《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

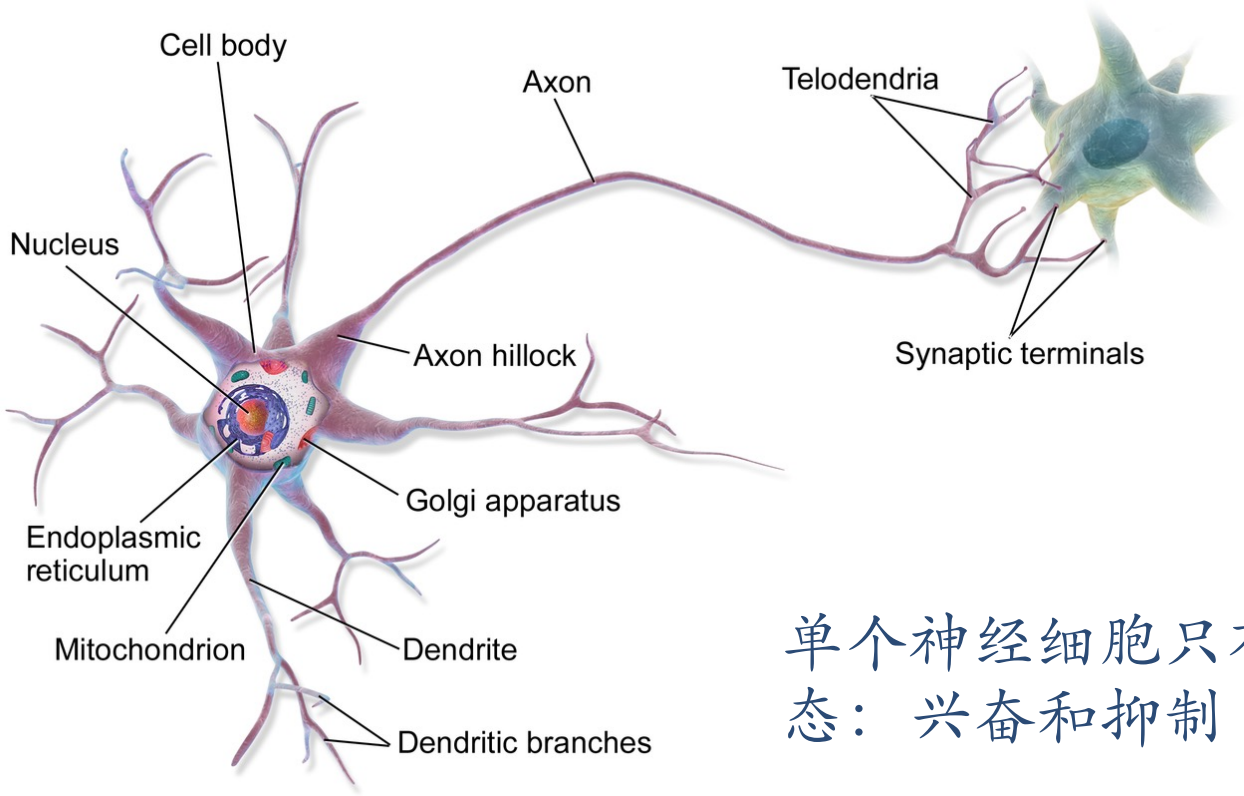
神经元

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。

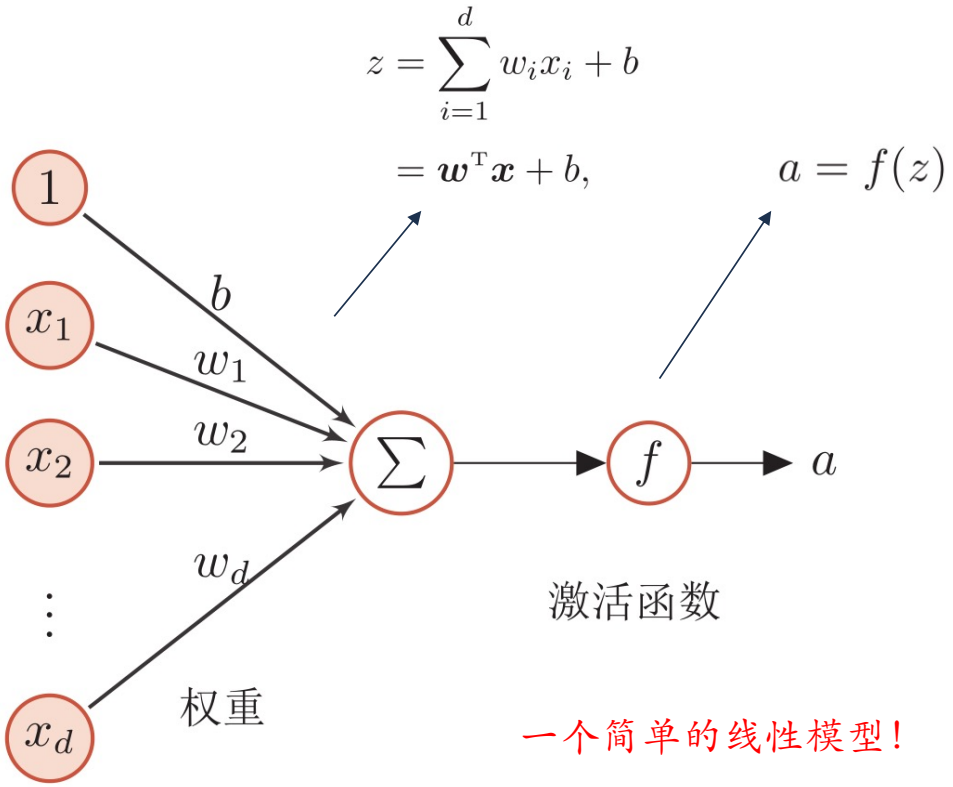
生物神经元

video: structure of brain



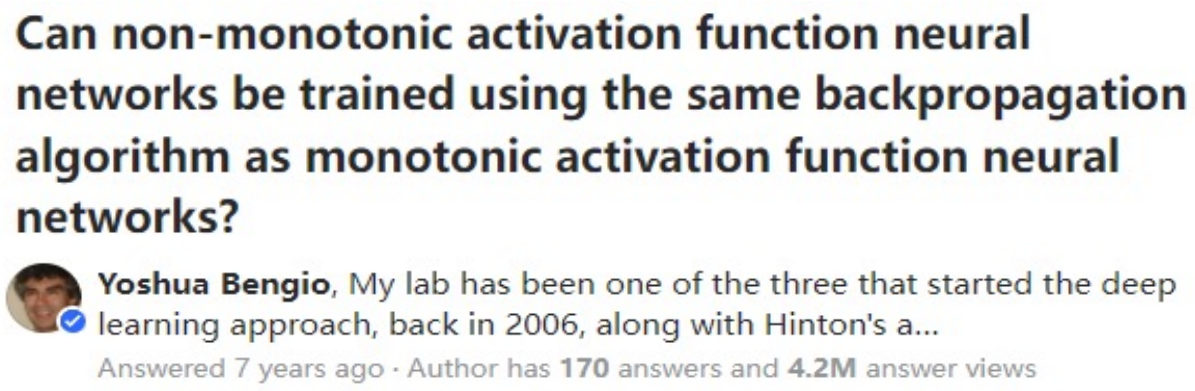
单个神经细胞只有两种状态：兴奋和抑制

人工神经元


$z = \sum_{i=1}^{d} w_i x_i + b$
$= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b,$
$a = f(z)$
1
x_1
x_2
x_d
b
w_1
w_2
w_d
$\sum$
f
a
激活函数
权重
一个简单的线性模型！

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和稳定性。
