Incorporating second order ideas into first class machine learning methods

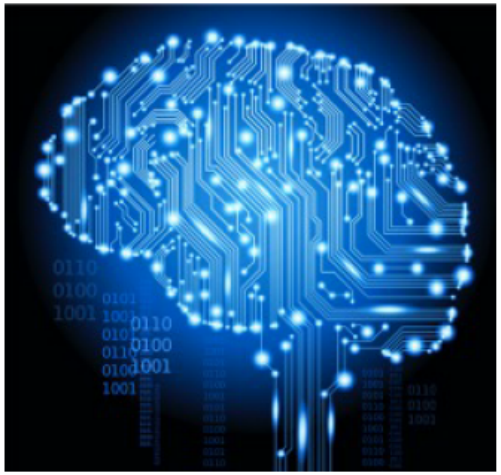
Michael W. Mahoney
ICSI and Department of Statistics, UC Berkeley

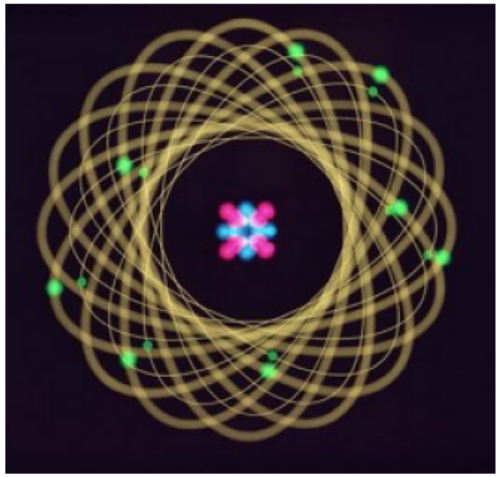
August 2021

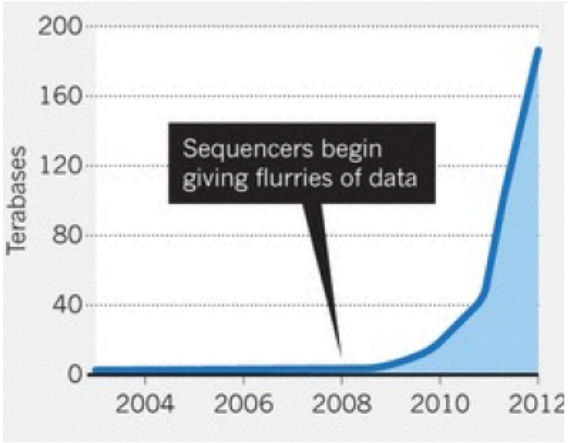
(Joint work with Fred Roosta, Amir Gholami, Zhewei Yao, Liam Hodgkinson, and others.)

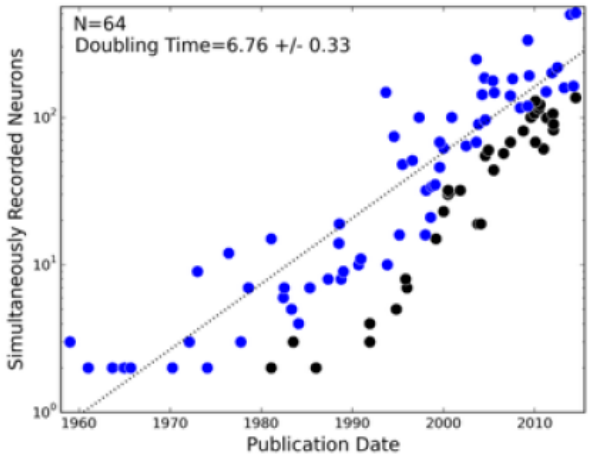
Outline

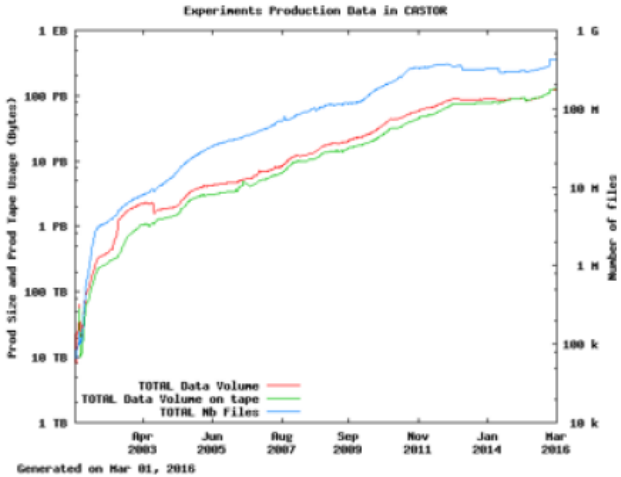
Big Data ... Massive Data ...











Scientific Computing and Machine Learning share the same challenges,
and use the same means,
but to get to different ends!

Machine Learning has been, and continues to be, very busy designing
efficient and effective optimization methods

Outline

First Order Methods

- Variants of Gradient Descent (GD):
  - Reduce the per-iteration cost of GD $\Rightarrow$ Efficiency
  - Achieve the convergence rate of the GD $\Rightarrow$ Effectiveness

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \nabla F(\mathbf{x}^{(k)})$$

First Order Methods

- E.g.: SAG, SDCA, SVRG, Prox-SVRG, Acc-Prox-SVRG, Acc-Prox-SDCA, S2GD, mS2GD, MISO, SAGA, AMSVRG, ...

1st order method and "over-fitting"

Challenges with "simple" 1st order method for "over-fitting":

- Highly sensitive to ill-conditioning
- Very difficult to tune (many) hyper-parameters

"Over-fitting" is difficult with "simple" 1st order method!

Remedy?

1. "Not-So-Simple" 1st order method, e.g., accelerated *and* adaptive

2. 2nd order methods, e.g.,  methods

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - [\nabla^2 F(\mathbf{x}^{(k)})]^{-1} \nabla F(\mathbf{x}^{(k)})$$

1. 2nd order methods: Stochastic Newton-Type Methods
   - Stochastic Newton (think: convex)
   - Stochastic Trust Region (think: non-convex)
   - Stochastic Cubic Regularization (think: non-convex)

Problem 2: Minimizing Finite Sum Problem

$$\min_{\mathbf{x}\in\mathcal{X}\subseteq\mathbb{R}^d} F(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$$

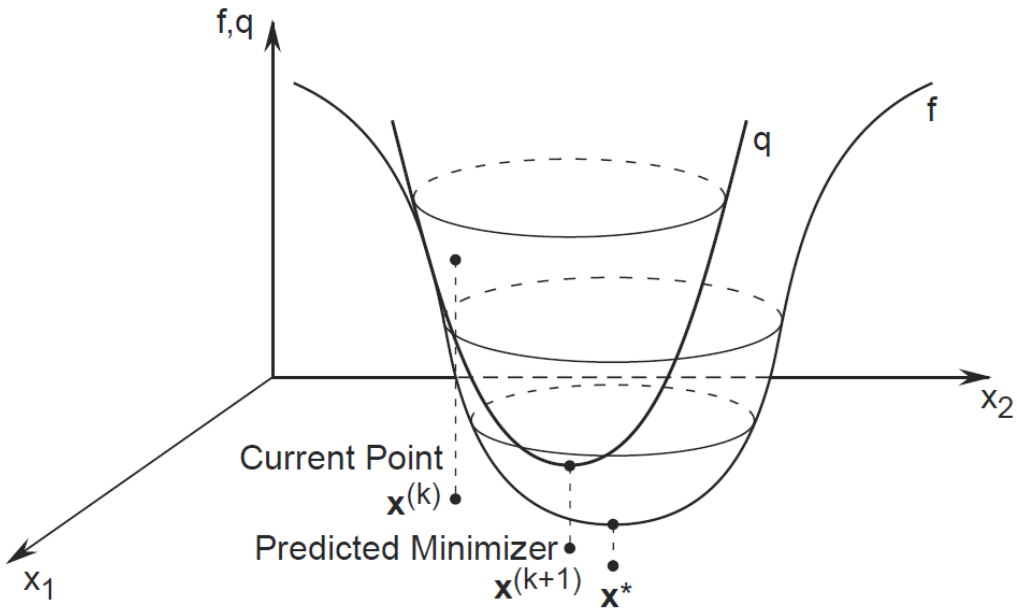
- f_i: (Non-)Convex and Smooth
- $n \gg 1$

Second Order Methods

- Deterministically approximating second order information cheaply
  - Quasi-Newton, e.g., BFGS and L-BFGS [Nocedal, 1980]
- Randomly approximating second order information cheaply
  - Sub-Sampling the Hessian [Byrd et al., 2011, Erdogdu et al., 2015, Martens, 2010, RM-I, RM-II, XYRRM, 2016, Bollapragada et al., 2016, ...]
  - Sketching the Hessian [Pilanci et al., 2015]
  - Sub-Sampling the Hessian and the gradient [RM-I & RM-II, 2016, Bollapragada et al., 2016, ...]

