A Parallelized, Adam-Based Solver for Reserve and Security Constrained AC Unit Commitment

Sam Chevalier

University of Vermont

MARIE CURIE ACTIONS

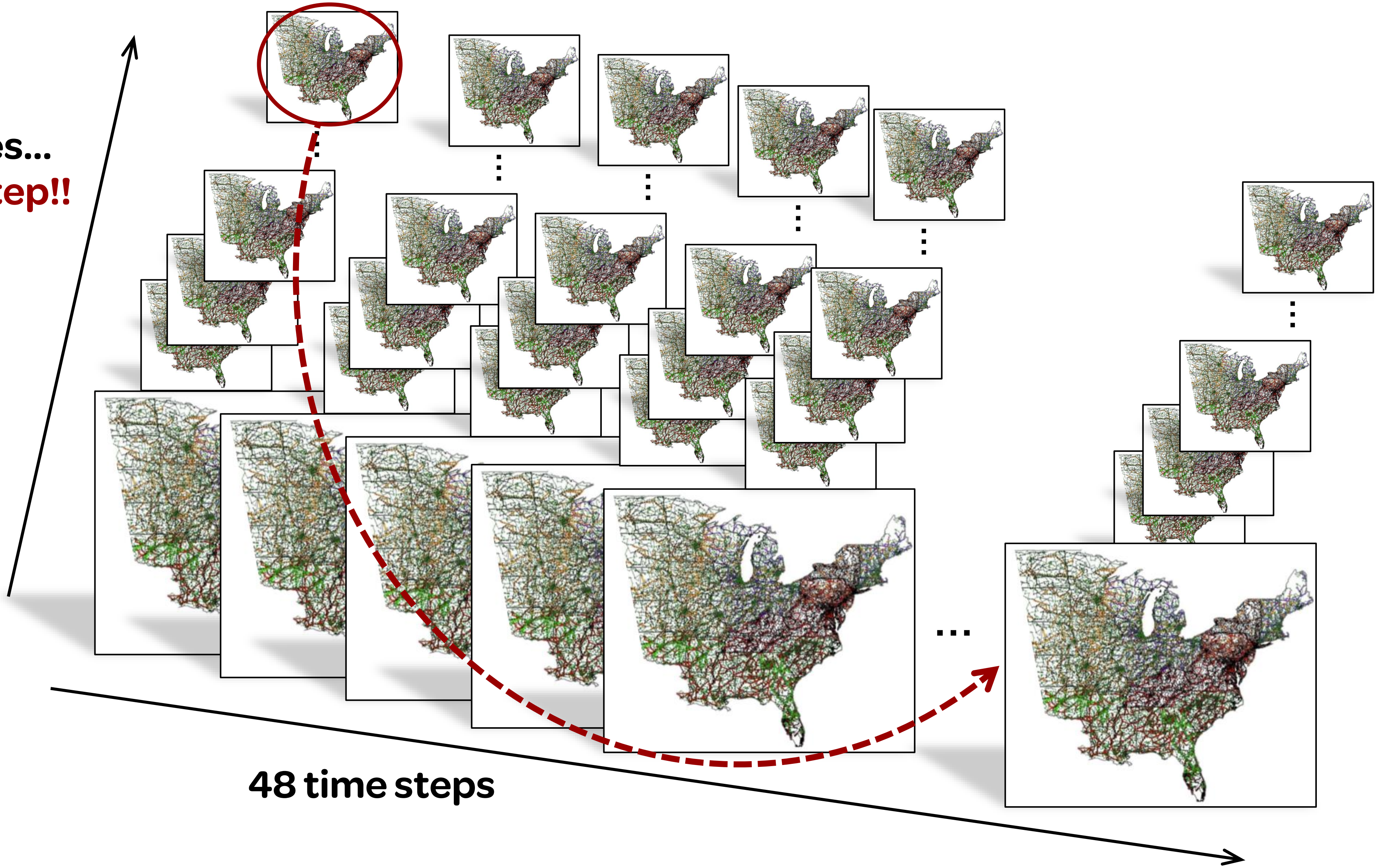
DTU Danmarks Tekniske Universitet

The UNIVERSITY of VERMONT

GRID OPTIMIZATION COMPETITION

GO3 – Mission Impossible

23,000+ contingencies...
at each time step!!



48 time steps

What is this paper?

Language Models are Few-Shot Learners

OpenAI

Abstract

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. While typically task-agnostic in architecture, this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions – something which current NLP systems still largely struggle to do. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, sometimes even reaching competitiveness with prior state-of-the-art fine-tuning approaches. Specifically, we train GPT-3, an autoregressive language model with 175 billion

→ $$\min_{\boldsymbol{x}} \ \mathcal{L}(\boldsymbol{x})$$

B Details of Model Training

To train all versions of GPT-3, we use Adam with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$, we clip the global norm of the gradient at 1.0, and we use cosine decay for learning rate down to 10% of its value, over 260 billion tokens (after 260 billion tokens, training continues at 10% of the original learning rate). There is a linear LR warmup over the first 375

Adam

- ✓ Tracks curvature – **good for NLPs!**
- ✓ **But, what about MI-N/LPs?**
- ✓ **But.. does Adam have enough citations??**

by DP Kingma · 2014 · Cited by 157139 –

- Almost 2 citations/minute

4th International Verification of Neural Networks Competition (VNN-COMP'23) →

CROWN

Winner of International Verification of Neural Networks Competitions (VNN-COMP 2021,2022)

