

# 人工神經網絡背後的數學原理！

原創 賈博文 Datawhale 昨天

↑ ↑ ↑ 關注後"星標"Datawhale  
每日干貨& 每月組隊學習 · 不錯過

Datawhale乾貨

作者：賈博文，浙江大學，Datawhale原創作者

本文約8000字 · 建議閱讀22分鐘

審稿人：阿澤 · Datawhale成員 · 復旦大學計算機碩士 · 目前在攜程擔任高級算法工程師。

## 前言

提到人工智能算法，人工神經網絡（ANN）是一個繞不過去的話題。但是對於新手，往往容易被ANN中一堆複雜的概念公式搞得頭大，最後只能做到感性的認識，而無法深入的理解。正好最近筆者本人也在經歷這個痛苦的過程，本著真理越辯越明的態度，索性坐下來認真的把這些頭大的問題梳理一番，試試看能不能搞清楚ANN背後的數學原理。

其實ANN 的流程概括來說倒不是很複雜，以最簡單的前饋神經網絡為例，無非就是

1. 搭建網絡架構（含有待定參數）
2. 通過比較輸出與標籤的差值定義損失函數（自變量為待定參數的函數）
3. 隨機給出一組初始參數

4. 給出一個 (sgd) / 一批 (Minibatch) 訓練樣例 (包括ANN的輸入值和輸出值)
5. 前向傳播得到預測標籤和損失
6. 利用梯度下降算法從後往前調整網絡參數 (誤差反向傳播, BP)
7. 得到所有參數值
8. 得到ANN並使用

其中4、5、6裡的内容是需要反复迭代的 (注意6是雙重循環)。

在ANN的一堆操作裡，梯度下降算法是一個相對獨立的過程，不妨就讓我們從梯度下降算法開始吧。

## 一、梯度下降到底在幹什麼

其實這個問題非常簡單，只是大家被梯度下降複雜的過程搞蒙了，忘記了它的本質。**梯度下降算法自始至終都在乾一件事——就是找到函數的極值點**，當然確切的說是極小值點。但是，這種方法不同於以往我們在高等數學裡學到的找極值點的方法。那麼我們首先就要問，求極值的經典方法不香嗎？

### 1.1 求極值：傳統的方法不香嗎？

要回答這個問題，讓我們先快速回顧一下在中學和大學裡學到的傳統的求極值點的方法。

對於一元函數來說，極值可能出現在一階導函數為0的點 (駐點) 或是導數不存在的點。

例如要找到  $f(x) = x^2 + 3x$  的極值點

求導得  $dv/dx = 2x + 3$

令  $dy/dx = 0$

就得到  $x = -1.5$  时，导函数为0。

注意，上面找到的只是可能的極值點，也就是極值存在的必要條件。還需要驗證一下充分條件，才能確定極值。這時，可以判斷二階導的正負性、或是判定一階導在可能的極值點兩邊的正負情況。回到我們的例子

当  $x < -1.5$  时， $2x + 3 < 0$

当  $x > -1.5$  时， $2x + 3 > 0$

说明函数在  $x = -1.5$  附近先下降、后上升

该点是一个极小值点

**對於二元函數，情況更複雜一些** $f(x, y)$ 。首先要找出該函數的駐點和偏導數不存在的點，這些點仍然只是可能的極值點。而二元函數的駐點需要同時滿足兩個偏導數為0的條件，即

$$\begin{cases} f_x(x, y) = 0 \\ f_y(x, y) = 0 \end{cases}$$

顯然，**這裡的駐點是需要解這樣一個二元方程才能求得的**。對於駐點分別求出其3個二階偏導數的值，再根據一些規則才能判斷是不是極值點。

還需注意這個判斷規則是不同於一元函數的，因為一元函數極值的充分條件只需要考察一個二階偏導數，而這裡則需要綜合考察二元函數的3個二階偏導數，計算量明顯增大了。對於偏導數不存在的情形還需要特判。

綜合以上，我們可以看出使用經典方法雖然能準確的解出極值點，但當函數自變量的個數很多時，用這種方法求解極值點還真的不香。比如：

- 該方法方法不具有普適性。所謂普適性，就是不能簡單的向多元推廣。從一元到二元的例子可以看出，函數的自變量個數每增加一元，就要研究新的求解方案。可以想像如果是三元函數，其二階偏導數的個數更多，則判斷極值的充分條件還要來的更加複雜。而ANN中可能會求解上億元函數的極值點。
- 其次，這種方法需要解多元方程組，而且這些方程還不一定是線性的。對於這種多元的非線性方程組，我們的直觀感受就是很難解出。事實上，雖然存在一些可供編程的數值計算解法，但計算量大，且求出的是近似解，具有一定的局限性。

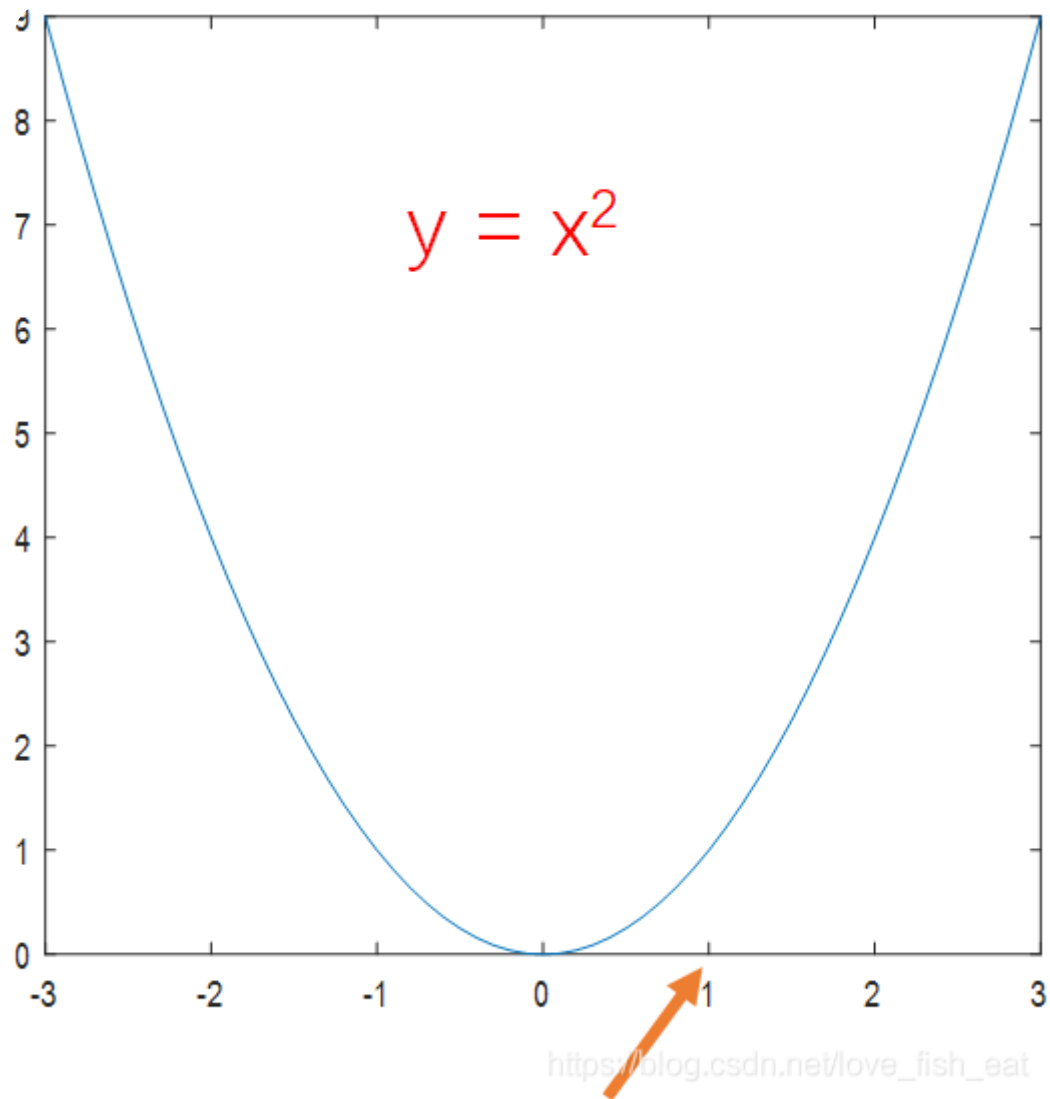
基於此，為了找出多元函數的極值點，我們還需另尋他法。這種方法要簡單易行，特別是要能簡單的向任意元函數推廣，而且這種方法要能夠適應計算機數值計算的特點，畢竟我們這套程序肯定是要放在電腦上跑的。而這就是**傳說中的梯度下降算法**。