Iterative Scheme

$$x^{(k+1)} = \arg \min_{\mathbf{x} \in \mathcal{D} \cap \mathcal{X}} \left\{ F(\mathbf{x}^{(k)}) + (\mathbf{x} - \mathbf{x}^{(k)})^T \mathbf{g}(\mathbf{x}^{(k)}) + \frac{1}{2\alpha_k} (\mathbf{x} - \mathbf{x}^{(k)})^T H(\mathbf{x}^{(k)}) (\mathbf{x} - \mathbf{x}^{(k)}) \right\}$$



Convex Problems

- Each f_i is smooth and weakly convex
- F is γ-strongly convex

"We want to design methods for machine learning that are not as ideal as Newton's method but have [these] properties: first of all, they tend to turn towards the right directions and they have the right length, [i.e.,] the step size of one is going to be working most of the time...and we have to have an algorithm that scales up for machine leaning."

Prof. Jorge Nocedal
IPAM Summer School, 2012
Tutorial on Optimization Methods for ML
(Video - Part I: 50' 03")

What do we need?

- Requirements:

(R.1) **Scale up:** $|\mathcal{S}|$ must be independent of n, or at least smaller than n and for $p \gg 1$, allow for inexactness

(R.2) **Turn to right directions:** $H(\mathbf{x})$ must preserve the spectrum of $\nabla^2 F(\mathbf{x})$ as much as possible

(R.3) **Not ideal but close:** Fast local convergence rate, close to that of Newton

(R.4) **Right step length:** Unit step length eventually works

Sub-sampling Hessian

Lemma (Uniform Hessian Sub-Sampling)

Given any $0 < \epsilon < 1$, $0 < \delta < 1$ *and* $\mathbf{x} \in \mathbb{R}^p$, *if*

$$|\mathcal{S}| \geq \frac{2\kappa^2 \ln(2p/\delta)}{\epsilon^2},$$

then

$$\Pr\left((1-\epsilon)\nabla^2 F(\mathbf{x}) \preceq H(\mathbf{x}) \preceq (1+\epsilon)\nabla^2 F(\mathbf{x})\right) \geq 1-\delta.$$

SSN-H Algorithm: Inexact Update

Algorithm 5 Globally Convergent SSN-H with inexact solve

1: **Input:** $\mathbf{x}^{(0)}$, $0 < \delta < 1$, $0 < \epsilon < 1$, $0 < \beta, \theta_1, \theta_2 < 1$
2: - Set the sample size, $|\mathcal{S}|$, with ϵ and δ
3: **for** $k = 0, 1, 2, \cdots$ until termination **do**
4: - Select a sample set, $\mathcal{S}$, of size $|\mathcal{S}|$ and form $H(\mathbf{x}^{(k)})$
5: - Update $\mathbf{x}^{(k+1)}$ with $H(\mathbf{x}^{(k)})$ and inexact solve
6: **end for**

Gloabl Convergence SSN-H: Inexact Update

Theorem (Global Convergence of Algorithm 5)

Using Algorithm 5 with $\theta_1 \approx 1/\sqrt{\kappa}$, *with high-probability, we have*

$$F(\mathbf{x}^{(k+1)}) - F(\mathbf{x}^*) \leq (1-\rho)\big(F(\mathbf{x}^{(k)}) - F(\mathbf{x}^*)\big),$$

where $\rho = \alpha_k \beta/\kappa$ *and* $\alpha_k \geq \frac{2(1-\theta_2)(1-\beta)(1-\epsilon)}{\kappa}$.

Local + Global

Theorem

For any $\rho < 1$ and $\epsilon \approx \rho/\sqrt{\kappa}$, Algorithm 5 is globally convergent and after $\mathcal{O}(\kappa^2)$ iterations, with high-probability achieves "problem-independent" Q-linear convergence, i.e.,

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq \rho \|\mathbf{x}^{(k)} - \mathbf{x}^*\|.$$

Moreover, the step size of $\alpha_k = 1$ passes Armijo rule for all subsequent iterations.

- Trust Region: Classical Method for Non-Convex Problem [Sorensen, 1982, Conn et al., 2000]

$$\mathbf{s}^{(k)} = \arg \min_{\|\mathbf{s}\| \leq \Delta_k} \langle \mathbf{s}, \nabla F(\mathbf{x}^{(k)}) \rangle + \frac{1}{2} \langle \mathbf{s}, \nabla^2 F(\mathbf{x}^{(k)}) \mathbf{s} \rangle$$

- Cubic Regularization: More Recent Method for Non-Convex Problem [Griewank, 1981, Nesterov et al., 2006, Cartis et al., 2011a, Cartis et al., 2011b]

$$\mathbf{s}^{(k)} = \arg \min_{\mathbf{s} \in \mathbb{R}^d} \langle \mathbf{s}, \nabla F(\mathbf{x}^{(k)}) \rangle + \frac{1}{2} \langle \mathbf{s}, \nabla^2 F(\mathbf{x}^{(k)}) \mathbf{s} \rangle + \frac{\sigma_k}{3} \|\mathbf{s}\|^3$$

- To get iteration complexity, all previous work required:

$$\left\| \left(H(\mathbf{x}^{(k)}) - \nabla^2 F(\mathbf{x}^{(k)}) \right) \mathbf{s}^{(k)} \right\| \leq C \|\mathbf{s}^{(k)}\|^2 \tag{1}$$

- Stronger than "Dennis-Moré"

$$\lim_{k \to \infty} \frac{\| \left(H(\mathbf{x}(k)) - \nabla^2 F(\mathbf{x}(k)) \right) \mathbf{s}(k) \|}{\|\mathbf{s}(k)\|} = 0$$

- We relaxed (1) to

$$\left\| \left(H(\mathbf{x}^{(k)}) - \nabla^2 F(\mathbf{x}^{(k)}) \right) \mathbf{s}^{(k)} \right\| \leq \epsilon \|\mathbf{s}^{(k)}\| \tag{2}$$

- Quasi-Newton, Sketching, Sub-Sampling satisfy Dennis-Moré and (2) but not necessarily (1)

Outline

Executive Summary

- We propose ADAHESSIAN, a novel second order optimizer that achieves new SOTA on various tasks:
  - CV: Up to 5.55% better accuracy than Adam on ImageNet
  - NLP: Up to 1.8 PPL better result than AdamW on PTB
  - Recommendation System: Up to 0.032% better accuracy than Adagrad on Criteo
- ADAHESSIAN achieves these by:
  - Low cost Hessian approximation, applicable to a wide range of NNs
  - A novel temporal and spatial smoothing scheme to reduce Hessian noise across iterations

Z Yao, A Gholami, S Shen, M. Mustafa, K Keutzer, M. W. Mahoney, ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning, arXiv: 2006.00719

AdaHessian Motivation

- Choosing the right hyper-parameter for optimizing a NN training has become a (very expensive) **dark-art!**

Problems with existing first-order solutions:

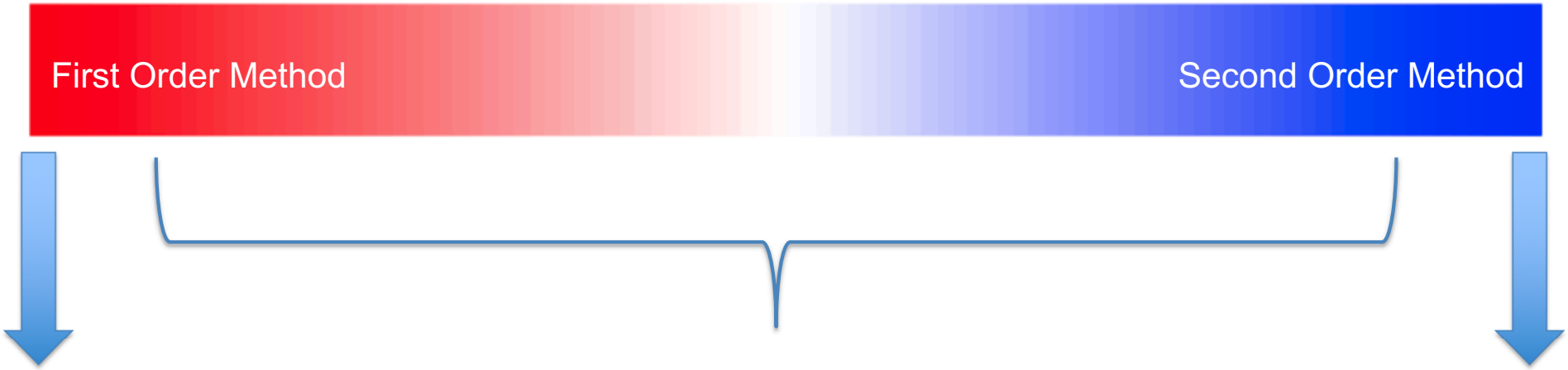
- Brute force hyper-parameter tuning
- No convergence guarantee unless taking *many* iterations
- *Even the choice of the optimizer is a hyper-parameter!**

Task	CV	NLP	Recommendation System
Optimizer Choice	SGD	AdamW	Adagrad

*BTW, not obvious if you just do popular things, e.g., ResNet50 training on ImageNet, since years of industrial scale (i.e., brute force) hyperparameter tuning and building systems for SGD-based methods mean those methods do well ...