We run our experiments on a machine with a single NVIDIA RTX 3090 GPU (24GB GPU memory), a AMD Ryzen 9 5950X CPU and 64GB memory. Our β-CROWN solver uses 1 CPU and 1 GPU only, except for the MLP models in Table 2 where 16 threads are used to compute intermediate layer bounds with Gurobi[2]. We use the Adam optimizer [21] to solve both $\hat{\alpha}$ and $\hat{\beta}$ in Eq. 12 with 20 iterations. The learning rates are set as 0.1 and 0.05 for optimizing $\hat{\alpha}$ and $\hat{\beta}$ respectively. We decay

Let's give Adam a try!

- C3E3N01576D1/scenario_027 (initialized with **copper plate ED**)
  - 1576 buses over 18 time periods
  - 147 contingencies at each time
  - Let's be clear: this is a "**baby**" system

Limitations of the Adam solver

(and how I overcame them)

Adam limitation 1: Adam doesn't know about constraints ☺

GO3 MI-NLP:

$$\begin{aligned}\min_{\boldsymbol{x}_d,\boldsymbol{x}_c} \quad & z^{\text{ms}}(\boldsymbol{x}_c,\boldsymbol{x}_d,\boldsymbol{y}) + z^{\text{ctg}}\\ \text{s.t.} \quad & \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}_c,\boldsymbol{x}_d)\\ & \boldsymbol{0} = \boldsymbol{h}_{\text{ctg}}(\boldsymbol{x}_c,\boldsymbol{x}_d,\boldsymbol{\theta}_k)\\ & A_c\boldsymbol{x}_c + A_d\boldsymbol{x}_d \geq \boldsymbol{0}\\ & \underline{\boldsymbol{x}}_c \leq \boldsymbol{x}_c \leq \overline{\boldsymbol{x}}_c\\ & \underline{\boldsymbol{x}}_d \leq \boldsymbol{x}_d \leq \overline{\boldsymbol{x}}_d\\ & \boldsymbol{x}_d \in \mathbb{Z}^{n_d}\\ & \boldsymbol{x}_c \in \mathbb{R}^{n_c}\end{aligned}$$

Solution: relax, penalize, reformulate, clip, project

$$\begin{aligned}\min_{\boldsymbol{x}_d,\boldsymbol{x}_c\in\mathcal{B}} \quad & z^{\text{ms}}(\boldsymbol{x}_c,\boldsymbol{x}_d,\boldsymbol{f}(\boldsymbol{x}_c,\boldsymbol{x}_d)) + z^{\text{ctg}}\\ & \qquad + \rho\cdot\sigma_s\left(A_c\boldsymbol{x}_c + A_d\boldsymbol{x}_d\right)\\ \text{s.t.} \quad & \boldsymbol{0} = \boldsymbol{h}_{\text{ctg}}(\boldsymbol{x}_c,\boldsymbol{x}_d,\boldsymbol{\theta}_k)\end{aligned}$$

General Goal: push everything up into the objective function

Adam limitation 1: Adam doesn't know about constraints ☺

- Ex1) slack reformulation

$$0 \leq s_{jt}^{+}$$
$$z_{jt}^{\mathrm{s}} = d_t c^{\mathrm{s}} s_{jt}^{+}$$
$$\sqrt{p_{jt}^2 + q_{jt}^2} \leq s_j^{\max} + s_{jt}^{+}$$

$$z_{jt}^{\mathrm{s}} = d_t c^{\mathrm{s}} \max((p_{jt}^2(\boldsymbol{x}) + q_{jt}^2(\boldsymbol{x}))^{1/2} - s_j^{\max}, 0)$$

- Ex2) auxiliary binary reformulation

$$u_{jt}^{\mathrm{su}} + u_{jt}^{\mathrm{sd}} \leq 1$$
$$u_{jt}^{\mathrm{on}} - u_{j,t-1}^{\mathrm{on}} = u_{jt}^{\mathrm{su}} - u_{jt}^{\mathrm{sd}}$$

$$u_{jt}^{\mathrm{su}} \triangleq +\max\left(u_{jt}^{\mathrm{on}} - u_{j,t-1}^{\mathrm{on}}, 0\right)$$
$$u_{jt}^{\mathrm{sd}} \triangleq -\min\left(u_{jt}^{\mathrm{on}} - u_{j,t-1}^{\mathrm{on}}, 0\right).$$

- Ex3) energy cost/value reformulation:

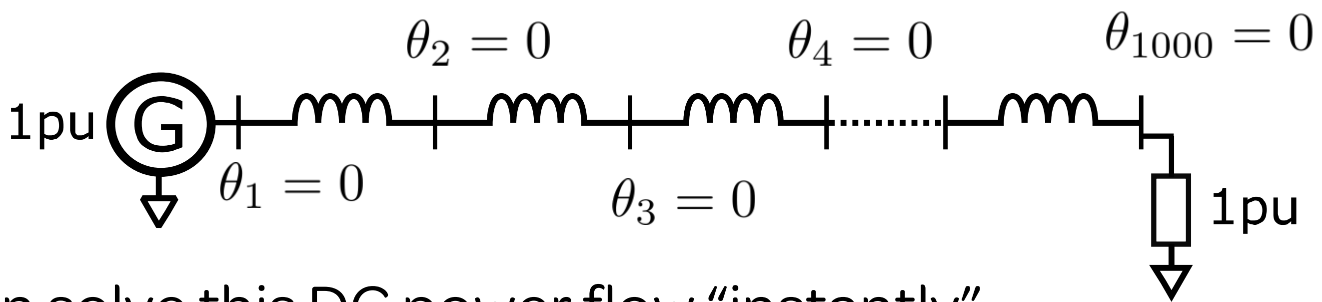
Lots of ReLUs!!!

