

《神经网络与深度学习》



前馈神经网络

<https://nndl.github.io/>

内容

▶ 神经网络

- ▶ 神经元

- ▶ 网络结构

▶ 前馈神经网络

- ▶ 参数学习

- ▶ 计算图与自动微分

- ▶ 优化问题



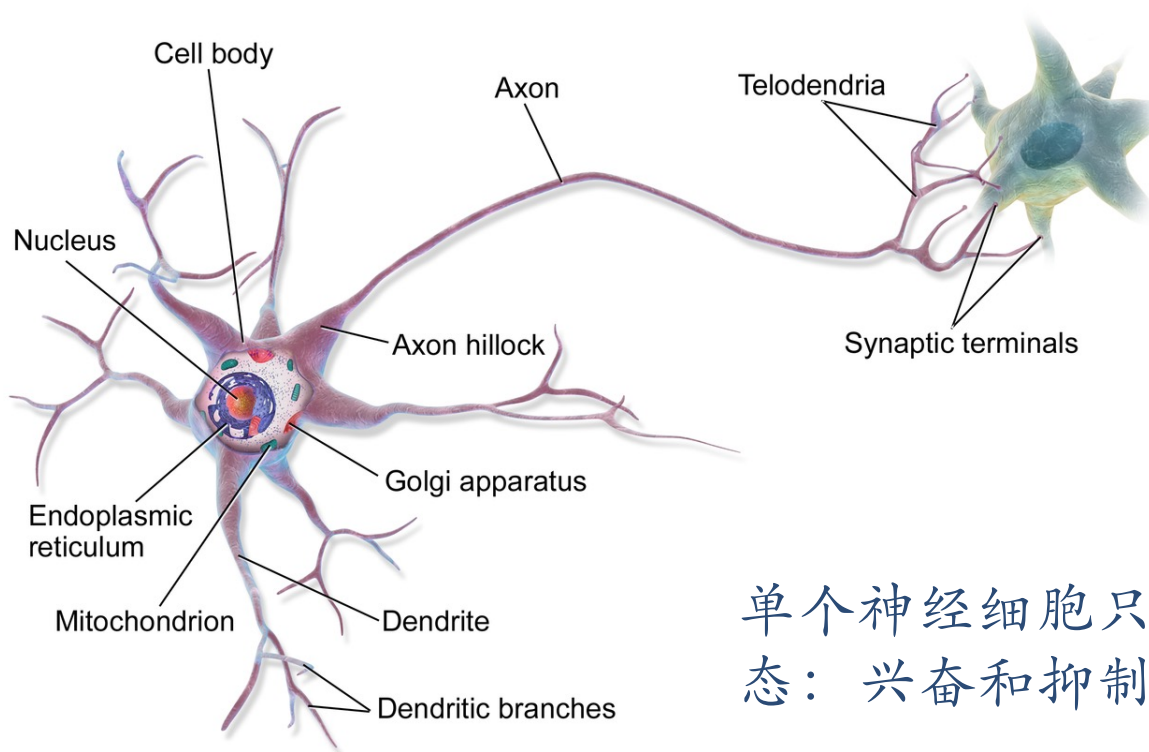
神经元

神经网络

- ▶ **神经网络最早是作为一种主要的连接主义模型。**
- ▶ 20世纪80年代后期，最流行的一种连接主义模型是分布式并行处理（Parallel Distributed Processing, PDP）网络，其有3个主要特性：
 - ▶ 1) 信息表示是分布式的（非局部的）；
 - ▶ 2) 记忆和知识是存储在单元之间的连接上；
 - ▶ 3) 通过逐渐改变单元之间的连接强度来学习新的知识。
- ▶ 引入误差反向传播来改进其学习能力之后，神经网络也越来越多地应用在各种机器学习任务上。

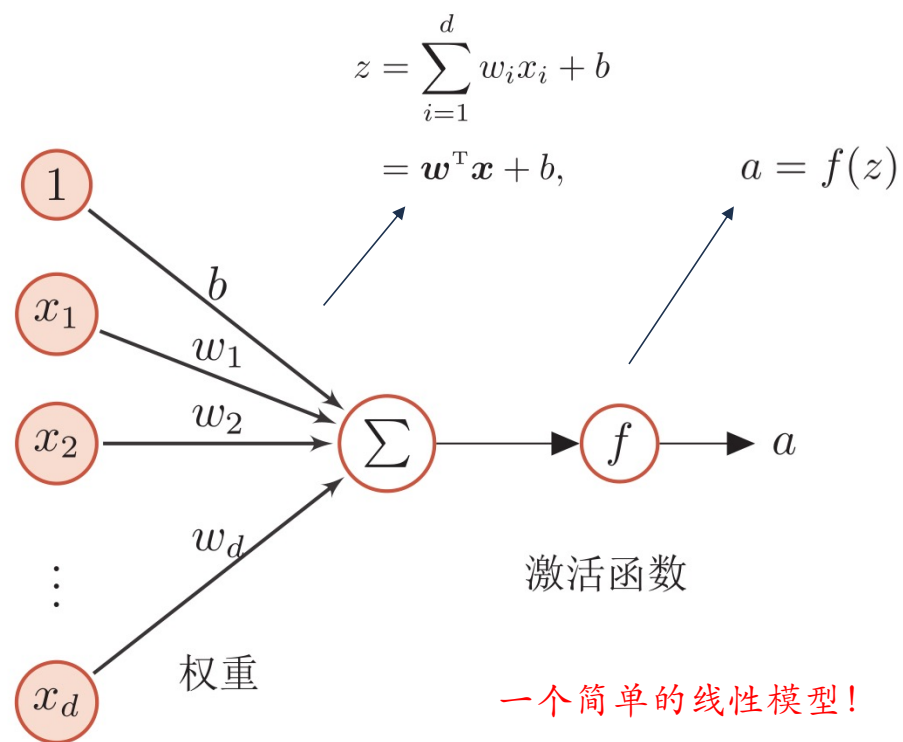
生物神经元

[video: structure of brain](#)



单个神经细胞只有两种状态：兴奋和抑制

人工神经元



激活函数的性质

- ▶ 连续并可导（允许少数点上不可导）的非线性函数。
 - ▶ 可导的激活函数可以直接利用数值优化的方法来学习网络参数。
- ▶ 激活函数及其导函数要尽可能的简单
 - ▶ 有利于提高网络计算效率。
- ▶ 激活函数的导函数的值域要在一个合适的区间内
 - ▶ 不能太大也不能太小，否则会影响训练的效率 and 稳定性。
- ▶ 单调递增
 - ▶ ???

