緣由

因爲本身工作上被指派到一個跟偏微分方程有關的任務, 於是我複習起一些以前學過的微分方程的知識. 想到幾年前有一篇叫做 NeuralODE 的論文, 雖然跟分派的任務大概沒什麼關係, 但是想說利用空閒 時間一併讀看看. Adjoint method 一詞我就是第一次在那篇論文裏讀到; 本身雖然是數學系畢業的, 但是讀的時候完全不懂那是什麼樣的一個方法, 所以覺得很不好意思. 於是就上網找找相關的解說, 發現似乎在應用的領域, 特別是氣象學裏, 這似乎是一個很多人使用的一種計算方式, 但是從開始 尋覓對於這個 adjoint method 的理解至今大約有一個禮拜的時間了, 我還是不是很清楚這個東西.

這個 notebook 的目的在於

1. 記錄下自己的理解
2. 讓別人更容易看到並幫助解決自己的問題
3. 將來有更正確的理解再往下繼續寫

```
# begin
#     using Pkg
#     Pkg.activate(Base.Filesystem.homedir() * "/.config/julia/projects/oft")
#     using Plots
#     using LinearAlgebra
# end
```

文章

以下是少數的幾篇我讀過的文章:

- **Notes on Adjoint Methods for 18.335, *Steven G. Johnson***
- **What Is an Adjoint Model?, *Ronald M. Errico***

Johnson 先生的文章是我第一個選擇讀的, 因爲它看起來似乎比較平易近人. 不過我大致讀過一次以後, 發現自己還是不太懂這一切到底在搞什麼, 於是該天晚上我轉而去讀 Errico 先生的文章. 我自己的經驗是 Errico 先生的文章寫得落落長, 但是不容易抓到作者到底要講什麼; 而 Johnson 先生的文章算式比較多, 但是還是有點隱晦.

諷刺的是, 我是在讀 Errico 覺得很睏, 很無趣的時候, 突然意識到 Neural Network 的簡單, 具體的例子在 Johnson 的文章裏應該對應到哪些函數, 哪些式子.

Johnson 先生把對於能應用 adjoint method 的問題歸結如下:

> 首先我們有一個向量 $p \in \mathbb{R}^{n_p}$ 可以想成是 Neural Network 的參數 (p for "***p**arameters*")
>
> 隨之, 我們有一個向量 $x \in \mathbb{R}^{n_x}$ 以及 一個 x 和 p 共同滿足的等式 $f(x,p) = 0$, 其中 $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$. (這個等式是我們之後要解 $x = x(p)$ 用的, 所以值域的維度要等於 x 所在空間的維度.)
>
> 最後是我們要解決的問題: 給一實值 (i.e. real-valued) 函數 $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}$, 試着找出這個函數的最小值 $g(x,p)$ 其中 x, p 滿足 $f(x,p) = 0$.

附註. Johnson 先生的文章比較像是上課用的講義, 寫得很快, 沒有很清楚, 以上算是加入我自己的理解和整理後的樣子, 希望沒有失真/正確性.

對於有學過 Neural Network 的讀者, 或許我們可以更具體地想成以下這個樣子:

> 我們把手上的 Neural Network 看成一個函數 $h : p \mapsto h(p)$, 比如說, 如果是作 MNIST classification 的話, 就是 $h : \mathbb{R}^{n_p} \to \mathbb{R}^{10}$.
>
> 上面的 $f(x,p) = 0$, 在我們現在看來就是 $f(x,p) = x - h(p) = 0$. (其實也就只是 $x = h(p)$.)
>
> 最後 g 在我們的這個實例裏, 就是 loss function (i.e. 我們要最小化我們的 loss.)

今天是 2021 年 6 月 7 號, 我的理解也就只到這裏而已, 而且我覺得更慘的是, 我並不覺得上面這個例子用 adjoint method 有什麼幫助. 以下是我的說明:

在我們具體的 Neural Network 例子裏, $g(x,p) = g(x)$; 亦即, g 其實只依賴於 x, 並不依賴於 p. 的確, x 本身就是 Neural Network 的 output 了, 一般的 loss, 像是 cross-entropy loss 此時只需要 x 和 labels 就可以算出來, 算式上並不額外依賴於 p.

如此一來, gradient descent 我們得算 (這裏我們用上面提到的 Johnson 先生的文章裏第 2 節 [Linear equations] 的符號) $\frac{dg}{dp} = g_x x_p$.

如果不知道 adjoint method, 這個東西我們會直接解:

$$\frac{dg}{dp} = g_x x_p = g_x \frac{dh}{dp}.$$

但是, 如果知道 adjoint method, 我們**還是只能這樣**:

$$\frac{dg}{dp} = g_x x_p = (g_x A^{-1})(b_p - A_p x).$$

讓我把這個講得更清楚一點, 並試圖說服讀者, 這裏 adjoint method 幫不上忙. 上面的式子是這樣來的: 記得我們的 $f(x,p) = x - h(p) = 0$ 嗎? 在 Johnson 先生的記號裏, 這可以寫成 $Ax - b = 0$, 其中 $A = I, b = b(p) = h(p)$. 於是乎, 我們可以把上面的式子繼續化簡, 看看最後我們可以得到什麼:

$$\frac{dg}{dp} = (g_x A^{-1})(b_p - A_p x) = (g_x I^{-1}) b_p = g_x \frac{dh}{dp} .$$

所以我們得到了一個一模一樣的式子.

話說, adjoint methods 或許在其他微分方程的例子真的帶來了什麼助力, 但是單就 Neural Network 來看, 我目前還不清楚它到底幫上了什麼忙, 就連 reverse-mode autodiff, 我也不太懂跟 adjoint method 有什麼關係. 如果讀者有人看到或想到了什麼, 歡迎和筆者聯絡.