Recent Developments in the Theory/Practice of Second Order Optimization for Machine Learning

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

October 2021

(Joint work with Z. Yao, **A. Gholami**, S. Shen, M. Mustafa, and K. Keutzer (ADAHESSIAN), **M. Derezinski**, B. Bartan, and M. Pilanci (Inversion Bias), and others (Recent Stuff in the Pipeline).

Outline

ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning

Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer,

Michael W. Mahoney

September 2020






Berkeley
UNIVERSITY OF CALIFORNIA

riselab
UC Berkeley

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!**

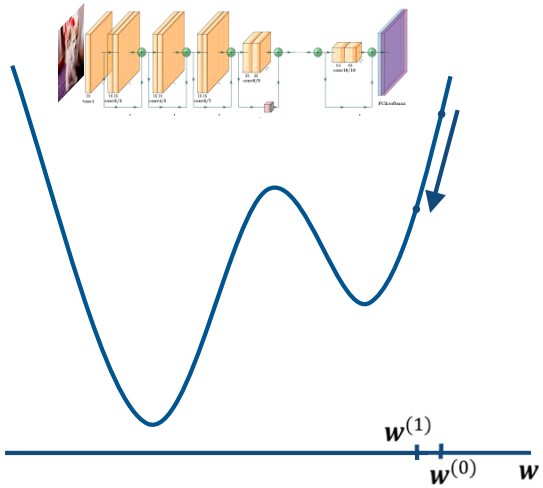
Task	CV	NLP	Recommendation System
Optimizer Choice	SGD	AdamW	Adagrad

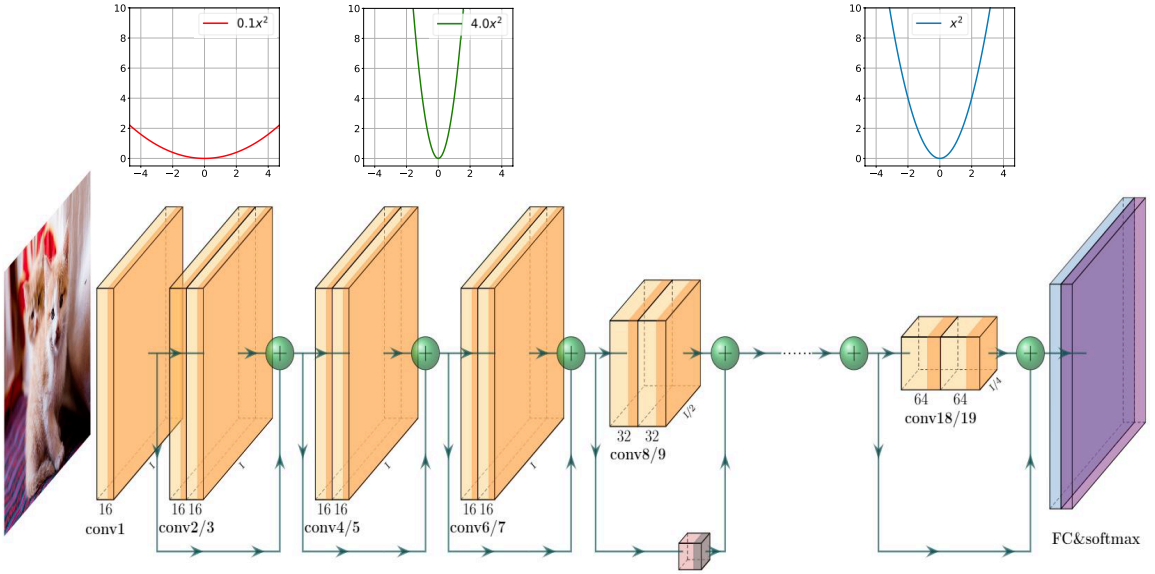
*BTW, not obvious if you just do popular things, e.g., ResNet50 training on ImageNet, since years of industrial scale (i.e., brute force) hyperparameter tuning and building systems for SGD-based methods mean those methods do well ...

SGD Based Training

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

$$w^1 = w^0 - \frac{\lambda}{B}\sum_{i=1}^{B}\frac{\partial E_i(w^0)}{\partial w}$$



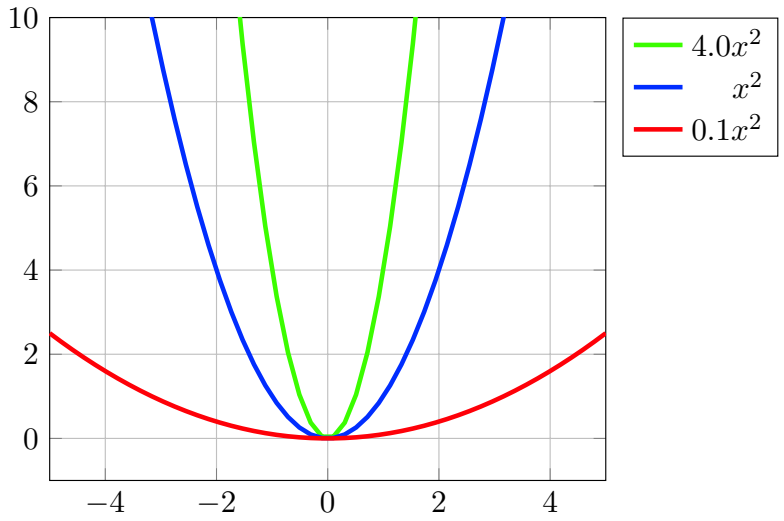


First and Second Order Methods

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

First Order Method — Second Order Method

$\Delta w_t = g_t$



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

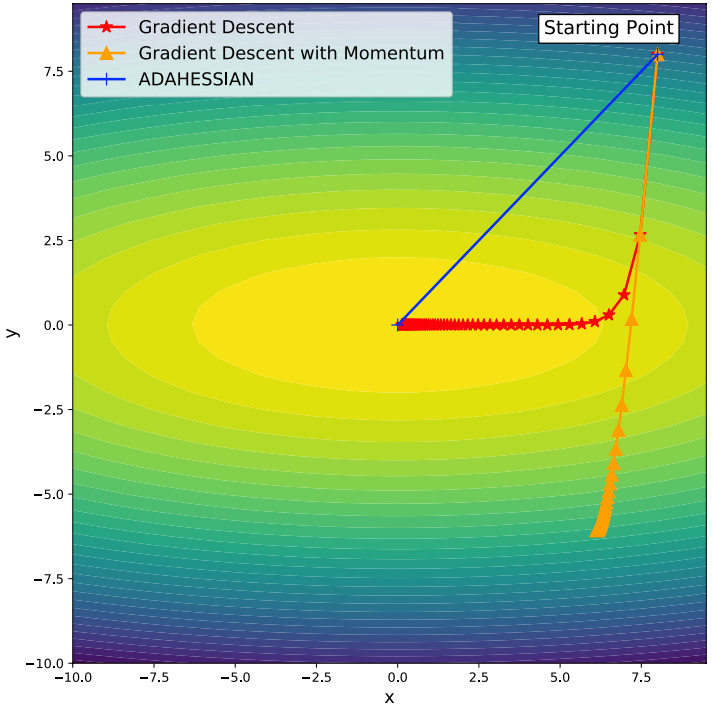
- At the origin, the first derivative of $y = 4x^2$, $y = x^2$, $y = 0.1\,x^2$ is all the same: 0
- The **second derivative** give more information: 8 , 2, and 0.2 respectively

First and Second Order Methods

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

First Order Method

$\Delta w_t = g_t$



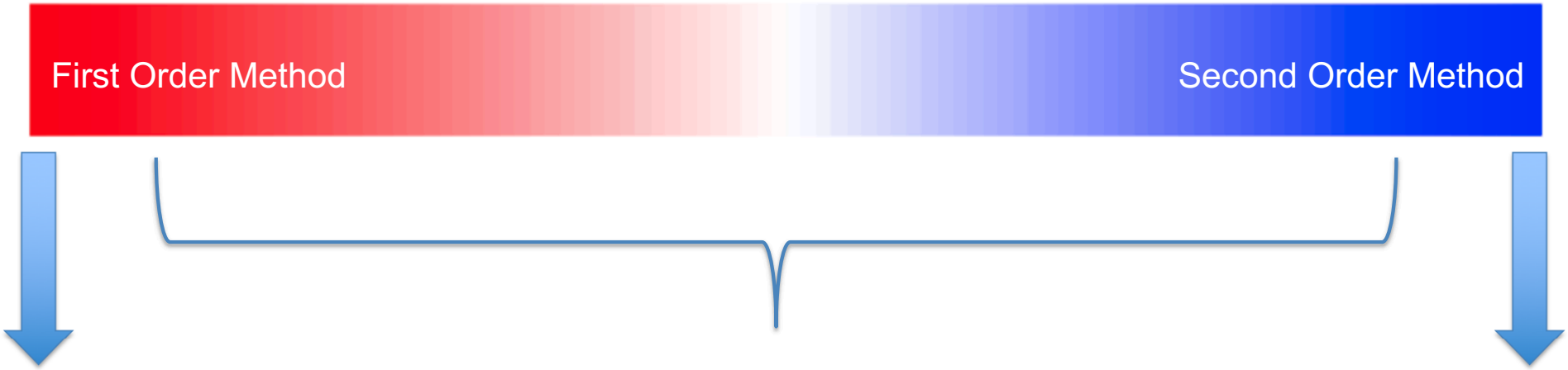
Second Order Method

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

$f = x^2 + 10y^2$

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?

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

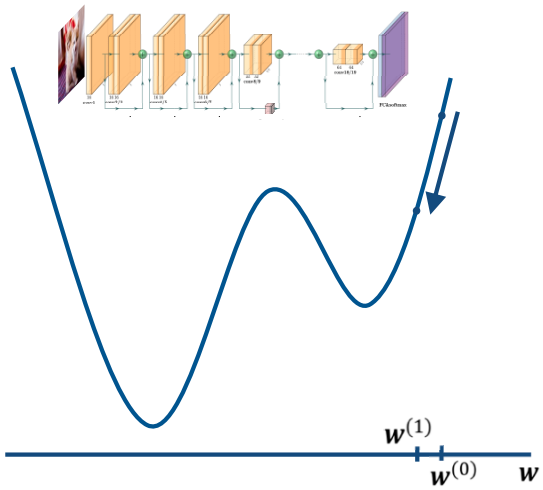
Mixture Form

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.

