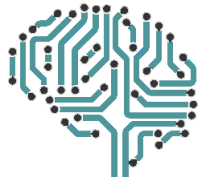
暨南大學
JINAN UNIVERSITY



机 器 学 习

第十章：人工神经网络

黄斐然

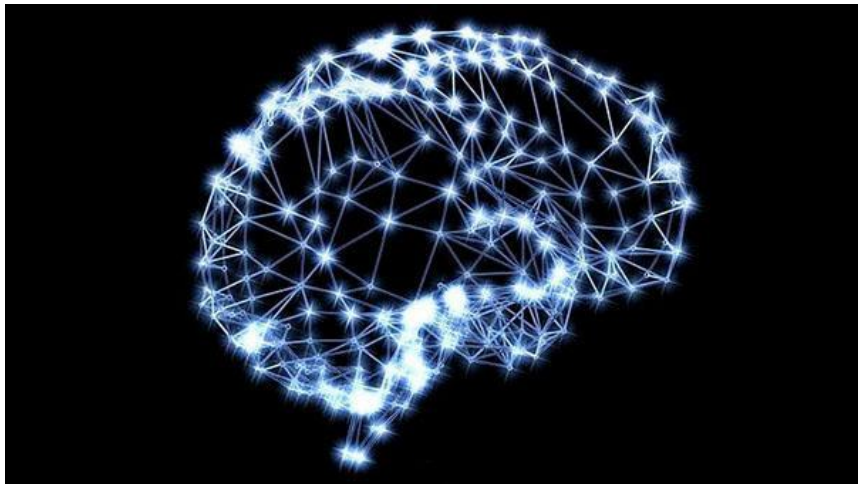
2022/5/9

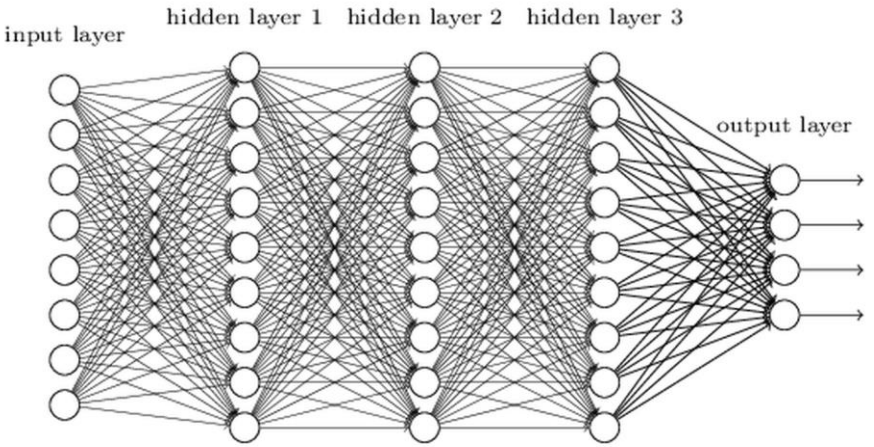
目录

目录

1 神经网络绪论

- 什么是神经网络：
  - 黑箱（不需要知道里面是什么）
  - 大脑（可以自动的学习一个任务目标）
  - 效果好（相比传统机器学习效果好）
  - 乐高（像积木一样，可以任意搭配组合）
  - 运算量大（相比传统算法需要更多计算资源）
  - ...



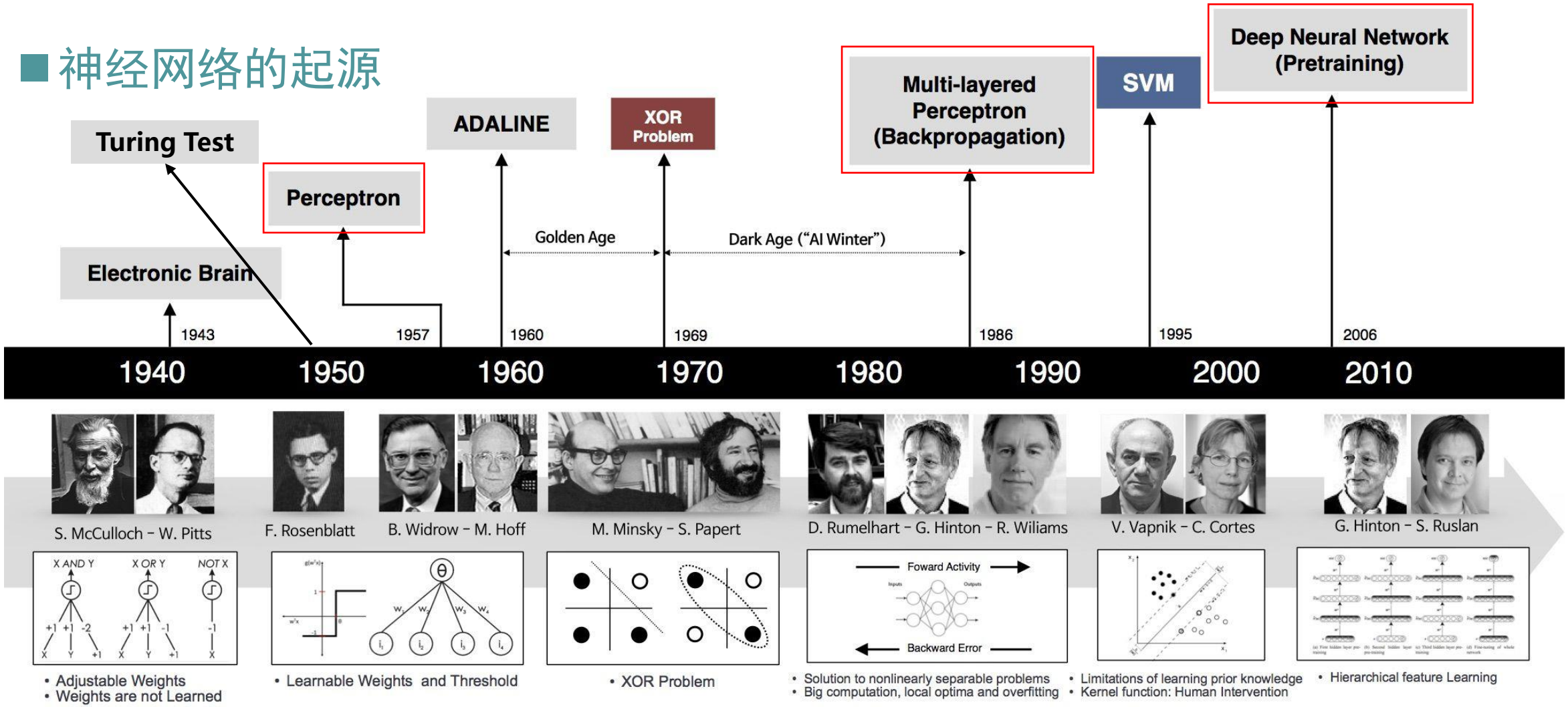


1 神经网络绪论

- 起源：机器学习
- 发展：图片分类、自然语言处理
- 应用：
  - 游戏AI（Alpha Go，绝悟AI）
  - 人脸识别（身份证识别）
  - 图像生成（DeepFake）
  - 自动驾驶（特斯拉，百度）
  - 数据可视化、数据降维（tSNE）
  - 搜索推荐系统（阿里，腾讯，YouTube）

1 神经网络绪论

■ 神经网络的起源



1 神经网络绪论

■神经网络是什么(Neural Networks)

