Making your Theory-to-Practice Work:

Online-to-Batch via Schedules & Schedule-Free Learning

Aaron Defazio

Research Scientist

FAIR, Meta Superintelligence Labs

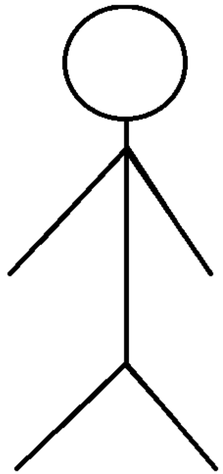
Meta

THE IDEAL WORLD

(THEORY DRIVEN FRONTIER RESEARCH)

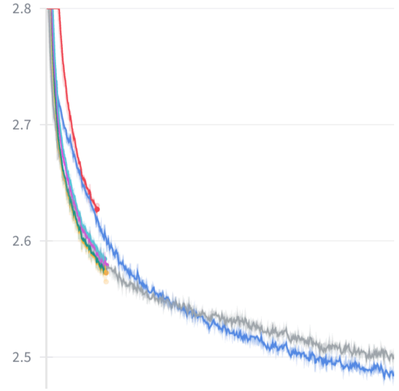
ME

(OPTIMIZATION RESEARCHER IN INDUSTRY)



COME UP WITH NEW THEORY-DRIVEN ALGORITHMIC INNOVATIONS

IMPLEMENT THEM EXACTLY … THEY WORK!



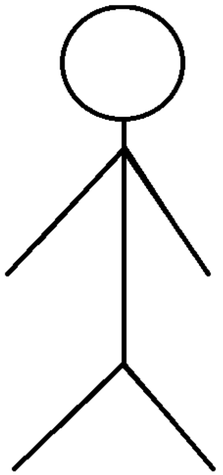
Why shouldn’t stochastic/online optimization algorithms actually work out of the box, exactly as the theory prescribes?

THE STATUS QUO

(THEORY-ISH DRIVEN FRONTIER RESEARCH)

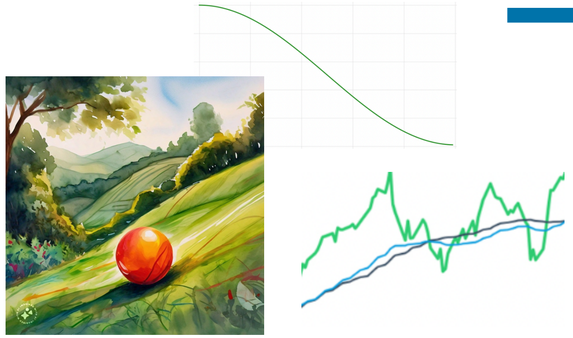
ME

(OPTIMIZATION RESEARCHER IN INDUSTRY)

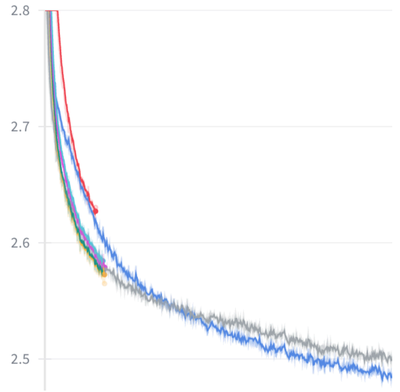


COME UP WITH NEW THEORY-DRIVEN ALGORITHMIC INNOVATIONS

LAYER A BUNCH OF HACKS ON TOP, ADD EMAS, ADD MOMENTUM, AND IGNORE THEORY-RECOMMENDED HYPERPARAMETERS



(SOMETIMES) KINDA WORKS

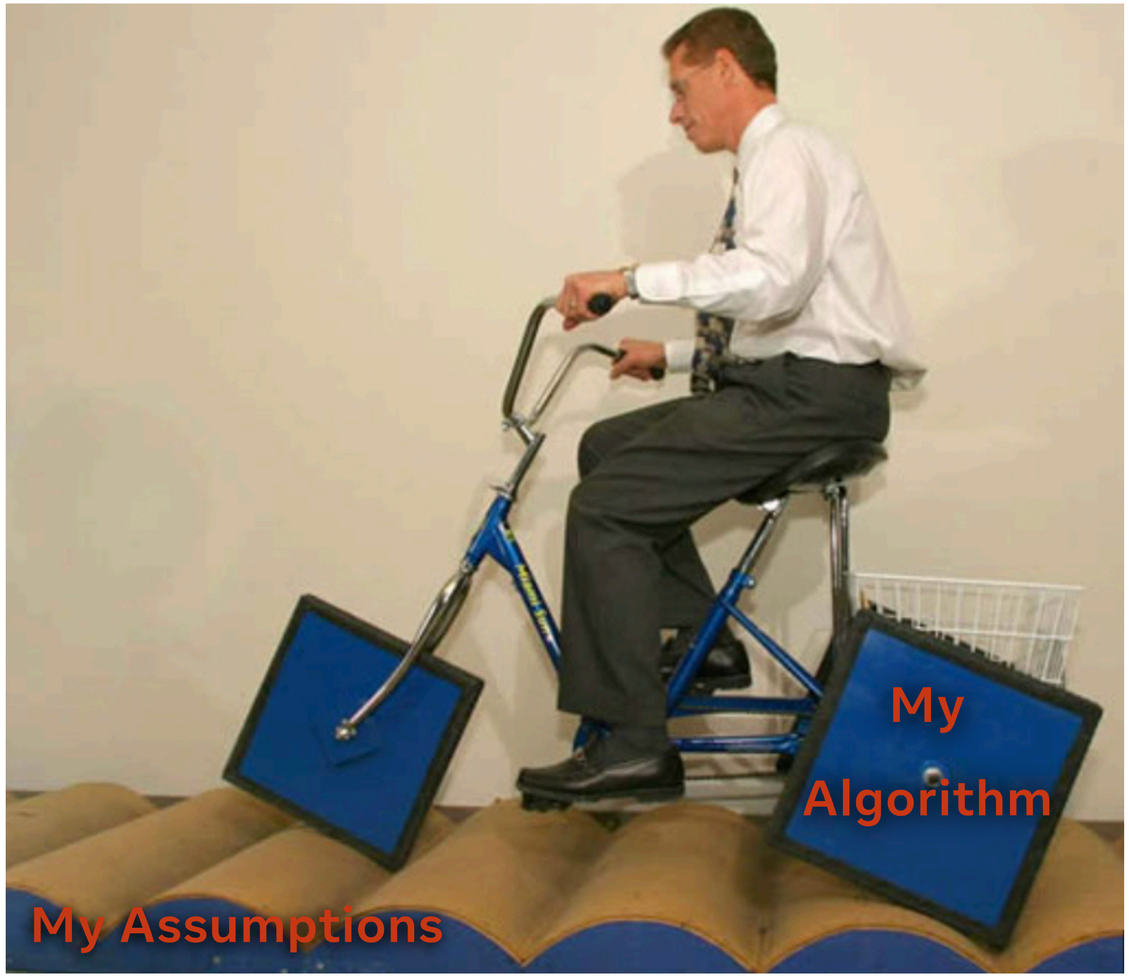


Aaron, you're too optimistic!

