A Truly Sparse and General Implementation of Gradient-based Synaptic Plasticity

Jamie Lohoff[1,2], Anil Kaya[1,2], Florian Assmuth[1,2], and Emre Neftci[1,2]

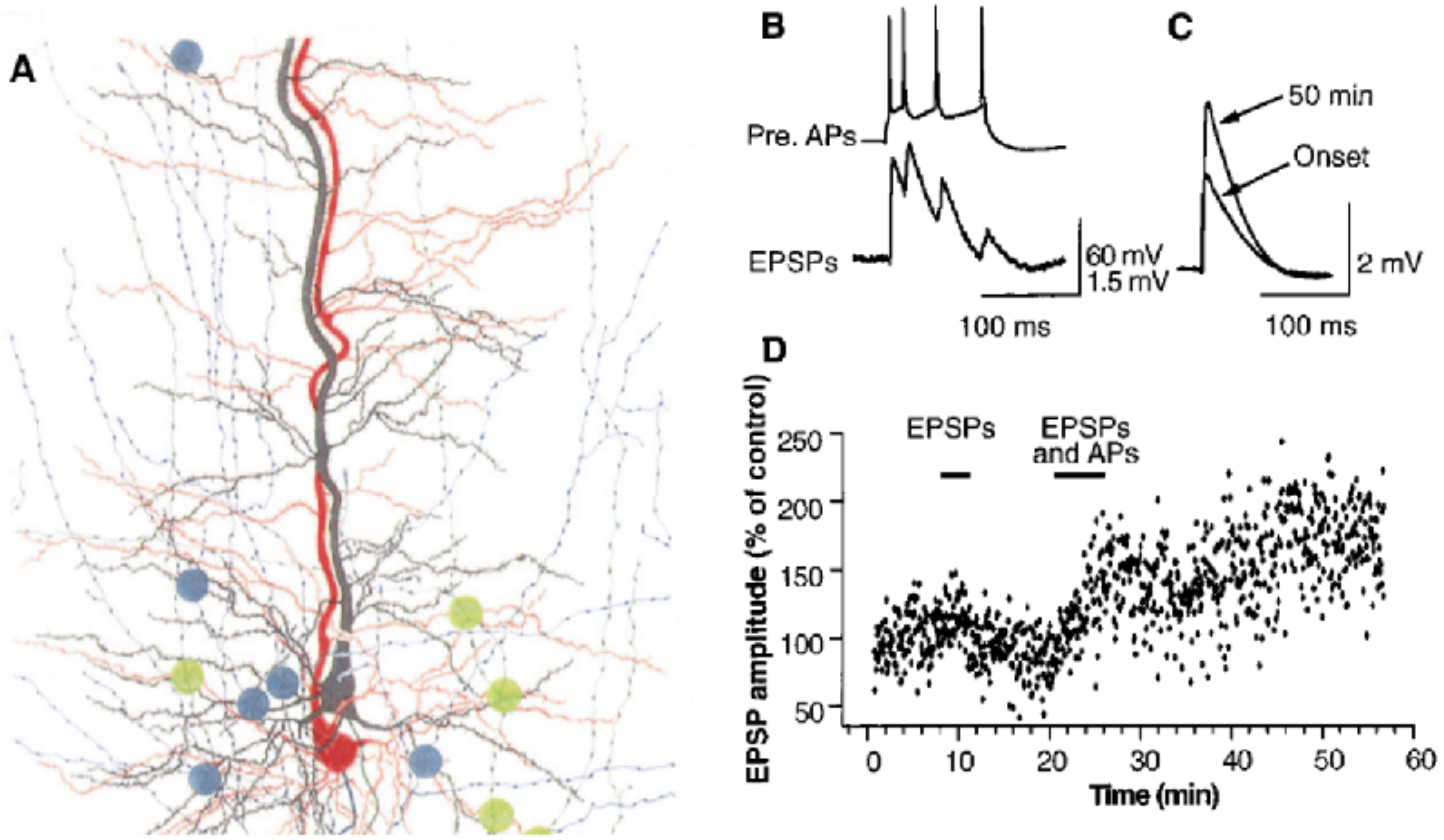
[1] Jülich Research Center, Peter-Grünberg-Institute 15, 52428 Jülich, Germany

[2] RWTH Aachen, Department of Electrical Engineering, 52062 Aachen, Germany

JÜLICH
Forschungszentrum

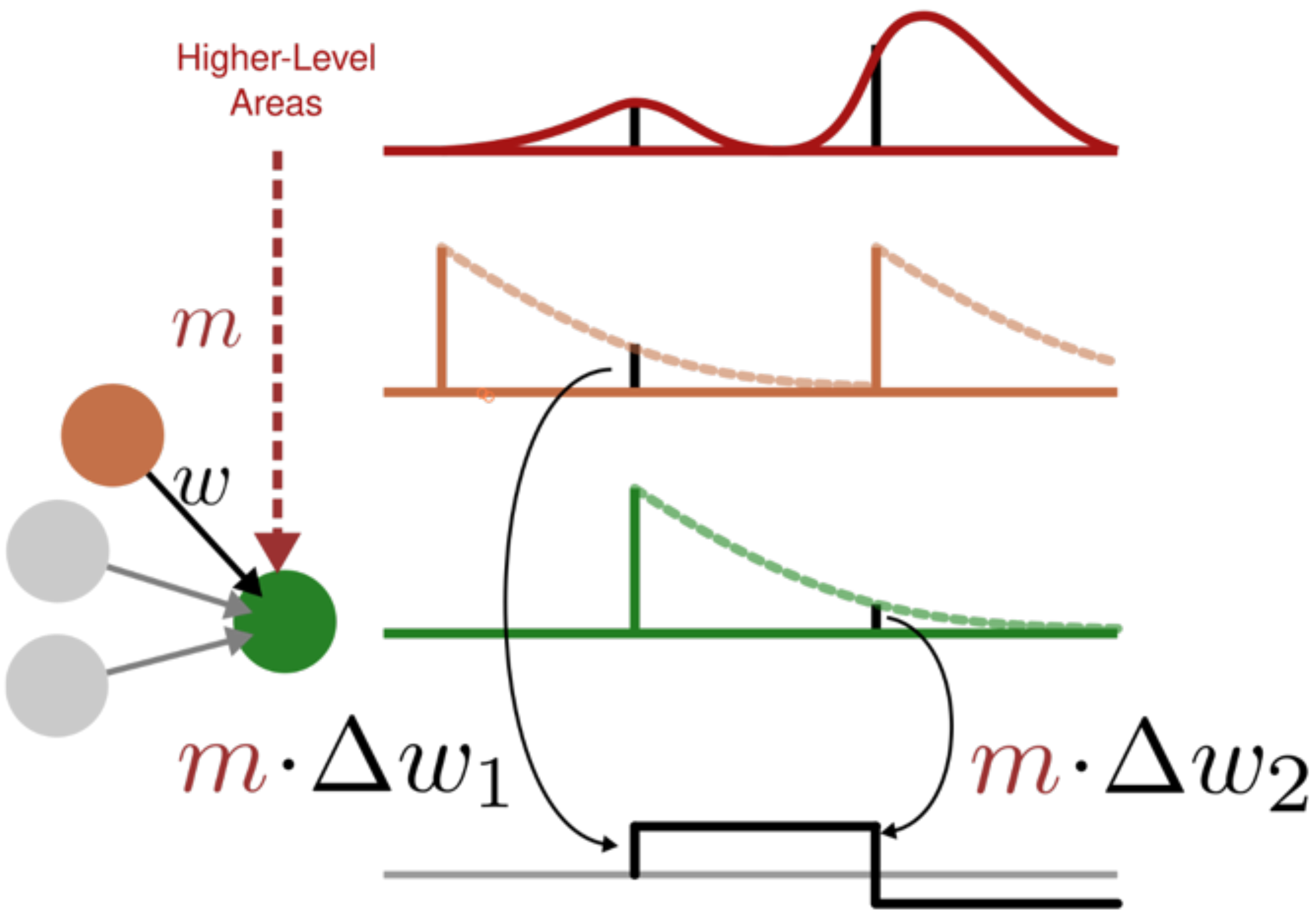
Gradient-based Synaptic Plasticity

Long-term potentiation (LTP) is regulated by coincidence of action potentials (AP) and postsynaptic potentials (EPSP)



Markram et al., Science, 1997

Three Factor Rules modulates learning with higher level information (reward, error, etc.)



Synaptic plasticity rules can be made compatible with a local approximation of gradient descent

Gradient-based synaptic weight update: $\Delta W \propto \frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial S^t} \Theta'(U^t) \frac{\partial U^t}{\partial W}$

Gradient-based Synaptic Plasticity

Synaptic plasticity rules can be made compatible with a local approximation of gradient descent

Gradient-based synaptic weight update: $\Delta W \propto \frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial S^t} \Theta'(U^t) \frac{\partial U^t}{\partial W}$

E-prop $\Rightarrow$ Eligibility Traces: $e_{ji}^t = \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \cdot \epsilon_{ji}^t, \quad \epsilon_{ji}^t = \frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \cdot \epsilon_{ji}^{t-1} + \frac{\partial \mathbf{h}_j^t}{\partial W_{ji}}$

Recurrence Relation!