$$0 \leq p_{jtm} \leq p_{jtm}^{\max}, \;\forall t \in T, j \in J^{\mathrm{pr,cs}}, m \in M_{jt}$$
$$p_{jt} = \sum_{m \in M_{jt}} p_{jtm}, \;\forall t \in T, j \in J^{\mathrm{pr,cs}}$$
$$z_{jt}^{\mathrm{en}} = d_t \sum_{m \in M_{jt}} c_{jtm}^{\mathrm{en}} p_{jtm}, \;\forall t \in T, j \in J^{\mathrm{pr,cs}}.$$

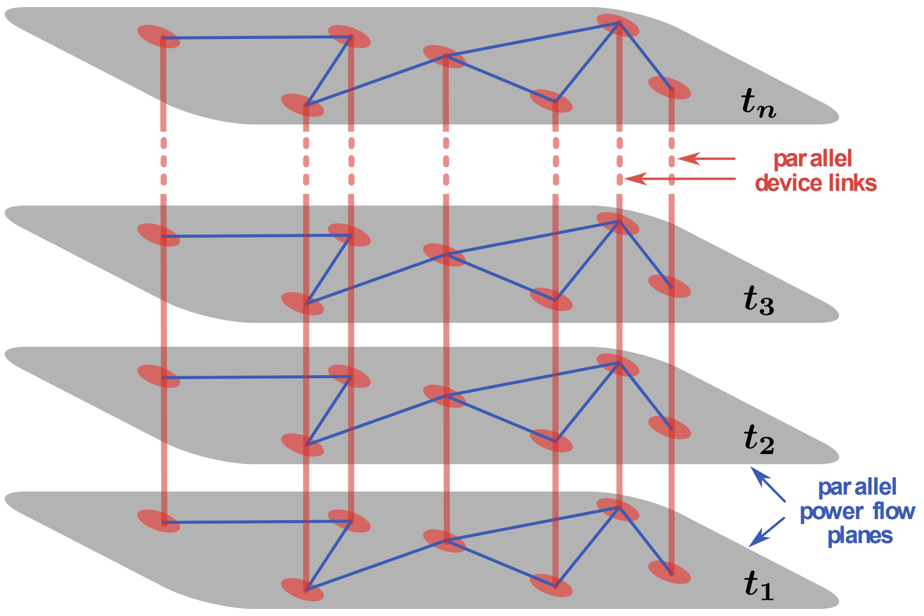
$$p_{jtm_L}^{\mathrm{cum,max}} = \sum_{l=1}^{L} p_{jtm_l}^{\max}$$
$$z_{jt}^{\mathrm{en}} = d_t \sum_{l=1}^{|M_{jt}|} c_{jtm_l}^{\mathrm{en}} \max\left(\min\left(p_{jt} - p_{jtm_{L=l-1}}^{\mathrm{cum,max}}, p_{jtm_l}\right), 0\right)$$

Adam limitation 2: Adam is <u>bad at</u> what "\" is <u>good at</u>

1. Gradient-based methods can be slow at trivial tasks!



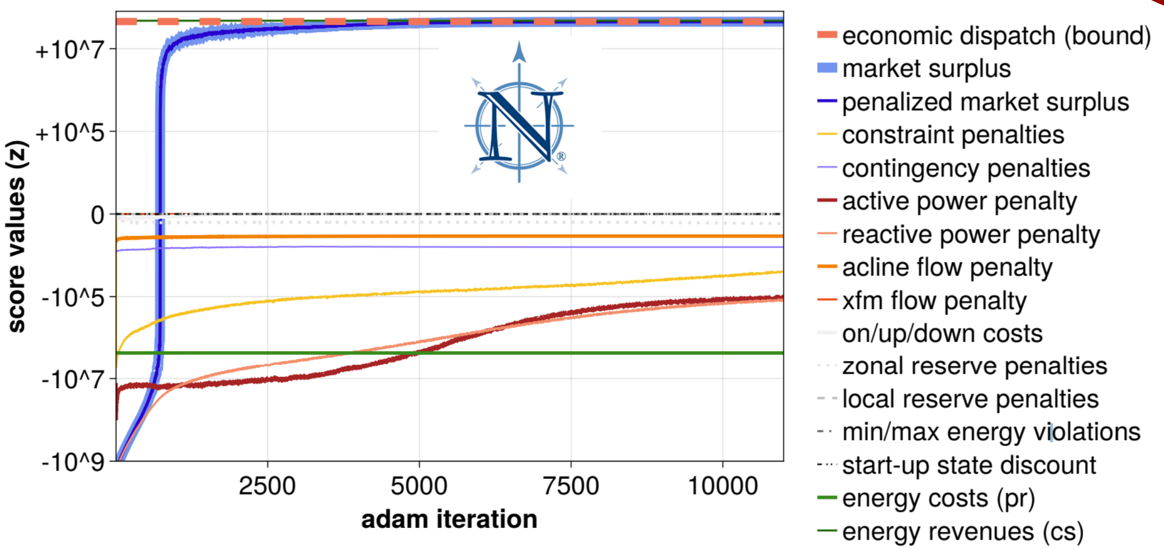
- Backslash can solve this DC power flow "instantly"
- Gradient methods will notice "pressure" on the boundaries, and then take a step
  - This pressure will slowly "**flow**" into the center of the circuit, until equilibrium is found –slow!!!
- **Solution:** parallel linearized power flow solves (across time) to "hot start" Adam



Adam limitation 3: Adam wants to be initialized!!!