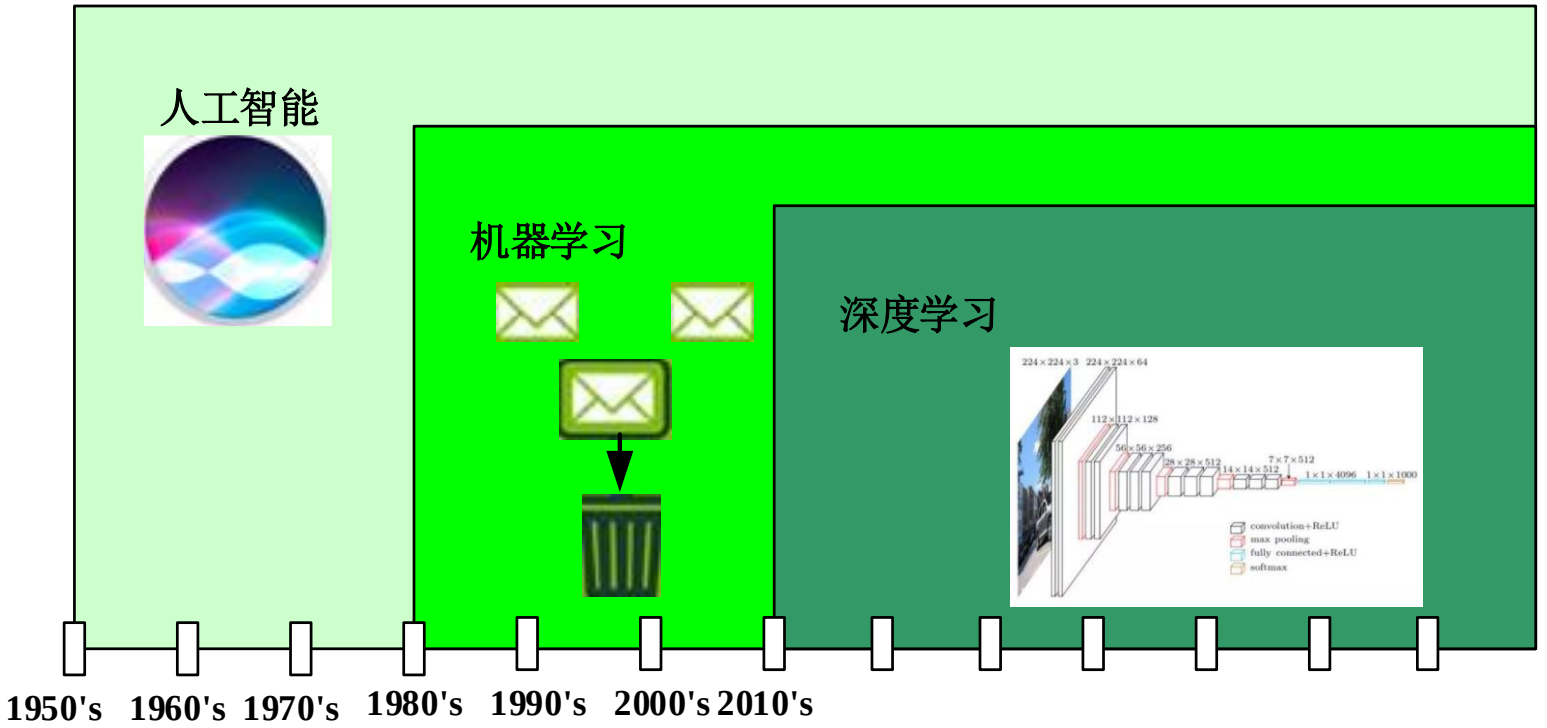
- **神经元**(Neurons)：又名神经原或神经细胞（英语：nerve cell），是神经系统的结构与功能单位之一。神经元能感知环境的变化，经过神经元对信息进行处理（激活），再将激活后的信息传递(输出）给其他的神经元。

- **神经网络**(Neural Networks)：是一种模仿生物神经网络（动物的中枢神经系统，特别是大脑）的结构和功能的数学模型或计算模型，用于对函数进行估计或近似。神经网络由大量的人工神经元联结进行计算。

- **与传统的机器学习方法的区别**：神经网络没有固定形式，可以DIY。这也决定了神经网络可以解决各种特定问题，只要我们能够针对这个特定的问题，想出一种合理的网络结构与优化目标，就可以解决该问题。这也是目前神经网络创新的主要方向。

1 神经网络绪论

■人工智能、机器学习、深度学习的关系



人工智能、机器学习、深度学习关系图

1 神经网络绪论

■ 神经网络的分类

■ 按照网络结构划分

- 多层感知机（最基础核心的神经网络）
- 卷积神经网络（多用于处理图像数据）
- 循环神经网络（多用于处理语言数据）

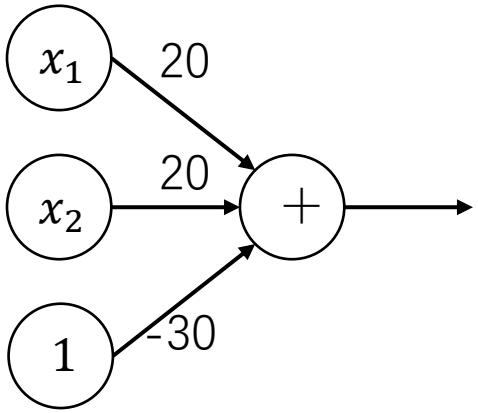
■ 当这些网络搭配不同的优化目标可以处理不同的任务

- 自然语言处理（循环神经网络，注意力网络）；图像处理（卷积神经网络，注意力网络）
- 生成式模型（变分自动编码机，对抗生成网络）
- 决策式模型（强化学习、循环神经网络、卷积神经网路、注意力网络）
- 推荐系统（多层感知机、强化学习、循环神经网络、卷积神经网路、注意力网络）

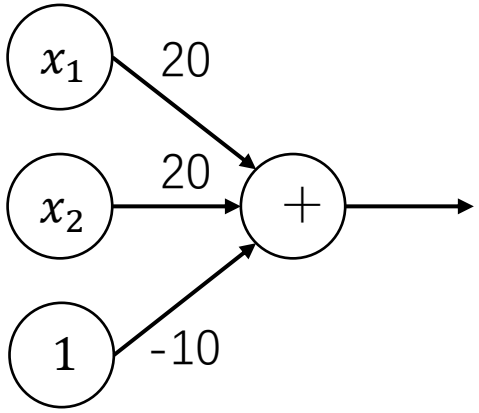
目录

2 从感知机到神经网络

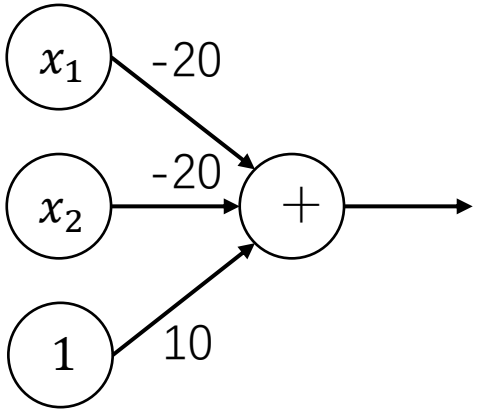
■ 简单的感知机



$(20x_1 + 20x_2 - 30)$



$(20x_1 + 20x_2 - 10)$



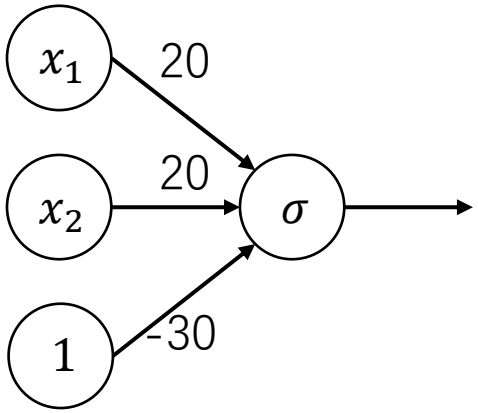
$(-20x_1 - 20x_2 + 10)$

符号函数： $\text{sign}(x) = \begin{cases} 1, & if\ x \geq 0 \\ -1, & otherwise \end{cases}$.

