Understanding and improving the numerical optimization underlying modern machine learning

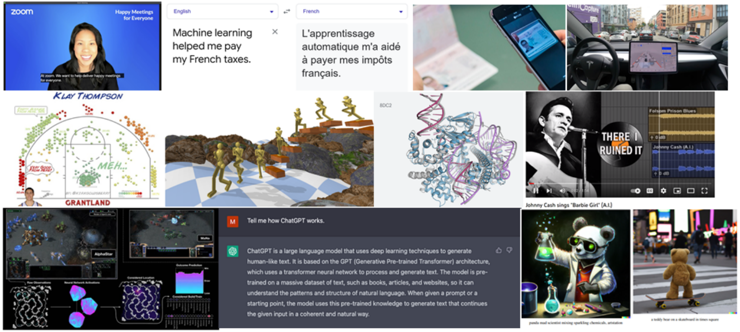
Mark Schmidt

University of British Columbia

February 20, 2024

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.



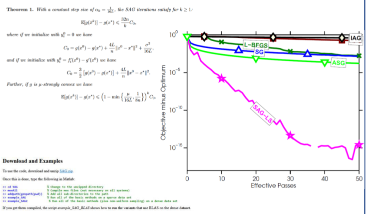
- Enormous new applications potential (health, engineering, science, and so on).
- Fundamental ML advances can impact many applications (as with ChatGPT).

Research Overview: ML Algorithms and Applications

- My research can be divided into two themes:

Improving Algorithms underlying ML Models

- My "core" research area.
- Prove theorems.
- Develop practical algorithms.
- Release code.



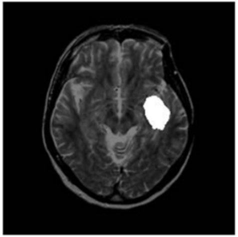
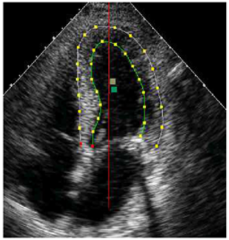
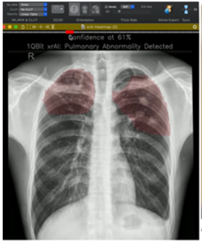
Applying ML Models to Real-World Problems

- Use ML as tool in applications.
- Contributions to many domains.
- Often develop new models.
- Try to take applications to deployment.

- Talk focus: understanding and improving SGD and Adam for modern ML.
- But first, I will overview 3 lines of research from each of the above two areas.

Applications Example 1: Medical Imaging

- Identifying abnormalities or assessing malignance in medical images.
  - Brain tumours, heart motion, breast cancer, x-ray abnormalities.

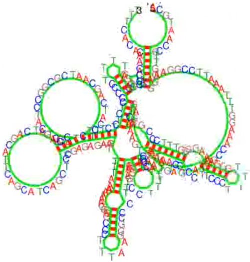





AI and X-ray for remote communities in Canada powered by Synthesis and Google Cloud

- XrAI system developed with 1QBit and Synthesis Health:
  - Instead of academic environment, tested "in the wild" in a clinical trial.
  - Can dramatically reduce some types of medical errors.
  - Approved by Health Canada as a medical device, and now deployed.

Applications Example 2: DNA Kinematics

- Estimating time required for DNA molecules to fold into different configurations.
  - Long-term goal of developing DNA computers.



-30.3 kcal/mol reached after 470.8 ms and 13308 transitions

- Introduced flexible biomechanical/kinetic model, parameters learned from papers.
  - Explains behaviours not captured by previous models.
  - Can predict results of published papers based on experimental conditions.
  - Awarded best student paper prize at conference on DNA-based computers.

Applications Example 3: Photo Retouching

- AutoPortrait: high-resolution automatic face retouching in photos.
  - Removes pimples/blemishes, stray hairs, small wrinkles, and so on.
  - Does not distort image/skin/features nor does it remove moles/freckles/tattoos.