Can non-monotonic activation function neural networks be trained using the same backpropagation algorithm as monotonic activation function neural networks?



Yoshua Bengio, My lab has been one of the three that started the deep learning approach, back in 2006, along with Hinton's a...

Answered 7 years ago · Author has 170 answers and 4.2M answer views

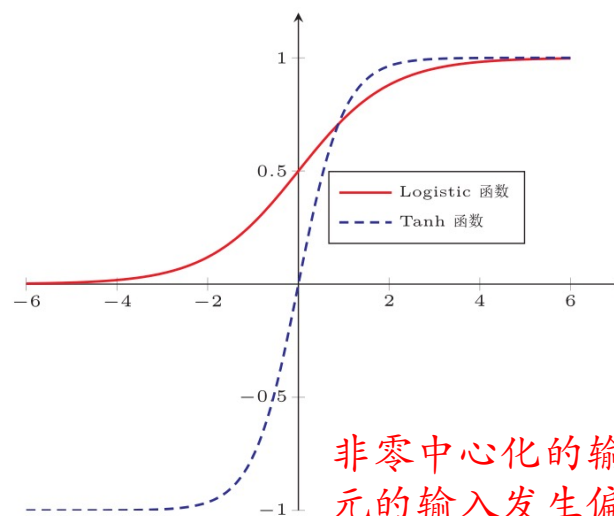
<https://www.quora.com/Can-non-monotonic-activation-function-neural-networks-be-trained-using-the-same-backpropagation-algorithm-as-monotonic-activation-function-neural-networks>

常见激活函数

Sigmoid型函数

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

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



非零中心化的输出会使得其下一层的神经元的输入发生偏置偏移 (bias shift)，并进一步使得梯度下降的收敛速度变慢。

►性质：

- 饱和函数(当 x 趋近于正无穷或者负无穷，其导数 $f'(x) = 0$.)
- Tanh函数是零中心化的，而logistic函数的输出恒大于0

当logistic函数的输出恒大于0

- ▶ 为什么对于logistic函数，使用梯度下降优化 w 时，如果 x 恒大于0，其收敛速度会比零均值化的输入更慢？

$$L = f(\sigma(w^T x), y^*)$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \sigma} \sigma(1 - \sigma) x$$

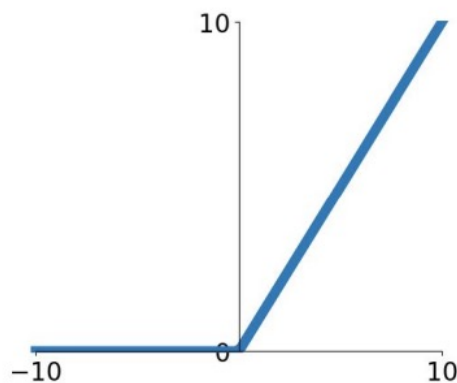
- ▶ 解决办法：

- ▶ 加入batch normalization层，强制输入在0的附近

常见激活函数

ReLU(Rectified Linear Unit)函数

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



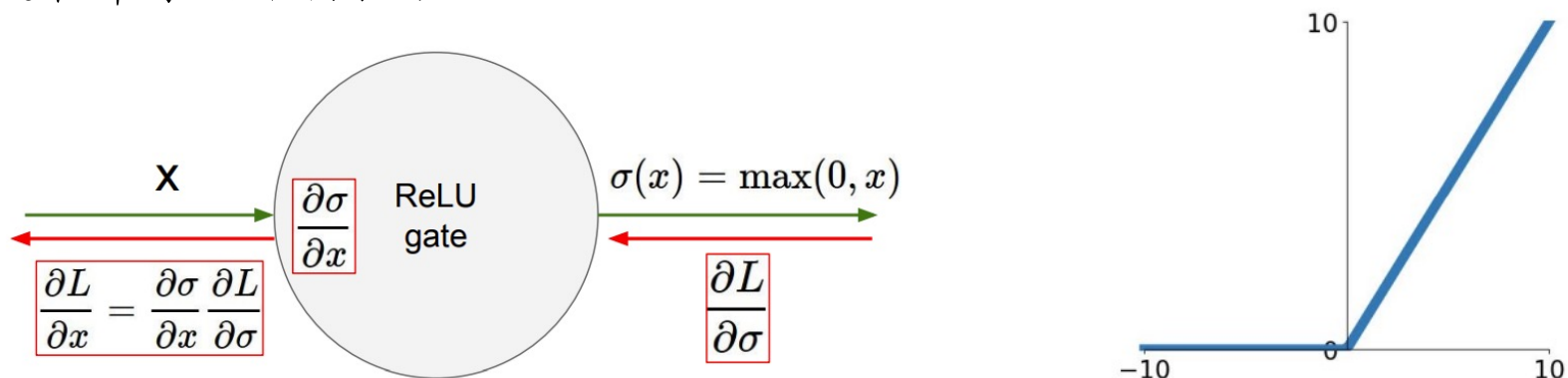
- ▶ 计算上更加高效
- ▶ 生物学合理性
 - ▶ 单侧抑制、宽兴奋边界
- ▶ 在一定程度上缓解梯度消失问题

▶ 性质：

- ▶ ReLU函数是非零中心化的
- ▶ 死亡ReLU问题 (Dying ReLU Problem)

死亡ReLU问题

如果参数在一次不恰当的更新后，隐藏层中的某一个ReLU神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是0，在以后的训练过程中永远不会被激活。



