Why does Adam work so well for LLMs? And can we find optimal per-variable step sizes?

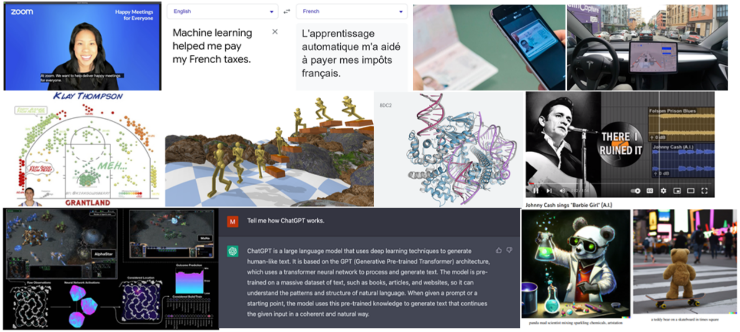
Mark Schmidt
Work led by Frederik Kunstner in collaboration with
Jacques Chen, J. Wilder Lavington (heavy-tailed noise).
Robin Yadav, Alam Milligan, Alberto Bietti (heavy-tailed labels).
Victor S. Portella, and Nick Harvey (multi-dimensional backtracking)

University of British Columbia

February 18, 2025

Machine Learning is Changing the World

- Machine learning (ML) is tool we use to analyze unprecedented amount of data.
    - We use ML every day in a variety of applications.



- Enormous new applications potential (health, engineering, science, and so on).
- Fundamental ML advances can impact many applications (as with ChatGPT).

Machine Learning with SGD and Adam

- "Learning" in most machine learning models is numerical optimization.
  - Trying to find parameters that minimize a cost function.

- Most popular choices of numerical optimizers are (from empirical work):
  - SGD (stochastic gradient descent) with momentum.
  - Adam (adaptive moment estimation) optimizer (and variants).

- I would argue that we do not know have a good understanding of why these work.
  - SGD + momentum has justification in some simplified settings.
  - No theoretical justification for why Adam is sometimes faster than SGD.

- Research topics we explore in this talk:
  - Why is Adam faster than SGD for language models?
  - Could we design a theoretically optimal adaptive method?

Stochastic Gradient Descent (SGD)

- For a minimizing a function f with parameters w, SGD with momentum uses:

$$w_{k+1} = \underbrace{w_k - \alpha_k g(w_k)}_{\text{SGD}} + \underbrace{\beta(w_k - w_{k-1})}_{\text{momentum}},$$

  - The iterate w_k is our guess of parameters on iteration k.
  - The step size α_k affects how far we move on iteration k.
  - The direction $g(w_k)$ is an unbiased estimate of the gradient of the expectaion.
    - Usually, $g(w_k)$ is the gradient of a randomly-chosen example or mini-batch.
  - The momentum rate β is how much we weight previous direction.

The Adam Optimizer

- The Adam optimizer (ignoring "bias correction"):

$$\begin{aligned} \mu_{k+1} &= \beta_1 \mu_k + (1 - \beta_1) g(w_k) && \text{(update momentum)} \\ v_{k+1} &= \beta_2 v_{k-1} + (1 - \beta_2) g(w_k) \circ g(w_k) && \text{(per-variable step size)} \\ w_{k+1} &= w_k - \alpha V_{k+1}^{-1} \mu_{k+1}. && \text{(second-order-ish update)} \end{aligned}$$

- Often interpreted as a Newton-like method (V_k looks likes a preconditioner).
- But if you remove the averages it is sign descent,

$$w_{k+1} = w_k - \alpha \text{sign}(\nabla f(w_k)),$$

which is not consistent with Newton in general.

  - Indeed, we do not havea good understanding of when/why Adam works.