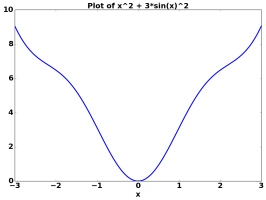
- Required developing a high-resolution convolutional neural network.
  - Outperformed professional artists in user study.
  - Led to creation of a startup (Skylab Technologies).
  - Used by studios around the world, has processed over 100 million images.

Other Applications Projects

- Some other applications projects I have been involved with:
  - Seismic imaging (used by WesternGeco and Exxon Mobil).
  - Detecting over-fishing in lakes (with Freshwater Fisheries Society of BC).
  - Ecological monitoring (being added to TimeLapse software)
  - Product recommendation (provides recommendations for > 4000 retailers).
  - Photobook cropping/color-correcting (used by Artona since 2019).
  - Object distances based on GTA5 (MIT Technology Review, Vice Motherboard).
  - Learning in self-driving cars with safety constraints (with Inverted AI).
- The rest of the talk will focus on algorithms research.
  - But feel free to ask me about these applications!

Algorithms Example 1: Relaxing "Strong Convexity"

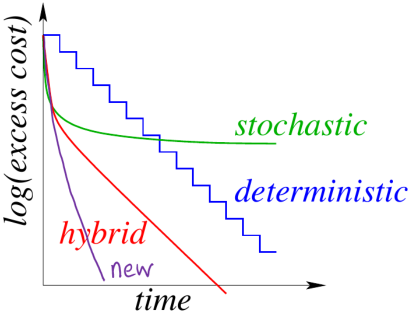
- Machine learning models are usually fit with variants of gradient descent.
  - Fast for strongly-convex functions, but most models are not strongly-convex.
  - People spent 25 years proposing conditions to fix this:
    - Essential strong convexity, weak strong convexity, restricted strong convexity, semi-strong convexity, optimal strong convexity, quadratic growth condition, error bounds, restricted secant inequality.
- We re-discovered Polyak-Łojasiewicz inequality (from 1963 in Russian/French):



- Simpler proofs and weaker than all above conditions.
- Simplified/generalized analyses of well-known problems.
  - Least squares, logistic regression, boosting, backprop, L1-regularization, SVMs, SDCA, SVRG.
- Does not require convexity or that solution is unique.
  - Holds for PCA and some neural networks.
- 1000 citations.
  - E-mail from Polyak thanking us for highlighting his old work.

Algorithms Example 2: Faster Stochastic Gradient Methods

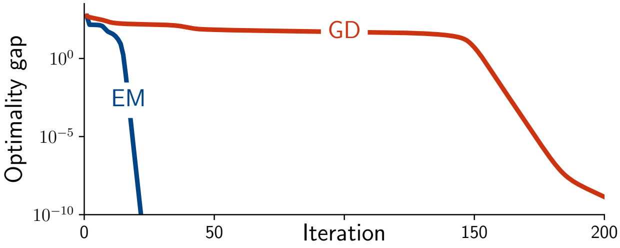
- Stochastic gradient descent (SGD) methods are widely-used in ML.
  - $O(1)$ iteration cost in terms of number of training examples.
  - But they have sublinear convergence rates.



- Our SAG algorithm was first method with $O(1)$ cost and linear convergence rate.
  - Led to subfield of variance-reduced SGD and faster algorithms for many problems.
    - Least squares, logistic regression, SVMs, PCA, CRFs, and so on.
  - Awarded 2018 Lagrange Prize in Continuous Optimization (1 award every 3 years).

Algorithms Example 3: Speed of EM for Missing Data

- Expectation maximization (EM):
  - Most common algorithm for handling missing data (60k citations).
  - Previous non-asymptotic analysis: "at least as fast as gradient descent".



- Homeomorphic-invariant analysis by writing it as a form mirror descent.
  - Under standard assumptions including "fitting a mixture of Gaussians"
  - Arbitrarily faster than previous rates, depending on "amount of missing information".
  - 10/10 from two reviewers and 2021 AI/Stats best paper (1 out of 1527).