First and Second Order Methods

General parameter update formula: $\theta_{t+1} = \theta_t - \eta_t \Delta\theta_t$



$\Delta\theta_t = H_t^0 g_t = g_t$

How about the middle part?

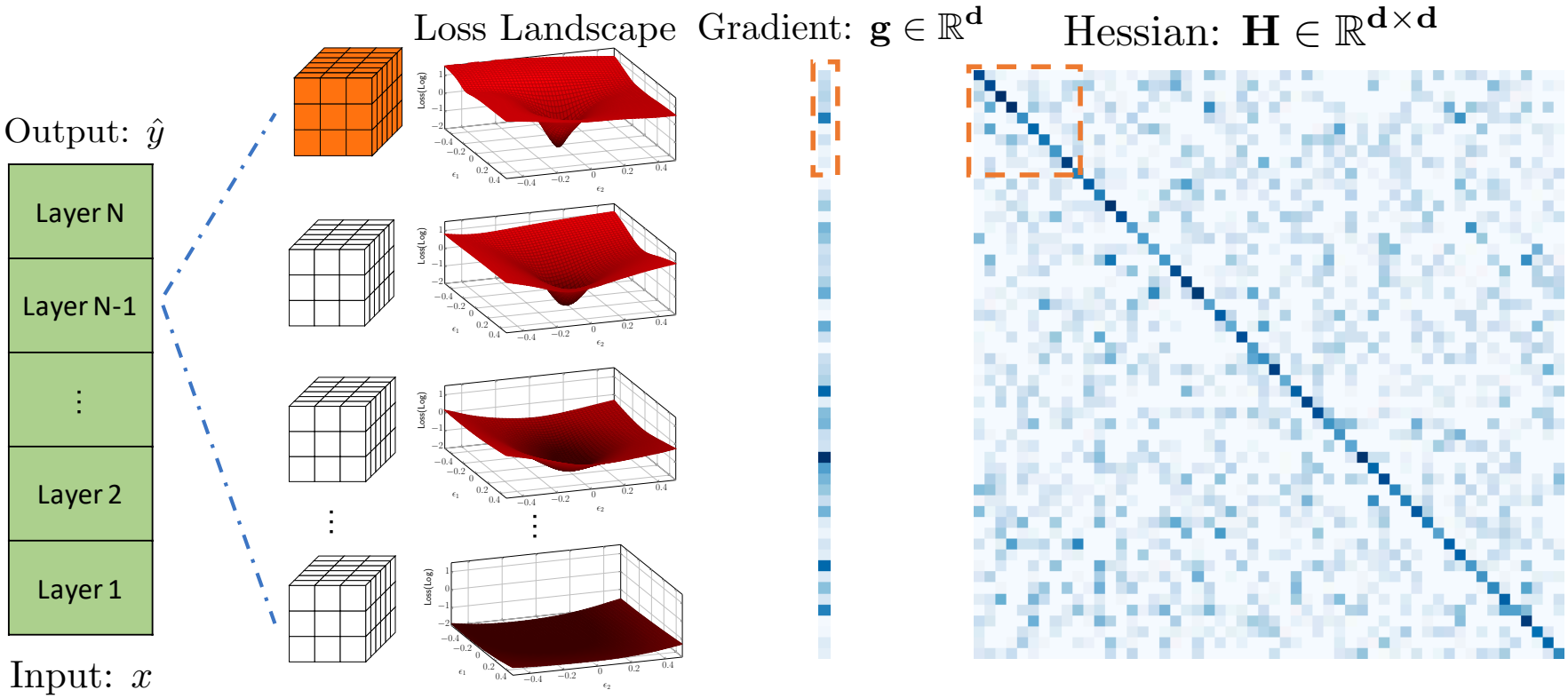
$\Delta\theta_t = H_t^{-1} g_t$

Mixture Form

Instead of using fully first or second order method, the following formula is used: $\Delta\theta_t = H_t^{-k} g_t, \quad 0 \leq k \leq 1$

- For convex problem, since $g_t^T H_t^{-k} g_t \geq 0$, $H_t^{-k} g_t$ is a descent direction.
- For simple problems, computing H_t^{-k} is not a problem and it can be done by an eigen-decomposition.
- However, for large scale machine learning problems (e.g., DNNs), forming/storing Hessian are impractical.

Opening the Black Box with Second Derivative



Pearlmutter BA. Fast exact multiplication by the Hessian. Neural computation. 1994.

Z. Yao*, A. Gholami*, Q. Lei, K. Keutzer, M. W. Mahoney, Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18, 2018.

Z. Yao*, A. Gholami*, K. Keutzer, M. W. Mahoney, PyHessian: Neural Networks Through the Lens of the Hessian **Spotlight at ICML'20 workshop** on Beyond First-Order Optimization Methods in Machine Learning, 2020.

Code: https://github.com/amirgholami/PyHessian

Different Optimizers

Table 1: *Summary of the first and second moments used in different optimization algorithms for updating model parameters ($w_{t+1} = w_t - \eta m_t / v_t$). Here β_1 and β_2 are first and second moment hyperparameters.*

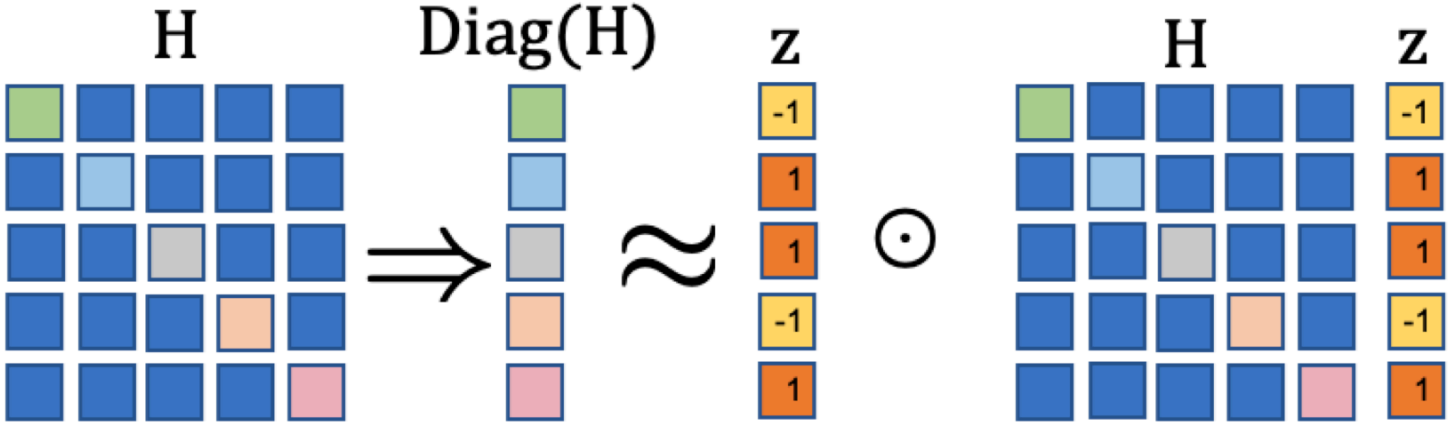
Optimizer	m_t	v_t
SGD [36]	$\beta_1 m_{t-1} + (1-\beta_1)\mathbf{g}_t$	1
Adagrad [16]	$\mathbf{g}_t$	$\sqrt{\sum_{i=1}^{t} \mathbf{g}_i \mathbf{g}_i}$
Adam [21]	$\frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \mathbf{g}_i}{1-\beta_1^t}$	$\sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \mathbf{g}_i \mathbf{g}_i}{1-\beta_2^t}}$
RMSProp [40]	$\mathbf{g}_t$	$\sqrt{\beta_2 v_{t-1}^2 + (1-\beta_2)\mathbf{g}_t \mathbf{g}_t}$
ADAHESSIAN	$\frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} \mathbf{g}_i}{1-\beta_1^t}$	$\left(\sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{D}_i^{(s)} \boldsymbol{D}_i^{(s)}}{1-\beta_2^t}}\right)^k$

H Robbins and S Monro. A stochastic approximation method. The annals of mathematical statistics, 1951
J Duchi, E Hazan, Y Singer. Adaptive subgradient methods for online learning and stochastic optimization, JMLR 2011
D Kingma and J Ba. Adam: A method for stochastic optimization, ICLR 2015
TTieleman and G Hinton. Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude, 2012
Z Yao, A Gholami, S Shen, M Mustafa, K Keutzer, MW Mahoney, ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning, arXiv: 2006.00719

How can we get Diagonal without explicitly forming the Hessian?