## 1.2 什麼是梯度？

梯度的概念其實也不難，但為了讓盡可能多的人明白這一概念，我們還是從一元函數開始吧。不過現在我們的目標是——用純粹數值計算的方法，**從函數上的某一點出發**，找到函數的極值。這裡我們只考察極小值。

### 1.2.1 一元函數找極值：從枚舉試探法到梯度下降法

以函數為例，讓我們看看如何找到極值。 $y = x^2$



既然是從函數上的某一點出發，那麼不妨設想我們在  $x = 1$  的地方，這個地方是不是極小值點呢，我們可以試探一下。

向右走  $0.5$ ，发现  $f(1.5) > f(1)$ ，说明这个方向是上升的方向，不应该选择这个方向；

向左走  $0.5$ ，发现  $f(0.5) < f(1)$ ，说明这个方向是下降的方向，选择这个方向；

再向左走 $0.5$ ，发现 $f(0) < f(0.5)$ ，说明这个方向是下降的方向，选择这个方向；

再向左走 $0.5$ ，发现 $f(-0.5) > f(0)$ ，说明这个方向是上升的方向，不应该选择这个方向。

至此，我们可以将 $x = 0$ 作为极小值点。

回顧這個過程，我們將尋找極小值點的過程抽象如下：

- 首先，選擇一個**方向**
- 試著沿該方向**走一小步**，並據此判斷該方向是否合理。如果合理，則走這一步；如果不合理，換一個方向
- 反復重复第二步，直到找到極小值點

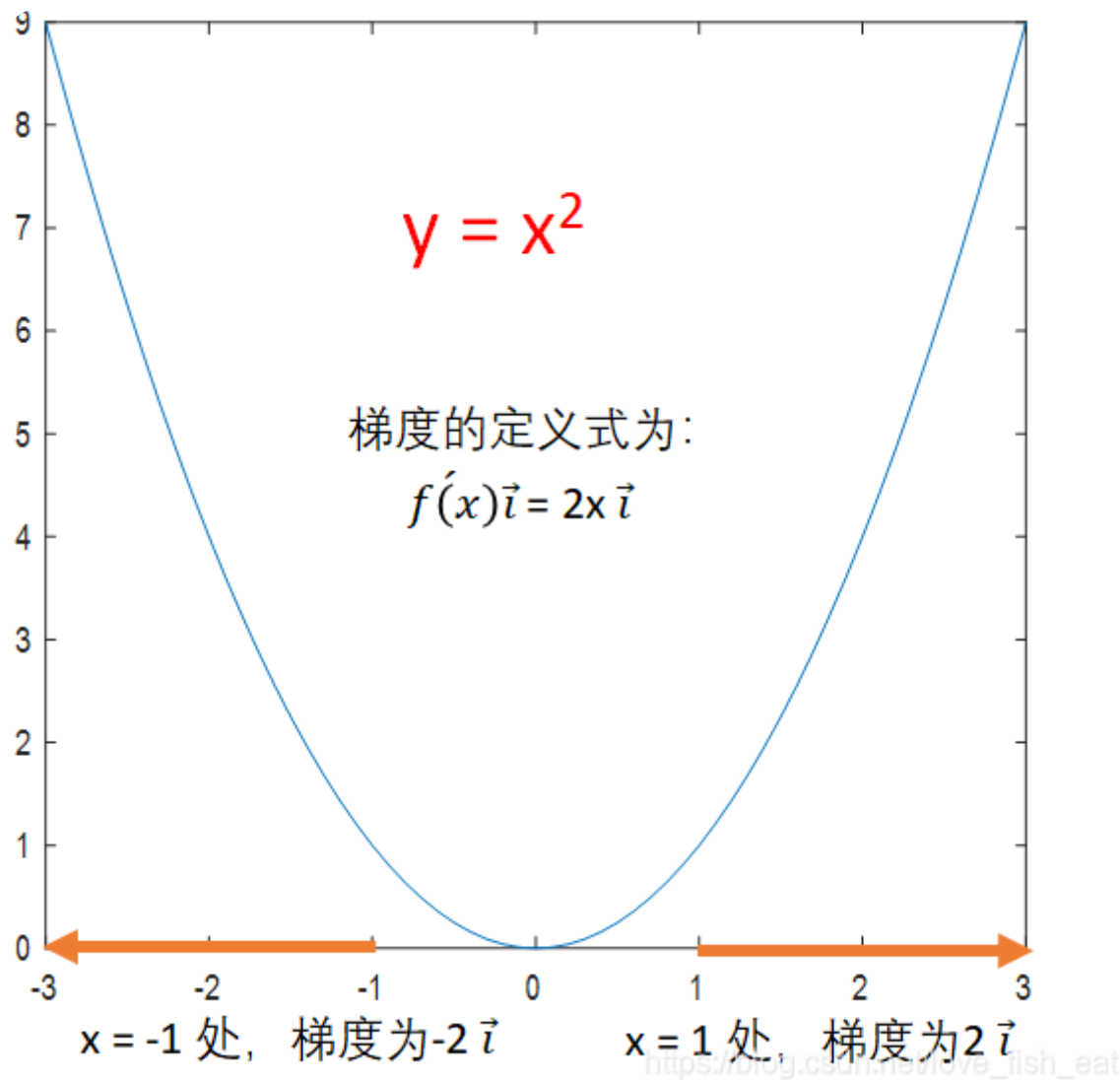
當然這裡還有幾點值得注意

- 第一，對於一元函數來說我們只有向左走或向右走兩個選項。換句話說，每一步我們的選擇是有限的，是可以枚舉的。因此，這個方法我把它稱之為**枚舉試探法**。
- 第二，判斷方向其實不必這樣試錯，直接求導就好。如果某點的導數值 $> 0$ ，說明在該點處函數是遞增的，為了找到極小值，應該向左走；而如果導數值 $< 0$ ，則反之向右走即可。
- 第三，這種方法是不一定能找到極小值的，能不能找到極值點受選擇的起始點以及每次前進的步長這兩個因素影響。

對於**第二點**，我們可以引出梯度的定義了。

梯度是一個向量，它總指向當前函數值增長最快的方向。而它的模長則是這個最快的增長率（導數）的值。想要得到梯度向量，也很簡單，它在 $x, y, z, \dots$ 等方向上的分量（坐標）就是相應的導數值。於是我們求導就可以了。

對於一元函數，函數變化的方向只有兩個，我們定義一種一維的向量來表示梯度，比如  $5\vec{i}$ ， $-5\vec{i}$ 。 $\vec{i}$  前的數為正時，代表向量指向x軸正向； $\vec{i}$  前的數為負時，代表向量指向x軸負向。由下圖可以看出，按照上述定義規定的梯度向量自然的指向了函數增長的方向，是不是很神奇。