Other Algorithms Projects

- Some other algorithms projects I have been involved with:
  - minFunc optimization code (100k downloads).
    - PyTorch L-BFGS implementation "heavily inspired" by this code.
  - Optimizing with simple constraints or simple non-smooth regualrizares.
    - 5 papers w/ 100 citations, AI/Stats best paper, NeurIPS oral, 10k downloads.
  - Tighter analysis of greedy coordinate optimization.
    - Dimension-independent rate, led to faster method for PageRank.
    - Justification for LIBSVM and improving ideal rule from $O(n^2)$ to $O(n \log n)$.
  - Practical natural gradient methods.
    - Non-conjugate, mixtures of conjugate, second-order versions.
    - Won 2021 Approximate Inference in Bayesian Deep Learning competition.
  - Target-based surrogate optimization (ICML 2023).
    - Reduces amount of environment interaction needed in imitation learning.

- This talk will not cover any of these projects.
  - But feel free to ask me about these or other numerical optimization problems!

Outline

Motivation: Over-Parameterized Models in Machine Learning

- Modern machine learning practitioners often do a weird thing:
  - Train (and get excellent performance) with models that are over-parameterized.
    - "The model is so complicated that you can fit the data perfectly".
    - The exact setting where we normally teach students that bad overfitting happens.
- Many state-of-the-art deep computer vision models are over-parameterized.
  - Models powerful enough to fit training set with random labels.
- Many recent papers study benefits of over-parameterization in various settings:
  - Optimizers may find global optima in problems we normally view as hard.
  - Algorithms may have implicit regularization that reduces overfitting.

Single-Slide Summary of these Sections

- For over-parameterized models, you need to re-think how optimization works!

1. SGD converges faster for over-parameterized models.
    - May help explain why it has been so difficult to develop faster algorithms.
2. We can design faster stochastic algorithms for over-parameterized models.
    - Over-parameterization allows Nesterov acceleration and second-order methods.
3. We can design robust stochastic algorithms that are easier to use.
    - Algorithms that adapt to the difficulty of the problem.

Stochastic Gradient Descent (SGD)

- For most ML models, we fit parameters w by minimizing an expectation,

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(w).$$

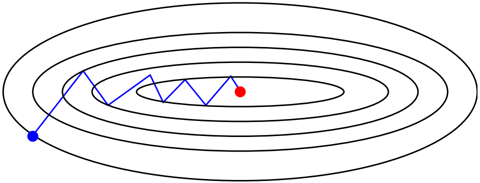
  - Function f measures prediction error on training example i.
  - We have n training examples and d parameters.
  - Includes linear least squares, logistic regression, neural networks, and so on.
- Among most common algorithms is stochastic gradient descent (SGD),

$$w_{k+1} = w_k - \alpha_k g(w_k).$$

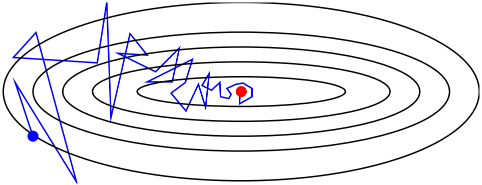
  - The iterate w_k is our guess of parameters on iteration k.
  - The step size α_k affects how far we move on iteration k.
  - The direction $g(w_k)$ is an unbiased estimate of the gradient of the expectaion.
    - Usually, $g(w_k)$ is the gradient of a randomly-chosen example or mini-batch.

Determnistic vs. Stochastic Gradient Descent

- Deterministic gradient descent converges with a small-enough constant step size.



  - But each iteration has a cost of $O(n)$ in terms of n.
- SGD needs a decreasing sequence of step sizes α_k to converge (slow).
  - To asymptotically reduce effect of variance in the gradient approximation.



  - But each iteration has a cost of $O(1)$ in terms of n.

How Fast do they Converge?

- Consider the classic assumption that eigenvalues of Hessian are bounded,

$$\mu I \preceq \nabla^2 f(x) \preceq LI,$$

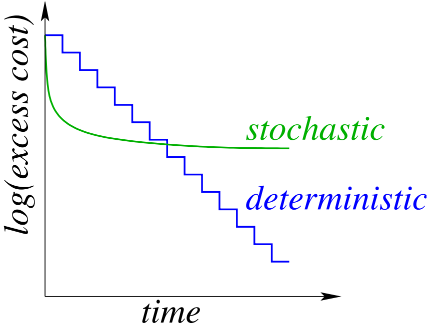
between positive constants μ and L for all x (holds for many problems).