E-prop is capable of online learning and could efficiently implemented, but in practice:

- Use BPTT + stop-gradient
  —> not online, memory-intensive

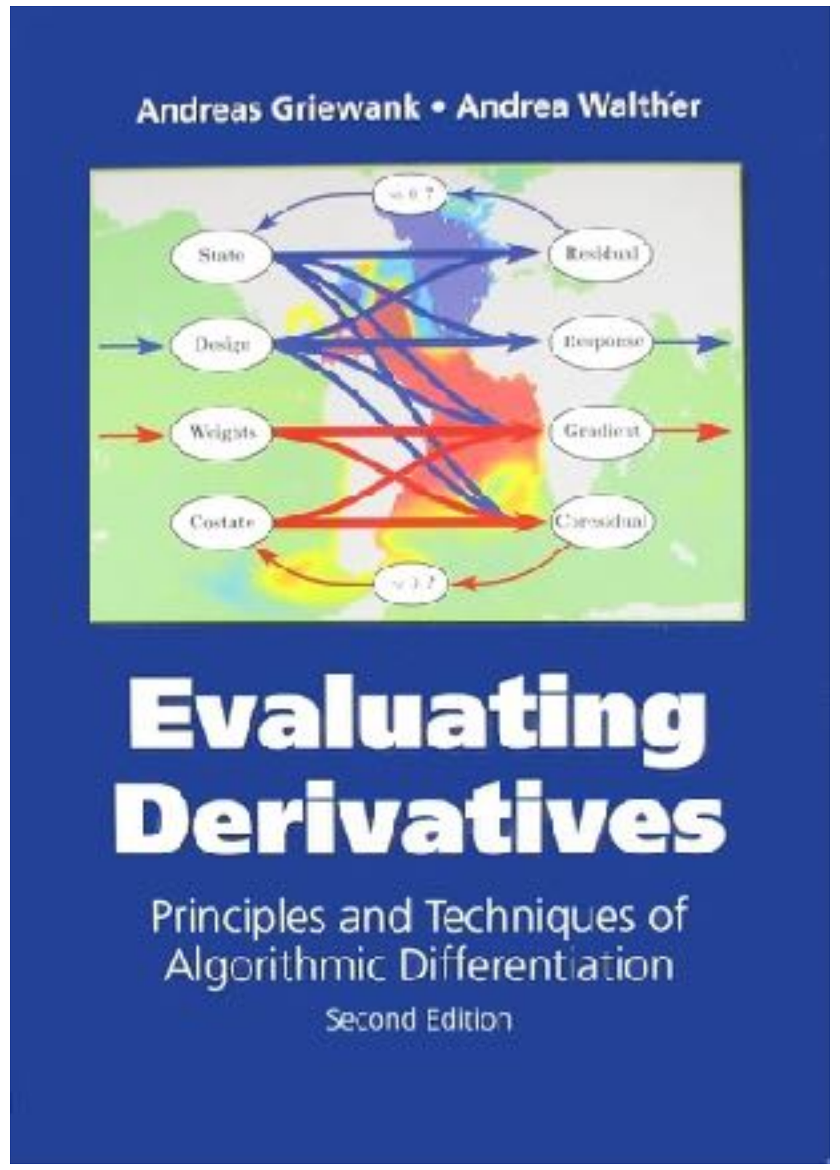
- Hand-implement eligibility traces
  —> does not scale well (in manpower etc.)

$\Rightarrow$ **We have the best of both worlds...**

What is Automatic Differentiation?

- How to compute derivatives algorithmically up to machine precision?

$$\cancel{\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} + O(\Delta^2)}$$



- Applications in ML, Robotics, Computational Fluid Dynamics, Finance, ...

- AutoDiff != gradient backpropagation

- Uses chain rule ⟹ just multiplication of Jacobians

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

$$\textit{single variable}: \; h'(x) = f'(g(x)) \cdot g'(x)$$

$$\textit{multi variable}: \; D\mathbf{h}(\mathbf{x}) = D\mathbf{f}(\mathbf{g}(\mathbf{x})) \cdot D\mathbf{g}(\mathbf{x})$$

- Elemental derivatives are hard-coded ⟹

```
defelemental(lax.sin_p, lambda x: jnp.cos(x))
defelemental(lax.asin_p, lambda x: 1./jnp.sqrt(1.0 - x**2))
defelemental(lax.cos_p, lambda x: -jnp.sin(x))
defelemental(lax.acos_p, lambda x: -1./jnp.sqrt(1.0 - x**2))
defelemental2(lax.tan_p, lambda out, primal: 1.+out**2)
defelemental(lax.atan_p, lambda x: 1./(1. + x**2))
```

Graph View on Automatic Differentiation

- Systematic construction of AutoDiff algorithms
- Algorithms tailored to functions $\Longrightarrow$ minimize computational resources

$$f(x_1, x_2) = \begin{pmatrix} \log \sin x_1 x_2 \\ x_1 x_2 + \sin x_1 x_2 \end{pmatrix} \Longrightarrow$$

Tape

$$\begin{array}{l} v_{-1} = x_1 \\ v_0 = x_2 \\ \hline v_1 = v_{-1} v_0 \\ v_2 = \sin v_1 \\ \hline v_3 = \log v_2 \\ v_4 = v_1 + v_2 \end{array}$$