由於梯度的方向正是函數增長最快的方向，所以**梯度的逆方向就成了函數下降最快的方向**。當然對於一元函數來說，沒有最快的方向的概念，因為畢竟就兩個方向而已，根本沒得比。不過有了梯度，我們就可以進一步簡化上述尋找極小值點的過程：

- 首先，求出某點的梯度
- 沿梯度的反方向移動一小步
- 反復進行第一、二步，直到找到極小值點

仍以函數，起始點 $x = 1$ 為例，讓我們看看如何用梯度找到極值。 $y = x^2$

初始 $x = 1$ ，步長 $step = 0.5$

#在我們的例子里，梯度的計算式為 $2xi$ 。 $i$ 是指向 $x$ 軸正向的單位向量

求 $x = 1$ 處的梯度為 $2i$ ，梯度反方向為 $-i$  #注意這裡我們只關注梯度的方向，至於梯度的模長則不必在意

沿此方向走一步， $x_{新} = x_{旧} + step * 负梯度方向上单位向量的坐标 = 1 + 0.5 * (-1) = 0.5$

求 $x = 0.5$ 處的梯度為 $i$ ，梯度反方向為 $-i$

沿此方向再走一步， $x_{新} = 0.5 - 0.5 * 1 = 0$

求 $x = 0$ 處的梯度為 $0$ ，說明到達極值點

以上就是用梯度找極小值點的過程，也就是梯度下降算法所做的事情，其實不難理解對吧。

可以看出，相比於枚舉試探法，**梯度下降法**明顯智能了許多，它直接給出了正確的方向，不需要我們一步步試探了。此外，使用梯度下降法不必再關注具體的函數值，只需要把注意力放在導數上，而且只關注一階導數即可。

在後面，我們還將給上面提到的步長 $step$ 換一個高大上的名字——**學習率**，這樣就完全是機器學習裡的叫法了。

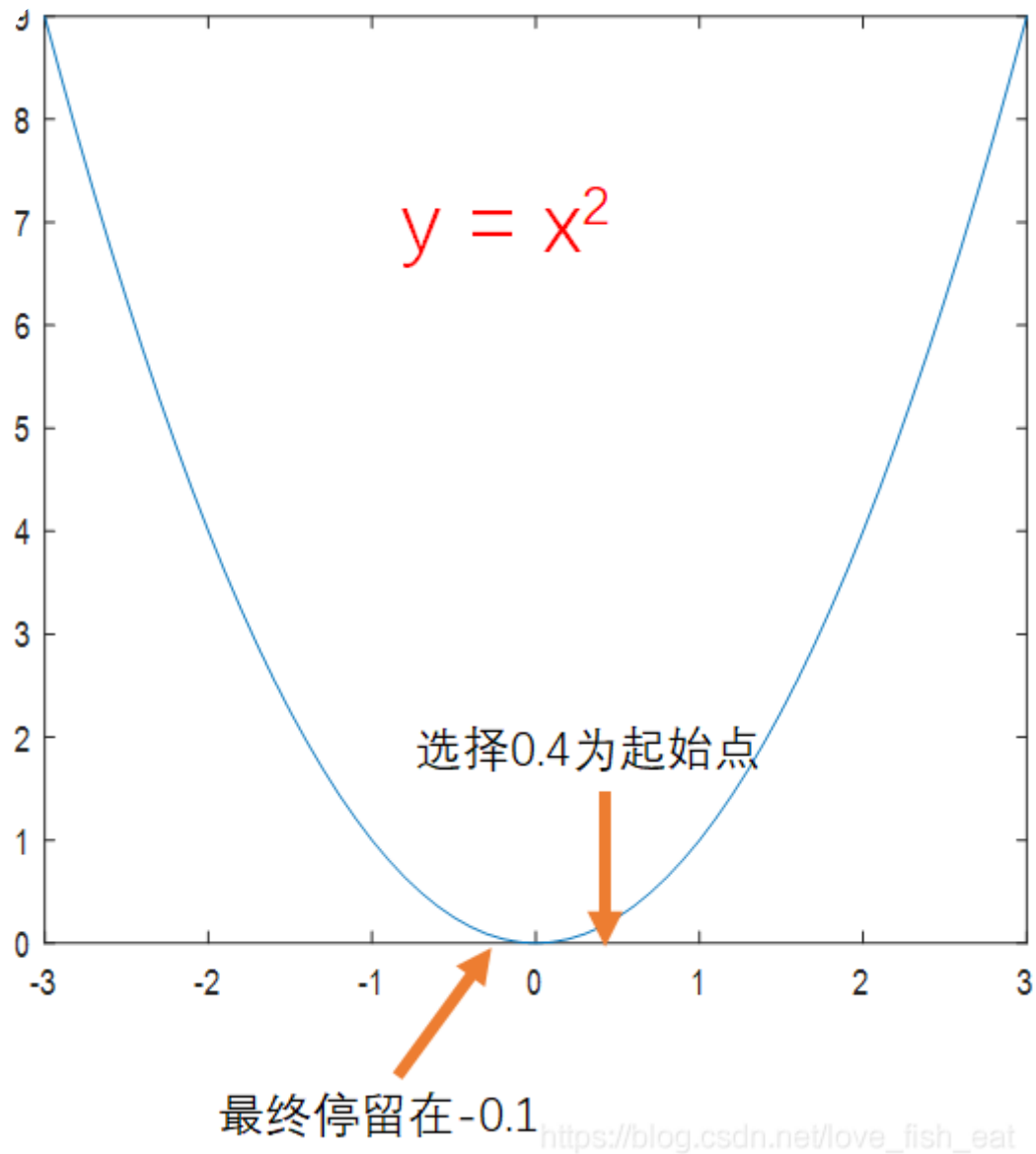


這裡用到梯度的時候，我進行了單位化操作，其實也可以不進行這一步，這樣當函數變化比較劇烈的時候，移動的距離就比較多；函數變化比較平緩的時候，移動的距離就比較短。比如，在我們這個例子裡，只需一輪迭代就能得到結果了。

```
初始x = 1, 学习率step = 0.5  
#在我们的例子里，梯度的计算式为2xi。i是指向x轴正向的单位向量  
求x = 1处的梯度为2i，梯度反方向为-2i #注意这里我们既关注梯度的方向，也关注梯度的模长  
沿此方向走一步，x新 = x旧 + step * 负梯度的坐标 = 1 + 0.5 * (-2) = 0  
求x = 0处的梯度为0，说明到达极值点
```

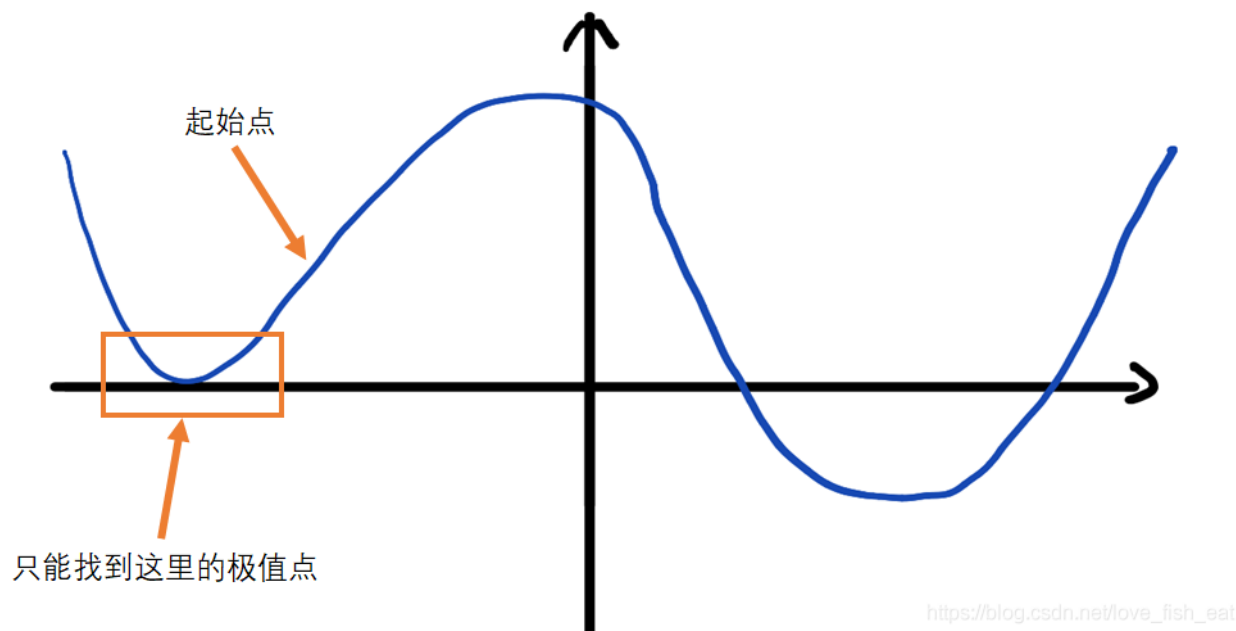
好了說完了梯度，對於前面**第三點**提到的找不到極值的情形，我們舉兩個具體的例子

