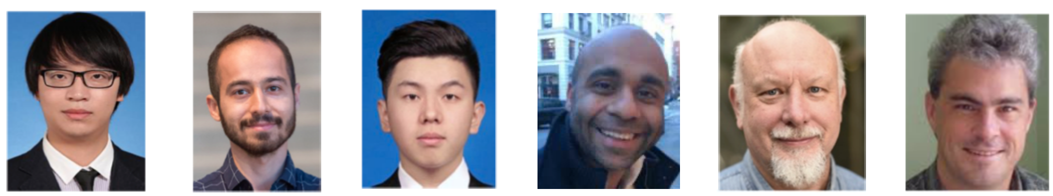
ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning

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

Michael W. Mahoney

September 2020



Berkeley UNIVERSITY OF CALIFORNIA

riselab UC Berkeley

One year ago: fall 2019

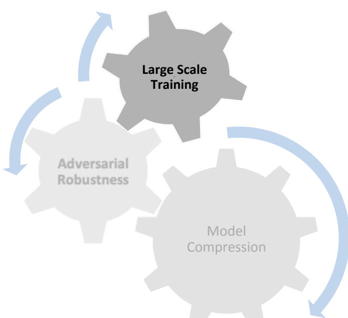
Making Deep Learning Revolution Practical Through Second Order Methods

Michael W. Mahoney

ICSI and Department of Statistics
University of California at Berkeley

Joint work with Amir Gholami, Zhewei Yao, and many others to be mentioned.

Second Order Methods



"Machine learning is high performance computing's first killer app for consumers" --- NVIDIA CEO 2015

Second-order methods (use Hessian info as well as gradient info) for:

- **Efficiency/inefficiency of training:** SGD, KFAC, and other 2nd order methods
- **Adversarial examples:** smoothing out ML objectives, using 2nd order methods
- **Quantizing large models:** using outlier metrics derived from 2nd order methods

Conclusions

"If I had asked people what they wanted, they would have said

- *faster SGD algorithms,*
- *better worst-case convergence rates,*
- *faster wall-clock times,*
- *better AutoML methods, ..."*

Second order methods

- sometimes do that,
- sometimes don't do that,
- more often lead to improvements---in timing/ robustness/reproducibility/understanding--- for more interesting and non-trivial reasons ...

Three years ago: fall 2017

Second order machine learning

Michael W. Mahoney

ICSI and Department of Statistics
UC Berkeley

Outline

- Machine Learning's "Inverse" Problem
- Your choice:
  - 1st Order Methods: FLAG n' FLARE, or
    - disentangle geometry from sequence of iterates
  - 2nd Order Methods: Stochastic Newton-Type Methods
    - "simple" methods for convex
    - "more subtle" methods for non-convex

Conclusions: Second order machine learning

- Second order methods
  - A simple way to go beyond first order methods
  - Obviously, don't be naïve about the details
- FLAG n' FLARE
  - Combine acceleration and adaptivity to get best of both worlds
- Can aggressively sub-sample gradient and/or Hessian
  - Improve running time at each step
  - Maintain strong second-order convergence
- Apply to non-convex problems
  - Trust region methods and cubic regularization methods
  - Converge to second order stationary point
  - Quite promising "preliminary results" in ML/DA applications

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!**

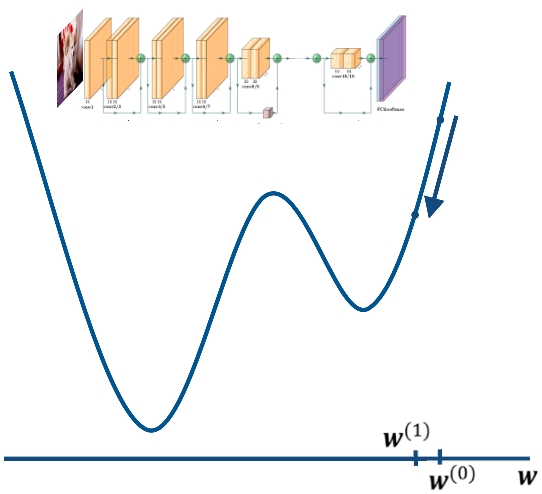
Task	CV	NLP	Recommendation System
Optimizer Choice	SGD	AdamW	Adagrad

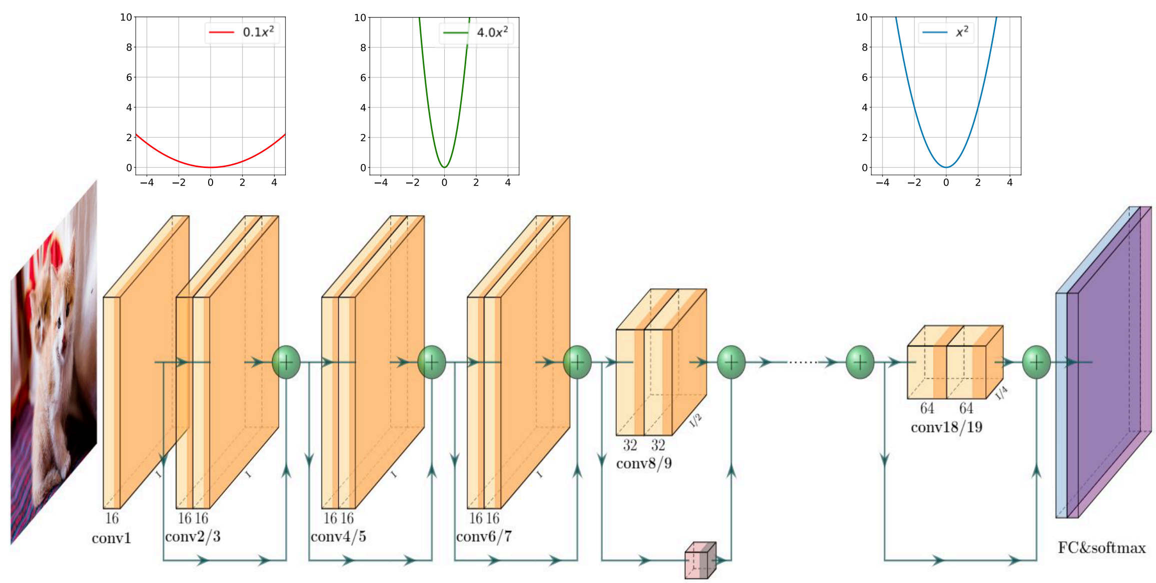
*BTW, not obvious if you just do popular things, e.g., ResNet50 training on ImageNet, since years of industrial scale (i.e., brute force) hyperparameter tuning and building systems for SGD-based methods mean those methods do well ...