- Gradient-based methods love to be hot started
  - They are very good at staying local!!! No barrier functions/parameters are flying off to infinity, and gradient steps are generally small

- **Solution:** initialize quasiGrad with a copper plate economic dispatch (LP)



Projection 2: Copper Plate Economic Dispatch [LP]

⋆ ***optionally** parallelizable across time instances*

$$\max_{\boldsymbol{x}_c, \boldsymbol{x}_d} \quad z^{\mathrm{ms}}$$

s.t. [9, eqs. (1)-(163)] (nominal GO3 formulation)

neglect:

- shunts, contingencies, integers (LP relax)
- all network variables $(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\tau}, \boldsymbol{\phi})$ and flow limits

impose:

- $\sum_{j \in J^{\mathrm{pr}}} p_{jt} = \sum_{j \in J^{\mathrm{cs}}} p_{jt} + \sum_{j \in J^{\mathrm{dc}}} p_{jt}^{\mathrm{fr/to}}, \ \forall t$ (p balance)
- $\sum_{j \in J^{\mathrm{pr}}} q_{jt} = \sum_{j \in J^{\mathrm{cs}}} q_{jt} + \sum_{j \in J^{\mathrm{dc}}} q_{jt}^{\mathrm{fr/to}}, \ \forall t$ (q balance)

Adam limitation 4: Adam doesn't know about integers/binaries

The central premise of my quasiGrad solver:

- ~~***Solve parallel GO problems with various integer relaxations, thus branching-and-bounding over a massive number of binaries and arriving at the optimal MINLP solution.***~~ ✗

- ***Solve penalized GO formulations with relaxed integers which are <u>sequentially rounded</u>, until all integers are fixed.*** ✓
  - *Solve NLP with 100% binaries relaxed*
    - *Feasibly project and fix the 25% which are closest to 1 or 0*
  - *Solve NLP with 75% binaries relaxed*
    - *Feasibly project and fix the 25% which are closest to 1 or 0*
  - *Solve NLP with 50% binaries relaxed*
    - *Feasibly project and fix the 10% which are closest to 1 or 0*
  - *Solve NLP with 60% binaries relaxed … etc.*

} Similar to Integer batch rounding (IBR)

Adam limitation 4: Adam doesn't know about integers/binaries

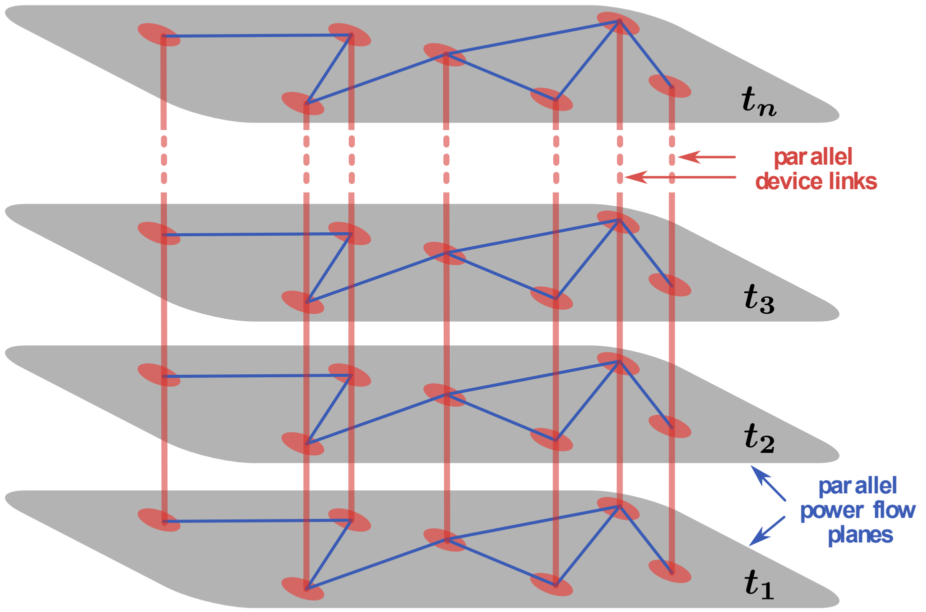
Objective: stay ***as close as possible*** *to the Adam NLP solution:*

