Higher-order automatic differentiation

Contents

1. Overview

Sometimes we want to have more than 1 order differentiation. An example is quadratic programming. However, it is very hard to find literature that can handle more than 1 order differentiation.

2. Manual calculation of higher order chain rules

It is possible to calculate higher order chain rules by hand, if the order is finite, but it is notoriously error prone. For example, we already use the chain rule of product for first order derivatives:

$$\frac{d}{dx}fg = f\frac{dg}{dx} + \frac{df}{dx}g \tag{1}$$

We can apply the differential and apply chain rule again to obtain:

$$\begin{aligned}\frac{d^2}{dx^2}fg &= \frac{d}{dx}\left(f\frac{dg}{dx} + \frac{df}{dx}g\right) \\ &= \frac{d^2f}{dx^2}g + 2\frac{df}{dx}\frac{dg}{dx} + f\frac{d^2g}{dx^2}\end{aligned} \tag{2}$$

and so on.

You could encode this rule in a Rust code like below[1]. Note that we use generics for the value type, and it may not be `Copy`, so they are cloned when they are used more than once.

```
Mul(lhs, rhs) => {
    let dlhs = derive(nodes, lhs, wrt)?;
    let drhs = derive(nodes, rhs, wrt)?;
    let d2lhs = derive2(nodes, lhs, wrt)?;
    let d2rhs = derive2(nodes, rhs, wrt)?;
    let vrhs = value(nodes, rhs)?;
    let vlhs = value(nodes, lhs)?;
    let cross = dlhs * drhs;
    d2lhs * vrhs + vlhs * d2rhs + cross.clone() + cross
}
```

We could go as far as Figure 1 with this approach. Here we differentiate a Gaussian function up to the second order differentiation.

[1] https://github.com/msakuta/rustograd/blob/c902c7066c19f6df336a2cc1af85e4f430bce99a/src/tape.rs#L364

$$
\begin{aligned}
f(x) &= \exp(-x^2) \\
\frac{df}{dx} &= -2x\exp(-x^2) \\
\frac{d^2f}{dx^2} &= (-2+4x)\exp(-x^2)
\end{aligned}
\tag{3}
$$

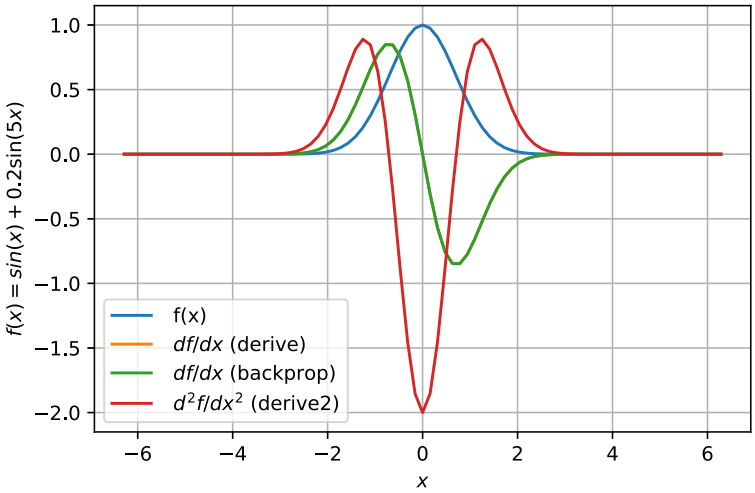

Figure 1: A Gaussian distribution and its up to the second order differentiations

But it has many obvious problems. It is not scalable, it only works for forward mode differentiation, it can't work for multivariate functions and it feels like not automatic at all.

3. Higher Order Automatic Differentiation with Dual Numbers

Another approach that I found in a paper is to use dual numbers, but more than just 2.

To recap, a dual number is something similar to a complex number, having 2 components like $x + y\varepsilon$. This ε is kind of an imaginary number that represents infinitesimal amount of change. Being an infinitesimal number, the dual number will be zero if squared: $\varepsilon^2 = 0$. Using dual numbers, you can derive product rule, for example:

$$
(f + f'\varepsilon)(g + g'\varepsilon) = fg + (fg' + gf')\varepsilon + \varepsilon^2 \tag{4}
$$

and you can bring ε^2 to zero to obtain

$$
fg' + gf' \tag{5}
$$

which corresponds to

$$
d(fg) = fdg + gdf \tag{6}
$$

The dual number is just a mathematical symbol to describe the definition of derivatives, that is,

$$
f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon} \tag{7}
$$

You could use extended dual to compute higher order differentials by having another dual number $x + a\varepsilon + b\eta$. The calculation should yield the same result as the manually calculated chain rules.