- 单调递增
  - ???



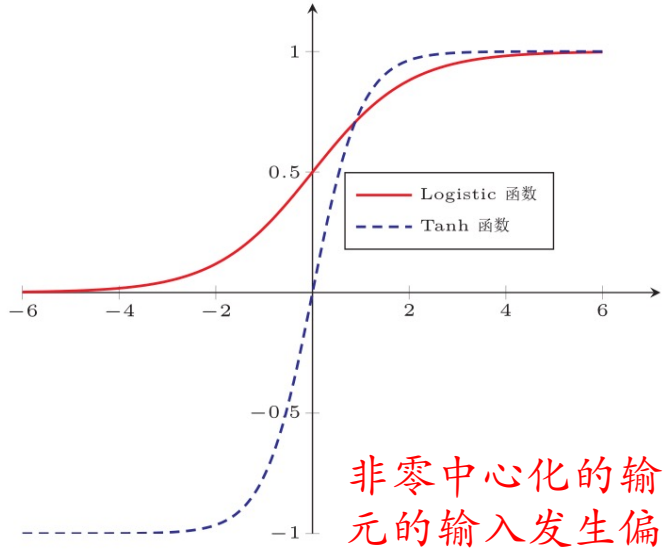
https://www.quora.com/Can-non-monotonic-activation-function-neural-networks-be-trained-using-the-same-backpropagation-algorithm-as-monotonic-activation-function-neural-networks

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后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？

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

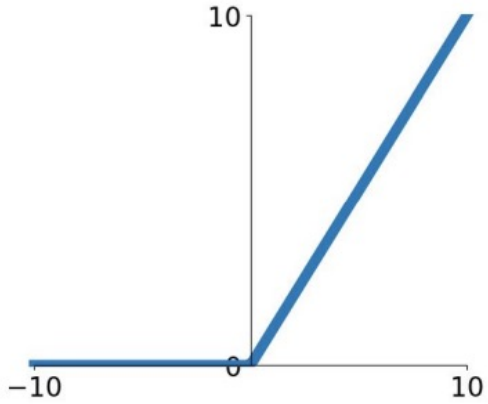
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \sigma}\sigma(1-\sigma)x$$

