

5 自动求导

5 自动求导

0 引言

1 向量的链式法则

1.1 情况1

1.2 情况2

1.3 情况3

2 链式求导实例

2.1 例1

2.2 例2

3 自动求导计算机原理

3.1 计算图原理

3.2 运行模式

4 自动求导的代码实现

5 finish

5.1 重要函数

5.2 代码全部

Important

B站视频链接 [5 自动求导](#)

0 引言

说个题外话，这个章节好难懂，看了好久也不是特别的理解，究竟在说些什么玩意儿，只能硬着头皮听下去了。

由于在进行深度学习神经网络的计算过程中对求导数这个运算较为重要因此，这里就需要进行实现自动求导这个功能，来帮助进行运算，这个玩意儿应该算是一个基础运算，在深度学习之中。

在使用前我们需要重新复习一下之前求导数的方法，也就是极为知名的**链式求导法则**。

我们文中提到的全部向量在第一次定义的时候均是列向量，也就是站着的那种向量类型

1 向量的链式法则

对于标量而言他的链式求导法则还是比较简单明了的，就用下面这个例子表示一下。

没有标量: $y = f(u)$, $u = g(x)$ 求 $\frac{\partial y}{\partial x}$?

$y - u - x$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

现在我们将标量的链式求导法则扩展到向量进行求解。

$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$
 \downarrow 拓展到向量

1. $y = f(u)$, $u = g(\vec{x}) \Rightarrow \frac{\partial y}{\partial \vec{x}} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial \vec{x}} \Rightarrow [\dots]$

2. $y = f(\vec{u})$, $\vec{u} = g(\vec{x}) \Rightarrow \frac{\partial y}{\partial \vec{x}} = \frac{\partial y}{\partial \vec{u}} \cdot \frac{\partial \vec{u}}{\partial \vec{x}} \Rightarrow [1, n]$

3. $\vec{y} = f(\vec{u})$, $\vec{u} = g(\vec{x}) \Rightarrow \frac{\partial \vec{y}}{\partial \vec{x}} = \frac{\partial \vec{y}}{\partial \vec{u}} \cdot \frac{\partial \vec{u}}{\partial \vec{x}} \Rightarrow [m \times n]$

Diagram illustrating the dimensions of the derivatives in the vector chain rule:

- For case 1: y is a scalar, \vec{x} is a column vector of size n . $\frac{\partial y}{\partial \vec{x}}$ is a row vector of size $1 \times n$.
- For case 2: y is a scalar, \vec{u} is a column vector of size k . $\frac{\partial y}{\partial \vec{u}}$ is a row vector of size $1 \times k$, and $\frac{\partial \vec{u}}{\partial \vec{x}}$ is a matrix of size $k \times n$. The result is a row vector of size $1 \times n$.
- For case 3: \vec{y} is a column vector of size m , \vec{u} is a column vector of size k . $\frac{\partial \vec{y}}{\partial \vec{u}}$ is a matrix of size $m \times k$, and $\frac{\partial \vec{u}}{\partial \vec{x}}$ is a matrix of size $k \times n$. The result is a matrix of size $m \times n$.

可以看到引入向量之后完全不一样了。

这是一个常量。

u 是一个标量.

则有 $\frac{\partial Y}{\partial \vec{x}} = \frac{\partial Y}{\partial u} \cdot \frac{\partial u}{\partial \vec{x}}$

$\Rightarrow \begin{cases} \frac{\partial Y}{\partial \vec{x}} \text{ 是一个 } n \text{ 列行向量} \\ \frac{\partial Y}{\partial u} \text{ 是一个单位标量} \\ \frac{\partial u}{\partial \vec{x}} \text{ 也是一个 } n \text{ 列行向量} \end{cases}$

$$\boxed{\text{long rectangle}} = \boxed{\text{small square}} \times \boxed{\text{long rectangle}}$$