Projection 1: Optimal Device Binary Projection [MILP]
⋆ *parallelizable across each device*

$$\min_{\boldsymbol{x}_c \in \mathbb{R}, \boldsymbol{x}_d \in \mathbb{Z}} \quad \left\| D_c^{g_i} \left(\boldsymbol{x}_c - \boldsymbol{x}_c^0 \right) \right\|_1 + \left\| D_d^{g_i} \left(\boldsymbol{x}_d - \boldsymbol{x}_d^0 \right) \right\|_1$$

s.t. $\boldsymbol{x}_{d,i} = \boldsymbol{x}_{d,i}^0, \; i \in \mathcal{F}$ (***fixed binaries***)
[9, eqs. (48)-(58)] (binary constraints)
[9, eqs. (68)-(74)] (ramp limits)
[9, eq. (98)-(108)] (reserve constraints)
[9, eq. (109)-(118)] (producer limits)
[9, eq. (119)-(128)] (consumer limits)

(Show projection)



Adam limitation 5: Adam needs a huge number of steps

- Gradient-computation is the bottleneck (show proof)

take this off

- **Solution:** backpropagation needs to be **hyper efficient**

$$\boldsymbol{g} = \nabla_{\boldsymbol{x}} \left(z^{\text{ms}}(\boldsymbol{x}) + \rho \cdot \sigma_s \left(A\boldsymbol{x} \right) \right) + \nabla_{\boldsymbol{x}} \boldsymbol{h}_{\text{ctg}}$$

- **All gradients in the quasiGrad solver are computed by hand** (example):

$$\nabla_{v_{it}} p_{jt}^{\text{fr}} = 2u_{jt}^{\text{on}} \left(\left(g_j^{\text{sr}} + g_j^{\text{fr}} \right) v_{it}/\tau_{jt}^2 + \left(-g_j^{\text{sr}} \cos\left(\theta_{it} - \theta_{i't} - \phi_{jt}\right) - b_j^{\text{sr}} \sin\left(\theta_{it} - \theta_{i't} - \phi_{jt}\right) \right) v_{i't}/\tau_{jt} \right)$$

$$\nabla_{v_{i't}} p_{jt}^{\text{fr}} = u_{jt}^{\text{on}} \left(\left(-g_j^{\text{sr}} \cos\left(\theta_{it} - \theta_{i't} - \phi_{jt}\right) - b_j^{\text{sr}} \sin\left(\theta_{it} - \theta_{i't} - \phi_{jt}\right) \right) v_{it}/\tau_{jt} \right)$$

- **Backpropagation is computed entirely by hand**
  - Example -- chain rule, from network variable to overload penalty to market surplus:

$$\nabla_x z^{\text{ms}} = \nabla_{z^{\text{base}}} z^{\text{ms}} \cdot \nabla_{z_t^{\text{t}}} z^{\text{base}} \cdot \nabla_{z_{jt}^{\text{s}}} z_t^{\text{t}} \cdot \nabla_{s_{jt}^{+}} z_{jt}^{\text{s}} \cdot \nabla_{s_{jt}^{\text{fr/to},+}} s_{jt}^{+} \cdot \nabla_{p/q_{jt}^{\text{fr/to},+}} s_{jt}^{\text{fr/to},+} \cdot \nabla_{x_{\text{lf}}} p/q_{jt}^{\text{fr/to},+},$$

$$x_{\text{lf}} \in \left\{ v_{it}, v_{i't}, \theta_{it}, \theta_{i't}, \tau_{jt}, \phi_{jt}, u_{jt}^{\text{on}} \right\}$$

Adam limitation 5: Adam needs a huge number of steps

- Type-stability. Memory pre-allocation. Also... multithreading
- Computers are dumb – the following will run sequentially on a single CPU thread:

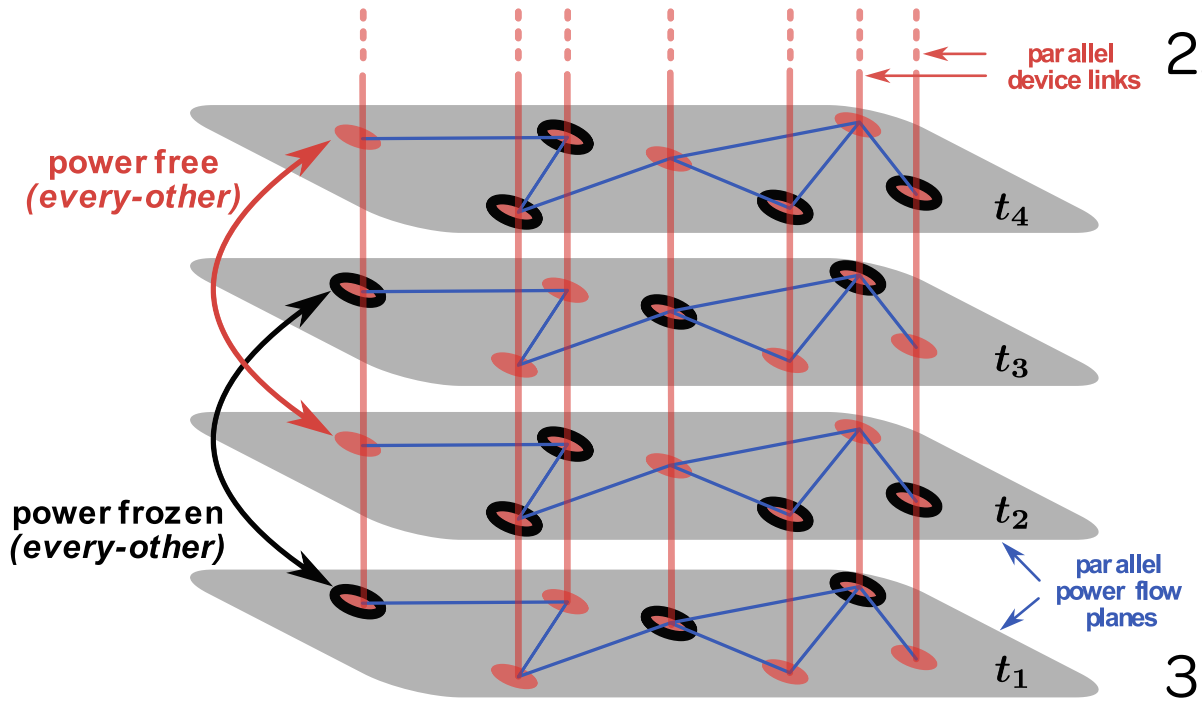
$$
\begin{aligned}
&\textbf{for } t \in T \\
&\qquad \nabla_{\boldsymbol{v}_t^{\mathrm{to}}} \boldsymbol{p}_t^{\mathrm{fr}} = \left(-\boldsymbol{g}^{\mathrm{sr}} \cos\left(\boldsymbol{\delta}\right) - \boldsymbol{b}^{\mathrm{sr}} \sin\left(\boldsymbol{\delta}\right)\right) \boldsymbol{v}_t^{\mathrm{fr}} / \boldsymbol{\tau}_t \\
&\textbf{end}
\end{aligned}
$$

