



**HAL**  
open science

# Adjoint computation and Backpropagation

Guillaume Pallez

► **To cite this version:**

Guillaume Pallez. Adjoint computation and Backpropagation. Meeting of the Royal Society – Numerical algorithms for high-performance computational science, Apr 2019, London, United Kingdom. hal-02400746

**HAL Id: hal-02400746**

**<https://inria.hal.science/hal-02400746>**

Submitted on 9 Dec 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# ADJOINT COMPUTATION AND BACKPROPAGATION

Guillaume Pallez (Aupy)  
Inria & University of Bordeaux

Meeting of the Royal Society, April 2019







“In climate modelling, Ice-sheet models use numerical methods to simulate the evolution, dynamics and thermodynamics of ice sheets.” (wikipedia)

### Model Algorithm (single timestep)

1. Evaluate driving stress  $\tau_d = \rho g h \nabla s$

2. Solve for velocities

*DO*  $i = 1, \text{max\_iter}$

i. Evaluate nonlinear viscosity  $\nu_i$  from iterate  $\mathbf{u}_i$

ii. Construct stress matrix  $A\{\nu_i\}$

iii. Solve linear system  $A \mathbf{u}_{i+1} = \tau_d$

iv. (Exit if converged)

*ENDDO*

3. Evolve thickness (continuity eqn)

Automatic differentiation (AD) tools generate code for adjoint of operations

# ICE-SHEET MODEL (I)

“In climate modelling, Ice-sheet models use numerical methods to simulate the evolution, dynamics and thermodynamics of ice sheets.” (wikipedia)

## Model Algorithm (single timestep)

1. Evaluate driving stress  $\tau_d = \rho g h \nabla s$

2. Solve for velocities

DO  $i = 1, \text{max\_iter}$

i. Evaluate nonlinear viscosity  $\nu_i$  from iterate  $\mathbf{u}_i$

ii. Construct stress matrix  $A\{\nu_i\}$

iii. Solve linear system  $A \mathbf{u}_{i+1} = \tau_d$

iv. (Exit if converged)

ENDDO

3. Evolve thickness (continuity eqn)

Automatic differentiation (AD) tools generate code for adjoint of operations

Simpler Version:

proc Model Algorithm( $u_0, \mathbf{y}$ )

begin

Do stuff;

for  $i = 0$  to  $n$  do

$\mathbf{u}_{i+1} = f_i(\mathbf{u}_i)$ ;

    Do stuff;

end

/\*  $F(u_0) = f_n \circ f_{n-1} \circ \dots \circ f_0(u_0)$  \*/

Compute  $\nabla F(u_0) \mathbf{y}$  ;

end

A quick reminder about the gradient:

$$F(u_0) = f_n \circ f_{n-1} \circ \dots \circ f_1 \circ f_0(u_0)$$

$$\nabla F(u_0) \mathbf{y} = Jf_0(u_0)^T \cdot \nabla(f_n \circ f_1)(u_1) \cdot \mathbf{y}$$

$$= Jf_0(u_0)^T \cdot Jf_1(u_1)^T \cdot \dots \cdot Jf_{n-1}(u_{n-1})^T \cdot Jf_n(u_n)^T \cdot \mathbf{y}$$

$$\begin{aligned} Jf^T &= \text{Transpose Jacobian matrix of } f; \\ u_{i+1} &= f_i(u_i) = f_i(f_{i-1} \circ \dots \circ f_0(u_0)). \end{aligned}$$



## A BETTER SOLUTION?

$$\nabla F(u_0)\mathbf{y} = Jf_0(u_0)^T \cdot Jf_1(u_1)^T \cdot \dots \cdot Jf_{n-1}(u_{n-1})^T \cdot Jf_n(u_n)^T \cdot \mathbf{y}$$

```
proc Algo A( $u_0, \mathbf{y}$ )  
begin  
  Do stuff;  
  for  $i = 0$  to  $n$  do  
     $u_{i+1} = f_i(u_i)$ ;  
    Do stuff;  
  end  
  Compute  $\nabla F(u_0)\mathbf{y}$ ;  
end
```

## A BETTER SOLUTION?

$$\nabla F(u_0)\mathbf{y} = Jf_0(u_0)^T \cdot Jf_1(u_1)^T \cdot \dots \cdot Jf_{n-1}(u_{n-1})^T \cdot Jf_n(u_n)^T \cdot \mathbf{y}$$

```
proc Algo A( $u_0, \mathbf{y}$ )  
begin  
  Do stuff;  
  for  $i = 0$  to  $n$  do  
     $u_{i+1} = f_i(u_i)$ ;  
    Do stuff;  
  end  
  Compute  $\nabla F(u_0)\mathbf{y}$ ;  
end
```

→ What is the problem with Algo A?