Why should convex optimization driven algorithms work on highly non-convex, non-smooth, unstable, low-precision, chicken-wire-and-duct-tape deep learning models with billions of parameters???

My theory is filled with naive assumptions and unrealistic simplifications!

If you could just develop methods with fewer assumptions, with more generality, maybe, just **maybe** they will work!



We are all thinking it...

THIS TALK

If you adopt a few recent innovations,

you can apply your theory-driven research in online learning to large-scale deep learning problems!

AND IT WORKS

THE TRICKS

1) Use **modern** online-to-batch conversions:

- Linear-Decay schedules
- Schedule-Free Learning

Using Polyak Averaging, or returning the last iterate **DOES NOT WORK WELL without many hacks**

2) Handle normalization layers and weight-decay properly

- Weight decay probably breaks your method

3) Use the right assumptions

- Bounded Gradients/Variance. Deep learning problems are **not** strongly convex, and have **negligible** usable smoothness.

PART 1) THEORY/PRACTICE MISMATCH

Classical theory for SGD says we should

- Return the average iterate (Polyak averaging)

$$x_{t+1} = x_t - \gamma_t g_t$$

$$\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t$$

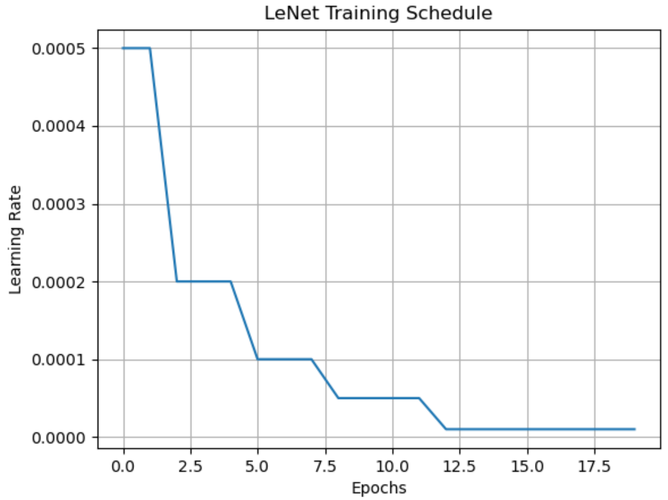
- Decrease the learning rate at a 1/sqrt(t) rate or use a flat LR

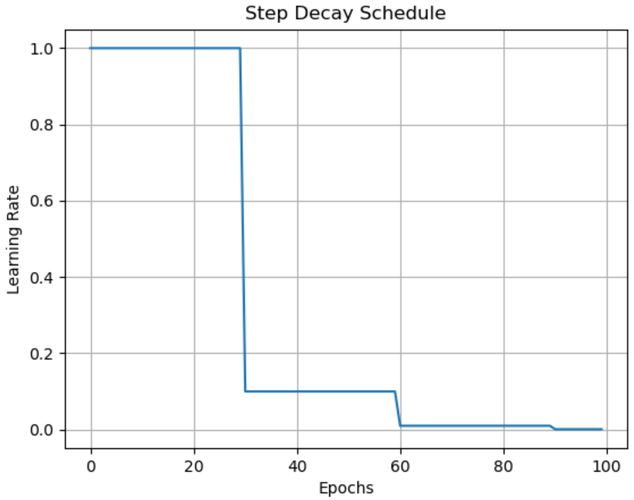
$$\gamma_t = \frac{D}{G\sqrt{t}} \quad \gamma_t = \frac{D}{G\sqrt{T}}$$

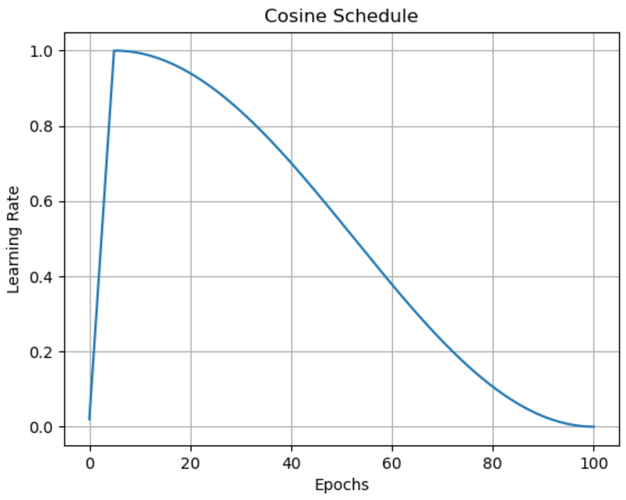
D = bound on domain diameter G = bound on gradient norm

Learning rate schedules suggested by classical theory **look nothing like** the best-performing schedules used by experimentalists

The average iterate also gives terrible results in practice!







1989 - 1998

Handwritten Digit Recognition with a Back-Propagation Network

Y. Le Cun, B. Boser, J. S. Denker, D. Henderson,
R. E. Howard, W. Hubbard, and L. D. Jackel
AT&T Bell Laboratories, Holmdel, N. J. 07733

Gradient-Based Learning Applied to Document Recognition

YANN LECUN, MEMBER, IEEE, LÉON BOTTOU, YOSHUA BENGIO, AND PATRICK HAFFNER

2012

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

"we adjusted manually throughout training. The heuristic which we followed was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate"

2017

Published as a conference paper at ICLR 2017

SGDR: STOCHASTIC GRADIENT DESCENT WITH WARM RESTARTS

Ilya Loshchilov & Frank Hutter
University of Freiburg
Freiburg, Germany,
{ilya,fh}@cs.uni-freiburg.de

A different perspective on Scheduling: Schedules are just an online-to-batch conversion!