還是函數，如果起始點選為0.4，而學習率仍為0.5，在採用單位化梯度向量的情形下，則無法找到事實上的極小值點 $y = x^2$



對於這種情況，我們可以通過**減小學習率**使結果盡可能精確，例如我們將學習率設置為0.1，就仍然能得到精確的結果。事實上，在實際操作中，一般也會把學習率設置為0.1。

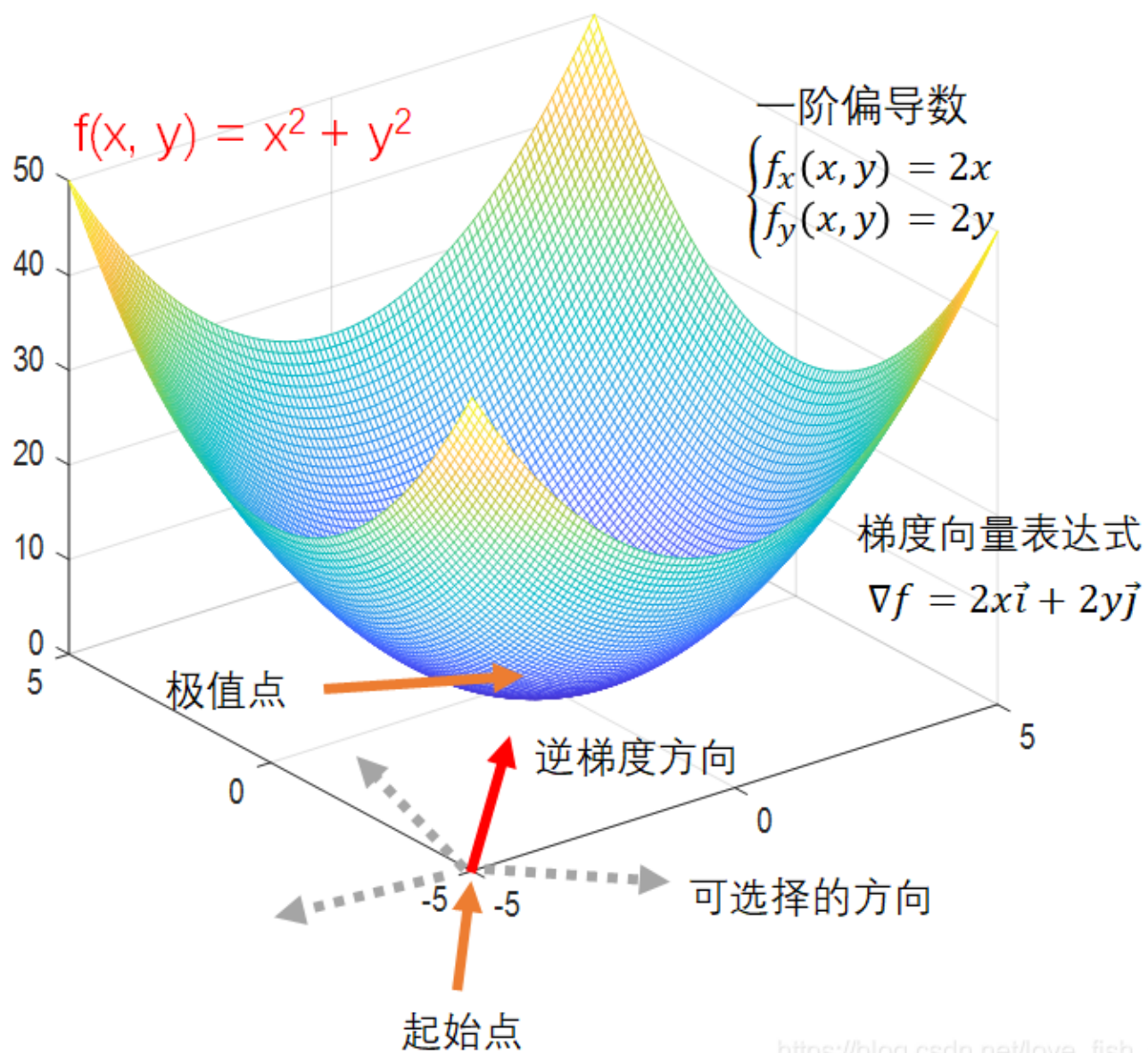
而對於這種有多個極值點的函數，這種方法是沒法找到全部極值點的，更遑論找到全局的極值點了。這時，我們可以在算法裡**加入一些隨機性**，使其有一定概率跳出可能陷入的局部極值點。



### 1.2.2 多元函數的梯度

前面說過梯度下降算法的好處之一在於可以很方便的向多維推廣，現在我們以二元函數為例，看看梯度是如何幫助我們找到極值點的。

這次我們的函數變成了，起始點選擇為  $(-5, -5)$ ，學習率仍設置為0.5。現在我們的目標是從這個點出發，找到該函數的極值點，我們知道這個極值點應該是  $(0, 0)$ 。 $f(x, y) = x^2 + y^2$



[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)

這裡與一元函數有幾點不同：

- 首先，二元函數描述的是一個自變量和兩個因變量之間的關係，也就是說函數的定義域是一個二維平面，我們要找的**極值點就在這個二維平面上**。
- 其次，由於是在二維平面上尋找極值點，我們每一步可以選擇的方向不再局限於一維時的向左或向右，而是瞬間變成了無窮多個方向。因此，枚舉試探法徹底宣告失效。還好我們有更智能的梯度下降法。
- 一元梯度定義式裡的導數現在已經換成了多元函數的偏導數。

好了，現在算法開始：

```

起始点坐标(-5,-5) · 学习率step = 0.5
#在我们的例子里，梯度的计算式为 $2xi + 2yj$  ·  $i$ 和 $j$ 分别是指向 $x$ 轴正向和 $y$ 轴正向的单位向量
求点(-5,-5)处的梯度为 $-10i-10j$  · 负梯度为 $10i+10j$  · 写成坐标形式就是 ( 10,10 )
在点(-5,-5)处沿此梯度走一步

根据公式 向量坐标 = 终点坐标 - 起点坐标 · 得终点坐标 = 起点坐标 + 向量坐标

这里 · 终点坐标是 ( x新, y新 ) · 起点坐标是 ( x旧, y旧 ) = ( -5, -5 )
向量坐标是负梯度坐标 = ( 10,10 ) · 再考虑学习率step · 就可以得到
( x新, y新 ) = ( -5, -5 ) + 0.5 * ( 10, 10 ) = ( 0, 0 )

求 ( 0, 0 ) 处的梯度为零向量 · 说明到达极值点

