第一节
概率论基础与蒙特卡洛

基础相关内容介绍

[Xin Li] 21/11

本节概述

- 1.1 概率论基础
  - 随机变量、观测值
  - 概率质量函数、概率密度函数
  - 期望、随机抽样
- 1.2 蒙特卡洛—通过随机样本估算真实值
  - 估算π值
  - 近似定积分
  - 近似期望

1.1 概率论基础

- 基本概念1

  - **随机变量：不确定性，其值取决于随机事件的结果**
  - **观测值：具体的结果，具有直观上的唯一性**

例：随机事件——抛硬币
抛一枚硬币，其正面朝上记为0，反面朝上记为1。
则 抛硬币的结果 便是随机变量（记为X，其值可能为1，也可能为0，具体值取决于最后抛完硬币的结果）。

在硬币抛完之后，我们就能够 观察到哪一面朝上 ，这时X便有了一个确定的值，该值即为观测值（记为x）。比如重复抛3次硬币，得到3个观测值： $x_1 = 1$, $x_2 = 0$, $x_3 = 1$

同理，同学们可以自行类比掷骰子的例子

1.1 概率论基础

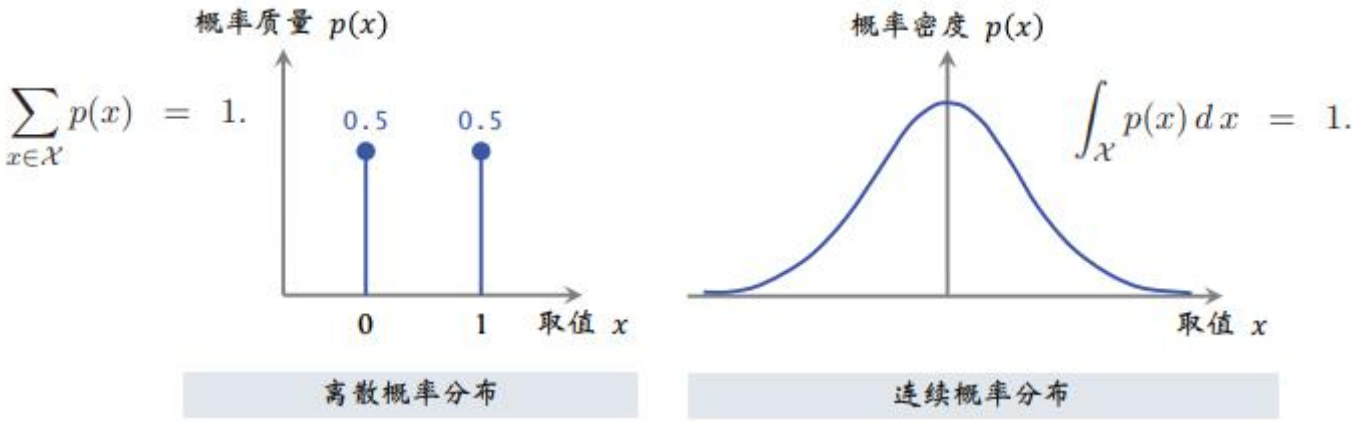
- 基本概念2
  - **概率质量函数(Probability Mass Function, PMF)：描述一个离散概率分布，即针对随机变量X是离散值（如 硬币面向、骰子点数、天气情况 等）的情况**
  - **概率密度函数(Probability Density Function, PDF)：描述一个连续概率分布，即针对随机变量X是连续值（如 温度、身高 等）的情况**

例：

在抛硬币的例子中，随机变量X的取值范围χ是集合{0,1}（离散值），其中取到各个随机变量的概率为$p(0)=0.5, p(1)=0.5$。则在这个例子中，X的概率质量函数便为：$p(0)=0.5, p(1)=0.5$。

以正态分布为例，随机变量X的取值范围χ为实数集合R（连续值），正态分布的概率密度函数为：

$$p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$



1.1 概率论基础

- 基本概念3
  - **期望：函数*f(x)*的期望定义为——设*P(X)*为X的概率密度函数/概率质量函数，则对于离散概率分布，*f(x)*的期望为：**

$$\mathbb{E}_{X\sim p(\cdot)}[f(X)] = \sum_{x\in\mathcal{X}} p(x)\cdot f(x).$$

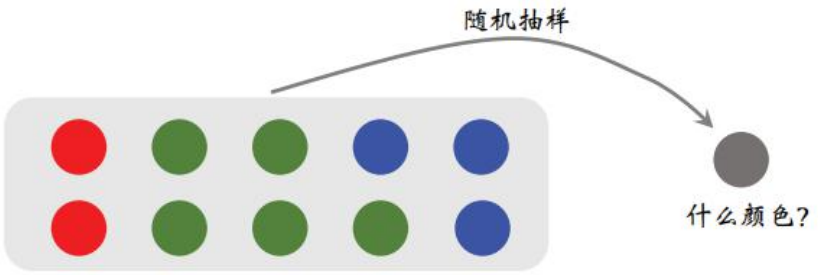
对于连续概率分布，*f(x)*的期望为：

$$\mathbb{E}_{X\sim p(\cdot)}[f(X)] = \int_{\mathcal{X}} p(x)\cdot f(x)\,dx.$$

  - **随机抽样：从总体N个单位中任意抽取n个单位作为样本，其中每个样本被抽中的概率相等。见小球抽取示例**

例：小球抽取

假设箱子内有10个小球（2红、5绿、3蓝），闭眼从箱子内随机抽取一个小球（每个小球被抽中的概率为1/10），当抽出小球后睁开眼睛时，便能观测到小球的颜色（比如红色），这个过程便叫做随机抽样。在上述随机抽样过程中，颜色是随机变量X，x=红色 是X的一个观测值。



