

Juliaと歩く 自動微分の世界

Julia Tokyo #11

@abap34




2023/02/03

自己紹介

@abap34

東工大 情報理工学院 情報工学系 B2
(もうすぐ消滅)

趣味

-  機械学習
-  個人開発
-  野球をする/みる

GitHub: @abap34

Twitter: @abap34

自己紹介

✓ Juliaをこんなことに使
ってます！

1. データ分析
2. 機械学習や数学の勉強・調べ物
3. 競プロ

✓ Juliaのこんなところが気に入ってます！

1. 綺麗な可視化・ベンチマークライブラリ
 - Plotまわり, `@code_...` マクロ, `BenchmarkTools.jl` たち
2. パッケージ管理ツール
 - 言語同梱で超便利 (友人間で共有するのに一苦勞の言語も)
3. すぐ書ける すぐ動く
 - Jupyter のサポート, 強力な REPL
4. 速い！！
 - 速度は正義
 - 裏が速いライブラリの「芸人」にならなくても、素直に書いてそのまま速い

Julia を使って解かれた・書かれたレポートたち

今日のお話

.....

:自作DLフレームワーク

one of 興味があるもの

機械学習(特に深層学習)

の基盤

one of 深層学習の基盤

自動微分

について話します.

こんなことはありませんか？

1. 深層学習フレームワークを使っているけど、あまり中身がわかっていない.
2. 微分を求めたいことがあって既存のライブラリを使っているけど、
どの場面でどれを使うのが適切かわからない
3. 自分の計算ルーチンに微分の計算を組み込みたいけどやり方がわからない
4. え、自動微分ってただ計算するだけじゃないの？何がおもしろいの？

[メインテーマ]

- 自動で微分を求めることのモチベーション
- 自動で微分を求めるアルゴリズムたちの紹介と実装
- 一般的な自動微分の実装
- 自動微分の先進的な研究 (Julia まわりを中心に)
- 微分ライブラリの紹介

こんなワードが出てきます：

勾配降下法, 数値微分, 数式微分, 自動微分, 誤差評価, 深層学習フレームワーク, Define and/by Run
計算グラフ, Wengert List, Source Code Transformation(SCT), SSA形式

[1] 微分と連続最適化

- 1.1 微分のおさらい
- 1.2 勾配降下法
- 1.3 勾配降下法と機械学習

[2] 自動で微分

- 2.1 微分の近似—数値微分
- 2.2 誤差なしの微分 —数式微分
- 2.3 式の微分からアルゴリズムの微分へ
- 2.4 自動微分とトレース
- 2.5 自動微分とソースコード変換

[3] Juliaに微分させる

- 3.1 FiniteDiff.jl/FiniteDifferences.jl
- 3.1 ForwardDiff.jl
- 3.2 Zygote.jl/Diffractor.jl
- 3.3 AbstractDifferentiation.jl

[4] Juliaの微分を拡張する

- 4.1 ChainRules

[5] まとめ

[6] 参考になる文献

[1] 微分と連続最適化

1.1 微分のおさらい

1.2 勾配降下法

1.3 勾配降下法と機械学習

微分の定義 from 高校数学

[定義1. 微分係数]

関数 f の x における微分係数は

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

●
偏微分の定義 from 大学数学

x_i について偏微分 $\leftrightarrow x_i$ 以外の変数を固定して微分

[定義2. 偏微分係数]

n 変数関数 f の (x_1, \dots, x_n) の x_i に関する偏微分係数

$$\frac{\partial f}{\partial x_i}(x_1, \dots, x_n) := \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_{i-1}, x_i + h, x_{i+1}, \dots, x_n) - f(x_1, \dots, x_n)}{h}$$

各 x_i について偏微分係数を計算して並べたベクトルを **勾配ベクトル** と呼ぶ

$$\nabla f(x_1, \dots, x_n) := \left(\frac{\partial f}{\partial x_1}(x_1, \dots, x_n), \dots, \frac{\partial f}{\partial x_n}(x_1, \dots, x_n) \right)$$

例) $f(x, y) = x^2 + xy$ の $(1, 2)$ における勾配ベクトルは

$$\begin{cases} \frac{\partial f}{\partial x} = 2x + y \\ \frac{\partial f}{\partial y} = x \end{cases} \Rightarrow \underline{\nabla f(1, 2) = (4, 1)}$$