- 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

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

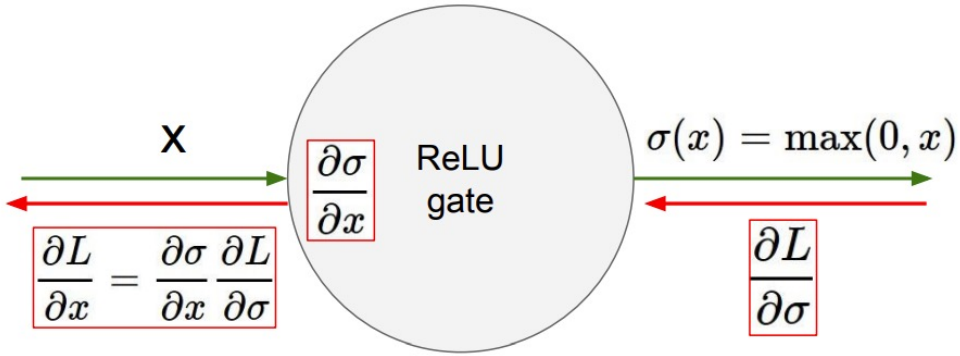


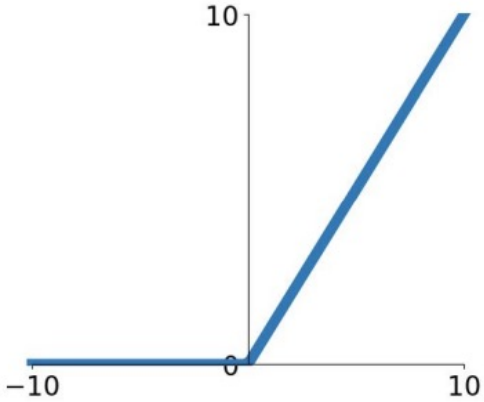
- 计算上更加高效
- 生物学合理性
  - 单侧抑制、宽兴奋边界
- 在一定程度上缓解梯度消失问题

- 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）

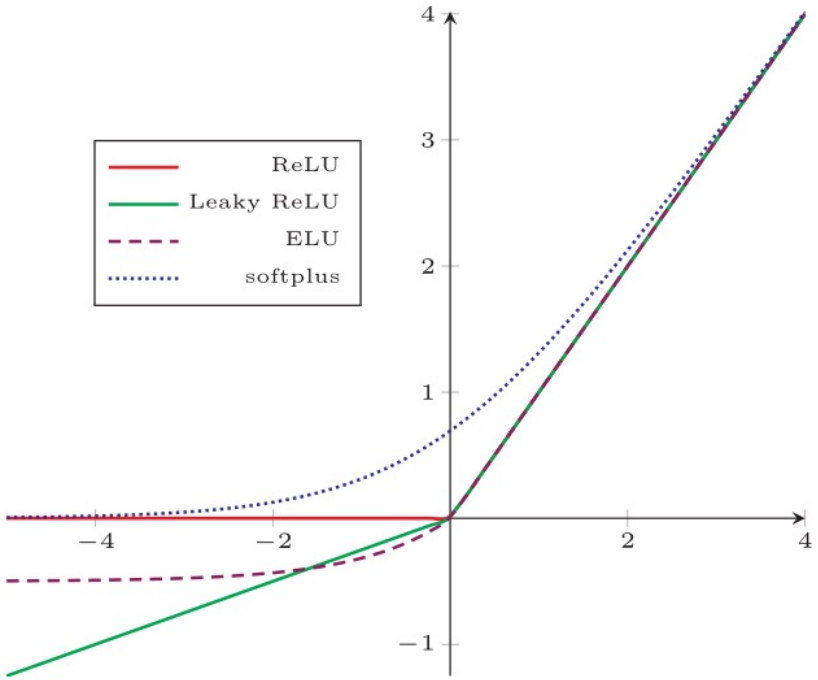
$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \gamma \min(0, x)$$