- Humans are smart – we can tell (Julia) to compute these derivatives on parallel threads:

$$
\begin{aligned}
&\texttt{Threads.@threads } \textbf{for } t \in T \\
&\qquad \nabla_{\boldsymbol{v}_t^{\mathrm{to}}} \boldsymbol{p}_t^{\mathrm{fr}} = \left(-\boldsymbol{g}^{\mathrm{sr}} \cos\left(\boldsymbol{\delta}\right) - \boldsymbol{b}^{\mathrm{sr}} \sin\left(\boldsymbol{\delta}\right)\right) \boldsymbol{v}_t^{\mathrm{fr}} / \boldsymbol{\tau}_t \\
&\textbf{end}
\end{aligned}
$$

Adam limitation 6: Adam solution needs to be "cleaned up"

- How do we run the final, ramp-constrained power flow clean-up? Backtracking?
  - Assume the generators are initially ramp-feasible, but power balance needs a cleanup



1. Separate devices in two groups (**a** and **b**)
2. Enforce the following at each time step:

$$
\begin{aligned}
t_1 :& \left[\boldsymbol{p}_0 + \boldsymbol{d}_1^{\mathrm{rd}} \leq \boldsymbol{p}_1 \leq \boldsymbol{p}_0 + \boldsymbol{d}_1^{\mathrm{ru}}\right]_a, \quad \left[\boldsymbol{p}_1 = \boldsymbol{p}_1^0\right]_b, \\
& \left[\boldsymbol{p}_1 + \boldsymbol{d}_2^{\mathrm{rd}} \leq \boldsymbol{p}_2 \leq \boldsymbol{p}_1 + \boldsymbol{d}_2^{\mathrm{ru}}\right]_a, \\
t_2 :& \left[\boldsymbol{p}_1 + \boldsymbol{d}_2^{\mathrm{rd}} \leq \boldsymbol{p}_2 \leq \boldsymbol{p}_1 + \boldsymbol{d}_2^{\mathrm{ru}}\right]_b, \quad \left[\boldsymbol{p}_2 = \boldsymbol{p}_2^0\right]_a, \\
& \left[\boldsymbol{p}_2 + \boldsymbol{d}_3^{\mathrm{rd}} \leq \boldsymbol{p}_3 \leq \boldsymbol{p}_2 + \boldsymbol{d}_3^{\mathrm{ru}}\right]_b, \\
& \vdots
\end{aligned}
$$

3. Power flow problems can be solved in parallel with ***guaranteed*** ramp-rate feasibility

Adam limitation 6: Adam solution needs to be “cleaned up”

- This trick, probably, is what allowed quasiGrad to find 3/6 feasible solutions on the 23k system:

model	scenario	ARPA-e Benc	Artelys_Columbi	Electric-Sta	Gatorgar	GOT-BSI-OPF	GravityX	LLGoMax	Occams razor	PACE	PGWOpt	quasiGrad	The Blackouts	TIM-GO	YongOptimization
C3E4N23643D1	**3**	61,152,061	63,340,112	0	0	99,123,868	0	0	0	0	0	48,878,908	0	104,686,125	104,073,565
C3E4N23643D1	4	0	70,444,810	0	0	91,584,129	0	0	0	0	0	0	0	39,494,965	69,793,156
C3E4N23643D2	**3**	0	353,804,820	0	0	589,696,576	0	0	0	0	0	598,132,224	0	588,991,290	600,577,211
C3E4N23643D2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3E4N23643D3	**3**	0	1,162,674,657	0	0	2,093,288,847	0	0	0	0	0	2,143,434,855	0	2,112,949,852	2,158,212,496
C3E4N23643D3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C3E4N23643	**6**	**61,152,061**	**1,650,264,399**	**0**	**0**	**2,873,693,419**	**0**	**0**	**0**	**0**	**0**	**2,790,445,988**	**0**	**2,846,122,232**	**2,932,656,427**
C3E4N23643D1	**2**	61,152,061	133,784,922	0	0	190,707,997	0	0	0	0	0	48,878,908	0	144,181,090	173,866,721
C3E4N23643D2	**2**	0	353,804,820	0	0	589,696,576	0	0	0	0	0	598,132,224	0	588,991,290	600,577,211
C3E4N23643D3	**2**	0	1,162,674,657	0	0	2,093,288,847	0	0	0	0	0	2,143,434,855	0	2,112,949,852	2,158,212,496