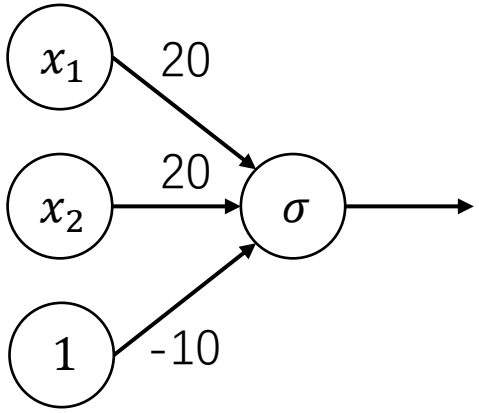
完整的神经元： $y = f(w^{\mathsf{T}}x + b)$

2 从感知机到神经网络

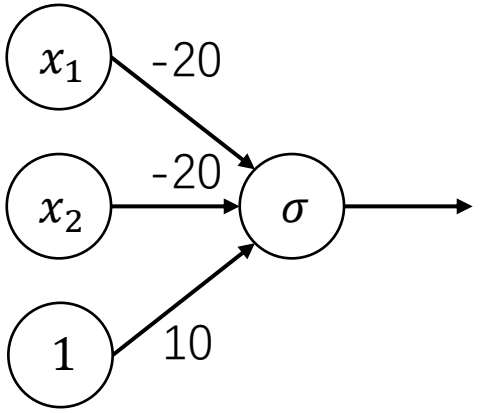
■ 带激活函数的感知机



$\sigma(20x_1 + 20x_2 - 30)$



$\sigma(20x_1 + 20x_2 - 10)$



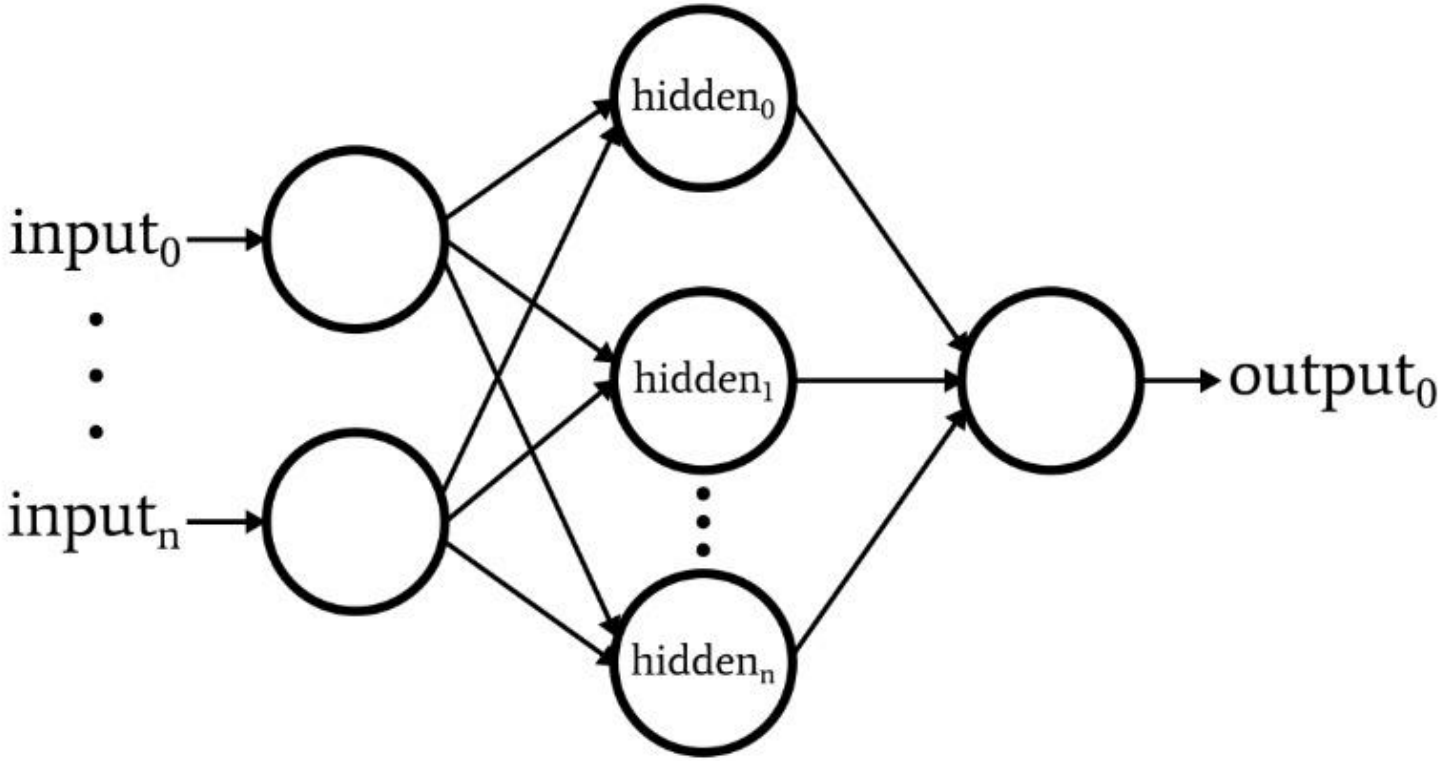
$\sigma(-20x_1 - 20x_2 + 10)$

激活函数: $\sigma(x) = \begin{cases} 1, & if\ x > 0 \\ 0, & otherwise \end{cases}$.

2 从感知机到神经网络

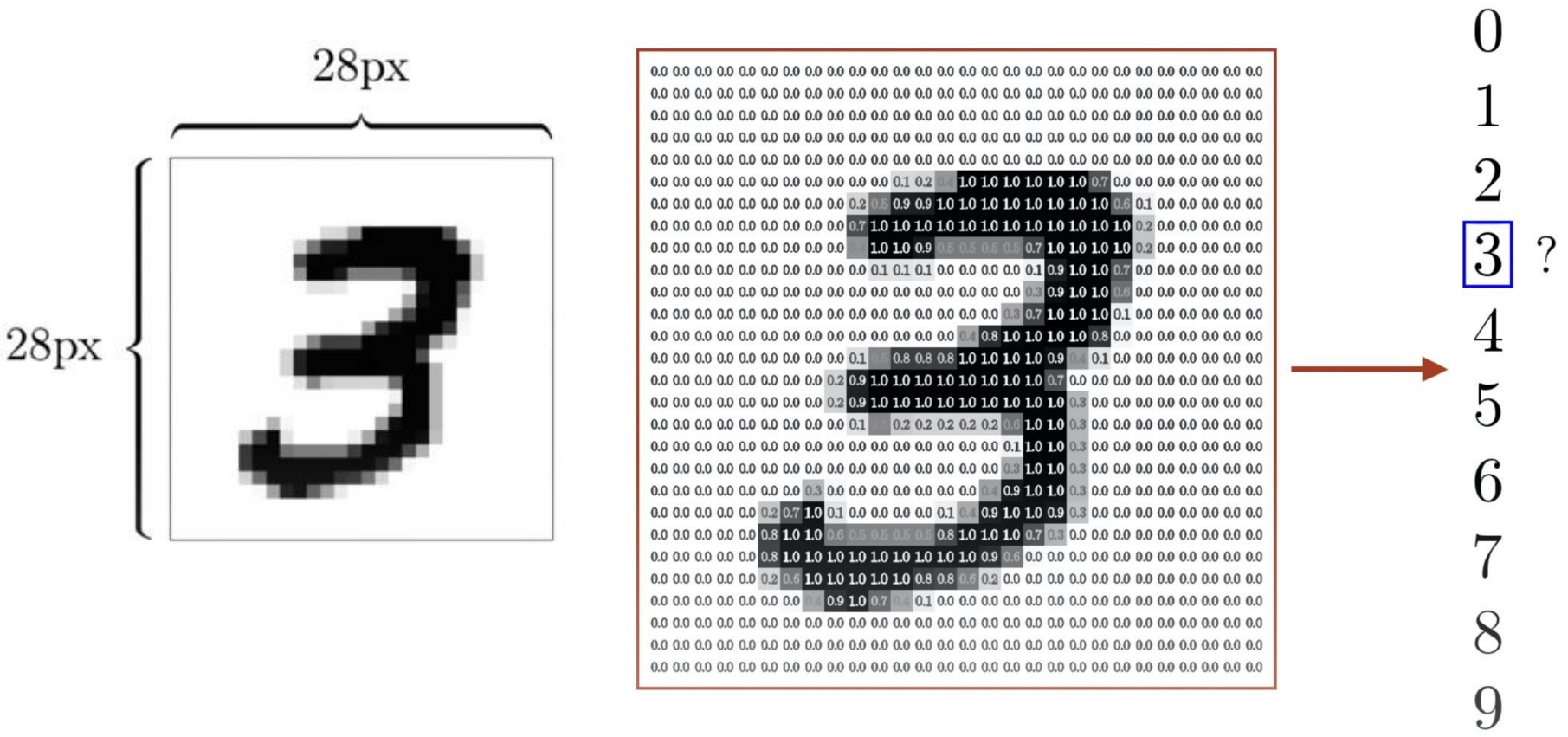
■ 多层感知机

- 隐藏层的输出 $h = \sigma(W^1 \mathrm{X} + b^1)$
- 输出层的输出 $y = \sigma(W^2 h + b^2)$
- 最终输出 $y = \sigma(W^T X + b)$