SGD Based Training

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

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



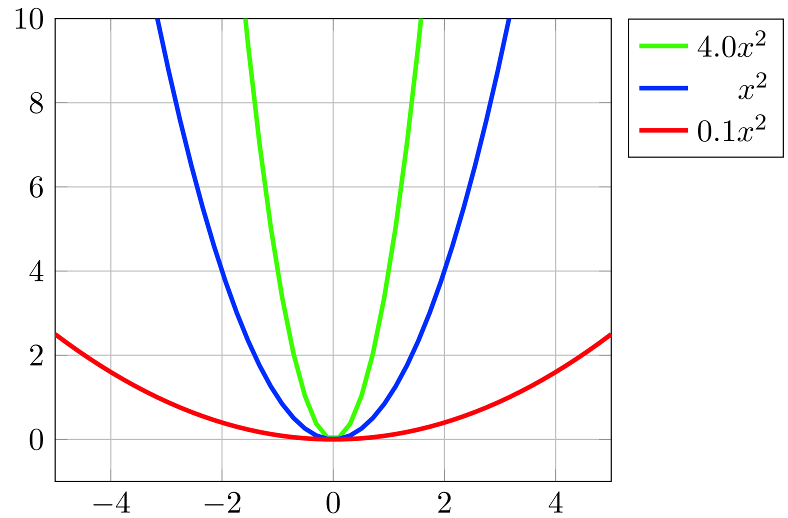


First and Second Order Methods

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

First Order Method | Second Order Method

$$\Delta w_t = g_t$$



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

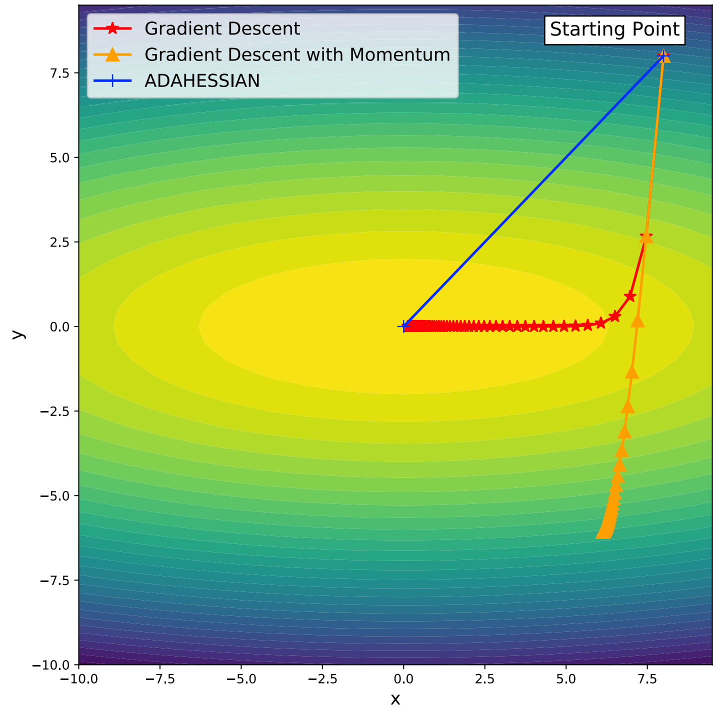
- At the origin, the first derivative of $y = 4x^2$, $y = x^2$, $y = 0.1x^2$ is all the same: 0
- The **second derivative** give more information: 8 , 2, and 0.2 respectively

First and Second Order Methods

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

First Order Method

$\Delta w_t = g_t$



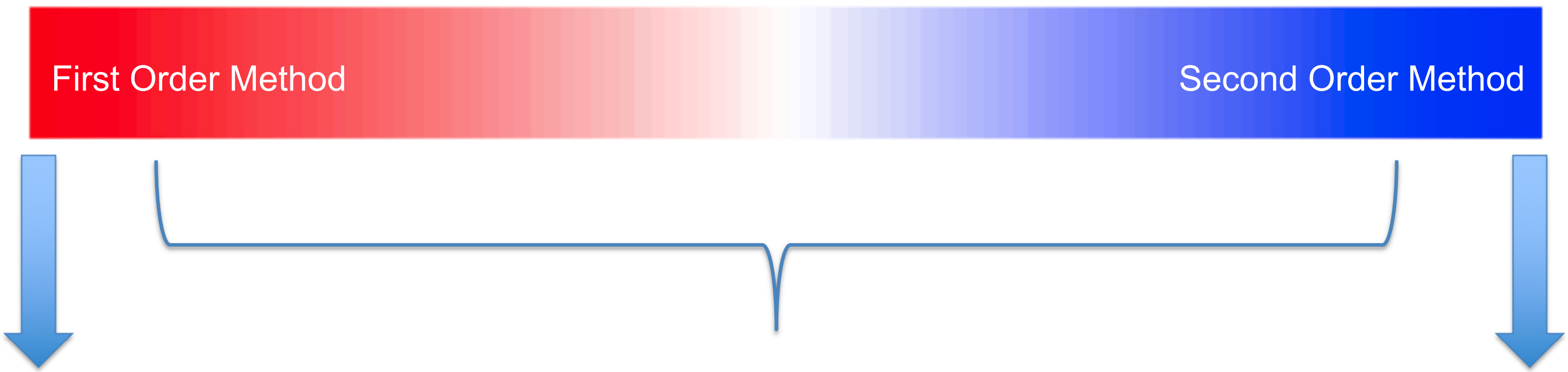
$$f = x^2 + 10y^2$$

Second Order Method

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

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: $\Delta\theta_t = H_t^{-k} g_t, \quad 0 \leq k \leq 1$