Constructing the computational graph

Tape

$v_{-1} = x_1$

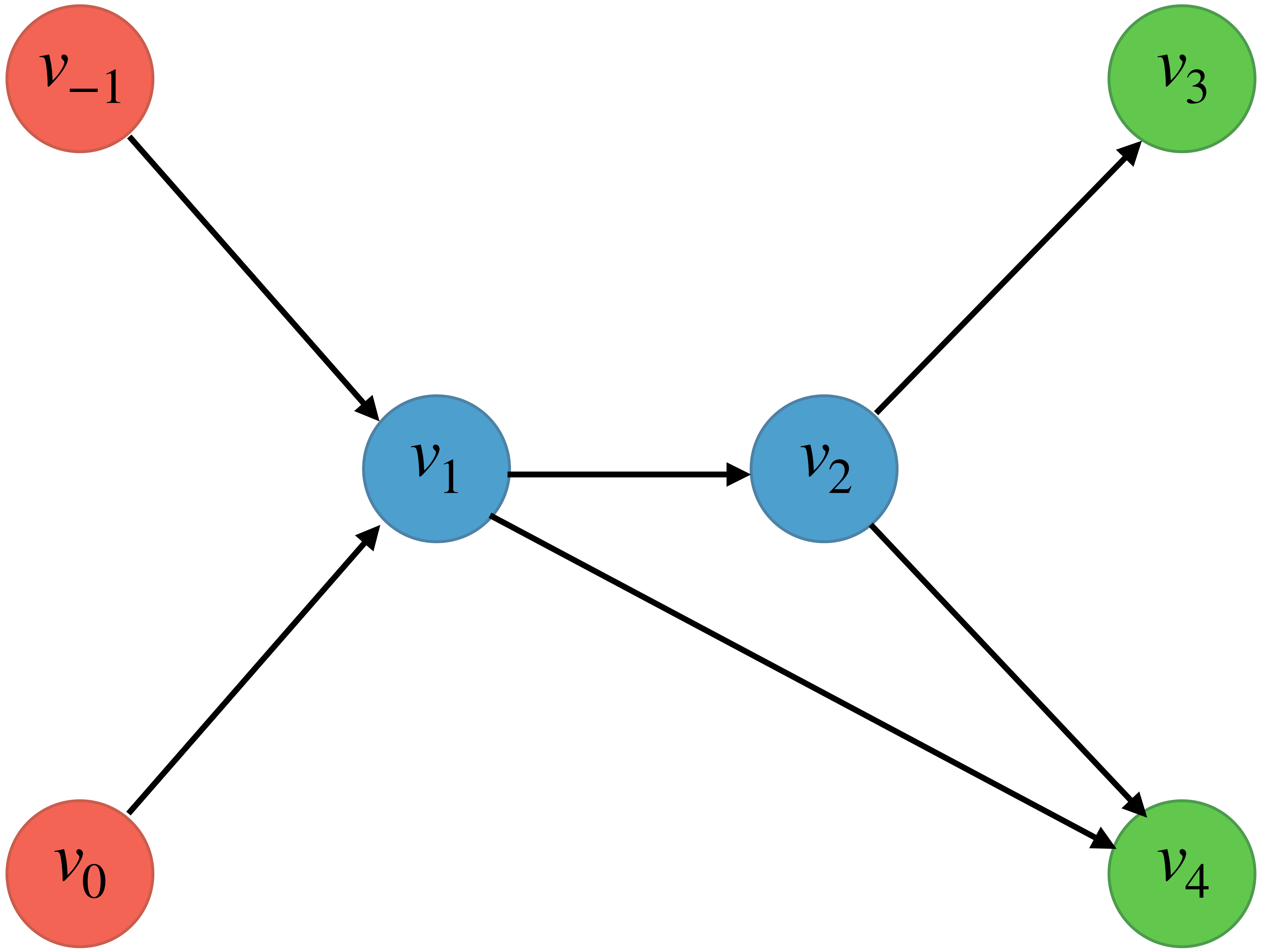
$v_0 = x_2$

$v_1 = v_{-1} v_0$

$v_2 = \sin v_1$

$v_3 = \log v_2$

$v_4 = v_1 + v_2$



Adding the Elemental Derivatives

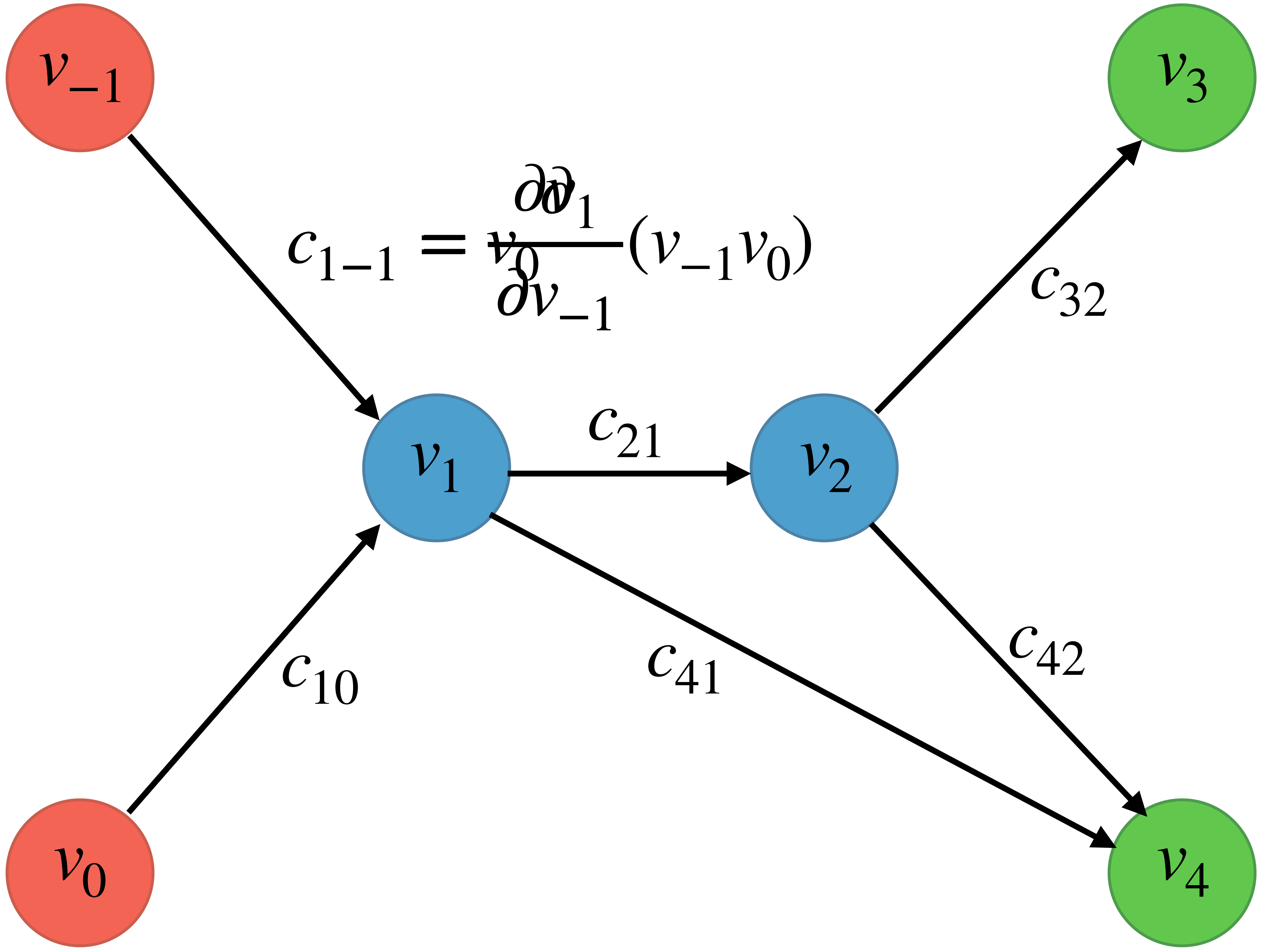
Tape

$v_{-1} = x_1$
$v_0 = x_2$

$v_1 = v_{-1} v_0$
$v_2 = \sin v_1$

$v_3 = \log v_2$
$v_4 = v_1 + v_2$

$c_{10} = v_{-1}$
$c_{21} = \cos v_1$
$c_{32} = 1/v_2$
$c_{41}, c_{4,2} = 1$



Constructing the AutoDiff Algorithm

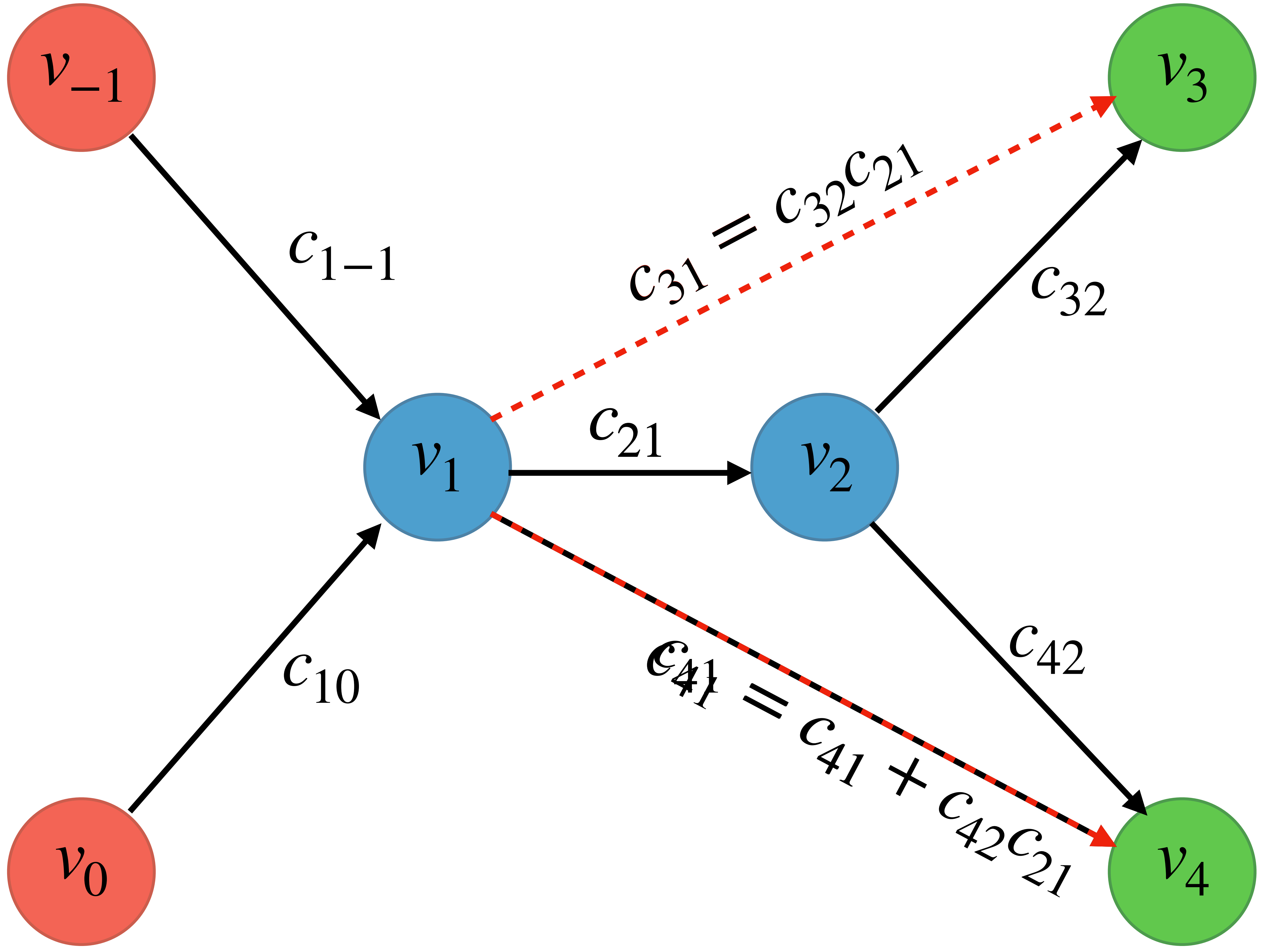
Tape

$v_{-1} = x_1$
$v_0 = x_2$

$v_1 = v_{-1}v_0$
$v_2 = \sin v_1$

$v_3 = \log v_2$
$v_4 = v_1 + v_2$

$c_{1-1} = v_0$
$c_{10} = v_{-1}$
$c_{21} = \cos v_1$
$c_{32} = 1/v_2$
$c_{41}, c_{4,2} = 1$