► 想象当某次异常输入 x 非常大，导致迭代更新后偏置 b 变得非常小，导致之后 ReLU 神经元都不会被激活。

► **解决办法：**

► Leaky ReLU 函数激活

常见激活函数(ReLU函数的改进)

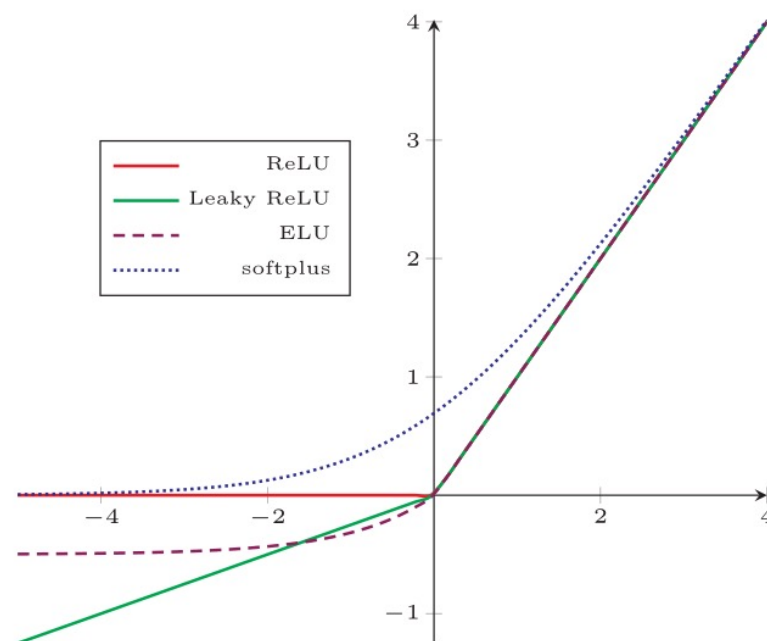
$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma \min(0, x)$$

第*i*个神经元

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

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

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



常见激活函数及其导数

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 函数	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU 函数	$f(x) = \max(0, x) + \min(0, \gamma(\exp(x) - 1))$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \log(1 + \exp(x))$	$f'(x) = \frac{1}{1+\exp(-x)}$

TLDR: In practice:

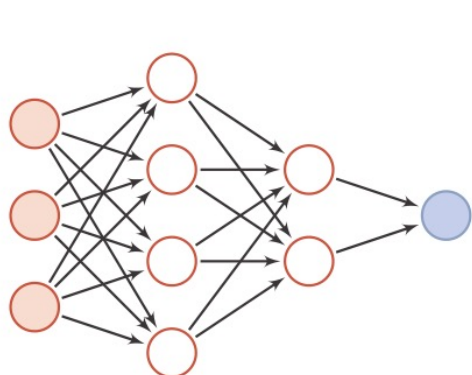
- Use **ReLU**. Be careful with your learning rates
- Try out **Leaky ReLU / Maxout / ELU**
- Try out **tanh** but don't expect much
- **Don't use sigmoid**

人工神经网络

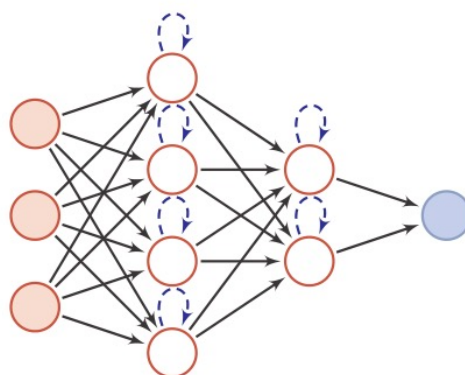
- ▶ 人工神经网络主要由大量的神经元以及它们之间的有向连接构成。因此考虑三方面：
- ▶ 神经元的激活规则
 - ▶ 主要是指神经元输入到输出之间的映射关系，一般为非线性函数。
- ▶ 网络的拓扑结构
 - ▶ 不同神经元之间的连接关系。
- ▶ 学习算法
 - ▶ 通过训练数据来学习神经网络的参数。

网络结构

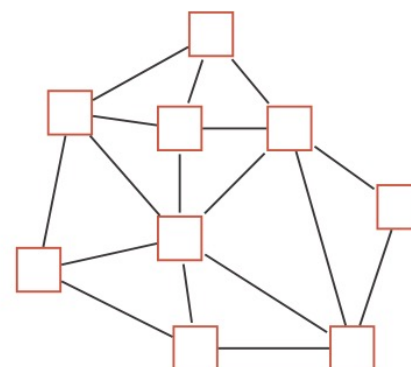
- ▶ 人工神经网络由神经元模型构成，这种由许多神经元组成的信息处理网络具有并行分布结构。



(a) 前馈网络



(b) 记忆网络



(c) 图网络

圆形节点表示一个神经元，方形节点表示一组神经元。



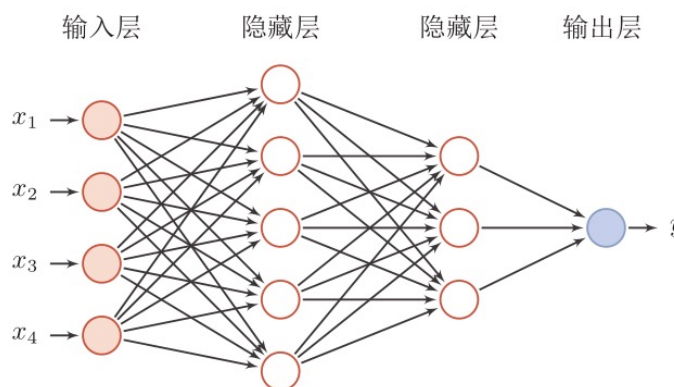


前馈神经网络

网络结构

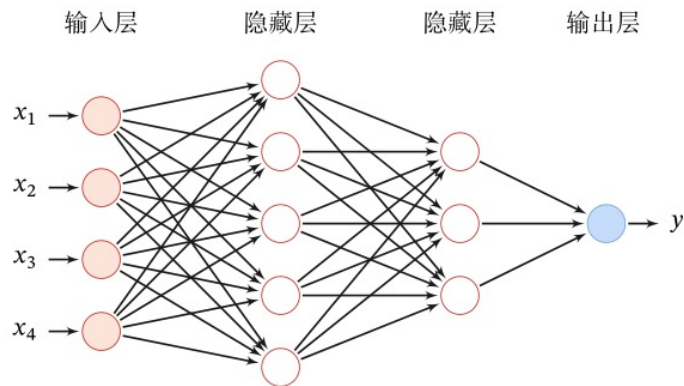
▶ 前馈神经网络

- ▶ 也叫全连接神经网络(feedforward neural network)、多层感知器(multilayer perceptron)
- ▶ 各神经元分别属于不同的层，层内无连接。
- ▶ 相邻两层之间的神经元全部两两连接。
- ▶ 整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。