Randomized Numerical Linear Algebra (RandNLA):

$$D = diag(H) = \mathbb{E}[z \odot (Hz)], \quad z \sim Rademacher(0.5)$$



$$\text{Diag(H)} = \mathbb{E}[\text{z} \odot (\text{Hz})]$$
$$\text{s.t.} \quad \text{z} \sim \text{Rademacher(0.5)}$$

Bekas, C.; Kokiopoulou, E.; and Saad, Y. 2007. An estimator for the diagonal of a matrix. Applied numerical mathematics 57(11-12): 1214–1229.

How can we get Diagonal without explicitly forming the Hessian?

The remaining question is how to compute D_t ?

- Hessian-vector product:

$$\frac{\partial g^T z}{\partial \theta} = \frac{\partial g^T}{\partial \theta} z + g^T \frac{\partial z}{\partial \theta} = \frac{\partial g^T}{\partial \theta} z = Hz.$$

- Randomized numerical linear algebra (RandNLA):

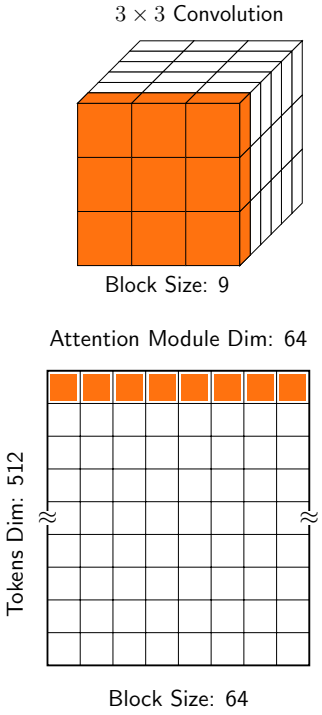
$$D = diag(H) = \mathbb{E}[z \odot (Hz)], \quad z \sim Rademacher(0.5)$$

- *Getting Hessian information takes roughly 2X backprop time!*

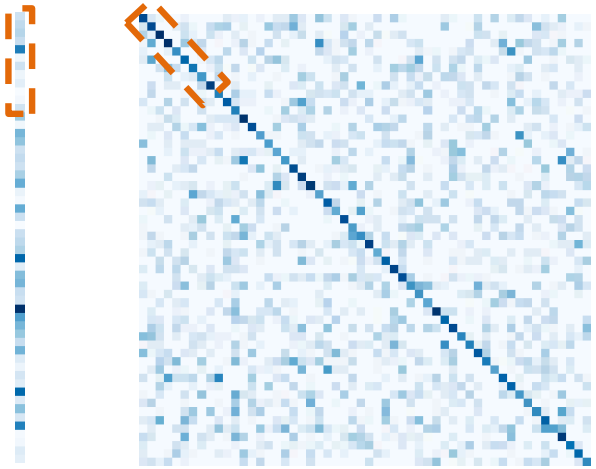
Pearlmutter BA. Fast exact multiplication by the Hessian. Neural computation. 1994.
Z. Yao*, A. Gholami*, Q. Lei, K. Keutzer, M. W. Mahoney, Hessian-based Analysis of Large Batch Training and Robustness to Adversaries, NeurIPS'18, 2018.
Z. Yao*, A. Gholami*, K. Keutzer, M. W. Mahoney, PyHessian: Neural Networks Through the Lens of the Hessian **Spotlight at ICML'20 workshop** on Beyond First-Order Optimization Methods in Machine Learning, 2020.
Code: https://github.com/amirgholami/PyHessian

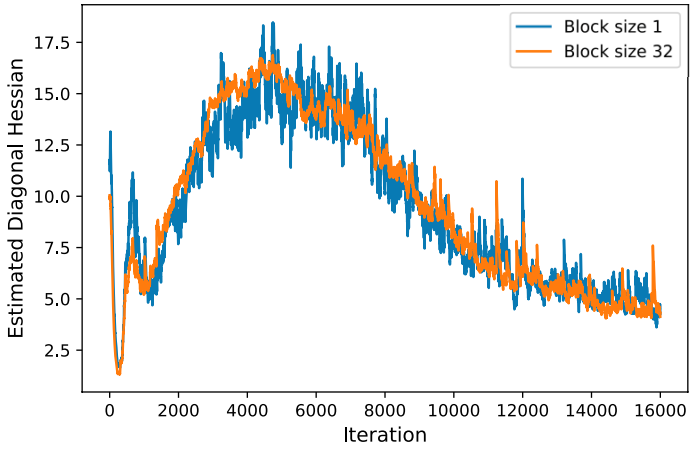
Spatial Smoothing

- We also incorporate spatial averaging to smooth out the stochastic Hessian noise across different iterations



Gradient: $\mathbf{g} \in \mathbb{R}^{\mathbf{d}}$ Hessian: $\mathbf{H} \in \mathbb{R}^{\mathbf{d} \times \mathbf{d}}$





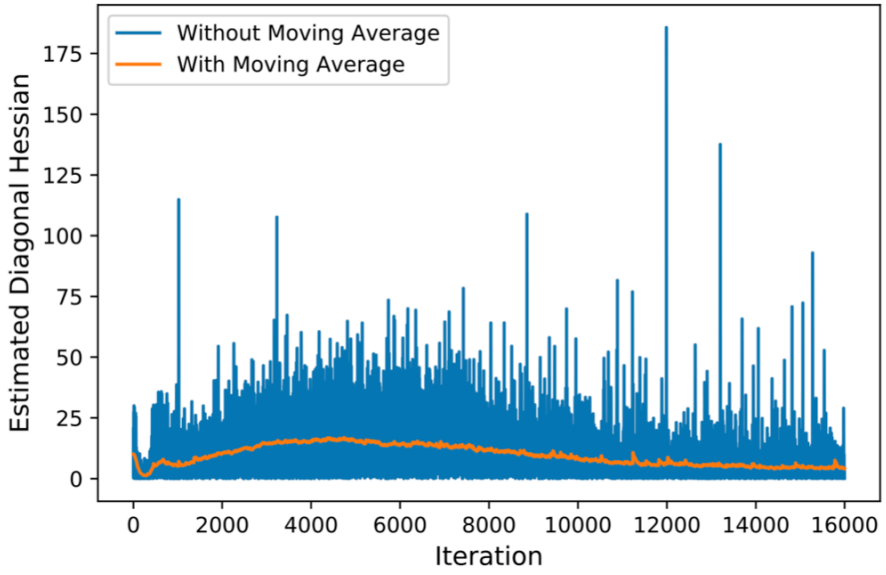
Machine Translation Task on IWSLT'14 Dataset

Examples of averaging for convolution (top, for CV) and multi-head attention (bottom, for NLP)

Variance Reduction

- Incorporating momentum for both first and second order term:

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i}g_i}{1-\beta_1^t}, \quad v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}D_iD_i}{1-\beta_2^t}}.$$



AdaHessian Algorithm

Algorithm 1: ADAHESSIAN

Require: Initial Parameter: θ_0
Require: Learning rate: η
Require: Exponential decay rates: β_1, β_2
Require: Block size: b
Require: Hessian Power: k
Set: $\bar{\mathbf{g}}_0 = 0$, $\bar{\boldsymbol{D}}_0 = 0$
for $t = 1, 2, \ldots$ **do** `// Training Iterations`
  $\mathbf{g}_t \leftarrow$ current step gradient
  $\boldsymbol{D}_t \leftarrow$ current step estimated diagonal Hessian
  Update m_t, v_t based on Eq. 10
  $\theta_t = \theta_{t-1} - \eta v_t^{-k} m_t$

