程式人

十分鐘系列

梯度下降法

（隱藏在深度學習背後的演算法）

陳鍾誠

2020 年 9 月 30 日

最近幾年

- 深度學習技術很熱門

人工智慧

- 在沉寂了二十幾年後

又再度引起人們的關注

所謂的深度學習

- 其實就是《層數較多的神經網路》

背後的演算法

- 仍然和三十年前一樣，同樣是
  - 梯度下降法
  - 反傳遞演算法

這兩種算法中

- 梯度下降法很簡單
- 而反傳遞演算法，則是
  《快速的梯度下降法》
- 只是計算梯度時，採用鏈鎖規則進行反傳遞而已！

梯度下降法的 python 程式如下

```python
# 使用梯度下降法尋找函數最低點
def gradientDescendent(f, p0, step=0.01):
    p = p0.copy()
    i = 0
    while (True):
        i += 1
        fp = f(p)
        gp = grad(f, p) # 計算梯度 gp
        glen = norm(gp) # norm = 梯度的長度 (步伐大小)
        print('{:d}:p={:s} f(p)={:.3f} gp={:s} glen={:.5f}'.format(i, str(p), fp, str(gp), glen))
        if glen < 0.00001:  # 如果步伐已經很小了，那麼就停止吧！
            break
        gstep = np.multiply(gp, -1*step) # gstep = 逆梯度方向的一小步
        p +=  gstep # 向 gstep 方向走一小步
    return p # 傳回最低點！
```

其中梯度的數學與程式為

《梯度》的數學定義如下：

$$\nabla_x f(x) = \left[\frac{\partial}{\partial x_1} f(x), \frac{\partial}{\partial x_2} f(x), \cdots, \frac{\partial}{\partial x_n} f(x)\right]^T = \frac{\partial}{\partial x} f(x)$$

```
# 函數 f 在點 p 上的梯度
def grad(f, p):
    gp = p.copy()
    for k in range(len(p)):
        gp[k] = df(f, p, k)
    return gp
```

梯度其實就是

- 對每個變數都偏微分所形成的向量
- 其中的偏微分之數學與程式如下

$$\frac{\partial}{\partial x_1} f(x) = \lim_{h \to 0} \frac{f(x_1, \ldots, x_i + h, \ldots, x_n) - f(x_1, \ldots, x_i, \ldots, x_n)}{h}$$

```
# 函數 f 對變數 p[k] 的偏微分: df / dp[k]
def df(f, p, k):
    p1 = p.copy()
    p1[k] = p[k]+step
    return (f(p1) - f(p)) / step
```

完整的梯度下降法程式請看

github.com/ccccourse/ai/blob/master/python/03-neuralnet/03-gd/gd1.py

31 lines (28 sloc) | 986 Bytes

```python
import numpy as np
from numpy.linalg import norm

# 函數 f 對變數 k 的偏微分: df / dk
def df(f, p, k, step=0.01):
    p1 = p.copy()
    p1[k] = p[k]+step
    return (f(p1) - f(p)) / step
```

https://github.com/ccccourse/ai/blob/master/python/03-neuralnet/03-gd/gd1.py

以下是簡單的測試程式

- 用來尋找 $f(x, y)=x^2+y^2$ 的最低點

```
import gd1 as gd

def f(p):
    [x,y] = p
    return x*x + y*y

p = [1.0, 3.0]
gd.gradientDescendent(f, p)
```