Second Derivative (Hessian)

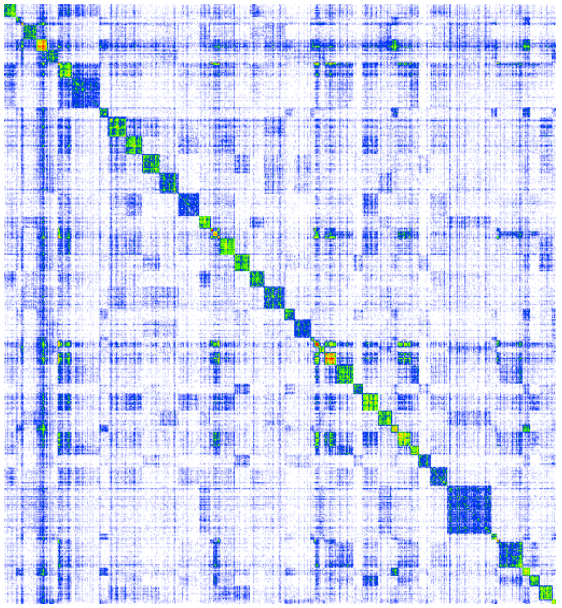
$$\min_w E(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$



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

$|W|$

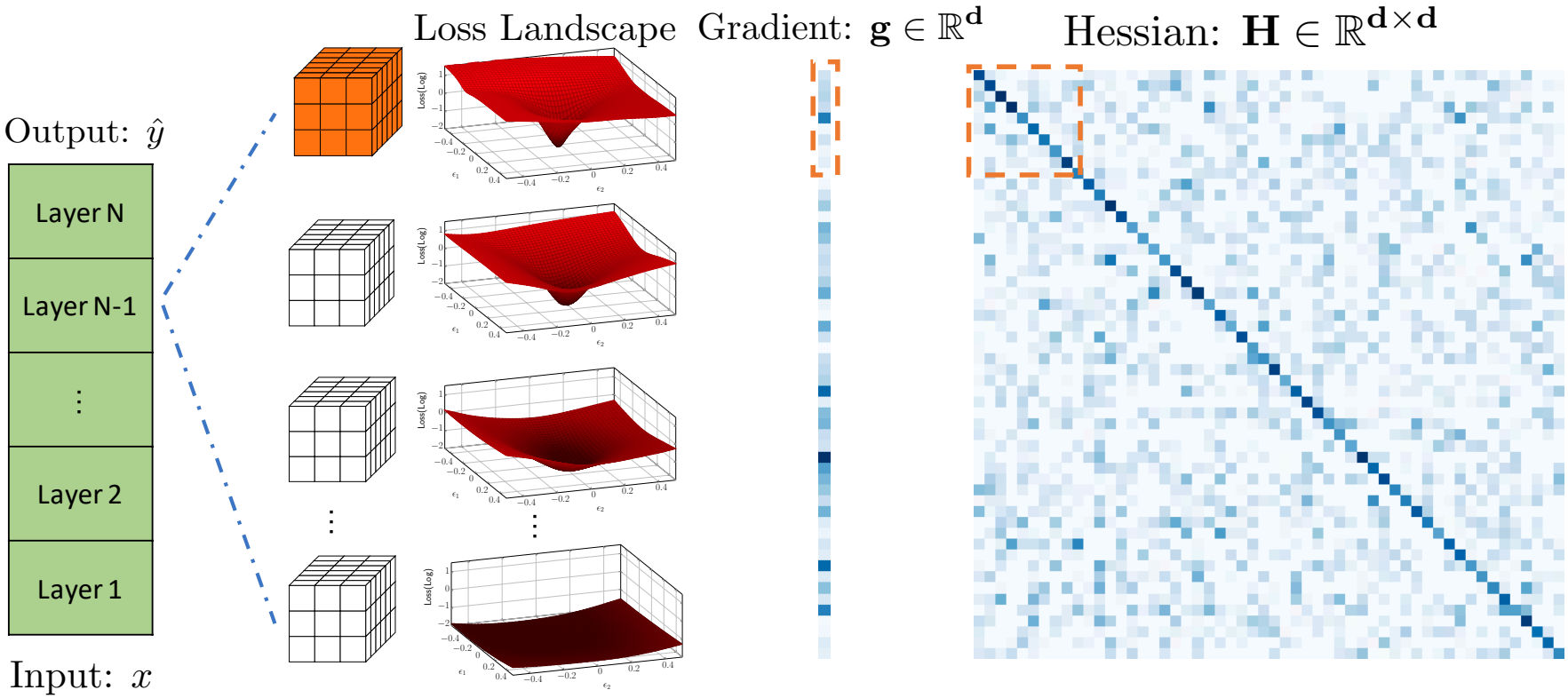
Hessian: $\frac{\partial^2 E}{\partial w^2} \in \mathcal{R}^{|W|\times|W|}$



$|W|$

$|W|$

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

Using Hessian Diagonal

Forming the Hessian is infeasible:

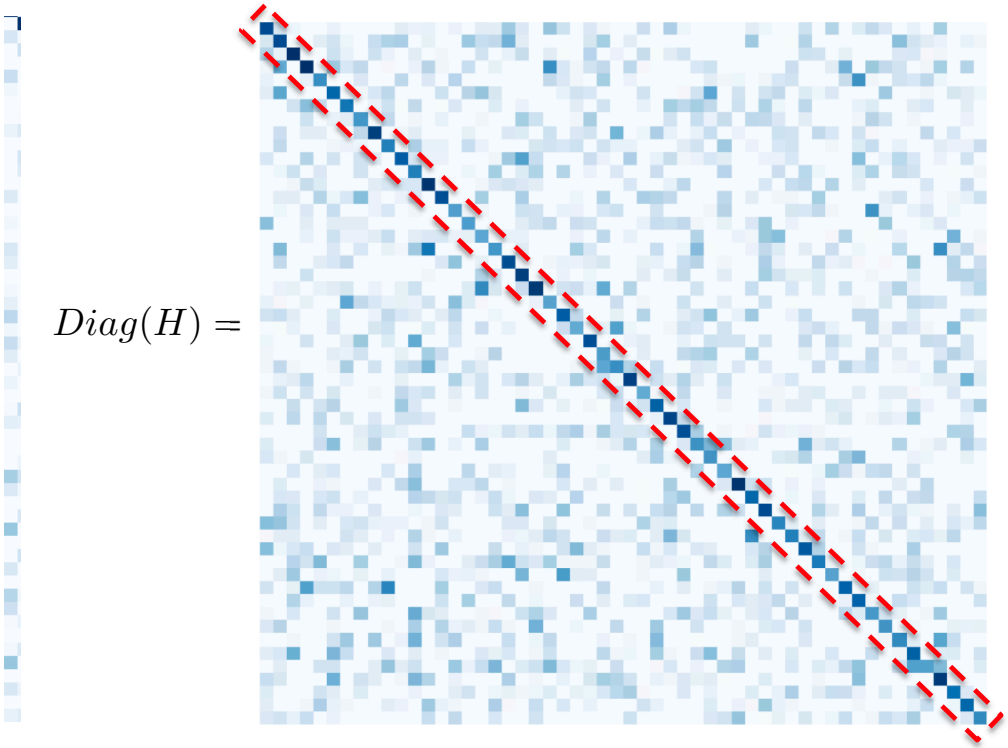
For ResNet50 (with 24M parameters)

Hessian is a matrix of size **24Mx24M**

What if we approximate the Hessian?

Idea: Use Hessian diagonal

$$g = \quad Diag(H) =$$



Pearlmutter BA. Fast exact multiplication by the Hessian. Neural computation. 1994.
Costas Bekas, Effrosyni Kokiopoulou, and Yousef Saad. An estimator for the diagonal of a matrix. Applied numerical mathematics, 57(11-12):1214– 1229, 2007
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 Workshop, 2020.
Code: https://github.com/amirgholami/PyHessian

AdaHessian

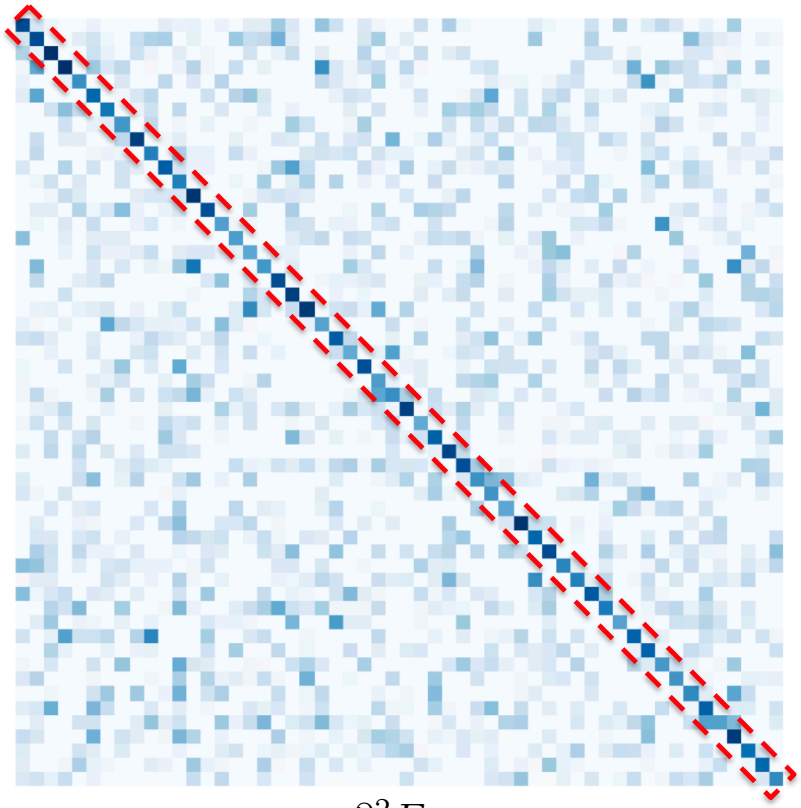
ADAHESSIAN algorithm is very simple and as follows:

$$w_{t+1} = w_t - \eta_t m_t / v_t,$$

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} g_i}{1-\beta_1^t},$$

$$v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} D_i D_i}{1-\beta_2^t}}.$$

Where D is the Hessian diagonal



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

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

Is computing H^{-1} practical? Of course not …

For ResNet50:

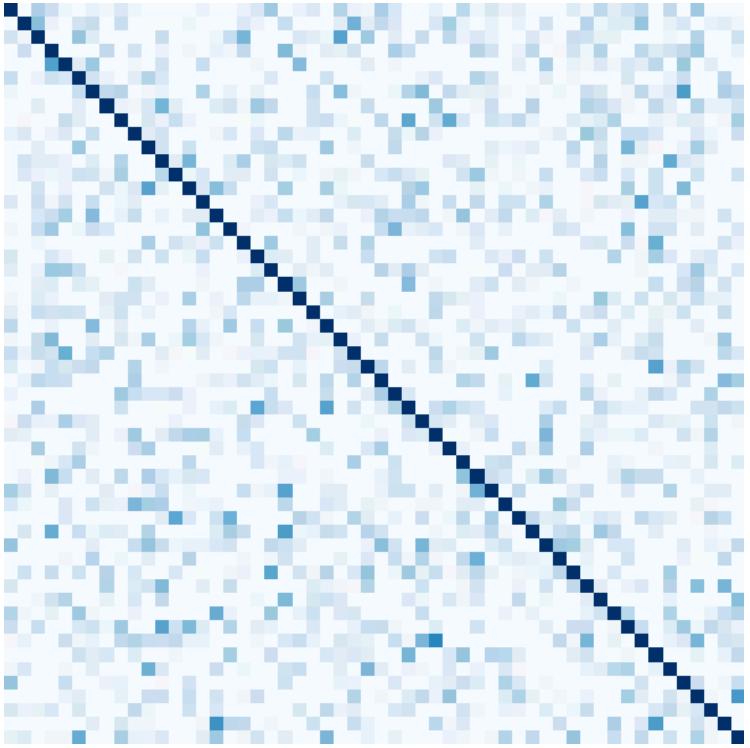
- # Parameters is 24M.
- |g| = 24M ~ 100 MB
- |H| = 24Mx24M ~ 2.4 PB

Can we:

- compute H?
- store H?
- compute H^{-1}?