Important Points for Empirical Results

- What hyper-parameters we modified in the experiments:
  - Fixed learning rate
  - Space averaging block size
- What hyper-parameters we did not modify in the experiments:
  - Learning rate schedule
  - Weight decay
  - Warmup schedule
  - Dropout rate
  - First and second order momentum coefficients, β_1/β_2

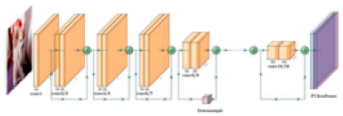
Some related Work: pyHessian

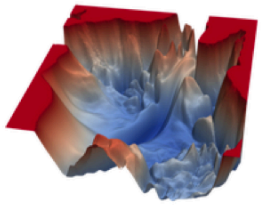
PyHESSIAN

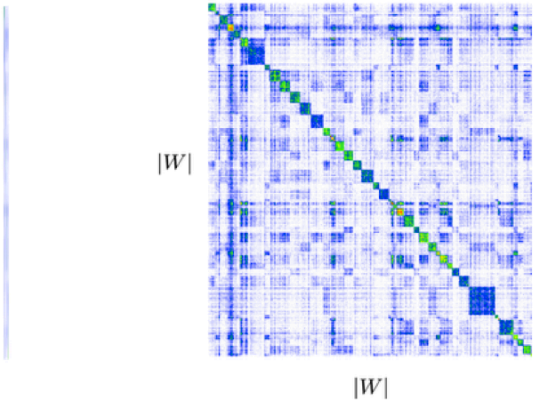
$$\min_w E(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$

Gradient: $\frac{\partial E}{\partial w} \in \mathcal{R}^{|W|}$

Hessian: $\frac{\partial^2 E}{\partial w^2} \in \mathcal{R}^{|W|\times|W|}$







Introduction

PyHessian is a pytorch library for Hessian based analysis of neural network models. The library enables computing the following metrics:

- Top Hessian eigenvalues
- The trace of the Hessian matrix
- The full Hessian Eigenvalues Spectral Density (ESD)

Compute lots of Hessian information for:

- Training (ADAHESSIAN)
- Quantization (HAWQ, QBERT)
- Inference

Also for:

- Validation: loss landscape
- Validation: model robustness
- Validation: adversarial data
- Validation: test hypotheses

Outline

Stochastic optimizers

In deep learning...

- Stochastic gradient descent (SGD)

$$w_{k+1} = w_k - \frac{\gamma}{|\Omega_k|} \sum_{i \in \Omega_k} \nabla f_i(w_k)$$

- Momentum
- Stochastic Newton methods
- Adam
- and *many* others...

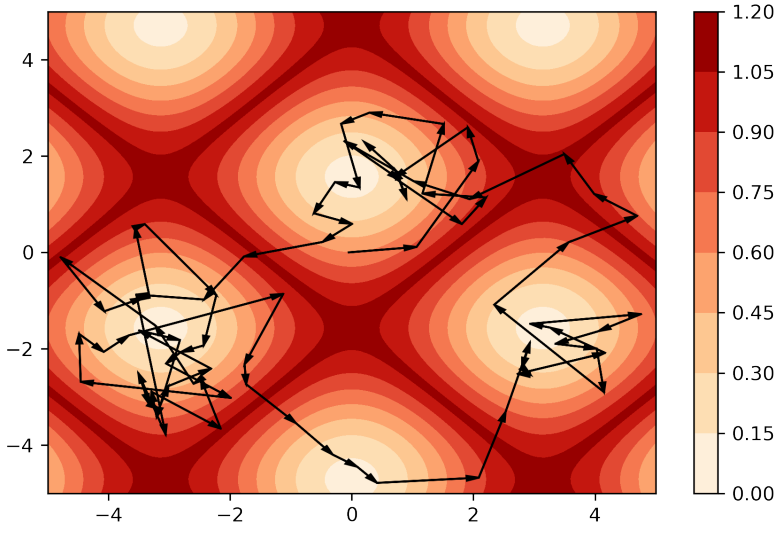
Based on classical (convex) optimization algorithms.

Stochastic component (minibatches) can allow them to work well in unconstrained *non-convex* settings.

Robbins, H., Monro, S. (1951) A stochastic approximation method. The Annals of Mathematical Statistics, pp.400-407

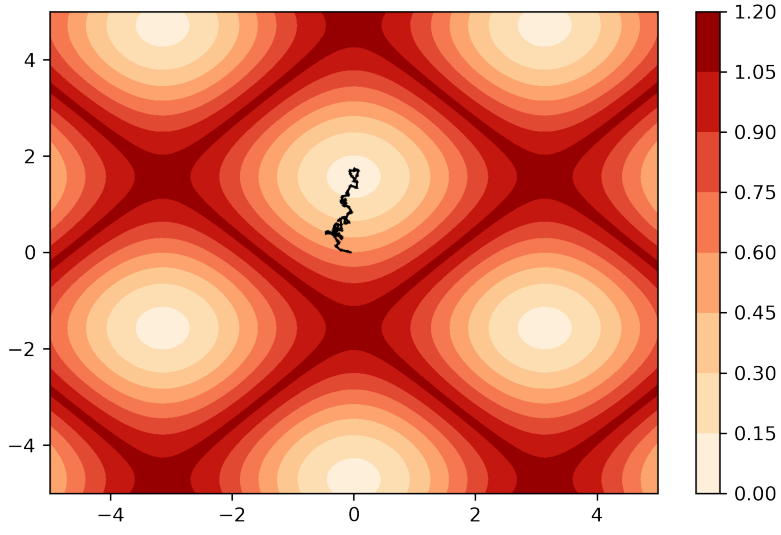
Phases of Training

Exploration
large learning rate



(sampler)

Exploitation
small learning rate



(optimizer)

Mandt, S., Hoffman, M., Blei, D. (2016) A variational analysis of stochastic gradient algorithms. ICML 2016, pp. 354–363.

A distributional approach

Investigate how a stochastic optimizer explores the loss landscape

1. Model stochastic optimization as a random dynamical system (Markov)
2. Fix all hyperparameters to particular values (time-homogeneous; no annealing)
3. Examine properties of the **stationary (invariant) distribution**

- Avoid continuous-time approximations

Our Findings

Multiplicative noise results in heavy-tailed stationary behaviour

- Tails of the stationary distribution are an indication of capacity to explore
- Decay rates in the tails that are slower than exponential are **heavy**, e.g.

$$\mathbb{P}(W > t) \approx ct^{-\alpha}$$

Heavy tails are significant

Recent efforts have empirically tied the presence of strong heavy tails during training with good generalization performance.

- Simsekli, U., Sagun, L., Gürbüzbalaban, M. (2019). A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks
- Martin, C. H., Peng, T., Mahoney, M. W. (2020). Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data.