  - And for SGD assume also that the variation in the gradients is bounded,

$$\mathbb{E}_i[\|\nabla f_i(w) - \nabla f(w)\|^2] \leq \sigma^2.$$

- After k iterations, methods find a w satisfying the following convergence rates:
  - $f(w) - f^* = O(\gamma^k)$ for gradient descent (linear rate).
  - $\mathbb{E}[f(w)] - f^* = O(1/k)$ for SGD (sublinear rate).

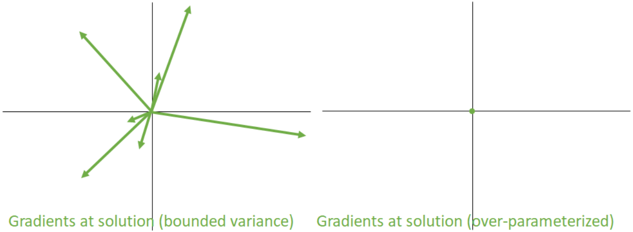
How Fast do they Converge?



- Deterministic has linear convergence $O(\gamma^k)$ but $O(n)$ iteration cost.
- Stochastic has sublinear convergence $O(1/k)$ but $O(1)$ iteration cost.
  - Many hybrid methods exist, and variance-reduced SGD can be even faster.
  - But for over-parameterized models, do not need these tricks since SGD is faster.

Effect of Over-Parameterization on SGD

- We say a model is over-parameterized if it can exactly fit all training examples.
  - Implies the interpolation assumption that $\nabla f_i(w_*) = 0$ for all i:



- Under over-parameterized models, the variance is 0 at minimizers.
  - And SGD converges with a sufficiently small constant step size.

Stochastic Convergence Rate under Over-Parameterization

- One way to measure over-parameterization is the strong growth condition (SGC),

$$\mathbb{E}[\|\nabla f_i(w)\|^2] \leq \rho \|\nabla f(w)\|^2,$$

which is implied by the Hessian assumptions and interpolation.

- We showed that SGD with constant step size and SGC achieves linear rate,

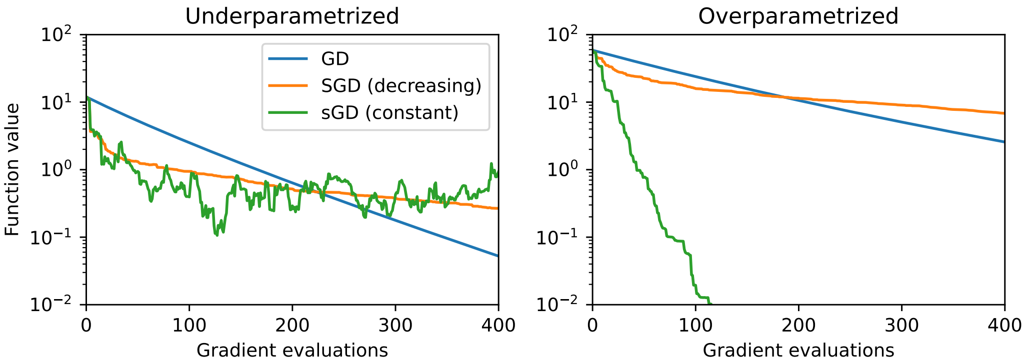
$$\mathbb{E}[f(w_k)] - f^* \leq \left(1 - \frac{\mu}{\rho L}\right)^k [f(w_0) - f^*],$$

which is the same rate as gradient descent up to the factor of ρ.

  - If $\rho = 1$ get the rate of deterministic gradient descent.

Stochastic Convergence Rates under Over-Parameterization

- Comparison of least squares performance in under-/over-parameterized models:



- SGD with constant step size works better for more complicated models!

