

Lecture 1: 预备知识

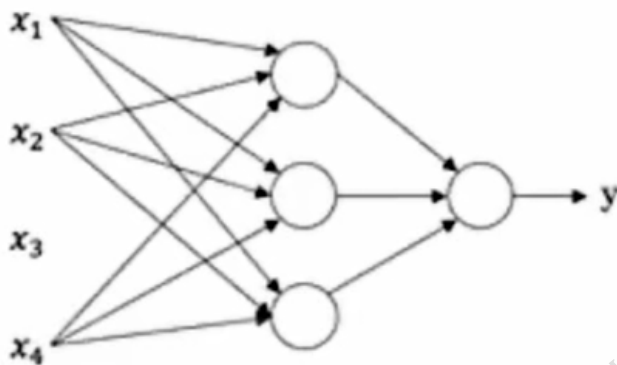
- 课程: [跟李沐学AI的个人空间-跟李沐学AI个人主页-哔哩哔哩视频](#)
- 书籍: [《动手学深度学习》 — 动手学深度学习 2.0.0 documentation](#)

神经网络介绍

概念: 神经网络介绍

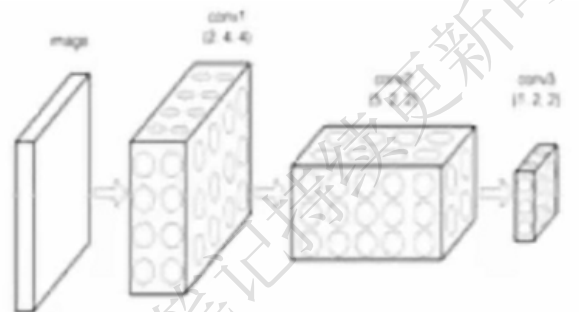
- 概念: 标准神经网络

- Standard NN



- 概念: CNN

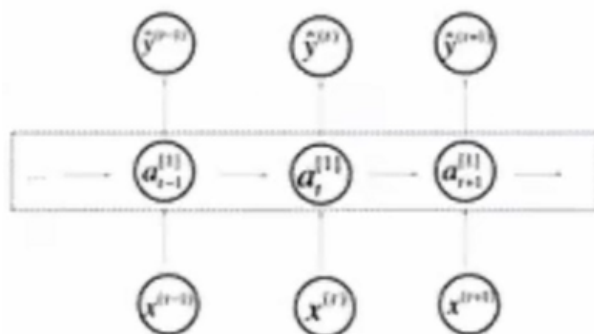
- Convolutional NN 卷积神经网络;



- 适用于图像处理;

- 概念: RNN

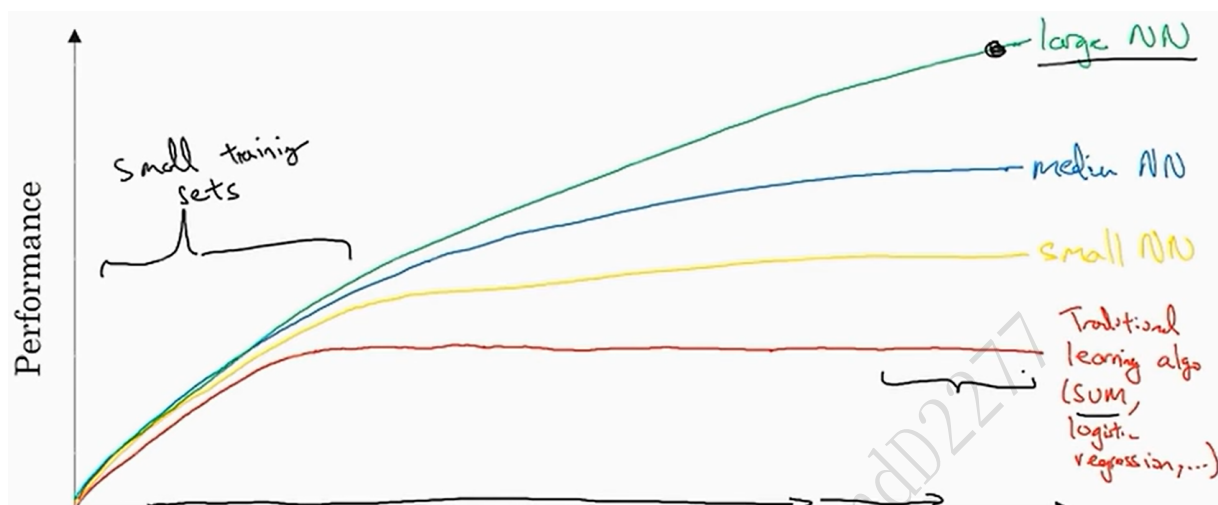
- Recurrent NN



- 适用于线性序列数据处理;