1.2 情况2

此时 y 是一个标量 而 \vec{u}, \vec{x} 均为列向量。

$$\vec{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_k \end{bmatrix}_{k \times 1} \quad \vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}_{n \times 1}$$

例 $\frac{\partial y}{\partial \vec{x}} = \frac{\partial y}{\partial \vec{u}} \times \frac{\partial \vec{u}}{\partial \vec{x}} \Rightarrow$

- $\frac{\partial y}{\partial \vec{x}}$ 是一个 n 列 行向量
- $\frac{\partial y}{\partial \vec{u}}$ 是一个 k 列 行向量
- $\frac{\partial \vec{u}}{\partial \vec{x}}$ 是一个 $(k \times n)$ 的行阵

The diagram consists of three hand-drawn shapes. On the left is a horizontal rectangle labeled 'n' above it. In the middle is a horizontal rectangle labeled 'k' above it. To the right of the middle rectangle is a large square labeled 'n' at the bottom right. There is an equals sign '=' between the first and second rectangles. Above the middle rectangle, there is a label 'k' and a label 'n'.

1.3 情况3

\vec{y} 是一个 m 行的向量 $m \times 1$
 \vec{x} 也是一个 n 行的向量 $n \times 1$ 则必有
 \vec{w} 也是一个 k 行的向量 $k \times 1$

$\frac{\partial \vec{y}}{\partial \vec{x}} = \frac{\partial \vec{y}}{\partial \vec{w}} \cdot \frac{\partial \vec{w}}{\partial \vec{x}}$

$\frac{\partial \vec{y}}{\partial \vec{x}}$ 是一个 m 行 n 列 ($m \times n$) 的矩阵
 $\frac{\partial \vec{y}}{\partial \vec{w}}$ 是一个 m 行 k 列 ($m \times k$) 的矩阵
 $\frac{\partial \vec{w}}{\partial \vec{x}}$ 是一个 k 行 n 列 ($k \times n$) 的矩阵

上面三种情况从简单到难依次罗列了向量求导过程中的形状和大小，现在对向量的链式求导法则应该是了熟于胸了，那么现在就开始试着采用这个求导法则来真的试试求导的几个例子吧

2 链式求导实例

2.1 例1

题目：令 \vec{x} 和 \vec{w} 均为 n 行的列向量，即 $\vec{x}, \vec{w} \in R^n$ 。 y 是一个标量，即 $y \in R$ 。令 $z = (\langle \vec{x}, \vec{w} \rangle - y)^2$ ，求 $\frac{\partial z}{\partial \vec{w}}$ 。

首先让我们分析一下这个题目，他让我们求导数，因此考虑采用链式求导法则进行求解。注意尖括号里面这两个向量做的是点乘，也就是内积哈。

首先先定义一些中间变量：

$$\text{令: } \begin{cases} a = \langle \vec{x}, \vec{w} \rangle \\ b = a - y \\ z = b^2 \end{cases} \quad (2-1)$$

则可以将进行链式求导法则得到

$$\begin{aligned}
\frac{\partial z}{\partial \vec{w}} &= \frac{\partial z}{\partial b} \times \frac{\partial b}{\partial a} \times \frac{\partial a}{\partial \vec{x}} \\
&= \frac{\partial b^2}{\partial b} \times \frac{\partial(a-y)}{\partial a} \times \frac{\partial(\langle \vec{x}, \vec{w} \rangle)}{\partial \vec{w}} \\
&= 2b \times 1 \times \vec{x}^T \\
&= 2(a-y)\vec{x}^T \\
&= 2(\langle \vec{x}, \vec{w} \rangle - y)\vec{x}^T
\end{aligned} \tag{2-2}$$

2.2 例2

题目：令 $X \in R^{m \times n}$, $\vec{w} \in R^n$, $\vec{y} \in R^m$, $z = \|X\vec{w} - \vec{y}\|^2$, 求 $\frac{\partial z}{\partial \vec{w}}$ 。