The Perplexity of the Adam Optimizer

- Adam is one of the most-cited works of all time across all fields:

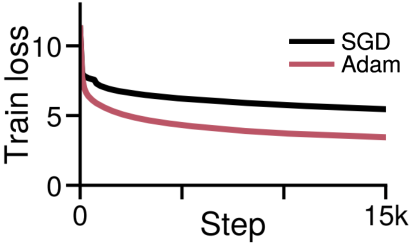
  [CITATION] **Adam**: A method for stochastic optimization
  DP **Kingma** - arXiv preprint arXiv:1412.6980, 2014
  ☆ Save 99 Cite Cited by 202984 Related articles

- Adam does not converge in general and fails on many simple problems.
  - "the most cited paper in all of optimization [...] proves an incorrect theorem with an unparsable convergence bound" [Ben Recht].
  - And many algorithms that "fix" its convergence are slower than original method.

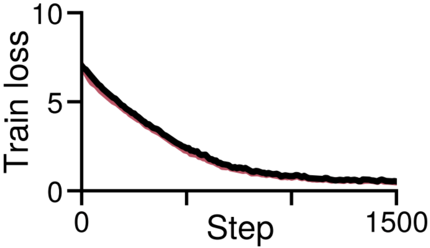
- But for many difficult problems no other algorithm consistently beats it.
  - Only 1 method beat Adam "baseline" variant in AlgoPerf self-tuning competition.
    - But winning method was an Adam variant with more-clever step sizes.

Adam on Natural Language vs. Computer Vision

- Adam versus SGD on language compared to vision:
  - Adam does not tend to outperform SGD on computer vision benchmarks.
  - Adam tends to outperform SGD by a wide margin on language benchmarks.



GPT2-small/WikiText-103



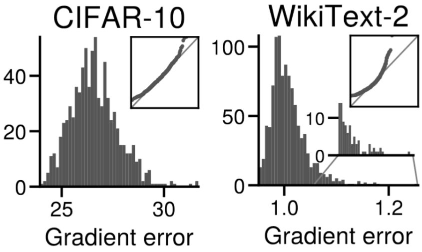
ResNet/ImageNet

Why is Adam faster on Language but not Vision?

- Adam is often applied to over-parameterized models?
  - Explains why Adam is fast, but not why it is faster than SGD.
- Adam has more hyper-parameters to tune than SGD.
  - True, but vision has same hyper-parameters.
- Adam approximates second-order information?
  - Maybe, but why would sign-like update only do this for language models?
- Adam co-evolved with network architectures?
  - True, but vision architectures have been changing too.
- Adam was sent from the future to speed progress in language models?

Heavy-Tailed Noise Hypothesis

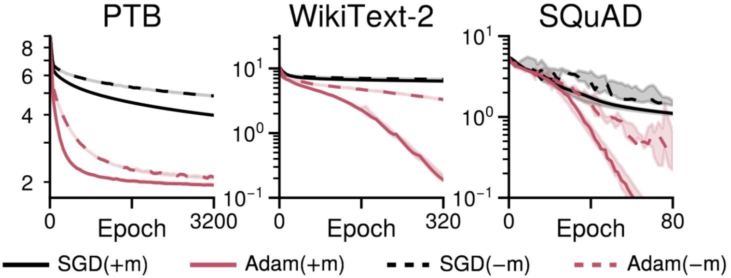
- Language models tend to have heavier-tailed noise than vision models.



- Maybe Adam handles heavy-tailed noise better than SGD?

Heavy-Tailed Noise does NOT Explain the Gap

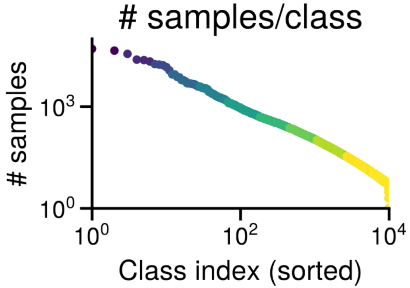
- We tested the heavy-tailed hypothesis by removing the noise.
  - By converging to using the entire dataset to estimate gradients.



- This should reduce gap, but instead gap gets bigger as you remove noise.
  - So gap cannot be due to noise.