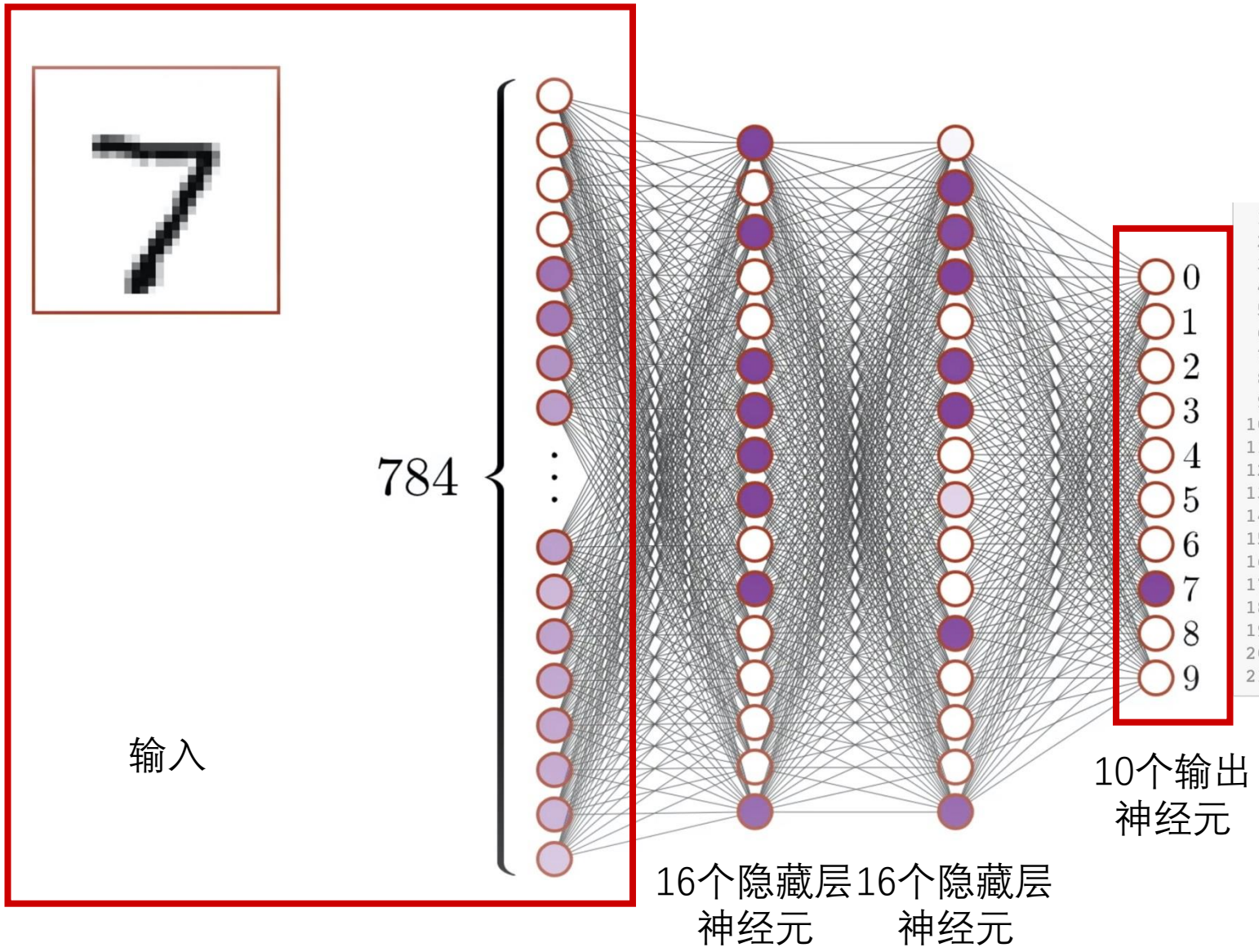
2 从感知机到神经网络

- 神经网络的使用分为三个主要部分：1、输入，2、传播，3、输出



https://www.youtube.com/watch?v=aircAruvnKk

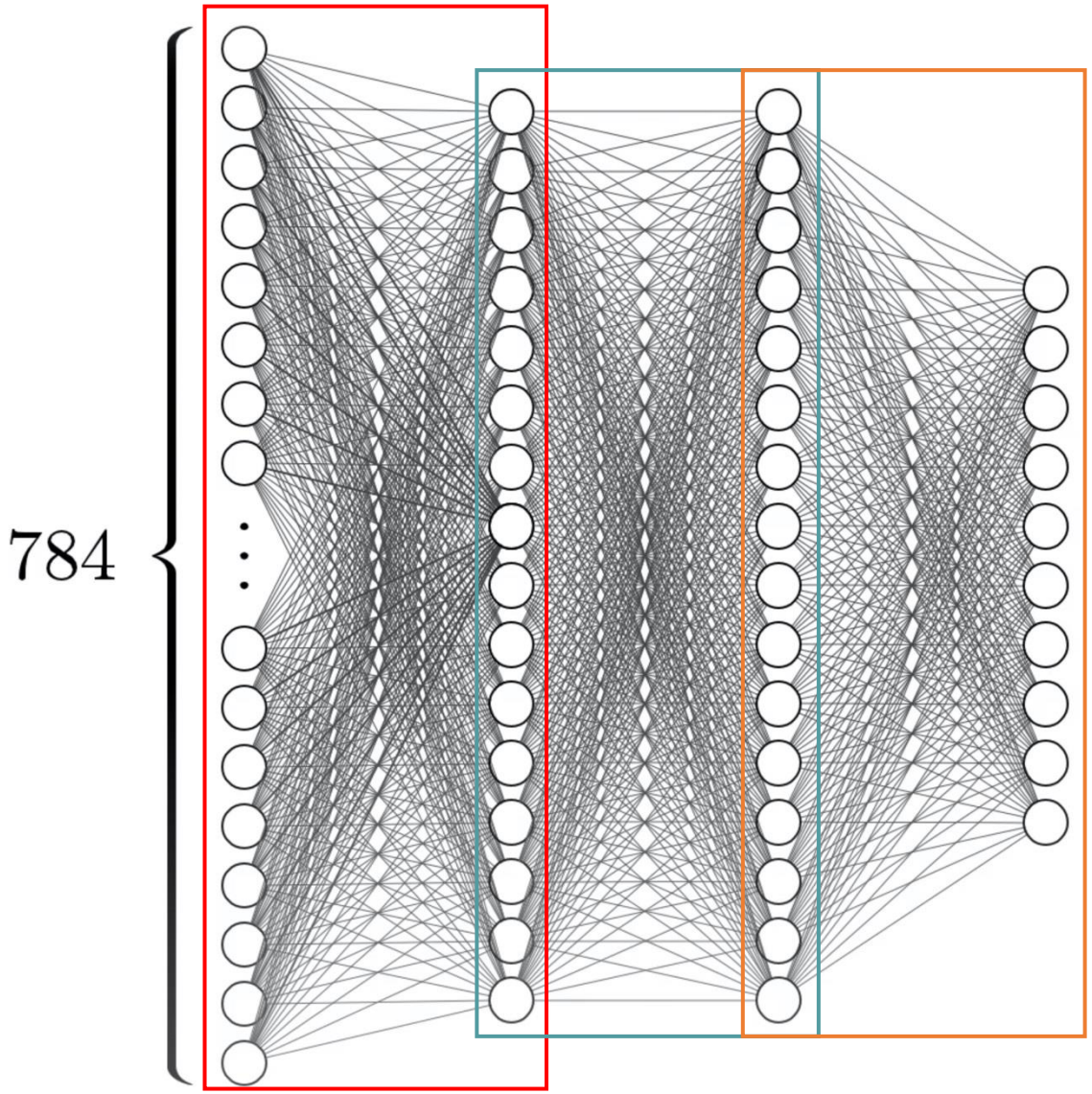
2 从感知机到神经网络




```python
class BNMLP(nn.Module):
    def __init__(self, hidden_ly = [16,16,10],droprate=0.5):
        super(BNMLP, self).__init__()
        mlp_list = []
        for i,hid_size in enumerate(hidden_ly):
            if i==0:
                pre = 28*28
            else:
                pre = hidden_ly[i-1]
            mlp_list.append(nn.BatchNorm1d(pre, momentum=0.1))
            linear = nn.Linear(pre, hidden_ly[i])
            nn.init.xavier_uniform_(linear.weight)
            mlp_list.append(linear)
            mlp_list.append(nn.ReLU())
            if i != len(hidden_ly)-1:
                mlp_list.append(nn.Dropout(p=droprate))
        self.mlp = nn.Sequential(*mlp_list)

    def forward(self, imgs):
        outputs = self.mlp(imgs)
        return outputs
```