首先和前面一样的方法先搞几个中间变量来，将z尽可能的规划的简单一些。

$$\text{令: } \begin{cases} a &= X\vec{w} \\ b &= a - \vec{y} \\ z &= \|b\|^2 \end{cases} \tag{2-3}$$

现在就可以采用**伟大的**链式法则来求解一下这个题目了

$$\begin{aligned}
\frac{\partial z}{\partial \vec{w}} &= \frac{\partial z}{\partial b} \times \frac{\partial b}{\partial a} \times \frac{\partial a}{\partial \vec{w}} \\
&= \frac{\partial \|b\|^2}{\partial b} \times \frac{\partial(a-\vec{y})}{\partial a} \times \frac{\partial(X\vec{w})}{\partial \vec{w}} \\
&= 2b^T \times 1 \times X \\
&= 2(a-\vec{y})^T X \\
&= 2(X\vec{w} - \vec{y})^T X
\end{aligned} \tag{2-4}$$

到这里，这两个举例应该让链式法则求导数这件事情变得比较的清晰了一些，但是我们总不能一直用手来算吧，一直用手算还叫什么自动求导呀！

现在就开始进行计算机的求导方法。

3 自动求导计算机原理

3.1 计算图原理

在计算机里面进行求导，其原理和我们手动求导的原理是一样的，也是利用中间变量配合链式法则进行求导，他的求导过程只需要下面两个东西。