New Hypothesis: Heavy-Tailed Labels

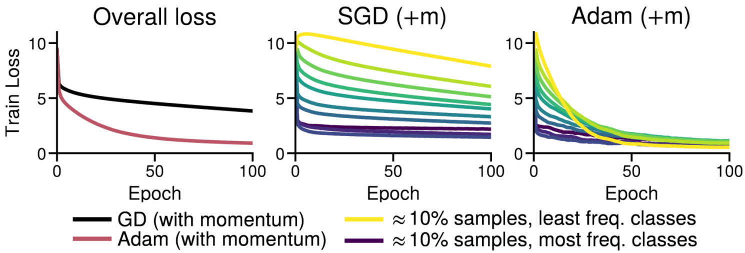
- Vision datasets usually have balanced labels.
  - 1000 cat images, 1000 dog images, 1000 car images, and so on.
- But language datasets have heavy tailed labels.



  - Word “the” appears a ton, but most words are rare.
- Could this help explain the gap?

Heavy-Tailed Labels

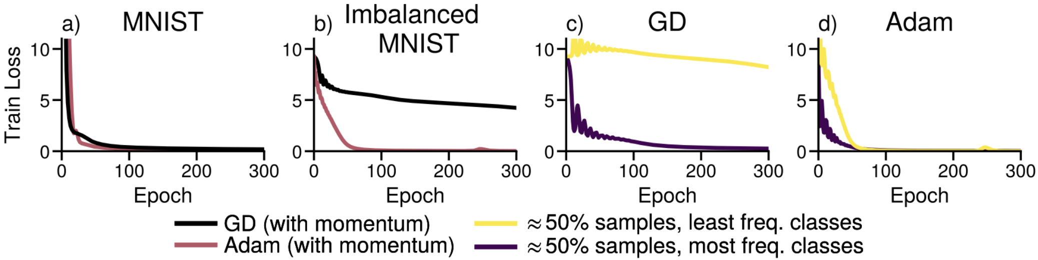
- SGD makes slow progress on rare labels.
  - So if most labels are rare, SGD converges slowly.



- Adam makes similar progress on all labels.
  - So if most labels are rare Adam still makes progress.

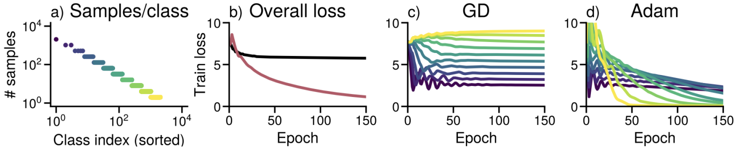
Testing Heavy-Tailed Label Predictions

- What tests could we do to support/refute heavy-tailed label explanation of gap?
- What happens if you make a computer vision dataset with heavy-tailed labels?
  - For example, make a version of MNIST with a large portion of rare labels.
  - Gap appears: Adam converge faster than SGD.



Testing Heavy-Tailed Label Predictions

- What tests could we do to support/refute heavy-tailed label explanation of gap?
- What happens if you make a computer vision dataset with heavy-tailed labels?
  - For example, make a version of MNIST with a large portion of rare labels.
  - Gap appears: Adam converge faster than SGD.
- What about a linear model with heavy-tailed labels?
  - For example, generate synthetic data with label frequency $\pi_k \propto 1/k$.
  - Gap appears: simplified setting that might lead to better theory and/or algorithms.

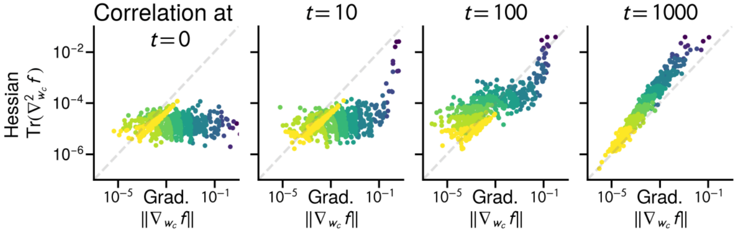


Testing Heavy-Tailed Label Predictions

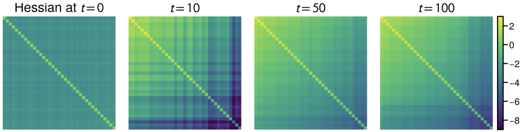
- What tests could we do to support/refute heavy-tailed label explanation of gap?
- What happens if you make a computer vision dataset with heavy-tailed labels?
  - For example, make a version of MNIST with a large portion of rare labels.
  - Gap appears: Adam converge faster than SGD.
- What about a linear model with heavy-tailed labels?
  - For example, generate synthetic data with label frequency $\pi_k \propto 1/k$.
  - Gap appears: simplified setting that might lead to better theory and/or algorithms.
- Could we improve SGD on language models by accounting for label distribution?
  - One strategy is to upweight loss of low-frequency examples.
  - Improves performance of SGD, but changes location of minimum.
    - Unless interpolating data.