Of course not ...

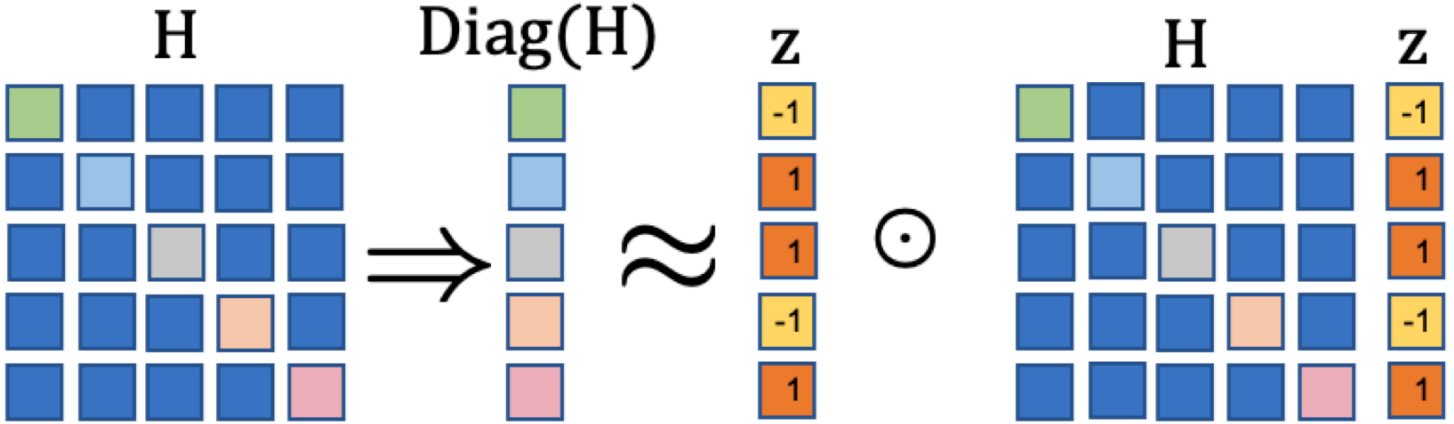
$g =$ 

$H =$ 

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

AdaHessian

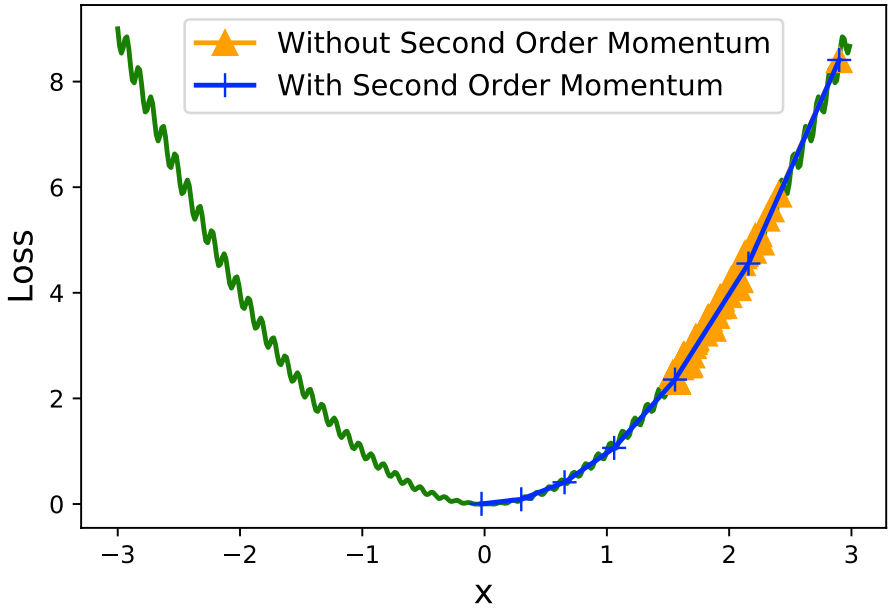
ADAHESSIAN algorithm is very simple and as follows:

$$w_{t+1} = w_t - \eta_t m_t / v_t,$$

$$m_t = \frac{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i} g_i}{1-\beta_1^t},$$

$$v_t = \sqrt{\frac{(1-\beta_2)\sum_{i=1}^{t} \beta_2^{t-i} D_i D_i}{1-\beta_2^t}}.$$

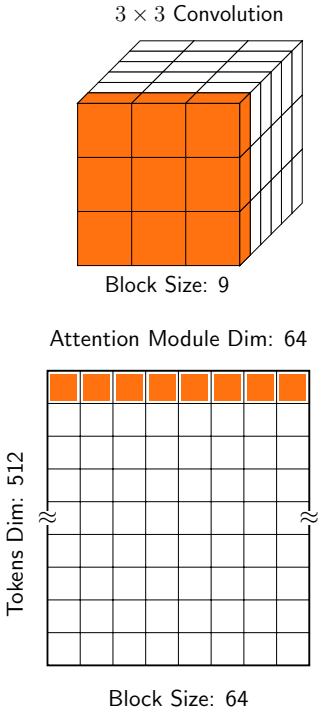
Where D is the Hessian diagonal



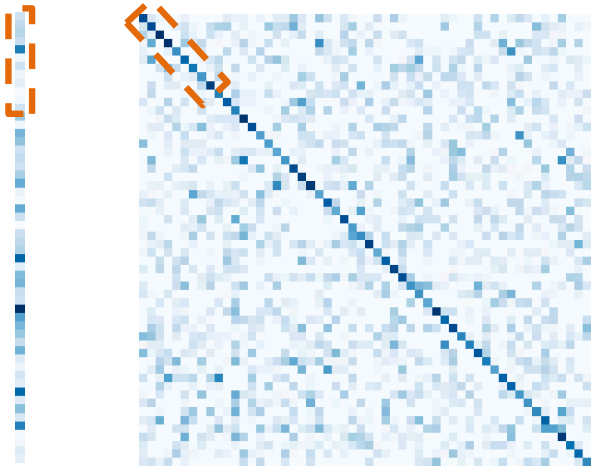
$$f(x) = x^2 + 0.1x\sin(x)$$

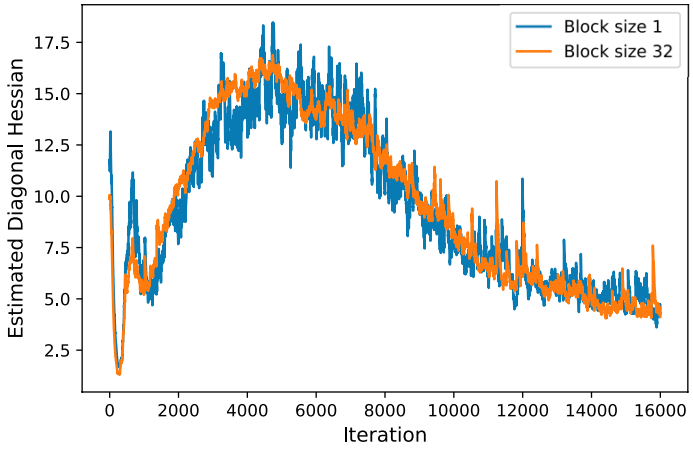
Spatial Smoothing

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



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





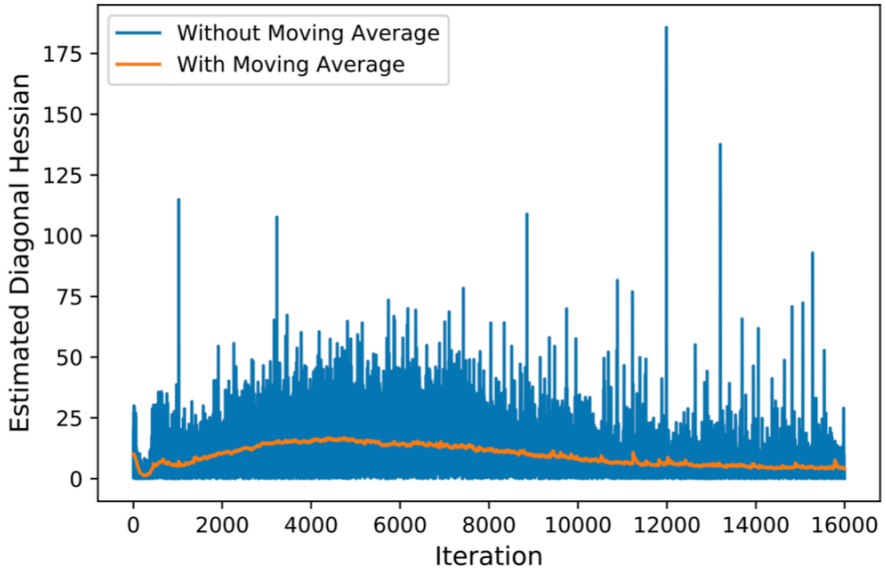
Machine Translation Task on IWSLT'14 Dataset

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

Results on Image Classification

Only learning rate and space averaging block size are tuned for ADAHESSIAN

Higher is better

Dataset	Cifar10		ImageNet
	ResNet20	ResNet32	ResNet18
SGD [36]	92.08 ± 0.08	$\mathbf{93.14 \pm 0.10}$	70.03
Adam [19]	90.33 ± 0.13	91.63 ± 0.10	64.53
AdamW [22]	91.97 ± 0.15	92.72 ± 0.20	67.41
ADAHESSIAN	$\mathbf{92.13 \pm 0.18}$	93.08 ± 0.10	**70.08**