前馈网络

给定一个前馈神经网络，用下面的记号来描述这样网络：



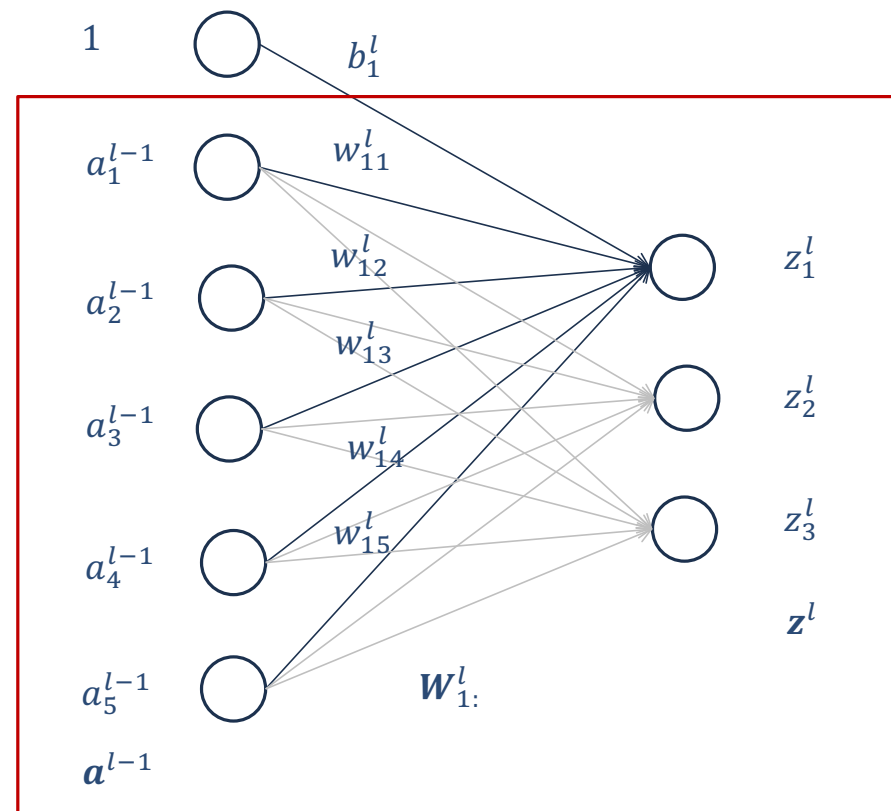
记号	含义
L	神经网络的层数 层数 L 一般只考虑隐藏层和输出层
M_l	第 l 层神经元的个数
$f_l(\cdot)$	第 l 层神经元的激活函数
$\mathbf{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	第 $l-1$ 层到第 l 层的权重矩阵
$\mathbf{b}^{(l)} \in \mathbb{R}^{M_l}$	第 $l-1$ 层到第 l 层的偏置
$\mathbf{z}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的净输入（净活性值）
$\mathbf{a}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的输出（活性值）

信息传递过程

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)},$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)}).$$

$$z_1^l = \mathbf{w}_{1:}^l \mathbf{a}^{l-1} + b_1^l$$



信息传递过程

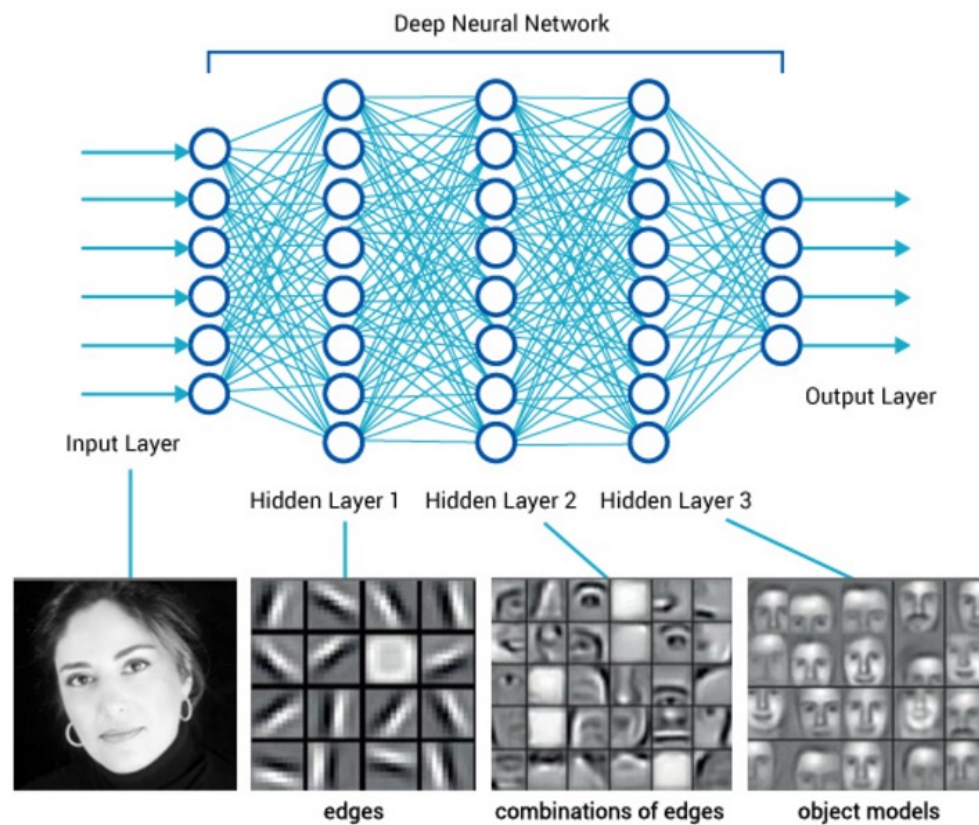
► 前馈神经网络通过下面公式进行信息传播。

$$\begin{aligned}\mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \\ \mathbf{a}^{(l)} &= f_l(\mathbf{z}^{(l)}).\end{aligned}$$