Mechanism: Gradient-Hessian and Diagonal Dominance

- Why would sign descent be a good curvature estimate?
- Gradient and Hessian elements become correlated across classes (not universal):



- Hessian becomes dominated by diagonal blocks:

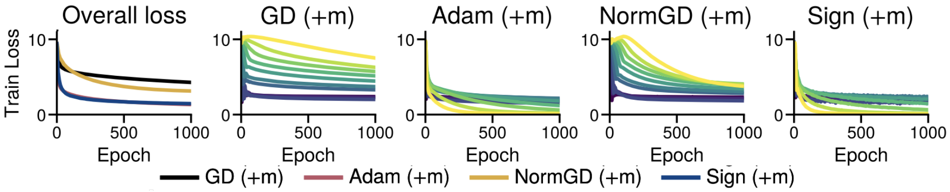


Towards a theory: Sign Descent

- In searching for a theory, we sought a simpler algorithm acting like Adam.
  - For large batches, Adam is similar to sign descent plus momentum.

$$w_{k+1} = w_k - \alpha \text{sign}(\nabla f(w_k)) + \beta(w_k - w_{k-1}),$$

  - Our experiments show this is sufficient to get the improved performance of Adam:
    - But normalized gradient does not perform like Adam.



  - Google Brain later "symbolically discovered" that such methods are effective (Lion).

Towards a theory: Sign Descent

- Can you ever prove that sign descent is faster than gradient descent?
- Yes, on the continuous flow in a very-simplified setting:

Theorem 3. *On the* simple imbalanced setting, *gradient flow and continuous time sign descent initialized at* $\mathbf{W} = 0$ *minimize the loss of class* k, $\ell_k(t) = -\log(\sigma(\mathbf{W}(t)\mathbf{e}_k)_k)$, *at the rate*

$$\textit{Gradient flow:} \quad \ell_k(t) = \Theta(1/\pi_k t), \qquad \textit{Continuous time sign descent:} \quad \ell_k(t) = \Theta\big(e^{-ct}\big).$$

- Gradient descent has a sublinear rate, with worse constants for rare classes.
- Normalized gradient has a linear rate.
- Sign descent has a linear rate, and is invariant to class distributions.

Heavy-Tailed Labels vs. Class Imbalance and Sparse Features

- Is the issue just due to class imbalance or sparse features?
  - No, class imbalance and sparse features can happen even with binary labels.
  - Heavy-tailed labels slowing down SGD is only seen with many possible labels.
- Class imbalance:
  - If 99% of examples are in one class, this class imbalance does not slow down SGD.
    - Will slowly learn minority class, but SGD still makes fast progress on overall loss.
  - Issue with heavy-tailed labels is that most examples are from rare classes.
- Sparse features:
  - Original AdaGrad paper considered sparse-but-informative features.
  - Can also lead to a gap.
    - Graph neural networks have SGD-Adam gap but not heavy-tailed labels.
  - But heavy-tailed labels do not require sparse features (input vs. output layer).
    - It is likely that type of sparsity in intermediate layers can also be important.

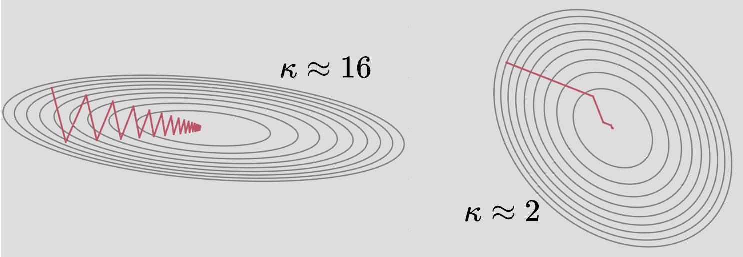
Outline

Diagonal Preconnditioning: Per-Variable Step Sizes

- Adam and many related methods use a step size for each variable

$$w_{k+1} = w_k - a_k \circ \nabla f(w_k).$$

- We can view this as gradient descent with re-scaled variables:



- Preserves $O(d)$ cost and can make the algorithm converge arbitrarily faster.
- This is also known as diagonal preconditioning.
  - Has large literature in numerical linear algebra, dating to Jacobi in 1845.
  - Name "preconditioning" dates to Turing in 1940s.