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

Results on Machine Translation

Only learning rate and space averaging block size are tuned for ADAHESSIAN

Higher BLEU score is better

Model	IWSLT14 small	WMT14 base
SGD	28.57 ± .15	26.04
AdamW [24]	35.66 ± .11	28.19
ADAHESSIAN	**35.79 ± .06**	**28.52**

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

Results on Language Modeling

Only learning rate and space averaging block size are tuned for ADAHESSIAN

Lower perplexity is better

Model	PTB Three-Layer	Wikitext-103 Six-Layer
SGD	59.9 ± 3.0	78.5
AdamW [24]	54.2 ± 1.6	20.9
ADAHESSIAN	**51.5 ± 1.2**	**19.9**

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

Results for SqueezeBERT on GLUE

The finetuning result for SqueezeBERT on GLUE benchmark

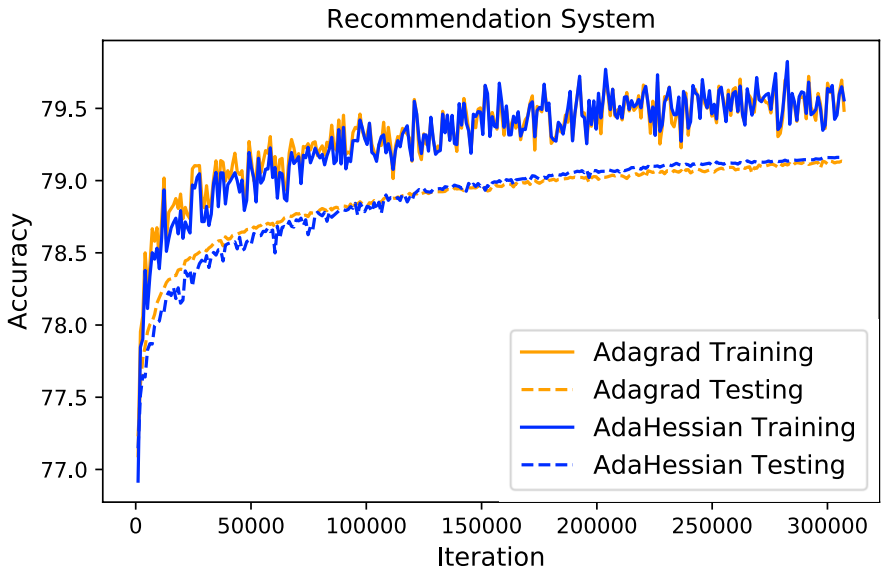
Higher accuracy is better

	RTE	MPRC	STS-B	SST-2	QNLI	QQP	MNLI-m	MNLI-mm	Avg.
AdamW$^+$ [20]	71.8	89.8	89.4	**92.0**	**90.5**	89.4	82.9	82.3	86.01
AdamW*	79.06	90.69	90.00	91.28	90.30	**89.49**	82.61	81.84	86.91
ADAHESSIAN	**80.14**	**91.94**	**90.59**	91.17	89.97	89.33	**82.78**	**82.62**	**87.32**

Iandola FN, Shaw AE, Krishna R, Keutzer KW. SqueezeBERT: What can computer vision teach NLP about efficient neural networks?. arXiv preprint arXiv:2006.11316, 2020.
Z Yao, A Gholami, S Shen, M Mustafa, K Keutzer, MW Mahoney, ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning, arXiv: 2006.00719, 2020.

Results on Recommendation Systems

Only learning rate and space averaging block size are tuned for ADAHESSIAN



Criteo Ad Kaggle Dataset	Test Accuracy
AdaGrad	79.135
ADAHESSIAN	**79.167**

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

Speed Comparison with SGD

- An important advantage is the not only AdaHessian achieves SOTA results but its per iteration cost is comparable to SGD
- Computing Hessian diagonal at every step results in only **2x (theoretically) and 3.2x (empirically)** overhead compared to SGD
  - This computation can be delayed to reduce this overhead down to **1.2x**

Hessian Comp. Freq.	1	2	3	4	5
Theoretical Cost (×SGD)	2×	1.5×	1.33×	1.25×	1.2×
ResNet20 (Cifar10)	92.13 ± .08	92.40 ± .04	92.06 ± .18	92.17 ± .21	92.16 ± .12
Measured Cost (×SGD)	2.42×	1.71×	1.47×	1.36×	1.28×
Measured Cost (×Adam)	2.27×	1.64×	1.42×	1.32×	1.25×

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

Robustness to Hyperparameter Tuning

Robustness to Learning Rate:

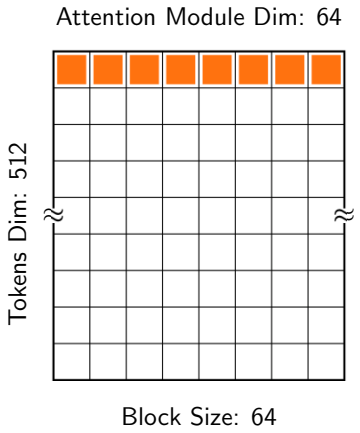
- AdaHessian still achieves acceptable performance even when scaling learning rate by10x, while ADAM diverges after just 6x scaling.

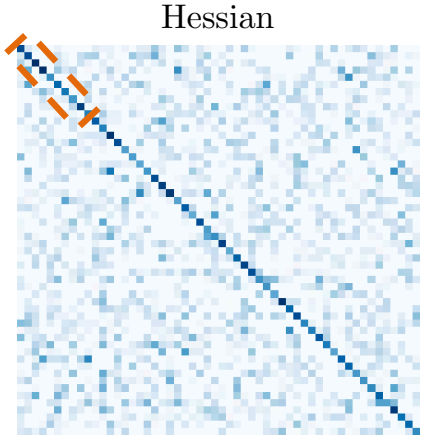
LR Scaling	0.5	1	2	3	4	5	6	10
AdamW	**35.42 ± .09**	35.66 ± .11	**35.37 ± .07**	**35.18 ± .07**	**34.79 ± .15**	14.41 ± 13.25	0.41 ± .32	Diverge
ADAHESSIAN	35.33 ± .10	**35.79 ± .06**	35.21 ± .14	34.74 ± .10	34.19 ± .06	**33.78 ± .14**	**32.70 ± .10**	**32.48 ± .83**

Result on IWSLT14.

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

Robustness to Spatial Averaging (Block Size)





Block Size	1	2	4	8	16	32	64	128
AdaHessian	35.67 ± .10	35.66 ± .07	35.78 ± .07	35.77 ± .08	35.67 ± .08	**35.79 ± .06**	35.72 ± .06	35.67 ± .11

Result on IWSLT14. The BLEU score of AdamW is 35.66
Choice of block size does not drastically change the performance.

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

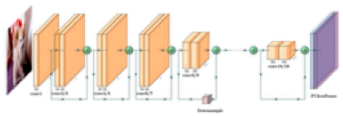
Some related Work: pyHessian

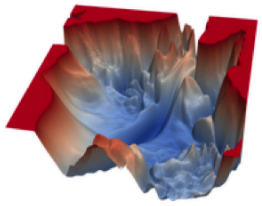
PyHESSIAN

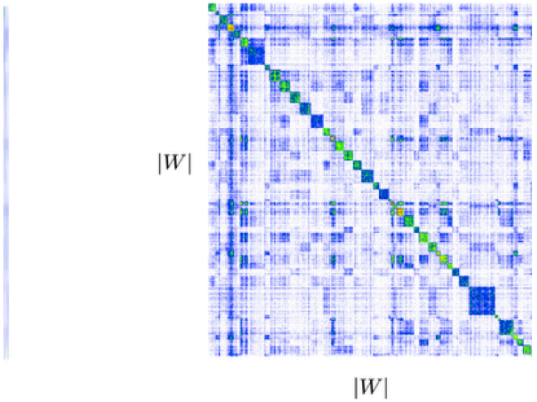
$$\min_w E(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$

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

Conclusions

- 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
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
Code: https://github.com/amirgholami/AdaHessian

Outline

Overcoming Inversion Bias in Distributed Newton's Method

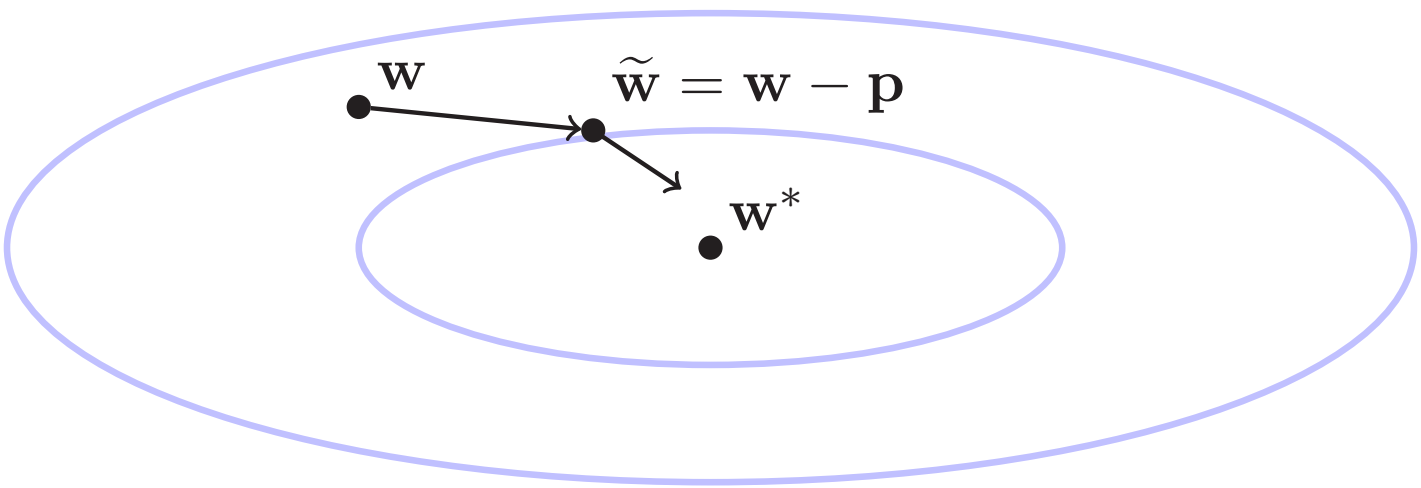
Michael W. Mahoney

ICSI and Department of Statistics
University of California, Berkeley

Joint work with Burak Bartan, Mert Pilanci, and **Michał Dereziński**

Convex optimization

$$\text{Find} \quad \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{w})$$



Why use second-order methods...