Adam limitation 7: Adam shouldn't "solve" contingencies

- Contingency penalties have a closed-form solution, involving a matrix inverse (PTDF)
- My strategy: at each iterative step of Adam, **(i)** evaluate a subset of contingencies, **(ii)** backpropagate through the worst of them, and then **(iii)** pass the gradients to Adam
  - **(i)** We solve contingencies by using an iterative solver (preconditioned conjugate gradient, or pcg) to solve the "base-case" DC power flow solution:

$$\hat{P} \leftarrow \mathtt{LLDL}\left(\hat{E}^T Y_x \hat{E}\right)$$

$$\hat{\boldsymbol{\theta}}_{tb} \approx \mathbf{pcg}(\hat{\boldsymbol{p}}_t^{\text{inj}} - \hat{E}^T \boldsymbol{b}_t, \hat{Y}_b, \hat{P}, \epsilon_{\text{pcg}})$$

  - By Sherman-Morrison-Woodbury, a rank-1 correction finds the contingency solution:

$$\hat{\boldsymbol{\theta}}_{tk} = \hat{\boldsymbol{\theta}}_{tb} - \boldsymbol{u}_k(\boldsymbol{w}_k^T \tilde{\boldsymbol{p}}_t^{\text{inj}})$$

  - With phase angles, we may now score a given contingency – worst offenders are tracked

Adam limitation 7: Adam shouldn't "solve" contingencies

- **(ii)** If a contingency score is high enough, we backpropagate through it – how???
  - (This is expensive, so we dynamically skip low scoring contingencies)
  - Say the contingency score is a nonlinear function of flows:

$$z_{tk}^{\text{ctg}} = f(\boldsymbol{p}_{tk}) \qquad \rightarrow \quad \nabla_{\boldsymbol{p}_{tk}} z_{tk}^{\text{ctg}} = \boldsymbol{d}_k$$

$$\boldsymbol{p}_{tk} = Y_{x,k}\hat{E}\hat{Y}_k^{-1}\hat{\boldsymbol{p}}_t^{\text{inj}}$$

  - We want the derivative of contingency score with respect to nodal injections:

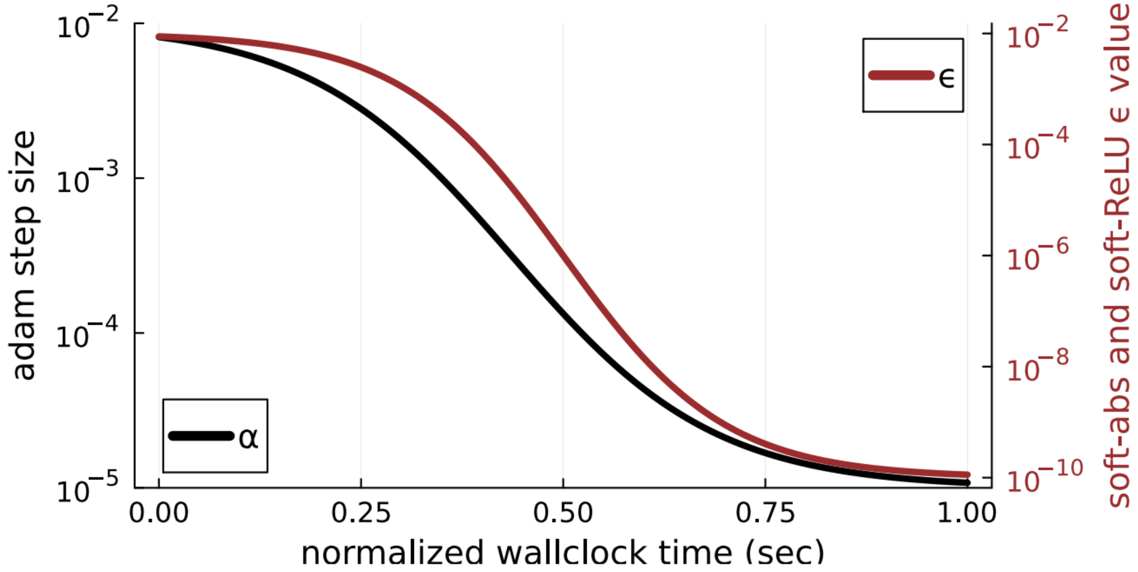
$$\nabla_{\hat{\boldsymbol{p}}_t^{\text{inj}}} z_{tk}^{\text{ctg}} = \left(Y_{x,k}\hat{E}\hat{Y}_k^{-1}\right)^T \boldsymbol{d}_k = \hat{Y}_k^{-1}\hat{E}^T Y_{x,k}\boldsymbol{d}_k$$