```

將上述過程抽象，我們就得到了**梯度下降算法的全部邏輯**：

我們要找函數的極小值點（使函數取值盡可能小的那一組自變量），因為，梯度的方向是函數值增長速度最快的方向，所以，沿著梯度的反方向函數值下降最快。

因此，只要沿著梯度的反方向一步步逼近就有可能找到那一組使函數取值盡可能小的自變量。

如何沿著梯度的反方向一步步逼近呢？

我們隨機指定一個起點坐標（一組自變量取值），然後沿著梯度的方向求出未知的終點坐標，梯度是一個向量，本身也具有坐標

由于 向量坐标 = 终点坐标 - 起点坐标

所以 终点坐标 = 起点坐标 + 向量坐标

提炼的公式 终点坐标 = 起点坐标 + 学习率 × 起点坐标处的负梯度向量的坐标

我们的例子

$$(0, 0) = (-5, -5) + 0.5 \times [ -(-10, -10) ]$$

↑            ↑                            ↑

终点坐标   起点坐标   (-5, -5)处的负梯度向量的坐标

二元函数情形

$$(x_{new}, y_{new}) = (x_{old}, y_{old}) + step \times [ - \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) ]_{(x_{old}, y_{old})}$$

↑            ↑                            ↑

终点坐标   起点坐标    $(x_{old}, y_{old})$ 处的负梯度向量的坐标

多元函数的一个参量

$$\omega_i^{new} = \omega_i^{old} - \eta \times \left( \frac{\partial f}{\partial \omega_i} \right)_{\omega^{old}}$$

↑            ↑                            ↑

终点坐标的  
第i个分量   起点坐标的  
第i个分量   坐标 $(\omega_1^{old}, \omega_2^{old}, \dots, \omega_n^{old})$   
处梯度的第i个分量

[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)

通过上面的迭代公式，无论是多少元的函数，它的一个个自变量们都会比较快的接近极值点（或者其近似）。这样我们就可以找到一组自变量值，使得函数值尽可能的小。

### 1.2.3 小結

- 梯度的計算公式為

$$\nabla f = \frac{\partial f}{\partial x} \vec{i} + \frac{\partial f}{\partial y} \vec{j} + \dots$$

- 梯度是一個向量，它總指向當前函數值增長最快的方向，而它的模長則是這個最快的增長率（導數）的值。
- 梯度下降法是一種通過數值計算求解函數極值點的方法
- 其過程概括來說就是順著梯度的反方向一步步逼近可能的極值點
- 使用梯度下降法的理由在於求極值點的其他方法（如傳統法、枚舉試探法）不具有可計算性，無法編程實現
- 梯度下降法可以很方便的向多元函數推廣，利於編寫程序
- 記住在這個過程中，我們要找的是極值點（使函數取極值的那一組自變量），而不是具體的極值
- 梯度下降法的劣勢在於不一定能找到全局最優解

## 二、人工神經網絡（ANN）

如果是第一次聽到人工神經網絡這個名詞，不免會覺得比較高大上，好像我們已經可以模仿神秘的神經系統了。其實它只是一個數學模型而已。當然ANN的效果是令人眼前一亮的，好像計算機一下子真的有了人的能力，可以識人、識物。