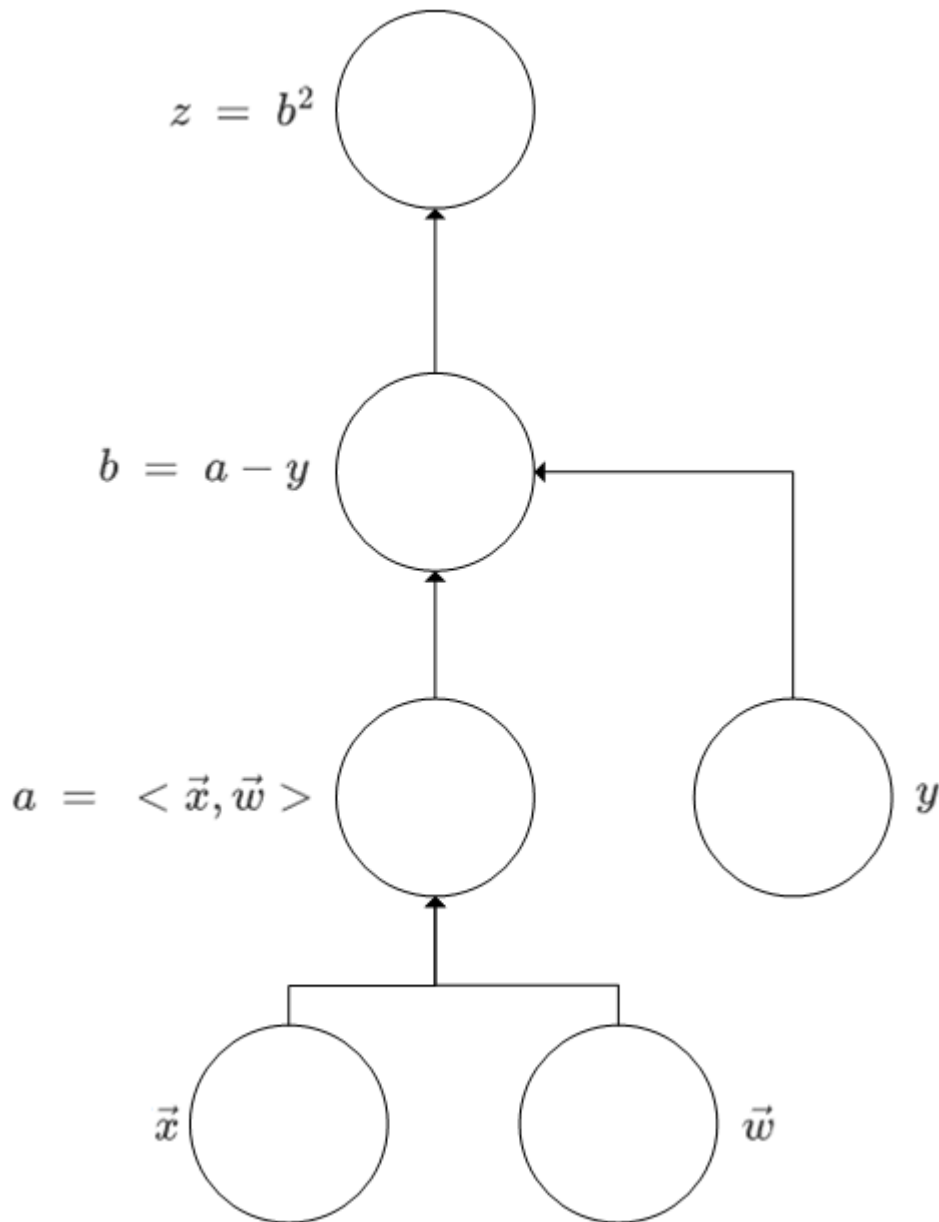
1. 计算的代码块也就是操作子。
2. 计算无环图片。

那么我们就拿着**例1**的题目进行举例说明。

以这个题目为例进行求解的过程主要分成了三个步骤

1. \vec{w} 和 \vec{x} 产生中间变量 a 。
2. a 和 y 产生中间变量 b 。
3. b 产生应变变量 z

那么我们可以画出他的计算图（无环的）



在计算机里面便是如此。

3.2 运行模式


不过在计算机的运行过程中对于求导的模式不仅仅是上面说的一种，实际上主要是两种运行模式。

上文说到的就是其中的一种也就是**正向运行**模式，另外一种就是**反向运行**模式也可以称之为**反向传递**。

正反向的区别也就是咱们思维的区别，一般的链式法则都是由很多的中间变量 u_i 组成的那么对于一个任何一个函数都可以像是下面这个样子进行求导：


$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \frac{\partial u_{n-1}}{\partial u_{n-2}} \dots \frac{\partial u_2}{\partial u_{n1}} \frac{\partial u_1}{\partial x}$$

那么对于**正向**模式而言，它是从后向前的一个求导方法的

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \frac{\partial u_{n-1}}{\partial u_{n-2}} \dots \frac{\partial u_2}{\partial u_{n1}} \frac{\partial u_1}{\partial x}$$