2 神经网络参数数量



$$784 \times 16 + 16 \times 16 + 16 \times 10$$

weights

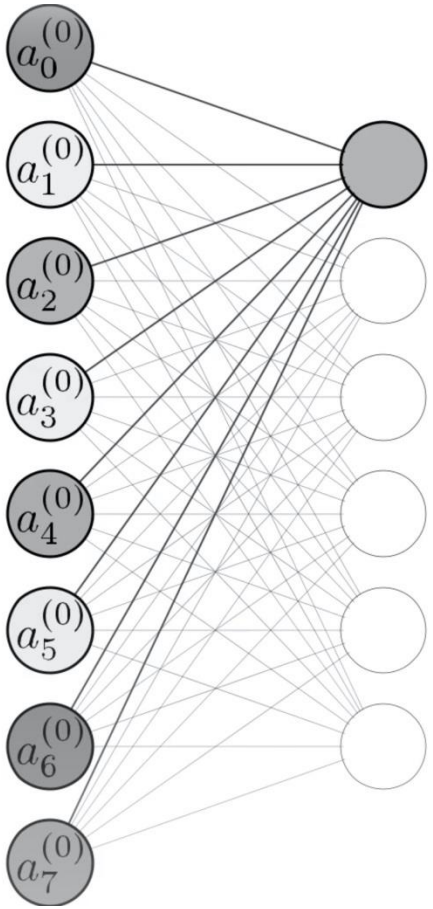
$$16 + 16 + 10$$

biases

$$13{,}002$$

2 从感知机到神经网络

■ 相邻两层神经元之间的信息传递



Sigmoid

$$a_0^{(1)} = \sigma\left(w_{0,0}\, a_0^{(0)} + w_{0,1}\, a_1^{(0)} + \cdots + w_{0,n}\, a_n^{(0)} + b_0 \right)$$

Bias

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right)$$

$$a^{(l+1)} = \sigma\left(W^{(l)} a^{(l)} + b^{(l)}\right)$$

目录

3 神经网络激活函数

- 激活函数：

  - 激活函数（Activation Function）是一种添加到人工神经网络中的函数，旨在帮助网络学习数据中的复杂模式。 类似于人类大脑中基于神经元的模型，激活函数最终决定了要发射给下一个神经元的内容。 在人工神经网络中，一个节点的激活函数定义了该节点在给定的输入或输入集合下的输出。

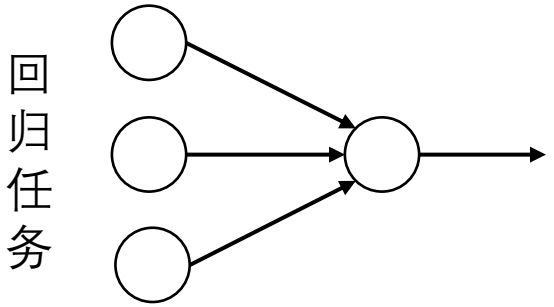
  - 如果不用激活函数（其实相当于恒等激活函数$f(x)=x$），在这种情况下每一层节点的输入都是上层输出的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，这种情况就是最原始的感知机（Perceptron）了，那么网络的逼近能力就相当有限。正因为上面的原因，我们决定引入非线性函数作为激励函数，这样深层神经网络表达能力就更加强大（不再是输入的线性组合，而是几乎可以逼近任意函数）。

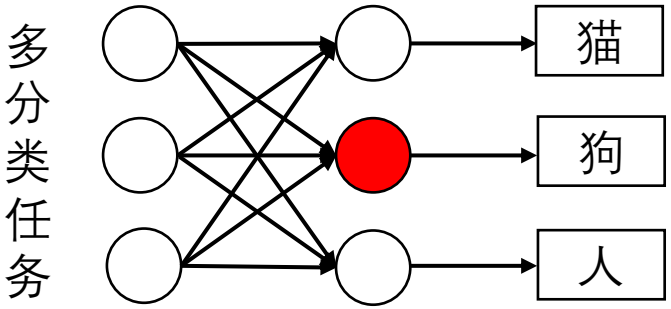
$$a^{(l+1)}=W^{(l)}a^{(l)}+b^{(l)}=W^{(l)}W^{(l-1)}\cdots W^{(0)}a^{(0)}+b^{(l)}+W^{(l)}b^{(l-1)}+\cdots=Wa^{(0)}+B$$

3 神经网络激活函数

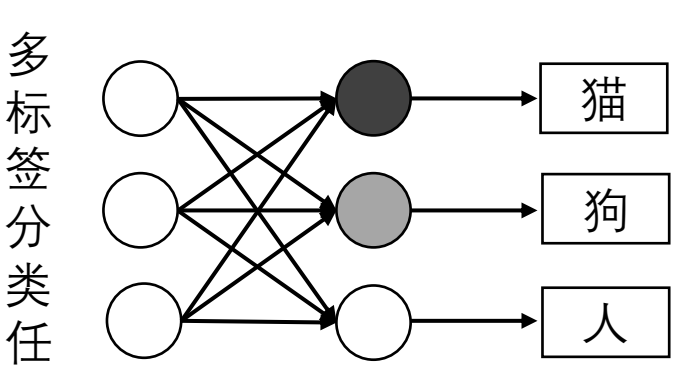
■ 神经网络的输出函数

- 神经网络输出主要分为几大类别，与我们的机器学习任务相关
  - 预测：回归，分类
  - 还原：图片生成，机器翻译
  - 决策：Making decisions
  - …





眼前这个动物是猫还是狗还是人



图片里有哪几种动物（共存的）

3 神经网络激活函数

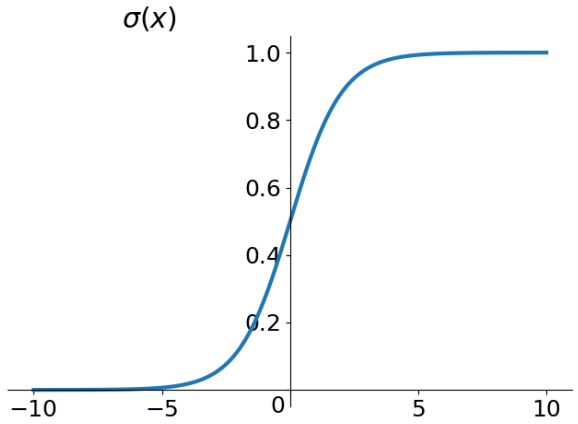
Sigmoid函数

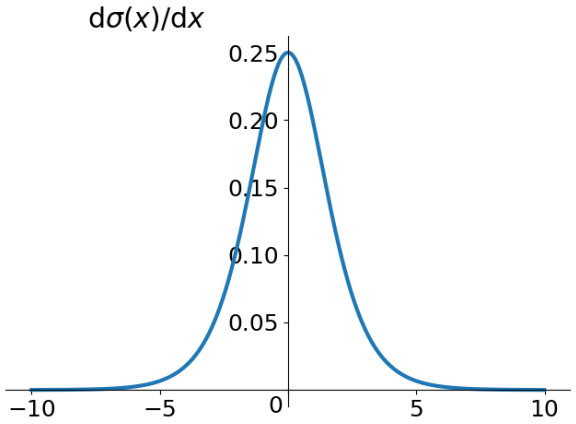
- 函数表达式：$\sigma(x)=\frac{1}{1+e^{-x}}$

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

- 将全部实数值映射到$(0,1)$区间

特别的，如果是非常大的负数，那么输出就是0；如果是非常大的正数，输出就是1.