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

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

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

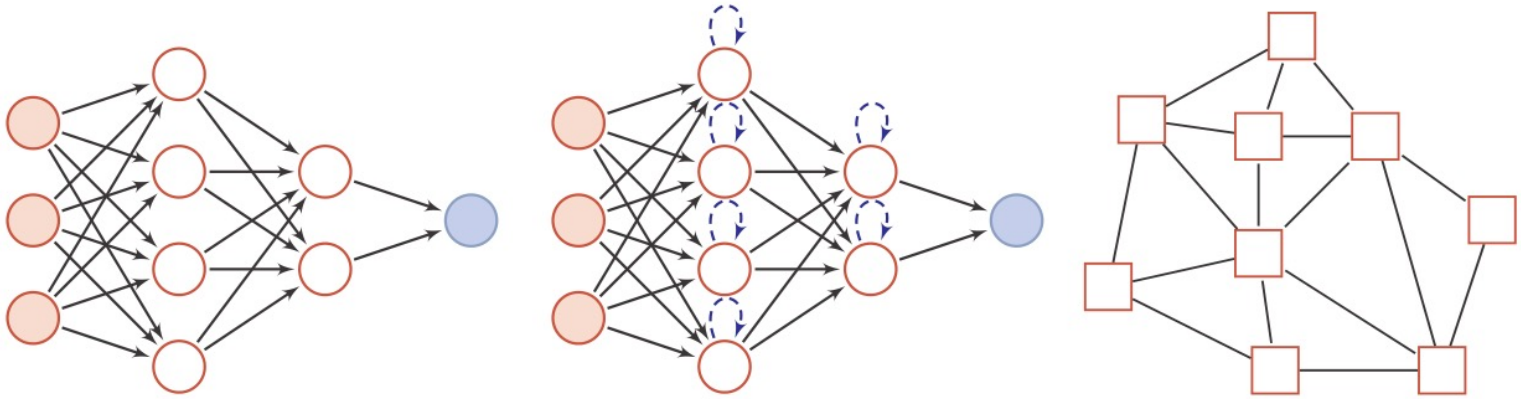
人工神经网络

- 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

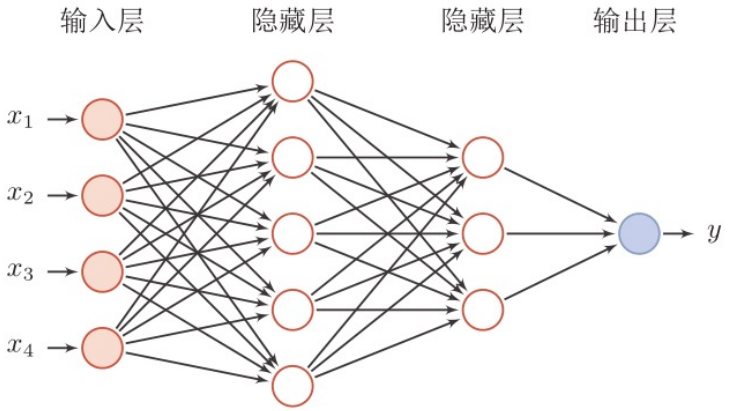
圆形节点表示一个神经元，方形节点表示一组神经元。


深度学习

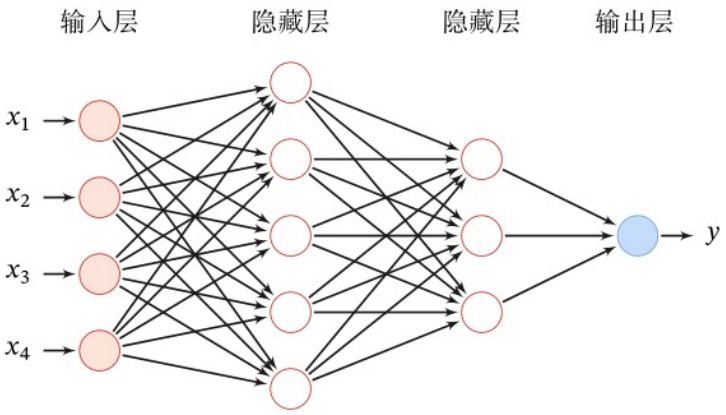
前馈神经网络

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
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
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	第 $l-1$ 层到第 l 层的权重矩阵
$\boldsymbol{b}^{(l)} \in \mathbb{R}^{M_l}$	第 $l-1$ 层到第 l 层的偏置
$\boldsymbol{z}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的净输入（净活性值）
$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的输出（活性值）