✓ 勾配ベクトルの重要ポイント

勾配ベクトルは関数の値が最も大きくなる方向を指し示す

[定理1. 勾配ベクトルの性質]

$-\nabla f(\boldsymbol{x})$ は

$$g(\boldsymbol{v}) = \lim_{h \rightarrow 0} \frac{f(\boldsymbol{x}) - f(\boldsymbol{x} + h\boldsymbol{v})}{h}$$

の最大値を与える。

($f(x, y) = x^2y + \frac{1}{x^2+y^2+1}$ のプロット)

実例で見てみる

$$f(x, y) = x^2y + \frac{1}{x^2+y^2+1}$$

計算すると、

$$\begin{cases} \frac{\partial f}{\partial x} = 2xy - \frac{2x}{(x^2 + y^2 + 1)^2} \\ \frac{\partial f}{\partial y} = x^2 - \frac{2y}{(x^2 + y^2 + 1)^2} \end{cases}$$

なので

$$\nabla f(0.5, -0.5) = \left(\frac{-17}{18}, \frac{25}{36} \right) = (-0.94, 0.694)$$

実例で見る

.....

勾配降下法

✓ 勾配ベクトルは関数の値が大きくなる方向を指し示す

⇒ $-\nabla f(x, y)$ の方向にちょっとずつ点を動かしていけば関数のそこそこ小さい値を取る点を探しに行ける

[最急降下法]

1. $\boldsymbol{x}^{(0)}, \alpha$ を適当に決める
2. $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha \nabla f(\boldsymbol{x}^{(k)})$ として $\boldsymbol{x}^{(k+1)}$ を更新する
3. 収束したと思ったら $\boldsymbol{x}^{(k)}$ を出力して終了. そうでなければ 2. に戻る

✓ 一般の f について大域的な解を求められる保証はないが、
そこそこ小さい値を取る点を探しに行ける

機械学習の典型的な問題設定

学習データ $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ が与えられたとき、

損失関数

$$L(\boldsymbol{w}; \mathcal{D})$$

をなるべく小さくする \boldsymbol{w} を求めよ

勾配降下法と深層学習

勾配降下法を使った深層学習モデル
のパラメータの最適化は、
実際やってみると非常に上手くいく

⇒ **今この瞬間も世界中の計算機が
せっせと勾配ベクトルを計算中**

勾配降下法と深層学習 (※ ギャグです)

- 2050年にはAI業務サーバの消費電力は 3000 Twh にのぼると予測されている [1]
- 日産リーフは 7.0 km/kWh で走るらしい

勾配降下法と深層学習 (※ ギャグです)

太陽 ~ 地球の距離は 1.5×10^8 km くらい.

$$(2.1 \times 10^{16}) / (1.5 \times 10^8) = 1.4 \times 10^5$$

⇒ 人類は、一年間で日産リーフを太陽に140000 台送りこめる電力を勾配の計算に費やしている。

学習データ $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ が与えられたとき、

損失関数

$$L(\boldsymbol{w}; \mathcal{D})$$

をなるべく小さくする \boldsymbol{w} を求めよ

勾配降下法で解くには...

∇L を使って

\boldsymbol{w} を更新していけば良い

学習データ $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ が与えられたとき、
損失関数

$$L(\boldsymbol{w}; \mathcal{D})$$

をなるべく小さくする \boldsymbol{w} を求めよ

勾配降下法で解くには...

∇L を使って

\boldsymbol{w} を更新していけば良い

勾配の計算法を考える

.....

さっきは $f(x, y) = x^2y + \frac{1}{x^2+y^2+1}$

⇒ 頑張って手で ∇f を求められた

深層学習の複雑なモデル...

$L(w; x, y) = \text{VeryComplicated}f(w; x, y)$



 < やりますよ

✓ 計算機に自動で微分させよう！

[2] 自動微分

2.1 微分の近似—数値微分

2.2 誤差なしの微分 —数式微分

2.3 式の微分からアルゴリズムの微分へ

2.4 自動微分とトレース

2.5 自動微分とソースコード変換

2.1 微分の近似—数値微分

```
function numerical_derivative(f::Function, x::Number)::Number
    g = numerical_operation(f, x)
    return g
end
```


微分の定義

$$\lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

をそのまま近似する

```
function numerical_derivative(f, x; h=1e-8)
    g = (f(x+h) - f(x)) / h
    return g
end

f(x) = sin(x)
f'(x) = cos(x)
x = π / 3

numerical_derivative(f, x) # 0.4999999969612645
f'(x)                      # 0.500000000000000001
```

+ 数値微分のメリット

✓ 実装が極めて容易

✓ f がなんでも計算可能.