https://github.com/ccccourse/ai/blob/master/python/03-neuralnet/03-gd/gdTest.py

該程式的執行結果如下

- 確實找到了 (x, y)=(0, 0) 這個最低點

```
PS D:\ccc\course\ai\python\03-neuralnet> python .\gradientDescendentTest.py
1:p=[1.0, 3.0] f(p)=10.000 glen=6.337
2:p=[0.99799 2.99399] f(p)=9.960 glen=6.325
3:p=[0.99598402 2.98799202] f(p)=9.920 glen=6.312
4:p=[0.99398205 2.98200604] f(p)=9.880 glen=6.299
5:p=[0.99198409 2.97603202] f(p)=9.841 glen=6.287
6:p=[0.98999012 2.97006996] f(p)=9.801 glen=6.274
7:p=[0.98800014 2.96411982] f(p)=9.762 glen=6.262
8:p=[0.98601414 2.95818158] f(p)=9.723 glen=6.249
9:p=[0.98403211 2.95225522] f(p)=9.684 glen=6.237
..中間省略 ...
6671:p=[-0.0049984  -0.00499523] f(p)=0.000 glen=0.000
6672:p=[-0.00499841 -0.00499524] f(p)=0.000 glen=0.000
6673:p=[-0.00499841 -0.00499525] f(p)=0.000 glen=0.000
6674:p=[-0.00499841 -0.00499526] f(p)=0.000 glen=0.000
```

更詳細的解說請參考

misavo.com/blog/陳鍾誠/書籍/人工智慧/03-神經網路/B-梯度下降法

陳鍾誠 / 書籍 / 人工智慧 / 03-神經網路 / B-梯度下降法

梯度下降法

深度學習 (Deep Learning) 是人工智慧領域當紅的技術，說穿了其實就是原本的《神經網路》(Neural Network)，不過由於加上了一些新的模型 (像是捲積神經網路 CNN, 循環神經網路 RNN 與生成對抗網路 GAN)，還有在神經網路的層數上加深很多，從以往的 3-4 層，提升到了十幾層，甚至上百層，於是我們給這些新一代的《神經網路》技術一個統稱，那就是《深度學習》。

雖然《深度學習》的神經網路層數變多了，《網路模型》也多了一些，但是背後的學習算法和運作原理並沒有多大改變，仍然是以《梯度下降》(Gradient Descendent) 和《反傳遞算法》(Back Propagation) 為主。

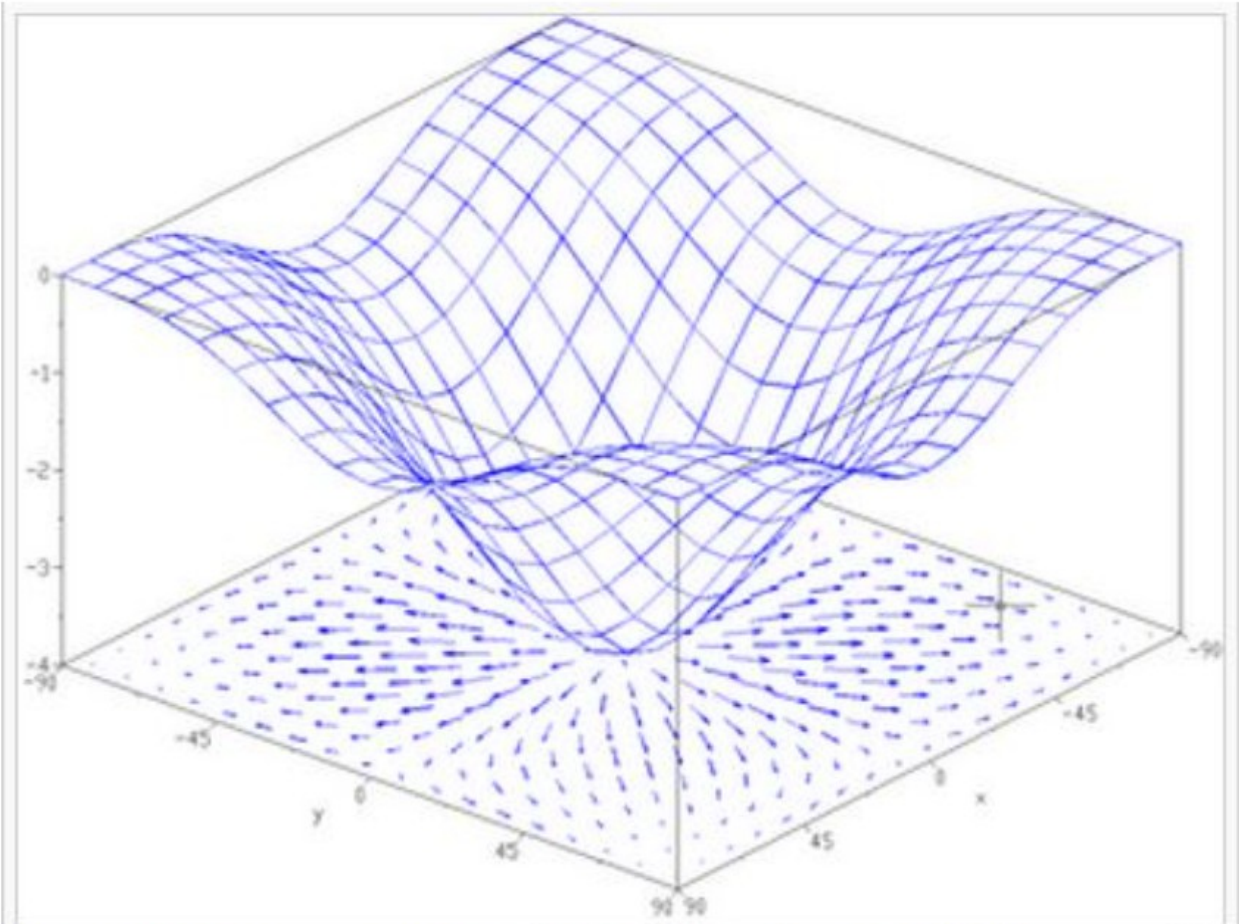
但是《梯度下降》和《反傳遞算法》兩者，幾乎都是以數學的形式呈現，其中《梯度》的數學定義如下：