3 神经网络激活函数

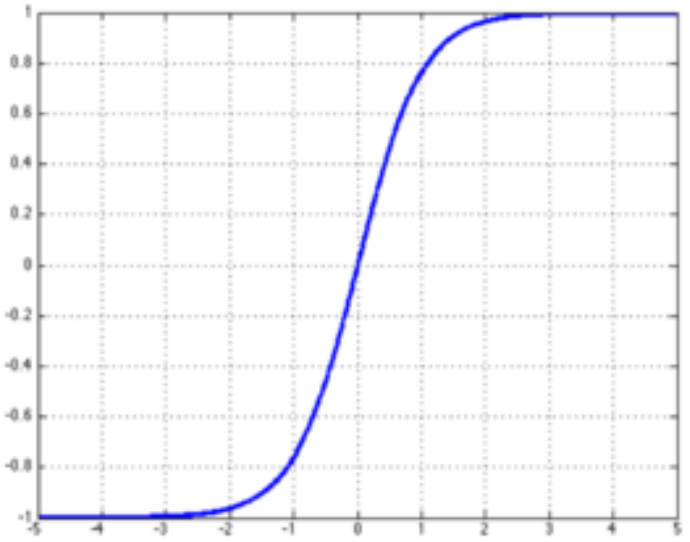
tanh函数

- 函数表达式：$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

  $tanh'(x) = 1 - tanh^2(x)$

- 将全部实数值映射到$(-1,1)$区间

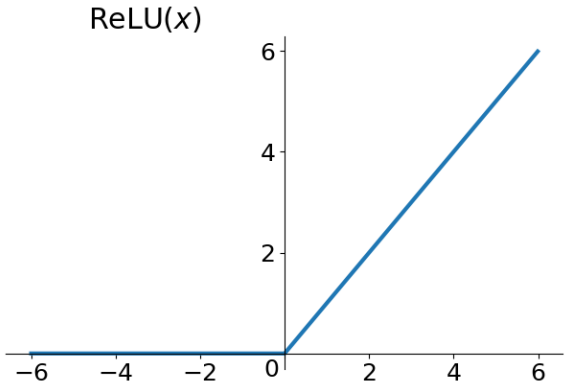
特别的，如果是非常大的负数，那么输出就是1；如果是非常大的正数，输出就是-1.

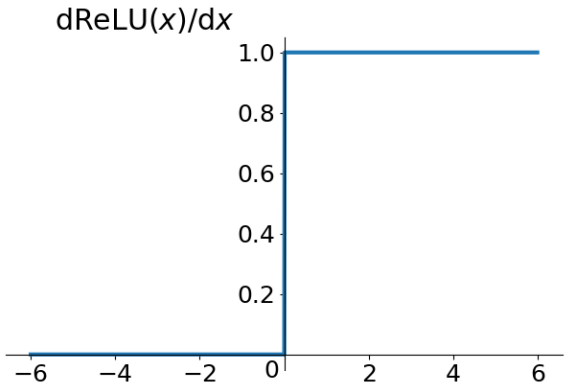


3 神经网络激活函数

Relu函数

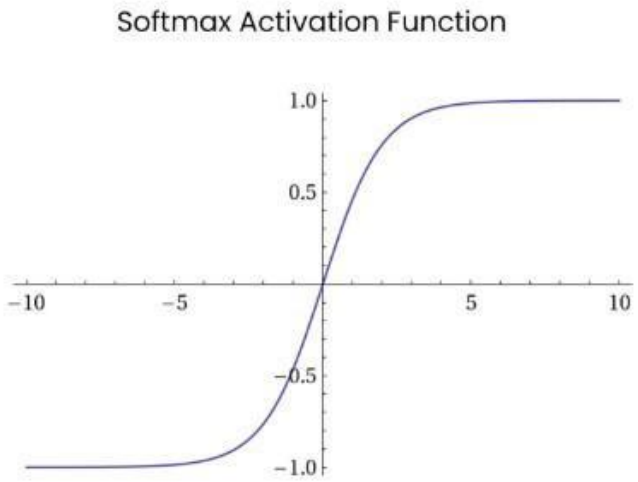
- Rectified Linear Unit 函数表达式：$\mathrm{ReLU}(x) = \max(0, x)$
- ReLU函数其实就是一个取最大值函数，注意这并不是全区间可导的，但是我们可以取sub-gradient。
- 对于ReLU函数, 当 x>0的时候，其导数为1；当x<0时，其导数为0。 则ReLU函数在 x=0的次梯度是 $c \leq \frac{f(x)-f(x_0)}{x-x_0}$，这里是次梯度可以有多个,可以取0,1之间的任意值。工程上为了方便取c=0即可。

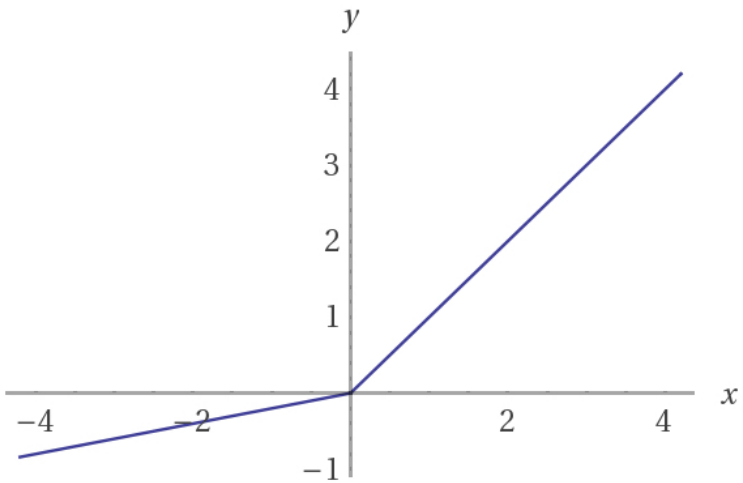




3 神经网络的激活函数

- 其他还有哪些激活函数?
  - Softmax函数
  - Leaky ReLU 函数
  - Maxout 函数
  - ……





目录

4 神经网络损失函数

- 回归损失函数在回归模型中，我们的神经网络将为每个我们试图预测的连续值提供一个输出节点。 我们用于回归模型的最流行的损失函数是均方误差损失函数。 假设有n个数据点：