概念: 为什么深度学习发展的这么好

- 概念：
 - 在数据量较小的时候，传统机器学习的算法和神经网络的算法并没有明显的强弱，更多取决于对于特征的提取；
 - 而当数据量十分大的时候，越大的神经网络、越多的数据，运行的就越好；
- 图示：



二元分类

数学符号

概念： 数学符号定义

- 输入数据：

-

$$\mathbf{x} \in \mathbb{R}^{n_x}$$

- 输出数据（二分类）：

-

$$y \in \{0, 1\}$$

- m 个数据的训练集：

-

$$\left\{ \left(x^{(1)}, y^{(1)} \right), \left(x^{(1)}, y^{(2)} \right), \dots, \left(x^{(m)}, y^{(m)} \right) \right\}$$

- 其中上标括号指的是数据；

- 输出数据的合并写法：

-

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \dots & y^{(m)} \end{bmatrix}$$

逻辑回归

概念： 线性回归和逻辑回归

- 概念：

- 以下是线性回归的定义，但不符合二分类问题；

Given x , want $\hat{y} = P(y = 1 | x)$

$x \in \mathbb{R}^{n_x}$

Parameters: $w \in \mathbb{R}^{n_x}, b \in \mathbb{R}$.

Output $\hat{y} = w^T x + b$

- 二分类问题需要使用逻辑回归, y 帽的值应该是 0~1 之间的数字, 表示概率, 因此应到使用 sigmoid 函数;

- 概念: 逻辑回归

- 使用 sigmoid 函数, 传入线性回归的方程:

$$\hat{y} = G(\underbrace{\omega^T x + b}_z)$$

- sigmoid:

$$G(z) = \frac{1}{1 + e^{-z}}$$

- 注意:

- 在神经网络中, 将 b 和 w 当做相互独立的参数会更加简单;

逻辑回归的代价函数

概念: 代价函数

- 概念:

- 误差函数可以用来检测算法运行情况;
- 一种定义损失的方式: $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
- 但在逻辑回归中通常不这样做, 因为研究的问题是一个非凸问题, 所以优化问题可能会产生多个局部最优解;

- 公式: 逻辑回归中的损失函数

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

- 解释:

- 如果使用平方损失函数, 平方误差需要尽可能的小, 在此损失函数中, 也要让其尽可能的小;

$$\text{If } y = 1: L(\hat{y}, y) = -\log \hat{y} \leftarrow \text{want } \log \hat{y}$$

- 同理在 $y=0$ 时;

- 代价函数:

- 将所有的样本的损失加起来, 求平均;

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- 代价函数可以衡量你的参数 w 和 b 在训练集上的效果;

- 这里的代价函数 J 是一个凸函数；

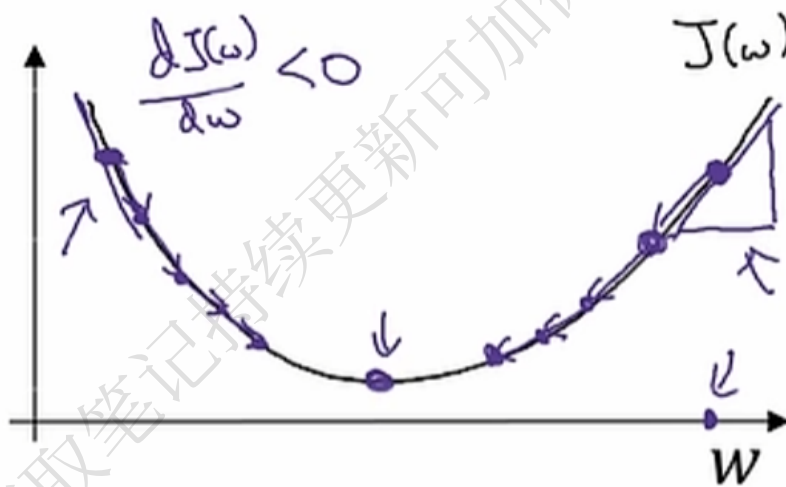
梯度下降法

概念：梯度下降法

- 概念：
 - 从初始点开始，然后朝最陡的下坡方向走一步，重复这个过程，收敛到这个全局最优值或接近全局最优值；
- 实现：用梯度下降法求参数 w 和 b
 - 公式：

$$w := w - \alpha \frac{dJ(w, b)}{dw}$$

- 同理求 b ；
- 注意：这里的求导符号应当使用偏导符号；
- 解释：
 - 这里的 α 表示学习率，学习率可以控制我们在每一次迭代或者梯度下降法中步长大小；
- 图示：