It corresponds to applying the definition of derivatives twice.

$$f''(x) = \lim_{\eta \to 0} \frac{\lim_{\varepsilon \to 0} \frac{f(x+\varepsilon+\eta)-f(x+\eta)}{\varepsilon} - f(x)}{\eta} \tag{8}$$

For example, you can derive the product rule by 3-dual numbers:

$$\begin{gathered}(x + a\varepsilon + m\eta)(y + b\varepsilon + n\eta) = \\ xy + (ya + xb)\varepsilon + (ym + xn)\eta + (yam + xbn)\varepsilon\eta + ab\varepsilon^2 + mn\eta^2\end{gathered} \tag{9}$$

Now, ε and η represent different infinitesimal value (see Equation 8). So they have asymmetric rules of products.

- $\varepsilon^2 = 2\eta$ (corresponds to $\frac{d}{dx}\frac{d}{dx}$)
- $\varepsilon\eta = 0$ (corresponds to the third order differentiation, which is ignored)
- $\eta^2 = 0$ (corresponds to even higher order differentiations)

So the final expression will be:

$$xy + (ya + xb)\varepsilon + (ym + xn + 2ab)\eta \tag{10}$$

Here, the coefficients of the η will be the second order differentiation, because the first and the second term will be subtracted by the definition of differentiation Equation 8.

$$ym + xn + 2ab \tag{11}$$

Now, if you look at it carefully, it corresponds to Equation 2.

Another way to look at it that was presented by the paper is to use matrices like below, although they are rarely used in production, because they are too redundant to represent on memory.

$$1 = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}, \varepsilon = \begin{pmatrix}0 & 2 & 0\\ 0 & 0 & 2\\ 0 & 0 & 0\end{pmatrix}, \eta = \begin{pmatrix}0 & 0 & 2\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix} \tag{12}$$

The paper generalizes the method to arbitrary order of differentiation as follows. First, we represent the i-th dual number as $\boldsymbol{i}_j$. Then we can write arbitrary function with linear combination of $\boldsymbol{i}_j$ like below.

$$\mathcal{D}(f) = \sum_{j=0}^{N} f^{(j)} \boldsymbol{i}_j \tag{13}$$

Now, addition and subtraction are easy:

$$\mathcal{D}(f_1 \pm f_2) = \sum_{j=0}^{N} \left(f_i^{(j)} \pm f_2^{(j)}\right) \boldsymbol{i}_j = \mathcal{D}(f_1) \pm \mathcal{D}(f_2) \tag{14}$$

The multiplication is more complicated, but you can obtain:

$$\mathcal{D}(f_1 f_2) = \sum_{j=0}^{N} \sum_{k=0}^{N-j} f_1^{(j)} f_2^{(k)} \binom{j+k}{k} \boldsymbol{i}_{j+k} \tag{15}$$

Encoding these rules in Rust code is a bit of hussle, but we can define a type `Dvec` that can handle arbitrary order of differentiations as you can find in the repository[2]. You can see that the result Figure 2 agrees with the manual differentiation Figure 1.

[2] https://github.com/msakuta/rustograd/blob/dnum/src/dvec.rs

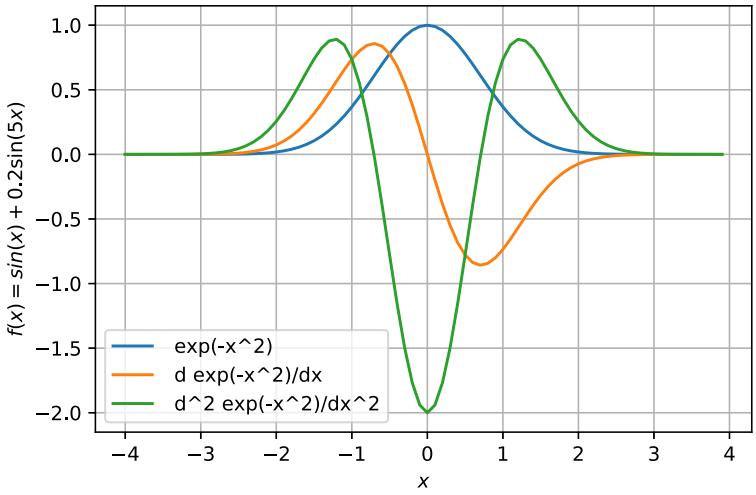

Figure 2: A Gaussian distribution differentiations using Dvec

4. Generating subgraphs

We can generate subgraphs from existing nodes. In this way, we can support higher order differentiation or even a function that includes derivatives.