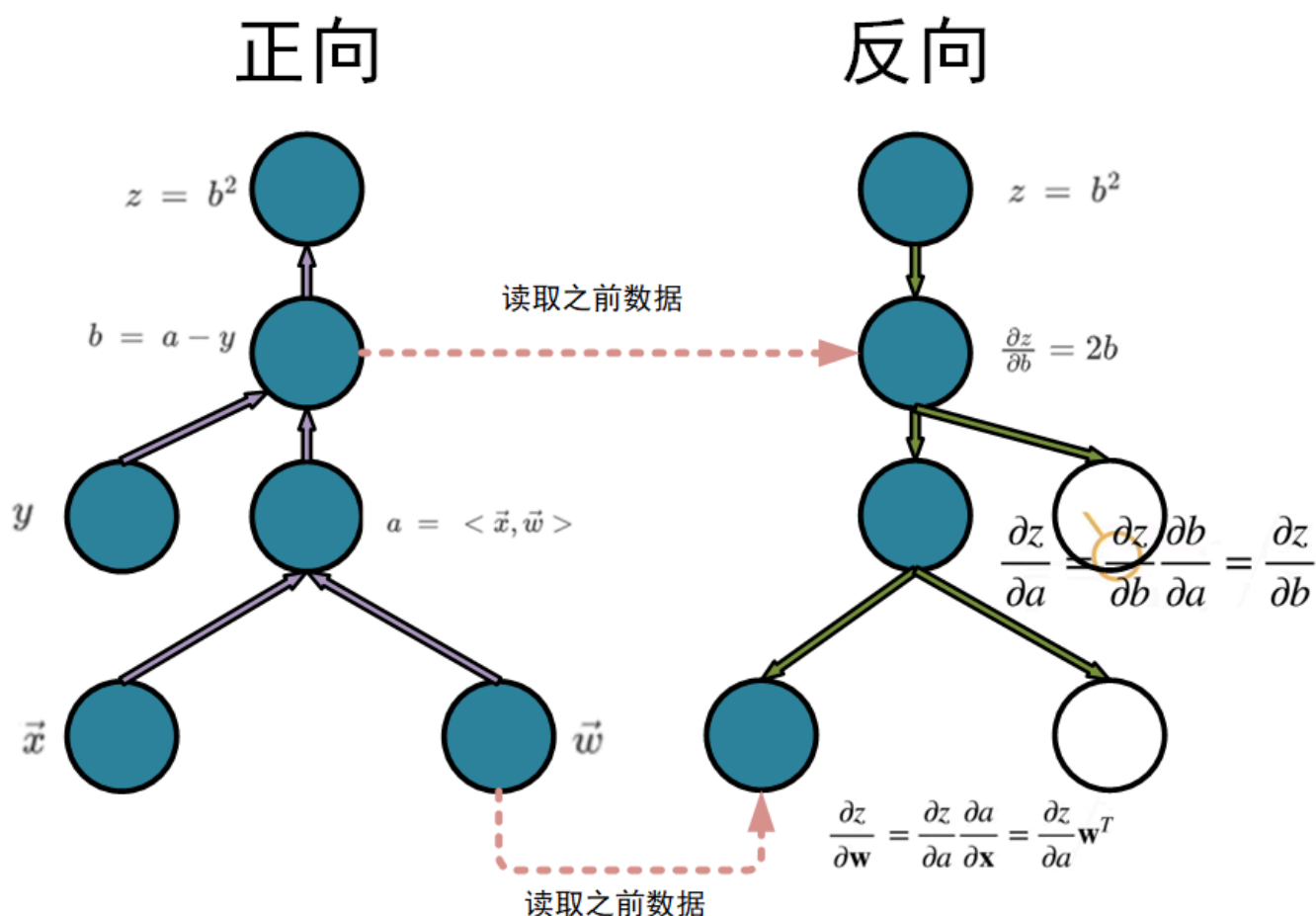
也就是先是对 x 然后逐步扩张到 y

而对于**反向**模式而言，它和**正向**模式恰好相反，他是从前向后的一个求导方法的

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \frac{\partial u_{n-1}}{\partial u_{n-2}} \dots \frac{\partial u_2}{\partial u_{n1}} \frac{\partial u_1}{\partial x}$$


也就是先是对 y 然后逐步扩张到 x

下面这个图就十分形象的表现了正向和反向的一个小区别，当然大多数时候咱们主要采用反向进行电脑求解



综上所述，正向求导和反向求导最显著的区别就是。

正向求导就是对一个玩意儿猛猛求，只有一个数据，我只存了一个数，不断的对这个数进行更新，最终得到函数。

反向求导就是在求解的过程总会保存曾经的数据，把数都存起来，到时候求什么就取需要的数来用。

好那么到现在这个求解的基本方法是已经搞得差不多了。现在开始对自动求导进行计算机上的演示。

4 自动求导的代码实现

首先需要说明一下在进行代码实现之前我们在计算机里面需要正确安装下面引用的库，并正确配置环境同时 `import` 进入

```
1 import torch
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import cv2 as cv
```

```
1 import torch
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import cv2 as cv
```

```
1 x = torch.arange(4.0) #定义向量x
2 x.requires_grad_(True) #开始存储梯度
3 x.grad #访问梯度
```

当前访问中 `x` 的梯度为 `None`

```
(DL) E:\code>E:/AppInstallation/0_4_annaconda/envs/DL/python.exe "e:/code/4.Deep Learning_basis/4 自动求导.py"
```