- 无论是从最左边为起始点还是最右边为起始点，因为是通过 w 减去变化率，最终都会下降到最低点，即全局最小点；

计算图

概念：计算图

- 概念：
 - 是用蓝色箭头画出来的，从左向右的计算；
- 举例：
 - 假设以下公式： $J(a, b, c) = 3(a + bc)$ ，则可以分解为以下计算图：

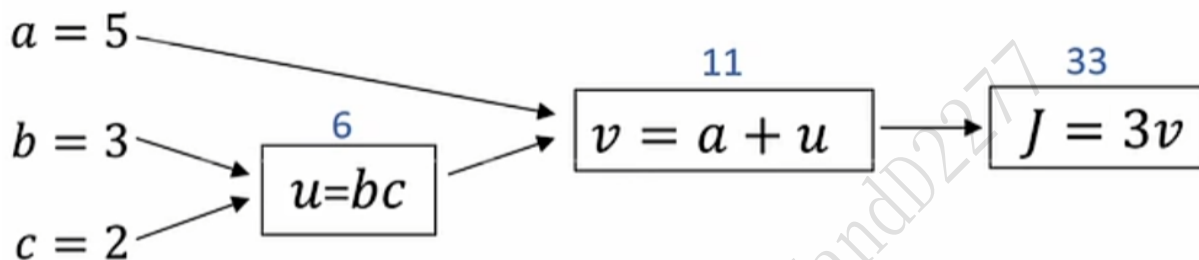
- $$J(a, b, c) = 3 \underbrace{\left(a + \underbrace{\underbrace{bc}_u}_{\frac{v}{j}} \right)}_j$$

$$u = bc$$

- 其中: $v = a + u$

$$J = 3v$$

- 可以利用上面三个计算, 设计计算图;



- 通过从左到右传播, 可以计算 **J** 的值;

- 概念: 反向传播与计算图求导

- 如果你想要计算最终输出变量对于变量 v 的导数, 比如 $\frac{dJ}{dv}$, 这就是一步反向传播;
- 如果涉及到多个变量, 则使用链式法则, 比如 $\frac{dJ}{da} = 3 = \frac{dJ}{dv} \frac{dv}{da}$

用计算图表示逻辑回归

概念: 用计算图表示逻辑回归

- 公式:

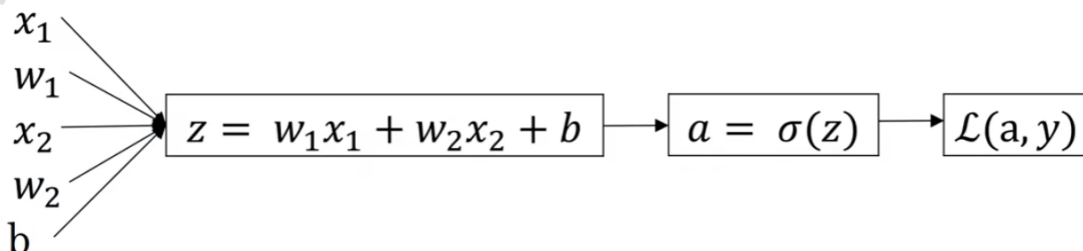
$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

- 分析:

- 对应的计算图:



- 对 w_1 和 w_2 分别进行反向传播、链式求导, 可以求得其在给定样本下的参数值;

计算 m 个样本的梯度下降

概念: 对于 m 个样本的梯度下降

- 分析：对于 m 个样本的梯度下降

- 已知当前的代价函数是损失函数的求和平均：

- $$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

- 因此可以利用级数的求导知识，直接对当前的代价函数求导，得到当前的 w 的平均导数，即全局梯度值：

- $$\frac{\partial}{\partial w_1} J(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{\partial}{\partial w_i} L(a^{(i)}, y^{(i)})$$

- 示例：代码实现

获取笔记持续更新可加微信：WandD2277

一次梯度下降()