Eliminating the 2ndVertex

Tape

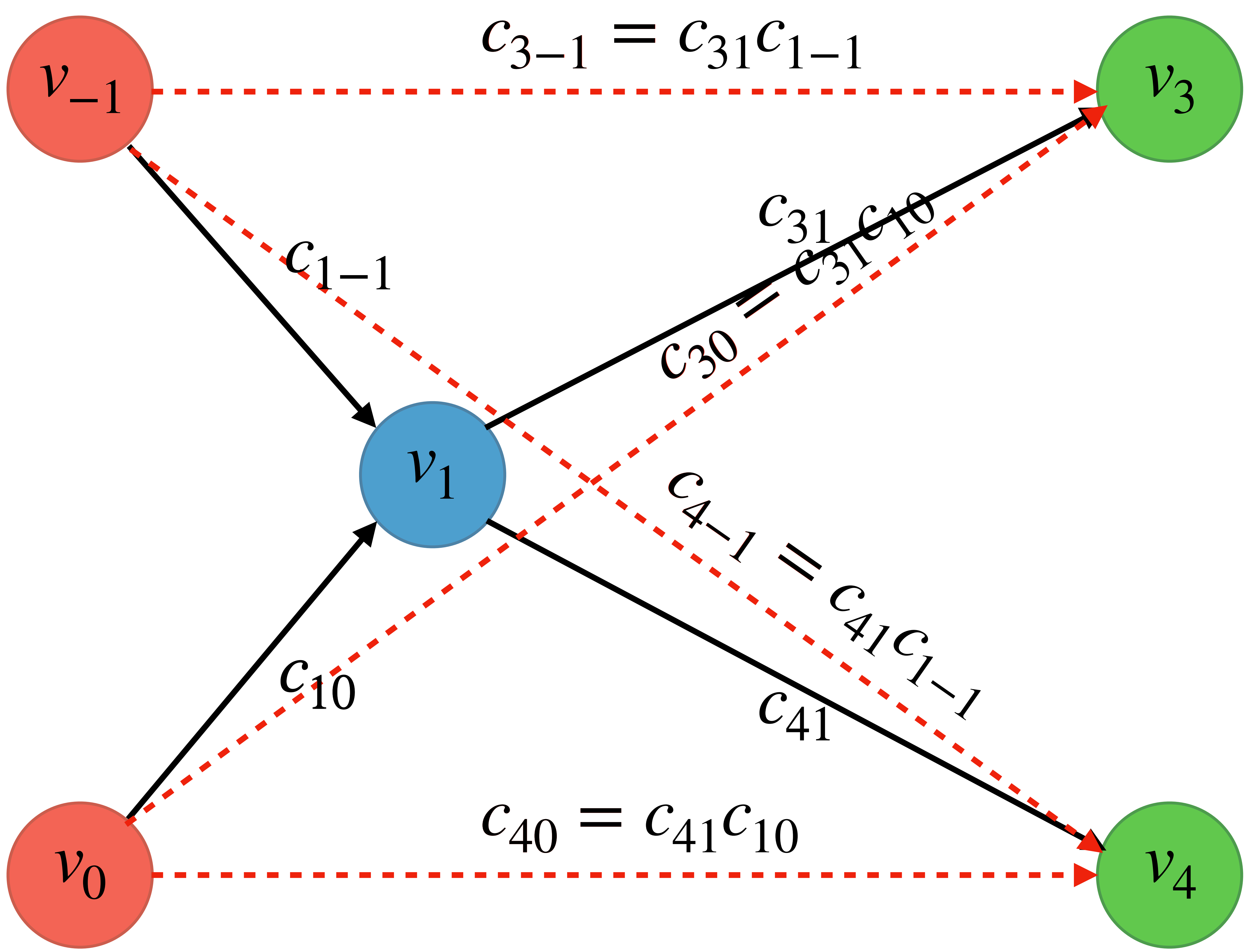
$v_{-1} = x_1$
$v_0 = x_2$

$v_1 = v_{-1}v_0$
$v_2 = \sin v_1$

$v_3 = \log v_2$
$v_4 = v_1 + v_2$

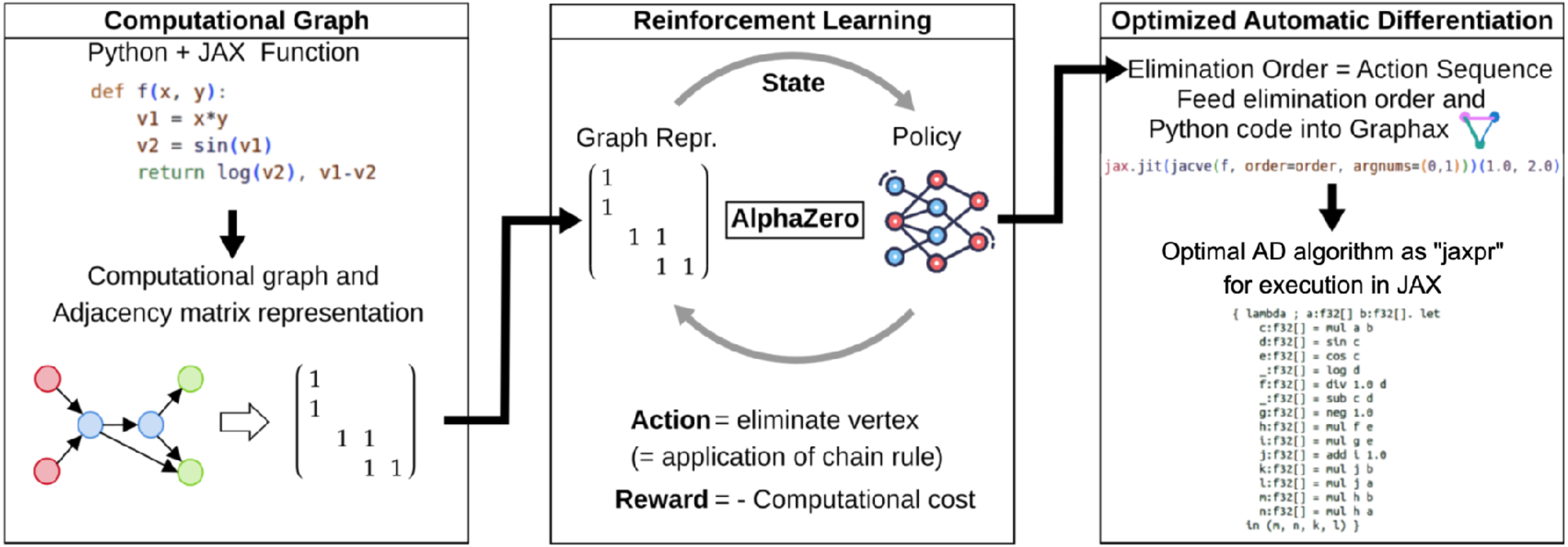
$c_{1-1} = v_0$
$c_{10} = v_{-1}$
$c_{21} = \cos v_1$
$c_{32} = 1/v_2$
$c_{41}, c_{4,2} = 1$

$c_{31} = c_{32}c_{21}$
$c_{41} = c_{41} + c_{42}c_{21}$

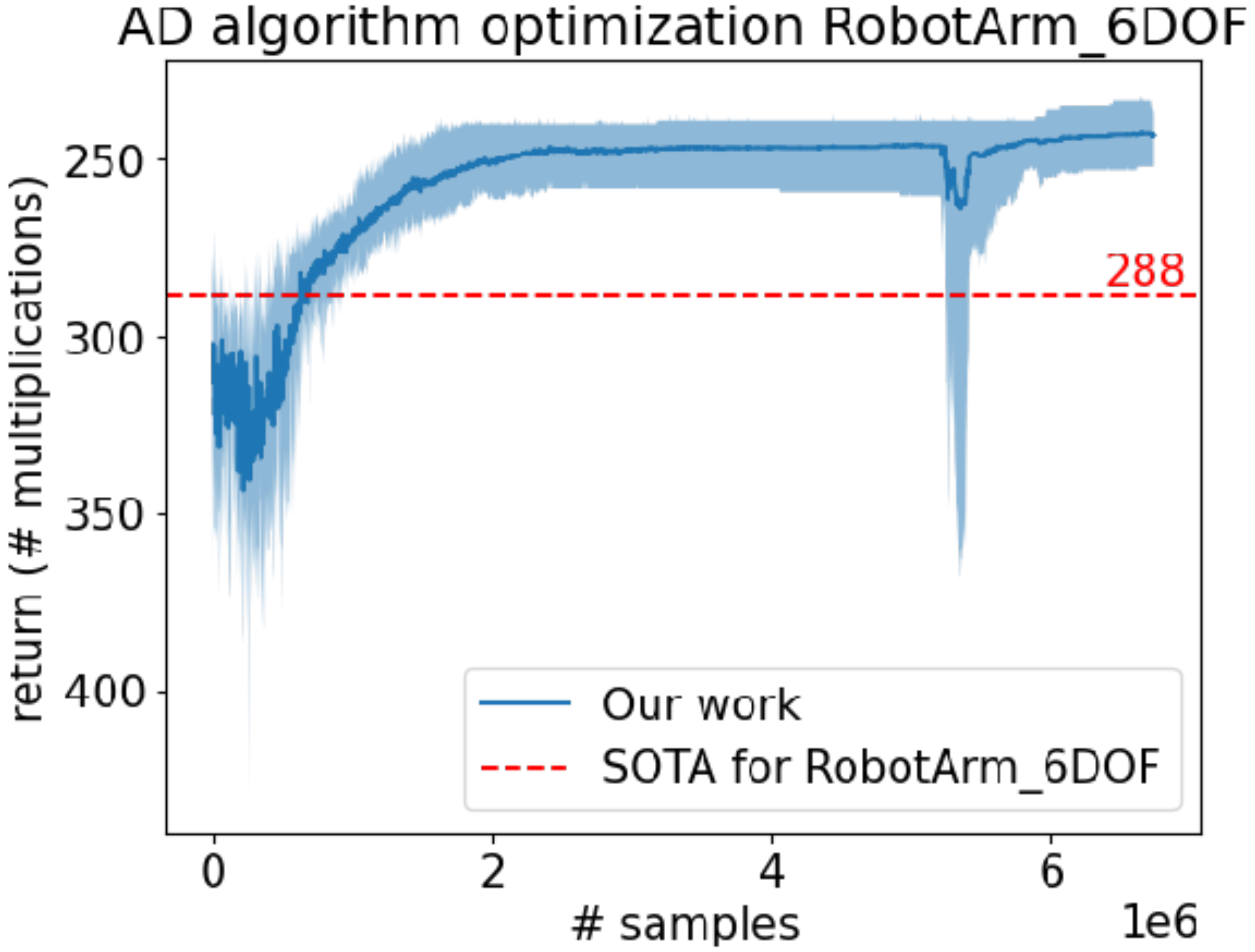


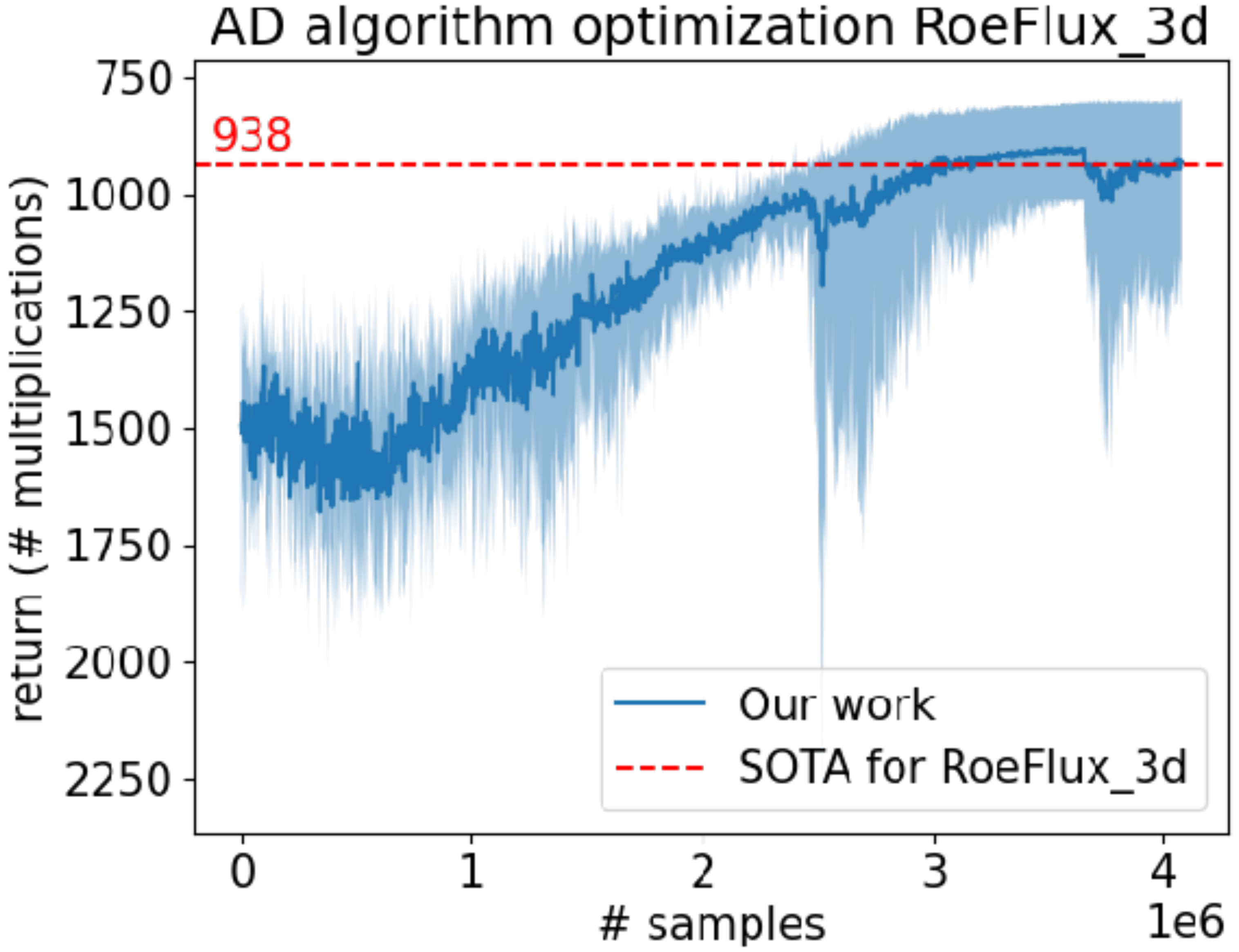
How to find good elimination orders?