1.1 概率论基础

假设箱子内有很多个球，其中红色球占20%，绿色球占50%，蓝色球占30%，此时抽到的球服从如下的离散概率分布：

P(红)=0.2，　　P(绿)=0.5，　　P(蓝)=0.3

用Python代码模拟上述 随机抽样 的过程，重复100次，输出抽样结果

```
from numpy.random import choice

samples = choice(['R', 'G', 'B'],
                 size=100,
                 p=[0.2, 0.5, 0.3])

print(samples)
```

随机变量的取值范围是集合 {R, G, B}。

重复抽样100次，函数返回长度为100的数组。

R, G, B 三种颜色的球被选中的概率分别是 0.2, 0.5, 0.3。

```
['B' 'R' 'R' 'G' 'B' 'B' 'G' 'B' 'G' 'G' 'G' 'R' 'R' 'G' 'G' 'B' 'R' 'G' 'G' 'B'
'B' 'G' 'B' 'G' 'G' 'R' 'G' 'G' 'B' 'R' 'G' 'G' 'R' 'G' 'G' 'R' 'G' 'G' 'G' 'G'
'G' 'B' 'G' 'B' 'R' 'R' 'G' 'G' 'G' 'B' 'B' 'G' 'R' 'R' 'B' 'G' 'G' 'G' 'B' 'B'
'G' 'G' 'G' 'B' 'B' 'G' 'G' 'B' 'G' 'G' 'B' 'G' 'R' 'G' 'B' 'G' 'R' 'B' 'B' 'G'
'G' 'G' 'R' 'G' 'G' 'R' 'R' 'G' 'G' 'G' 'G' 'G' 'B' 'G' 'B' 'G' 'G' 'G' 'R' 'B']
```

1.2 蒙特卡洛

- 方法理解
  - 大数定理的本质是用大样本数据计算出的频率来估计概率。
  - 蒙特卡洛方法是大数定理的一个经典应用。其核心思想是通过模拟出来的大量样本集或者随机过程来近似我们想要研究的实际问题对象。

1.2 蒙特卡洛

- 示例1：估算π值
  - 问题描述：假如我们不知道π值（≈3.1415926），我们能否用随机样本来估算π值？
  - 近似原理：

正方形面积 $a_1 = 2^2 = 4$；圆形面积 $a_2 = \pi \cdot 1^2 = \pi$。

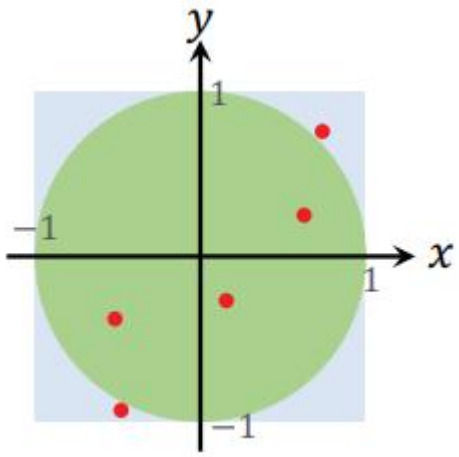
假设一（伪）随机数生成器可以均匀生成[-1,1]×[-1,1]之间的点，则该点落在圆形区域（$x^2 + y^2 \leq 1$）的概率为$p = \frac{a_2}{a_1} = \frac{\pi}{4}$。

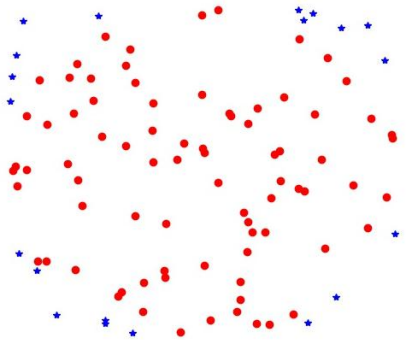
假设随机抽样了n个点，圆内点的数量为随机变量M，则M的期望为：

$$\mathrm{E}[M] = pn = \frac{\pi n}{4}$$

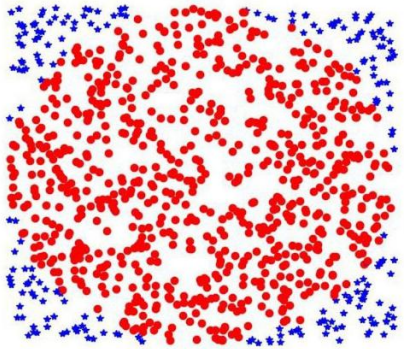
根据大数定理，当n足够大时，随机变量M的真实观测值m就会接近于$\frac{\pi n}{4}$，故