Hyper-Gradient Descent

- Hyper-gradient descent is an alternative to Adam for setting a_k:
  - Take the gradient with respect to the step sizes, $\nabla_{a_k} f(w_k - a_k \circ \nabla f(w_k))$.
  - Do gradient descent on the step sizes, $a_{k+1} = a_k - \gamma \nabla_{a_k} f(w_k - a_k \circ \nabla f(w_k))$.
- Has been reinvented many times over the last 60 years:

(Maclaurin et al., 2015). Methods have been proposed to tune the step-size (Masse and Ollivier, 2015), a preconditioner (Moskovitz et al., 2019), any hyperparameter (Baydin et al., 2018), or to maintain a model of the objective (Bae et al., 2022). "Stacking" such optimizers recursively has been shown to reduce the dependency on user-specified hyperparameters in practice (Chandra et al., 2022). This idea pre-dates the hypergradient nomenclature; Kesten (1958) presents a method to update the step-size based on the sign of successive gradients, and Saridis (1970) presents a control perspective for per-coordinate step-sizes, which can be cast as a hypergradient update to a diagonal preconditioner.[1] This approach has led to *adaptive gain* methods such as Delta-Delta and variants (Barto and Sutton, 1981; Jacobs, 1988; Silva and Almeida, 1990; Sutton, 1992a,b), and further developed using the sign of the hypergradient (Riedmiller and Braun, 1993), full-matrix updates (Almeida et al., 1999), a larger history (Plagianakos et al., 2001), updates in log-space (Schraudolph, 1999; Schraudolph et al., 2005), heuristics to adjust the outer step-size (Mahmood et al., 2012), or multiplicative weight updates (Amid et al., 2022). While showing promising practical performance in some settings, existing methods are

- Arguably, at the moment it "only works in papers".
  - No version has seen widespread use.
  - Can be sensitve to the step size(s) used on the hyper-gradients.
  - Even if stable, no guarantee it would converge faster*.

Do we have good diagonal preconditioners?

- Consider the following textbook problem:
  - Minimize a smooth and strongly-convex function ($\mu I \preceq \nabla^2 f(w) \preceq LI$).
- Gradient descent with optimal step size achieves an error after k iterations of

$$f(w_k) - f_* \leq \left(1 - \frac{1}{\kappa}\right)^k [f(x_0) - f_*],$$

  and practical algorithms perform within constant factor of this rate (next slide).
- Nesterov acceleration replaces condition number κ with $\sqrt{\kappa}$.
- Optimal diagonal preconditioner replaces κ with $\tilde{\kappa}$.
  - Condition number under best re-scaling of variables.
  - Note that $\tilde{\kappa}$ can be arbitrarily smaller than κ.

- However, no existing method performs within a known factor of $\tilde{\kappa}$ rate.
  - AdaGrad, Adam, hyper-gradient descent, diag(Hessian), quasi-Newton, and so on.
- This work: multidimensional backtracking.
  - First method performing within known factor of $\tilde{\kappa}$ rate.