► 前馈计算：

$$\mathbf{x} = \mathbf{a}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{a}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \dots \rightarrow \mathbf{a}^{(L-1)} \rightarrow \mathbf{z}^{(L)} \rightarrow \mathbf{a}^{(L)} = \phi(\mathbf{x}; \mathbf{W}, \mathbf{b})$$

深层前馈神经网络



应用到机器学习

- ▶ 神经网络可以作为一个“万能”函数来使用，可以用来进行复杂的特征转换，或逼近一个复杂的条件分布。

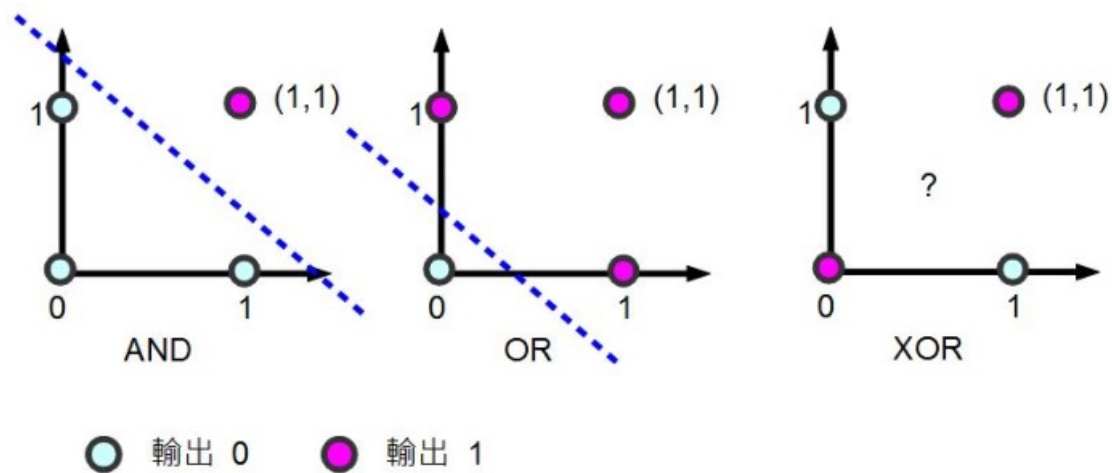
$$\hat{y} = g(\varphi(\mathbf{x}), \theta)$$

分类器 神经网络

- ▶ 如果 $g(\cdot)$ 为Logistic回归，那么Logistic回归分类器可以看成神经网络的最后一层。

XOR问题

<http://playground.tensorflow.org>



►前馈神经网络能否解决XOR问题?



参数学习

应用到机器学习

► 对于多分类问题

- 如果使用Softmax回归分类器，相当于网络最后一层设置C个神经元，其输出经过Softmax函数进行归一化后可以作为每个类的条件概率。

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}^{(L)})$$

- 采用交叉熵损失函数，对于样本(x,y)，其损失函数为

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^T \log \hat{\mathbf{y}} = -\log p_{\theta}(y^*|x)$$

参数学习

- ▶ 给定训练集为 $D = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$ ，将每个样本 $\mathbf{x}^{(n)}$ 输入给前馈神经网络，得到网络输出为 $\hat{y}^{(n)}$ ，其在数据集 D 上的结构化风险函数为：