$$\pi \approx \frac{4m}{n}$$

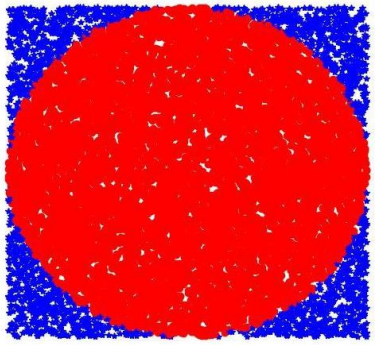




n=100



n=1000



n=10000

1.2 蒙特卡洛

- 示例2：近似定积分
  - 问题描述：给定一个函数 $f(x)=\frac{1}{x}$，要求计算f在区间1到2上的定积分：

$$S=\int_1^2 \frac{1}{x}dx$$

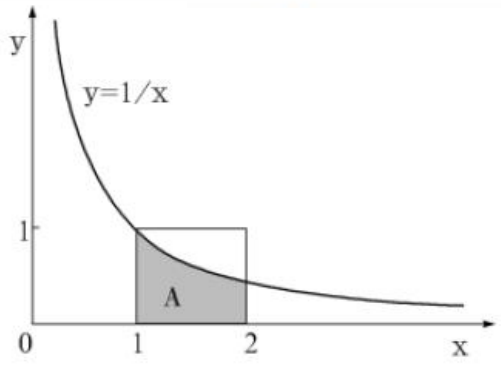
  - 近似原理：

上述积分本质是求下图阴影部分的面积。

首先在矩形区域[1,2]内做随机抽样得到n个随机点(x_i, y_i)，假设满足$\frac{1}{x_i} \geq y_i$的点有m个，可以得到近似公式为：

$$S=\int_1^2 \frac{1}{x}dx \approx \frac{m}{n} \times (2-1) \times (1-0)$$

与示例1类似，当n足够大时，$\frac{m}{n} \approx S=\int_1^2 \frac{1}{x}dx=\ln 2-\ln 1=\ln 2$



用该方法同样能估计出e的值，请同学们课下自行思考

1.2 蒙特卡洛

- 示例3：近似期望
  - 问题描述：定义X是d维随机变量，其取值范围是集合$\Omega \subset \mathrm{R}^{\mathrm{d}}$，函数$p(x)$=P(X=$x$)是X的概率密度函数（描述变量X在取值点x附近的可能性）。设$f: \Omega \to \mathrm{R}$是任意的多元函数，它关于X的期望是：

$$\mathbb{E}_{X\sim p(\cdot)}\Big[f(X)\Big] = \int_{\Omega} p(\boldsymbol{x}) \cdot f(\boldsymbol{x})\, d\,\boldsymbol{x}.$$

  - 近似原理：
    期望是定积分，可以采用示例2中的定积分方法求解，但现在已知$p(x)$，可以进行非均匀抽样。具体步骤为：
    1. 按照概率密度函数$p(x)$，在集合 Ω 上做非均匀随机抽样，得到 n 个样本，记作向量 $x_1, \ldots, x_n \sim p(\cdot)$。样本数量 n 由用户自己定，n 越大，计算量越大，近似越准确。
    2. 对函数$f(x_1), \ldots, f(x_n)$求平均：

$$q_n = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{x}_i).$$

    3. 返回q_n作为期望 $\mathbb{E}_{X\sim p(\cdot)}\Big[f(X)\Big]$ 的估计值。

第二节
深度学习基础

深度学习基本内容介绍

[Xin Li] 21/11

本节概述

- 2.1 线性模型
  - 线性回归——y为连续变量
  - 逻辑斯蒂（logistic）回归——y为二元离散变量
  - Softmax分类器——y为多元离散变量
- 2.2 神经网络
  - 全连接神经网络（多层感知机）——FC/MLP
  - 卷积神经网络——CNN
- 2.3 梯度下降与反向传播
  - 梯度下降——GD&SGD
  - 反向传播——BP算法原理

2.1 线性模型

- 线性回归

$$f(x) = w_1x_1 + w_2x_2 + \ldots + w_dx_d + b = x^Tw + b$$

$$x = [x_1, x_2, \ldots, x_d]^T ; w = [w_1, w_2, \ldots, w_d]^T ; b \subset R$$

例：房价预测

假设房屋有d个属性，将其记作向量x； w可以看做这些属性在房价中所占的权重；b视作市面上房价的均值或者中位数，该值与房屋的具体属性无关。

假如我们拥有了房屋销售价格（y;黄色部分）和房屋属性（x;绿色部分）数据（训练集），我们希望通过这些数据来计算最优的w和b，使其代入$f(x)$能够对房价做出较为准确的预测

$$L(w,b) = \frac{1}{2n}\sum_{i=1}^{n}\left[f(x_i;w,b) - y_i\right]^2 . \quad R(w) = \lambda\|w\|_2^2 \quad 或 \quad R(w) = \lambda\|w\|_1 .$$