`None`

好让我们继续定义函数，随便定义一个比较简单的函数 $y = 2x^T x$

```
1 y = 2*torch.dot(x,x) #定义函数y = 2 xT x
2 y.backward() #求导、反向传递
3 print(x.grad) #看x的梯度
```

```
y = 2*torch.dot(x,x) #定义函数y = 2 xT x
y.backward() #求导、反向传递
print(x.grad) #看x的梯度
```

可以看到结果

```
(DL) E:\code>E:/AppInstallation/0_4_annaconda/envs/DL/python.exe "e:/code/4.Deep Learning_basis/4 自动求导.py"
tensor([ 0.,  4.,  8., 12.])
```

我们知道哈 $y = 2x^2$ 这东西的导数为 $y' = 4x$ 那我们验证一下正不正确哈。


```
1 Text = x.grad ==4*x
2 print(Text)
```

```
(DL) E:\code>E:/AppInstallion/0_4_annaconda/envs/DL/python.exe "e:/code/4.Deep
Learning_basis/4 自动求导.py"
tensor([True, True, True, True])
```

可以看到运行的结果是 `True` 说明咱们的反向求导正确了。

现在我们想要重新定义一个新的函数，在使用前一定要先进行之前梯度缓存的清除工作才可以进行创建。

```
1 x.grad.zero_() #清除之前的梯度缓存
2 y = x.sum() #重新建立一个新的函数
3 print(y)
4 y.backward() #求导、反向传递
5 GRAD_x = x.grad #访问梯度
6 print(GRAD_x)
```

```
(DL) E:\code>E:/AppInstallion/0_4_annaconda/envs/DL/python.exe "e:/code/4.Deep
Learning_basis/4 自动求导.py"
tensor(6., grad_fn=<SumBackward0>)
tensor([1., 1., 1., 1.])
```

注意sum就是向量x和全1向量的内积

也就是全部加一起

5 finish

5.1 重要函数

1. 星号 `*`：对于标量也就是一个数据直接相乘 $4*4=16$ ，但是对于两个形状相同的向量而言是对应元素相乘。
2. `sum()`：这个函数的意思就是将其向量本身和一个全1向量做内积
3. `.grad.zero_()`：清除缓存。
4. `.grad`：访问当前梯度，这个默认是 `false`，需要用6的函数进行开启。
5. `.backward()`：求导、反向传递。
6. `.requires_grad_()`：开始存储梯度，`True` 允许，`False` 不允许。
7. `.detach()`：就是将一个东西和本身的这个向量分离，二者无关，可以视作常量。

5.2 代码全部

```

1  import torch
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import cv2 as cv
5  import math
6
7  x = torch.arange(4.0) #定义向量x
8  x.requires_grad_(True) #开始存储梯度
9  x.grad #访问梯度
10
11 y = 2*torch.dot(x,x) #定义函数 $y = 2 \mathbf{x}^T \mathbf{x}$ 
12 y.backward() #求导、反向传递
13
14 Text = x.grad == 4*x
15
16 x.grad.zero_() #清除之前的梯度缓存
17 y = x.sum() #重新建立一个新的函数
18 #print(y)
19 y.backward() #求导、反向传递
20 GRAD_x = x.grad #访问梯度
21 #print(GRAD_x)
22
23 x.grad.zero_() #清除之前的梯度缓存
24 y=x*x
25 # 等价于 y.backward(torch.ones(len(x)))
26 y.sum().backward() #求导、反向传递
27 GRAD_x = x.grad
28 # print(GRAD_x)
29
30 x.grad.zero_()
31 y = x* x #对应元素相乘
32 print(y)
33 u = y.detach() #u就相当于一个常量 并不是变量
34 z = u * x
35 z.sum().backward() #求导、反向传递
36 GRAD_x = x.grad
37 Test = x.grad == u #检验
38 # print(GRAD_x)
39 # print(Test)

```