$$\nabla_x f(x) = \left[\frac{\partial}{\partial x_1} f(x), \frac{\partial}{\partial x_2} f(x), \cdots, \frac{\partial}{\partial x_n} f(x)\right]^T = \frac{\partial}{\partial x} f(x)$$

若把《梯度》當成一個《巨型算子》可以寫為如下形式：

$$\nabla_x = \left[\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \cdots, \frac{\partial}{\partial x_n}\right]^T = \frac{\partial}{\partial x}$$

梯度就是斜率最大的方向

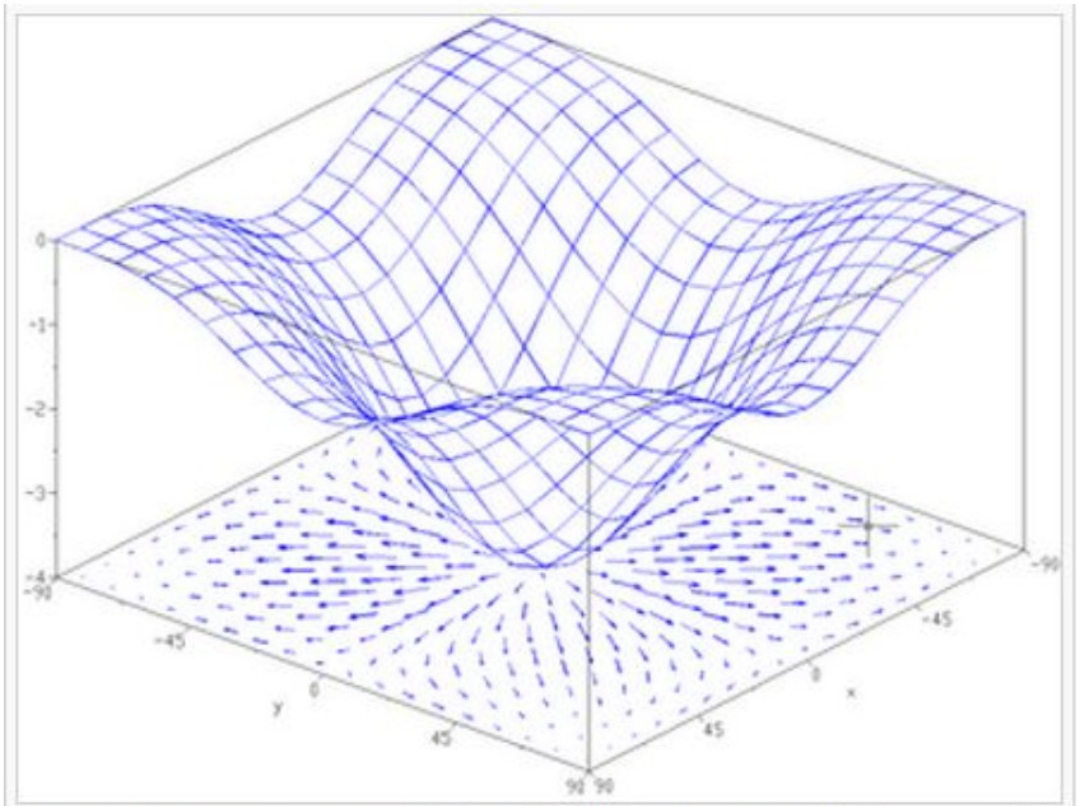


The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

其實梯度就是斜率最大的那個方向，所以梯度下降法，其實就是朝著斜率最大的方向走。

朝著正梯度走，會一直上升

- 而朝著逆梯度走，就會沿著最陡的方向下降。



The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

其實梯度就是斜率最大的那個方向，所以梯度下降法，其實就是朝著斜率最大的方向走。

梯度下降法的程式

- 20 行就能搞定！

```
# 使用梯度下降法尋找函數最低點
def gradientDescendent(f, p0, step=0.01):
    p = p0.copy()
    i = 0
    while (True):
        i += 1
        fp = f(p)
        gp = grad(f, p) # 計算梯度 gp
        glen = norm(gp) # norm = 梯度的長度 (步伐大小)
        print('{:d}:p={:s} f(p)={:.3f} gp={:s} glen={:.5f}'.format(i, str(p), fp, str(gp), glen))
        if glen < 0.00001:  # 如果步伐已經很小了，那麼就停止吧！
            break
        gstep = np.multiply(gp, -1*step) # gstep = 逆梯度方向的一小步
        p +=  gstep # 向 gstep 方向走一小步