Lohoff & Neftci. **Optimizing Automatic Differentiation with Deep Reinforcement Learning.** 38th Conference on Advances in Neural Information Processing (NeurIPS) 2024. Spotlight Paper.

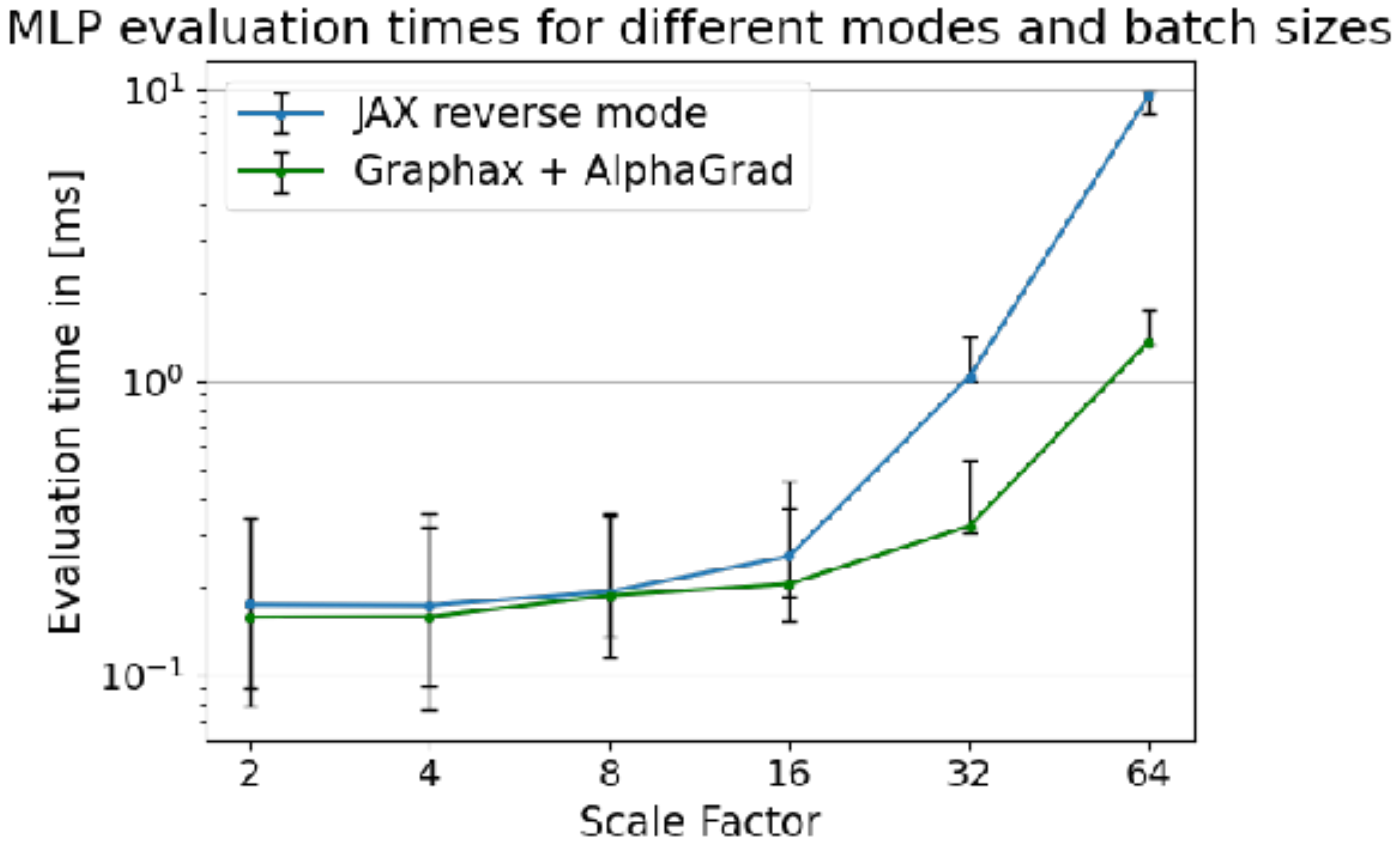


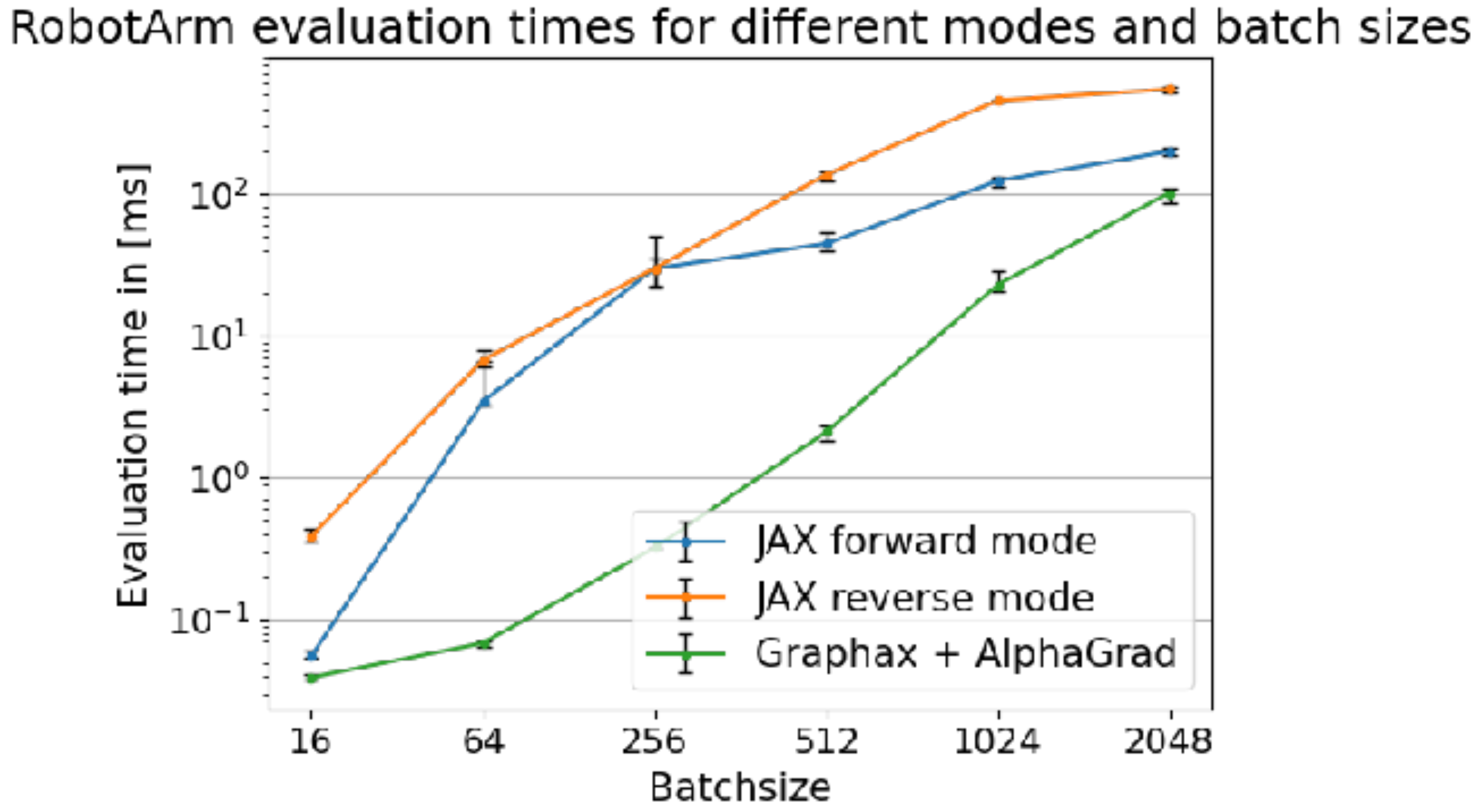
And does it actually work?





Task	Forward	Reverse	Markowitz	AlphaGrad
RoeFlux_1d	620	364	407	320
RobotArm_6DOF	397	301	288	231
HumanHeartDipole	240	172	194	149
PropaneCombustion	151	90	111	88
Random function g	632	566	451	417
BlackScholes	545	572	350	312
RoeFlux_3d	1556	979	938	811
Random function f	17728	9333	12083	6374
2-layer MLP†	10930	392	4796	398 (389)
Transformer†	135010	4688	51869	4831 (4656)