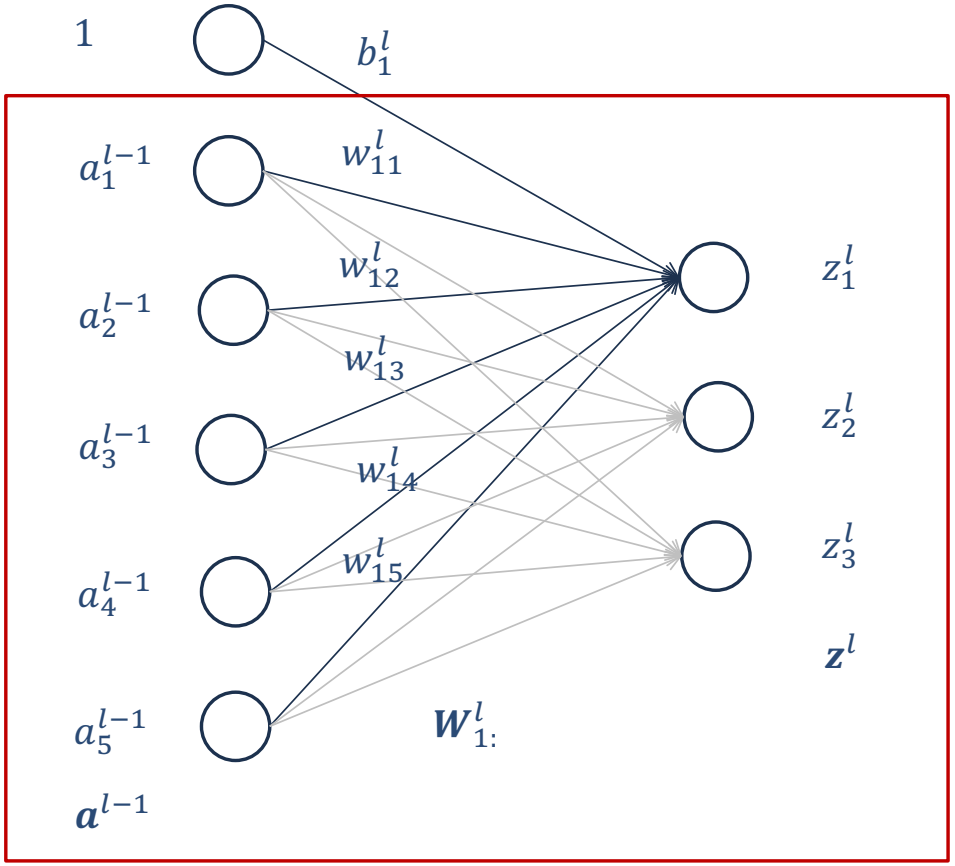
层数L一般只考虑隐藏层和输出层

信息传递过程

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

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

$$z_1^l = \boldsymbol{W}_{1:}^l \boldsymbol{a}^{l-1} + b_1^l$$



信息传递过程

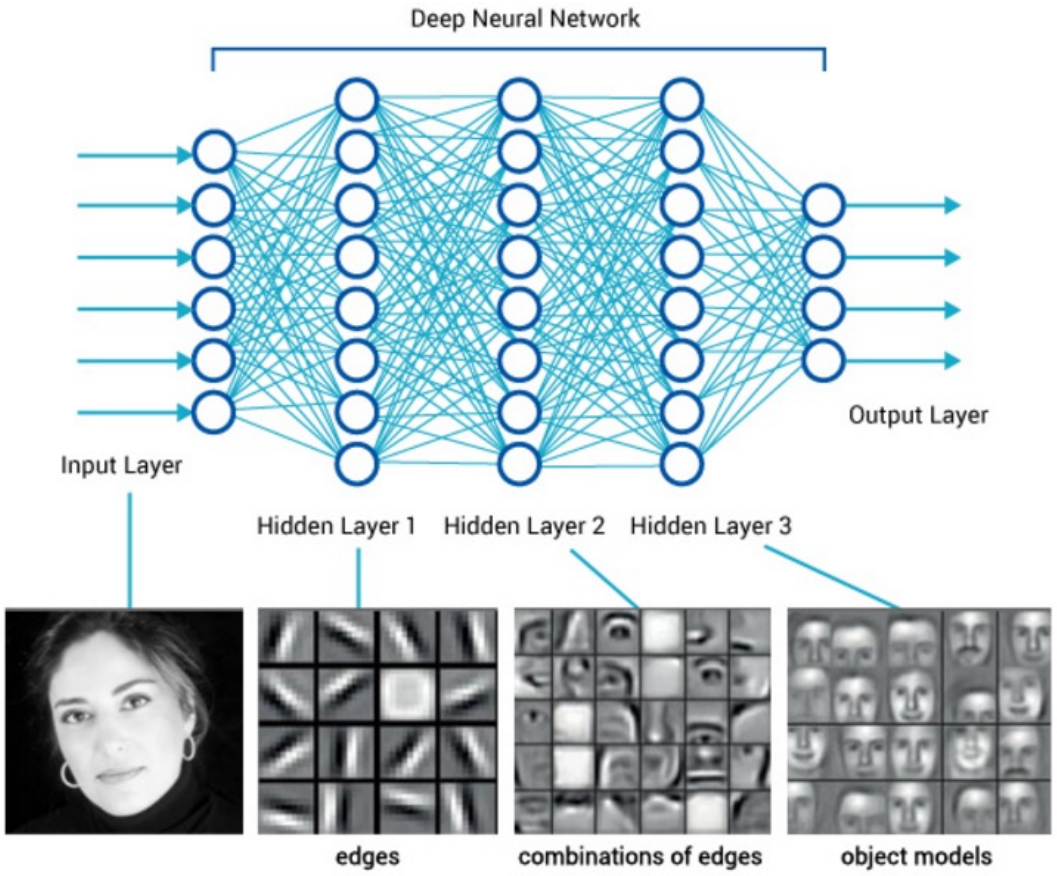
- 前馈神经网络通过下面公式进行信息传播。

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

- 前馈计算：

$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \to \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)} = \phi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}))$$

深层前馈神经网络


Deep Neural Network
Input Layer
Hidden Layer 1
Hidden Layer 2
Hidden Layer 3
Output Layer
edges
combinations of edges
object models

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。

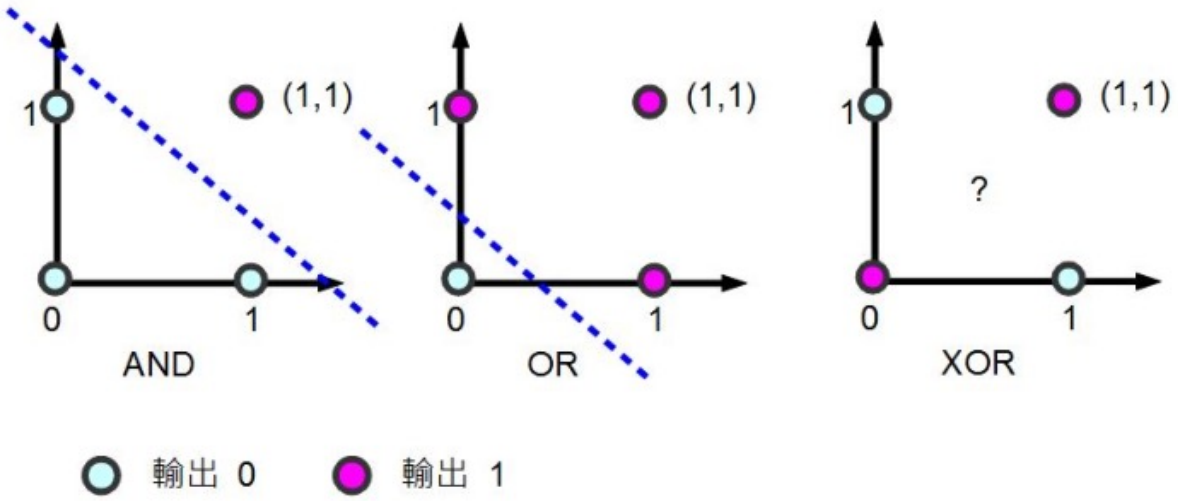
$$\hat{y} = g(\varphi(\mathbf{x}), \theta)$$