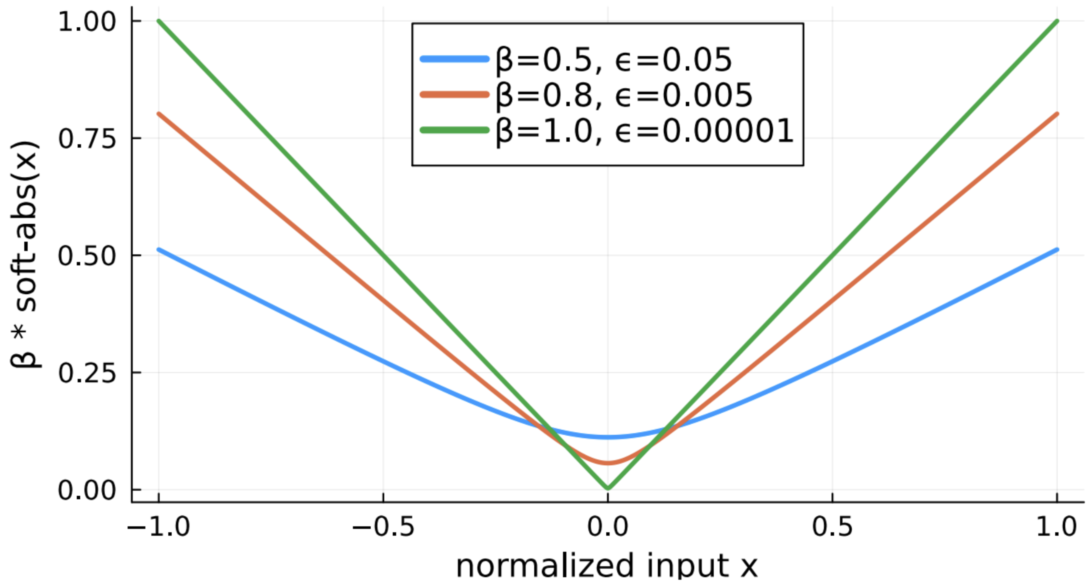
  - This gradient is then applied to all variables which affect all nodal injections! These gradients are filtered and given to Adam.

Adam limitation 8: Adam needs explicit step-size decay***

- Adam converges much more efficiently if step-size is explicitly decayed
- quasiGrad decays steps and iteratively tightens constraint/balance penalties

$$10^6|z_p| \;\rightarrow\; \beta 10^6\sqrt{z_p^2+\epsilon^2}$$





Let's check on our d1 score!

More generally:

Best Result:

Rank	Team Ensemble	Division 2 Score 163,579,841,300	$k
1	GOT-BSI-OPF	162,941,475,726	100
2	TIM-GO	162,270,256,651	90
3	YongOptimization	160,165,088,341	80
4	Artelys_Columbia	157,359,267,058	70
5	GravityX	156,131,225,903	60
	ARPA-e Benchmark	156,014,230,887	
	quasiGrad	155,168,735,676	
	Occams razor	145,494,618,835	
	Electric-Stampede	139,357,283,507	
	LLGoMax	116,812,192,654	
	The Blackouts	114,098,832,983	
	Gatorgar	10,263,109,863	

Worst Result:

Team	1
ARPA-e Benchmark	0
Artelys_Columbia	29
Electric-Stampede	0
Gatorgar	3
GOT-BSI-OPF	47
GravityX	105
LLGoMax	3
Occams razor	4
quasiGrad	0
The Blackouts	56
TIM-GO	89
YongOptimization	331

In Conclusion

Jesse Holzer Carleton Coffrin Christopher DeMarco Ray Duthu
Stephen Elbert Brent Eldridge Tarek Elgindy Scott Greene
Nongchao Guo Elaine Hale Bernard Lesieutre Terrence Mak
Colin McMillan Hans Mittelmann Hyungseon Oh
Richard O'Neill Thomas Overbye Bryan Palmintier
Farnaz Safdarian Ahmad Tbaileh Pascal Van Hentenryck
Arun Veeramany Jessica Wert

thank you!

quasiGrad stands on the shoulders of giants

- Contingency evaluation via rank-1 perturbations [**Jess Holzer**]
- Iterative Batch Rounding (IBR) and general inspiration [**Hassan Hajazi**]
- Sparse Jacobian construction/updates [**Bolognani/Dörfler**]
- Contingency selection based on real-time computations [**Baker/Boulder**]
- Scalar homotopy factor for tightening penalty relaxations [**Amrit Pandey/CMU**]
- GO benchmark: PowerModelsSecurityConstrained.jl [**Carelton (et al?)**]
- Distributed slack for device-constrained power flow [**Sairaj Dhople, et al.**]
- JuMP.jl, IterativeSolvers.jl, Gurobi(.jl), LoopVectorization.jl, LimitedLDLFactorizations.jl
- **More general thank-you:** Dan Molzahn, Spyros Chatzivasileiadis, Amrit Pandey, Mads Almassalkhi, the PNNL team

Learn More:

- Recent PSCC Submission: https://arxiv.org/abs/2310.06650
- Supplemental Information: https://samchevalier.github.io/docs/SI.pdf
- quasiGrad.jl: https://github.com/SamChevalier/quasiGrad

```
Pkg.add(url="https://github.com/SamChevalier/quasiGrad")
```

```
using BenchmarkTools
using quasiGrad
using Revise
```