杂度的指标加到损失函数中，以一定程度上限制模型变得过于复杂，从而避免过拟合。正则化会使模型倾向于更简单（参数值不高）。

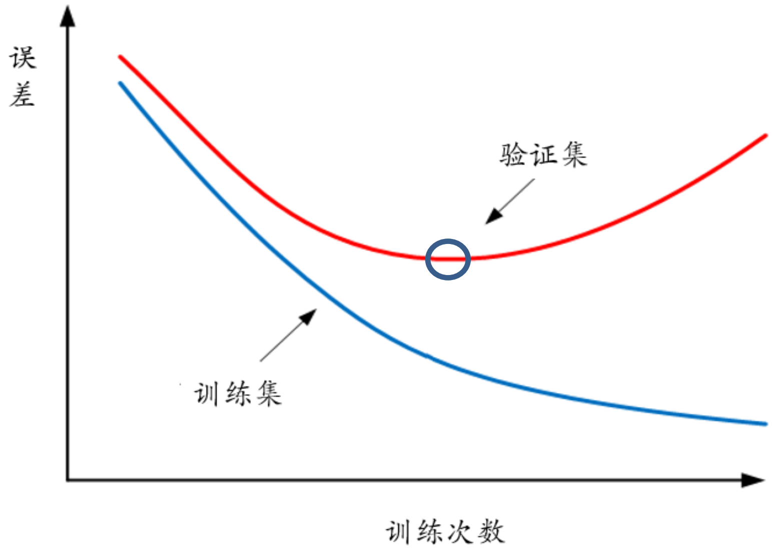
$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})+\boxed{\frac{\lambda}{2m}||w||_2^2}$$ L2正则化

11.4 神经网络中的常见问题解决方案

❑ 过拟合问题的解决方法

✓ 早停机制（early-stopping）

当验证集误差已经达到最小值时，继续训练模型误差虽会继续减小，但是会导致过拟合。因此，我们可以在验证集误差由最小值开始变大时，提前停止训练的策略（即增加一个停止判定规则），即“早停”。早停能够有效防止过拟合，但不适当的使用也会导致模型的训练不充分。

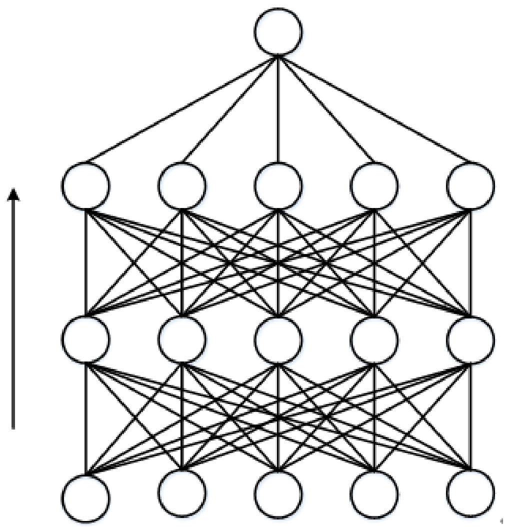


11.4 神经网络中的常见问题解决方案

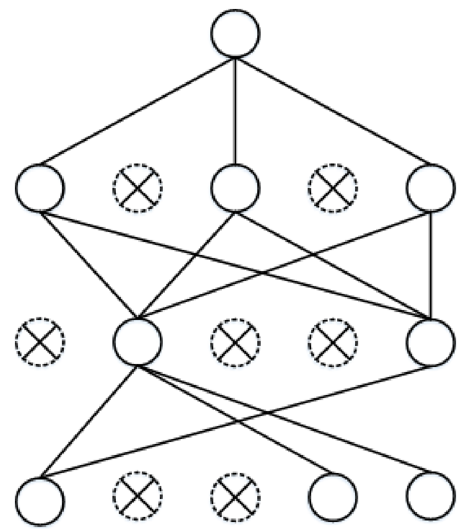
❑ 过拟合问题的解决方法

✓ dropout策略

Dropout在训练过程中按照给定的概率p**随机删除**一些隐藏层的神经元（输入层和输出层的神经元不变），只更新没有被删除的神经元参数，通过减少神经元个数的方式防止过拟合，随机性的引入提高模型鲁棒性。



标准神经网络



使用dropout的神经网络

❑ 过拟合问题

✓ 一个有经验的算法工程师通常在模型的正式训练之前，会在一个很小的数据集上进行多轮训练直至过拟合。

思考：

为什么有些时候算法工程师在“寻求”过拟合?