Forward mode and directional derivatives

William de Vazelhes

MBZUAI

March 2, 2022

We will study those two papers:

- Gradients without Backropagation: [Baydin et al.(2022)Baydin, Pearlmutter, Syme, Wood, and Torr]
- Learning by directional gradient descent: [Silver et al.(2021)Silver, Goyal, Danihelka, Hessel, and van Hasselt]

Backpropagation

Most deep learning pipelines use backpropagation.
Example:

$$y = f(g(h(x)))$$

Need to process the whole chain (forward pass), then backpropagate through it (backward pass).
Chain can be very long (e.g. recurrent neural networks). We need to keep all the activations in memory $(h(x), g(h(x)), f(g(h(x))))$

Solution: directional derivative

We can try to estimate the gradient with an **exact** directional derivative:

$$\hat{\nabla}_{exact} f(x) = \langle \nabla f(x), u \rangle u$$

Can be computed efficiently using forward mode automatic differentiation (see next slides): just need (≈ 3) forward pass, and past activations can be forgotten.

Note: Similar to Zeroth-order, but exact

$$\hat{\nabla}_{FD} f(x) = \frac{f(x+\epsilon u) - f(x)}{\epsilon} u \approx \langle \nabla f(x), u \rangle u$$

(Note: 2 forward pass)
But it is biased:

$$\mathbb{E}_u \hat{\nabla}_{FD} f(x) = \nabla f_\epsilon(x) \neq \nabla f(x)$$

When the exact directional derivative is unbiased:

$$\mathbb{E}_u \nabla_{\text{exact}} f(x) = \nabla f(x)$$

Question

How can we compute the exact directional derivative ?

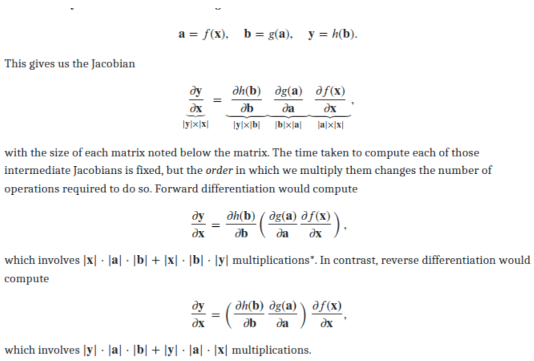
$$\hat{\nabla}_{exact} f(x) = \langle \nabla f(x), u \rangle u$$

$\rightarrow$ use the JVP: Jacobian Vector Product (Ju (gradient is a 1-row Jacobian))

$$J_F(M) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}.$$

(JVP: efficient implementation existing in AD libraries: JAX, pytorch, tensorflow...)

How is it implemented? Forward-mode vs Backwards mode



$$\mathbf{a} = f(\mathbf{x}), \quad \mathbf{b} = g(\mathbf{a}), \quad \mathbf{y} = h(\mathbf{b}).$$

This gives us the Jacobian

$$\underbrace{\frac{\partial \mathbf{y}}{\partial \mathbf{x}}}_{|\mathbf{y}|\times|\mathbf{x}|} = \underbrace{\frac{\partial h(\mathbf{b})}{\partial \mathbf{b}}}_{|\mathbf{y}|\times|\mathbf{b}|} \underbrace{\frac{\partial g(\mathbf{a})}{\partial \mathbf{a}}}_{|\mathbf{b}|\times|\mathbf{a}|} \underbrace{\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}}_{|\mathbf{a}|\times|\mathbf{x}|},$$

with the size of each matrix noted below the matrix. The time taken to compute each of those intermediate Jacobians is fixed, but the *order* in which we multiply them changes the number of operations required to do so. Forward differentiation would compute

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial h(\mathbf{b})}{\partial \mathbf{b}} \left(\frac{\partial g(\mathbf{a})}{\partial \mathbf{a}} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right),$$

which involves $|\mathbf{x}| \cdot |\mathbf{a}| \cdot |\mathbf{b}| + |\mathbf{x}| \cdot |\mathbf{b}| \cdot |\mathbf{y}|$ multiplications*. In contrast, reverse differentiation would compute

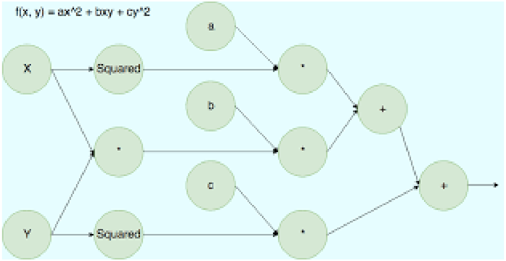
$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left(\frac{\partial h(\mathbf{b})}{\partial \mathbf{b}} \frac{\partial g(\mathbf{a})}{\partial \mathbf{a}} \right) \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}},$$

which involves $|\mathbf{y}| \cdot |\mathbf{a}| \cdot |\mathbf{b}| + |\mathbf{y}| \cdot |\mathbf{a}| \cdot |\mathbf{x}|$ multiplications.

