AD Master Class: Advanced Adjoint Techniques

Checkpointing and external functions: Manipulating the DAG

Viktor Mosenkis
viktor.mosenkis@nag.co.uk

1 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

- How to make and fill a gap
- Difference between make/fill gap and Jacobian preaccumulation
- Use gaps to control memory usage in adjoint mode.

Algorithmic Differentiation

$$F : \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad \boldsymbol{y} = F(\boldsymbol{x})$$

- **Tangent-Linear Model (TLM)** $\dot{F}$ **(forward mode)**
  $\dot{F}(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \underbrace{F'(\boldsymbol{x})}_{\in \mathbf{R}^{m \times n}} \cdot \underbrace{\dot{\boldsymbol{x}}}_{\in \mathbf{R}^n} = \dot{\boldsymbol{y}}$
  - $F'(\boldsymbol{x})$ at $O(n) \cdot Cost(F)$
  - exact derivatives
  - $\frac{Cost(\dot{F})}{Cost(F)} \approx 2$
- **Adjoint Model (ADM)** $\bar{F}$ **(reverse mode)**
  $\bar{F}(\boldsymbol{x}, \bar{\boldsymbol{y}}) = \underbrace{\bar{\boldsymbol{y}}}_{\in \mathbf{R}^m} \cdot \underbrace{F'(\boldsymbol{x})}_{\in \mathbf{R}^{m \times n}} = F'(\boldsymbol{x})^T \cdot \bar{\boldsymbol{y}} = \bar{\boldsymbol{x}}$
  - $F'(\boldsymbol{x})$ at $O(m) \cdot Cost(F)$
  - exact derivatives
  - $\frac{Cost(\bar{F})}{Cost(F)} < 30$

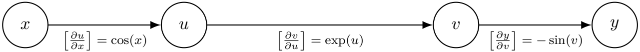
Adjoint Model

- For the adjoint model we need to reverse the control flow of the code
- AD tools record the computational graph of the program (DAG)
- Storing the DAG requires significant amount of memory.
- To run the adjoint model without running out of memory, we need to manipulate the DAG

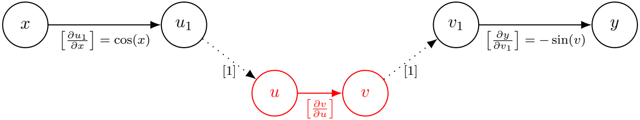
Example: Making/filling gap

$$F(x) = f \circ g \circ h(x) = \cos(\exp(\sin(x)))$$

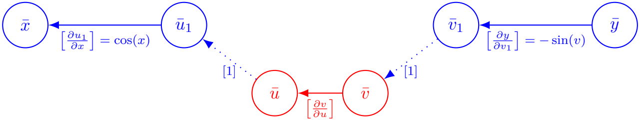
- no gap



- make gap



- fill gap



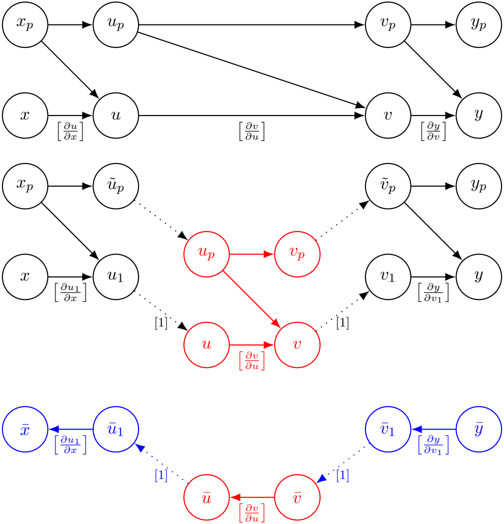
Example: Making/filling gap with AD tool

- So far we used symbolic information to fill the gap (more on this in the next masterclass)

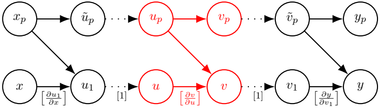
- What if we can't differentiate the function to close the gap

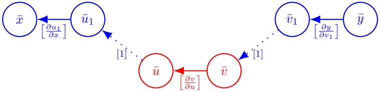
- We can use the AD tool to compute the derivative and use it to fill the gap (we call this checkpointing)

Making/Filling Gap: General Case $(y, y_p) = F(x, x_p)$



Making/Filling Gap: General Case





A gap in the tape is introduced by calling a user-defined function make_gap to record the following gap_data:

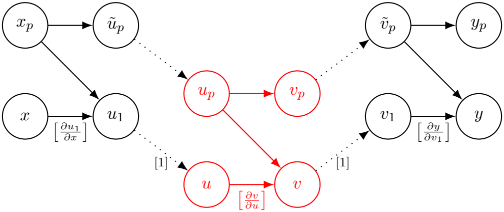
- Tape location of active gap inputs u_1 to write $\bar{u}_1 := \bar{u}$ correctly;
- adjoint gap input checkpoint $\subset (u, u_p, v, v_p)$ in order to initialize interpretation of the gap correctly;
- tape location of active gap outputs v_1 in order to initialize $\bar{v} := \bar{v}_1$ correctly; requires execution of $g(u, u_p, v, v_p)$.