$$Loss = \frac{1}{2n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2$$

- 多分类，假设总共有K个类别：

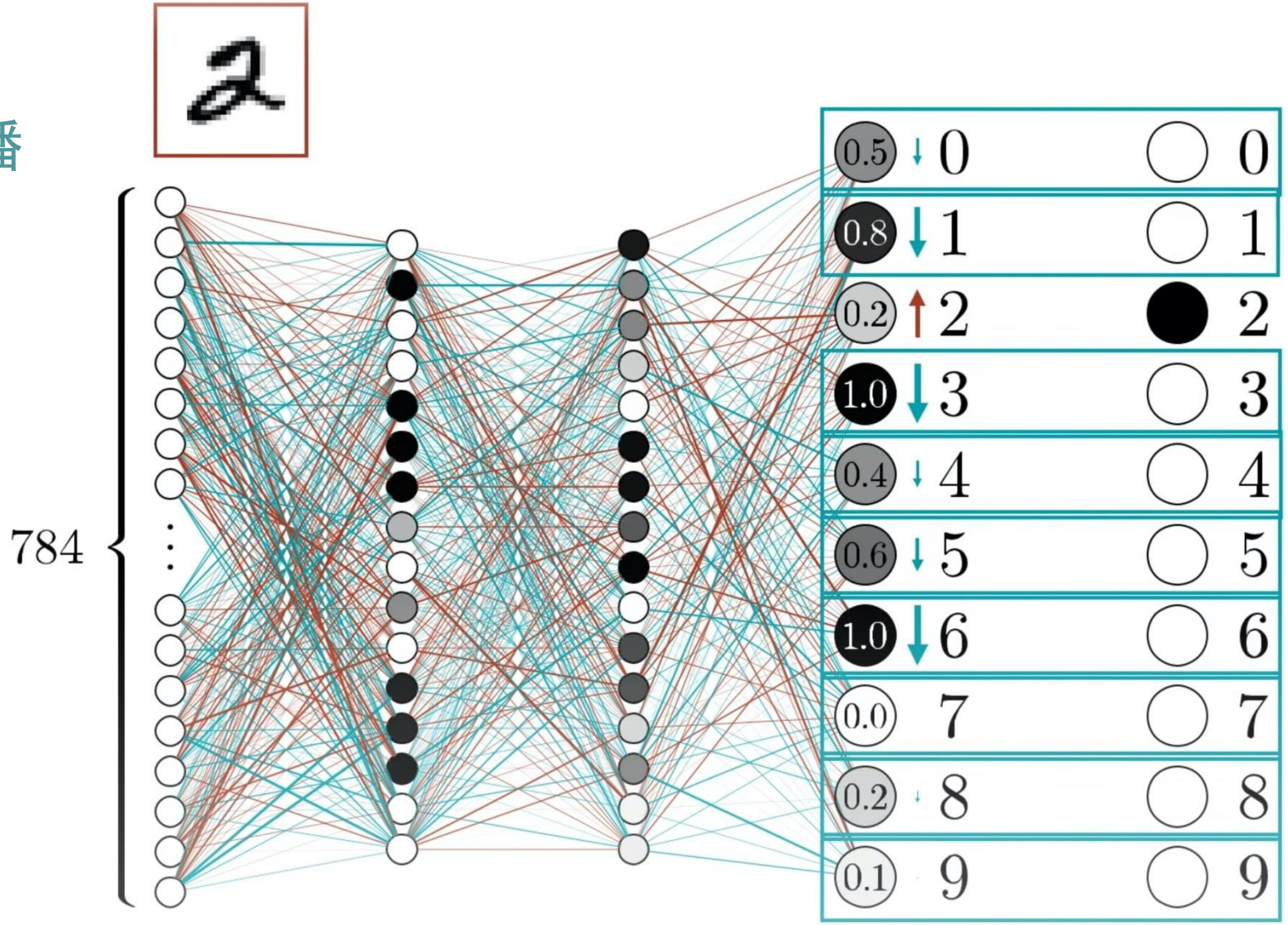
$$Loss = \sum_{i=0}^{n}\sum_{k=0}^{K} y_{i,k} \cdot -\log(\hat{y}_{i,k})$$

- 对于二分类问题中，我们可以得到如下式子：

$$Loss = \sum_{i=0}^{n} y_i \cdot -\log(\hat{y}_i) + (1 - y_i)(-\log(1 - \hat{y}_i))$$

4 神经网络损失函数

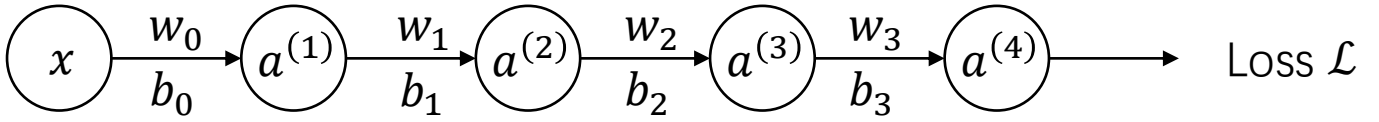
■ 神经网络的反向传播



4 神经网络损失函数

■ 神经网络的反向传播

- 假设每一层只有一个神经元且对于每一层$a^{(l+1)} = \sigma(w_l a^{(l)} + b_l)$，其中$\sigma(\cdot)$为sigmoid函数

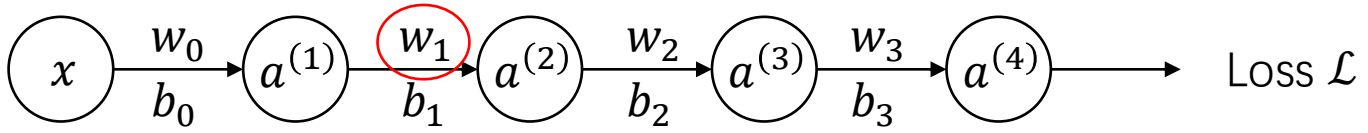


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \cdot \sigma'\left(w_3 a^{(3)} + b_3\right) a^{(3)}$$

4 神经网络损失函数

■ 神经网络的反向传播

■ 假设每一层只有一个神经元且对于每一层$a^{(l+1)} = \sigma(w_l a^{(l)} + b_l)$，其中$\sigma(\cdot)$为sigmoid函数



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \cdot \sigma'\left(w_3 a^{(3)} + b_3\right) a^{(3)}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \cdot \sigma'\left(w_3 a^{(3)} + b_3\right) w_3 \cdot \sigma'\left(w_2 a^{(2)} + b_2\right) a^{(2)}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial a^{(4)}} \frac{\partial a^{(4)}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial w_1}$$

$$= \frac{\partial \mathcal{L}}{\partial a^{(4)}} \cdot \sigma'\left(w_3 a^{(3)} + b_3\right) w_3 \cdot \sigma'\left(w_2 a^{(2)} + b_2\right) w_2 \cdot \sigma'\left(w_1 a^{(1)} + b_1\right) a^{(1)}$$

$$w_1 = w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w_1}$$

目录

5 神经网络优化器

- 给定计算好的Loss，神经网络的优化器将误差反向传播给神经网络中的所有参数，并通过计算出的更新值对参数进行更新。

- 常用的方法有：
  - SGD
  - Momentum
  - Adagrad
  - RMSprop
  - Adam

```
# 定义优化器
optimizer = torch.optim.Adam(per.parameters(), lr=0.1)
```

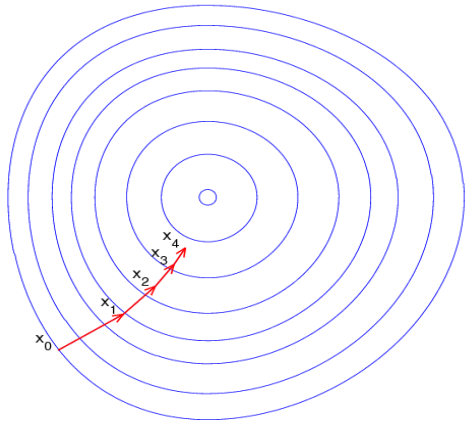
5 神经网络优化器

■ 梯度下降法

- 一个一阶最优化算法，通常也称为最陡下降法。 要使用梯度下降法找到一个函数的局部极小值，必须向函数上当前点对应梯度（或者是近似梯度）的反方向的规定步长距离点进行迭代搜索。如果相反地向梯度正方向迭代进行搜索，则会接近函数的局部极大值点；这个过程则被称为梯度上升法。