...when there is SGD?

- Sensitive to hyper-parameters
- Limited effectiveness for large batch training

See, e.g., [DMK+18, GVY+18]

Second-order:

- No hyper-parameter tuning
- Supports large batch training

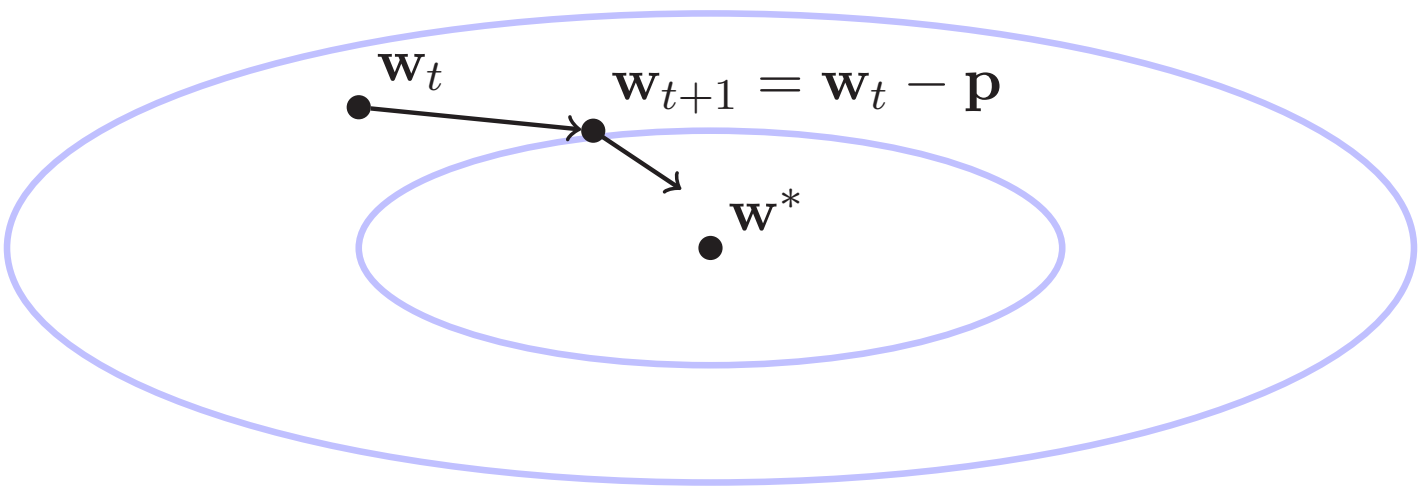
Recent interest in second-order methods:

- theoretical analysis [RKM19, RLXM18, WRKXM18]
- empirical (including DNNs) [GKC+19, FKR+18, KRMG18]

Newton's method

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n} \ell_j(\mathbf{w}^\top \mathbf{x}_j) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$\mathbf{p} = \big[\ \underbrace{\nabla^2 \mathcal{L}(\mathbf{w})}_{\text{Hessian } \mathbf{H}}\ \big]^{-1} \underbrace{\nabla \mathcal{L}(\mathbf{w})}_{\text{gradient } \mathbf{g}}$$

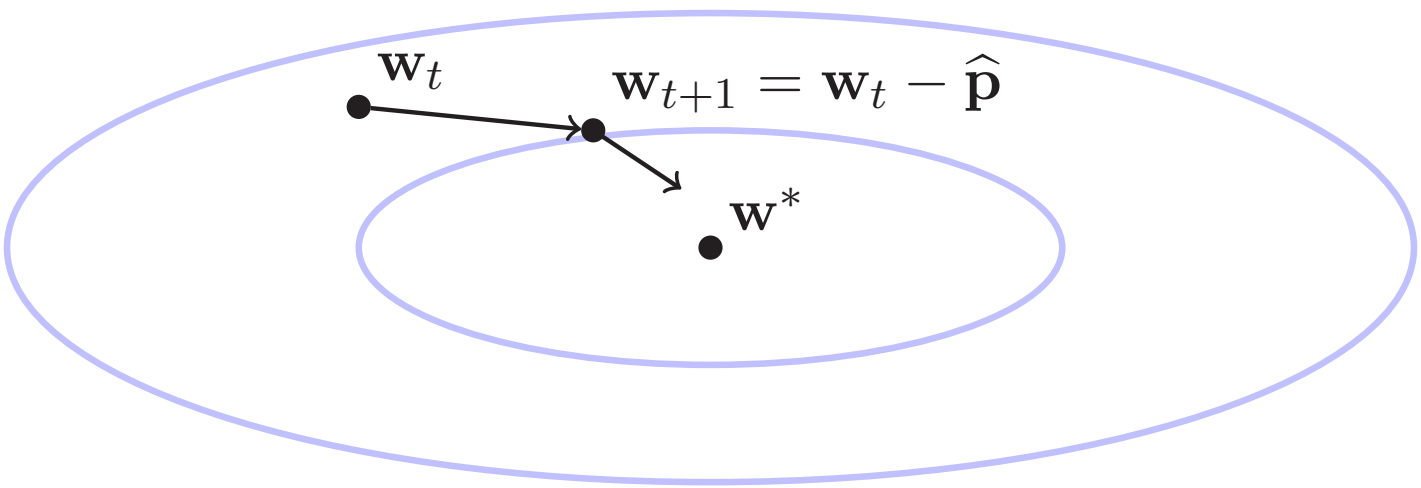


Newton's method

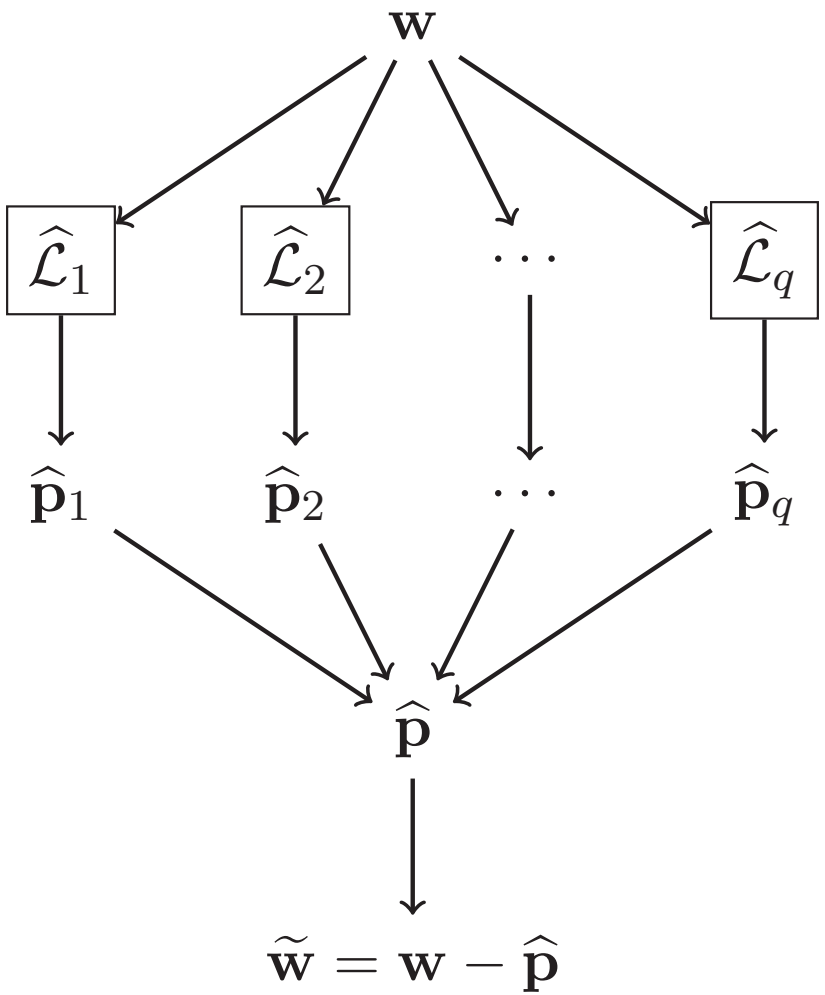
Approximate Newton's method

$$\widehat{\mathcal{L}}(\mathbf{w}) \;=\; \frac{1}{m}\sum_{j\in S}\ell_j(\mathbf{w}^\top \mathbf{x}_j) \;+\frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$\widehat{\mathbf{p}} = \big[\quad \underbrace{\nabla^2\widehat{\mathcal{L}}(\mathbf{w})}_{\text{Hessian estimate } \widehat{\mathbf{H}}} \quad\big]^{-1} \underbrace{\nabla\mathcal{L}(\mathbf{w})}_{\text{gradient } \mathbf{g}}$$



Distributed Newton's method



Question: How to combine local Newton estimates $\widehat{\mathbf{p}}_1, ..., \widehat{\mathbf{p}}_q$?

Model averaging

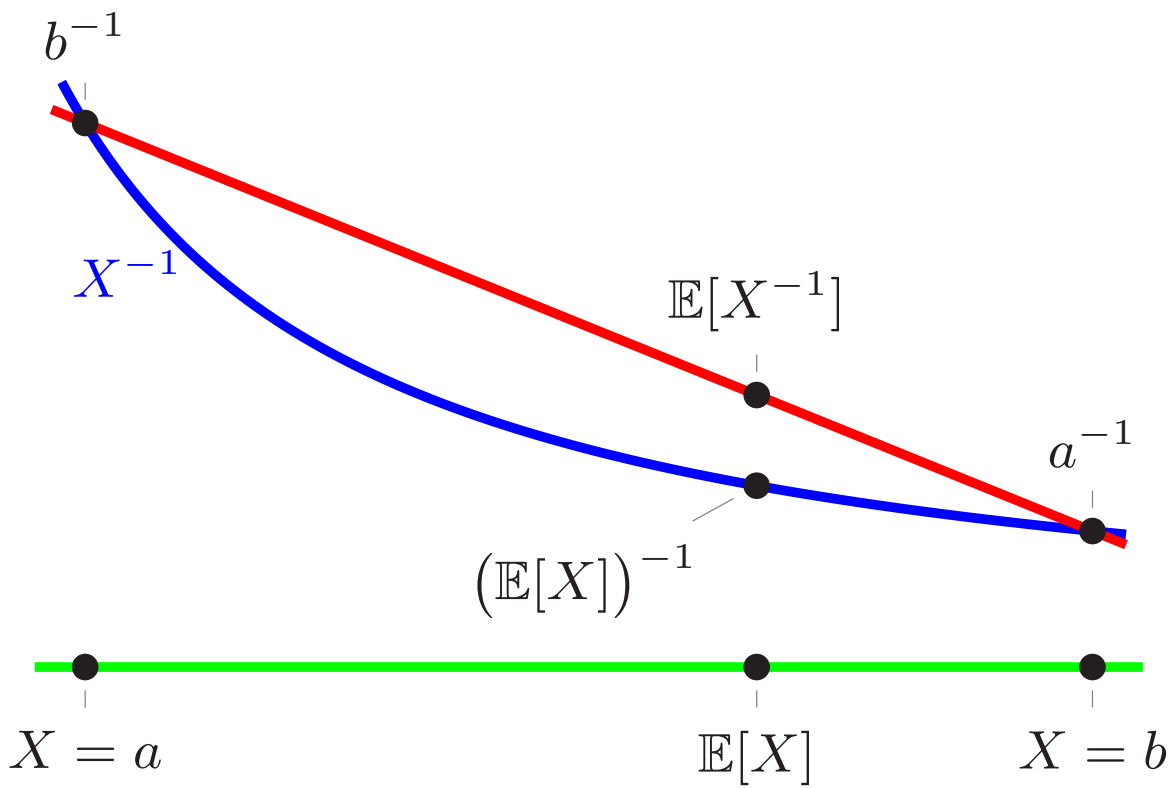
Standard averaging leads to biased estimates:

$$\lim_{q\to\infty} \frac{1}{q}\sum_{t=1}^{q} \widehat{\mathbf{p}}_t \;\neq\; \mathbf{p} \qquad (q \text{ is the number of machines})$$

$$\mathbb{E}\big[\widehat{\mathbf{H}}^{-1}\big] \neq \mathbf{H}^{-1}, \qquad \text{even though} \quad \mathbb{E}\big[\widehat{\mathbf{H}}\big] = \mathbf{H}.$$

General phenomenon: Inversion bias

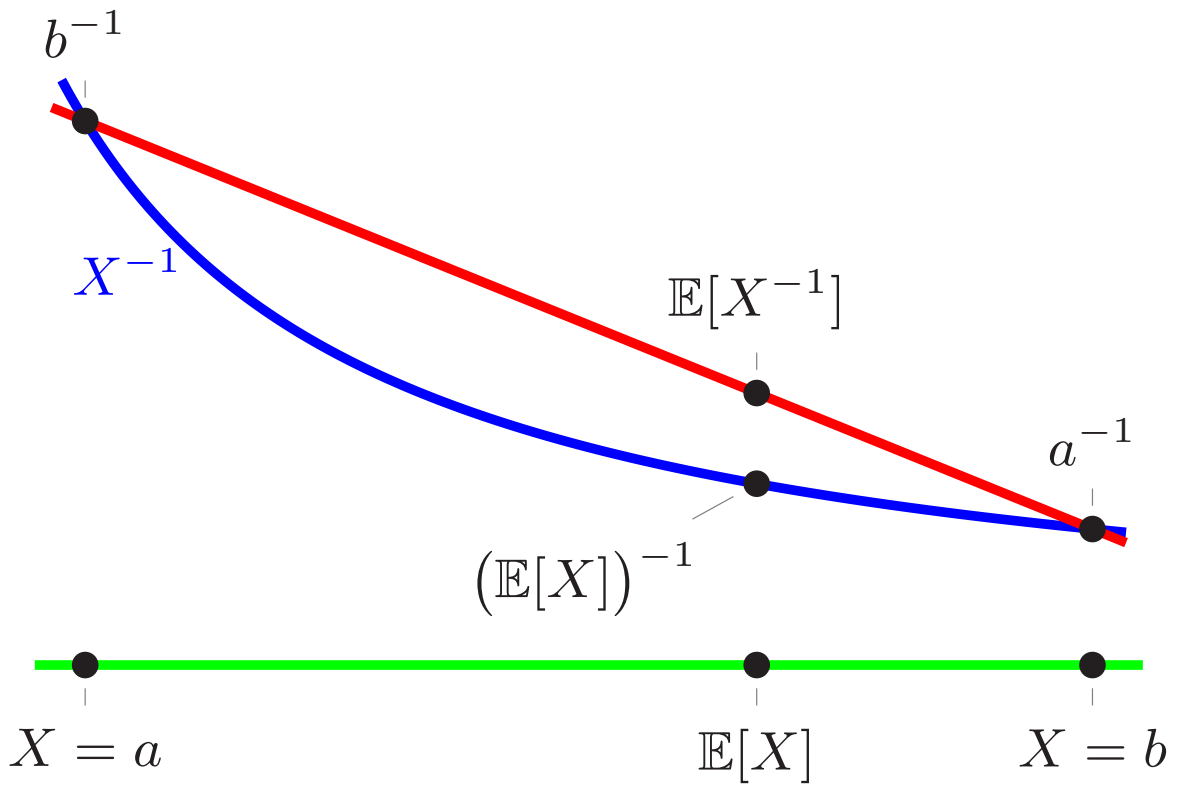
Inversion bias: $\mathbb{E}[X^{-1}] \neq \left(\mathbb{E}[X]\right)^{-1}$ for random X



General phenomenon: Inversion bias

Inversion bias: $\mathbb{E}[X^{-1}] \neq \left(\mathbb{E}[X]\right)^{-1}$ for random X

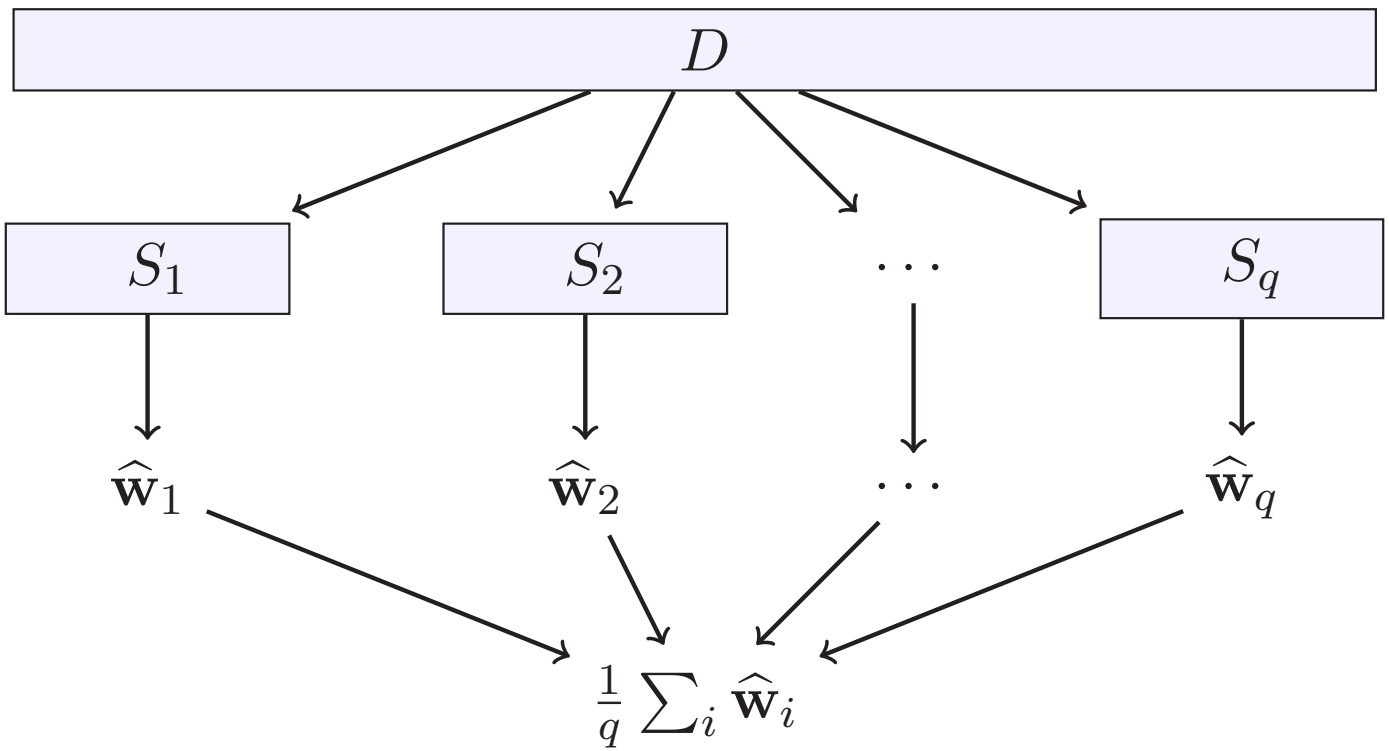
Extends to inverting high-dimensional random matrices



Inversion bias in model averaging

1. Bagging
2. Distributed optimization
3. Federated learning

$$\mathbf{w}^* - \frac{1}{q}\sum_i \widehat{\mathbf{w}}_i \overset{q\to\infty}{\longrightarrow} \underbrace{\mathbf{w}^* - \mathbb{E}[\widehat{\mathbf{w}}_i]}_{\text{Bias}}$$



Determinantal correction

Hessian estimate: $\widehat{\mathbf{H}} = \nabla^2 \widehat{\mathcal{L}}(\mathbf{w})$

Inversion bias: $\mathbb{E}\left[\widehat{\mathbf{H}}^{-1}\right] \neq \mathbf{H}^{-1}$

Determinantal correction

Hessian estimate: $\widehat{\mathbf{H}} = \nabla^2 \widehat{\mathcal{L}}(\mathbf{w})$

Inversion bias: $\mathbb{E}\big[\widehat{\mathbf{H}}^{-1}\big] \neq \mathbf{H}^{-1}$

Correction: $$\frac{\mathbb{E}\big[\det(\widehat{\mathbf{H}})\widehat{\mathbf{H}}^{-1}\big]}{\mathbb{E}\big[\det(\widehat{\mathbf{H}})\big]} = \mathbf{H}^{-1}$$

Determinantal correction

$$\text{Hessian estimate:} \qquad \widehat{\mathbf{H}} = \nabla^2 \widehat{\mathcal{L}}(\mathbf{w})$$

$$\text{Inversion bias:} \qquad \mathbb{E}\big[\widehat{\mathbf{H}}^{-1}\big] \neq \mathbf{H}^{-1}$$

$$\text{Correction:} \qquad \frac{\mathbb{E}\big[\det(\widehat{\mathbf{H}})\widehat{\mathbf{H}}^{-1}\big]}{\mathbb{E}\big[\det(\widehat{\mathbf{H}})\big]} = \mathbf{H}^{-1}$$

Two strategies of using the correction:

1. Weighted averaging instead of uniform averaging
   Determinantal averaging [**D**M19]
2. Joint sampling instead of uniform sampling
   Surrogate sketches [**D**BPM20]