$R(w)$为正则项，通常用来防止过拟合

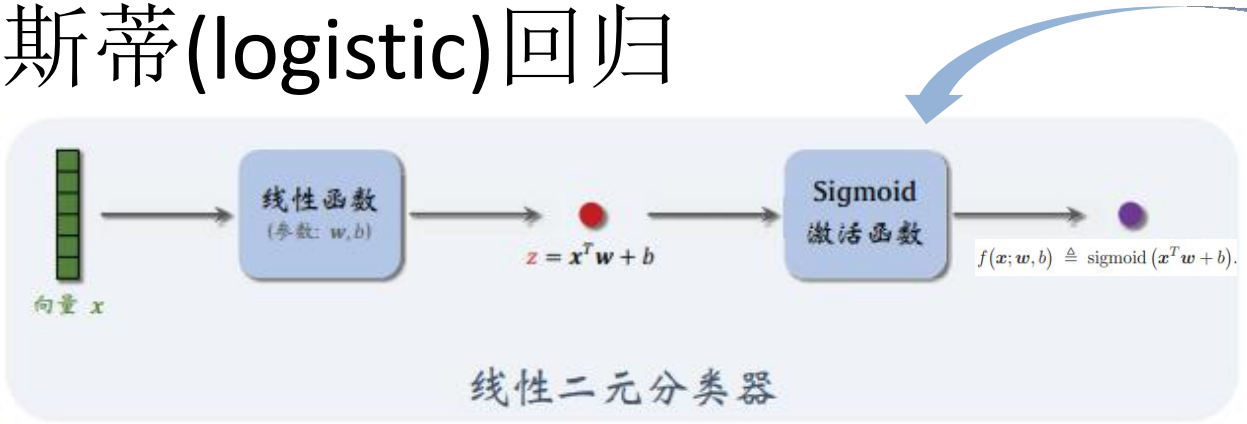
$$(w^\star, b^\star) = \underset{w,b}{\operatorname{argmin}} \; L(w,b) + R(w).$$

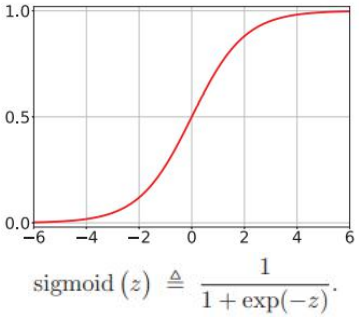
销售价格	卧室数	浴室数	房屋面积	停车面积	楼层数	房屋评分	建筑面积	地下室面积	建筑年份	修复年份	房屋所在纬度	房屋所在经度
545000	3	2.25	1670	6240	1	8	1240	430	1974	0	47.6413	-122.113
785000	4	2.5	3300	10514	2	10	3300	0	1984	0	47.6323	-122.036
765000	3	3.25	3190	5283	2	9	3190	0	2007	0	47.5534	-122.002
720000	5	2.5	2900	9525	2	9	2900	0	1989	0	47.5442	-122.138
449500	5	2.75	2040	7488	1	7	1200	840	1969	0	47.7289	-122.172
248500	2	1	780	10064	1	7	780	0	1958	0	47.4913	-122.318
675000	4	2.5	1770	9858	1	8	1770	0	1971	0	47.7382	-122.287
730000	2	2.25	2130	4920	1.5	7	1530	600	1941	0	47.573	-122.409
311000	2	1	860	3300	1	6	860	0	1903	0	47.5496	-122.279
660000	2	1	960	6263	1	6	960	0	1942	0	47.6646	-122.202
435000	2	1	990	5643	1	7	870	120	1947	0	47.6802	-122.298
350000	3	1	1240	10800	1	7	1240	0	1959	0	47.5233	-122.185
385000	3	2.25	1630	1598	3	8	1630	0	2008	0	47.6904	-122.347
235000	2	1	930	10505	1	6	930	0	1930	0	47.4337	-122.329
350000	3	1	1300	10236	1	6	1300	0	1971	0	47.5028	-121.77
1350000	4	1.75	2000	3728	1.5	9	1820	180	1926	0	47.643	-122.299
459900	3	1.75	2580	11000	1	7	1290	1290	1951	0	47.5646	-122.181
430000	6	3	2630	8800	1	7	1610	1020	1959	0	47.7166	-122.293
718000	5	2.75	2930	7663	2	9	2930	0	2013	0	47.5308	-122.184
220000	4	1	1440	8250	1	7	1440	0	1959	0	47.4325	-122.291
1500000	4	3.25	3470	5222	2	10	2830	640	2005	0	47.6845	-122.209
140000	3	1.5	1200	2002	2	8	1200	0	1966	0	47.4659	-122.189
200000	4	2.5	1720	8638	2	8	1720	0	1994	0	47.2704	-122.313
475000	4	1.75	1650	7775	1	7	1150	500	1950	0	47.715	-122.354
179000	3	1	920	5200	1	6	920	0	1969	0	47.3603	-122.085
400000	3	2.5	1460	1319	3	8	1460	0	2002	0	47.698	-122.349
348125	3	1	1400	8451	1.5	7	1400	0	1953	0	47.7719	-122.337

一般来说：
y表示数据的真实值
$f(x)$代表模型预测值

2.1 线性模型

- 逻辑斯蒂(logistic)回归





- **Sigmoid：输出范围：0-1，直观理解为置信率** $\text{sigmoid}(z) \triangleq \frac{1}{1+\exp(-z)}$.
- **交叉熵：衡量两个离散概率分布的差别，概率分布越接近则交叉熵越小**

$\boldsymbol{p} = [p_1, \cdots, p_m]^T$ 和 $\boldsymbol{q} = [q_1, \cdots, q_m]^T$ $\sum_{j=1}^m p_j = 1$, $\sum_{j=1}^m q_j = 1$ $H(\boldsymbol{p}, \boldsymbol{q}) = -\sum_{j=1}^m p_j \cdot \ln q_j$.

