

定义

一个函数的梯度就是它的值上升速度最快的方向，我们一般用 $\nabla_w f$ 表示函数 f 对 w 的梯度。从计算的角度来讲，一个函数的梯度就是它对每个变量的导数，例如

$$f(\boldsymbol{x}) = x_1^2 - \log x_2 \text{ 对于 } \boldsymbol{x} \text{ 梯度就是}$$

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{bmatrix} 2x_1 \\ -x_2^{-1} \end{bmatrix}$$

在任意一点，例如 $\boldsymbol{x} = [1, 2]^\top$ 处，梯度方向就是 $[2, -1/2]^\top$

梯度下降

我们已经提到，训练机器学习的模型就是一个通过优化 \boldsymbol{w} 来最小化loss function的过程，那么通过不断的向负梯度方向移动，我们就能不断减少目标函数的值，直到收敛。

举例：接上文的例子，我们现在有数据

$$\begin{array}{ll} \boldsymbol{x}_1 = [0.2, 0.5, 0.7]^\top & \boldsymbol{y}_1 = [0.8, 0.8]^\top \\ \boldsymbol{x}_2 = [0.4, 0.6, 0.8]^\top & \boldsymbol{y}_2 = [0.7, 0.9]^\top \\ \boldsymbol{x}_3 = [0.3, 0.6, 0.9]^\top & \boldsymbol{y}_3 = [0.9, 0.9]^\top \end{array}$$

那么模型的输出就是

$$\begin{array}{l} \hat{\boldsymbol{y}}_1 = W \cdot \boldsymbol{x}_1 = \begin{bmatrix} 0.2w_{11} + 0.5w_{12} + 0.7w_{13} \\ 0.2w_{21} + 0.5w_{22} + 0.7w_{23} \end{bmatrix} \\ \hat{\boldsymbol{y}}_2 = W \cdot \boldsymbol{x}_2 = \begin{bmatrix} 0.4w_{11} + 0.6w_{12} + 0.8w_{13} \\ 0.4w_{21} + 0.6w_{22} + 0.8w_{23} \end{bmatrix} \\ \hat{\boldsymbol{y}}_3 = W \cdot \boldsymbol{x}_3 = \begin{bmatrix} 0.3w_{11} + 0.6w_{12} + 0.9w_{13} \\ 0.3w_{21} + 0.6w_{22} + 0.9w_{23} \end{bmatrix} \end{array}$$

假设我们选择 ℓ_2 的平方为损失函数，那么

$$\begin{aligned} \sum_i \mathcal{L}\{\hat{\boldsymbol{y}}, \boldsymbol{y}\} &= \sum_i \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2 \\ &= \left\| \begin{bmatrix} 0.2w_{11} + 0.5w_{12} + 0.7w_{13} - 0.8 \\ 0.2w_{21} + 0.5w_{22} + 0.7w_{23} - 0.8 \end{bmatrix} \right\|_2^2 \\ &\quad + \left\| \begin{bmatrix} 0.4w_{11} + 0.6w_{12} + 0.8w_{13} - 0.7 \\ 0.4w_{21} + 0.6w_{22} + 0.8w_{23} - 0.9 \end{bmatrix} \right\|_2^2 \\ &\quad + \left\| \begin{bmatrix} 0.3w_{11} + 0.6w_{12} + 0.9w_{13} - 0.9 \\ 0.3w_{21} + 0.6w_{22} + 0.9w_{23} - 0.9 \end{bmatrix} \right\|_2^2 \end{aligned}$$

