AD Master Class: Advanced Adjoint Techniques

Checkpointing and external functions: Injecting Symbolic Information

Viktor Mosenkis
viktor.mosenkis@nag.co.uk

15 October 2020

AD Masterclass Schedule and Remarks

AD Masterclass Schedule

- 1 October 2020 | Checkpointing and external functions 1
- 15 October 2020 | Checkpointing and external functions 2
- 29 October 2020 | Guest lecture by Prof Uwe Naumann on Advanced AD topics in Machine Learning
- 12 November 2020 | Monte Carlo
- 19 November 2020 | Guest lecture by Prof Uwe Naumann on Adjoint Code Design Patterns applied to Monte Carlo
- 25 November 2020 |Computing Hessians

Remarks

- Please submit your questions via the questions panel at any time during this session, these will be addressed at the end.
- A recording of this session, along with the slides will be shared with you in a day or two.

Dialogue

We want this webinar series to be interactive (even though it's hard to do)

- We want your feedback, we want to adapt material to your feedback
- Please feel free to contact us via email to ask questions at any time
- We'd love to reach out offline, discuss what's working, what to spend more time on
- For some orgs, may make sense for us to do a few bespoke sessions

- This is an advanced course
- We assume that you are familiar with the material from the first Masterclass series
- You will get access to the materials from the first Masterclass series via email in a day or two
- Also it is not a pre-requisite we recommend to review the material from the previous series
- We will try to give references to the previous Masterclass series whenever possible

Outcomes

- Learn how to use gaps to inject symbolic information into the tape for
  - solver for system of linear equations
  - root finding
  - unconstrained optimization
- Look at memory management issues in the context of making/filling gaps if code has more than one output
- Checkpointing strategies

Recall

In the previous masterclass we learned how to make and fill gaps in our DAG/tape. This is a very powerful technique that allows us to do many things

- control amount memory used by the tape (checkpointing, previous masterclass)
- introduce handwritten algorithmic adjoints into the code
- use tangent mode for parts of the code
- use finite difference for parts of the code (e.g. to differentiate through routines without available source code)
- use derivative information from third party (e.g. Jacobian calculated on FPGA/GPU, library routines that provide adjoint implementation)
- use symbolic adjoints

Difference between Symbolic and Algorithmic adjoints

For a given function $F : \mathbb{R}^n \to \mathbb{R}^m, \quad \boldsymbol{y} = F(\boldsymbol{x})$ both symbolic and algorithmic adjoints compute

$$\bar{F}(\boldsymbol{x}, \bar{\boldsymbol{y}}) = \bar{\boldsymbol{y}} \cdot F'(\boldsymbol{x}) = F'(\boldsymbol{x})^T \cdot \bar{\boldsymbol{y}} = \bar{\boldsymbol{x}}$$

- **algorithmic adjoint**: differentiates the implementation of the function by differentiating the language intrinsics
- **symbolic adjoint**: differentiates the underlying mathematical function/model.

Symbolic Adjoint: Linear Equation System

Consider system of linear equations $Ax = b$, where $A \in \mathbb{R}^{n \times n}$, $x, b \in \mathbb{R}^n$.

Partial differentiation w.r.t. b yields

$$\frac{\partial A}{\partial b} x + A \frac{\partial x}{\partial b} = \frac{\partial b}{\partial b}$$

and hence as $\frac{\partial b}{\partial b} = I_n$ and $\frac{\partial A}{\partial b} = 0$

$$\frac{\partial x}{\partial b} = A^{-1}$$

Transposing the equation and multiplying both sides with $\bar{x}$ we get

$$\underbrace{(\frac{\partial x}{\partial b})^T \bar{x}}_{=\bar{b}} = A^{-T} \bar{x}$$

Hence $A^{-T} \bar{x} = \bar{b}$.

Symbolic Adjoint: Linear Equation System

Consider system of linear equations $Ax = b$, where $A \in \mathbb{R}^{n \times n}$, $x, b \in \mathbb{R}^n$.