- 梯度的本意是一个向量（矢量），表示某一函数在该点处的方向导数沿着该方向取得最大值，即函数在该点处沿着该方向（此梯度的方向）变化最快，变化率最大（为该梯度的模）。

$$w_1 = w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w_1}$$

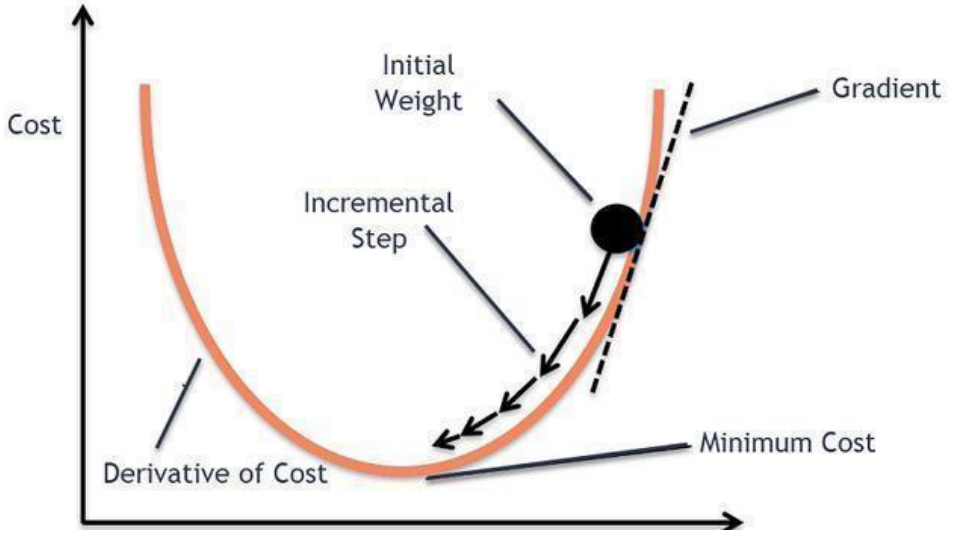


5 神经网络优化器

- SGD，又称 mini-batch SGD，每次利用一小批样本计算平均损失，通过在mini-batch规模的样本上计算出的损失对参数进行，可以降低参数更新时的方差，使得收敛更加稳定，另一方面可以充分地利用深度学习库中高度优化的矩阵操作来进行更有效的梯度计算。通常来说batch的大小为2的k次方，一般来说，针对不同大小的数据集，我们通常从128开始尝试，并以2的倍数向上增加或向下减少。

- 假设损失函数为$J(\theta)$，则参数更新公式如下：

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$



5 神经网络优化器

- Adam是目前最常用的优化方法，强烈推荐当成默认的优化方法来用。Adam结合了RMSprop中的梯度的平方和以及Momentum中的动量梯度。对学习率不敏感，且能够自动的对学习率进行调整。如果 mt 和 vt 被初始化为 0 向量，那它们就会向 0 偏置，偏差校正的 mt 和 vt 来抵消这些偏差：

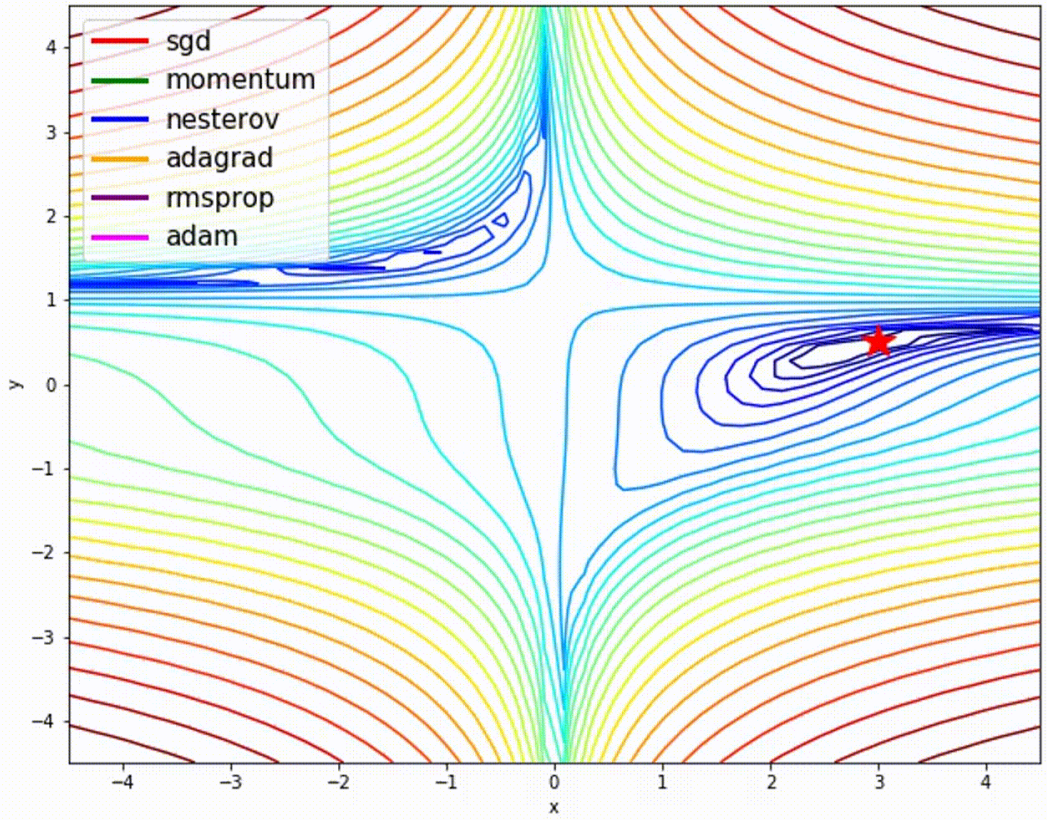
- 假设损失函数为$J(\theta)$，则参数更新公式如下：

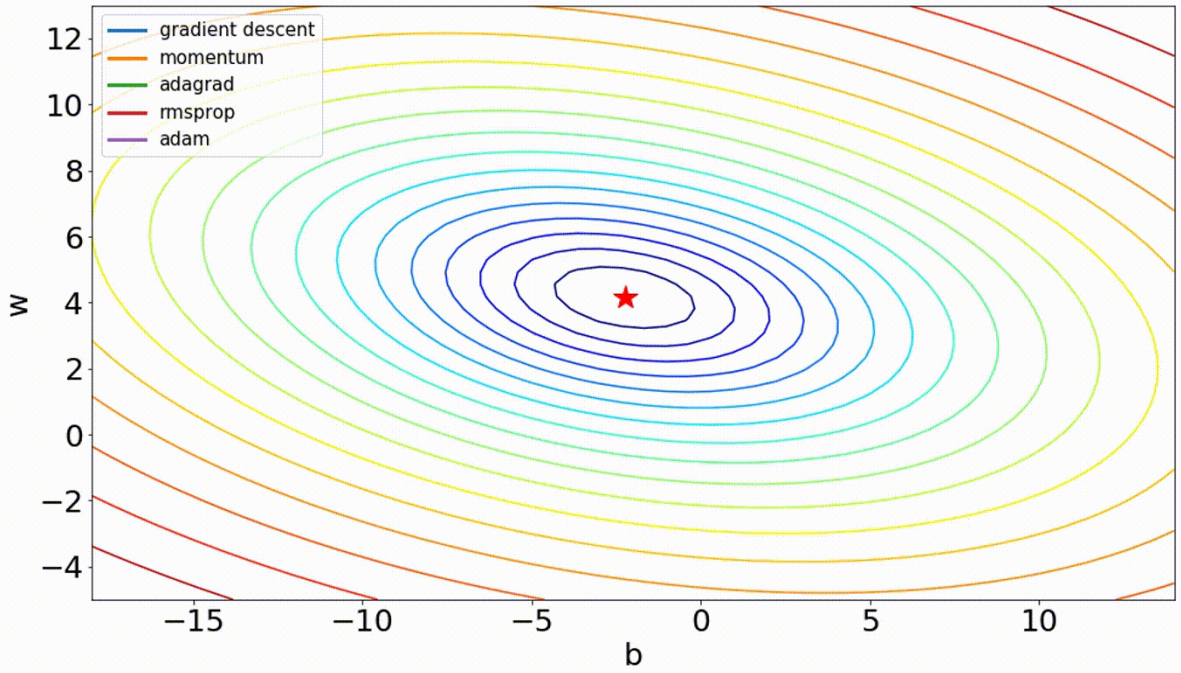
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

- 通常来说，β_1取值为0.9，β_2 取值为0.999，η 取值为0.001。但是，Adam容易收敛到局部最优，如果对效果有强迫症或者用于刷分的话。可以使用Momentum，结合手调学习率。

5 神经网络的优化器——效果对比





练习

- 学习与实践《动手学深度学习》课程
- 链接：https://zh-v2.d2l.ai/chapter_preface/index.html
- 学习内容（Pytorch版）：
  - 1. 前言
  - 2. 预备知识
  - 3. 线性神经网络
  - 4. 多层感知机
  - 5. 深度学习计算

问题？

暨南大學
JINAN UNIVERSITY