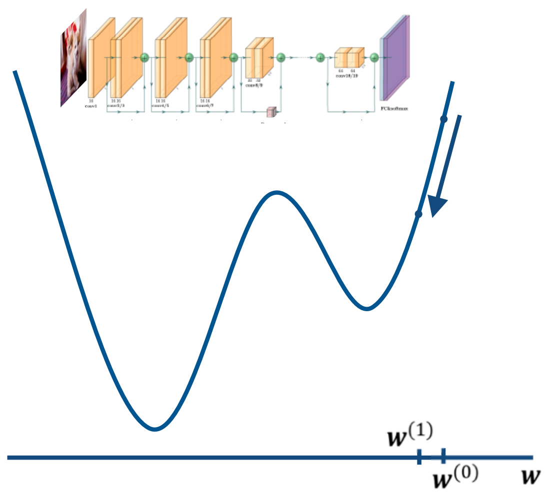
- For convex problem, since $g_t^T H_t^{-k} g_t \geq 0$, $H_t^{-k} g_t$ is a descent direction.
- For simple problems, computing H_t^{-k} is not a problem and it can be done by an eigen-decomposition.
- However, for large scale machine learning problems (e.g., DNNs), forming/storing Hessian are impractical.

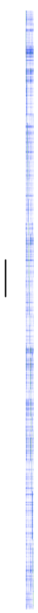
Second Derivative (Hessian)

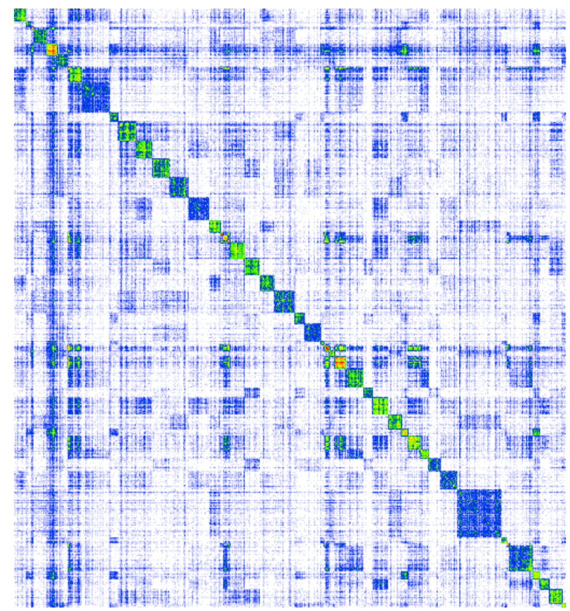
$$\min_w E(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i)$$

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

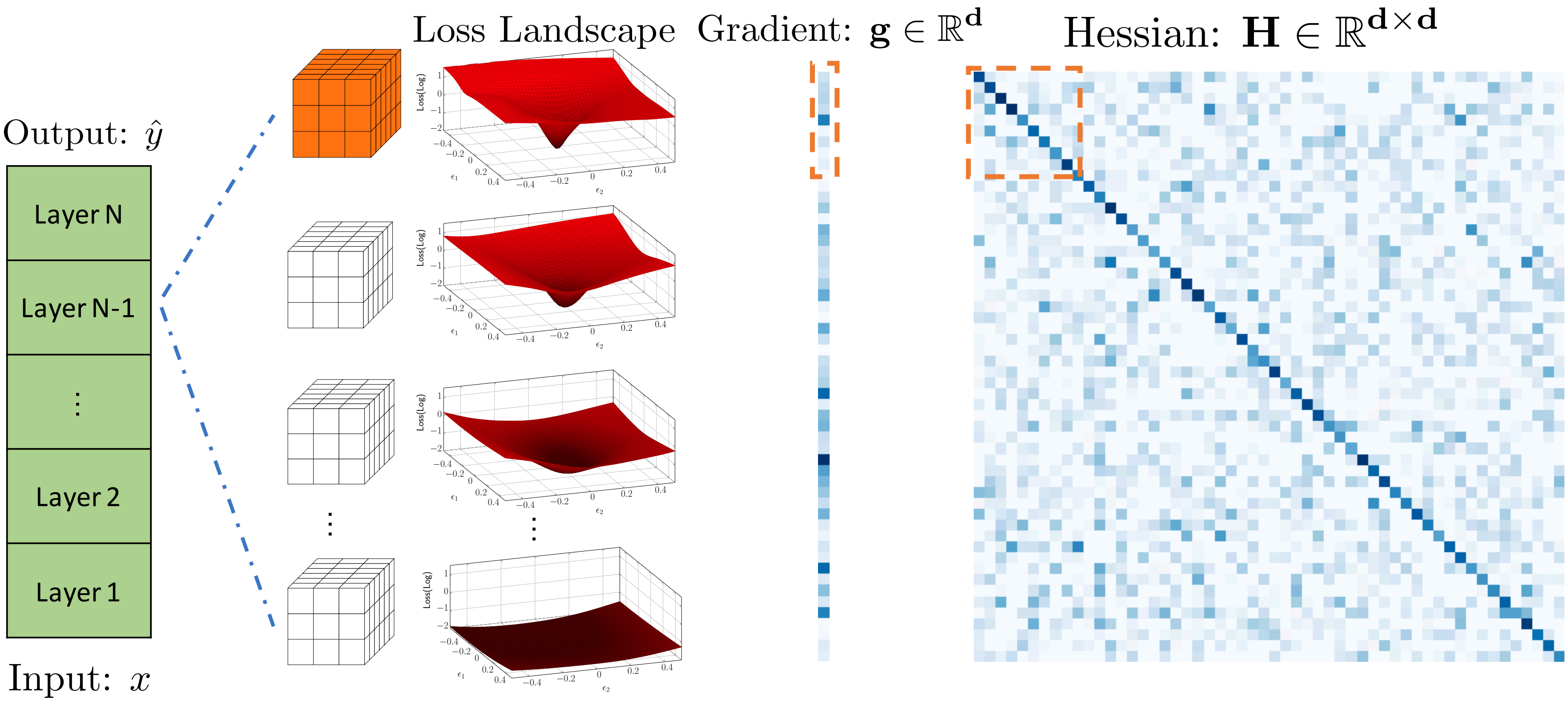
Hessian: $\frac{\partial^2 E}{\partial w^2} \in \mathcal{R}^{|W| \times |W|}$