Beyond Strong-Convexity and Over-Parameterization

- We have considered various generalizations:
  - Introduced a weak growth condition with better worst-case rate.
  - Considered functions that are convex but not strongly convex.
  - Considered functions satisfying PL inequality (can be non-convex).
  - Considered general non-convex functions that are bounded below.
    - Same step size achieves fast rate for all settings, so SGD adapts to the problem.

- Conditions can be generalized to consider "close to over-parameterized",

$$\mathbb{E}[\|\nabla f_i(w)\|^2] \leq \rho\|\nabla f(w)\|^2 + \sigma^2,$$

where constant α gives quick convergence to region region of size $O(\alpha\sigma^2/|B|)$.

  - If not close to over-paramerized, decrease α or increase the batch size.
    - GPT-3 uses a batch size of 3 million.

How long does my algorithm take?

- In CS, ask "how long does this take" instead of "error on iteration k"?
  - Want a big-O bound on runtime, and to know whether this is optimal.

- Converting from error on iteration k to a runtime bound:
  - Pick a desired accuracy ϵ.
  - Find smallest k such that error is below ϵ.

- Example of gradient descent:
  - Error on iteration k is $(1 - 1/\kappa)^k[f(x_0) - f^*]$, for $\kappa = L/\mu$ (condition number)
  - Smallest k such that error is below ϵ is $O(\kappa \log(1/\epsilon))$.
    - Where we treat initial sub-optimality as constant.

Accelerated Gradient Descent

- There exist faster algorithms only requiring $O(\sqrt{\kappa}\log(1/\epsilon))$ iterations.
  - These are called accelerated methods.

- First accelerated method is the heavy-ball method for quadratic functions,

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) + \underbrace{\beta_k(w_k - w_{k-1})}_{\text{momentum}}.$$

- First accelerated gradient method for convex functions was Nesterov's method,

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) + \beta_k(w_k - w_{k-1}) - \underbrace{\alpha_k\beta_k(\nabla f(w_k) - \nabla f(w_{k-1}))}_{\text{second-order-ish correction}},$$

and achieves acceleration in different settings with appropriate α_k and β_k.
  - Stochastic variants have worked well empirically for training neural networks.

Provably-Accelerated SGD?

Assumption	Regular	Accelerated	Accelerated?
Deterministic	$\tilde{O}(n\kappa)$	$\tilde{O}(n\sqrt{\kappa})$	yes.
SGD (variance bounded)	$\tilde{O}(\kappa + \sigma^2/\mu\epsilon)$	$\tilde{O}(\sqrt{\kappa} + \sigma^2/\mu\epsilon)$	if $\kappa > \sigma^2/\mu\epsilon$.
SGD (variance reduced)	$\tilde{O}(n + \kappa)$	$\tilde{O}(n + \sqrt{n\kappa})$	if $\kappa > n$.
SGD (over-param)	$\tilde{O}(\rho\kappa)$	$\tilde{O}(\sqrt{\rho\kappa})$	yes.

- Unlike previous assumptions, over-parameterization allows fully-accelerated SGD.
  - Our first result: ρ outside square root. 2024 result: acceleration helps with noise.

- We have shown that over-parameterization allows second-order SGD methods:
  - Show superlinear rate with slower-growing batch size than previous work.
  - Includes self-concordant analysis, L-BFGS analysis, and Hessian-free implementation.

- We have shown FTL has constant regret for online imitation learning.
  - Without over-parameterization regret is sublinear but unbounded.

Outline

Setting the Step Size

- Unfortunately, these faster rates have a serious practical issue.
  - They are sensitive to the choice of step size (which depend on L and/or μ).
  - Performance significantly degrades under a poor choice of step size.
- You could search over several plausible guesses for the step size.
  - But searching is slow and a fixed step size may be sub-optimal anyways.
  - It would be better to adapt the step size as you go.
- Prior methods do not guarantee fast convergence possible for over-parameterized.
  - Meta-learning, heuristics, adaptive, online learning, probabilistic line-search.
  - Or they require increasing batch sizes (stochastic line-search and trust-region).
- We proposed a simple stochastic line search.
  - Achieves fast convergence rates in a variety of over-parameterized settings.
  - Outperforms a variety of methods in practice on many standard benchmarks.