## A BETTER SOLUTION?

$$\nabla F(u_0)\mathbf{y} = Jf_0(u_0)^T \cdot Jf_1(u_1)^T \cdot \dots \cdot Jf_{n-1}(u_{n-1})^T \cdot Jf_n(u_n)^T \cdot \mathbf{y}$$

```
proc Algo A( $u_0, \mathbf{y}$ )
begin
  Do stuff;
  for  $i = 0$  to  $n$  do
     $u_{i+1} = f_i(u_i)$ ;
    Do stuff;
  end
  Compute  $\nabla F(u_0)\mathbf{y}$ ;
end
```

→ What is the problem with Algo A?

```
proc Algo B( $u_0, \mathbf{y}$ )
begin
  Do stuff;
  for  $i = 0$  to  $n$  do
     $u_{i+1} = f_i(u_i)$ ;
    Do stuff;
     $v_{i+1} = v_i \cdot Jf_{i+1}(u_{i+1})^T$ ;
  end
end
```

## A BETTER SOLUTION?

$$\nabla F(u_0)\mathbf{y} = Jf_0(u_0)^T \cdot Jf_1(u_1)^T \cdot \dots \cdot Jf_{n-1}(u_{n-1})^T \cdot Jf_n(u_n)^T \cdot \mathbf{y}$$

```
proc Algo A( $u_0, \mathbf{y}$ )
begin
  Do stuff;
  for  $i = 0$  to  $n$  do
     $u_{i+1} = f_i(u_i)$ ;
    Do stuff;
  end
  Compute  $\nabla F(u_0)\mathbf{y}$ ;
end
```

→ What is the problem with Algo A?

```
proc Algo B( $u_0, \mathbf{y}$ )
begin
  Do stuff;
  for  $i = 0$  to  $n$  do
     $u_{i+1} = f_i(u_i)$ ;
    Do stuff;
     $v_{i+1} = v_i \cdot Jf_{i+1}(u_{i+1})^T$ ;
  end
end
```

→ What is the problem with Algo B?

# A BETTER SOLUTION?

$$\nabla F(u_0)\mathbf{y} = Jf_0(u_0)^T \cdot Jf_1(u_1)^T \cdot \dots \cdot Jf_{n-1}(u_{n-1})^T \cdot Jf_n(u_n)^T \cdot \mathbf{y}$$

```

proc Algo A(u0, y)
begin
  Do stuff;
  for i = 0 to n do
    | ui+1 = fi(ui);
    | Do stuff;
  end
  Compute ∇F(u0)y;
end

```

→ What is the problem with Algo A?

```

proc Algo B(u0, y)
begin
  Do stuff;
  for i = 0 to n do
    | ui+1 = fi(ui);
    | Do stuff;
    | vi+1 = vi · Jfi+1(ui+1)T;
  end
end

```

→ What is the problem with Algo B?

$$\nabla F(u_0)\mathbf{y} = \left( \left( \dots \left( Jf_0^T \cdot Jf_1^T \right) \dots \cdot Jf_{n-1}^T \right) \cdot Jf_n^T \right) \cdot \mathbf{y} \quad n \text{ MatMat ops}$$

$$\nabla F(u_0)\mathbf{y} = Jf_0^T \cdot \left( Jf_1^T \cdot \dots \cdot \left( Jf_{n-1}^T \cdot \left( Jf_n^T \cdot \mathbf{y} \right) \dots \right) \right) \quad n \text{ MatVec ops}$$

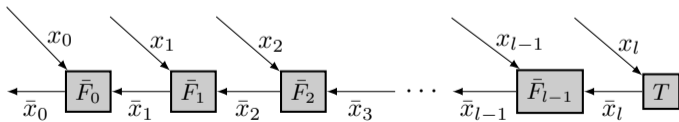
## ADJOINT COMPUTATION

$$\begin{aligned} F_i(x_i) &= x_{i+1} & i < l \\ \bar{F}_i(x_i, \bar{x}_{i+1}) &= \bar{x}_i & i \leq l \end{aligned}$$



# ADJOINT COMPUTATION

$$F_i(x_i) = x_{i+1} \quad i < l$$
$$\bar{F}_i(x_i, \bar{x}_{i+1}) = \bar{x}_i \quad i \leq l$$