实际上可以轻易的求出损失函数对于每个 $w_{m,n}$ 的导数，例如

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{11}} &= 0.2 * 2 * (0.2w_{11} + 0.5w_{12} + 0.7w_{13} - 0.8) \\ &\quad + 0.4 * 2 * (0.4w_{11} + 0.6w_{12} + 0.8w_{13} - 0.7) \\ &\quad + 0.3 * 2 * (0.3w_{11} + 0.6w_{12} + 0.9w_{13} - 0.9) \\ &= 0.58w_{11} + 1.04w_{12} + 1.46w_{13} - 1.42 \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{12}} &= 0.5 * 2 * (0.2w_{11} + 0.5w_{12} + 0.7w_{13} - 0.8) \\ &\quad + 0.6 * 2 * (0.4w_{11} + 0.6w_{12} + 0.8w_{13} - 0.7) \\ &\quad + 0.6 * 2 * (0.3w_{11} + 0.6w_{12} + 0.9w_{13} - 0.9) \\ &= 1.04w_{11} + 1.94w_{12} + 2.74w_{13} - 2.72 \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{13}} &= 0.7 * 2 * (0.2w_{11} + 0.5w_{12} + 0.7w_{13} - 0.8) \\ &\quad + 0.8 * 2 * (0.4w_{11} + 0.6w_{12} + 0.8w_{13} - 0.7) \\ &\quad + 0.9 * 2 * (0.3w_{11} + 0.6w_{12} + 0.9w_{13} - 0.9) \\ &= 1.46w_{11} + 2.74w_{12} + 3.88w_{13} - 3.86 \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{21}} &= 0.2 * 2 * (0.2w_{21} + 0.5w_{22} + 0.7w_{23} - 0.8) \\ &\quad + 0.4 * 2 * (0.4w_{21} + 0.6w_{22} + 0.8w_{23} - 0.9) \\ &\quad + 0.3 * 2 * (0.3w_{21} + 0.6w_{22} + 0.9w_{23} - 0.9) \\ &= 0.58w_{21} + 1.04w_{22} + 1.46w_{23} - 1.58 \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{22}} &= 0.5 * 2 * (0.2w_{21} + 0.5w_{22} + 0.7w_{23} - 0.8) \\ &\quad + 0.6 * 2 * (0.4w_{21} + 0.6w_{22} + 0.8w_{23} - 0.9) \\ &\quad + 0.6 * 2 * (0.3w_{21} + 0.6w_{22} + 0.9w_{23} - 0.9) \\ &= 1.04w_{21} + 1.94w_{22} + 2.74w_{23} - 2.96 \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{23}} &= 0.7 * 2 * (0.2w_{21} + 0.5w_{22} + 0.7w_{23} - 0.8) \\ &\quad + 0.8 * 2 * (0.4w_{21} + 0.6w_{22} + 0.8w_{23} - 0.9) \\ &\quad + 0.9 * 2 * (0.3w_{21} + 0.6w_{22} + 0.9w_{23} - 0.9) \\ &= 1.46w_{21} + 2.74w_{22} + 3.88w_{23} - 4.18 \end{aligned}$$

也就是说，对于任意一点 $\boldsymbol{w} = [w_{11}, \dots, w_{23}]^\top$ ，损失函数的梯度方向就是

$$\nabla_w \mathcal{L} = \begin{bmatrix} 0.58w_{11} + 1.04w_{12} + 1.46w_{13} - 1.42 \\ 1.04w_{11} + 1.94w_{12} + 2.74w_{13} - 2.72 \\ 1.46w_{11} + 2.74w_{12} + 3.88w_{13} - 3.86 \\ 0.58w_{21} + 1.04w_{22} + 1.46w_{23} - 1.58 \\ 1.04w_{21} + 1.94w_{22} + 2.74w_{23} - 2.96 \\ 1.46w_{21} + 2.74w_{22} + 3.88w_{23} - 4.18 \end{bmatrix}$$

只要我们不断沿着负梯度方向迭代当前 \boldsymbol{w} ，损失函数的值就会不断下降。因此，我们可以得到一个显然的结论，那就是优化过程会收敛于梯度为0的地方。

由于例子中的问题过于简单，显然可以看出问题收敛于

$$\nabla_w \mathcal{L} = \begin{bmatrix} 0.58w_{11} + 1.04w_{12} + 1.46w_{13} - 1.42 \\ 1.04w_{11} + 1.94w_{12} + 2.74w_{13} - 2.72 \\ 1.46w_{11} + 2.74w_{12} + 3.88w_{13} - 3.86 \\ 0.58w_{21} + 1.04w_{22} + 1.46w_{23} - 1.58 \\ 1.04w_{21} + 1.94w_{22} + 2.74w_{23} - 2.96 \\ 1.46w_{21} + 2.74w_{22} + 3.88w_{23} - 4.18 \end{bmatrix} = \mathbf{0}$$

这个问题有解析解，也就是

$$\begin{bmatrix} 0.58 & 1.04 & 1.46 & 0 & 0 & 0 \\ 1.04 & 1.94 & 2.74 & 0 & 0 & 0 \\ 1.46 & 2.74 & 3.88 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.58 & 1.04 & 1.46 \\ 0 & 0 & 0 & 1.04 & 1.94 & 2.74 \\ 0 & 0 & 0 & 1.46 & 2.74 & 3.88 \end{bmatrix} \cdot \begin{bmatrix} w_{11} \\ w_{12} \\ w_{13} \\ w_{21} \\ w_{22} \\ w_{23} \end{bmatrix} = \begin{bmatrix} 1.42 \\ 2.72 \\ 3.86 \\ 1.58 \\ 2.96 \\ 4.18 \end{bmatrix}$$