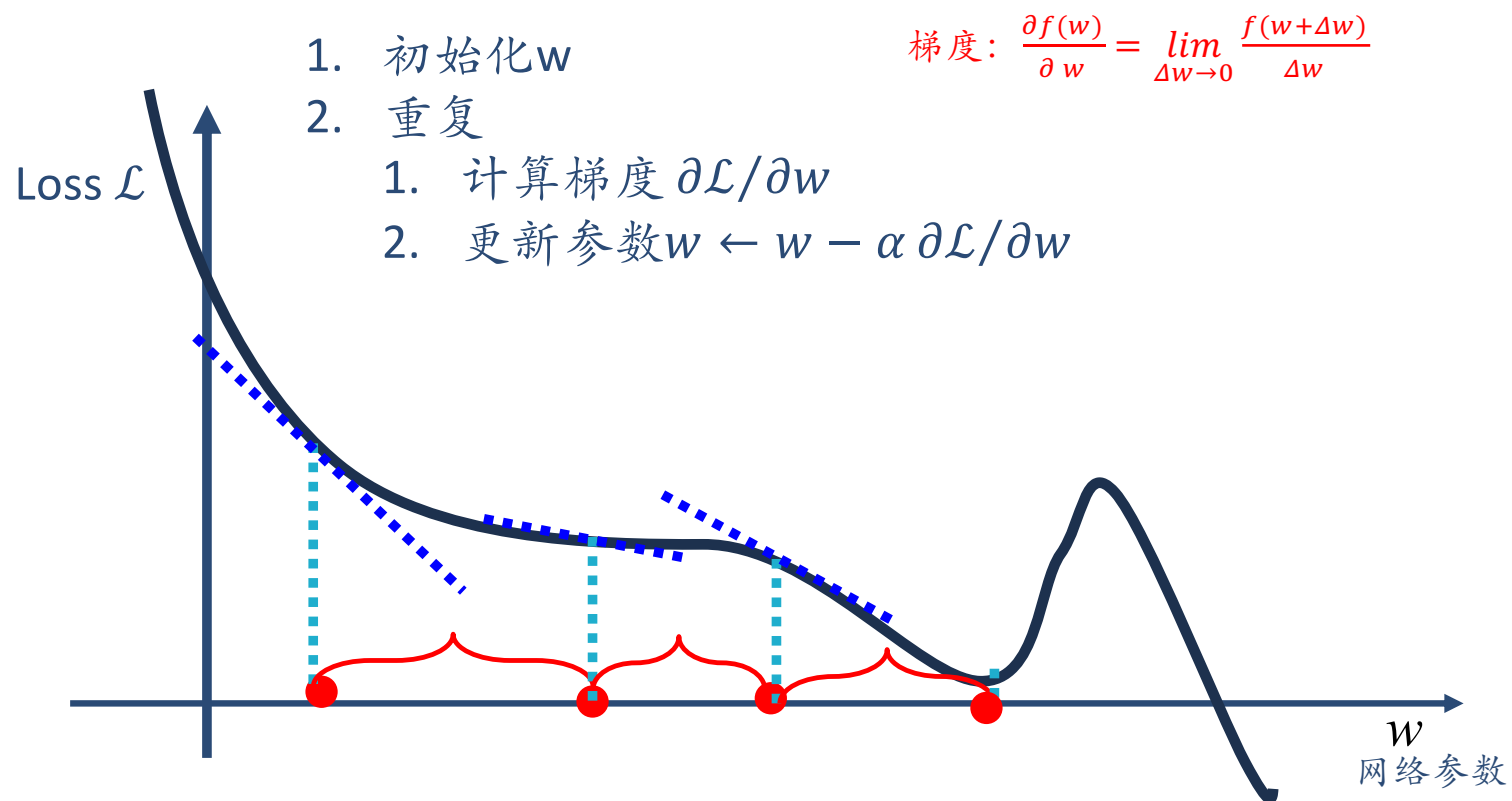
$$\mathcal{R}(W, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2} \lambda \|W\|_F^2$$

- ▶ 梯度下降

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$

梯度下降



如何计算梯度？

▶神经网络为一个复杂的复合函数

▶链式法则

$$y = f^5(f^4(f^3(f^2(f^1(x))))) \rightarrow \frac{\partial y}{\partial x} = \frac{\partial f^1}{\partial x} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial f^4}{\partial f^3} \frac{\partial f^5}{\partial f^4}$$

▶反向传播算法

▶根据前馈网络的特点而设计的高效方法

▶一个更加通用的计算方法

▶自动微分（Automatic Differentiation, AD）

反向传播算法

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}},$$

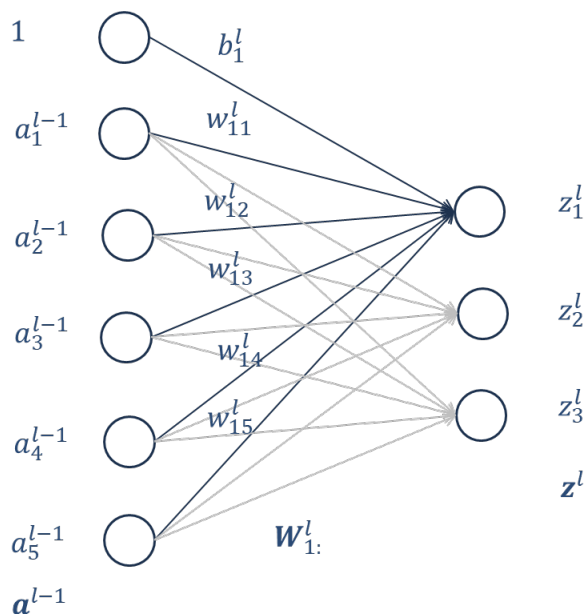
$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}.$$

► 只需要计算三个偏导数

$$\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}}, \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \text{ 和 } \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

反向传播算法

$$\mathbf{z}^{(l)} = W^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$



$$\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \longrightarrow$$

$$\begin{aligned} \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \dots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \dots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}} \right] \\ &= \left[0, \dots, \frac{\partial (\mathbf{w}_{i:}^{(l)} \mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \dots, 0 \right] \\ &= \left[0, \dots, a_j^{(l-1)}, \dots, 0 \right] \\ &\triangleq \mathbb{I}_i(a_j^{(l-1)}) \in \mathbb{R}^{m^{(l)}}, \end{aligned}$$

误差项

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{m^{(l)}}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \longrightarrow$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

计算 $\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{m^{(l)}}$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$$

$$\mathbf{z}^{(l+1)} = W^{(l+1)} \mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}$$

$$\begin{aligned} \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} &= \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} \\ &= \text{diag}(f'_l(\mathbf{z}^{(l)})) \end{aligned}$$

$$\begin{aligned} \delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\ &= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\ &= \text{diag}(f'_l(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^T \cdot \delta^{(l+1)} \\ &= f'_l(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^T \delta^{(l+1)}), \end{aligned}$$

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} = (W^{(l+1)})^T$$

- 第 l 层的一个神经元的误差项是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和。然后再乘上该神经元激活函数的梯度。

反向传播算法

在计算出上面三个偏导数之后, 公式 (4.49) 可以写为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \mathbb{I}_i(a_j^{(l-1)})\delta_i^{(l)} = \delta_i^{(l)} a_j^{(l-1)}.$$

进一步, $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^\top.$$

同理, $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层偏置 $\mathbf{b}^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