Comparison of two strategies

Determinantal averaging

- consistent global estimate: $\widehat{\mathbf{p}} \underset{m\to\infty}{\longrightarrow} \mathbf{p}$
- works with uniform sampling

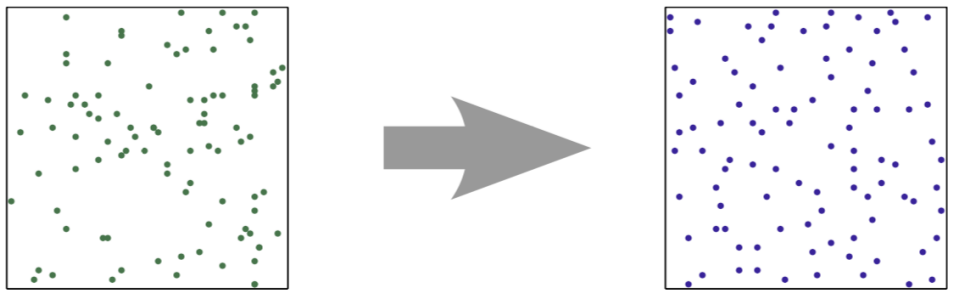
Surrogate sketching

- unbiased local estimates: $\mathbb{E}[\widehat{\mathbf{p}}_t] = \mathbf{p}$
- samples from a Determinantal Point Process (DPP)

Determinantal Point Processes (DPPs)

Non-i.i.d. randomized selection of a data subset S

Negative correlation: $\Pr(i \in S \mid j \in S) < \Pr(i \in S)$



i.i.d. (left) versus DPP (right)

- Fast algorithms: [C**D**V20] (NeurIPS'20)

  "Sampling from a k-DPP without looking at all items"

- Learn more: [**D**M20] (Notices of the AMS)

  "Determinantal point processes in randomized numerical linear algebra"

Image from [KT12]

Correcting inversion bias in Distributed Newton

Baseline: Uniform averaging of biased estimates [WRKXM18]

Convergence rate: $\|\mathbf{w}_{t+1} - \mathbf{w}^*\| = \widetilde{O}\big(\underbrace{\sqrt{\frac{d}{qm}}}_{\text{“variance”}} + \underbrace{\frac{d}{m}}_{\text{“bias”}} \big) \cdot \|\mathbf{w}_t - \mathbf{w}^*\|$

q - number of machines
m - data points per machine
d - number of parameters

	Method	Convergence rate	Trade-offs
	Baseline	$\sqrt{\frac{d}{qm}} + \frac{d}{m}$	Var | Bias | Cost

[**D**M19] *“Distributed estimation of the inverse Hessian by determinantal averaging”*, at NeurIPS’19.

Correcting inversion bias in Distributed Newton

Baseline: Uniform averaging of biased estimates [WRKXM18]

Convergence rate: $\|\mathbf{w}_{t+1} - \mathbf{w}^*\| = \widetilde{O}\big(\underbrace{\sqrt{\frac{d}{qm}}}_{\text{“variance”}} + \underbrace{\frac{d}{m}}_{\text{“bias”}} \big) \cdot \|\mathbf{w}_t - \mathbf{w}^*\|$

q - number of machines
m - data points per machine
d - number of parameters

	Method	Convergence rate	Trade-offs
	Baseline	$\sqrt{\frac{d}{qm}} + \frac{d}{m}$	Var | Bias | Cost
[DM19]	*Determinantal averaging*	$\frac{d}{\sqrt{qm}}$	Var | Bias | Cost

[DM19] *“Distributed estimation of the inverse Hessian by determinantal averaging”*, at NeurIPS’19.

Correcting inversion bias in Distributed Newton

Baseline: Uniform averaging of biased estimates [WRKXM18]

Convergence rate: $\|\mathbf{w}_{t+1} - \mathbf{w}^*\| = \widetilde{O}\big(\underbrace{\sqrt{\frac{d}{qm}}}_{\text{“variance”}} + \underbrace{\frac{d}{m}}_{\text{“bias”}} \big) \cdot \|\mathbf{w}_t - \mathbf{w}^*\|$

q - number of machines
m - data points per machine
d - number of parameters

	Method	Convergence rate	Trade-offs
	Baseline	$\sqrt{\frac{d}{qm}} + \frac{d}{m}$	Var | Bias | Cost
[DM19]	*Determinantal averaging*	$\frac{d}{\sqrt{qm}}$	Var | Bias | Cost
[DBPM20]	*Surrogate sketching*	$\sqrt{\frac{d}{qm}}$	Var | Bias | Cost

[DBPM20] *“Debiasing distributed second order optimization with surrogate sketching and scaled regularization”*, at NeurIPS’20.

Correcting inversion bias in Distributed Newton

Baseline: Uniform averaging of biased estimates [WRKXM18]

Convergence rate: $\|\mathbf{w}_{t+1} - \mathbf{w}^*\| = \widetilde{O}\Big(\underbrace{\sqrt{\frac{d}{qm}}}_{\text{“variance”}} + \underbrace{\frac{d}{m}}_{\text{“bias”}} \Big) \cdot \|\mathbf{w}_t - \mathbf{w}^*\|$

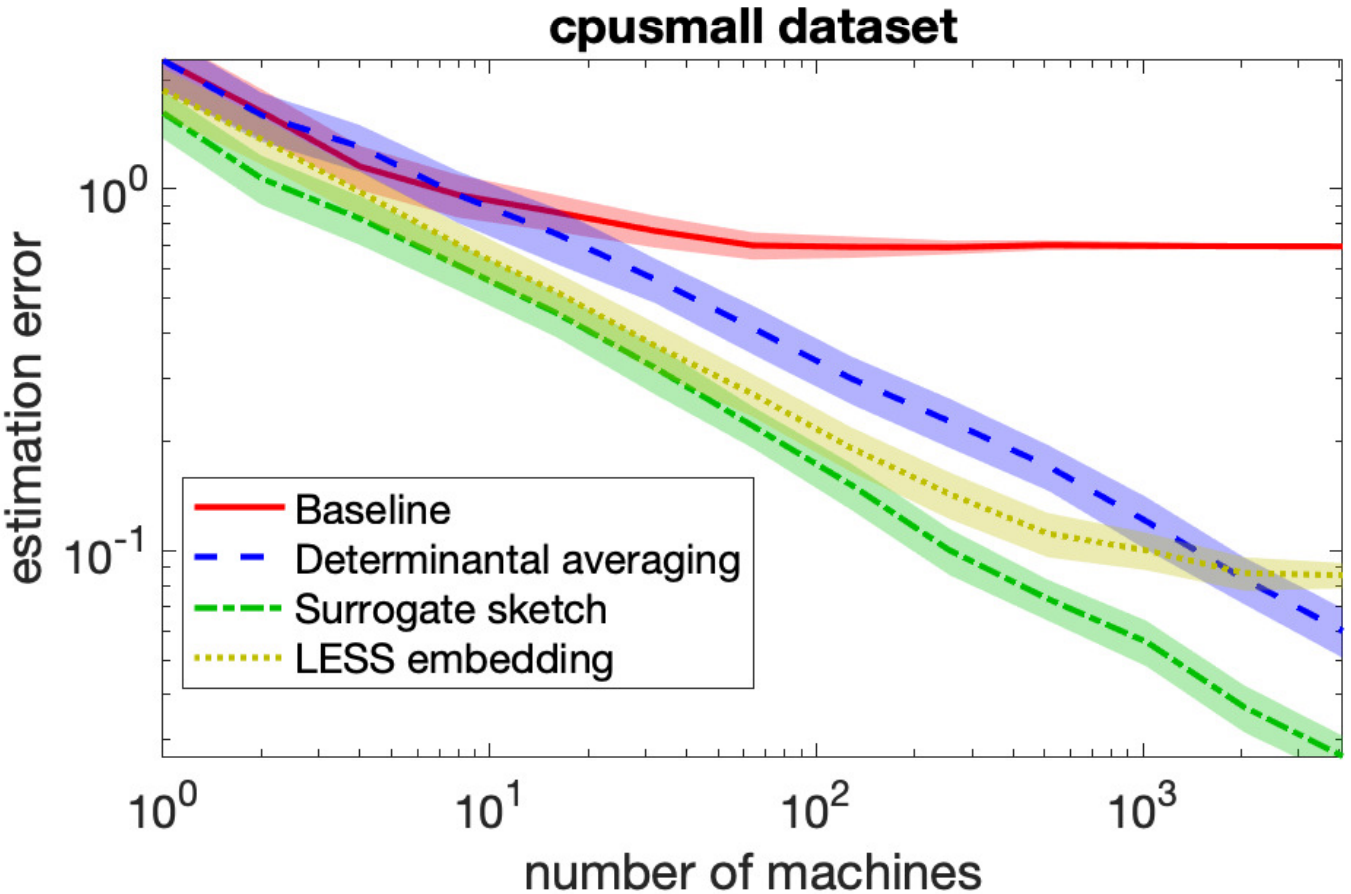
q - number of machines
m - data points per machine
d - number of parameters

	Method	Convergence rate	Trade-offs
	Baseline	$\sqrt{\frac{d}{qm}} + \frac{d}{m}$	Var | Bias | Cost
[**D**M19]	*Determinantal averaging*	$\frac{d}{\sqrt{qm}}$	Var | Bias | Cost
[**D**BPM20]	*Surrogate sketching*	$\sqrt{\frac{d}{qm}}$	Var | Bias | Cost
[**D**LDM20]	*LESS embeddings*	$\sqrt{\frac{d}{qm}} + \frac{\sqrt{d}}{m}$	Var | Bias | Cost

[**D**LDM20] *“Sparse sketches with small inversion bias”*, Preprint at arXiv:2011.10695.

Bias-variance trade-offs in model averaging

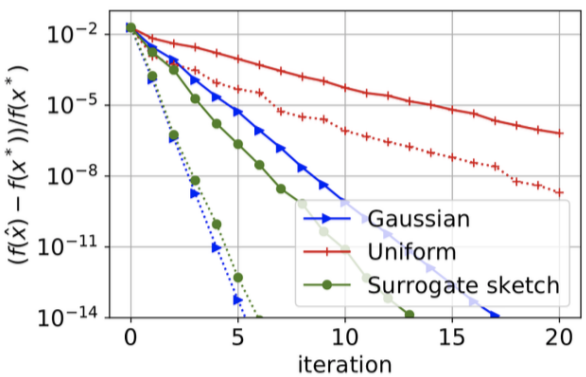
$$\text{estimation error} \quad = \quad \left\| \frac{1}{q} \sum_{i=1}^{q} \widehat{\mathbf{p}}_i - \mathbf{p}^* \right\|$$



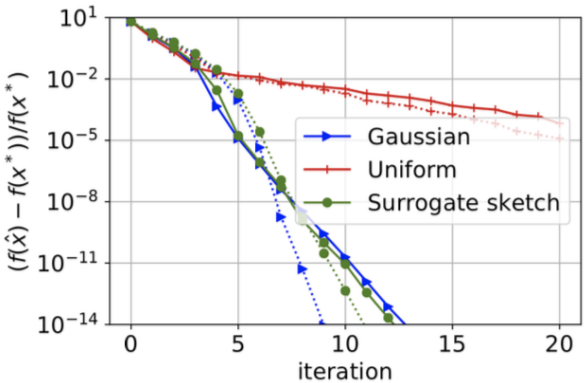
Experiments: Effect of implicit regularization

Question: Should local regularizer match the global λ?