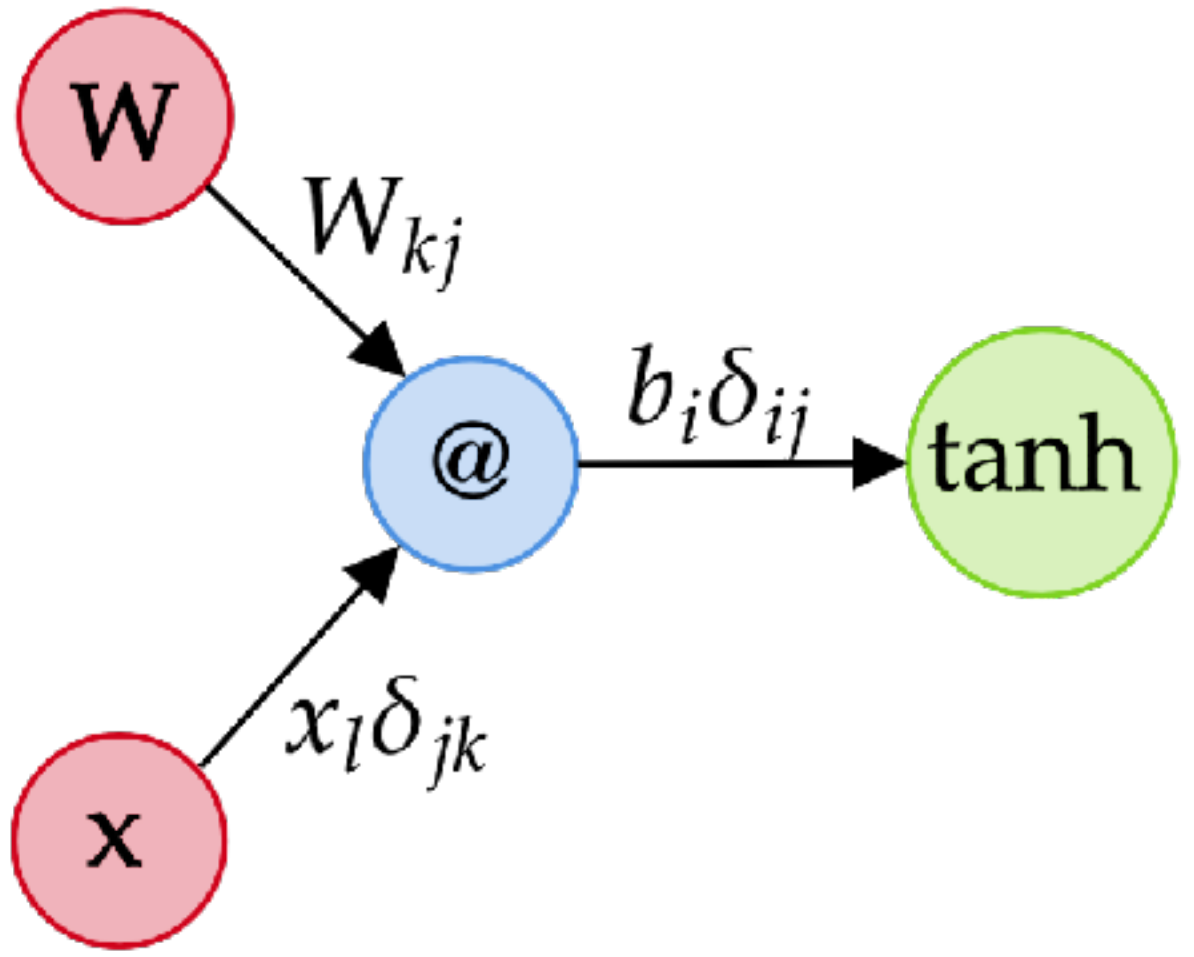




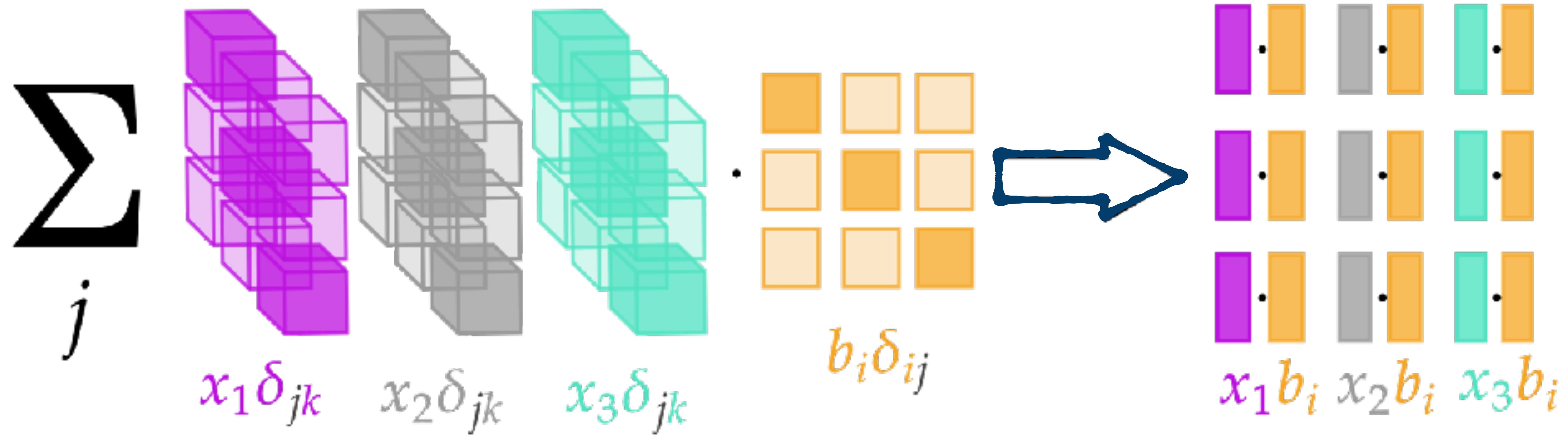
Task	Forward	Reverse	Markowitz	AlphaGrad
RoeFlux_1d	$3.03^{+0.17}_{-0.27}$	$3.08^{+0.17}_{-0.23}$	$2.87^{+0.22}_{-0.66}$	$2.19^{+0.30}_{-0.35}$
RobotArm_6DOF	$8.85^{+0.21}_{-0.17}$	$8.48^{+0.32}_{-0.20}$	$8.55^{+0.38}_{-0.36}$	$6.05^{+0.34}_{-0.35}$
HumanHeartDipole	$16.97^{+1.23}_{-2.39}$	$16.87^{+1.45}_{-3.90}$	$16.42^{+1.29}_{-2.73}$	$15.94^{+1.11}_{-1.10}$
PropaneCombustion	$36.91^{+2.87}_{-1.50}$	$36.47^{+1.45}_{-0.51}$	$36.99^{+1.55}_{-1.11}$	$36.45^{+2.47}_{-1.31}$
Random function g	$82.41^{+2.85}_{-2.97}$	$81.64^{+2.82}_{-3.56}$	$82.97^{+1.38}_{-1.17}$	$80.56^{+1.61}_{-2.30}$
BlackScholes	$5.04^{+0.23}_{-0.33}$	$5.02^{+0.30}_{-0.39}$	$5.03^{+0.23}_{-0.29}$	$4.77^{+0.23}_{-0.29}$
RoeFlux_3d	$72.26^{+3.22}_{-6.02}$	$83.98^{+5.69}_{-6.68}$	$91.27^{+11.59}_{-16.49}$	$63.92^{+4.45}_{-5.76}$
Random function f	$12.42^{+0.66}_{-0.25}$	$11.54^{+0.15}_{-0.27}$	$20.20^{+0.26}_{-0.31}$	$9.12^{+0.08}_{-0.06}$
2-layer MLP†	$760.63^{+80.01}_{-53.30}$	$29.67^{+1.33}_{-4.05}$	$317.65^{+53.30}_{-19.34}$	$28.73^{+1.32}_{-3.78}$
2-layer MLP†(GPU)	$11.04^{+0.31}_{-0.13}$	$0.30^{+0.02}_{-0.03}$	$1.01^{+0.02}_{-0.05}$	$0.29^{+0.02}_{-0.03}$
Transformer†	$990.09^{+35.42}_{-31.11}$	$39.07^{+4.59}_{-7.67}$	$498.94^{+20.34}_{-19.62}$	$40.38^{+5.26}_{-3.62}$
Transformer†(GPU)	$21.38^{+0.22}_{-0.06}$	$0.29^{+0.02}_{-0.03}$	$16.17^{+15.04}_{-0.09}$	$0.28^{+0.02}_{-0.01}$

Tensorized Vertex Elimination - Sparse AutoDiff

$$f(\mathbf{W}, \mathbf{x}) = \tanh(\mathbf{W} \cdot \mathbf{x})$$



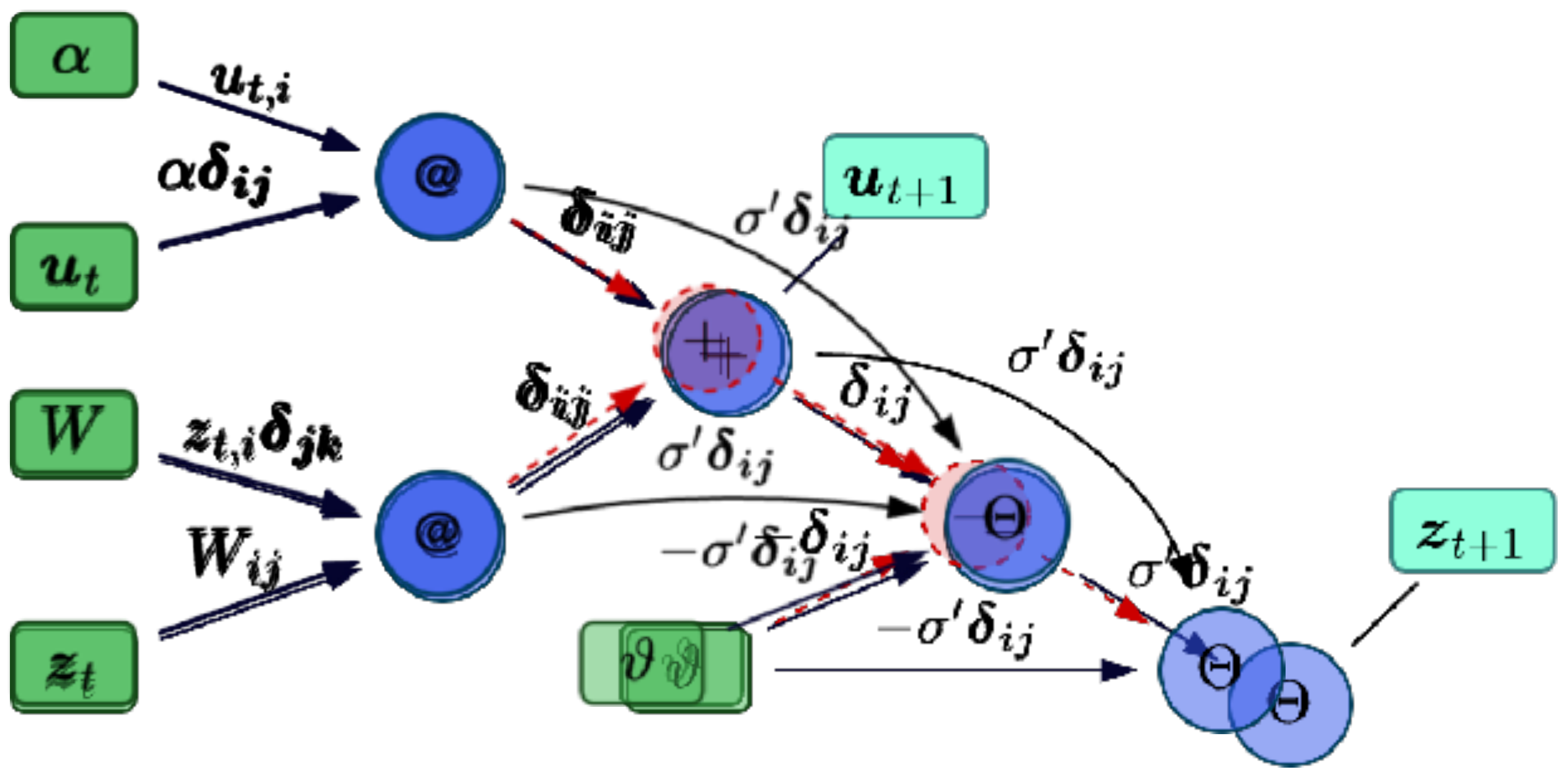
Graphax instead does fast elementwise multiplication