但其實稍加抽象便能發現，這個東西無非就是個**分類器**，它的輸入是一張圖片，或者確切的說就是一堆代表像素點的數值，而輸出則是一個類別。



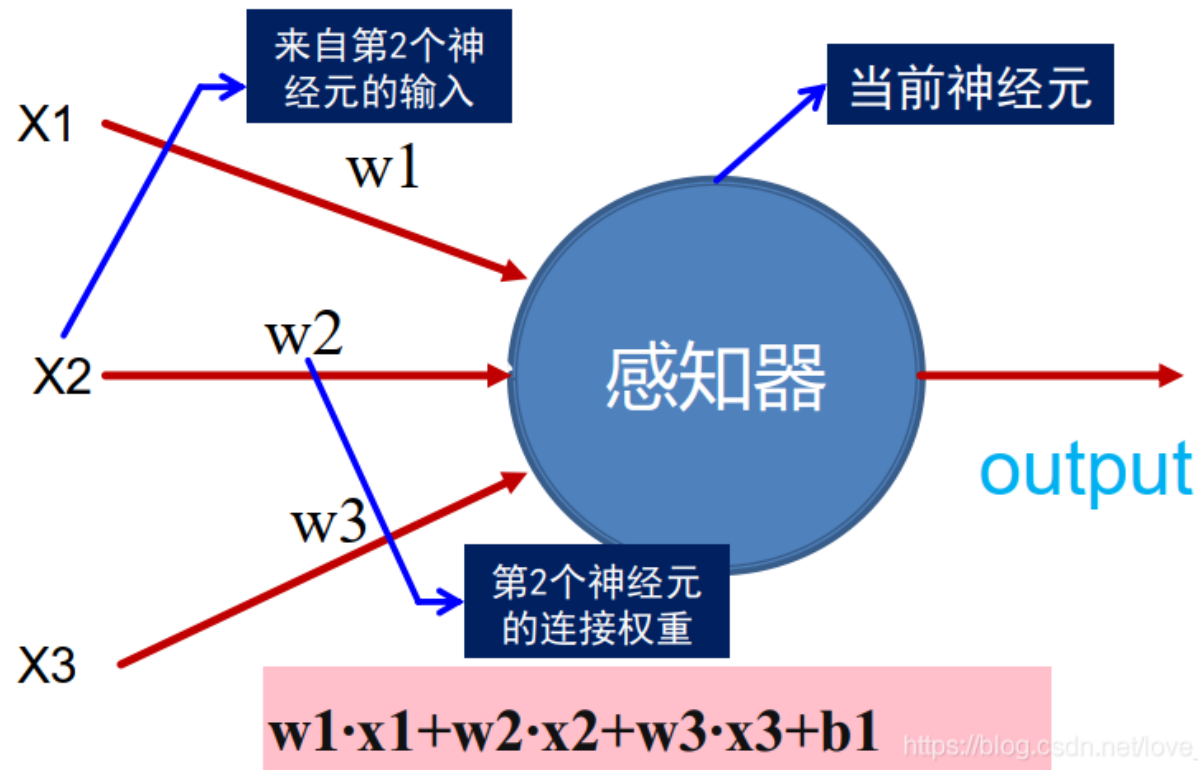
所以說白了，所謂的人工神經網絡其實就是一個超大規模的函數。

這就好比飛機和鳥兒的關係。讓飛機飛起來靠的不是依葫蘆畫瓢造一個人工鳥，而是靠流體力學中的原理建立數學模型，然後計算得出飛機的尺寸、造型，並設計相應的發動機。

## 2.1 神經元的數學模型

盜一張老師ppt裡的圖說明問題，可以看出ANN中的每一個節點（也就是所謂的神經元）就是這樣一個簡單的線性函數模型。

## □ McCulloch-Pitts神经元模型 (1943)



當然通過**激活函數**我們可以製造一點非線性的因素，以提高模型的表達能力。這樣的話下面的神經元就代表這樣一個函數

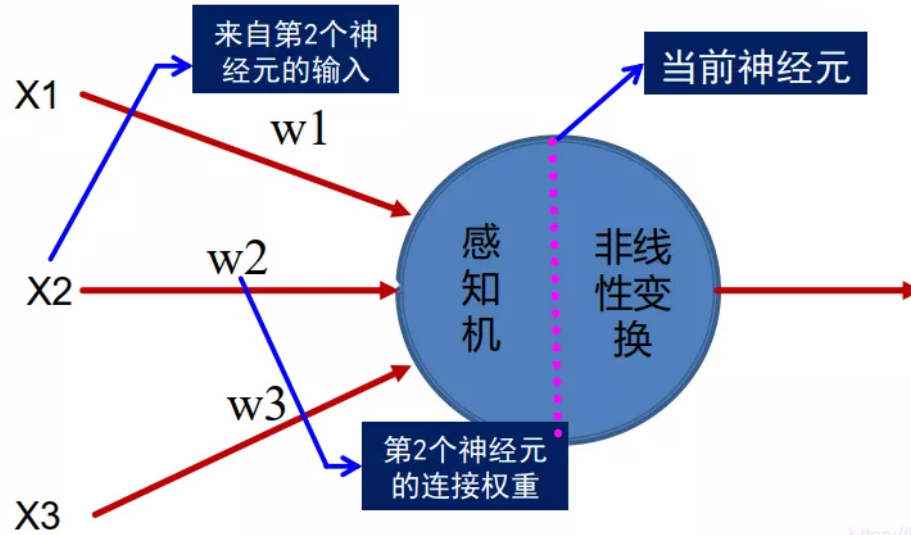
$$out(u) = \frac{1}{1 + e^{-u}}$$

其中，，這裡 $w1$ ,  $w2$ ,  $w3$ ,  $b$ 都是參數， $x1$ ,  $x2$ ,  $x3$ 是函數的輸入，也就是因變量。

$$u(x_1, x_2, x_3) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

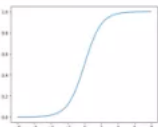
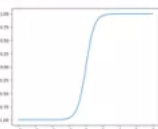

## □ 神经网络-激活函数

感知器函数  $z = wx + b$       激活函数  $\sigma(z) = \frac{1}{1 + e^{-z}}$



常用的激活函数在這裡（仍然盗用老师的ppt，捂脸逃~）

## 常用的激活函数：对输入信息进行非线性变换

名称	函数	图像	导数	值域
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$		$f'(x) = f(x) * (1 - f(x))$	$(0, 1)$
Tanh	$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$		$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{cases}$		$f'(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ 1, & \text{for } x > 0 \end{cases}$	$[0, +\infty)$

神经网络使用非线性函数作为激活函数（activation function），通过对多个非线性函数进行组合，来实现对输入信息的非线性变换

[https://blog.csdn.net/love\\_fish\\_sai](https://blog.csdn.net/love_fish_sai)

以上就是所谓的人工神经元或者叫**人造神经元**，很多很多这样的神经元按一定规则相连就构成了ANN，所以我才说ANN就是一个超大规模的函数而已。

是不是和你想像中的高大上的神经元大相径庭，但是我们现在所谓的人工智能其实就是这样的数学模型而已。无论是简单的图像分类器还是战胜人类的AlphaGo，都是靠这样的**数学计算**算出来结果的，而不是靠什么化腐朽为神奇的力量。

## 2.2 ANN是如何煉成的？

知道了ANN的本質，現在就讓我們看看得到一個ANN需要怎麼做？這裡，請留意我們會遇到不同功能的函數，千萬不要搞混了。

既然ANN是一個超大規模的函數，那麼首先我們做的就是搭建起這個函數的架構，也就是設計人工神經網絡的架構。這時這個函數就有一堆參數待定了。接下來我們準備一堆訓練數據訓練ANN，也就是把上面提到的待定參數都給他確定了。模型完成，可以使用。

顯然，最關鍵的是第三步——**確定未知參數**。

這裡首先解釋訓練數據，我們知道ANN是一個分類器也是一個函數，這個函數讀取一些輸入值，經過複雜的計算後得到輸出值，這些輸出值可以被解釋為類別。而訓練數據就是輸入值和最後的輸出值都已知的一組數據，換句話說就是已知一組函數的自變量和因變量的對應關係。

再說的明白點，我們的任務就是，已知函數的架構、函數的一組輸入值和輸出值，但不知道函數的一些參數，現在要推出這些未知參數。我把這裡我們要求出的這個函數稱之為目標函數。於是，一言以蔽之，我們的任務就是求出目標函數的未知參數。

為了完成這個任務，我們引出另一個重要的概念——損失函數。

### 2.2.1 損失函數

在這裡，我們玩一點小心機。注意了，這裡很關鍵！！

既然我們已知目標函數的一組輸入和輸出，而未知其參數，那麼我們不妨將計就計將這些未知參數直接視為因變量，而將目標函數的輸入直接代入進去，這樣我們不就得到了一個**自變量是目標函數的所有未知參數且函數整體完全已知**的函數了嗎？

這時，如果能找到一組合適的未知參數，這個函數應該能輸出和已知輸入對應的輸出完全一致的值。

於是我們可以通過作差比較定義損失函數了

损失函数

$$L(y, \tilde{y}) = \begin{cases} \frac{1}{M} \sum_{i=1}^M |y_i - \tilde{y}_i| \\ \frac{1}{M} \sum_{i=1}^M (y_i - \tilde{y}_i)^2 \end{cases}$$

[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)

上圖給出了損失函數的兩種形式（除此之外還有交叉熵損失函數等其他類型），一般需要根據不同的任務類型選取適當的損失函數。為了便於初學者理解，後文將以第二種均方誤差的形式做講解。這裡之所以出現了求和符號，是因為ANN的輸出端可能對應了不止一個函數，這些函數可以分別表示把一張圖片分成不同類別的概率。後面我們引入一個直觀的例子，一看便知。

這裡一定要注意，損失函數看起來雖然還有目標函數的影子，但實際已經完全不同了。我們列表比較一下

	目標函數	損失函數
表現形式	$f(i_1, i_2, i_3 \dots)$	$\text{loss}(w_1, w_2, w_3 \dots)$
生成方式	事先搭好框架，再通過訓練得出待定參數	將目標函數的輸出與實際值作差得到框架，然後代入一個具體的訓練樣例（包括輸入值與標籤值）
自變量	$(i_1, i_2, i_3 \dots)$ ——神經網絡的輸入值（實際場合中可以是一張圖片的所有像素值）	$(w_1, w_2, w_3 \dots)$ ——目標函數的待定參數
函數值（因變量）含義	屬於不同分類的概率	預測值與實際值的差值（越小越好）
特點	我們最終想要得到的函數，可以用來作圖像分類。是線性函數與非線性函數的組合，規模很大，自變量與參數都很多	用來求出目標函數的過渡函數。非負，最小值為0，一般要使用梯度下降法找到極值點

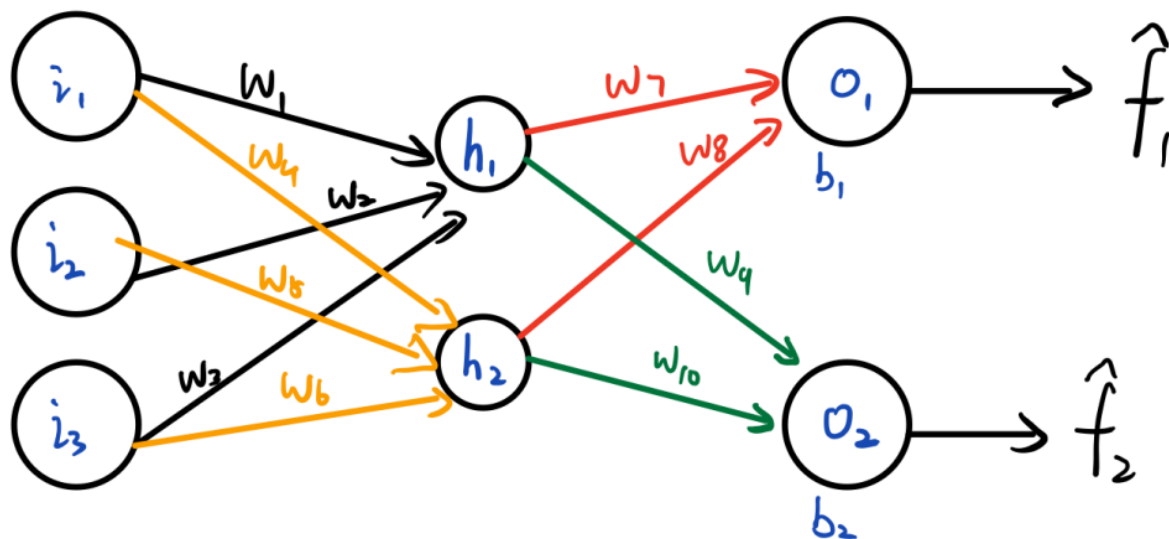
舉個例子看看函數變異的過程吧。設原函數為，這是一個關於的二元函數，其中a, b, c均是常數，也可以叫待定參數。現在我們給出一組具體的函數輸入值比如，令把它們代入函數，並且將a, b, c視為變量，則函數變成了關於a, b, c的三元函數，記作。 $f(x_1, x_2) = ax_1^2 + bx_2^2 + c$   
 $x_1, x_2$   
 $x_1 = -1, x_2 = 3$   
 $f(a, b, c) = a + 9b + c$