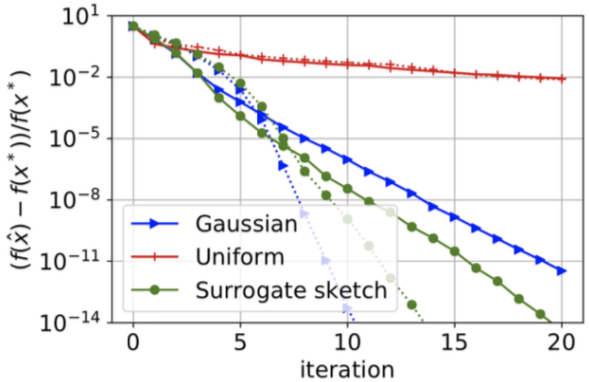
Distributed Newton with 100 machines for logistic regression



(a) statlog-australian-credit



(b) breast-cancer-wisc



(c) ionosphere

dashed lines: *local regularizer* $= \lambda \cdot \left(1 - \frac{d_\lambda}{m}\right)$

$$\text{regularized loss:} \quad \mathcal{L}(\mathbf{w}) \;=\; \frac{1}{n}\sum_{j=1}^{n} \ell_j(\mathbf{w}^\top \mathbf{x}_j) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

[DBPM20] *"Debiasing distributed second order optimization with surrogate sketching and scaled regularization"*, at NeurIPS'20.

Conclusions

- Distributed Newton's method suffers from inversion bias
- We can correct this bias with:
  - Weighted averaging instead of uniform averaging
    Determinantal averaging
  - Joint sampling instead of uniform sampling
    Surrogate sketches
  - Scaled local regularization in place of the global regularizer
    $\lambda' = \lambda \cdot (1 - \frac{d_\lambda}{m})$

References I

Daniele Calandriello, Michał Dereziński, and Michal Valko.
Sampling from a k-dpp without looking at all items.
In Conference on Neural Information Processing Systems, 2020.

Michał Dereziński, Burak Bartan, Mert Pilanci, and Michael W Mahoney.
Debiasing distributed second order optimization with surrogate sketching and scaled regularization.
In Conference on Neural Information Processing Systems, 2020.

Michał Dereziński, Zhenyu Liao, Edgar Dobriban, and Michael W Mahoney.
Sparse sketches with small inversion bias.
arXiv preprint arXiv:2011.10695, 2020.

Michał Dereziński and Michael W Mahoney.
Distributed estimation of the inverse hessian by determinantal averaging.
In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 11401–11411. Curran Associates, Inc., 2019.

Michał Dereziński and Michael W Mahoney.
Determinantal point processes in randomized numerical linear algebra.
Notices of the American Mathematical Society, 68(1):34–45, 2021.

References II

Michał Dereziński, Dhruv Mahajan, S. Sathiya Keerthi, S. V. N. Vishwanathan, and Markus Weimer.
Batch-expansion training: An efficient optimization framework.
In Amos Storkey and Fernando Perez-Cruz, editors, Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, volume 84 of Proceedings of Machine Learning Research, pages 736–744, Playa Blanca, Lanzarote, Canary Islands, 09–11 Apr 2018.

Chih-Hao Fang, Sudhir B Kylasa, Fred Roosta, Michael W. Mahoney, and Ananth Grama.
Newton-ADMM: A Distributed GPU-Accelerated Optimizer for Multiclass Classification Problems.
arXiv e-prints, page arXiv:1807.07132, Jul 2018.

Vipul Gupta, Swanand Kadhe, Thomas Courtade, Michael W. Mahoney, and Kannan Ramchandran.
OverSketched Newton: Fast Convex Optimization for Serverless Systems.
arXiv e-prints, page arXiv:1903.08857, Mar 2019.

Noah Golmant, Nikita Vemuri, Zhewei Yao, Vladimir Feinberg, Amir Gholami, Kai Rothauge, Michael W. Mahoney, and Joseph Gonzalez.
On the Computational Inefficiency of Large Batch Sizes for Stochastic Gradient Descent.
arXiv e-prints, page arXiv:1811.12941, Nov 2018.

References III

Sudhir B. Kylasa, Farbod Roosta-Khorasani, Michael W. Mahoney, and Ananth Grama.
GPU Accelerated Sub-Sampled Newton's Method.
arXiv e-prints, page arXiv:1802.09113, Feb 2018.

Alex Kulesza and Ben Taskar.
Determinantal Point Processes for Machine Learning.
Now Publishers Inc., Hanover, MA, USA, 2012.

Farbod Roosta-Khorasani and Michael W. Mahoney.
Sub-sampled newton methods.
Math. Program., 174(1–2):293–326, March 2019.

Fred Roosta, Yang Liu, Peng Xu, and Michael W. Mahoney.
Newton-MR: Newton's Method Without Smoothness or Convexity.
arXiv e-prints, page arXiv:1810.00303, Sep 2018.

Shusen Wang, Farbod Roosta-Khorasani, Peng Xu, and Michael W Mahoney.
GIANT: Globally improved approximate newton method for distributed optimization.
In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems 31, pages 2332–2342. Curran Associates, Inc., 2018.

Outline

"Physics-informed" ML: Be careful!

Computer Science > Machine Learning

[Submitted on 2 Sep 2021]

Characterizing possible failure modes in physics-informed neural networks

Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, Michael W. Mahoney

Recent work in scientific machine learning has developed so-called physics-informed neural network (PINN) models. The typical approach is to incorporate physical domain knowledge as soft constraints on an empirical loss function and use existing machine learning methodologies to train the model. We demonstrate that, while existing PINN methodologies can learn good models for relatively trivial problems, they can easily fail to learn relevant physical phenomena even for simple PDEs. In particular, we analyze several distinct situations of widespread physical interest, including learning differential equations with convection, reaction, and diffusion operators. We provide evidence that the soft regularization in PINNs, which involves differential operators, can introduce a number of subtle problems, including making the problem ill-conditioned. Importantly, we show that these possible failure modes are not due to the lack of expressivity in the NN architecture, but that the PINN's setup makes the loss landscape very hard to optimize. We then describe two promising solutions to address these failure modes. The first approach is to use curriculum regularization, where the PINN's loss term starts from a simple PDE regularization, and becomes progressively more complex as the NN gets trained. The second approach is to pose the problem as a sequence-to-sequence learning task, rather than learning to predict the entire space-time at once. Extensive testing shows that we can achieve up to 1-2 orders of magnitude lower error with these methods as compared to regular PINN training.

Comments:	22 pages
Subjects:	**Machine Learning (cs.LG)**; Artificial Intelligence (cs.AI); Numerical Analysis (math.NA); Computational Physics (physics.comp-ph)
Journal reference:	NeurIPS 2021
Cite as:	arXiv:2109.01050 **[cs.LG]**
	(or arXiv:2109.01050v1 **[cs.LG]** for this version)

Submission history

From: Aditi Krishnapriyan [view email]

[v1] Thu, 2 Sep 2021 16:06:45 UTC (2,501 KB)

First PINN paper in top ML venue — opportunities . . . and caveats.

Second derivatives: Be careful!

Statistics > Machine Learning

[Submitted on 2 Mar 2021 (v1), last revised 17 Mar 2021 (this version, v2)]

Hessian Eigenspectra of More Realistic Nonlinear Models

Zhenyu Liao, Michael W. Mahoney

Given an optimization problem, the Hessian matrix and its eigenspectrum can be used in many ways, ranging from designing more efficient second-order algorithms to performing model analysis and regression diagnostics. When nonlinear models and non-convex problems are considered, strong simplifying assumptions are often made to make Hessian spectral analysis more tractable. This leads to the question of how relevant the conclusions of such analyses are for more realistic nonlinear models. In this paper, we exploit deterministic equivalent techniques from random matrix theory to make a \emph{precise} characterization of the Hessian eigenspectra for a broad family of nonlinear models, including models that generalize the classical generalized linear models, without relying on strong simplifying assumptions used previously. We show that, depending on the data properties, the nonlinear response model, and the loss function, the Hessian can have \emph{qualitatively} different spectral behaviors: of bounded or unbounded support, with single- or multi-bulk, and with isolated eigenvalues on the left- or right-hand side of the bulk. By focusing on such a simple but nontrivial nonlinear model, our analysis takes a step forward to unveil the theoretical origin of many visually striking features observed in more complex machine learning models.

Comments: Identical to v1, except for the inclusion of some additional references
Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG); Spectral Theory (math.SP)
Cite as: arXiv:2103.01519 **[stat.ML]**
(or arXiv:2103.01519v2 **[stat.ML]** for this version)

Submission history

From: Zhenyu Liao [view email]
[v1] Tue, 2 Mar 2021 06:59:52 UTC (93 KB)
[v2] Wed, 17 Mar 2021 00:29:08 UTC (95 KB)

Hessians in ML $\neq$ Hessians in scientific computing.

RandNLA in BLAS/LAPACK . . . finally!

Prospectus for the Next LAPACK and ScaLAPACK Libraries:
Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC)*

James Demmel
University of California at Berkeley

Jack Dongarra
University of Tennessee, Oak Ridge National Laboratory, and University of Manchester

Julie Langou
University of Tennessee

Julien Langou
University of Colorado Denver

Piotr Luszczek
University of Tennessee

Michael W. Mahoney
ICSI and University of California at Berkeley

July 13, 2020

Including: Randomized BLAS, least-squares[1], low-rank approximation, etc., etc.
(led by **Riley Murray**)

[1] View over/under-determined LS problems i.t.o. structured "saddle point systems" . . . opening the door to lower-level LAPACK-type optimizations for broader class of stochastic second order optimization methods.

Outline

Conclusions

Stochastic second order optimization:

- Theory: Builds on basic RandNLA principles, but need finer control for algorithmic-statistical tradeooffs
- Implementations: ADAHessian (and pyHessian); and Randomized BLAS/LAPACK
- Applications: Both scientific computing and machine learning, but they are quite different