Heavier tails imply wider exploration

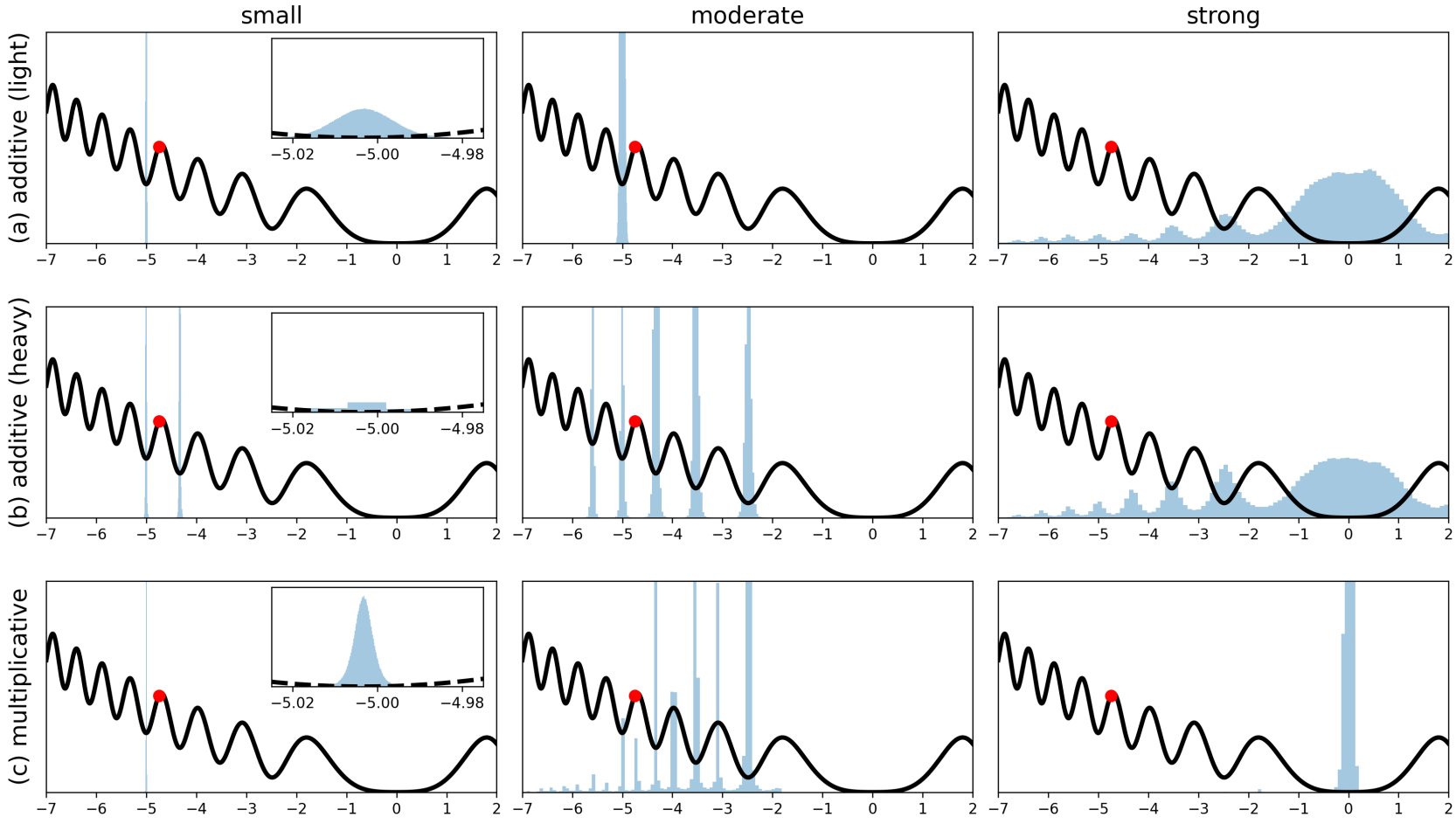


Figure: Histograms of 10^6 iterations of GD with combinations of small, moderate, and strong vs. light additive, heavy additive, and multiplicative noise, applied to a **non-convex objective** & initial starting location for the optimization.

Generalization

Does heavy-tailed exploration imply better generalization? **Yes.**

Theorem (Simsekli et al., 2020)

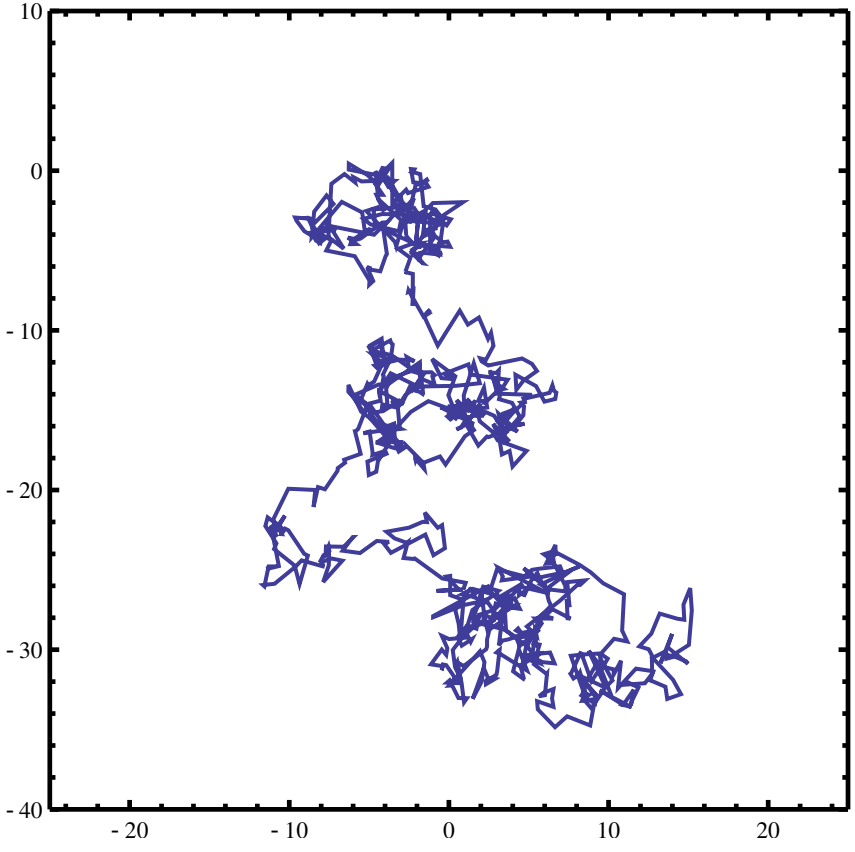
For a process W_t with **Hausdorff dimension** d_{H} (decreases as W_t exhibits more heavy-tailed fluctuations)

$$\sup_{t\in[0,1]} |\hat{\mathcal{R}}_n(W_t) - \mathcal{R}(W_t)| \leq C_{\mathcal{R}} \sqrt{\frac{d_{\mathrm{H}} \log n}{n} + \frac{\log(1/\gamma)}{n}}$$

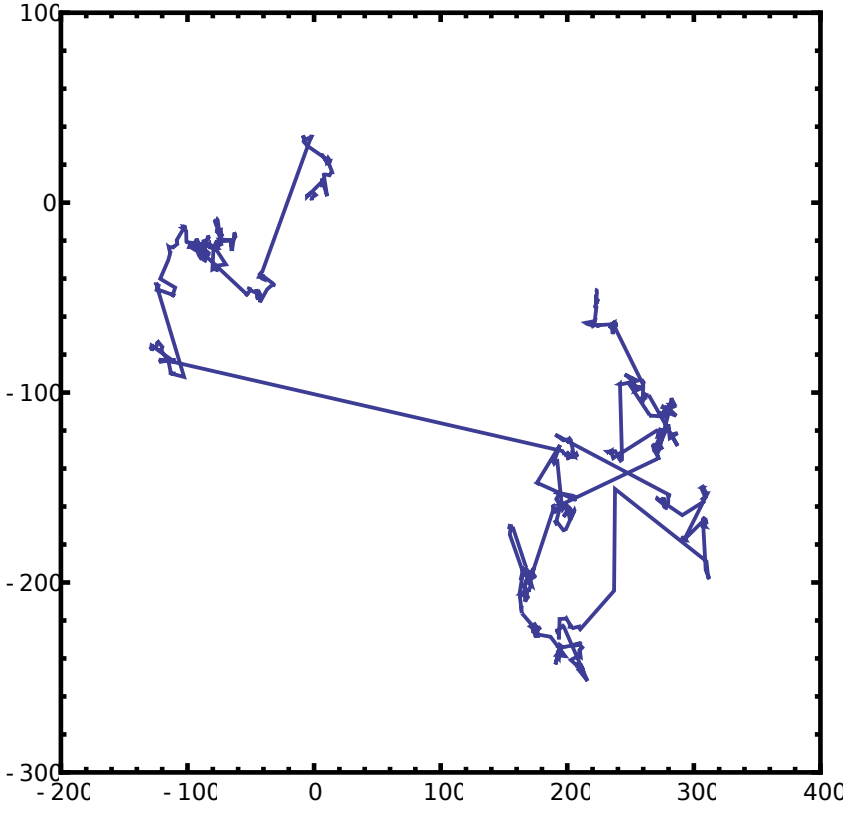
with probability $1-\gamma$.

Generalization

Brownian motion
light-tailed



Levy flight
heavy-tailed



Establishing heavy tails

Ridge regression

Consider least squares linear regression with L^2 regularization:

$$M^* = \underset{M \in \mathbb{R}^{d \times m}}{\arg\min} \frac{1}{2}\mathbb{E}\| Y - MX \|^2 + \frac{1}{2}\lambda \| M \|_F^2,$$

where

- $X \in \mathbb{R}^d$ are the inputs
- $Y \in \mathbb{R}^m$ are the labels

Ridge regression

Lemma

The iterates M_k of **minibatch SGD** satisfy the following: for $W_k = \text{vec}(M_k)$,

$$W_{k+1} = A_k W_k + B_k,$$

where

$$A_k = I \otimes \left((1-\lambda) I - \gamma n^{-1} \sum_{i=1}^{n} X_{ik} X_{ik}^\top \right), \quad B_k = -\gamma n^{-1} \sum_{i=1}^{n} Y_{ik} X_{ik}^\top$$

There is both **additive** and **multiplicative** noise.