数値微分のメリット ~ 実装が容易

これだけで完了

```
function numerical_derivative(f, x; h=1e-8)
    g = (f(x+h) - f(x)) / h
    return g
end

f(x) = sin(x)
f'(x) = cos(x)
x = π / 3

numerical_derivative(f, x) # 0.4999999969612645
f'(x)                     # 0.500000000000000001
```

数値微分のメリット ~ 実装が容易

多変数関数への拡張 $\dots i$ 番目を固定して繰り返し計算

```
function numerical_gradient(f, x::Vector; h=1e-8)
    n = length(x)
    g = zeros(n)
    y = f(x...)
    for i in 1:n
        x[i] += h
        g[i] = (f(x...) - y) / h
        x[i] -= h
    end
    return g
end
```

✓ 実装完了

数値微分のメリット ~ なんでも計算可能

```
function numerical_derivative(f, x; h=1e-8)
    g = (f(x+h) - f(x)) / h
    return g
end
```

f がどんなに訳のわからない演算でも
 $f(x+h)$ さえ $f(x)$ が計算できればOK

数値微分のデメリット

1. 打ち切り誤差が生じる
2. 桁落ちも起こる
3. 計算コストが高い

数値微分の誤差 ~ 打ち切り誤差

本来は極限を取るのに小さい値で
誤魔化すので誤差が発生



実際どれくらいの誤差が発生する？

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

[定理2. 数値微分の誤差]

$$f'(x) - \frac{f(x+h) - f(x)}{h} = O(h)$$

[証明]

テイラー展開すると、

$$\begin{aligned} f'(x) - \frac{1}{h}(f(x+h) - f(x)) &= f'(x) - \frac{1}{h}(f(x) + f'(x)h + O(h^2) - f(x)) \\ &= O(h) \end{aligned}$$

実験:

$O(h)$ なら、 h をどんどん小さくすればいくらでも精度が良くなるはず？

```
H = [0.1^i for i in 4:0.5:10]
E = similar(H)

for i in eachindex(H)
    d = numerical_derivative(f, x, h=H[i])
    E[i] = abs(d - f'(x))
end

plot(H, E)
```

誤差の最小化

実際

$h < 10^{-8}$ くらいになるとむしろ
精度が悪化する

丸め誤差と打ち切り誤差のトレードオフ

h が小さくなると、分子の引き算が非常に近い値の引き算になる

⇒ 桁落ちが発生し全体として悪化

$$\frac{f(x + h) - f(x)}{h}$$

誤差への対応

1. 打ち切り誤差 ⇨ 計算式の変更
2. 桁落ち ⇨ h の調整？

数値微分の改良 ~ 打ち切り誤差の改善

1. 打ち切り誤差への対応

微分の (一般的な) 定義をそのまま計算する方法:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

は **前進差分** と呼ばれる

数値微分の改良 ~ 打ち切り誤差の改善

ところで、

$$\lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x-h)}{2h}$$



これを近似してみても良さそう？

中心差分による 2 次精度の数値微分

実はこれの方が精度がよい！

[定理3. 中心差分の誤差]

$$f'(x) - \frac{f(x+h) - f(x-h)}{2h} = O(h^2)$$

同じようにテイラー展開をするとわかる

また、簡単な計算で一般の n について誤差 $O(h^n)$ の近似式を得られる (下参照)

中心差分と同様に x から左右に $\frac{n}{2}$ ずつとってこれの評価の重みつき和を考えてみます。

すると、テイラー展開の各項を足し合わせて $f'(x)$ 以外の係数を 0 にすることを考えることで公比が各列 $-\frac{n}{2}, -\frac{n-1}{2}, \dots, \frac{n}{2}$ で初項 1 のヴァンデルモンド行列を A として $Ax = e_2$ を満たす x を h で割ったのが求めたい重みとわかります。あとはこれの重み付き和をとればいいです。同様に k 階微分の近似式も得られます。