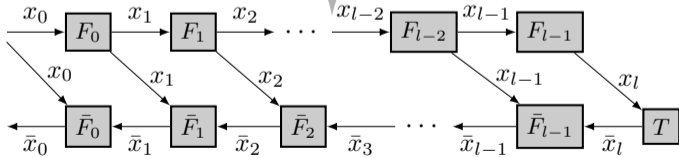




# ADJOINT COMPUTATION

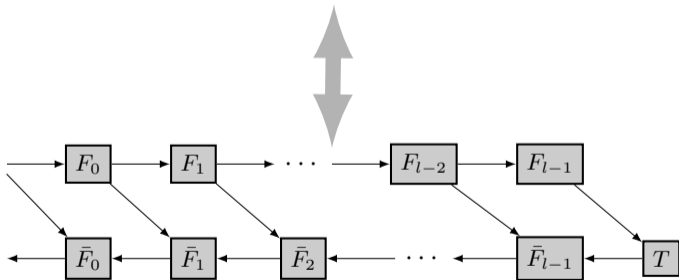
$$F_i(x_i) = x_{i+1} \quad i < l$$

$$\bar{F}_i(x_i, \bar{x}_{i+1}) = \bar{x}_i \quad i \leq l$$



# ADJOINT COMPUTATION

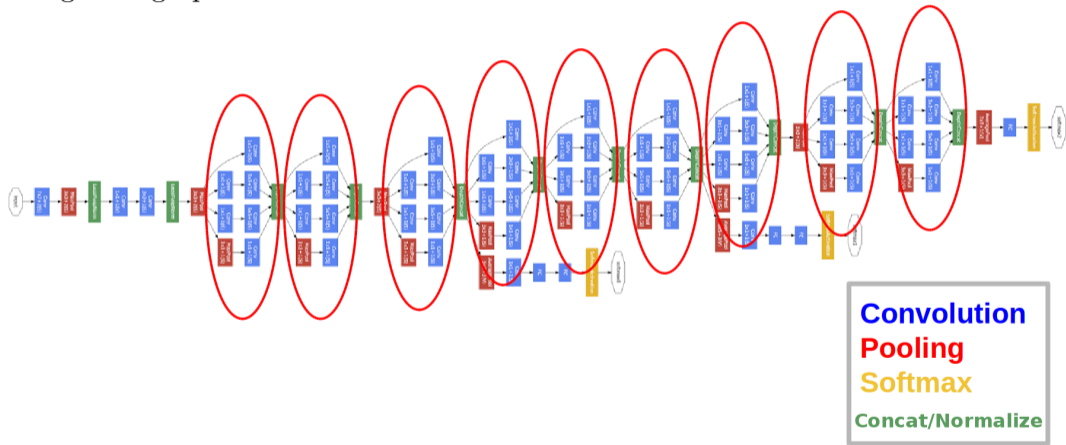
$$F_i(x_i) = x_{i+1} \quad i < l$$
$$\bar{F}_i(x_i, \bar{x}_{i+1}) = \bar{x}_i \quad i \leq l$$





# RELATION TO IA? (I)

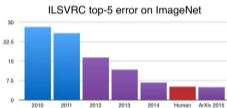
GoogleNet graph:



## Derivatives in machine learning

“Backprop” and gradient descent are at the core of all recent advances

### Computer vision



Top-5 error rate for ImageNet (NVIDIA devblog)

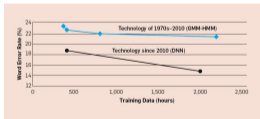


Faster R-CNN (Ren et al. 2015)



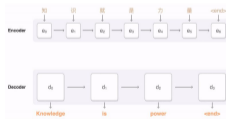
NVIDIA DRIVE PX 2 segmentation

### Speech recognition & synthesis

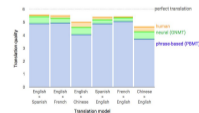


Word error rates (Huang et al., 2014)

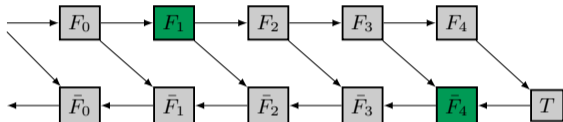
### Machine translation



Google Neural Machine Translation System (GNMT)



# MODEL OF EXECUTION



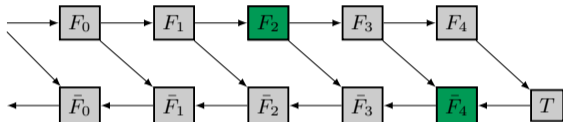
Example of execution

Strategy Time Space

---

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

# MODEL OF EXECUTION



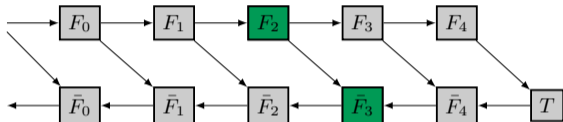
Example of execution

Strategy Time Space

---

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

# MODEL OF EXECUTION



Example of execution

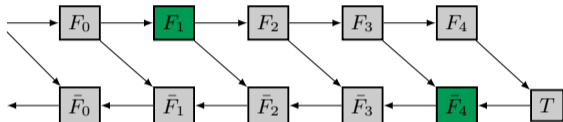
Strategy    Time    Space

---

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .



# MODEL OF EXECUTION



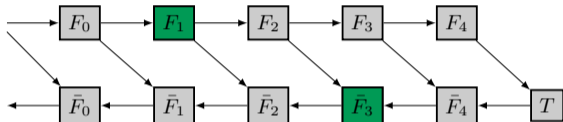
Example of execution

Strategy Time Space

---

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

# MODEL OF EXECUTION



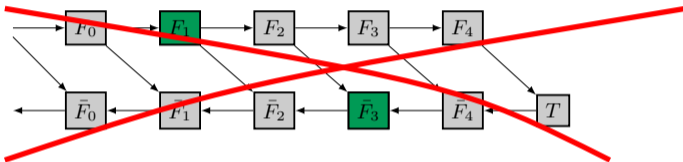
Example of execution

Strategy Time Space

---

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

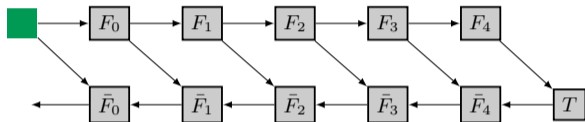
# MODEL OF EXECUTION



Example of execution  
Strategy    Time    Space

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

# MODEL OF EXECUTION



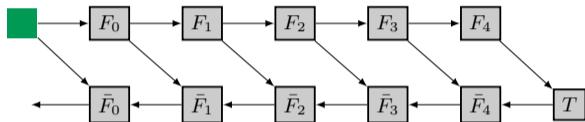
Example of execution

Strategy    Time    Space

---

- Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

# MODEL OF EXECUTION



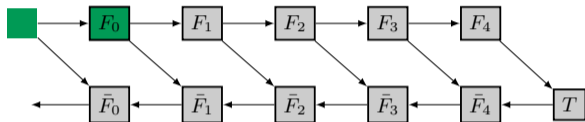
Example of execution

Strategy    Time    Space

---

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .
  - ▶ Cost to write:  $w_m = 0$ ,
  - ▶ Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



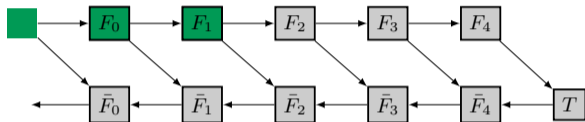
Example of execution

Strategy    Time    Space

---

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .
  - ▶ Cost to write:  $w_m = 0$ ,
  - ▶ Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



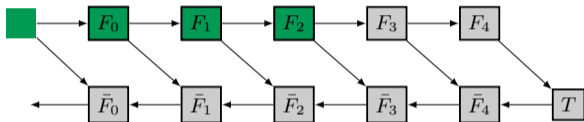
Example of execution

Strategy    Time    Space

---

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .
  - ▶ Cost to write:  $w_m = 0$ ,
  - ▶ Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Example of execution

Strategy    Time    Space

---

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .
  - ▶ Cost to write:  $w_m = 0$ ,
  - ▶ Cost to read:  $r_m = 0$ .