Partial differentiation w.r.t. A yields

$$\frac{\partial A}{\partial A} x + A \frac{\partial x}{\partial A} = \frac{\partial b}{\partial A}$$

and hence as $\frac{\partial A}{\partial A} = I_n$ and $\frac{\partial b}{\partial A} = 0$

$$\frac{\partial x}{\partial A} = -A^{-1} x$$

Transposing the equation and multiplying both sides with $\bar{x}$ we get

$$\underbrace{\left(\frac{\partial x}{\partial A}\right)^T \bar{x}}_{=\bar{A}} = -x^T \underbrace{A^{-T} \bar{x}}_{=\bar{b}}$$

Hence $\bar{A} = -x^T \bar{b}$.

Symbolic Adjoint: Linear Equation System

Summing up the results for the system of linear equations $Ax = b$, where $A \in \mathbb{R}^{n \times n}$, $x, b \in \mathbb{R}^n$

the adjoints $\bar{A}$ and $\bar{b}$ satisfy the following equations

$$A^T \cdot \bar{b} = \bar{x}$$

and

$$\bar{A} = -x^T \cdot \bar{b}.$$

Hence to compute $\bar{A}$ and $\bar{b}$ we first need to solve the system of linear equations to compute $\bar{b}$ and then use this result to compute $\bar{A}$.

Note: Decomposition of A can be reused for the computation of $\bar{b}$.

Symbolic Adjoint: Linear Equation System

Advantages

- reduce computational costs from $O(n^3)$ to $O(n^2)$
- reduce memory requirements of the tape from $O(n^3)$ to $O(n^2)$ (only factorization of A and solution vector must be stored)
- can be computed even without source code (black-box) routines

Disadvantages

- the derivatives of the factorization of A is not computed
- assumes availability of exact primal solution (problematic for iterative solvers)

Symbolic Adjoint: Root Finder

Consider a function $F : \mathbb{R}^{n+p} \to \mathbb{R}^m$, $y = F(x, \lambda)$, yielding a system of nonlinear equations $F(x, \lambda) = 0$ in x.

Differentiating the system of nonlinear equations at the solution x w.r.t. to parameter λ yields

$$\frac{dF}{d\lambda} = \frac{\partial F}{\partial \lambda} + \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial \lambda} = 0.$$

and hence $\frac{\partial x}{\partial \lambda} = -\left(\frac{\partial F}{\partial x}\right)^{-1} \cdot \frac{\partial F}{\partial \lambda}$. Transposing the equation and multiplying with $\bar{x}$ we get

$$\underbrace{\left(\frac{\partial x}{\partial \lambda}\right)^T \cdot \bar{x}}_{=\bar{\lambda}} = -\left(\frac{\partial F}{\partial \lambda}\right)^T \cdot \left(\frac{\partial F}{\partial x}\right)^{-T} \cdot \bar{x}$$

Hence $\bar{\lambda} = -\left(\frac{\partial F}{\partial \lambda}\right)^T \cdot \left(\frac{\partial F}{\partial x}\right)^{-T} \cdot \bar{x}$.

Symbolic Adjoint: Root Finder

Consequently to compute $\bar{\lambda}$ the symbolic adjoint solver needs to solve the linear system

$$(\frac{\partial F}{\partial x})^T \cdot z = -\bar{x}$$

Followed by a single call of the adjoint model of F seeded with the solution of z yielding

$$\bar{\lambda} = (\frac{\partial F}{\partial \lambda})^T \cdot z.$$

Symbolic Adjoint: Root Finder

Advantages

- reduce memory requirements as there is no need to tape the iterations of the root finder
- reduced computational costs, requires only computation of $\frac{\partial F}{\partial x}$, solving linear equation system and one call of $\bar{F}$.
- can be computed even without source code (black-box) routines

Disadvantages

- assumes convergence of the primal solver
- computation of $\frac{\partial F}{\partial x}$ can be expensive
- cannot compute the derivatives w.r.t. starting point x
- user must provide routines to compute $\frac{\partial F}{\partial x}$ and $\bar{F}$

Symbolic Adjoint: Unconstrained Optimization

An unconstrained optimization problem can be regarded as a root finding problem for the first-order optimality condition

$$\frac{\partial F}{\partial x}(x, \lambda) = 0$$

.