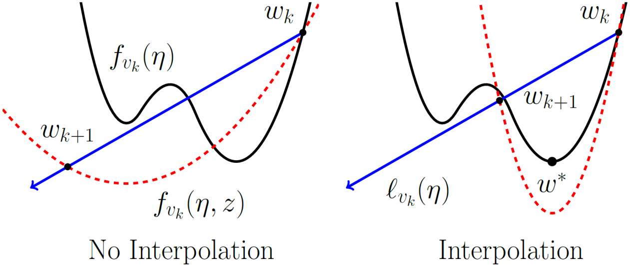
Stochastic Line Search

- An Armijo line-search on the mini-batch selects a step size satisfying

$$f_{i_k}(w_k - \alpha_k \nabla f_{i_k}) \leq f_{i_k}(w_k) - c\alpha_k \|\nabla f_{i_k}(w_k)\|^2,$$

for some constant $c > 0$.

- Without interpolation this does not work (satisfied by steps that are too large).



- With over-parameterization, can guarantee sufficient progress towards solution.

Stochastic Line Search

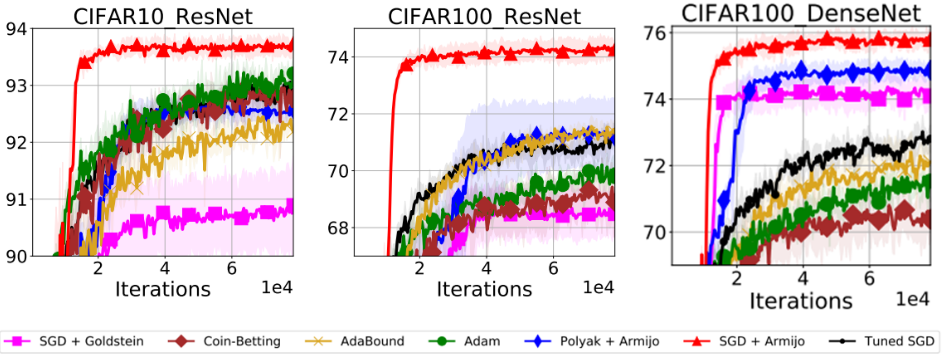
- To find a large step satisfying Armijo, can use a backtracking line search.
  1. Start with some initial step size.
  2. Test the Armijo condition (requires an extra forward pass for neural networks).
  3. If condition is not satisfied, decrease step size and go to 2.

- The line search guarantees same rates as when we know smoothness constant.
  - For strongly-convex, convex, and PL objectives (adapts to problem difficulty).
  - Works for non-convex if initial step sizes are not too large (work needed).

- We expect the line-search to converge faster in practice.
  - Step sizes set by theory only guarantee worst-case improvement.
  - Line searches can get lucky and decrease the function by larger amounts.

Experimental Results with Stochastic Line Search

- We did a variety of experiments, including training CNNs on standard problems.
  - Better than fixed step sizes, adaptive methods, alternate adaptive step sizes.



Stochastic Line Search - Discussion

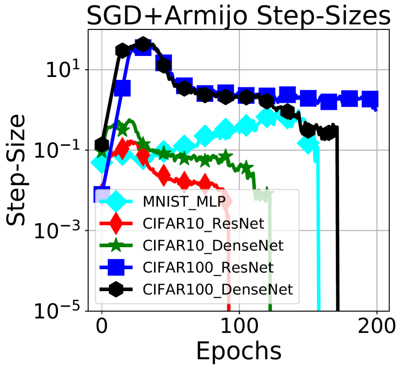
- Our implementation initializes by slightly increasing previous step size.
  - Median number of times we test Armijo condition was 1.
  - So overall cost is less than cost of trying 2 fixed step sizes.

- We ran synthetic experiments controlling degree of over-parameterization.
  - With over-parameterization, the stochastic line search works great.
  - If close to over-parameterized, line search still works really well.
    - Theory can be modified to handle case of being close to over-parameterized.
  - If far from over-parameterized, line search catastrophically fails.