$$x_{t+1} = x_t - \gamma_t g_t$$

$$\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t$$

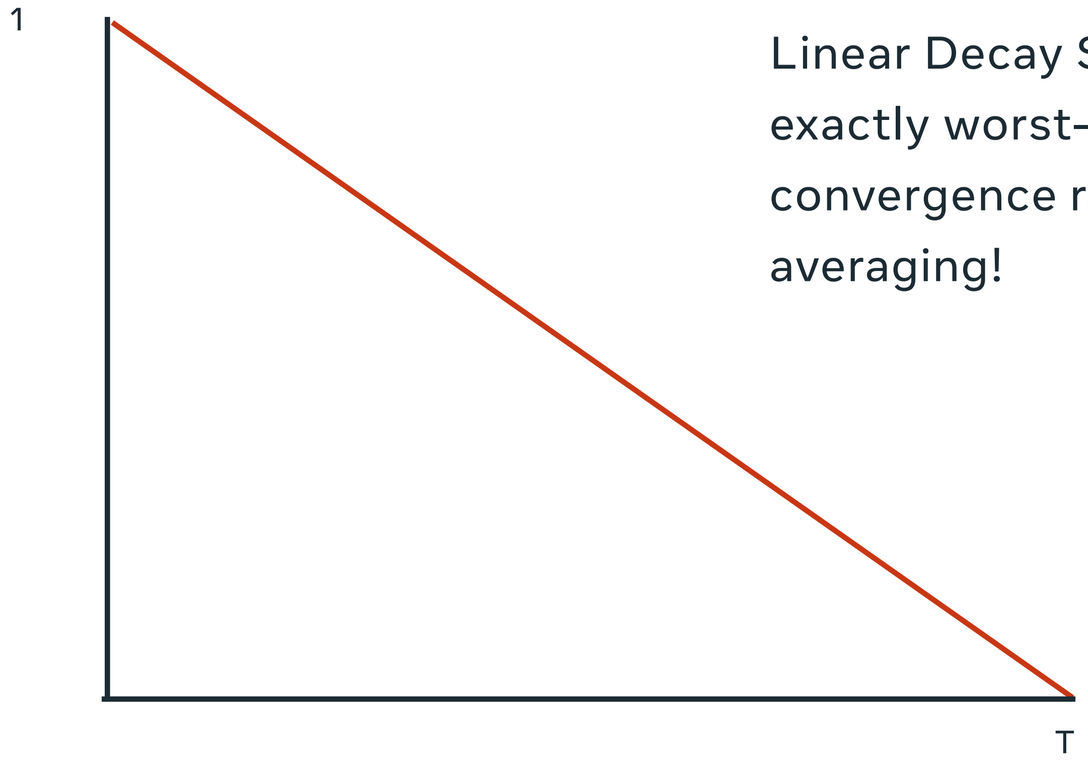
The schedules used by experimentalists are not replacing this $D/G\sqrt{T}$ part!

They are actually replacing **averaging.**

High-Performance schedules arise naturally from **theory** by analyzing the last iterate x_T rather than $\bar{x}_T$

Theoretically Optimal Schedules
(for convex problems)

$$\gamma_t = \frac{D}{G\sqrt{T}} \cdot \left(1 - \frac{t}{T}\right)$$



Linear Decay Schedule

Linear Decay Schedules give exactly worst-case optimal convergence rates without averaging!

$$f(\bar{x}_T) - f_* \leq \frac{DG}{\sqrt{T}}$$

EXACT CONVERGENCE RATE OF THE LAST ITERATE IN SUBGRADIENT METHODS

MOSLEM ZAMANI* AND FRANÇOIS GLINEUR †

Optimal Linear Decay Learning Rate Schedules and Further Refinements

---

Aaron Defazio
Fundamental AI Research Team, Meta

Ashok Cutkosky
Boston University

Harsh Mehta
Google Research

Konstantin Mishchenko
Samsung AI Center

Theorem 1. *Suppose $z_1, \ldots, z_T$ is some arbitrary sequence of vectors. Let $w_1, \ldots, w_T$ be an arbitrary sequence of non-negative numbers. Recall that we define $\Delta_t = z_{t+1} - z_t$ and $x_1 = z_1$. For $t \geq 1$, suppose x_{t+1} satisfies:*

$$x_{t+1} = x_t + \frac{w_{t+1:T}}{w_{1:T}} \Delta_t,$$

then for any x_*:

$$\mathbb{E}[f(x_T) - f_*] \leq \mathbb{E}\left[\sum_{t=1}^{T} \frac{1}{w_{1:T}} \langle w_t \cdot g_t, z_t - x_* \rangle\right].$$

Let us take a moment to consider the implications of this Theorem in the simplest setting of $w_t = 1$ for all t. In this case, it is well-known that by setting $\Delta_t = -\eta g_t$ for $\eta = \frac{D}{G\sqrt{T}}$, one guarantees $\sum_{t=1}^{T} \langle g_t, z_t - x_* \rangle \leq DG\sqrt{T}$. Thus, we immediately obtain the following important corollary:

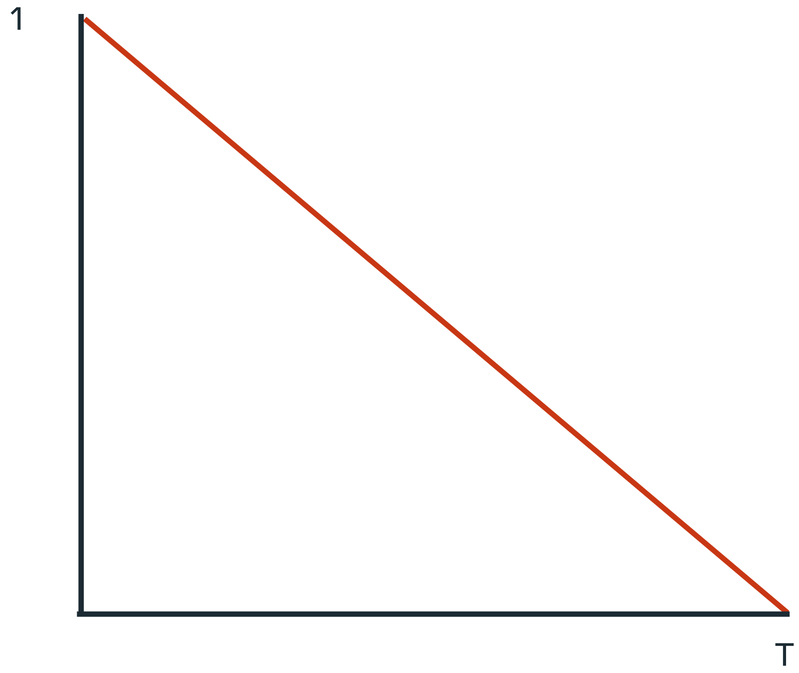
Corollary 2. *Set $x_{t+1} = x_t - \frac{D}{G\sqrt{T}} \left(1 - \frac{t}{T}\right) g_t$, then:*

$$\mathbb{E}[f(x_T) - f_*] \leq \frac{DG}{\sqrt{T}}.$$

I.e. prove convergence for a fixed step size version of your online learning method (z updates),

then get **last iterate convergence** for stochastic setting using linear decay step sizes for free!