This data is stored in the tape together with a reference to a user-defined function fill_gap to increment $\bar{u}_1$ with $\left(\frac{\partial v}{\partial u}\right)^T \cdot \bar{v}$.

Making Gap: Support by dco/c++

User function make_gap:

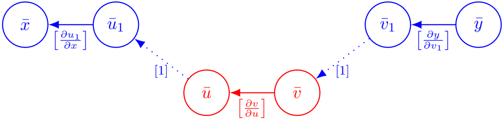
- u :=dco::value(u_1); $u_p := \tilde{u}_p$
- gap_data->write_data(z^-), where $z^- \in (u_1, \tilde{u}_p, \varnothing)$
- $g(u, u_p, v, v_p)$
- dco::value(v_1) $:= v$
- DCO_MODE::global_tape->register_variable(v_1)
- gap_data->write_data(z^+), where $z^+ \in (v_1, v_p, \varnothing)$
- DCO_MODE::global_tape->insert_callback(fill_gap,gap_data)



Filling Gap (General Case): Support by dco/c++

User function `fill_gap`:

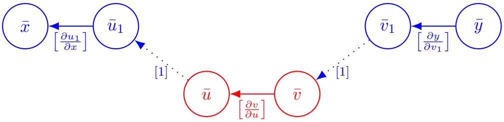
- `gap_data->read_data`(z), where $z \equiv (z^-, z^+)$
- $\bar{v}$:=`dco::derivative`(v_1)
- Compute the adjoint of u with $\bar{g}(z, \bar{u}, \bar{v})$
- `dco::derivative`$(\bar{u}_1) += \bar{u}$



Filling Gap (Checkpointing): Support by dco/c++

User function `fill_gap`:

- `gap_data->read_data`(z), where $z \equiv (z^-, z^+) = (u_1, \tilde{u}_p, v_1)$
- Make tape of $g(u, u_p, v, v_p)$
  - $u = u_1$, $u_p = \tilde{u}_p$
  - run activated version of $g(u, u_p, v, v_p)$
- set $\bar{v}$: `dco::derivative`(v) =`dco::derivative`(v_1)
- Compute $\bar{u}$ by interpreting the tape of $g(u, u_p, v, v_p)$
  - `tape->interpret_adjoint()`
- $\bar{u}_1$ is automatically updated with $\bar{u}$ as $u = u_1$.



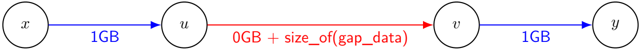
Checkpointing to reduce the required tape size

$$F(x) = f \circ g \circ h(x)$$

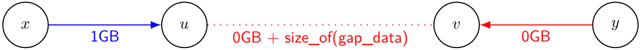
Taping F with

- no checkpoint requires 3GB for the tape

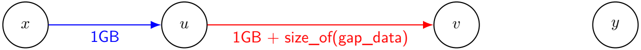
- with checkpointing of g
  - during tape recording: 2GB + size of the gap_data



  - after interpretation of f: 1GB + size of the gap_data



  - after filling the gap: 2GB + size of the gap_data



Checkpointing vs. Jacobian preaccumulation

- Jacobian preaccumulation
  - performed during tape recording
  - computes the Jacobian of the function, thus additional costs if the gap function has more than one output.
- Checkpointing
  - the gap is created during tape recording and filled during interpretation
  - Jacobian is not computed (no additional costs if the gap function has more than one output)
  - the function value of the gap function is computed twice (during make and fill gap)

Checkpointing Strategies

Developing a checkpointing strategy for code is complicated

- 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
- A successful checkpointing strategy depends on many factors such as
  - size of the checkpoint (gap_data)
  - structure of the code
  - alternative ways of filling the gap

We will touch on some ideas for checkpointing strategies in the next Masterclass

Summary

- How to make/fill gap, that can be used for
  - checkpointing
  - symbolic adjoints (Masterclass 2)
  - derivatives of black box routines (Masterclass 2)

- Use make/fill gap for checkpoints (reduce memory requirements)

- Compared Jacobian preaccumulation and checkpointing

AD Master Class 2: Checkpointing and external functions 2

In the next class we will

- Learn how to use gaps to inject symbolic information into the tape for
  - linear algebra
  - root finding
  - unconstrained optimization
- Checkpointing strategies
- Look at some implementation issues in this context that need particular care e.g.
  - external function callbacks
  - memory management if code has more than one output

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