Kesten (1973): $\mathbb{P}(\|A_k\| > 1) > 0 \implies$ heavy tails

The Kesten mechanism

Heavy tails (power laws) arise gradually **over time** due to the presence of noise on **multiple scales**

$$W_{k+1} = f_k(W_k) \approx A_k W_k + B_k$$

A_k	B_k
logarithmic scale multiplicative noise $D^1 f_k$	linear scale additive noise $D^0 f_k$

General stochastic optimization

In machine learning, solving problems of the form

$$w^* = \arg\min_w f(w), \quad f(w) := \mathbb{E}_{\mathcal{D}} \ell(w, X),$$

for a loss ℓ depending on weights w and data X from some dataset $\mathcal{D}$.

General stochastic optimization

Fixed point iteration: if Ψ is chosen such that fixed points of $\mathbb{E}_{\mathcal{D}}\Psi(\cdot, X)$ are minimizers of f, then

$$w_{k+1} = \mathbb{E}_{\mathcal{D}}\Psi(w_k, X)$$

either diverges, or converges to w^*.

General stochastic optimization

Estimating the expectation gives a **stochastic optimizer:**

$$W_{k+1} = \frac{1}{n} \sum_{i=1}^{n} \Psi(W_k, X_{ik}), \qquad X_{ik} \overset{\text{iid}}{\sim} X$$

where X_{ik} is the i-th datum from the k-th minibatch.

- Assuming data is shuffled in each epoch
- Forms a time-homogeneous Markov chain for fixed hyperparameters

Stochastic optimization as a Markov chain

The sequence of **iterated random functions**

$$W_{k+1} = \Psi(W_k, X_k) \qquad X_k \overset{\text{iid}}{\sim} X.$$

Equivalently, as a root-finding problem:

$$W_{k+1} = W_k - \tilde{\Psi}(W_k, X_k) \qquad \text{(Borovkov)}$$

Assume this Markov chain is **ergodic**.


Diaconis, P., Freedman, D. (1999) Iterated Random Functions. SIAM Review. 41(1), 45–76.


Alsmeyer, G. (2003) On the Harris recurrence of iterated random Lipschitz functions and related convergence rate results. Journal of Theoretical Probability, 16(1):217247,

Every iterative stochastic optimization algorithm in ML (with fixed hyperparameters) can be written as a Markov chain in this way.

SGD & SGD with momentum

Minibatch SGD: For minibatch size n and step size γ,

$$\Psi(w, X) = w - \gamma n^{-1} \sum_{i=1}^{n} \nabla \ell(w, X_i).$$

Momentum: Incorporating velocity v,

$$\Psi\left(\begin{pmatrix} v \\ w \end{pmatrix}, X\right) = \frac{1}{n} \sum_{i=1}^{n} \begin{pmatrix} \eta v + \nabla \ell(w, X_i) \\ w - \gamma(\eta v + \nabla \ell(w, X_i)) \end{pmatrix}$$

Main Result

Theorem

Suppose X is non-atomic and there exist k_Ψ, K_Ψ, M_Ψ, w^* such that as $\|w\| \to \infty$,

$$k_\Psi(X) - o(1) \leq \frac{\|\Psi(w, X) - \Psi(w^*, X)\|}{\|w - w^*\|} \leq K_\Psi(X) + o(1).$$

If $\mathbb{P}(k_\Psi(X) > 1) > 0$ and $\mathbb{E} \log K_\Psi(X) < 0$, for some $\mu, \nu, C_\mu, C_\nu > 0$,

$$C_\mu(1 + t)^{-\mu} \leq \mathbb{P}(\|W_\infty\| > t) \leq C_\nu t^{-\nu}.$$

II. Factors influencing tail behaviour

Run SGD w/ constant step size on two-layer NN with L^2 loss using Wine Quality UCI dataset.

$\hat{\alpha}$ is an estimate of the tail exponent α such that

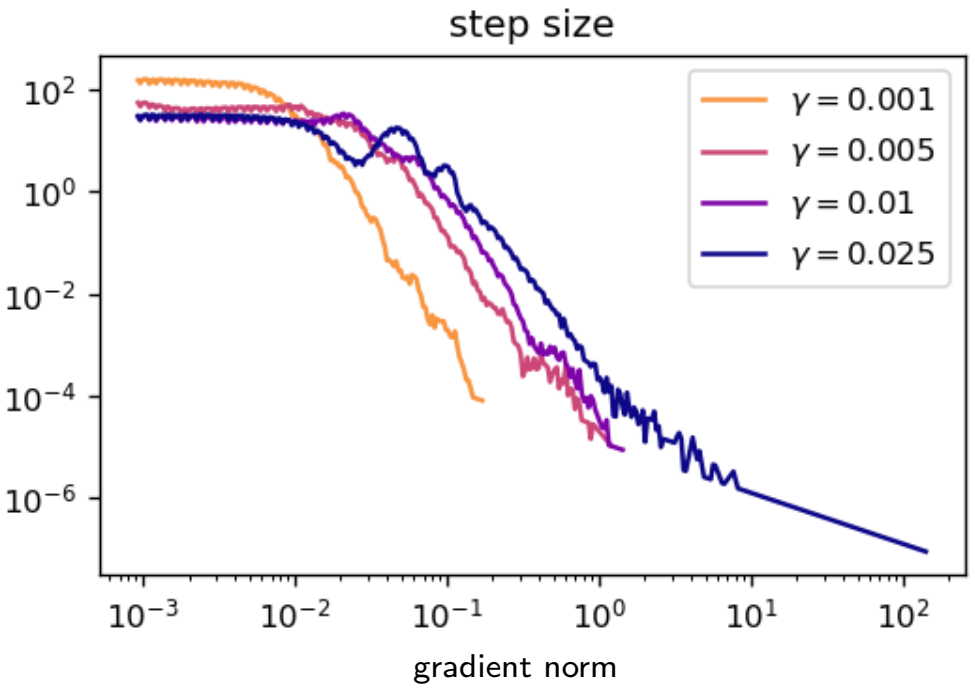
$$\mathbb{P}(\|D_\infty\| > t) \approx ct^{-\alpha}$$

- for fluctuations $D_k \doteq W_{k+1} - W_k$ (for SGD, corresponds to **gradient norm**)
- $D_\infty = \lim_{k\to\infty} D_k$ has the same tail exponent as W_k

Factors: step size

Prediction: larger step sizes $\Longrightarrow$ heavier tails

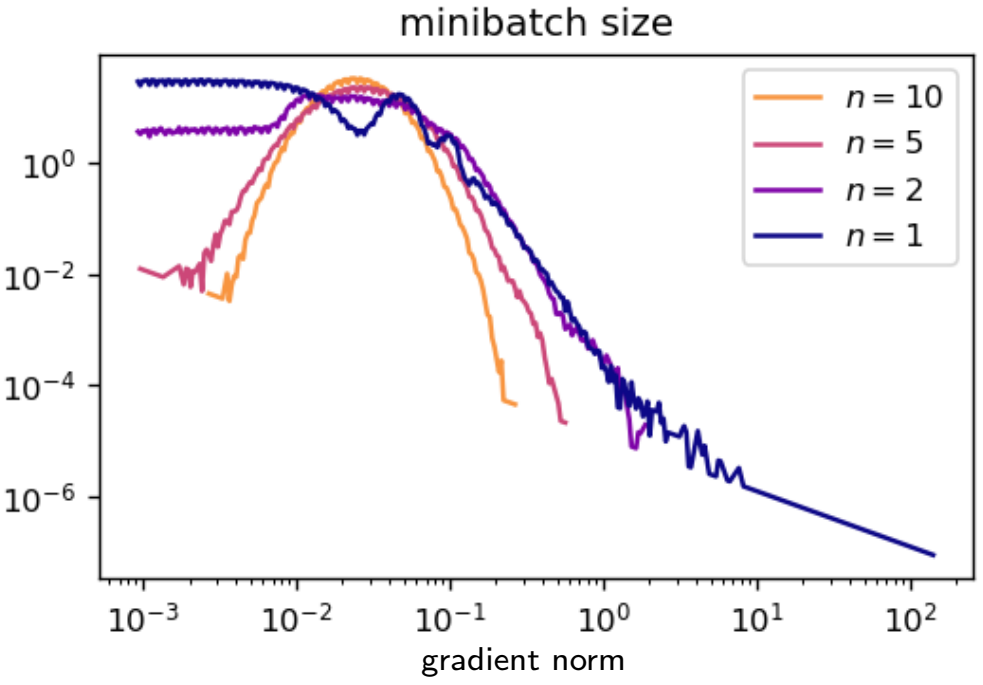
step size	
γ	$\hat{\alpha}$
0.001	4.12 ± 0.04
0.005	3.70 ± 0.02
0.01	3.71 ± 0.04
0.025	2.97 ± 0.03