- We have used the stochastic line-search in other algorithms.
  - Meng et al. [2020] use it to set the step size in a second-order method.
  - Vaswani et al. [2020] show that it speeds up Adam empirically.

Armijo Step Size and Step Size Collapse

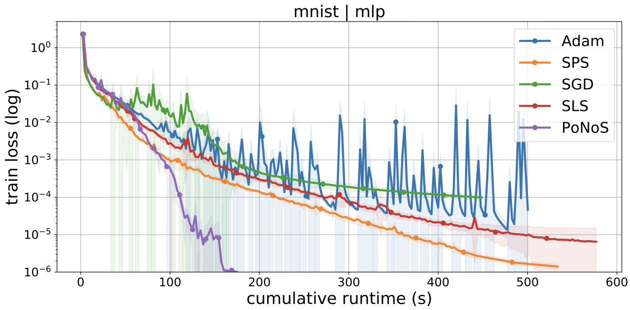
- Different datasets need different step sizes at different times.



- When training neural networks, sometimes step size collapsed.

Non-Monotonic Stochastic Line Search and Polyak Resetting

- Several authors considered stochastic Polyak step size for over-parameterized.
  - Requires bounding f^* but has closed form and avoids backtracking/collapse.

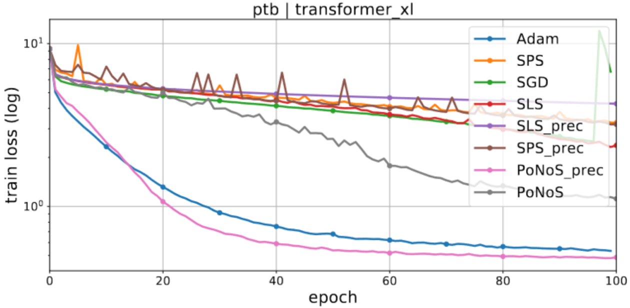


- NeurIPS 2023: Polyak non-monotone stochastic (PoNoS).
  - Much faster than previous methods in practice and does not seem to collapse.
  - Initializes line search using an adaptive Polyak step size.
  - Line search that does not require function to decrease at every step.

Outline

SGD vs. Adam on Vision and Language Data

- SGD with line search seems to work great for computer vision models.
  - Even if we allow competing methods to tune their step size for free.
- But for language models, Adam tends to work better (with tuned step size).



- You can add a line search to Adam (PoNoS_prec above).
  - But why is there a gap between Adam and SGD?

The Perplexity of the Adam Optimizer

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

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

- One of the most-cited works of all time across all fields:

Adam: A method for stochastic **optimization**
DP Kingma, J Ba - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org
... stochastic **optimization** methods. Finally, we discuss AdaMax, a variant of **Adam** based on ... We propose **Adam**, a method for efficient stochastic **optimization** that only requires first-order ...
Save Cite Cited by 168383 Related articles All 27 versions

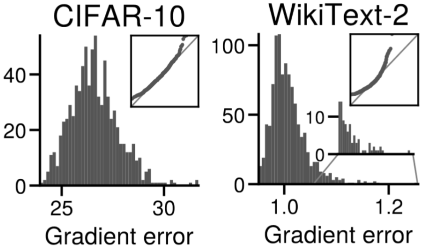
- Does not converge in general and fails on many simple problems.
- But for many difficult problems no other algorithm consistently beat it.
  - And many algorithms that "fix" its convergence are slower than original method.

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 empirical Fisher work better 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?
  - My favourite explanation.

Heavy-Tailed Noise Hypothesis

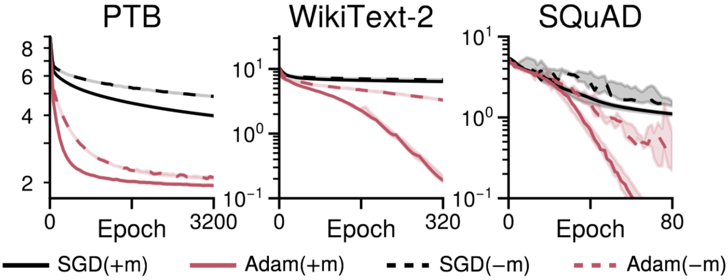
- Language models tend to have heavier-tailed noise than vision models.