{

$J = 0$; $dw_1 = 0$; $dw_2 = 0$; $db = 0$;

for $i = 1$ to m

{

$$z^{(i)} = w^T x^{(i)} + b ;$$

$$a^{(i)} = \sigma(z^{(i)}) \quad // \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

$$J += - [y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

}

$J / = m$; // 取 J 的平均值

$dw_1 / = m$; $dw_2 / = m$; $db / = m$;

$$w_1 := w_1 - \alpha \cdot dw_1 ;$$

$$w_2 := w_2 - \alpha \cdot dw_2 ;$$

$$b := b - \alpha \cdot db ;$$

} $\Rightarrow$ 一次梯度更新

}

向量化

概念：向量化 **vectorization**

• 引入：

- 在上面的分析中，只假设有 w_1 和 w_2 两个参数。如果有 m 个参数，则此时算法复杂度将非常大，会有两层 for 循环；
- 因此可以将参数向量化，来消除代码中的显式 for 循环；
- 即： **Whenever possible, avoid explicit for-loops**

- 概念：什么是向量化

- 假设当前要进行指数运算：
$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \quad u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}.$$

- 使用非向量化的版本计算：

```
u = np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

- 使用向量化的版本计算：

- `import numpy as np`
- `u = np.exp(v)`
- 这里将会对向量 `v` 里面的每一个元素进行指数运算；

向量化应用于逻辑回归

概念：向量化应用于梯度回归

- 分析：梯度回归的代码
- 非向量化的代码：

```
J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to n:
    z(i) = wTx(i) + b
    a(i) = σ(z(i))
    J += -[y(i) log ŷ(i) + (1 - y(i)) log(1 - ŷ(i))]
    dz(i) = a(i)(1 - a(i))
    dw1 += x1(i) dz(i)
    dw2 += x2(i) dz(i)
    db += dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m, db = db/m
```

- 向量化的代码进行的变化：

- `dw = np.zeros((n - x), 1)` // 将dw变成n维的零矩阵
- `dw += x(i) dz(i)`
- `dw /= m`

- 分析：逻辑回归中的向量化
- 原本：

- $$z^{(1)} = w^T x^{(1)} + b \quad z^{(2)} = w^T x^{(2)} + b \quad z^{(3)} = w^T x^{(3)} + b$$

$$a^{(1)} = \sigma(z^{(1)}) \quad a^{(2)} = \sigma(z^{(2)}) \quad a^{(3)} = \sigma(z^{(3)})$$

- 向量化:

- 数学公式:

- $$Z = [z^{(1)} z^{(2)} \dots z^{(m)}] = w^T X + [b \quad b \dots b] = [w^T x^{(1)} + b, \quad w^T x^{(2)} + b, \quad \dots \quad w^T x^{(m)} + b]$$

- Python 代码:

- $$Z = \text{np.dot}(w \cdot T, X) + b$$

- 注意: 这里的 b 是一个标量, 也就是一个 1×1 的矩阵, 但是为了与 $\text{np.dot}(w \cdot T, X)$ 相加, 因此 Python 会自动将 b 广播为一个 $1 \times m$ 的矩阵;

概念: 使用向量化计算 m 个训练数据的梯度

- 分析:

- $dz^{(1)} = a^{(1)} - y^{(1)} \quad dz^{(2)} = a^{(2)} - y^{(2)} \dots$
 - $dz = [dz^{(1)} dz^{(2)} \dots dz^{(m)}]$
 - $A = [a^{(1)} \dots a^{(n)}], \quad Y = [y^{(1)} \dots y^{(n)}]$
 - $dz = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(1)} \quad \dots]$

- 向量化求导示例:

- 非向量化时: $db = \frac{1}{m} \sum_{i=1}^n dz^{(i)}$
 - Python 中: $1/m \text{ np.sum}(dz)$

- 代码:

- 未向量化版本:

$$J = 0, \quad dw1 = 0, \quad dw2 = 0, \quad db = 0$$

for i = 1 to n:

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$

$$dz^{(i)} = a^{(i)}(1 - a^{(i)})$$

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

$$J = J/m, \quad dw_1 = dw_1/m, \quad dw_2 = dw_2/m, \quad db = db/m$$

- 向量化版本:

$$Z = w^T X + b$$

$$= \text{np.dot}(w \cdot T, X) + b$$

$$A = \sigma(z)$$

$$dz = A - Y$$

-

$$dw = \frac{1}{m} \times dz^T$$
$$db = \frac{1}{m} np \cdot \sin(dz)$$

-

$$w := w - \alpha dw$$
$$b := b - \alpha db$$

获取笔记持续更新可加微信: WandD2277