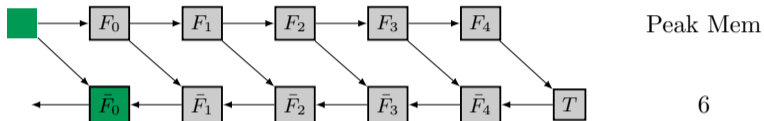








# MODEL OF EXECUTION



Example of execution

Strategy    Time    Space

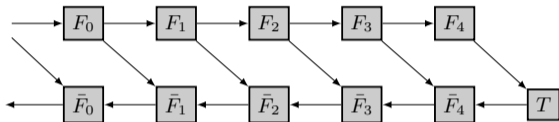
---

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- ▶ Cost to write:  $w_m = 0$ ,
- ▶ Cost to read:  $r_m = 0$ .



# MODEL OF EXECUTION



Peak Mem

6

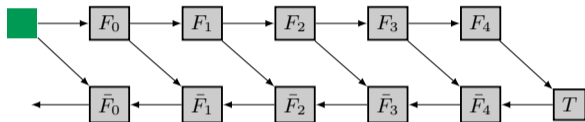
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Peak Mem

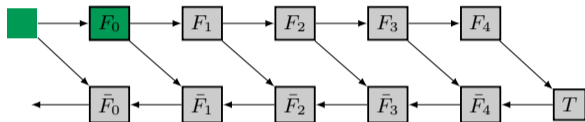
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Peak Mem

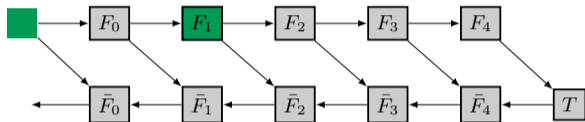
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Peak Mem

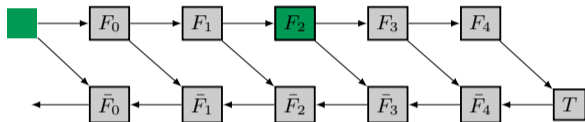
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Peak Mem

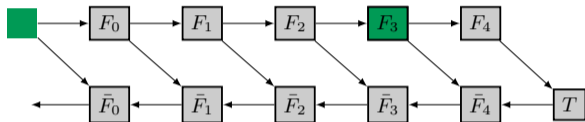
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Peak Mem

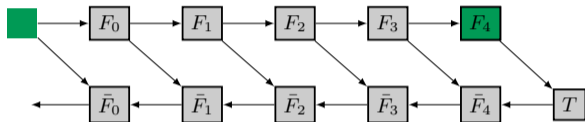
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Peak Mem

Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

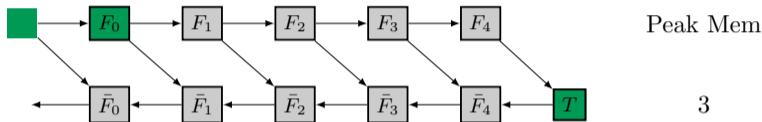
► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .





# MODEL OF EXECUTION



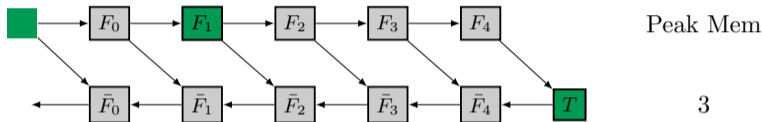
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



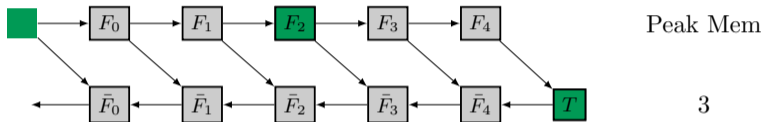
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



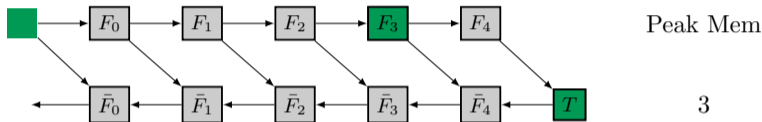
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



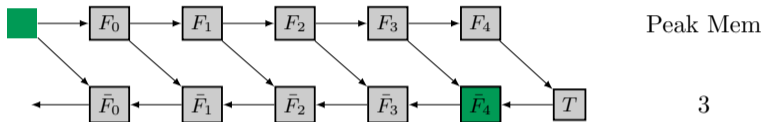
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"		

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



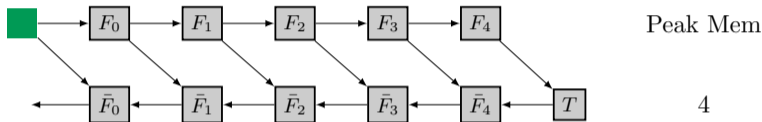
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Example of execution

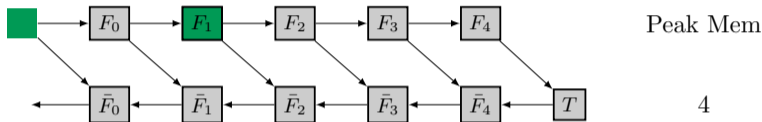
Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .



# MODEL OF EXECUTION



Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”	\$\$\$	\$
No reuse	\$\$	\$\$

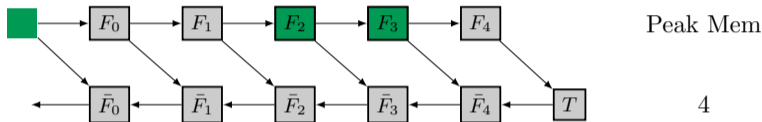
► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .





# MODEL OF EXECUTION



Example of execution

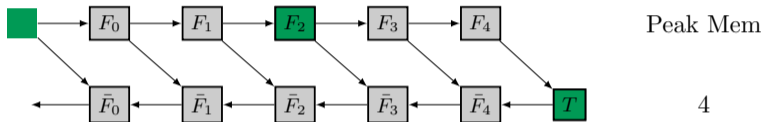
Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .



# MODEL OF EXECUTION



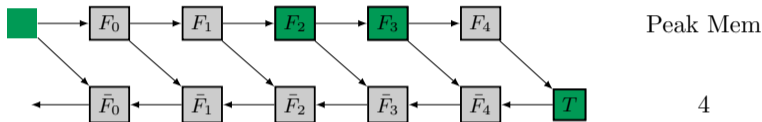
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$

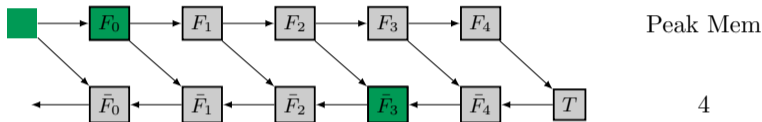
► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .





# MODEL OF EXECUTION



Example of execution

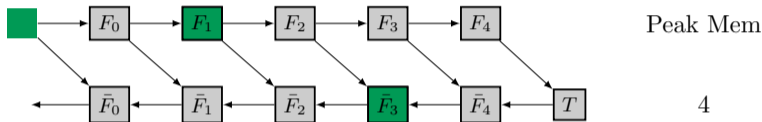
Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .



# MODEL OF EXECUTION



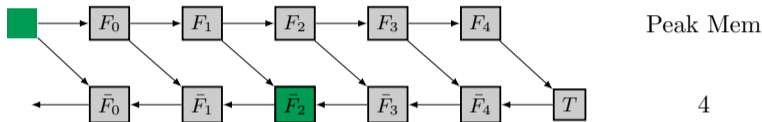
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



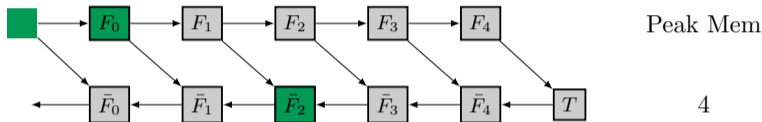
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



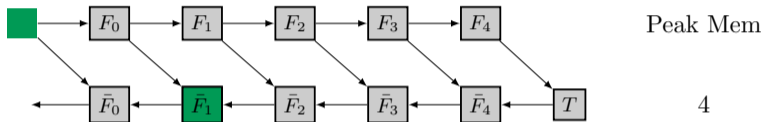
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



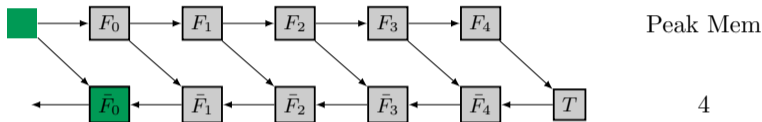
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



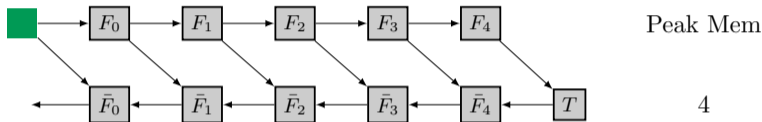
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store “none”	\$\$\$	\$
No reuse	\$\$	\$\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Example of execution

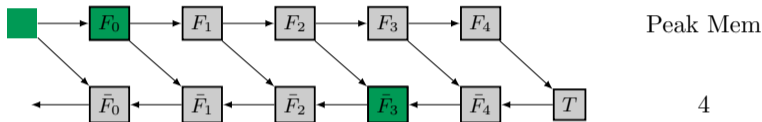
Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$
Memory Reuse	\$	\$

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- ▶ Cost to write:  $w_m = 0$ ,
- ▶ Cost to read:  $r_m = 0$ .



# MODEL OF EXECUTION



Example of execution

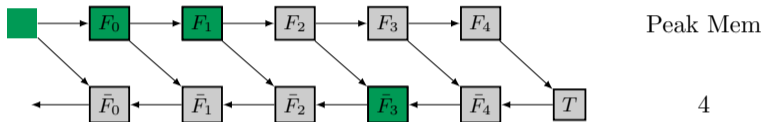
Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$
Memory Reuse	\$	\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .



# MODEL OF EXECUTION



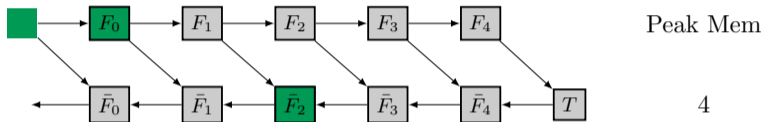
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$
Memory Reuse	\$	\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



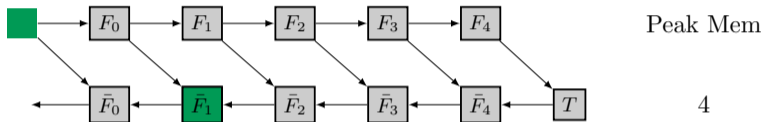
Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$
Memory Reuse	\$	\$

- ▶ Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

- ▶ Cost to write:  $w_m = 0$ ,
- ▶ Cost to read:  $r_m = 0$ .

# MODEL OF EXECUTION



Example of execution

Strategy	Time	Space
Store all	\$	\$\$\$
Store "none"	\$\$\$	\$
No reuse	\$\$	\$\$
Memory Reuse	\$	\$

► Memory to store output of computations ( $x_i$  or  $\bar{x}_i$ ). Initial state: contains  $x_0$ .

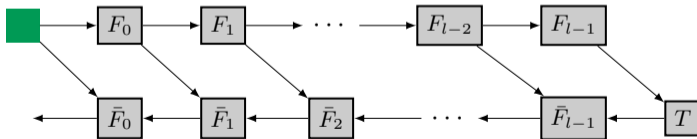
- Cost to write:  $w_m = 0$ ,
- Cost to read:  $r_m = 0$ .



# PROBLEM FORMULATION

We want to minimize the makespan of:

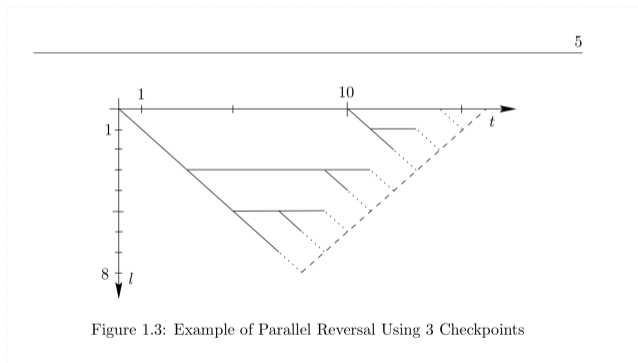
		Initial state:
AC graph:	size $l$	$\mathcal{M}_{\text{ini}} = \{x_0\}$ $S_{\text{ini}} = \emptyset$
Steps:	$u_f, u_b$	
Memory:	$c_m, w_m = r_m = 0,$	
Storage $k$ :	$c_k, w_k, r_k,$	



**Question:** How to organize the reverse execution of intermediate steps?  
What do we store, what do we recompute?

- ▶ Store all: memory expensive
- ▶ Recompute all: compute expensive
- ▶ Intermediate status?

**Griewand and Walther, 2000:**  $\text{REVOLVE}(l, c_m)$ , optimal algorithm with  $c_m$  memory slots.



Source: Andrea Walther's PhD thesis, 1999

# STORAGE HIERARCHY

**A., Herrmann, Hovland, Robert, 2015:** Optimal algorithm for two level of storage: cheap bounded memory and costly unbounded disks.

**A., Herrmann, 2019:** Library of optimal schedules for any number of storage level.

(<https://gitlab.inria.fr/adjoint-computation>)

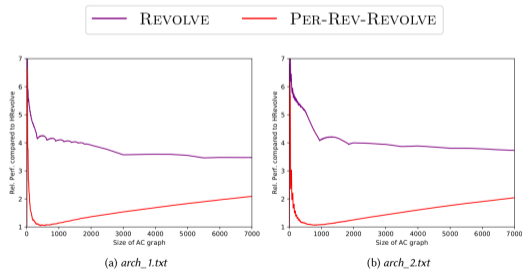
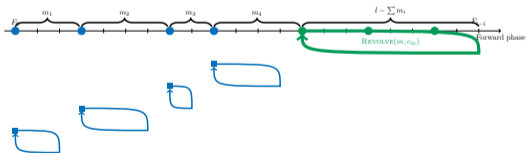


Fig. 5. Relative performance of the heuristics compared to the optimal solution on hierarchical platforms for large graph sizes.



## WHAT DIRECTIONS FOR AI?

Then what? Are we done? Just let AD people and ML people talk together!

## WHAT DIRECTIONS FOR AI?

Then what? Are we done? Just let AD people and ML people talk together!  
**Cut the middle-(scheduling)-people!**



Source: A graffiti in Paris (twitter)

## WHAT DIRECTIONS FOR AI?

While the core of the algorithms remain similar, the problematics are different:

- ▶ Shallower graphs ( $O(100 - 1000)$  levels).
- ▶ Cost functions (time/memory) are not necessarily uniform.
- ▶ Graphs with more structure than chains.
- ▶ Multi-Learners/Hyperparameter tuning (independent graphs executed simultaneously), shared memory?
- ▶ Etc.

## WHAT DIRECTIONS FOR AI?

While the core of the algorithms remain similar, the problematics are different:

- ▶ Shallower graphs ( $O(100 - 1000)$  levels).
- ▶ Cost functions (time/memory) are not necessarily uniform.
- ▶ Graphs with more structure than chains.
- ▶ Multi-Learners/Hyperparameter tuning (independent graphs executed simultaneously), shared memory?
- ▶ Etc.