Linear Decay (kinda) emulates Averaging



$$\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t$$

Gradient from t=1 appears in all terms in the average: **weight 1**

Gradient from t=T/2 appears in half the terms in the in the average: **weight 1/2**

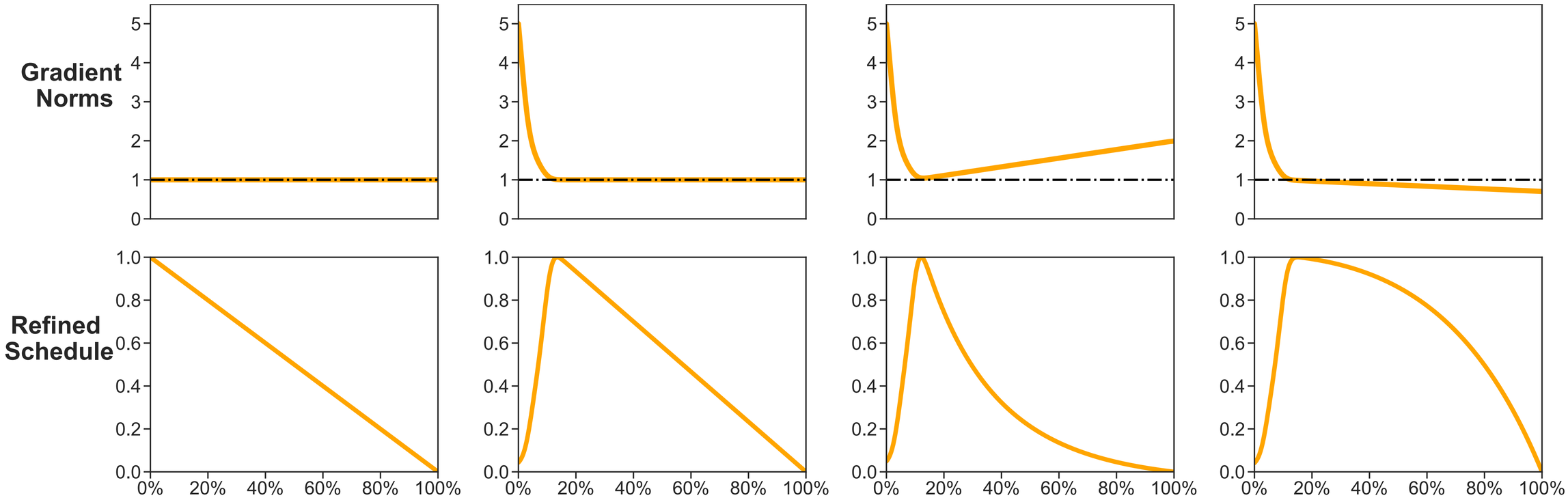
Gradient from t=3T/4 appears in 1/4 of the terms in the in the average: **weight 1/4**

.... same weighting as for linear decay

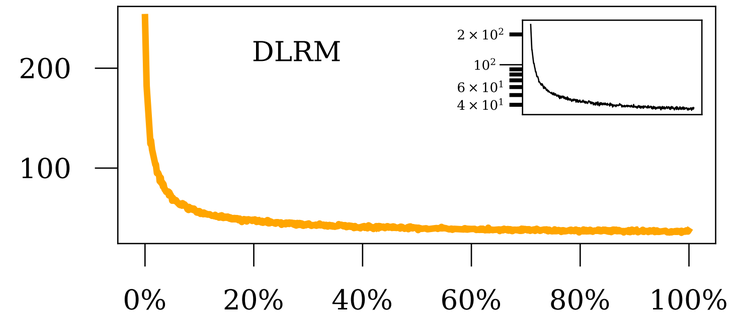
Our theory explains learning rate warmup!

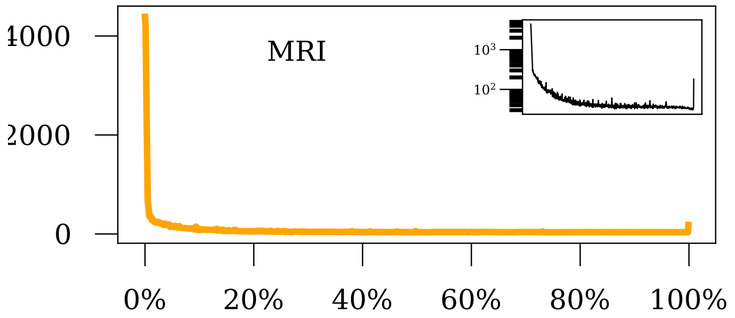
If you carefully optimize the upper bound we derive, warmup naturally arises

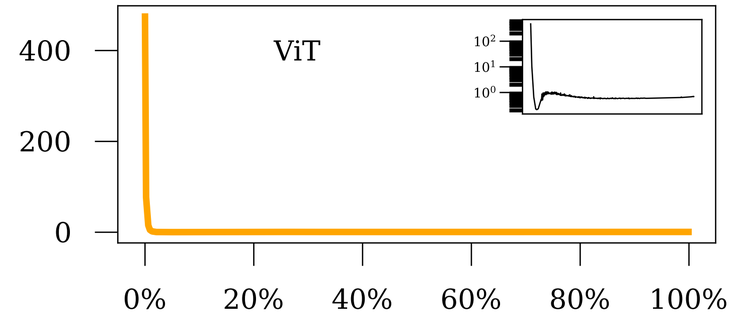
The behavior of the gradient norm sequence during training dictates how the schedule should look

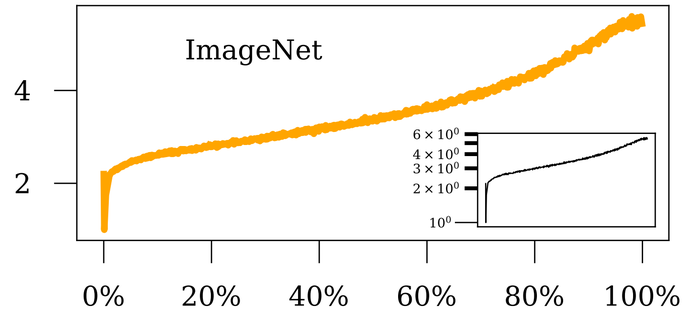


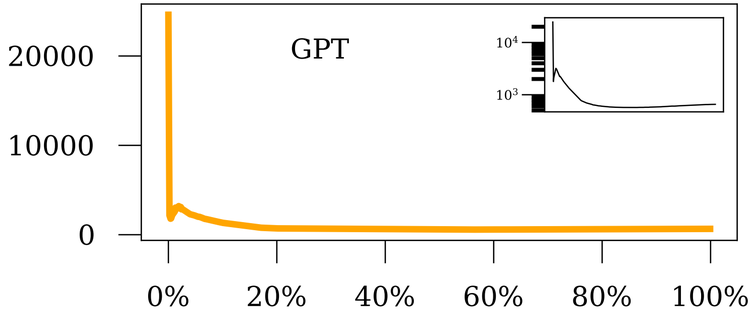
The initial G-norm drop is common on ML problems