数値微分の改良 ~ 桁落ちへの対応

2. 桁落ちへの対応

Q. 打ち切り誤差と丸め誤差のトレードオフで h を小さくすればいいというものじゃないことはわかった。じゃあ、最適な h は見積もれる？

A. 最適な h は f の n 階微分の大きさに依存するから簡単ではない。

例) 中心差分 $\frac{f(x+h) - f(x-h)}{h}$ は $h_{best} \approx \sqrt[3]{\frac{3\sqrt{2}\varepsilon}{|f'''(x)|}}$ くらい？

⇒ $f'(x)$ がわからないのに $f'''(x)$ を使った式を使うのは現実的でない。

しょうがないので $h = \left(\frac{(n+1)!}{\sqrt{n}f(x)} \varepsilon \right)^{\frac{1}{n+1}}$ に線を引いてみると...

結構いい感じ？



デメリット3. 計算コストが高い

$f : \mathbb{R}^n \rightarrow \mathbb{R}$ の x における勾配ベクトル $\nabla f(x)$ を求める

```
function numerical_gradient(f, x::Vector; h=1e-8)
    n = length(x)
    g = zeros(n)
    y = f(x...)
    for i in 1:n
        x[i] += h
        g[i] = (f(x...) - y) / h
        x[i] -= h
    end
    return g
end
```

⇒ 関数を n 回評価する必要がある.

n 回評価は致命的

✅ 応用では f が重く, n が大きくなりがち $\Rightarrow n$ 回評価は高コスト

自動微分の勉強で参考になる文献

1. 久保田光一, 伊里正夫 「アルゴリズムの自動微分と応用」 コロナ社 (1998)
 - i. 自動微分そのものについて扱ったおそらく唯一の和書です. 詳しいです.
 - ii. 形式的な定義から、計算グラフの縮小のアルゴリズムや実装例と基礎から実用まで触れられています.
 - iii. サンプルコードは、FORTRAN, (昔の) C++ です. 😊
2. 斉藤康毅 「ゼロから作るDeep Learning ③」 O'Reilly Japan (2020)
 - i. トレースベースの Reverse AD を Python で実装します.
 - ii. Step by step で丁寧に進んでいくので、とてもおすすめです.
 - iii. 自動微分自体について扱った本ではないため、その辺りの説明は若干手薄かもしれません.
3. Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2015). Automatic differentiation in machine learning: A survey. ArXiv. /abs/1502.05767
 - i. 機械学習 x AD のサーベイですが、機械学習に限らず AD の歴史やトピックを広く取り上げてます.
 - ii. 少し内容が古くなっているかもしれません.
4. [Differentiation for Hackers](#)
 - i. Flux.jl や Zygote.jl の開発をしている Mike J Innes さんが書いた自動微分の解説です. Juliaで動かしながら勉強できます. おすすめです.
5. Innes, M. (2018). Don't Unroll Adjoint: Differentiating SSA-Form Programs. ArXiv. /abs/1810.07951
 - i. Zygote.jl の論文です. かなりわかりやすいです.
6. Gebremedhin, A. H., & Walther, A. (2019). An introduction to algorithmic differentiation. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10(1), e1334.
<https://doi.org/10.1002/widm.1334>
 - i. 実装のパラダイムやCheckpoint, 並列化などかなり広く触れられています
7. [Zygote.jl のドキュメントの用語集](#)
 - i. 自動微分は必要になった応用の人がやったり、コンパイラの人やったり、数学の人がやったりで用語が乱立しまくっているのでこちらを参照して整理すると良いです
 - ii. 僕の知る限り、(若干のニュアンスがあるかもしれませんが) Reverse AD の別表現として以下があります.
Backward Mode AD = Reverse Mode AD = Fast Differentiation = Adjoint Differentiation + その訳語たち, 微妙な表記揺れたち
8. [JuliaDiff](#)
 - i. Julia での微分についてまとまっています.
9. [Chainer のソースコード](#)
 - i. Chainer は Python製の深層学習フレームワークですが、既存の巨大フレームワークと比較すると、裏も Pythonでとても読みやすいです.
 - ii. 気になる実装があったら当たるのがおすすめです. 議論もたくさん残っているのでそれを巡回するだけでとても勉強になります.