Classic Backtracking

- Consider the following backtracking procedure to a single step size:
  - Start with a large guess α.
  - Test if w_{k+1} using this α satisfies a sufficient decrease condition

$$f(w_{k+1}) \leq f(w_k) - \frac{\alpha}{2}\|\nabla f(w_k)\|^2.$$

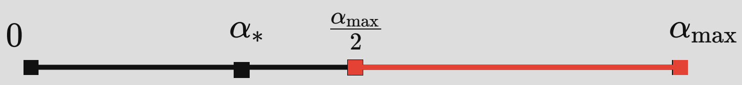
  - If not, divide α in half and try again until it is satisfied.
    - "Valid" step size: α small enough to satisfy sufficient decrease everywhere.
- This procedure gets the rate of the optimal step size up to a factor of 2,

$$f(w_k) - f_* \leq \left(1 - \frac{1}{2\kappa}\right)^k [f(x_0) - f_*].$$

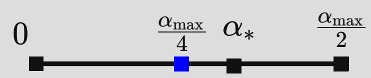
- Many variations exist, and a I do not recommend this exact procedure in practice.
  - For neural nets, above procedure can make sharpness explode.
  - Polyak and Malitsky-Mischenko step sizes: no bactracking and have factor of 4.
  - Non-monotone line-search works better and seems to avoid sharpness explosion.

"Cutting a Set" view of Classic Backtracking

- An alternative way to interpret backtracking based on sets:
  - We maintain an interval $[0, \alpha_{\text{max}}]$ known to contain optimal step size α_*.



  - As our guess for α_*, we try the step size $\alpha = \alpha_{\text{max}}/2$.
  - If $\alpha_{\text{max}}/2$ violates sufficient decrease, we cut the interval by setting $\alpha_{\text{max}} = \alpha$.



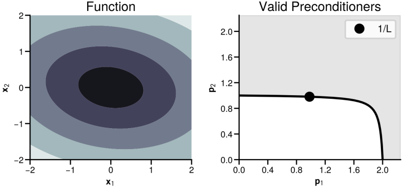
- Maintains that $\alpha_{\text{max}} \geq \alpha_*$ (optimal is in set), and that $\alpha \geq \alpha_*/2$ (not too small).
- But in practice often get lucky and can use a much larger step size than α_*.

Generalizing to more than one step size

- We can define sufficient decrease for valid per-variable step sizes a,

$$f(w_{k+1}) \leq f(w_k) - \frac{1}{2}\langle \nabla f(w_k), a \circ \nabla f(w_k)\rangle.$$

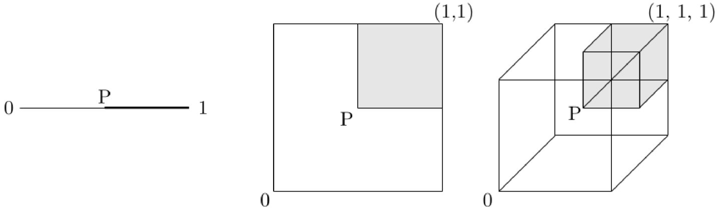
- With d per-variable step sizes the interval is replaced by a d-dimensional space.
  - Each point in space is a set of d per-variable step sizes.
  - Two-dimensional example where first variable can use a larger step size:



  - If set of step sizes a is invalid, what parts of the space can we rule out?
  - If set of step sizes a is invalid, where should we search next?
  - Can we maintain the $O(d)$ cost?

Sufficient Decrease and Curse of Dimensionality

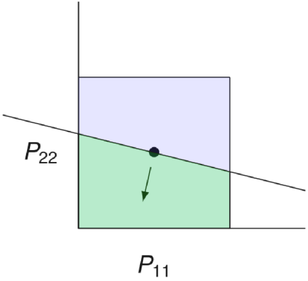
- For invalid step sizes, increasing any step size remains invalid.
- Used by classic backtracking, but not very informative in higher dimensions:



- Sufficient decrease does not rule out much space.
- Do not know whether you can "increase one variable while decreasing another".

Multi-Dimensional Backtracking with Hyper-Gradients

- Key insight behind multi-dimensional backtracking for convex problems:
  - **Hyper-gradient gives a separating hyper-plane on valid step sizes**.
    - Speficially, gradient of sufficient decrease condition with respect to step sizes.

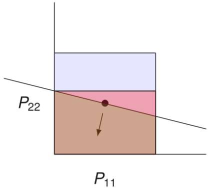


  - If we have a set containing optimal preconditioners:
    - If a preconditioner is invalid, hyper-gradient tells us direction of valid preconditioners.

  - Allows you to cut off large parts of the space even in high dimensions.

Efficient Multi-Dimensional Backtracking

- We do not want to store/manipulate a set of cutting planes.
- Consider the box method of just storing largest box containing current plane.