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

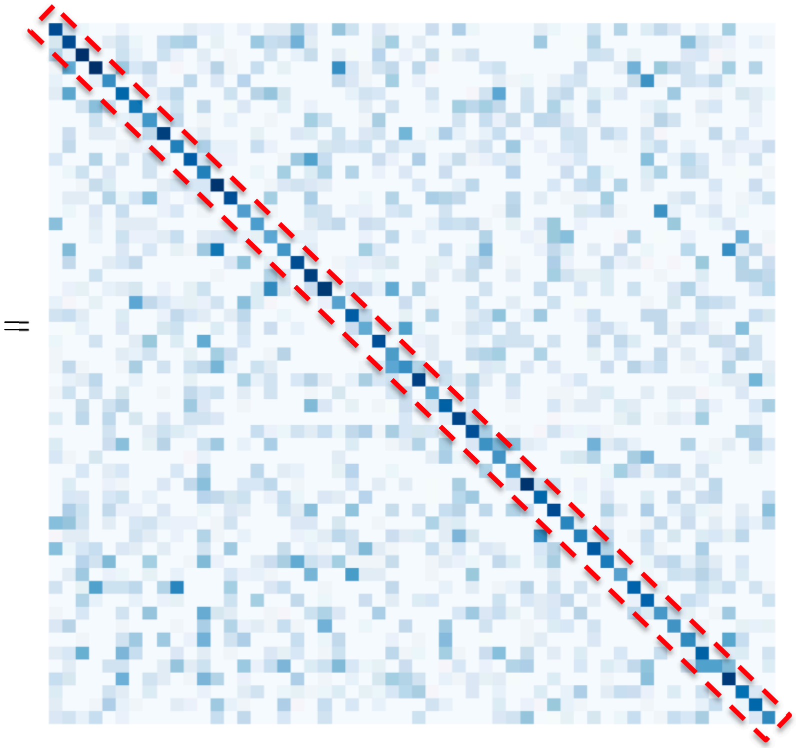
Forming the Hessian is infeasible:

For ResNet50 (with 24M parameters)

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

What if we approximate the Hessian?

Idea: Use Hessian diagonal

$g =$  $Diag(H) =$ 

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

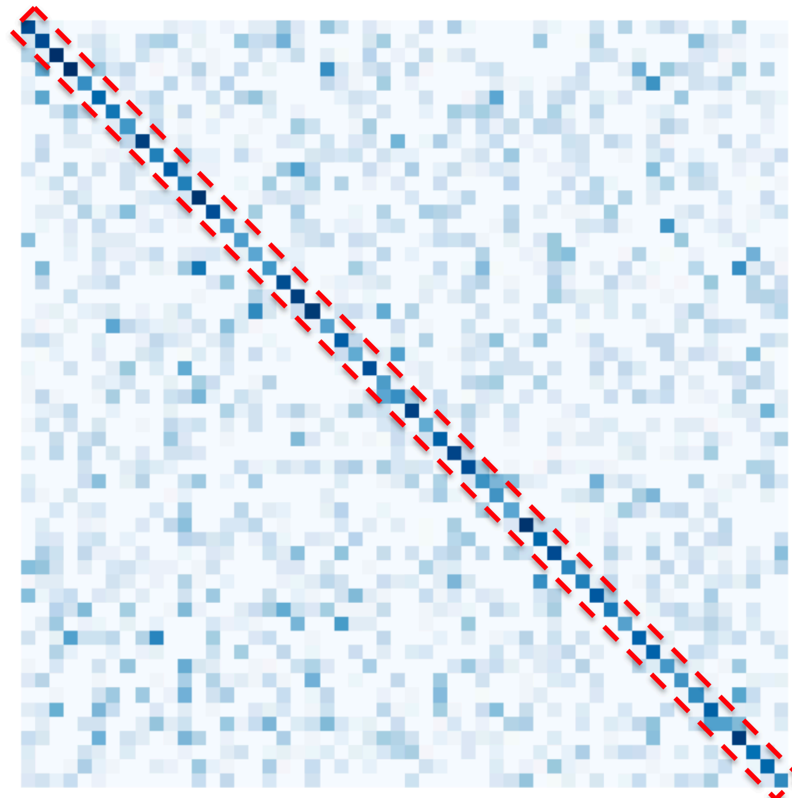
ADAHESSIAN algorithm is very simple and as follows:

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

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

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

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

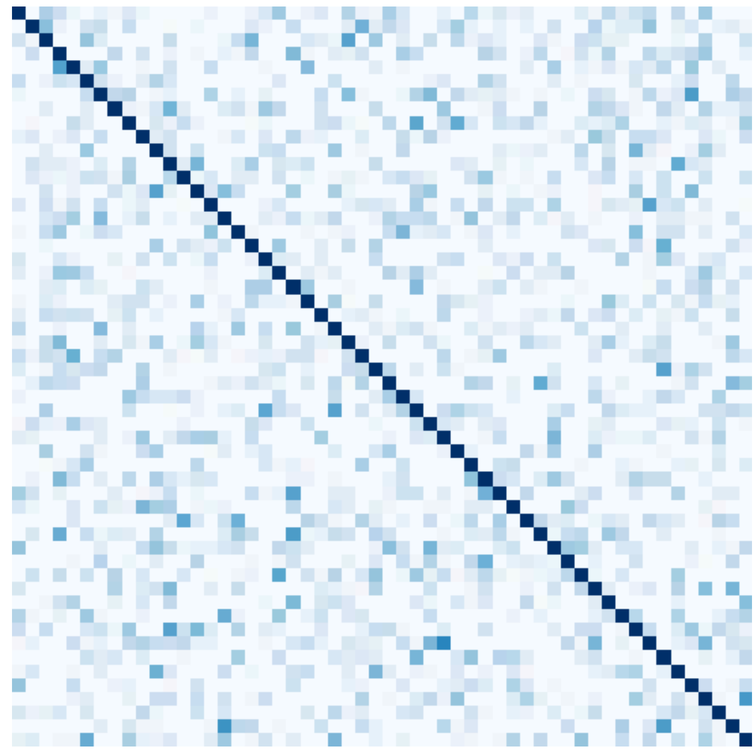
Is computing H^{-1} practical? Of course not ...