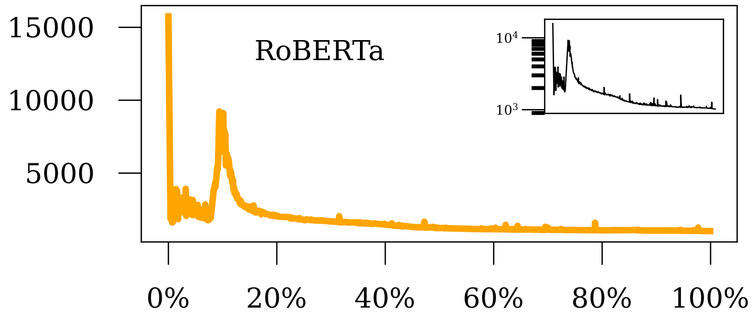












Beyond Schedules

Schedule-Free Learning - closing the theory/practice gap

- An alternative to schedules that doesn't need to know the stopping time T in advance
- Maintains the same **theoretically optimal** rate of convergence as linear decay schedules and Polyak averaging
- Works as well and often better than cosine or linear decay schedules!

The Road Less Scheduled

Aaron Defazio[1]
Fundamental AI Research Team, Meta

Xingyu (Alice) Yang[2]
Fundamental AI Research Team, Meta

Harsh Mehta
Google Research

Konstantin Mishchenko
Samsung AI Center

Ahmed Khaled
Princeton University

Ashok Cutkosky[3]
Boston University

[1]Research Co-lead [2] Engineering Co-lead [3] Senior Author

Schedule-Free Learning Paradigm

Interpolation beta=0.9

(A kind of momentum)

$$y_t = (1-\beta)z_t + \beta x_t$$

Base optimizer update

$$z_{t+1} = z_t - \gamma \nabla f(y_t)$$

$$x_{t+1} = \left(1 - \frac{1}{t+1}\right) x_t + \frac{1}{t+1} z_{t+1}$$

Running Average

Equivalent to:

$$x_t = \frac{1}{t} \sum_{i=1}^{t} z_i$$

Schedule-Free does momentum in a different, more gradual way....

$$y_t = (1-\beta)z_t + \beta x_t$$

$$z_{t+1} = z_t - \gamma \nabla f(y_t)$$

$$x_{t+1} = \left(1 - \frac{1}{t+1}\right) x_t + \frac{1}{t+1} z_{t+1}$$

$\beta = 0.9$ results in the current gradient evaluation point y containing 0.1 of the most recent gradient g_{t-1}

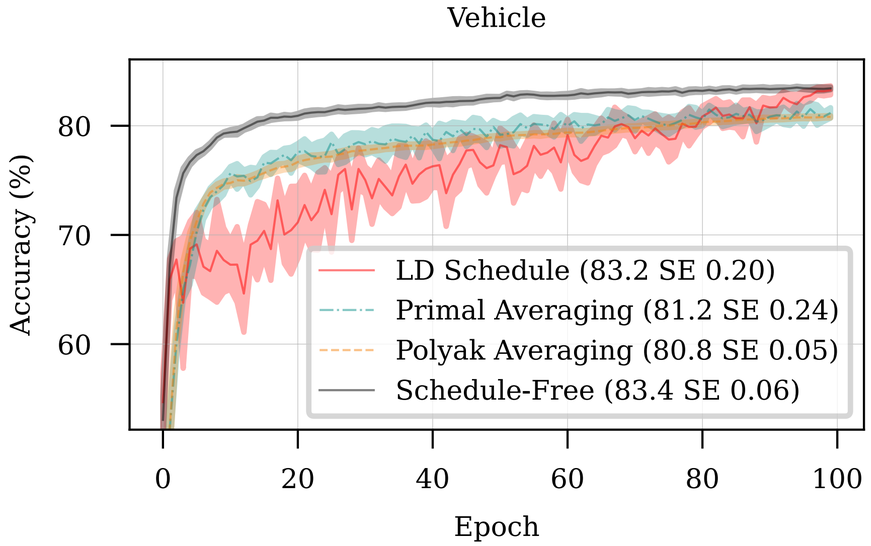
Classical momentum does the same thing! 0.1 of the most recent gradient is included in the step

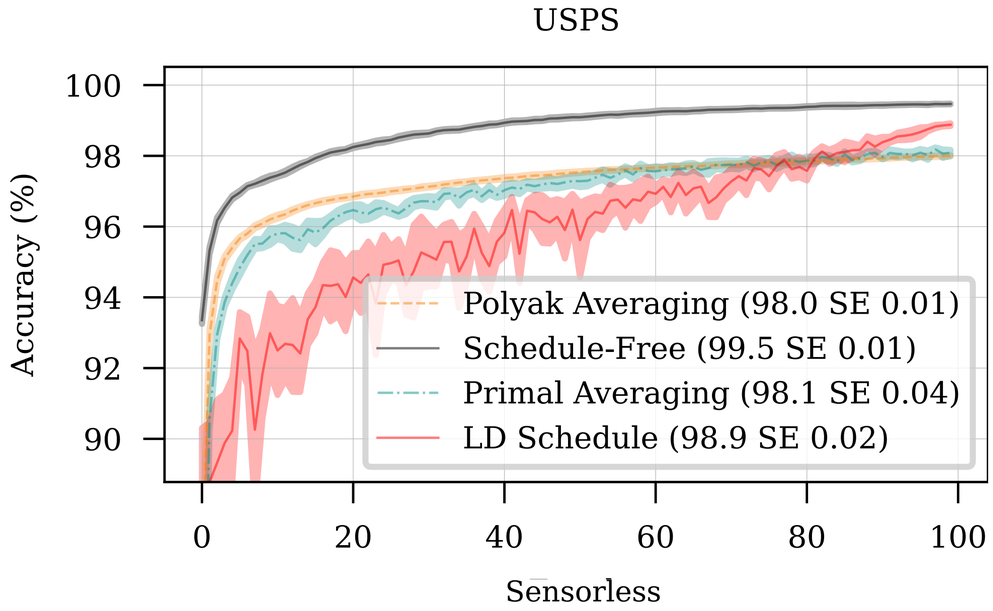
$$m_{t+1} = \beta m_t + \left(1-\beta\right) \nabla f(x_t)$$