例：疾病检测——排查癌症

排查癌症时需要进行血检，血检中含有d项指标（白细胞数量、含氧量、各种激素含量等等），将其记作d维向量$\boldsymbol{x}$，医生需要基于$\boldsymbol{x}$来判断病人是否患有癌症。如果医生的判断为$\boldsymbol{y}$ =1，则要求病人进行进一步的检测；如果医生的判断为$\boldsymbol{y}$ =0，则表示该病人未患癌。

现假设已收集了n份血检报告$\boldsymbol{x}$和最终诊断$\boldsymbol{y}$作为训练集，与线性回归相同，我们同样需要计算最优的$\boldsymbol{w}$和$\boldsymbol{b}$，使其能够对疾病做出较为准确的预测。

$$L(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^n H\left(\begin{bmatrix} y_i \\ 1-y_i \end{bmatrix}, \begin{bmatrix} f(\boldsymbol{x}_i; \boldsymbol{w}, b) \\ 1-f(\boldsymbol{x}_i; \boldsymbol{w}, b) \end{bmatrix}\right) \qquad \min_{\boldsymbol{w}, b} L(\boldsymbol{w}, b) + R(\boldsymbol{w}).$$

2.1 线性模型

- 线性回归VS逻辑斯蒂回归
  - 线性回归得到连续的结果，逻辑斯蒂回归得到离散的结果。
  - 线性回归适用于自变量和因变量呈线性关系。逻辑斯蒂回归不要求。
  - 逻辑斯蒂回归分析因变量取某个值的概率与自变量的关系，而线性回归是直接分析因变量与自变量的关系。
  - 逻辑斯蒂回归的本质是用来解决二分类问题（多分类用Softmax分类器）。

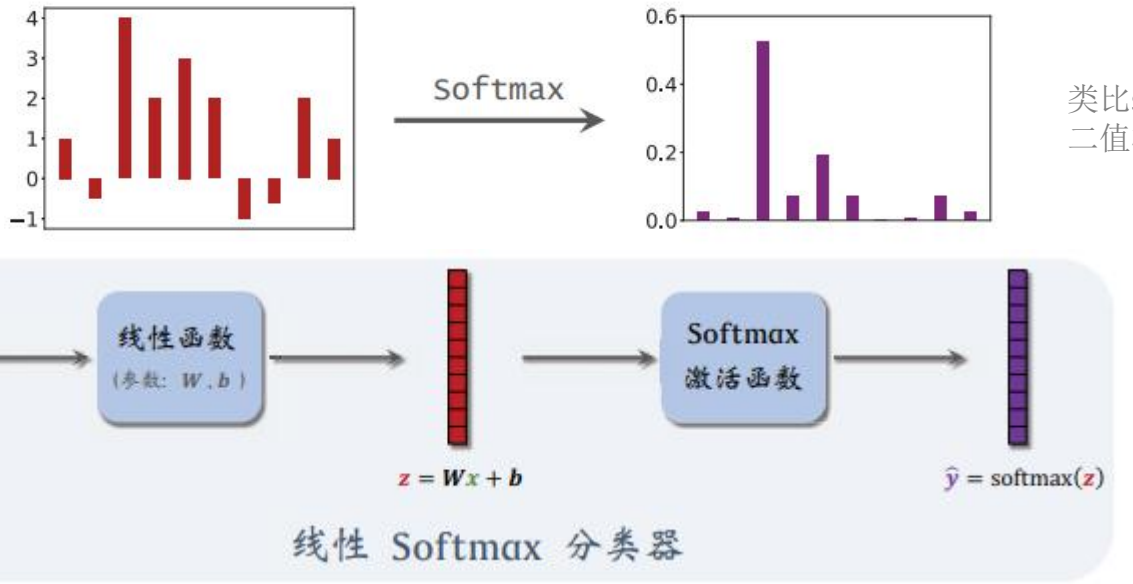
2.1 线性模型

- Softmax分类器

设$\boldsymbol{z} = [z_1, \ldots, z_k]^T$是任意k维向量，其元素可正可负，Softmax的输出：

$$\text{softmax}(\boldsymbol{z}) \triangleq \frac{1}{\sum_{l=1}^{k} \exp(z_l)} \left[\exp(z_1), \exp(z_2), \cdots, \exp(z_k) \right]^T$$