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 ...

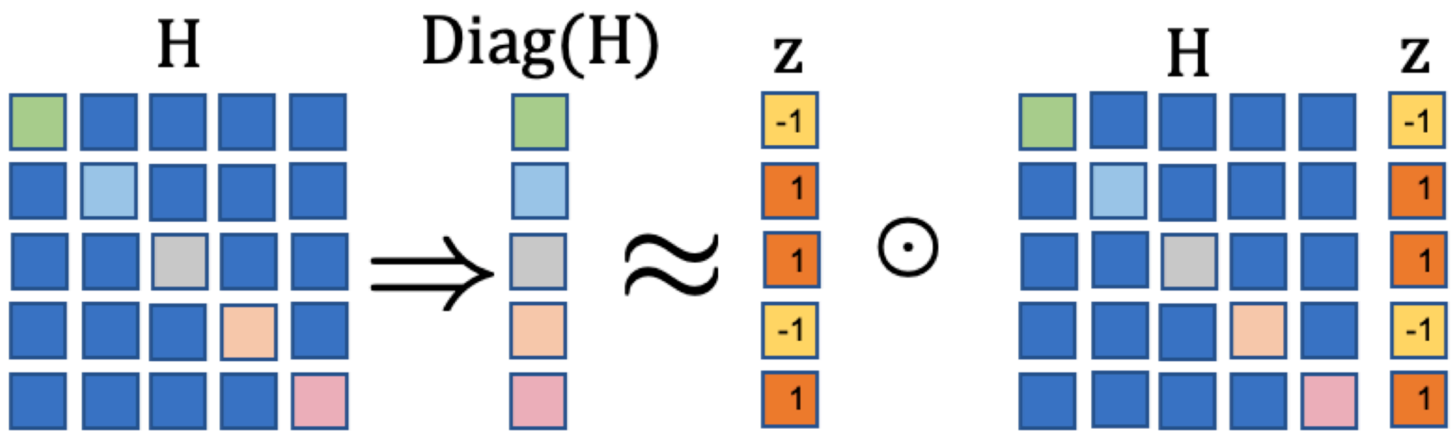
$g =$ 

$H =$ 

How can we get Diagonal without explicitly forming the Hessian?

Randomized Numerical Linear Algebra (RandNLA):

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



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

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

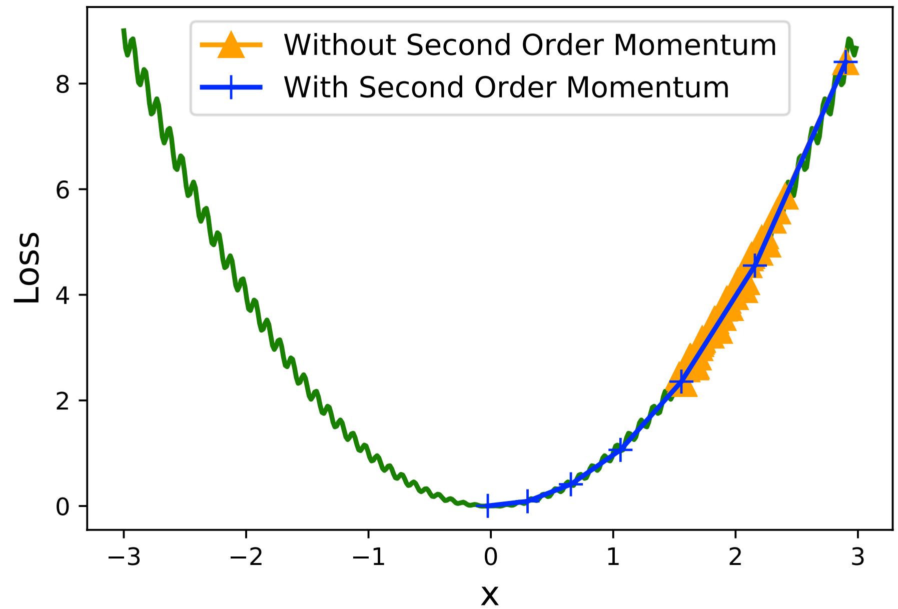
ADAHESSIAN algorithm is very simple and as follows:

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

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

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

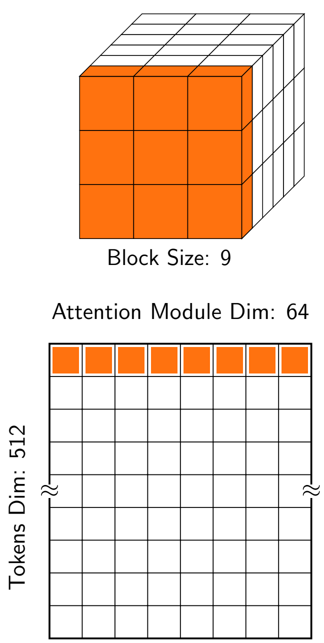
Where D is the Hessian diagonal



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

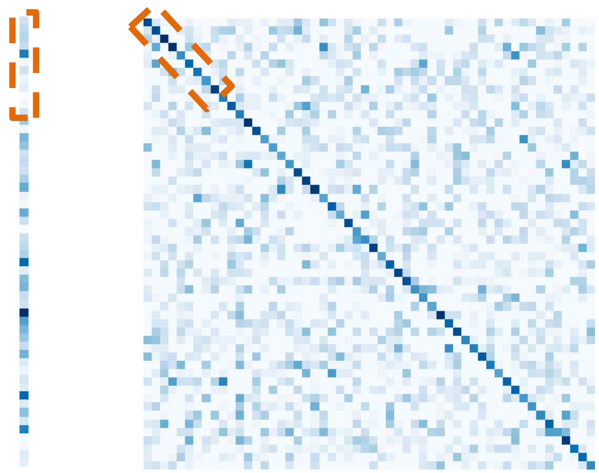
Spatial Smoothing

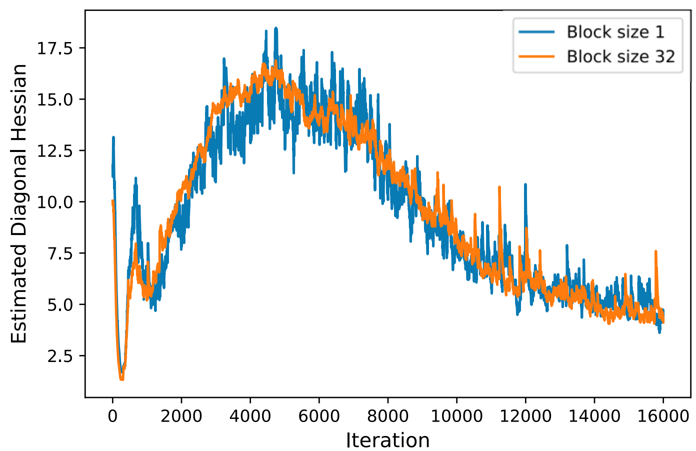
- We also incorporate spatial averaging to smooth out the stochastic Hessian noise across different iterations



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

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



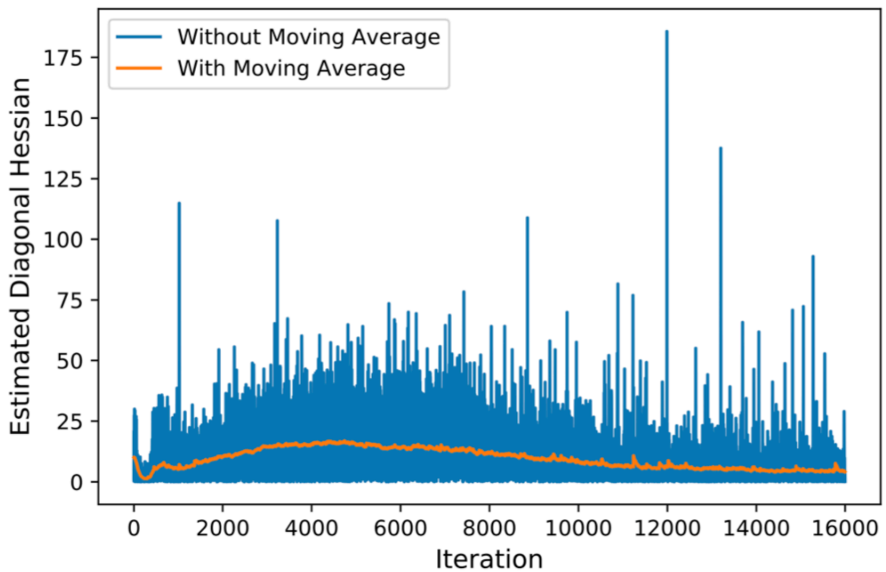


Machine Translation Task on IWSLT'14 Dataset

Variance Reduction

- Incorporating momentum for both first and second order term:

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



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 ResNet20	Cifar10 ResNet32	ImageNet 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	$28.57 \pm .15$	26.04
AdamW [24]	$35.66 \pm .11$	28.19
ADAHESSIAN	$\mathbf{35.79 \pm .06}$	**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	$\mathbf{51.5 \pm 1.2}$	**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

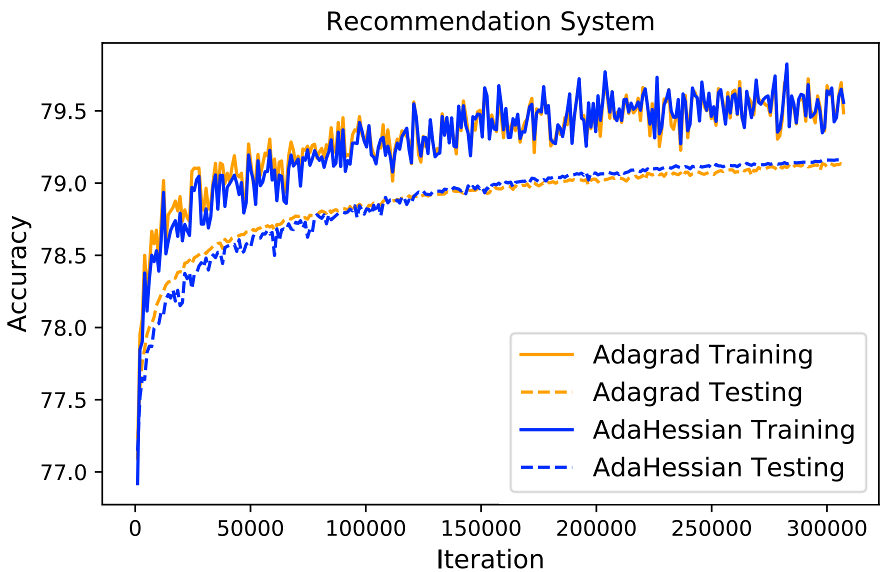
Higher accuracy is better

	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)	$92.13 \pm .08$	$92.40 \pm .04$	$92.06 \pm .18$	$92.17 \pm .21$	$92.16 \pm .12$
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:

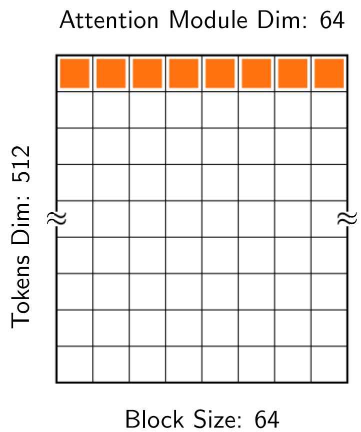
- AdaHessian still achieves acceptable performance even when scaling learning rate by10x, while ADAM diverges after just 6x scaling.