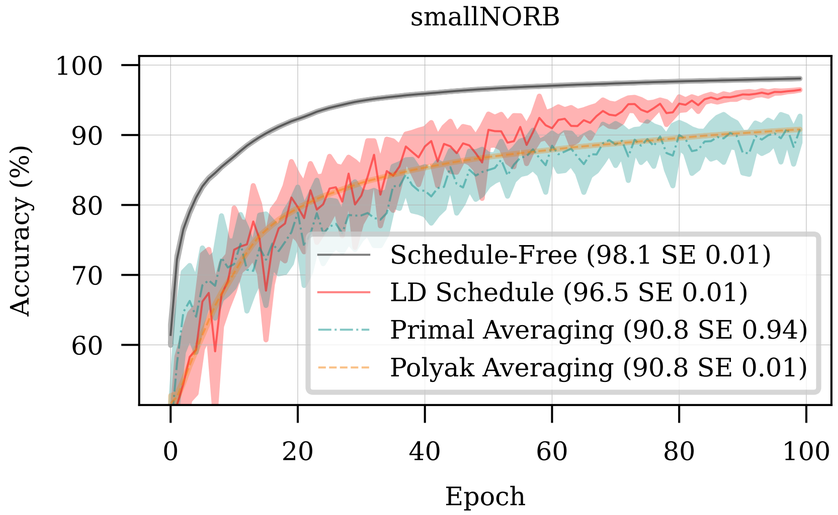
$$x_{t+1} = x_t - \alpha_t m_{t+1}$$

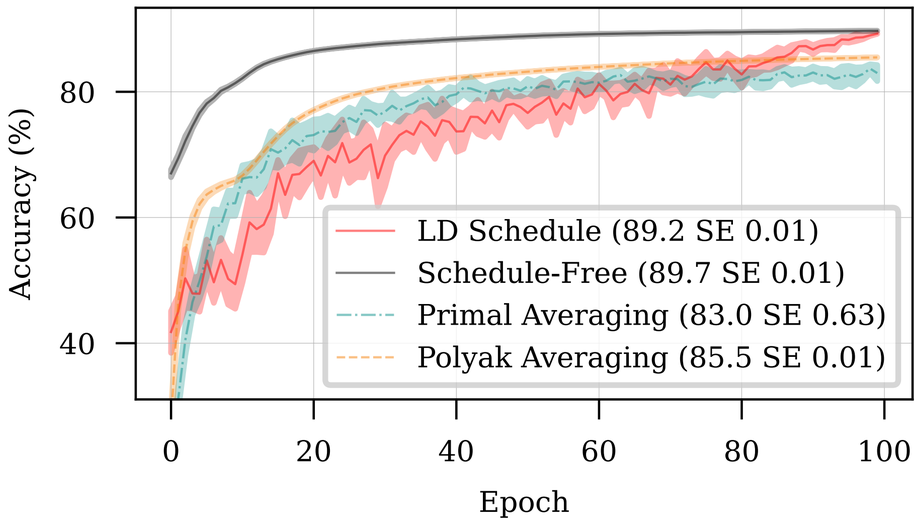
But classical momentum incorporates the rest of the gradient over the next ~10 steps, whereas Schedule-Free incorporates it much **slower**, of the reminder of optimization

Even for convex problems, Schedule-Free outperforms classical averaging and linear decay schedules!

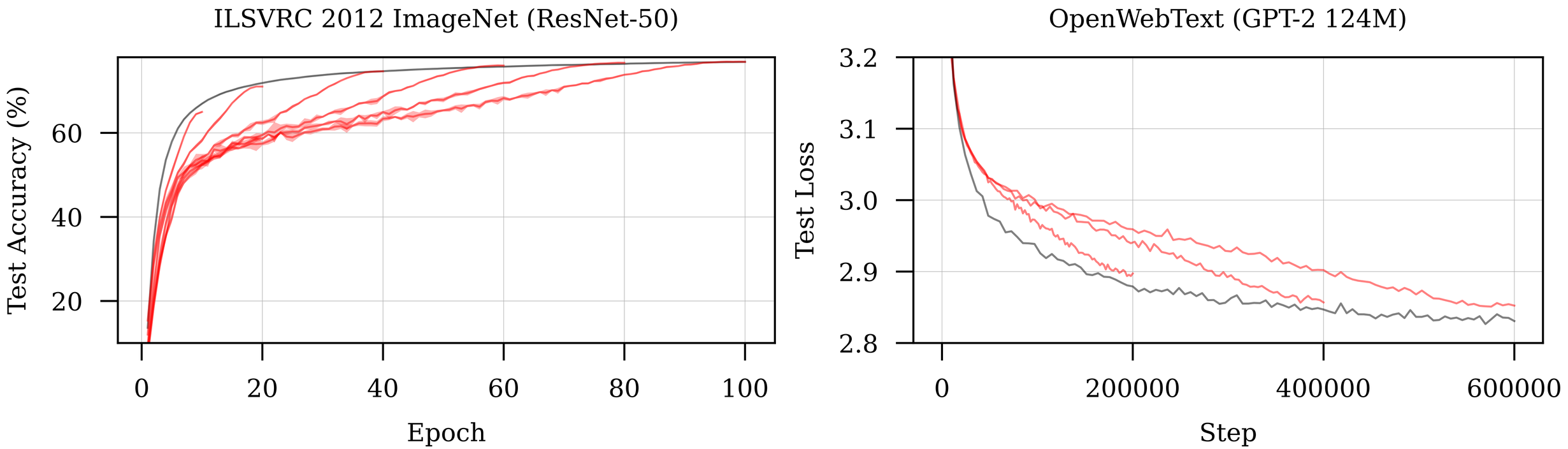








Varying cosine-schedule length shows how Schedule-Free closely tracks the Pareto frontier of Loss v.s. training time.