也是个k维向量，其元素非负并且相加等于1。



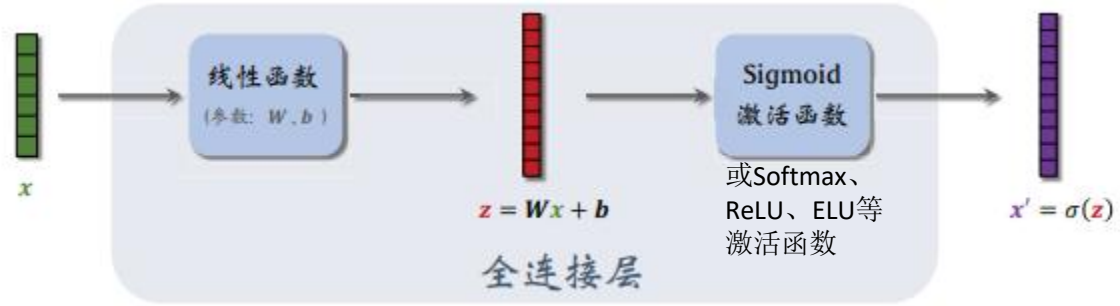
类比sigmoid，softmax将二值概率扩展到多值概率

例：MNIST手写数字识别

2.2 神经网络

- 全连接神经网络（多层感知机）
  - 问题：采用线性Softmax分类器对MNIST数据集识别只有90%的准确率，而人类识别的准确率接近100%。线性分类器性能差的原因在于模型太小，无法充分利用n=60000个训练样本。
  - 解决方法：将“线性函数+激活函数”这样的结构层层堆积，得到一个多层网络，从而提高预测的准确率。

全连接层
包含一个线性函数和一个激活函数



全连接神经网络
（多层感知机）

$$
\text{隐层}\begin{cases}
\text{第 1 层:} & \boldsymbol{x}^{(1)} = \sigma_1\left(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(0)} + \boldsymbol{b}^{(1)}\right), \\
\text{第 2 层:} & \boldsymbol{x}^{(2)} = \sigma_2\left(\boldsymbol{W}^{(2)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(2)}\right), \\
\vdots & \vdots
\end{cases}
$$

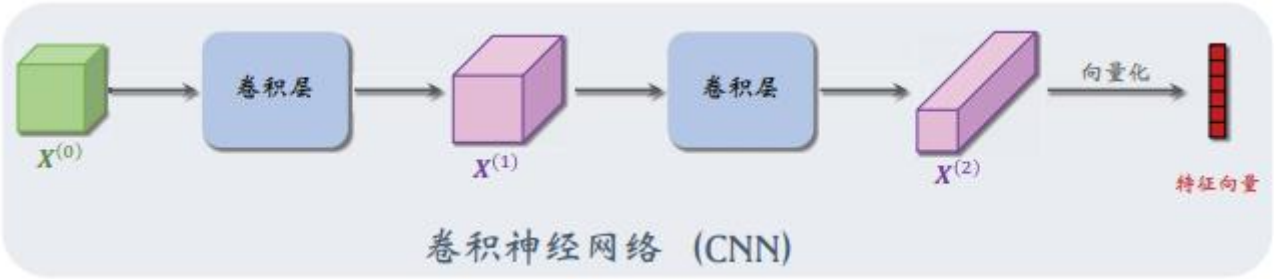
$$
\text{输出层}\quad \text{第 } l \text{ 层:} \quad \boldsymbol{x}^{(l)} = \sigma_l\left(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}\right),
$$

i 隐层输出维度由用户指定，激活函数通常使用ReLU。
ii 输出层的维度由问题本身决定，激活函数的选择取决于具体问题，二分类问题用 Sigmoid，多分类问题用 Softmax，回归问题通常用线性激活函数。

2.2 神经网络

- 卷积神经网络

  - 卷积神经网络区别于全连接网络：
    - ✓ 全连接层换成了卷积层
    - ✓ 卷积层的输入和输出都是矩阵或者三阶张量
    - ✓ 卷积层之后通常有一个**ReLU**激活函数
    - ✓ 最后一个卷积层输出是提取的特征向量



零阶张量为标量（实数），一阶张量为向量，二阶张量为矩阵，以此类推。

2.3 梯度下降与反向传播

- 梯度下降
  - **提出目的：2.1与2.2节中线性模型和神经网络的训练都可以描述成一个优化问题：**

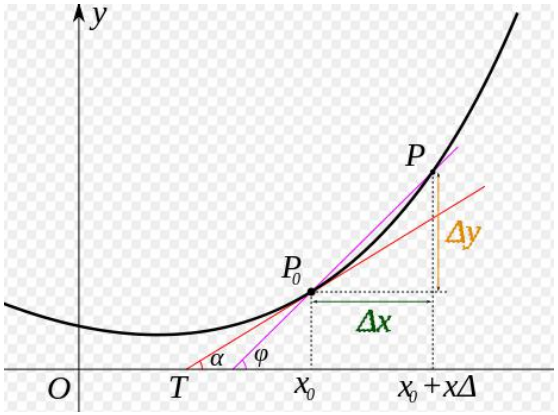
$$\min_{\boldsymbol{w}^{(1)},\cdots,\boldsymbol{w}^{(l)}} L\left(\boldsymbol{w}^{(1)},\cdots,\boldsymbol{w}^{(l)}\right).$$