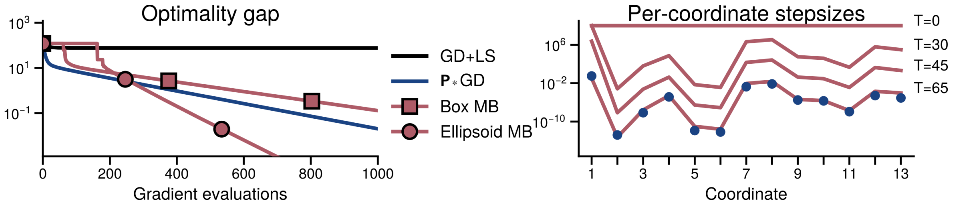
- Backtracking along all coordinates from largest point in box by $2d$ gives a rate

$$f(w_k) - f_* \leq \left(1 - \frac{1}{2d\tilde{\kappa}}\right)^k [f(x_0) - f_*].$$

- Paper also gives improved method using axis-aligned ellipsoids:
  - Improves factor of d to $\sqrt{d}$ while maintaining $O(d)$ iteration cost.

Multi-Dimensional Backtracking for Linear Regression

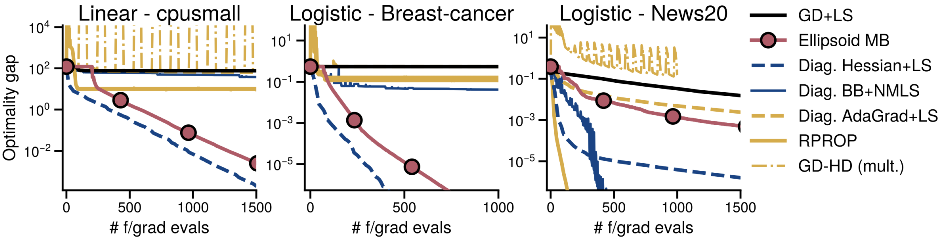
- Comparison of methods for linear regression.
  - We can expensively compute optimal preconditioner in this setting.



- Box method converges to be close to optimal preconditioner.
- For this problem we have $\kappa = 10^{14}, \sqrt{\kappa} = 10^7, \tilde{\kappa} = 10^2$.
  - So diagonal preconditioning helps much more than acceleration.
- Ellipsoid method outperforms optimal preconditioner.
  - "Got lucky" and made extra progress with invalid preconditioners (which is typical).

Multi-Dimensional Backtracking for Logistic Regression

- Comparison to adaptive preconditioning methods for logistic regression:



- MDB tends to find a good preconditioner when there is one.
  - But is sometimes outperformed by current practical algorithms.
- Like quasi-Newton, but tries to maximize progress and not approximate Hessian.
  - Approximating Hessian is only the right thing to do asymptotically.

Open Problems

- Relaxing strong convexity.
  - Paper considers PL functions, unclear how to handle general non-convex.
- Practical implementation details.
  - Only trick in experiments was multiplying all step sizes by 1.1 after accepted steps.
  - More-clever forward tracking and/or combining with existing tricks.
  - Generalizing to consider momentum and stochastic gradients.
- Reducing $\sqrt{d}$ factor.
  - Easy to get best of classic and multi-dimensional backtracking.
  - Could interpolate between classic and multi-dimensional with groups of variables.
  - Recent online scaling work of Gao et al. [2024].
    - Use online learning for hyper-gradient descent to remove $\sqrt{d}$ asymptotically.
- Connecting to non-asymptotic dense quasi-Newton results [Jin et al., 2024].
  - Initially slower than MDB but eventually faster, though with expensive iterations.

Overall Talk Summary

- Diagonal preconditioning methods like Adam are wildly popular.
- Effectiveness does not seem to be due to how they handle noise.
- Effectiveness of Adam seems to be largely due to heavy-tailed labels.
  - Large fraction of examples have rare labels.
- We gave the first diagonal preconditioning method with optimality guarantees.
  - Multi-dimensional backtracking uses cutting planes to rule out sets of step sizes.
- Thank you for the invite and coming to listen.