Simple LIF Layer:

$$\mathbf{u}_{t+1} = \alpha\mathbf{u}_t + \mathbf{W} \cdot \mathbf{z}_t$$

$$\mathbf{z}_{t+1} = \Theta(\mathbf{u}_t - \vartheta)$$



Temporal Credit Assignment

Recurrent Neural Network: $h_t = f(h_{t-1}, x_t, \theta)$

Gradient: $\dfrac{\mathrm{d}L}{\mathrm{d}\theta} = \sum_t \dfrac{\mathrm{d}L}{\mathrm{d}h_t}\dfrac{\partial h_t}{\partial \theta}$

$$\frac{\mathrm{d}L}{\mathrm{d}h_t} = \frac{\partial L}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial h_t}$$

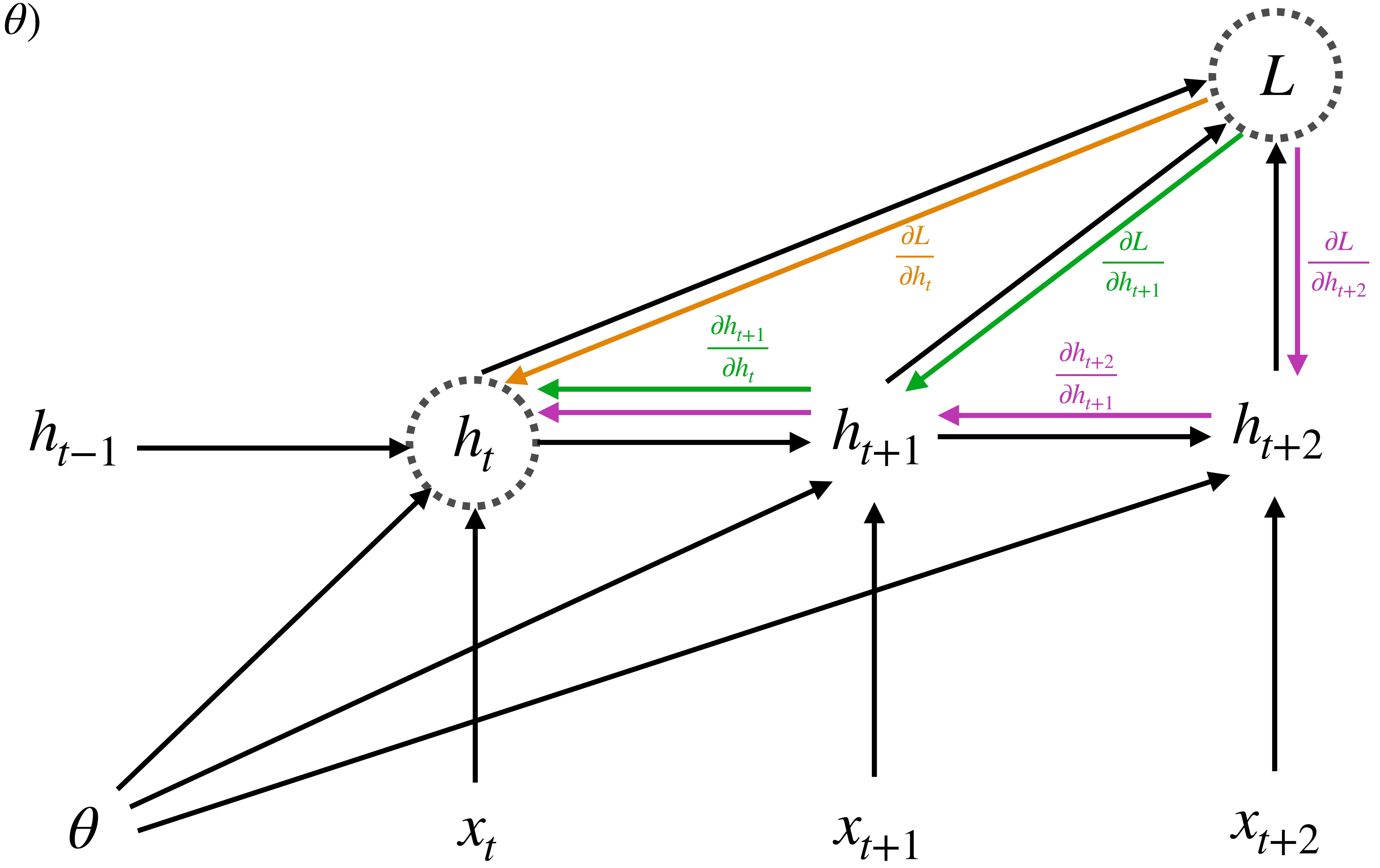
$$\frac{\mathrm{d}L}{\mathrm{d}h_t} = \left(\frac{\partial L}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial h_{t+1}} + \frac{\partial L}{\partial h_{t+1}}\right)\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial h_t}$$

$$c_t = (c_{t+2}H_{t+1} + d_{t+1})H_t + d_t$$

BPTT: $c_t = c_{t+1}H_t + d_t$

Compute Scaling: $\mathcal{O}(n^2 T)$

Memory Scaling: $\mathcal{O}(nT)$



More Temporal Credit Assignment

Gradient: $\dfrac{\mathrm{d}L}{\mathrm{d}\theta} = \sum_t \boxed{\dfrac{\partial L}{\partial h_t}\dfrac{\mathrm{d}h_t}{\mathrm{d}\theta}}$

$$\frac{\mathrm{d}h_{t+2}}{\mathrm{d}\theta} = \frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial \theta} + \frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial \theta} + \frac{\partial h_{t+2}}{\partial \theta}$$

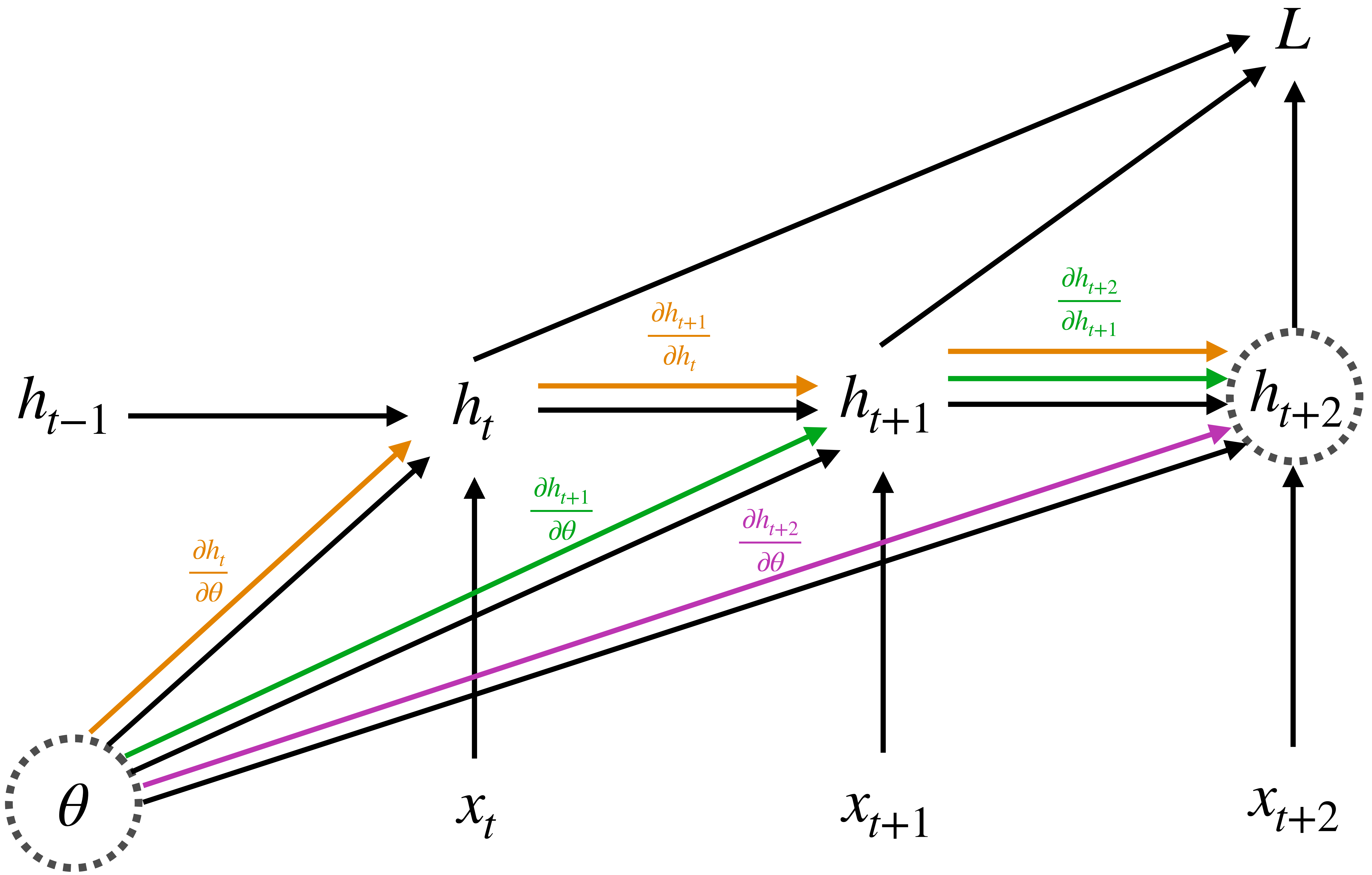
$$\frac{\mathrm{d}h_{t+2}}{\mathrm{d}\theta} = \frac{\partial h_{t+2}}{\partial h_{t+1}}\left(\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial \theta} + \frac{\partial h_{t+1}}{\partial \theta}\right) + \frac{\partial h_{t+2}}{\partial \theta}$$

$$G_{t+2} = H_{t+2}(H_{t+1}G_t) + F_{t+1}) + F_{t+2}$$

RTRL: $G_t = H_t G_{t-1} + F_t$

Compute Scaling: $\mathcal{O}(n^4 T)$

Memory Scaling: $\mathcal{O}(n^3)$



Gradient-based Synaptic Plasticity (again)