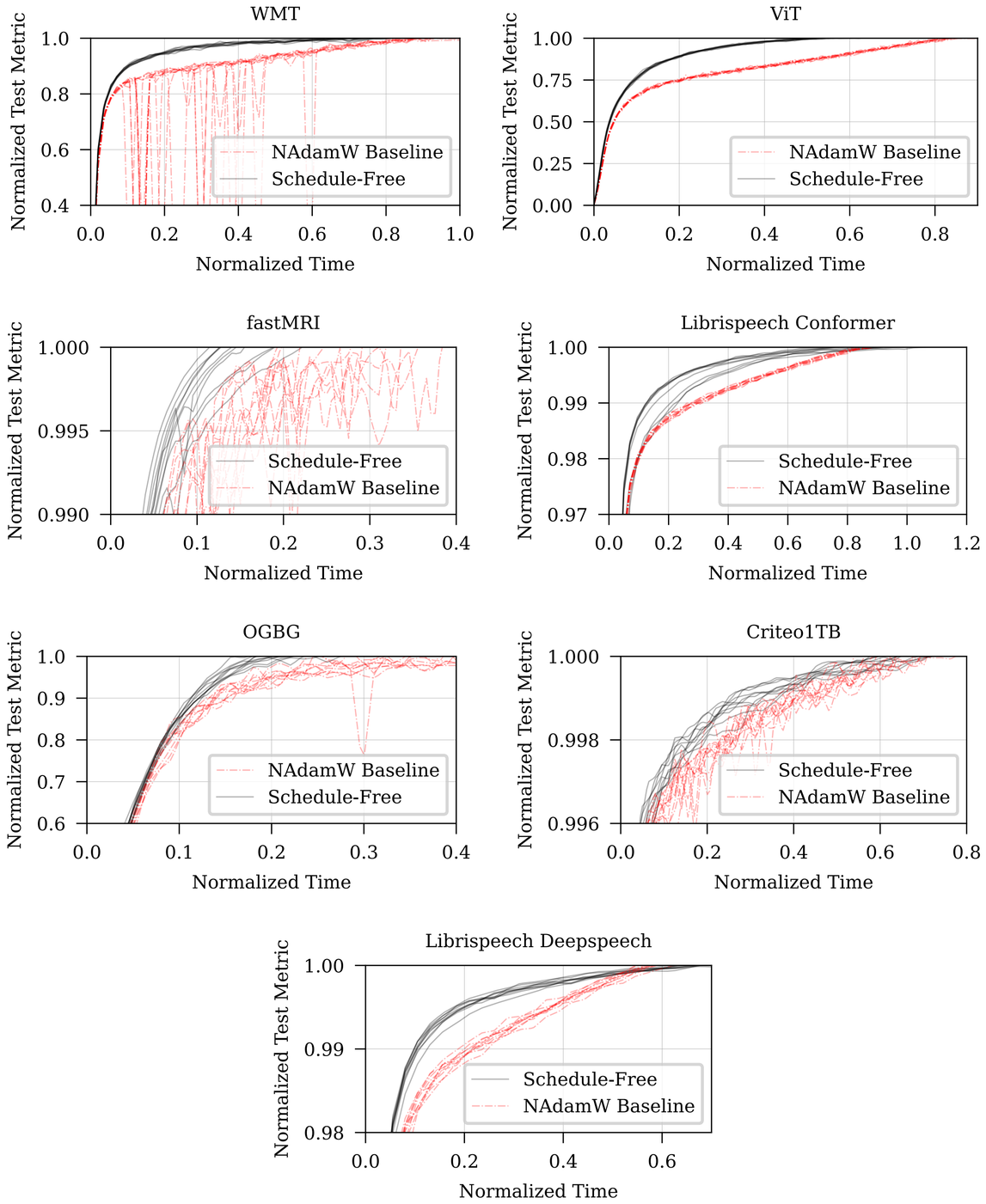


Loss curves look like 1/sqrt(T) curves, just like our theory predicts! Yay!

Schedule-Free AdamW also won the

MLCommons AlgoPerf 2024 Algorithmic Efficiency Challenge Self Tuning Track!

Schedule-Free runs have much smoother loss curves and faster convergence



Online-to-Batch Theory

Theorem 2. *Let F be a convex function. Let $\zeta_1, \dots, \zeta_T$ be an iid sequence such that $F(x) = \mathbb{E}_\zeta[f(x, \zeta)]$. Let $z_1, \dots, z_T$ be arbitrary vectors and let $w_1, \dots, w_T$ and $\beta_1, \dots, \beta_T$ be arbitrary numbers in $[0, 1]$ such that z_t, w_t and β_t are independent of $\zeta_t, \dots, \zeta_T$. Set:*

$$x_t = \frac{\sum_{i=1}^t w_i z_i}{\sum_{i=1}^t w_i} = x_{t-1} \underbrace{\left(1 - \frac{w_t}{\sum_{i=1}^t w_i}\right)}_{\triangleq 1 - c_t} + \underbrace{\frac{w_t}{\sum_{i=1}^t w_i}}_{\triangleq c_t} z_t \tag{9}$$

$$y_t = \beta_t x_t + (1 - \beta_t) z_t \tag{10}$$

$$g_t = \nabla f(y_t, \zeta_t). \tag{11}$$

Then we have for all $x_\star$:

$$\mathbb{E}[F(x_T) - F(x_\star)] \leq \frac{\mathbb{E}[\sum_{t=1}^T w_t \langle g_t, z_t - x_\star \rangle]}{\sum_{i=1}^T w_i}. \tag{12}$$

Beta is a free “momentum-like” parameter that can be tuned in the range [0,1] for best performance.

Unlike standard momentum, it won’t break your convergence theory!

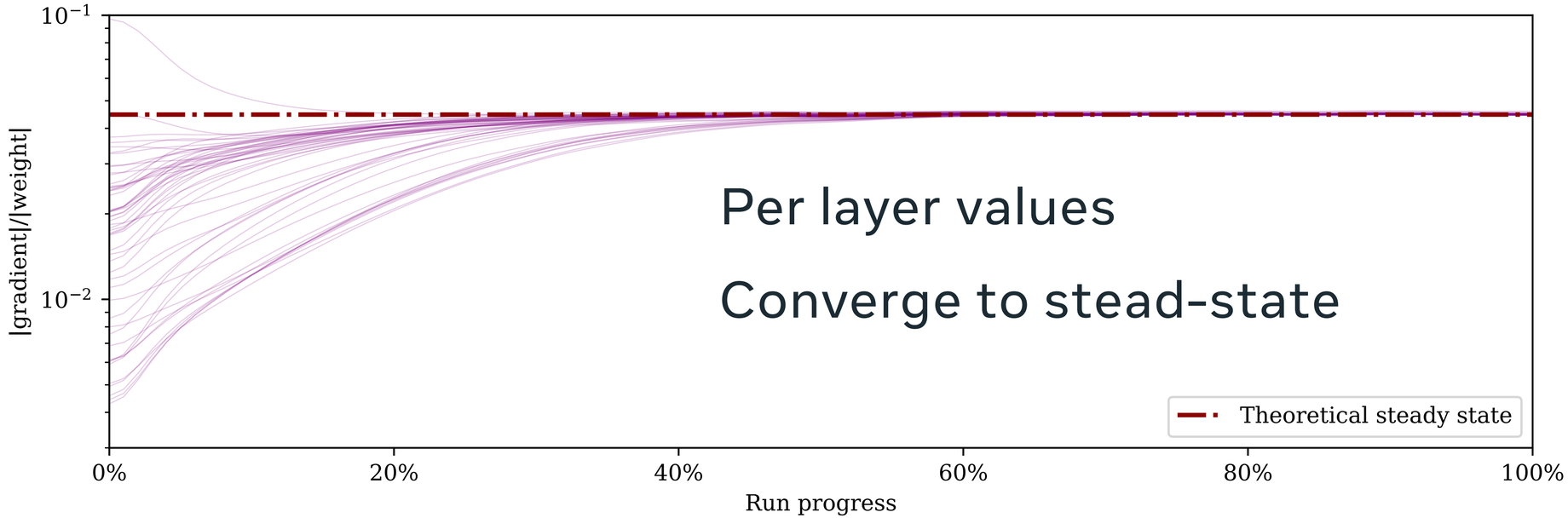
PART 2

Weight-Decay and Normalization is breaking your optimizer!