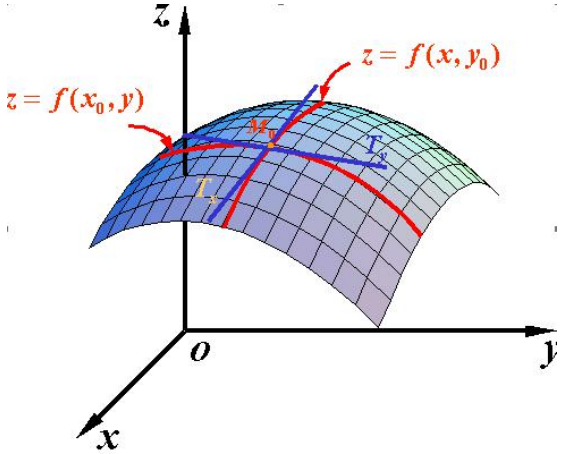
  对于这样一个无约束的最小化问题，通常使用梯度下降（GD）或者随机梯度下降（SGD）解决。

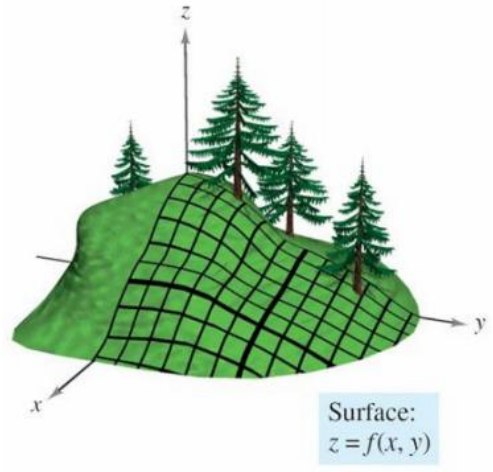
  - **梯度：目标函数L关于一个变量$\boldsymbol{w}^{(i)}$的梯度记作：**

$$\underbrace{\nabla_{\boldsymbol{w}^{(i)}} L\left(\boldsymbol{w}^{(1)},\cdots,\boldsymbol{w}^{(l)}\right) \triangleq \frac{\partial L\left(\boldsymbol{w}^{(1)},\cdots,\boldsymbol{w}^{(l)}\right)}{\partial \boldsymbol{w}^{(i)}}}_{\text{两种符号都表示 } L \text{ 关于 } \boldsymbol{w}^{(l)} \text{ 的梯度}}, \quad \forall\, i=1,\cdots,l.$$

梯度矩阵的大小与变量的大小保持一致。



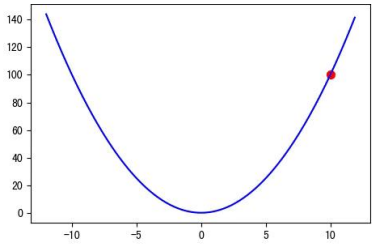


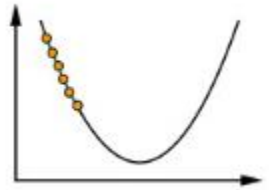


2.3 梯度下降与反向传播

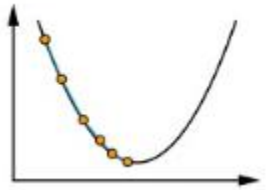
- 梯度下降
  - **梯度下降：**最小化目标函数应该沿着梯度反方向对优化变量$\boldsymbol{w}^{(i)}$做更新。沿着梯度反方向走就叫做梯度下降（GD）。设当前的优化变量为$\boldsymbol{w}_{now}^{(1)}, \ldots, \boldsymbol{w}_{now}^{(l)}$。GD更新优化变量的方式为（$\alpha$为学习率）：

$$\boldsymbol{w}_{\text{new}}^{(i)} \leftarrow \boldsymbol{w}_{\text{now}}^{(i)} - \alpha \cdot \nabla_{\boldsymbol{w}^{(i)}} L\left(\boldsymbol{w}_{\text{now}}^{(1)}, \cdots, \boldsymbol{w}_{\text{now}}^{(l)}\right), \qquad \forall\, i = 1, \cdots, l.$$

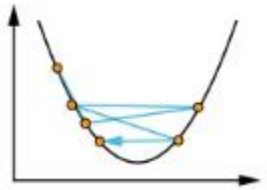




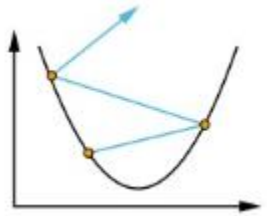
α过小



α适中



α较大



α过大

  - **随机梯度下降：**当目标函数可以写成连加或者期望的形式就可以用随机梯度下降（SGD）求解该问题。函数$\boldsymbol{F}_j$隐含第j个训练样本$(\boldsymbol{x}_j, \boldsymbol{y}_j)$。SGD用单个训练数据即可对模型参数进行一次更新，大大加快了训练速度。