*Pffiew, just saved my job*

## WHAT DIRECTIONS FOR AI?

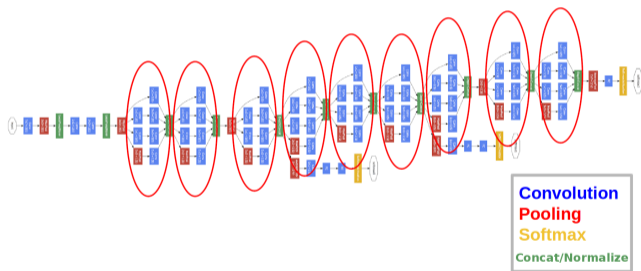
While the core of the algorithms remain similar, the problematics are different:

- ▶ Shallower graphs ( $O(100 - 1000)$  levels).
- ▶ Cost functions (time/memory) are not necessarily uniform.
- ▶ **Graphs with more structure than chains.**
- ▶ Multi-Learners/Hyperparameter tuning (independent graphs executed simultaneously), shared memory?
- ▶ Etc.

*Pffiew, just saved my job*

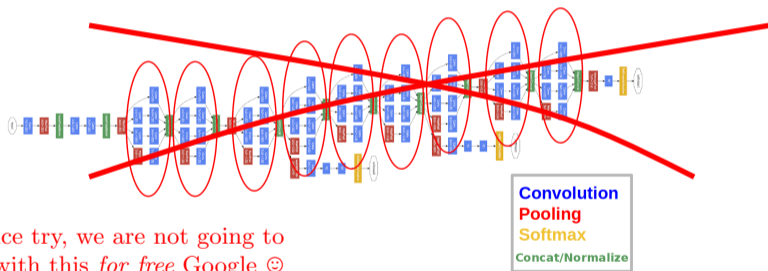
# DIR. FOR AI: GRAPH STRUCTURE

Remember this google network:



# DIR. FOR AI: GRAPH STRUCTURE

Remember this google network:



Nice try, we are not going to help you with this *for free* Google ☺

# DIR. FOR AI: GRAPH STRUCTURE II

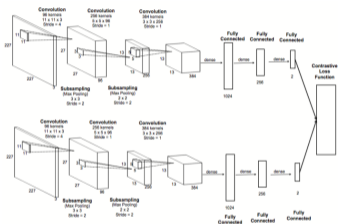


Figure 1: Siamese Neural Network Architecture

Source: Rao et al., *A Deep Siamese Neural Network (...)*, 2016

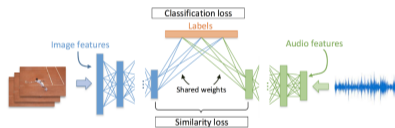


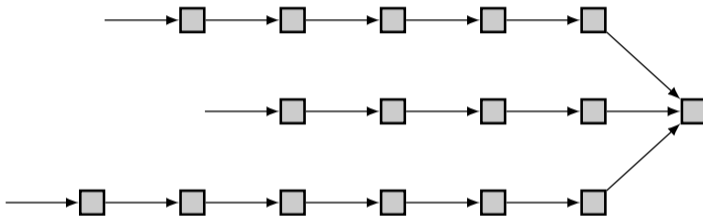
Fig. 2. Schematic of the used architecture.

Source: Surís et al., *Cross-Modal Embeddings for Video and Audio Retrieval*, 2018



## DIR. FOR AI: GRAPH STRUCTURE II

Pitchfork graph<sup>1</sup> (aka join graphs):



Theorem (A., Beaumont, Herrmann, Shilova, 2019)

*Given a bounded memory and a pitchfork with a bounded number of “teeth”, we can find in polynomial time the solution that backpropagates it in minimal time.*

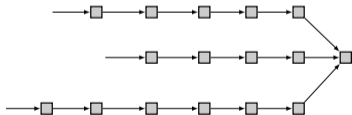
---

<sup>1</sup>This should not be seen as an endorsement of the YJ movement, I don't think I'm allowed to give publicly my opinion.

# A GRASP OF THE PROOF? (I)

Three phase algorithm:

- 1 Forward phase
- 2 Turn
- 3 Backward phase

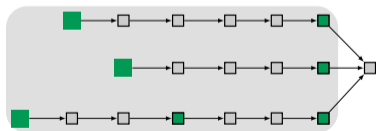


# A GRASP OF THE PROOF? (I)

Three phase algorithm:

- 1 **Forward phase**
- 2 Turn
- 3 Backward phase

► Traverse all branches. Write some intermediate data

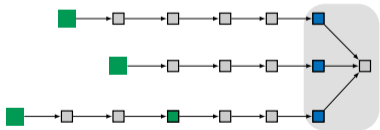


## A GRASP OF THE PROOF? (I)

Three phase algorithm:

- 1 Forward phase
- 2 **Turn**
- 3 Backward phase

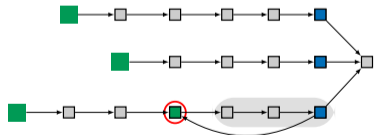
- ▶ Traverse all branches. Write some intermediate data
- ▶ *Backpropagate* the handle of the pitchfork



## A GRASP OF THE PROOF? (I)

Three phase algorithm:

- 1 Forward phase
- 2 Turn
- 3 **Backward phase**



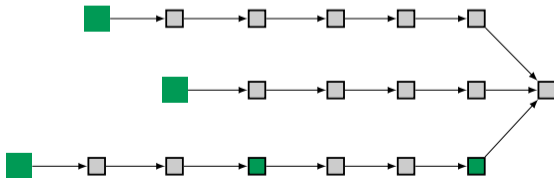
- ▶ Traverse all branches. Write some intermediate data
- ▶ *Backpropagate* the handle of the pitchfork
- ▶ Iteratively, read some checkpointed data from one of the branches, backpropagate a subset of the graph (can write additional intermediate data)

## A GRASP OF THE PROOF? (II)

It relies on key properties of the **backward** phase:

- ▶ Stability of execution
- ▶ Checkpoint persistence

which give us a multi-phase approach.