Factors: minibatch size

Prediction: smaller batch sizes $\Longrightarrow$ heavier tails

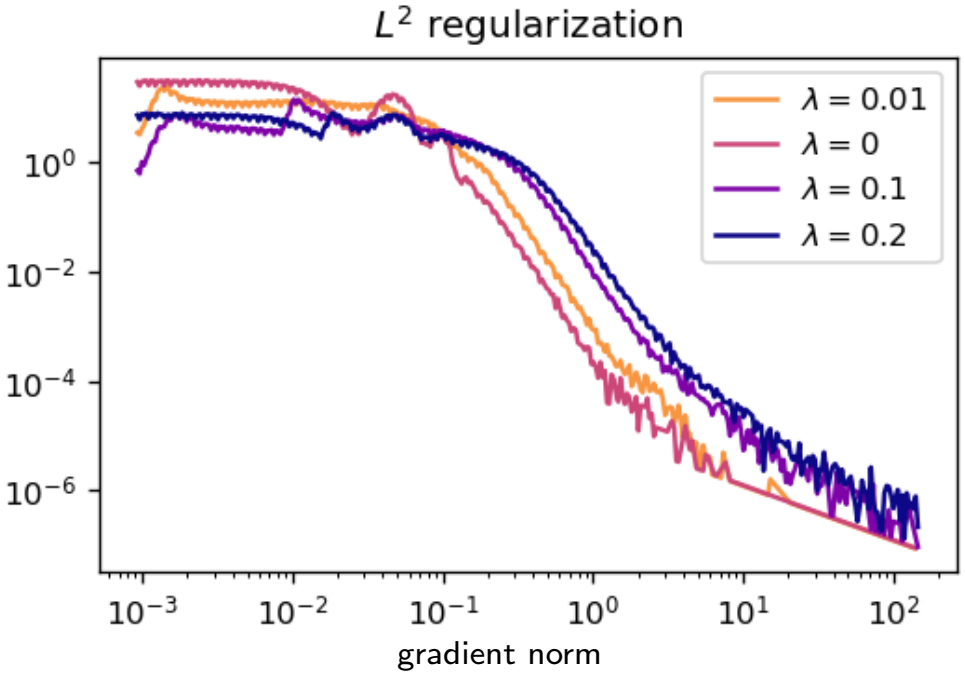
minibatch size	
n	$\hat{\alpha}$
10	5.99 ± 0.05
5	4.98 ± 0.07
2	3.62 ± 0.03
1	2.97 ± 0.03



Factors: L^2 regularization

Prediction: more regularization $\Longrightarrow$ heavier tails

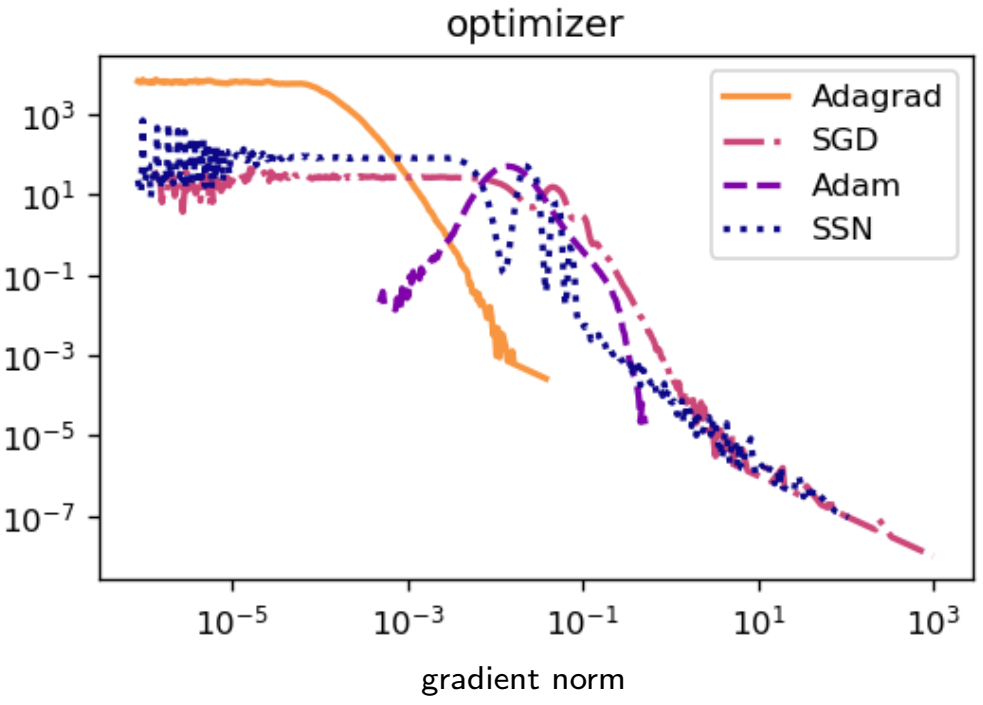
L^2 regularization	
λ	$\hat{\alpha}$
10^{-4}	2.97 ± 0.03
0.01	3.02 ± 0.02
0.1	2.77 ± 0.01
0.2	2.55 ± 0.01



Factors: optimizer

Prediction: SGD, SSN heavier than Adagrad, Adam

optimizer	
	$\hat{\alpha}$
Adagrad	3.2 ± 0.1
Adam	2.119 ± 0.005
SGD	2.93 ± 0.03
SSN	0.79 ± 0.04



Factors: depth

The theory isn't particularly informative

Empirically, however...

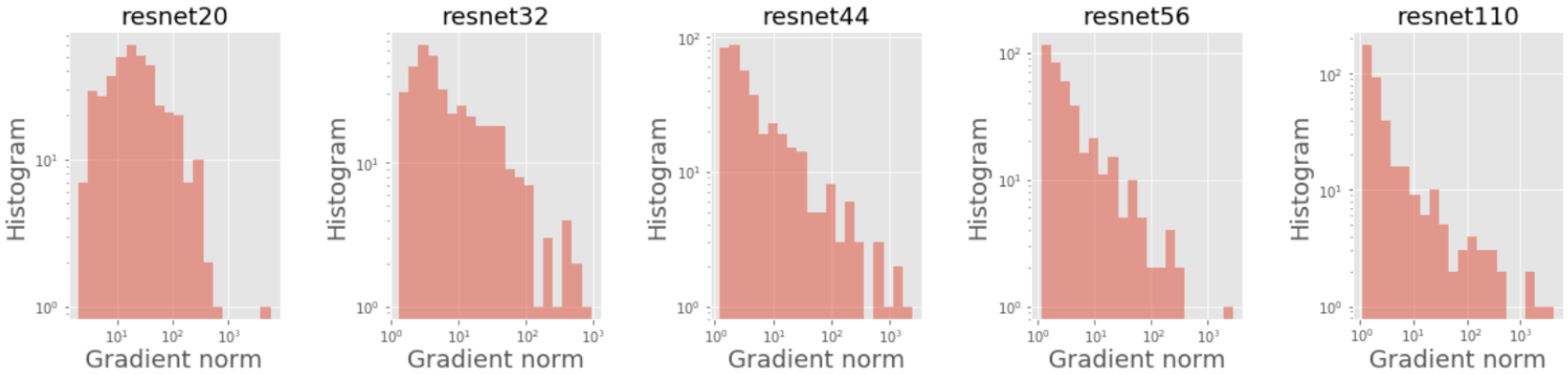


Figure: Histograms of gradient norms for varying architectures. Courtesy of Yaoqing Yang.

So it seems greater depth $\implies$ heavier tails

Summary

Multiplicative noise is a critical element for understanding performance of stochastic optimizers

- Results in heavy-tailed stationary behaviour
- Far-reaching, but efficient, exploration

Future work:

- Improve precision for tail exponent estimates in more specific models (e.g. deep neural nets)
- The Kesten mechanism in the spectral domain
- Generalization bounds in discrete time

Outline

Conclusions

- Theory for second-order stochastic optimization
  - Faster and similar convergence for convex-like problems
- Practice for second-order stochastic optimization
  - Implementations and downstream use cases are *very* different
  - Existing theory often fail to provide even qualitative guidance
- Theory for second-order stochastic optimization
  - Relate to Markov processes and random recurrence relations
- Other possible theoretical approaches make connections to
  - Dynamical systems more generally
  - Non-asymptotic randomized linear algebra
  - Heavy-tailed random matrix theory