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

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

参数学习

- 给定训练集为$D = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，将每个样本$\boldsymbol{x}^{(n)}$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

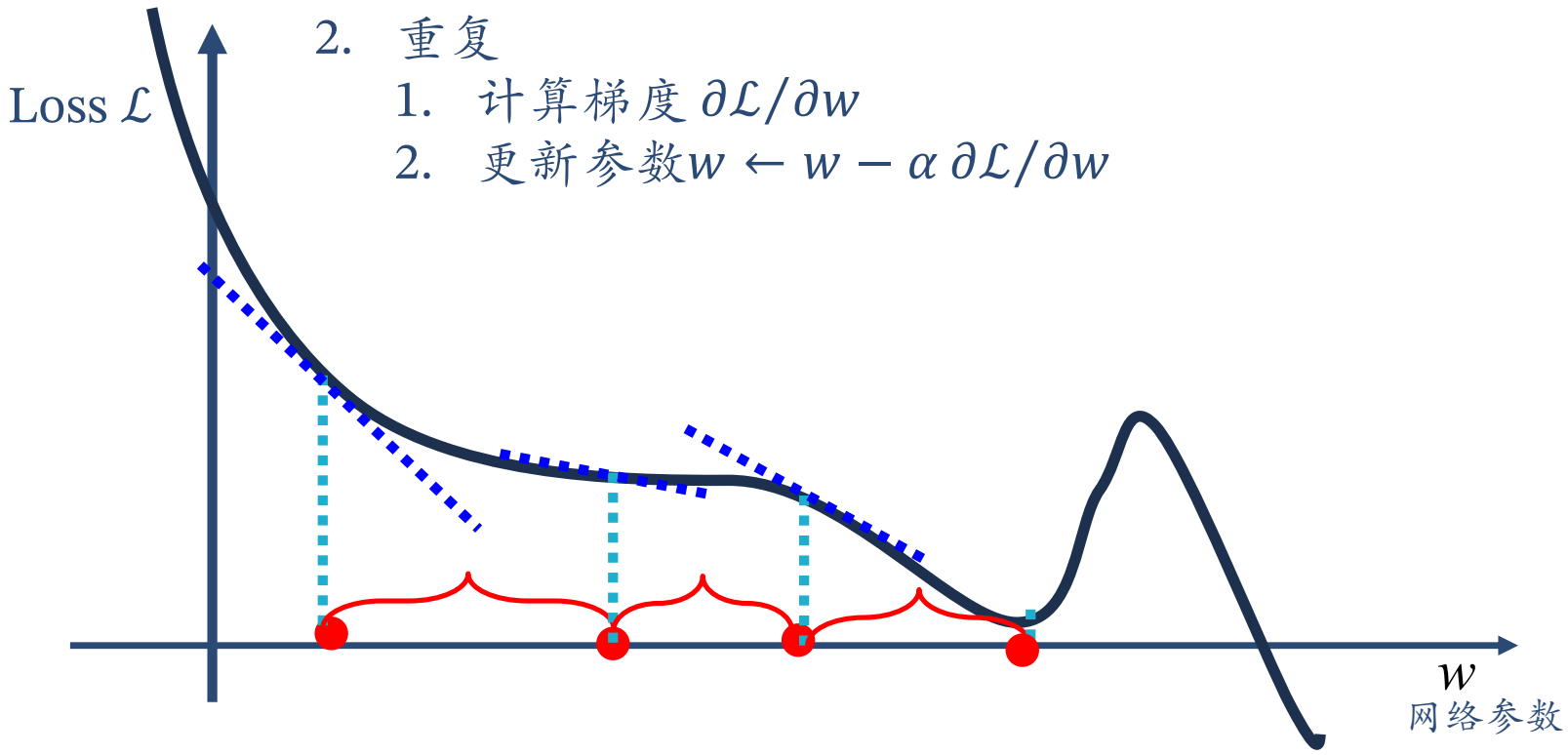
$$W^{(l)} \leftarrow W^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

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

梯度下降

梯度：$\frac{\partial f(w)}{\partial w} = \lim_{\Delta w \to 0} \frac{f(w+\Delta w)}{\Delta w}$

1. 初始化w
2. 重复
   1. 计算梯度 $\partial \mathcal{L}/\partial w$
   2. 更新参数 $w \leftarrow w - \alpha \, \partial \mathcal{L}/\partial w$