Weight Decay = Regularization?

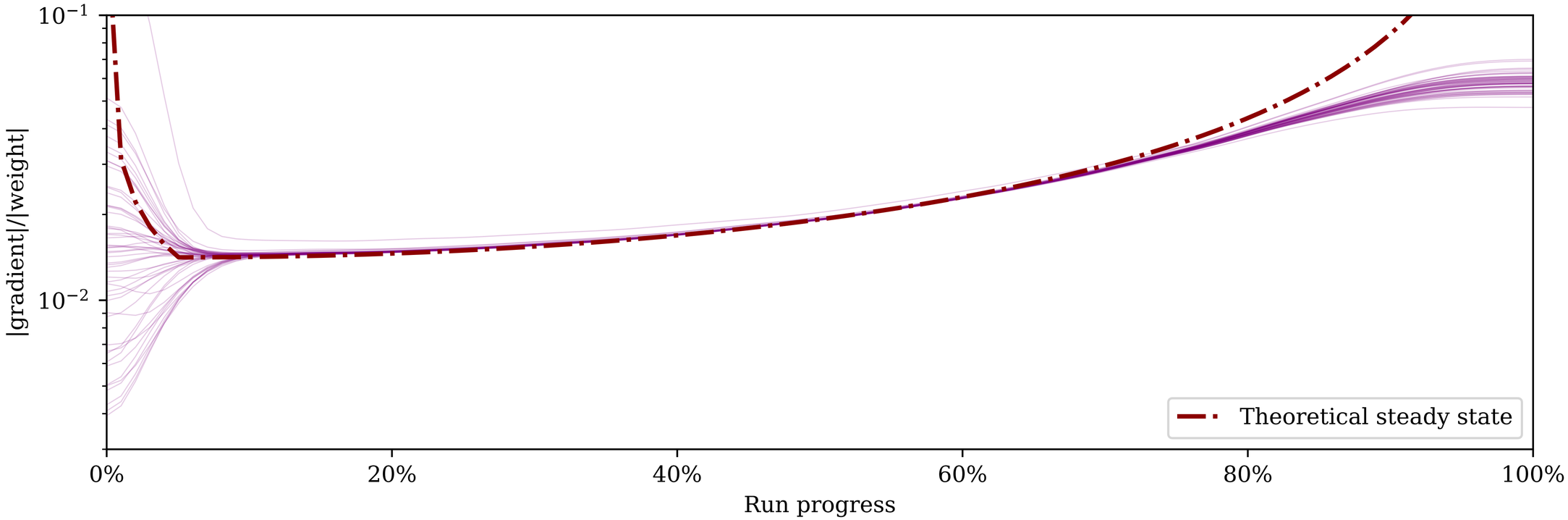
WRONG if your network contains normalization layers!

$$\frac{\|g_t\|}{\|x_t\|} = \sqrt{\frac{2\lambda}{\gamma}}$$



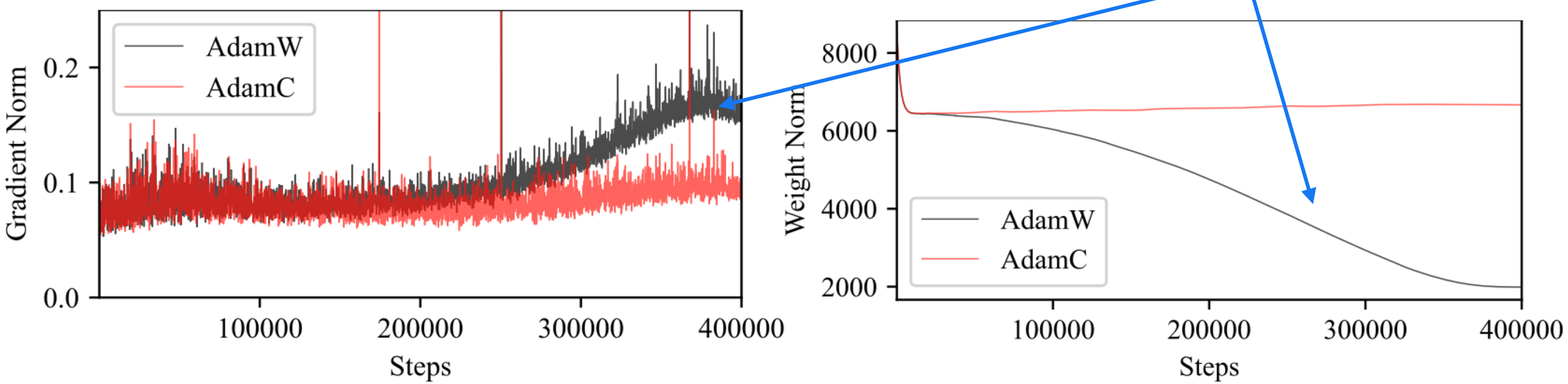
ImageNet training with fixed step size, shows this behavior clearly!

Simple changes like using a learning rate schedule cause non-stationary behavior



Why Does this matter?

Changing learning rate
induces changes in norms
(Yikes!)



If you don't understand and correct for this, algorithms that

rely on gradient and weight magnitudes or past history WILL NOT WORK!

- Polyak Step size
- AdaGrad-Norm
- Coin-betting approaches

The Fix:

Two approaches

- Initialize weights close to steady state and use steady-state preserving techniques like AdamC
- Project the normalized weights (per row) onto a fixed radius hypersphere

Projecting is better but tricky
to get it to work right

Why Gradients Rapidly Increase Near the End of Training

Aaron Defazio [1]

[1]FAIR at Meta

During long-duration Large Language Model (LLM) training runs the gradient norm increases rapidly near the end of training. In this short note, we show that this increase is due to an unintended interaction between weight decay, normalization layers, and the learning rate schedule. We propose a simple correction that fixes this behavior while also resulting in lower loss values throughout training.

Date: June 14, 2025
Correspondence: adefazio@meta.com

Meta

PART 3

Make the right assumptions

This part is more of an opinion…

1. Methods that rely on strong convexity won’t work on deep learning problems (Duh)
2. Methods that rely on Lipschitz smoothness also typically break down!
3. Surprisingly, methods that use convexity and bounded gradients (Online Learning!) often work well
4. Some results from the non-convex case work, but it’s typically too pessimistic

Deep Learning training dynamics appear to be dominated by noise, and methods that adapt to noise rather than smoothness work best.

Thank you!


Meta