With similar arguments as for the root finding problem we obtain that

$$\underbrace{\left(\frac{\partial x}{\partial \lambda}\right)^T \cdot \bar{x}}_{=\bar{\lambda}} = -\left(\frac{\partial^2 F}{\partial \lambda \partial x}\right)^T \cdot \left(\frac{\partial^2 F}{\partial x^2}\right)^{-T} \cdot \bar{x}$$

Hence

$$\bar{\lambda} = -\left(\frac{\partial^2 F}{\partial \lambda \partial x}\right)^T \cdot \left(\frac{\partial^2 F}{\partial x^2}\right)^{-T} \cdot \bar{x}.$$

Symbolic Adjoint: Unconstrained Optimization

Consequently to compute $\bar{\lambda}$ the symbolic adjoint solver needs to solve the linear system

$$(\frac{\partial^2 F}{\partial x^2})^T \cdot z = -\bar{x}$$

Followed by a single call of the second-order adjoint model of F seeded with the solution of z yielding

$$\bar{\lambda} = (\frac{\partial^2 F}{\partial \lambda \partial x})^T \cdot z.$$

Symbolic Adjoint: Unconstrained Optimization

Advantages

- reduce memory requirements as there is no need to tape the iterations of the unconstrained optimizer
- reduced computational costs, requires only computation of $\frac{\partial^2 F}{\partial x^2}$, solving linear equation system and one call of second-order adjoint model of F.
- can be computed even without source code (black-box) routines

Disadvantages

- assumes convergence of the primal solver
- computation of $\frac{\partial^2 F}{\partial x^2}$ can be expensive
- cannot compute the derivatives w.r.t. starting point x
- user must provide routines to compute $\frac{\partial^2 F}{\partial x^2}$ and second-order adjoint model of F.

Making and filling gaps: Memory management

- Typically the checkpoint (gap) data is created during creating the gap and freed once the gap is filled. This approach ensures that no memory leaks are created.

- In case that the tape must be interpreted several times the checkpoint data should persist after the gap is filled till the last tape interpretation

- external memory management is required to ensure correct tape interpretation and avoid memory leaks

Summary Symbolic adjoints

Symbolic adjoints should be an important part of the toolbox of all AD code developers. As they

- can significantly reduce memory requirements for the tape
- can reduce computational costs
- can be used as alternative to checkpointing
- efficient differentiation of black-box routines

Still using symbolic adjoints have drawbacks

- cannot compute all derivatives
- assume convergence (availability of exact primal solution)
- difficult to validate
- deriving the formula can be complicated

Running adjoint code without running out memory

How can I run my adjoint code without running out memory?

- checkpointing (previous Masterclass) allows you to control memory used by the tape
- Checkpointing single function will typically not solve the problem of running out of memory for the adjoint mode
- A (checkpointing) strategy is necessary to avoid running out of memory for big codes

Checkpointing Strategies

- You can afford as many checkpoints as you need
  - distribute checkpoints equidistantly
  - checkpoint each loop iteration (e.g. in time-stepping procedure)
  - small additional costs as the function value for each gap function is executed only twice
- Checkpoints are expensive and you can afford only certain amount of them
  - binomial or multi-level checkpointing schemes should be used
  - tool support through revolve for (pseudo) time-stepping procedures

Checkpointing Strategies and beyond

Successful AD developer should not restrict himself to checkpointing to control memory usage but rather consider different available strategies to control memory usage of the tape and improve performance

- use the full functionality of creating and filling gaps
  - create checkpoints
  - insert symbolic information
  - use handwritten (source transformation based) adjoints
- exploit structure of the code e.g.
  - path wise adjoints in Monte Carlo code (Masterclass 4)
- preaccumulate Jacobians

Summary

In this Masterclass we

- learned how to use gaps to inject symbolic information into the tape for
  - solver for linear equation system
  - root finding
  - unconstrained optimization
- looked at memory management issues in the context of making/filling gaps if code has more than one output
- touched memory control/checkpointing strategies

AD Master Class 3: On Advanced AD topics in Machine Learning

In the next class our guest lecturer Prof. Uwe Naumann will discuss

- AD for Network pruning
- AD for significance analysis
- Outlook on networks moving beyond the simple forms used today and start including more general information about the systems they attempt to model

You will see a survey on your screen after exiting from this session.

We would appreciate your feedback.

We are now moving on the Q&A Session