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

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

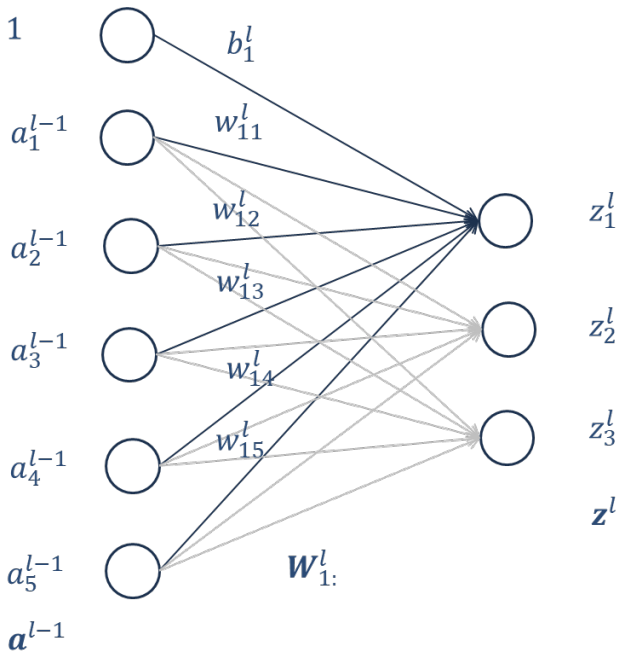
$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(l)}}.$$

- 只需要计算三个偏导数

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

反向传播算法

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



$$\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} = \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right]$$

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

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

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

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

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

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

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

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = \mathrm{diag}(f_l'(\mathbf{z}^{(l)}))$$

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

- 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

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

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

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

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 $\mathcal{D}=\{(\boldsymbol{x}^{(n)},y^{(n)})\}_{n=1}^{N}$, 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 $M_l, 1\leq l\leq L$.

随机初始化 $\boldsymbol{W},\boldsymbol{b}$;

repeat

  对训练集 $\mathcal{D}$ 中的样本随机重排序;

  for $n=1\cdots N$ **do**

    从训练集 $\mathcal{D}$ 中选取样本 $(\boldsymbol{x}^{(n)},y^{(n)})$;

    前馈计算每一层的净输入 $\boldsymbol{z}^{(l)}$ 和激活值 $\boldsymbol{a}^{(l)}$, 直到最后一层;

    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

    $\forall l,\quad \frac{\partial \mathcal{L}(y^{(n)},\hat{y}^{(n)})}{\partial \boldsymbol{W}^{(l)}}=\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}}$; // 公式 (4.68)

    $\forall l,\quad \frac{\partial \mathcal{L}(y^{(n)},\hat{y}^{(n)})}{\partial \boldsymbol{b}^{(l)}}=\delta^{(l)}$; // 公式 (4.69)

    // 更新参数

    $\boldsymbol{W}^{(l)}\leftarrow \boldsymbol{W}^{(l)}-\alpha(\delta^{(l)}(\boldsymbol{a}^{(l-1)})^{\mathrm{T}}+\lambda \boldsymbol{W}^{(l)})$;

    $\boldsymbol{b}^{(l)}\leftarrow \boldsymbol{b}^{(l)}-\alpha\delta^{(l)}$;

  end

until 神经网络模型在验证集 $\mathcal{V}$ 上的错误率不再下降;