綜上，求目標函數的過程，就變成了尋找損失函數極小值點的過程，而尋找極小值點不正可以用上面介紹的梯度下降法實現嗎？

### 2.2.2 一個實例：關於鍊式求導和誤差反向傳播（BP）

行文至此，有關ANN的重要概念，我們還剩下鍊式求導和誤差反向傳播（BP）沒有提及，讓我們用一個實例融會貫通一下。

考慮下面這個簡單的ANN：



[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)



這個ANN只有4個神經元，分別是。它輸出兩個目標函數，均是輸入變量的函數，分別由神經元輸出。可以記為 $h_1, h_2, o_1, o_2, i_1, i_2, i_3, o_1, o_2$

$$\begin{cases} o_1 \rightarrow \hat{f}_1(i_1, i_2, i_3) \\ o_2 \rightarrow \hat{f}_2(i_1, i_2, i_3) \end{cases}$$

[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)

這裡給加上帽子，表示這兩個函數（即目標函數）的函數值是預測值，區別於訓練數據給出的實際標籤值。而均是目標函數的待定參數，這裡我們假定神經元均採用sigmoid激活函數，即，而神經元不採用激活函數。 $f_1, f_2(w_1, w_2, \dots, w_{10}, b_1, b_2)h_1, o_1, o_2g(u) = \frac{1}{1+e^{-u}}h_2$

現在定義損失函數為

$$Loss(\hat{f}_1, \hat{f}_2) = \frac{1}{2}[(\hat{f}_1 - f_1)^2 + (\hat{f}_2 - f_2)^2]$$

注意接下來我們會將具體的一組輸入變量帶進去，這樣損失函數就被視作以為自變量的多元函數（具體的自變量變化過程參見上文描述）。其中， $i_1, i_2, i_3$  是中間變量，它們均是以為自變量的多元函數。 $(i_1, i_2, i_3)(w_1, w_2, \dots, w_{10}, b_1, b_2)f_1, f_2(w_1, w_2, \dots, w_{10}, b_1, b_2)$

現在只要給出一個包含輸入輸出數據的訓練樣例，損失函數就成為不含未知參數的完全確定的函數。而我們要做的就是找到這個損失函數的極小值。

按照梯度下降算法的推導，此時我們只要按照下面的步驟就可以找出這個極小值：

1. 隨機指定一組初始參數( $w_1, w_2, \dots, w_{10}, b_1, b_2$ )
2. 計算Loss函數關於各個參數的偏導數，注意這一步要代入參數的具體數值，也就是說這一步得到的是一個數
3. 按照梯度下降的公式更新各個參數值直到滿足一定條件為止

其中2、3裡的内容是需要反复迭代的。

現在，我們以其中的幾個參數為例，看看在調整過程中會遇到什麼新問題。

先試試調整吧，這時我們需要求出損失函數對自變量的偏導數值（**注意是數值，不是表達式**），為此寫出它的依賴關係： $w_7 w_7$

$$g(u) = \frac{1}{1 + e^{-u}}$$

$$Loss(\hat{f}_1, \hat{f}_2) = \frac{1}{2} [(\hat{f}_1 - f_1)^2 + (\hat{f}_2 - f_2)^2]$$

$$\hat{f}_1(w_7, w_8, b_1) = g(w_7 h_1 + w_8 h_2 + b_1) = g(u_1)$$

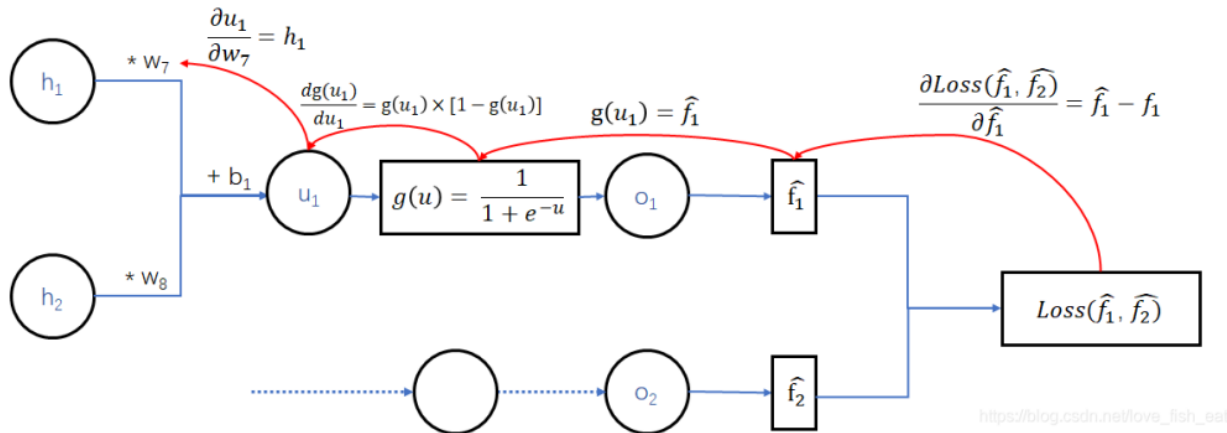
$$h_1(w_1, w_2, w_3) = g(w_1 i_1 + w_2 i_2 + w_3 i_3) = g(u_2)$$

$$h_2(w_4, w_5, w_6) = w_4 i_1 + w_5 i_2 + w_6 i_3$$

$$u_1 = w_7 h_1 + w_8 h_2 + b_1$$

$$u_2 = w_1 i_1 + w_2 i_2 + w_3 i_3$$

[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)



這裡，Loss函數依賴於變量，但與無關。回想多元函數求偏導數的規則，我們對求導時，應將視為常數。 $\hat{f}_1, \hat{f}_2, w_7, w_8, b_1$

而依賴於變量，因此這裡應按照複合函數求導法則，即傳說中的**鍊式求導法則**，先讓Loss函數對變量求導，再令對求導，即： $\frac{\partial Loss}{\partial \hat{f}_1} w_7, w_8, b_1, \frac{\partial \hat{f}_1}{\partial w_7}$

$$\begin{aligned}
 \left( \frac{\partial Loss}{\partial w_7} \right)_w &= \left( \frac{\partial Loss}{\partial \hat{f}_1} \right)_w \times \left( \frac{\partial \hat{f}_1}{\partial w_7} \right)_w \\
 &= \left( \frac{\partial Loss}{\partial \hat{f}_1} \right)_w \times \left( \frac{dg}{du_1} \right)_w \times \left( \frac{\partial u_1}{\partial w_7} \right)_w \\
 &= (\hat{f}_1 - f_1)_w \times [g(u_1) \times (1 - g(u_1))]_w \times (h_1)_w
 \end{aligned}$$