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

Heavy-Tailed Noise does NOT Explain the Gap

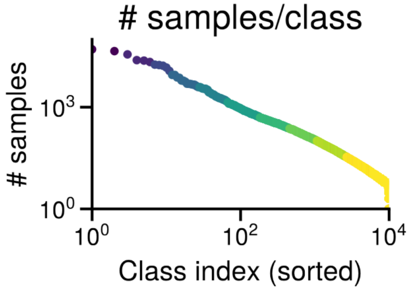
- We tested the heavy-tailed hypothesis by removing the noise.
  - By using the entire dataset to estimate gradients.



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

New Hypothesis: Heavy-Tailed Labels

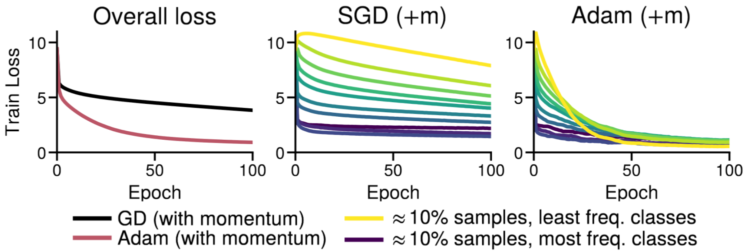
- Vision datasets usually have balanced labels.
    - 1000 cat images, 1000 dog images, 1000 car images, and so on.
- But language datasets have heavy tailed labels.



    - 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.
- What happens if you make a vision dataset with heavy-tailed labels?
  - Gap appears: Adam converge faster than SGD.

Simplifying Adam: Sign Descent plus Momentum

- Heavy-tailed labels can even make Adam outperform SGD for linear models.

- In searching for a theory, we sought a simpler algorithm acting like Adam.
  - Found Adam behaviour can be replicated with sign descent plus momentum,

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

  - Afterwards Google Brain "symbolically discovered" a similar method (Lion),

Symbolic Discovery of Optimization Algorithms

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, Quoc V. Le

We present a method to formulate algorithm discovery as program search, and apply it to discover optimization algorithms for deep neural network training. We leverage efficient search techniques to explore an infinite and sparse program space. To bridge the large generalization gap between proxy and target tasks, we also introduce program selection and simplification strategies. Our method discovers a simple and effective optimization algorithm, **Lion** (***EvoLved Sign Momentum***). It is more memory-efficient than Adam as it only keeps track of the momentum. Different from adaptive optimizers, its update has the same magnitude for each parameter calculated through the sign operation. We compare Lion with widely used optimizers, such as Adam and Adafactor, for training a variety of models on different tasks. On image classification, Lion boosts the accuracy of ViT by up to 2% on ImageNet and saves up to 5x the pre-training compute on JFT. On vision-language contrastive learning, we achieve 88.3% ***zero-shot*** and 91.1% ***fine-tuning*** accuracy on ImageNet, surpassing the previous best results by 2% and 0.1%, respectively. On diffusion models, Lion outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x. For autoregressive, masked language modeling, and fine-tuning, Lion exhibits a similar or better performance compared to Adam. Our analysis of Lion reveals that its performance gain grows with the training batch size. It also requires a smaller learning rate than Adam due to the larger norm of the update produced by the sign function. Additionally, we examine the limitations of Lion and identify scenarios where its improvements are small or not statistically significant. Lion is also successfully deployed in production systems such as Google search ads CTR model.

- With simpler algorithm and models, we are getting close to a theory!
  - Could to lead to faster algorithms with nice theoretical properties.
  - These might be useful not only be sparse labels but for sparse rewards.

Summary

- Research focuses on improving ML algorithms and using ML in applications.
- For over-parameterized models, plain SGD is fast and faster versions are possible.
- For over-parmaeterized models, better performance using adaptive step sizes.
- Adam works well for heavy-tailed labels, due to approximating sign of gradient.
- Thank you for the invite and coming to listen.