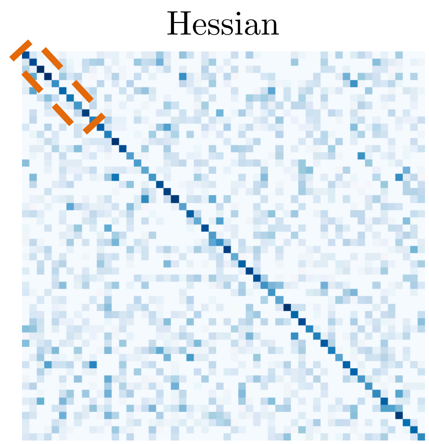
LR Scaling	**0.5**	**1**	**2**	**3**	**4**	**5**	**6**	**10**
AdamW	**35.42 ± .09**	35.66 ± .11	**35.37 ± .07**	**35.18 ± .07**	**34.79 ± .15**	14.41 ± 13.25	0.41 ± .32	Diverge
ADAHESSIAN	35.33 ± .10	**35.79 ± .06**	35.21 ± .14	34.74 ± .10	34.19 ± .06	**33.78 ± .14**	**32.70 ± .10**	**32.48 ± .83**

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

- Much work has shown benefits of first-order methods, **but in practice SGD is *very* brittle.**
  - Jin C, Ge R, Netrapalli P, Kakade SM, Jordan MI. How to escape saddle points efficiently, 2017
  - Duchi JC, Bartlett PL, Wainwright MJ. Randomized smoothing for stochastic optimization, 2012
  - Lee JD, Simchowitz M, Jordan MI, Recht B. Gradient descent only converges to minimizers, 2016
  - Dauphin, Y.N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. and Bengio, Y., Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014
  - Xu P, Roosta F, Mahoney MW, Second-Order Optimization for Non-Convex Machine Learning: An Empirical Study, 2018

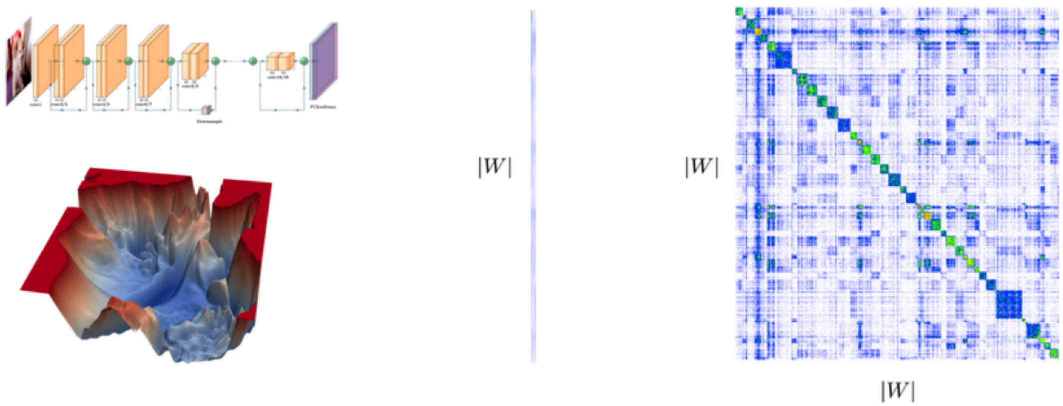
Some related Work

- Second-Order methods have been extensively explored in scientific computing, but they have not **yet** been been used as much as first-order methods for ML. Recent work includes:
  - Schaul T, Zhang S, LeCun Y. No more pesky learning rates, 2013
  - Bollapragada R, Mudigere D, Nocedal J, Shi HJ, Tang PT. A progressive batching L-BFGS method for machine learning, 2018
  - Martens J, Grosse R. Optimizing neural networks with kronecker-factored approximate curvature, 2015
  - Roosta-Khorasani F and Mahoney MW, Sub-Sampled Newton Methods I: Globally Convergent Algorithms, 2016
  - Wang S, Roosta-Khorasani F, Xu P, Mahoney MW. GIANT: Globally improved approximate Newton method for distributed optimization, 2018
  - Pilanci, Mert and Wainwright, Martin J, Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence, 2017
  - Bottou L, Curtis FE, Nocedal J. Optimization methods for large-scale machine learning, 2018

Some related Work: pyHessian

PyHESSIAN

$$\min_{w} E(w) = \frac{1}{N}\sum_{i=1}^{N} cost(w, x_i) \qquad \text{Gradient: } \frac{\partial E}{\partial w} \in \mathcal{R}^{|W|} \qquad \text{Hessian: } \frac{\partial^2 E}{\partial w^2} \in \mathcal{R}^{|W| \times |W|}$$



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

Thank You!

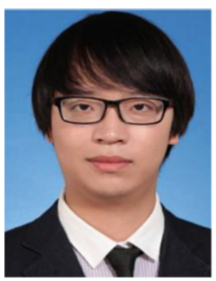
Please contact us if you have any questions:

{zheweiy, amirgh} @ berkeley.edu

mmahoney @ stat.berkeley.edu

Hessian tutorial: https://github.com/amirgholami/PyHessian/tree/master/pyhessian

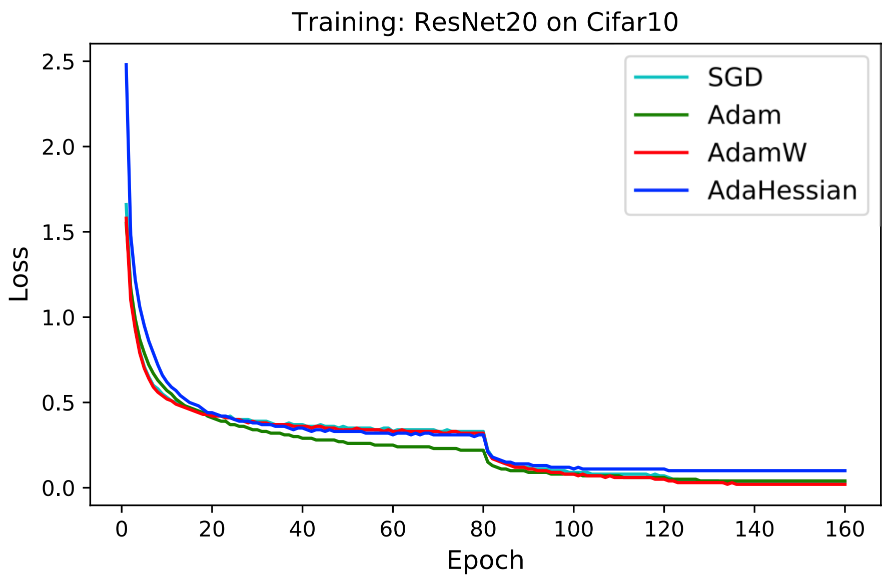
AdaHessian tutorial: https://github.com/yaozhewei/analyze_ada_hessian

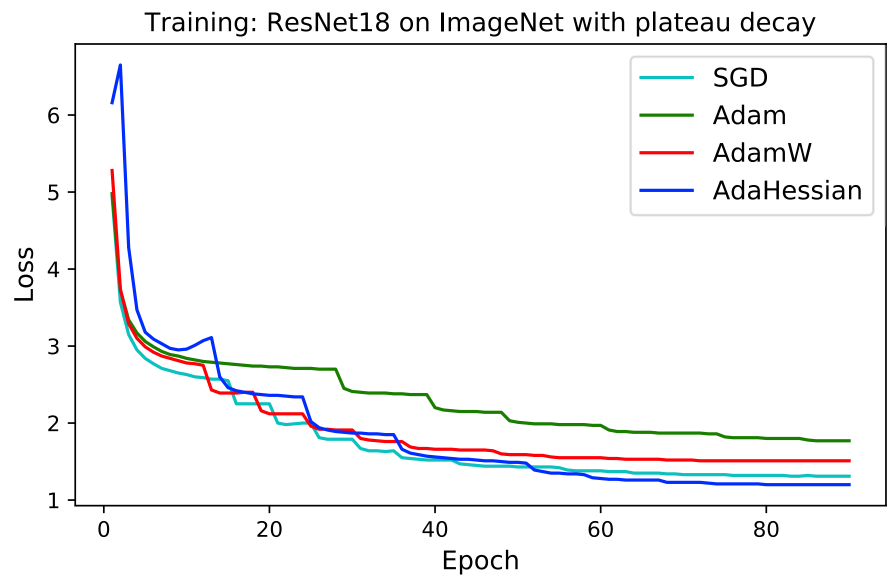




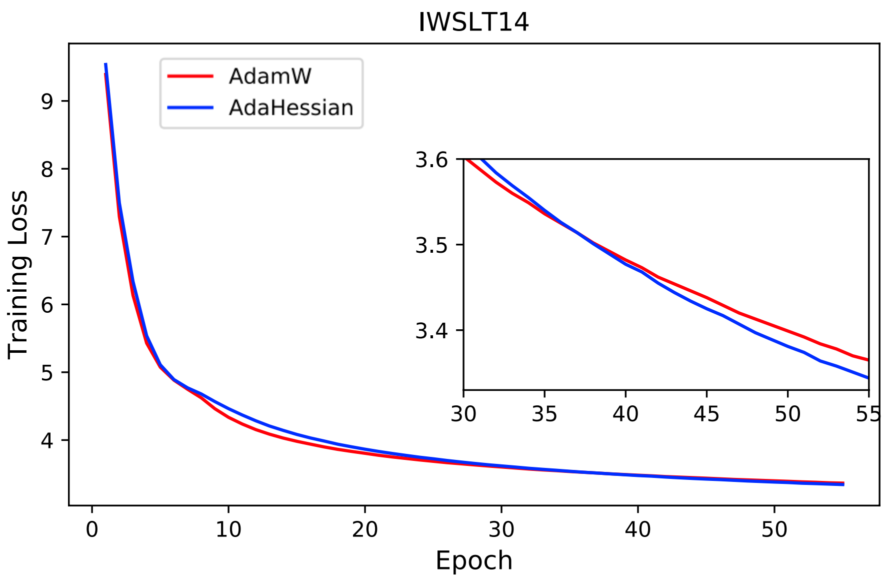

Extra

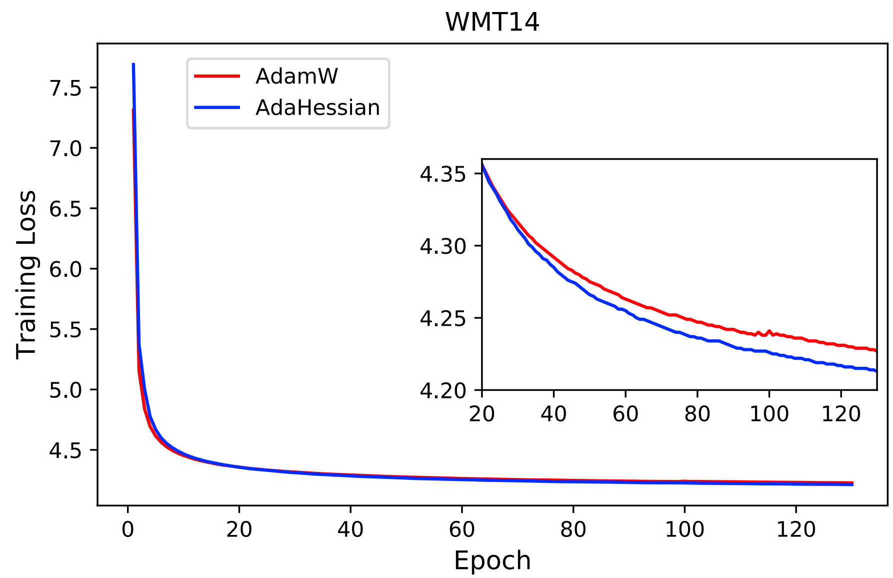
Results on Image Classification





Results on Machine Translation





Results on Language Modeling