- BM:
  - need to compute the whole graph, **then** backpropagate through the whole graph
  - good for $f : \mathbb{R}^n \to \mathbb{R}$
- FM:
  - memory saving: forward pass with forgetting.
  - Good for $f : \mathbb{R} \to \mathbb{R}^n$ (**But for jvp: efficient even for** $f : \mathbb{R}^n \to \mathbb{R}$: because dot product with u on the right)

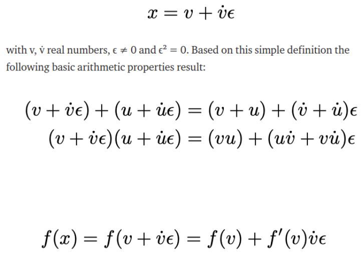
Computational graph

In practice, libraries make use of the computational graph :



Computational graph: Forward mode using **dual numbers**

Similar to complex numbers ($a + bi, i^2 = -1$), we consider **dual numbers** ($v + \dot{v}\epsilon, \epsilon^2 = 0$)



$$x = v + \dot{v}\epsilon$$

with v, $\dot{v}$ real numbers, $\epsilon \neq 0$ and $\epsilon^2 = 0$. Based on this simple definition the following basic arithmetic properties result:

$$(v + \dot{v}\epsilon) + (u + \dot{u}\epsilon) = (v + u) + (\dot{v} + \dot{u})\epsilon$$
$$(v + \dot{v}\epsilon)(u + \dot{u}\epsilon) = (vu) + (u\dot{v} + v\dot{u})\epsilon$$

$$f(x) = f(v + \dot{v}\epsilon) = f(v) + f'(v)\dot{v}\epsilon$$

$\rightarrow$ We start with say $\mathbf{z} + \epsilon\mathbf{u}$. Then, at each node n, we compute: $n(v)$, ($n'(v)$), and then $n'(v)\dot{v}$ (so 3 computations). Then for merging (addition, multiplication) we use the rules above. We keep everytime, separately, both the "v" and the "$\dot{v}$". The final "$\dot{v}$" is the JVP $J(z)\mathbf{u}$.

Real-life view in libraries (continued)

So it is convenient to implement: for each node (elementary function), just need to code it as a “dual” node:

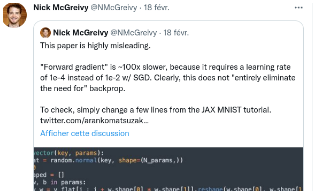
$$y + \dot{y}\epsilon = n(x + \dot{x}\epsilon)$$

$$(y, \dot{y}) = n(x, \dot{x})$$

using the rules above

Remarks on Twitter

- **High variance/Curse of dim.** (if 1 random direction)



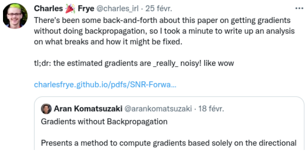
Nick McGreivy @NMcGreivy · 18 févr.

Nick McGreivy @NMcGreivy · 18 févr.
This paper is highly misleading.

"Forward gradient" is ~100x slower, because it requires a learning rate of 1e-4 instead of 1e-2 w/ SGD. Clearly, this does not "entirely eliminate the need for" backprop.

To check, simply change a few lines from the JAX MNIST tutorial.
twitter.com/arankomatsuzak...
Afficher cette discussion

```
vector(key, params):
at = random.normal(key, shape=(N_params,))
0
aped = []
w, b in params:
```



Charles 🎉 Frye @charles_irl · 25 févr.
There's been some back-and-forth about this paper on getting gradients without doing backpropagation, so I took a minute to write up an analysis on what breaks and how it might be fixed.

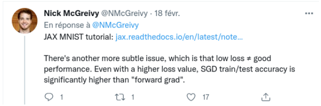
tl;dr: the estimated gradients are _really_ noisy! like wow

charlesfrye.github.io/pdfs/SNR-Forwa...

Aran Komatsuzaki @arankomatsuzaki · 18 févr.
Gradients without Backpropagation

Presents a method to compute gradients based solely on the directional

- **Generalization**



Nick McGreivy @NMcGreivy · 18 févr.
En réponse à @NMcGreivy
JAX MNIST tutorial: jax.readthedocs.io/en/latest/note...

There's another more subtle issue, which is that low loss ≠ good performance. Even with a higher loss value, SGD train/test accuracy is significantly higher than "forward grad".

Remarks on Twitter: what if we use our ZO-HT ?

- **High variance/Curse of dim.** ZO-HT: we keep only k components of x. Preliminary results indicate that we need $O(k)$ instead of $O(d)$ random directions.
- **Generalization** ZO-HT tries to learn a sparsity-constrained solution (maybe related: “Stability and Risk Bounds of Iterative Hard Thresholding” (XT Yuan), “The Lottery Ticket Hypothesis” (Frankle))

Conclusion

An “Exact ZO”: could be useful if we want to:

- have the advantages of ZO (low cost)
- but without the smoothing bias (because of ϵ)

(if we have access to differentiation)

References

https://math.stackexchange.com/questions/2195377/
reverse-mode-differentiation-vs-forward-mode-differentiati
http://www.ams.org/publicoutreach/feature-column/
fc-2017-12 https://blog.demofox.org/2014/12/30/
dual-numbers-automatic-differentiation/
https://towardsdatascience.com/
forward-mode-automatic-differentiation-dual-numbers-8f4735
https://mostafa-samir.github.io/auto-diff-pt1/
https://medium.com/tebs-lab/
deep-neural-networks-as-computational-graphs-867fcaa56c9
https:
//twitter.com/charles_irl/status/1497129714195992578
https://twitter.com/arankomatsuzaki/status/
1494488254304989228

Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr.
Gradients without backpropagation.
arXiv preprint arXiv:2202.08587, 2022.

David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt.
Learning by directional gradient descent.
In *International Conference on Learning Representations*, 2021.