## A GRASP OF THE PROOF? (II)

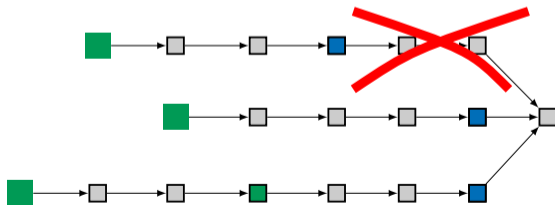
It relies on key properties of the **backward** phase:

- ▶ Stability of execution
- ▶ Checkpoint persistence

which give us a multi-phase approach.

### Lemma (Stability 1)

*If  $F_i$  is “backpropagated”, then there are no  $F_j$  for  $i \leq j$ .*



## A GRASP OF THE PROOF? (II)

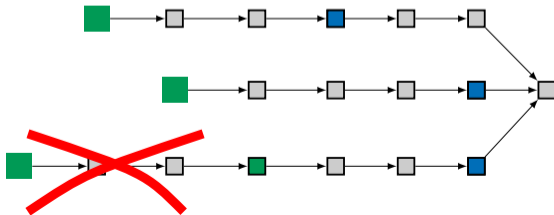
It relies on key properties of the **backward** phase:

- ▶ Stability of execution
- ▶ Checkpoint persistence

which give us a multi-phase approach.

### Lemma (Checkpoint persistence)

*If  $x_i$  is stored, until  $F_i$  is “backpropagated”, there are no  $F_j$  for  $j < i$ .*





## A GRASP OF THE PROOF? (II)

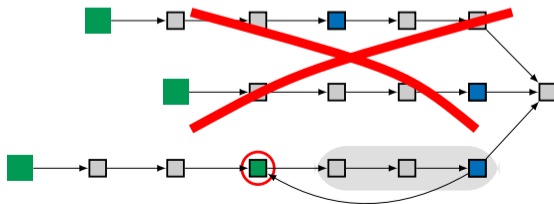
It relies on key properties of the **backward** phase:

- ▶ Stability of execution
- ▶ Checkpoint persistence

which give us a multi-phase approach.

### Lemma (Stability 2)

*If  $x_i$  is read, then there are no  $F_j$  on other branches until it is backpropagated.*



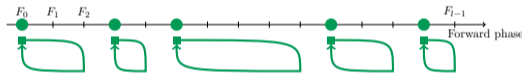
## A GRASP OF THE PROOF? (II)

It relies on key properties of the **backward** phase:

- ▶ Stability of execution
- ▶ Checkpoint persistence

which give us a multi-phase approach.

In this case, for a given forward phase, we get a multi-phase backward phase:



- ▶ Where do we schedule the checkpoints in the forward phase?
- ▶ In which order do we execute the subsegment on each branch?

IS IT WORTH IT?

## IS IT WORTH IT?

- ▶ From a scheduling perspective: **Yes!** (new fun problems)

- ▶ From a scheduling perspective: **Yes!** (new fun problems)
- ▶ From an adjoint perspective: **Yes!**  
With a memory of size  $O(M)$ :
  - ▶ *Store All* can execute a graph of size  $O(M)$  in time  $O(M)$ ;
  - ▶ *Revolve* can execute a graph of size  $O(e^M)$  in time  $O(Me^M)$ !
  - ▶ *H-Revolve* improves performance by a factor of magnitude.

- ▶ From a scheduling perspective: **Yes!** (new fun problems)
- ▶ From an adjoint perspective: **Yes!**  
With a memory of size  $O(M)$ :
  - ▶ *Store All* can execute a graph of size  $O(M)$  in time  $O(M)$ ;
  - ▶ *Revolve* can execute a graph of size  $O(e^M)$  in time  $O(Me^M)$ !
  - ▶ *H-Revolve* improves performance by a factor of magnitude.
- ▶ Machine Learning perspective: deeper networks!

- ▶ From a scheduling perspective
- ▶ From an adjoint perspective  
With a memory of size  $C$ 
  - ▶ *Store All* can execute a
  - ▶ *Revolve* can execute a
  - ▶ *H-Revolve* improves performance
- ▶ Machine Learning perspective

**A radical new neural network design could overcome big challenges in AI**

Researchers borrowed equations from calculus to **redesign the core machinery of deep learning** so it can model continuous processes like changes in health.

by Karen Hao December 12, 2018

**D**avid Duvenaud was collaborating on a project involving medical data when he ran up against a major shortcoming in AI.

An AI researcher at the University of Toronto, he wanted to build a deep-learning model that would predict a patient's health over time. But data from medical records is kind of messy: throughout your life, you might visit the doctor at different times for different reasons, generating a smattering of measurements at arbitrary intervals. A **traditional neural network struggles to handle this**. Its design requires it to learn from data with clear stages of observation. Thus it is a poor

problems)

the  $O(M)$ ;  
the  $O(Me^M)$ !  
magnitude.

Thanks