反向传播算法

算法 4.1 使用反向传播算法的随机梯度下降训练过程

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$, 验证集 \mathcal{V} , 学习率 α , 正则化系数 λ , 网络层数 L , 神经元数量 $M_l, 1 \leq l \leq L$.

```
1 随机初始化  $\mathbf{W}, \mathbf{b}$ ;  
2 repeat  
3   对训练集  $\mathcal{D}$  中的样本随机重排序;  
4   for  $n = 1 \cdots N$  do  
5     从训练集  $\mathcal{D}$  中选取样本  $(\mathbf{x}^{(n)}, y^{(n)})$ ;  
6     前馈计算每一层的净输入  $\mathbf{z}^{(l)}$  和激活值  $\mathbf{a}^{(l)}$ , 直到最后一层;  
7     反向传播计算每一层的误差  $\delta^{(l)}$ ; // 公式 (4.63)  
      // 计算每一层参数的导数  
8      $\forall l, \quad \frac{\partial \mathcal{L}(y^{(n)}, \mathbf{y}^{(n)})}{\partial \mathbf{W}^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^\top$ ; // 公式 (4.68)  
9      $\forall l, \quad \frac{\partial \mathcal{L}(y^{(n)}, \mathbf{y}^{(n)})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$ ; // 公式 (4.69)  
      // 更新参数  
10     $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \alpha (\delta^{(l)} (\mathbf{a}^{(l-1)})^\top + \lambda \mathbf{W}^{(l)})$ ;  
11     $\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \delta^{(l)}$ ;  
12  end  
13 until 神经网络模型在验证集  $\mathcal{V}$  上的错误率不再下降;  
输出:  $\mathbf{W}, \mathbf{b}$ 
```



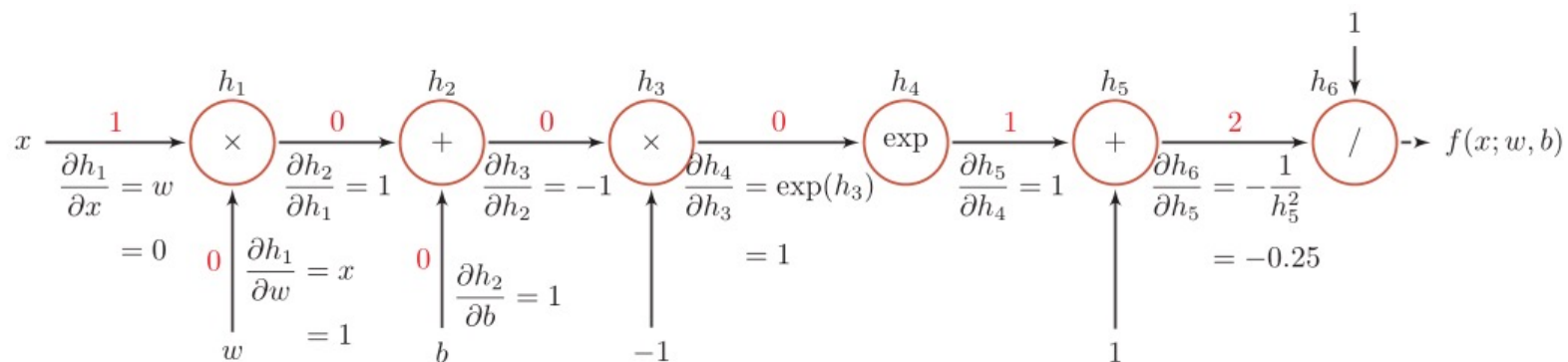
计算图与自动微分

计算图与自动微分

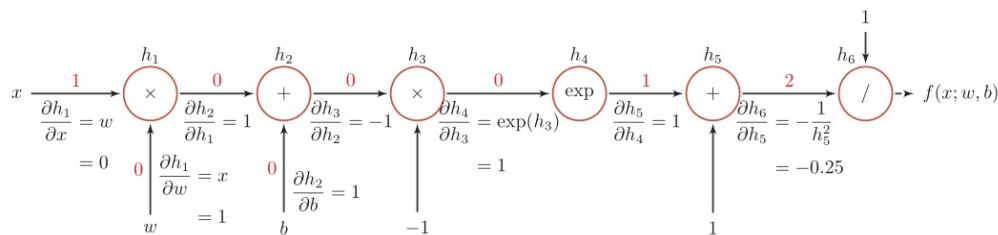
► 自动微分是利用链式法则来自动计算一个复合函数的梯度。

$$f(x; w, b) = \frac{1}{\exp(-(wx + b)) + 1}.$$

► 计算图



计算图



函数	导数	
$h_1 = x \times w$	$\frac{\partial h_1}{\partial w} = x$	$\frac{\partial h_1}{\partial x} = w$
$h_2 = h_1 + b$	$\frac{\partial h_2}{\partial h_1} = 1$	$\frac{\partial h_2}{\partial b} = 1$
$h_3 = h_2 \times -1$	$\frac{\partial h_3}{\partial h_2} = -1$	
$h_4 = \exp(h_3)$	$\frac{\partial h_4}{\partial h_3} = \exp(h_3)$	
$h_5 = h_4 + 1$	$\frac{\partial h_5}{\partial h_4} = 1$	
$h_6 = 1/h_5$	$\frac{\partial h_6}{\partial h_5} = -\frac{1}{h_5^2}$	

当 $x = 1, w = 0, b = 0$ 时，可以得到

$$\begin{aligned}
 \frac{\partial f(x; w, b)}{\partial w} \Big|_{x=1, w=0, b=0} &= \frac{\partial f(x; w, b)}{\partial h_6} \frac{\partial h_6}{\partial h_5} \frac{\partial h_5}{\partial h_4} \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial w} \\
 &= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 \times 1 \\
 &= 0.25.
 \end{aligned}$$

自动微分

- ▶ 如果函数和参数之间有多条路径，可以将这多条路径上的导数再进行相加，得到最终的梯度。

反向传播算法 (自动微分的反向模式)

- ▶ 前馈神经网络的训练过程可以分为以下三步
 - ▶ 前向计算每一层的状态和激活值，直到最后一层
 - ▶ 反向计算每一层的参数的偏导数
 - ▶ 更新参数

静态计算图和动态计算图

- ▶ 静态计算图是在编译时构建计算图，计算图构建好之后在程序运行时不能改变。
 - ▶ Theano和Tensorflow
- ▶ 动态计算图是在程序运行时动态构建。两种构建方式各有优缺点。
 - ▶ DyNet, Chainer和PyTorch
- ▶ 静态计算图在构建时可以进行优化，并行能力强，但灵活性比较低。动态计算图则不容易优化，当不同输入的网络结构不一致时，难以并行计算，但是灵活性比较高。