输出: $\boldsymbol{W},\boldsymbol{b}$

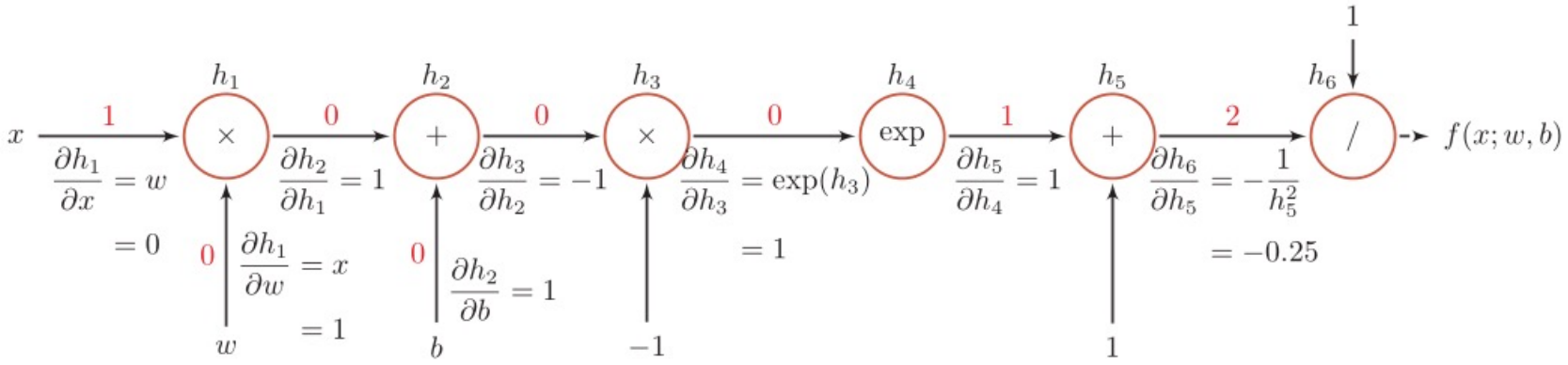
计算图与自动微分

计算图与自动微分

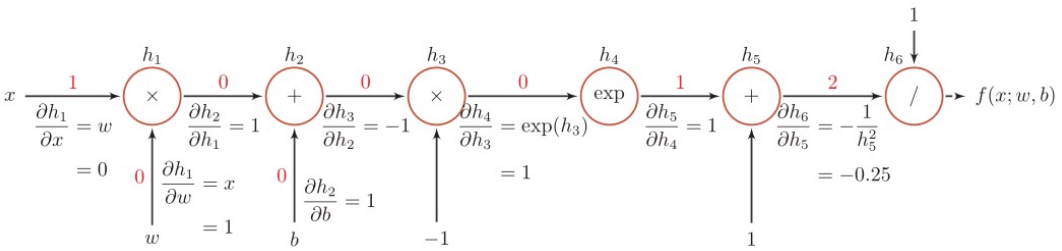
- 自动微分是利用链式法则来自动计算一个复合函数的梯度。

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

- 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

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

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，但灵活性比较差低。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？

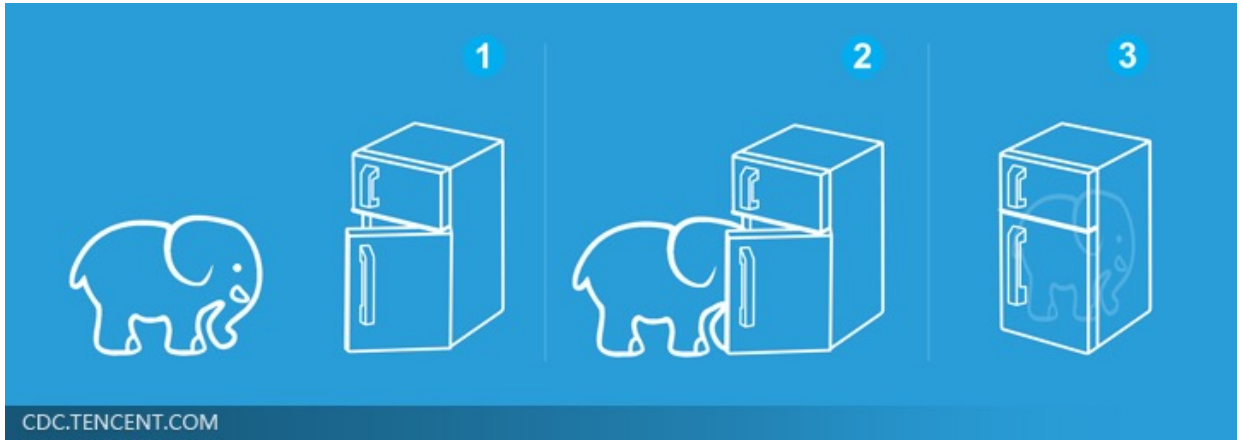

Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
from theano import *
What I actually do

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 ……



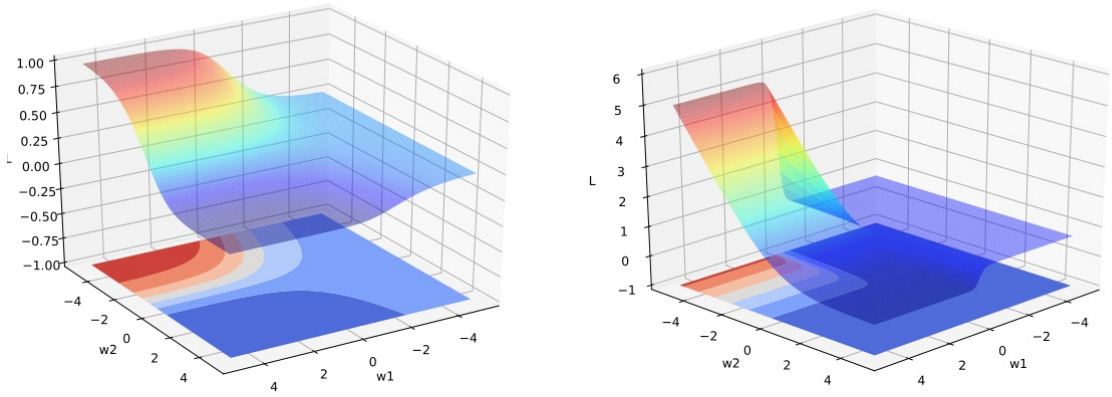
优化问题

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

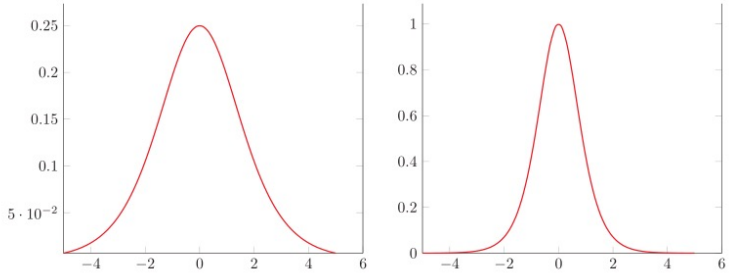
图 4.9 神经网络 $y = \sigma(w_2\sigma(w_1x))$ 的损失函数

优化问题

- 梯度消失问题（Vanishing Gradient Problem）

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

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



(a) logistic 函数的导数　　(b) tanh 函数的导数