    return p # 傳回最低點！
```

所有的深度學習技術

- 都是從這個算法開始的！

只是這種簡單的梯度下降法有個缺點

- 就是當變數多的時候，速度會很慢！
- 所以才需要用《反傳遞演算法》加快

反傳遞演算法

- 只是利用《反傳遞》的方式
- 透過鏈鎖規則，一層一層回算《梯度》

反傳遞背後的數學法則

- 就是微積分的鏈鎖規則

$$\frac{\partial f(q,z)}{\partial x} = \frac{\partial q(x,y)}{\partial x} \frac{\partial f(q,z)}{\partial q}$$

當然

- 現在的深度學習技術
- 使用的神經網路又深又複雜

但是

- 背後的演算法
- 仍然是反傳遞演算法
- 也就是《快速版的梯度下降法》

也就是上面那個

- 短短的二十幾行程式碼

github.com/ccccourse/ai/blob/master/python/03-neuralnet/03-gd/gd1.py

```
18  def gradientDescendent(f, p0, step=0.01):
19      p = p0.copy()
20      i = 0
21      while (True):
22          i += 1
23          fp = f(p)
24          gp = grad(f, p) # 計算梯度 gp
25          glen = norm(gp) # norm = 梯度的長度 (步伐大小)
26          print('{:d}:p={:s} f(p)={:.3f} gp={:s} glen={:.5f}'.format(i, str(p), fp, str(gp), glen))
27          if glen < 0.00001:  # 如果步伐已經很小了，那麼就停止吧！
28              break
29          gstep = np.multiply(gp, -1*step) # gstep = 逆梯度方向的一小步
30          p +=  gstep # 向 gstep 方向走一小步
31      return p # 傳回最低點！
```

對程式人來講

- 梯度下降法，其實非常簡單！
- 完全只是建構在《偏微分》運算上的一個小小算法罷了。

但是

- 這樣的算法
- 正在改變整個世界！

這就是

- 程式的力量之所在！

希望您會喜歡

我們今天的

- 十分鐘系列！

我們下次見

Bye bye !