如何实现？



Getting started: 30 seconds to Keras

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(output_dim=64, input_dim=100))
model.add(Activation("relu"))
model.add(Dense(output_dim=10))
model.add(Activation("softmax"))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd', metrics=['accuracy'])

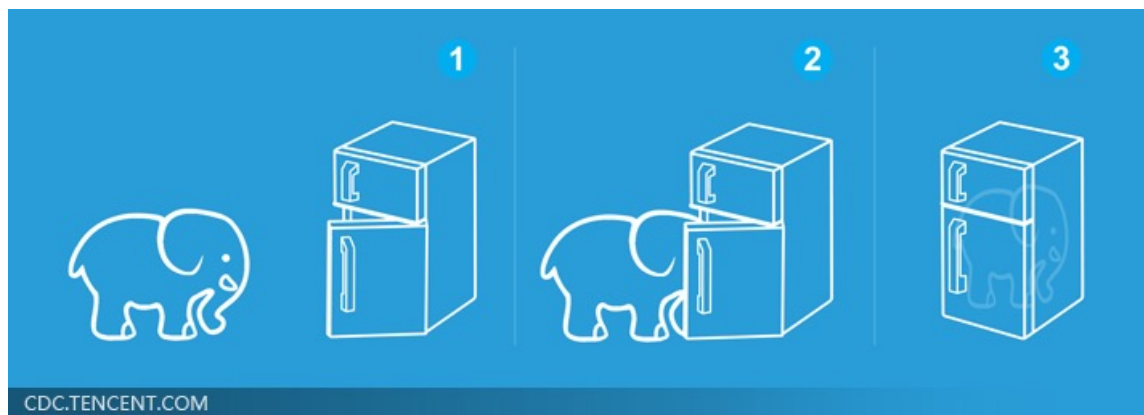
model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)

loss = model.evaluate(X_test, Y_test, batch_size=32)
```

深度学习的三个步骤



Deep Learning is so simple





优化问题

优化问题

▶ 难点

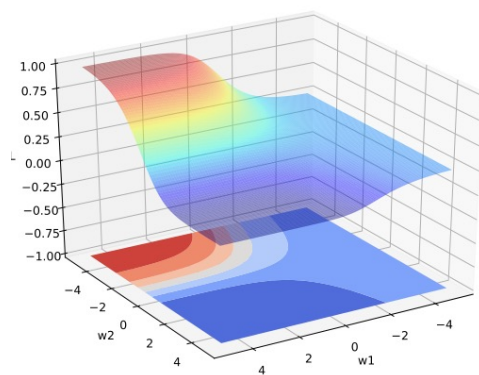
- ▶ 参数过多，影响训练
- ▶ 非凸优化问题：即存在局部最优而非全局最优解，影响迭代
- ▶ 梯度消失问题，下层参数比较难调
- ▶ 参数解释起来比较困难

▶ 需求

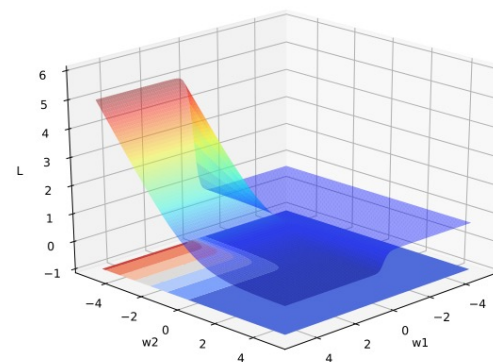
- ▶ 计算资源要大
- ▶ 数据要多
- ▶ 算法效率要好：即收敛快

优化问题

► 非凸优化问题



(a) 平方误差损失



(b) 交叉熵损失

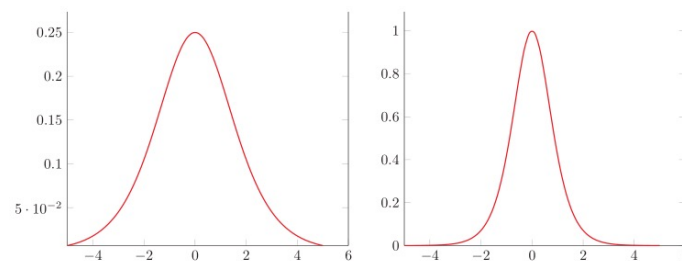
图 4.9 神经网络 $y = \sigma(w_2\sigma(w_1x))$ 的损失函数

优化问题

► 梯度消失问题 (Vanishing Gradient Problem)

$$y = f^5(f^4(f^3(f^2(f^1(x)))))$$

$$\frac{\partial y}{\partial x} = \frac{\partial f^1}{\partial x} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial f^4}{\partial f^3} \frac{\partial f^5}{\partial f^4}$$



(a) logistic 函数的导数

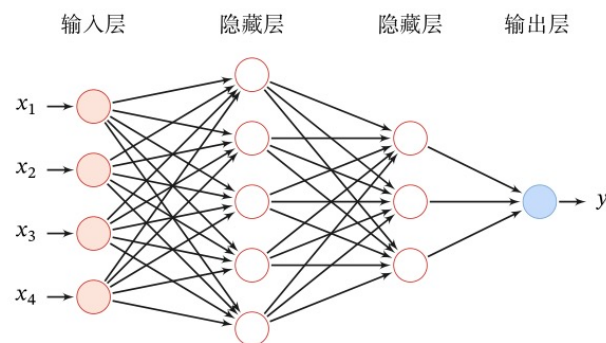
(b) tanh 函数的导数

教学内容

► 激活神经元

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 函数	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU 函数	$f(x) = \max(0, x) + \min(0, \gamma(\exp(x) - 1))$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \log(1 + \exp(x))$	$f'(x) = \frac{1}{1+\exp(-x)}$

► 前馈神经网络



► 参数学习：反向传播算法

► 计算图与自动微分