可以轻易求出

$$\begin{bmatrix} w_{11} \\ w_{12} \\ w_{13} \\ w_{21} \\ w_{22} \\ w_{23} \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1.5 \\ 0.5 \\ -0.5 \\ 2.5 \\ -0.5 \end{bmatrix}$$

或者变形为

$$W = \begin{bmatrix} -1.5 & 1.5 & 0.5 \\ -0.5 & 2.5 & -0.5 \end{bmatrix}$$

(下面提供解决这个解析解的计算代码)

```
In [1]: import numpy as np
A = np.array([0.58, 1.04, 1.46, 0, 0, 0,
              1.04, 1.94, 2.74, 0, 0, 0,
              1.46, 2.74, 3.88, 0, 0, 0,
              0, 0, 0, 0.58, 1.04, 1.46,
              0, 0, 0, 1.04, 1.94, 2.74,
              0, 0, 0, 1.46, 2.74, 3.88]).reshape(6,6)

y = np.array([1.42, 2.72, 3.86, 1.58, 2.96, 4.18]).reshape(-1,1)

w = np.matmul(np.linalg.inv(A), y)
print(w)
```

```
[[-1.5]
 [ 1.5]
 [ 0.5]
 [-0.5]
 [ 2.5]
 [-0.5]]
```

然而现实往往是残酷的：一般的机器学习（特别是深度学习）的模型不会是 $f_w(\boldsymbol{x}) = W \cdot \boldsymbol{x}$ 这么直观，因此往往没有解析解，这时候就需要用到数值优化。最简单也是最常用的就是梯度下降法，这里最关键的就是3个因素：

- 起始点 \boldsymbol{w} ，一般是随机选取
- 搜索方向，也就是负梯度方向（也有其他基于梯度的改进方案）
- 步长，也叫学习率（往往用 α 或者 `lr` 表示，需要人为设置）
- 停止条件，一般用early stop原则，后面会解释到。

整个优化的过程就是：

- 选取起始点 \boldsymbol{w}
- 计算当前点的梯度 ∇
- 更新参数，也就是把参数向负梯度方向移动 α 的距离，也就是 $w := w - \alpha \cdot \nabla$

下面的代码体现了一个梯度优化的过程，这个例子中我们以迭代20次为停止条件：

```
In [2]: import numpy as np

# 选取随机起始点
w = np.random.randn(6).reshape(-1,1)
# 选取学习率
alpha = 0.25

# 用于计算梯度的矩阵
A = np.array([0.58, 1.04, 1.46, 0, 0, 0,
              1.04, 1.94, 2.74, 0, 0, 0,
              1.46, 2.74, 3.88, 0, 0, 0,
              0, 0, 0, 0.58, 1.04, 1.46,
              0, 0, 0, 1.04, 1.94, 2.74,
              0, 0, 0, 1.46, 2.74, 3.88]).reshape(6,6)

y = np.array([1.42, 2.72, 3.86, 1.58, 2.96, 4.18]).reshape(-1,1)

for i in range(21001):
    # 计算梯度
    nabla = np.matmul(A, w)-y
    # 更新参数
    w = w - alpha * nabla

    # 显示参数变化
    if not i%3000:
        print(f'第{i}次迭代后的参数为: \n{np.round(w.flatten(),2)}')
```

```
第0次迭代后的参数为:
[ 0.98  2.53  0.23 -0.16  0.42  0.2 ]
第3000次迭代后的参数为:
[-1.53  1.59  0.45 -0.35  1.99 -0.2 ]
第6000次迭代后的参数为:
[-1.51  1.53  0.48 -0.46  2.36 -0.41]
第9000次迭代后的参数为:
[-1.5   1.51  0.5  -0.49  2.46 -0.48]
第12000次迭代后的参数为:
[-1.5   1.5   0.5  -0.5   2.49 -0.49]
第15000次迭代后的参数为:
[-1.5   1.5   0.5  -0.5   2.5  -0.5]
第18000次迭代后的参数为:
[-1.5   1.5   0.5  -0.5   2.5  -0.5]
第21000次迭代后的参数为:
[-1.5   1.5   0.5  -0.5   2.5  -0.5]
```

可以看出，通过数值方法，我们也能得到最优解。

补充

梯度下降是深度学习中最具主导地位的优化方法。我们可以看出，这个梯度下降法的过程中最复杂的部分就是推导梯度，然而这个过程可以被许多工具自动化进行（事实上，除了梯度计算之外的很多更多其他功能也已经被实现），比如PyTorch, TensorFlow等等。