RTRL Recurrence Relation: $G_t = H_t G_{t-1} + F_t$

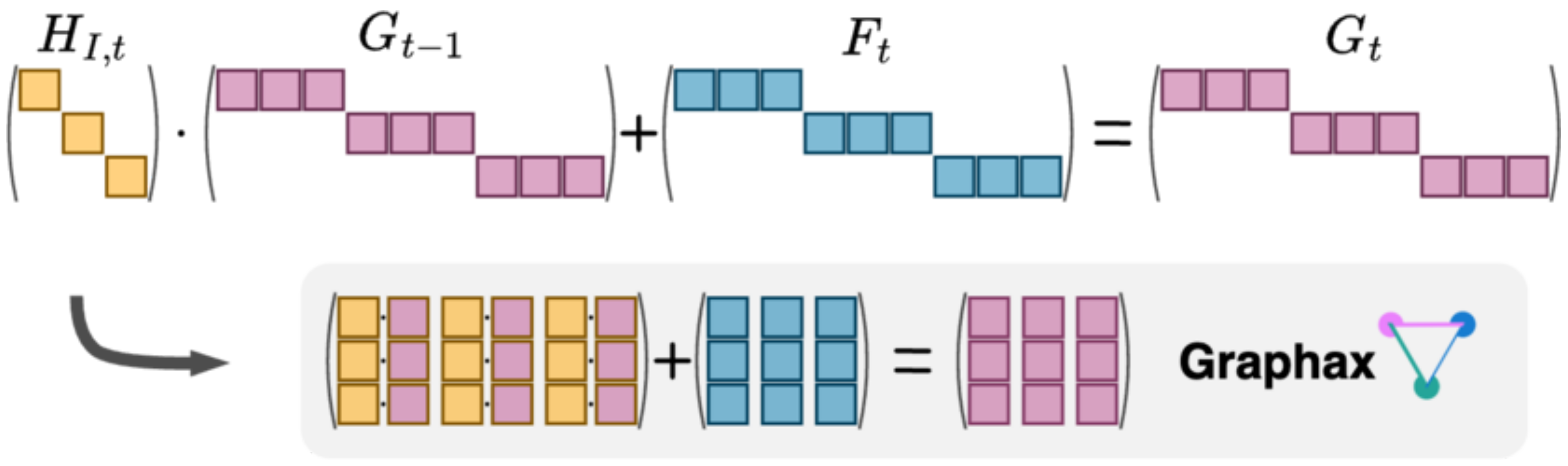
$\Downarrow$

Examine closer: $H_t := \dfrac{\mathrm{d}\mathbf{h}_t}{\mathrm{d}\mathbf{h}_{t-1}} = \dfrac{\mathrm{d}\mathbf{u}_t}{\mathrm{d}\mathbf{u}_{t-1}} + \cancel{\dfrac{\mathrm{d}\mathbf{u}_t}{\mathrm{d}\mathbf{z}_t}\dfrac{\mathrm{d}\mathbf{z}_t}{\mathrm{d}\mathbf{u}_{t-1}}}$

Neglect Explicit Recurrence!

$\Downarrow$

H_t becomes diagonal, so G_t stays diagonal:



Scaling Laws:

	RTRL	BPTT
Compute	$\mathcal{O}(n^4T)$	$\mathcal{O}(n^2T)$
Memory	$\mathcal{O}(n^3)$	$\mathcal{O}(nT)$

Simple LIF Neuron:

$$\mathbf{u}_{t+1} = \alpha\mathbf{u}_t + \mathbf{W}\cdot\mathbf{z}_t$$
$$\mathbf{z}_{t+1} = \Theta(\mathbf{u}_t - \vartheta)$$

$\Rightarrow h_t = \mathbf{u}_t$

New Compute Scaling: $\mathcal{O}(n^2T)$

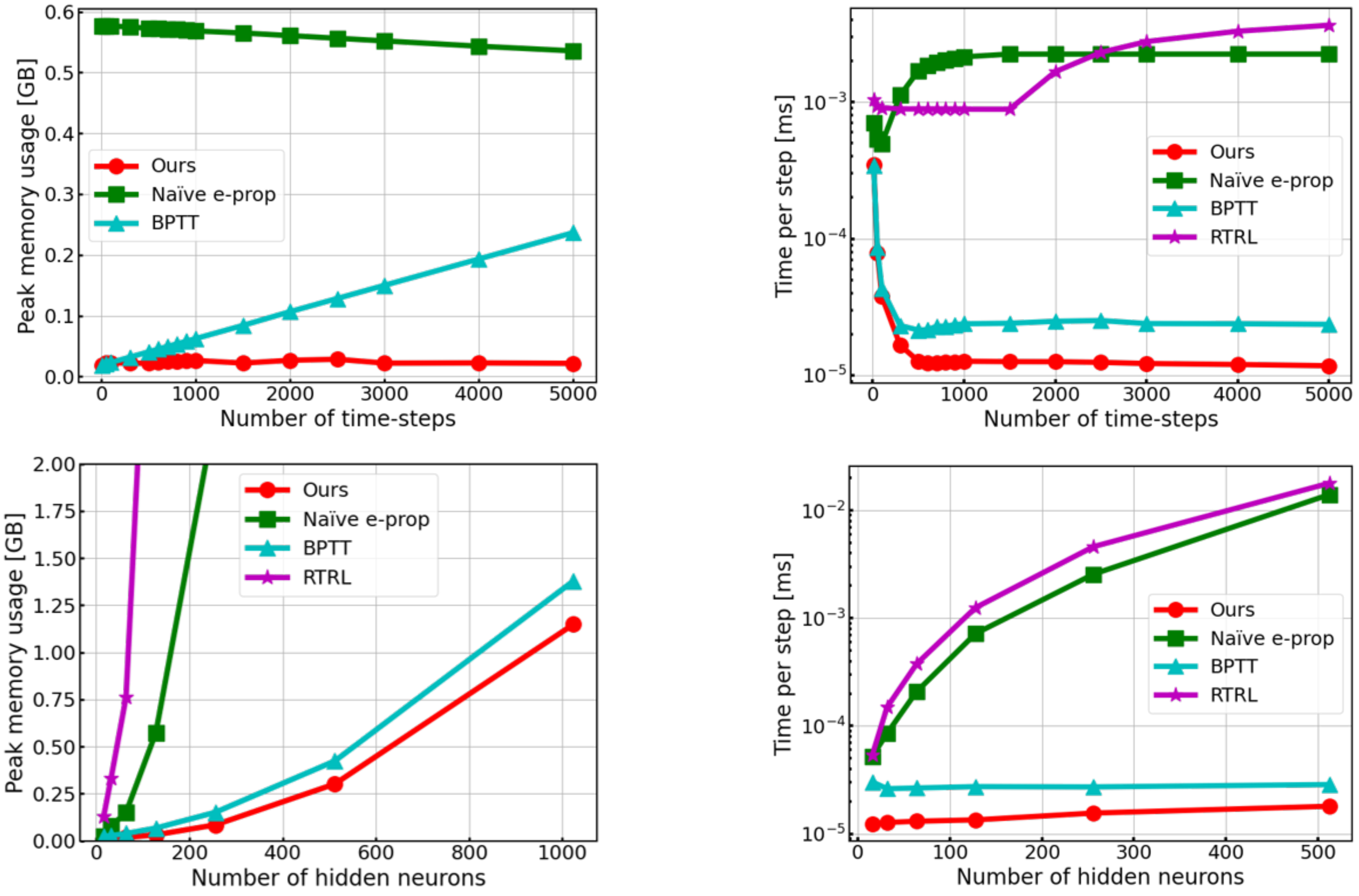
New Memory Scaling: $\mathcal{O}(n^2)$

Zenke and Neftci. **Brain-inspired learning on neuromorphic substrates**. Proceedings of the IEEE 109, Issue 5. 2021.

Bellec et al. **A solution to the learning dilemma for recurrent networks of spiking neurons**. Nature Communications 11. 2020.

James M Murray **Local online learning in recurrent networks with random feedback.** *eLife 8:e43299. 2019.*

Our Method in Action:



Outlook into the Future

\+ \+ Graphax = **Really Fast and Memory-efficient SNN Training**

- Generalize to **multiple layers** to enable large-scale training & meaningful applications

  Kaiser et al. **Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)**. Frontiers of Neuroscience 14. 2020.

  Bohnstingl et al. **Online spatiotemporal learning in deep neural networks**. IEEE TNNLS 34, Issue 11. 2023.

- Application to **Dendritic Neural Networks** to make them scalable (with Yiota Poirazis Group)

  Chavlis & Poirazi. **Dendrites endow artificial neural networks with accurate, robust and parameter-efficient learning**. Arxiv:2404.03708. 2024.

- Test more intricate neuron types, train SNNs on **very long time-horizons**

- Use Vertex Elimination formalism to find **novel approximation methods** (e.g. with RL)

- Include **HW constraints** in the search for good gradient-based Synaptic Plasticity