其實從神經網絡的圖中可以很清楚的看出求導鏈。

這裡有幾個要點：

- 首先，式子的每一項均加下標 $w$ ，表示要將具體的一組代入式子，得到一個數值  
 $(w_1, w_2, \dots, w_{10}, b_1, b_2)$
- 其次，當 $g(u)$ 表示sigmoid函數時，對其求導的結果就是 $g(u) * [1 - g(u)]$
- 函數對變量求偏導時，雖然也是函數，但它是關於自變量的函數，與無關，因此視為常數。  
這樣，對求偏導的結果就是 $u_1(w_7, w_8, b_1) = w_7 h_1 + w_8 h_2 + b_1 w_7 h_1 w_1, w_2, w_3 w_7 w_7 h_1$
- 等式右端最後得出的三項，在給出一個訓練樣例，並指定初始參數後，是可以獨立計算出結果的 $(w_1, w_2, \dots, w_{10}, b_1, b_2)$

求出損失函數對的偏導數值，我們就可以按照梯度下降算法推導的公式，調整這個參數了！ $w_7$

---

現在，再來看看靠前的參數是怎麼調整的，我們以和為例 $w_1 w_6$

還是老規矩，對照神經網絡圖，先寫出它的依賴關係：

$$g(u) = \frac{1}{1 + e^{-u}}$$

$$Loss(\hat{f}_1, \hat{f}_2) = \frac{1}{2} [(\hat{f}_1 - f_1)^2 + (\hat{f}_2 - f_2)^2]$$

$$\hat{f}_1(w_7, w_8, b_1) = g(w_7 h_1 + w_8 h_2 + b_1) = g(u_1)$$

$$\hat{f}_2(w_9, w_{10}, b_2) = g(w_9 h_1 + w_{10} h_2 + b_2) = g(u_3)$$

$$h_1(w_1, w_2, w_3) = g(w_1 i_1 + w_2 i_2 + w_3 i_3) = g(u_2)$$

$$h_2(w_4, w_5, w_6) = w_4 i_1 + w_5 i_2 + w_6 i_3$$

$$u_1 = w_7 h_1 + w_8 h_2 + b_1$$

$$u_2 = w_1 i_1 + w_2 i_2 + w_3 i_3$$

$$u_3 = w_9 h_1 + w_{10} h_2 + b_2$$

[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)

可以看出和，分別是函數和的變量，而函數均與和有關，所以Loss函數需要對均求偏導。

$w_1 w_6$   
 $h_1 h_2 \hat{f}_1, \hat{f}_2 h_1 h_2 \hat{f}_1, \hat{f}_2$

依然按照**鍊式求導法則**對求偏導，有： $w_1$

$$\begin{aligned}
 \left(\frac{\partial Loss}{\partial w_1}\right)_w &= \left(\frac{\partial Loss}{\partial \hat{f}_1}\right)_w \times \left(\frac{\partial \hat{f}_1}{\partial h_1}\right)_w \times \left(\frac{\partial h_1}{\partial w_1}\right)_w + \left(\frac{\partial Loss}{\partial \hat{f}_2}\right)_w \times \left(\frac{\partial \hat{f}_2}{\partial h_1}\right)_w \times \left(\frac{\partial h_1}{\partial w_1}\right)_w \\
 &= \left(\frac{\partial Loss}{\partial \hat{f}_1}\right)_w \times \left(\frac{dg}{du_1}\right)_w \times \left(\frac{\partial u_1}{\partial h_1}\right)_w \times \left(\frac{dg}{du_2}\right)_w \times \left(\frac{\partial u_2}{\partial w_1}\right)_w \\
 &\quad + \left(\frac{\partial Loss}{\partial \hat{f}_2}\right)_w \times \left(\frac{dg}{du_3}\right)_w \times \left(\frac{\partial u_3}{\partial h_1}\right)_w \times \left(\frac{dg}{du_2}\right)_w \times \left(\frac{\partial u_2}{\partial w_1}\right)_w \\
 &= \{(\hat{f}_1 - f_1)_w \times [g(u_1) \times (1 - g(u_1))]\}_w \times (w_7)_w + \{(\hat{f}_2 - f_2)_w \times [g(u_3) \times (1 - g(u_3))]\}_w \times (w_9)_w \\
 &\quad \times [g(u_2) \times (1 - g(u_2))]\}_w \times i_1
 \end{aligned}$$

[https://blog.csdn.net/love\\_fish\\_eat](https://blog.csdn.net/love_fish_eat)

對求偏導，有： $w_6$

$$\begin{aligned}
 \left(\frac{\partial Loss}{\partial w_6}\right)_w &= \left(\frac{\partial Loss}{\partial \hat{f}_1}\right)_w \times \left(\frac{\partial \hat{f}_1}{\partial h_2}\right)_w \times \left(\frac{\partial h_2}{\partial w_6}\right)_w + \left(\frac{\partial Loss}{\partial \hat{f}_2}\right)_w \times \left(\frac{\partial \hat{f}_2}{\partial h_2}\right)_w \times \left(\frac{\partial h_2}{\partial w_6}\right)_w \\
 &= \left[ \left(\frac{\partial Loss}{\partial \hat{f}_1}\right)_w \times \left(\frac{dg}{du_1}\right)_w \times \left(\frac{\partial u_1}{\partial h_2}\right)_w + \left(\frac{\partial Loss}{\partial \hat{f}_2}\right)_w \times \left(\frac{dg}{du_3}\right)_w \times \left(\frac{\partial u_3}{\partial h_2}\right)_w \right] \times \left(\frac{\partial h_2}{\partial w_6}\right)_w \\
 &= \{(\hat{f}_1 - f_1)_w \times [g(u_1) \times (1 - g(u_1))]\}_w \times (w_8)_w + \{(\hat{f}_2 - f_2)_w \times [g(u_3) \times (1 - g(u_3))]\}_w \times (w_{10})_w \times i_3
 \end{aligned}$$

注意紅框圈出來的部分是不是有些眼熟？

事實上，這一部分已經在調整後層參數的時候計算過了（請回看計算時的計算公式）。因此在編程時，可以讓程序保存中間結果，這裡直接拿來用。 $w_7$

現在縱觀整個過程，我們驚奇的發現，對於ANN，當我們需要使用它時，是從最前面給出輸入，然後一步步往後計算得出這個龐大複雜函數的輸出的；而當我們需要訓練它時，則是從最後面的參數開始，一步步向前求導，調整各個參數的。並且計算前面的參數時一般都會用到之前計算過的中間結果。

這樣，ANN調整參數的過程就可以看作是一個**誤差反向傳播**（BP）的過程。

之所以會這樣反向傳播，是因為神經網絡中靠後的參數依賴的中間變量少、複合層數少，而靠前的參數則經過層層複合，求導鏈會拉的很長。

### 2.2.3 最後的一點小問題

以上我們將求偏導的過程整個過了一遍，而求偏導只是梯度下降算法的一環。

程序跑起來之後，我們會對每一個訓練樣例，一遍遍的求偏導，直到基本上找到極小值點。

也就是說，按照梯度下降算法，每一個訓練樣例都會最終給出一組參數值。

一個訓練集中顯然會有多個訓練樣例，因此最終會得到**好多組各不相同的參數值**。

而ANN的訓練目標是**確定一組參數值**，得到一個有一定效果的很複雜的函數，

### 怎麼解決？

對每一個參數，我們可以將不同訓練樣例得出的不同值求一個平均。也可以構建一個更大的損失函數，即將每一個訓練樣例生成的損失函數求和，然後用梯度下降算法找到這個累計損失函數的



極小值點。

這樣，我們就能通過訓練，最終確定ANN這個大函數的所有待定參數，然後用它來做一些神奇的事情。

## Datawhale

### 和学习者一起成长

一个专注于AI的开源组织，让学习不再孤独



长按扫码关注我們

“乾貨學習 · 點贊三連 ↓