$$f(x) = g(x) + \frac{dh(x)}{dx} \tag{16}$$

5. Differentiation of $\arctan(y, x)$

To compute differentiation of $\arctan(y, x)$,

$$\begin{aligned} f(x) &= \arctan(x) \\ \frac{\mathrm{d}f(x)}{\mathrm{d}x} &= \left(\frac{\mathrm{d}f^{-1}(x)}{\mathrm{d}x}\right)^{-1} \\ &= \left(\frac{\mathrm{d}}{\mathrm{d}x}\tan(x)\right)^{-1} = \left(\frac{\mathrm{d}}{\mathrm{d}x}\frac{\sin(x)}{\cos(x)}\right)^{-1} \\ &= \left(\frac{\cos(x)}{\cos(x)} - \frac{\sin(x)}{\cos^2(x)}\right)^{-1} = \left(\frac{\cos^2(x) - \sin(x)}{\cos^2(x)}\right)^{-1} \\ &= \frac{\cos^2(x)}{\cos^2(x) - \sin(x)} \end{aligned} \tag{17}$$

6. Differentiation of Matrix Products

Let's say we have matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times l}$, both potentially a function of some variable. How do we calculate the derivative $d(AB)$?

First, let's write the total derivative.

$$d\{AB_{ij}\} = d\sum_{k=1}^{m} a_{ik}\ b_{kj} = \sum_{k=1}^{m}\left(da_{ik}\ b_{kj} + a_{ik}\ db_{kj}\right)\ (i \in [1, n], j \in [1, l]) \tag{18}$$

Let's consider the case of derivative with respect to a_{pq}. Note that this is a 4th order tensor with indices i, j, p and q. It's messy, but we will simplify quite a bit later.

$$\frac{\partial\{AB_{ij}\}}{\partial a_{pq}} = \frac{\sum_k b_{kj}\ \partial a_{ik}}{\partial a_{pq}} = b_{qj}\delta_{ip} \tag{19}$$

Similarly, we can compute

$$\frac{\partial\{AB_{ij}\}}{\partial b_{pq}} = \frac{\sum_k a_{ik}\ \partial b_{kj}}{\partial b_{pq}} = a_{ip}\delta_{jq} \tag{20}$$

The question is, how much contribution does each input variable has to the output variable. We would have to accumulate the influence through every path that goes through the product.

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{l}\frac{\partial\{AB_{ij}\}}{\partial a_{pq}} &= \sum_j b_{qj} \\ \sum_{i=1}^{n}\sum_{j=1}^{l}\frac{\partial\{AB_{ij}\}}{\partial b_{pq}} &= \sum_i a_{ip}\end{aligned} \tag{21}$$

Actually, the story is a bit more complicated, because the matrix product is often an intermediate step in the chain of computations. When we backpropagate, we have a term like below by the product rule.

$$f_{ij}\frac{\partial\{AB_{ij}\}}{\partial a_{pq}} \tag{22}$$

where f_{ij} is some arbitrary function.

Therefore, Equation 21 would be written like below, which is really dot products.

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=1}^{l} f_{ij}\frac{\partial\{AB_{ij}\}}{\partial a_{pq}} &= \sum_j f_{ij}b_{qj} = \boldsymbol{f}_{[i,:]}\boldsymbol{b}^T_{[q,:]} = FB^T \\ \sum_{i=1}^{n}\sum_{j=1}^{l} f_{ij}\frac{\partial\{AB_{ij}\}}{\partial b_{pq}} &= \sum_i f_{ij}a_{ip} = \boldsymbol{f}^T_{[:,j]}\boldsymbol{a}_{[:,p]} = F^T A\end{aligned} \tag{23}$$

Here, we use notation $\boldsymbol{b}_{[q,:]}$ to indicate the q-th row vector of B and $\boldsymbol{a}_{[:,p]}$ to indicate the p-th column vector of A, similar to numpy or PyTorch notation, because I don't know any better way of representing it…

Anyway, it ends up with simple enough matrix multiplications.

7. Literature

There are many researches on this topic, but they are much less than the first order differentiation, and they tend to be mathematically abstract and hard to apply. These are the only few papers that I found useful (in the sense that is understandable and implementable by me)

https://kenndanielso.github.io/mlrefined/blog_posts/3_Automatic_differentiation/3_5_higher_order.html

https://pp.bme.hu/eecs/article/download/16341/8918/87511