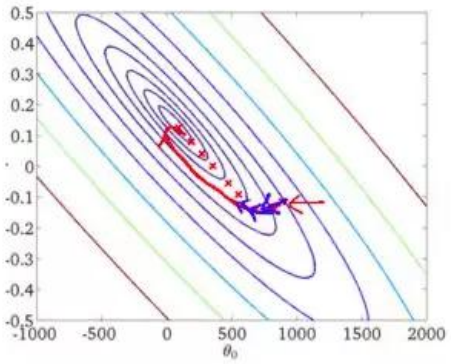
$$L\left(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}\right) = \frac{1}{n}\sum_{j=1}^{n} F_j\left(\boldsymbol{w}^{(1)}, \cdots, \boldsymbol{w}^{(l)}\right).$$

$$\boldsymbol{w}_{\text{new}}^{(i)} \leftarrow \boldsymbol{w}_{\text{now}}^{(i)} - \alpha \cdot \underbrace{\nabla_{\boldsymbol{w}^{(i)}} F_j\left(\boldsymbol{w}_{\text{now}}^{(1)}, \cdots, \boldsymbol{w}_{\text{now}}^{(l)}\right)}_{\text{随机梯度}}, \qquad \forall\, i = 1, \cdots, l.$$

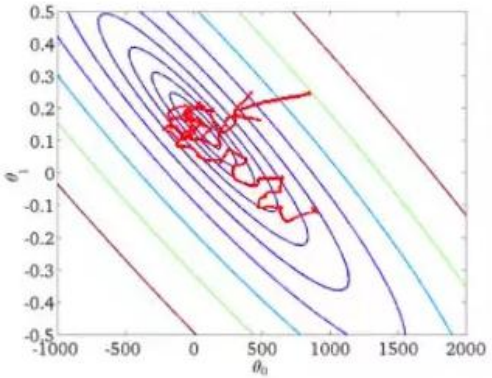
  - **SGD变体：**Adam、RMSProp、Momentum、AdaGrad

2.3 梯度下降与反向传播

- GD VS. SGD
  - **GD：**
    - ✓ 更新每一参数时都使用所有样本进行更新，适用于数据集较小的情况
    - ✓ 迭代次数相对较少
    - ✓ 损失函数为凸函数时可以达到全局最优解，但当损失函数非凸时不能保证结果为全局最优
    - ✓ 当数据集较大时计算较为耗时
  - **SGD：**
    - ✓ 在训练过程中随机优化某一条训练数据上的损失函数
    - ✓ 迭代次数相对较多
    - ✓ 由于某一条数据上损失函数更小并不代表在全部数据上的损失函数更小，因此即使为凸函数时也不一定为全局最优解
    - ✓ 每一轮更新速度大大加快



GD



SGD

2.3 梯度下降与反向传播

- 反向传播
  - 反向传播是一种与最优化方法（如梯度下降法）结合使用，用来训练人工神经网络的常见方法。反向传播依据微积分中的链式法则，沿着从输出层到输入层的顺序，依次计算并存储目标函数有关神经网络各层的中间变量以及参数的梯度。

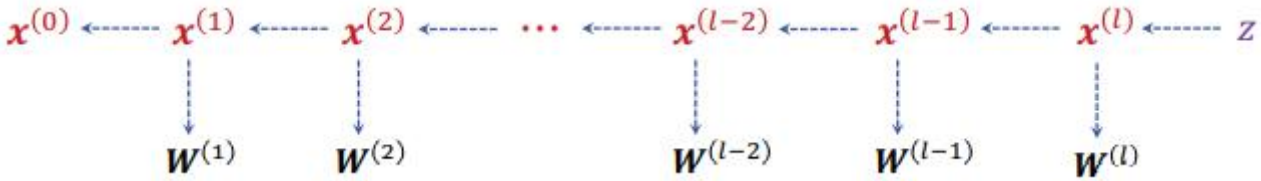
例：全连接网络反向传播

全连接网络回顾：

$$
\begin{aligned}
&\text{第 1 层：} && \boldsymbol{x}^{(1)} = \sigma_1\left(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(0)}\right),\\
&\text{第 2 层：} && \boldsymbol{x}^{(2)} = \sigma_2\left(\boldsymbol{W}^{(2)}\boldsymbol{x}^{(1)}\right),\\
&\quad\vdots && \quad\vdots\\
&\text{第 } l \text{ 层：} && \boldsymbol{x}^{(l)} = \sigma_l\left(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)}\right).
\end{aligned}
$$

损失函数 $z = H\left(\boldsymbol{y}, \boldsymbol{x}^{(l)}\right),$

计算z关于每一参数变量的梯度：$\dfrac{\partial z}{\partial \boldsymbol{W}^{(1)}}, \dfrac{\partial z}{\partial \boldsymbol{W}^{(2)}}, \cdots, \dfrac{\partial z}{\partial \boldsymbol{W}^{(l)}}.$



$for\ i = l, \ldots, 1$ → $\dfrac{\partial z}{\partial \boldsymbol{W}^{(i)}} = \dfrac{\partial \boldsymbol{x}^{(i)}}{\partial \boldsymbol{W}^{(i)}} \cdot \dfrac{\partial z}{\partial \boldsymbol{x}^{(i)}}$ → $\dfrac{\partial z}{\partial \boldsymbol{x}^{(i-1)}} = \dfrac{\partial \boldsymbol{x}^{(i)}}{\partial \boldsymbol{x}^{(i-1)}} \cdot \dfrac{\partial z}{\partial \boldsymbol{x}^{(i)}}$

该梯度用于更新参数$\boldsymbol{w}^{(i)}$　　该梯度被传播到下一层，用于循环计算