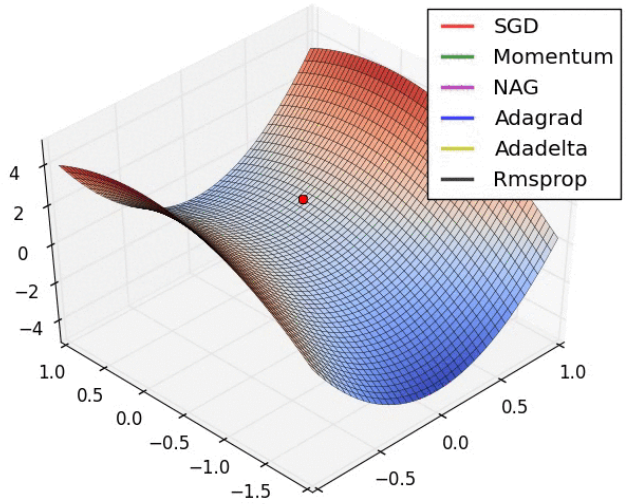
Second-order optimization for AI using HSS Matrices on distributed-memory setup

Severin Reiz, Technical University of Munich, Germany
Chair of Scientific Computing
Advisor: Prof. H.-J. Bungartz



France-Japan-Germany trilateral workshop
Convergence of HPC and Data Science for Future Extreme Scale Intelligent Applications
Session Chair: Michel Daydé

Motivation and Significance

Numerical optimization everywhere in scientific computing

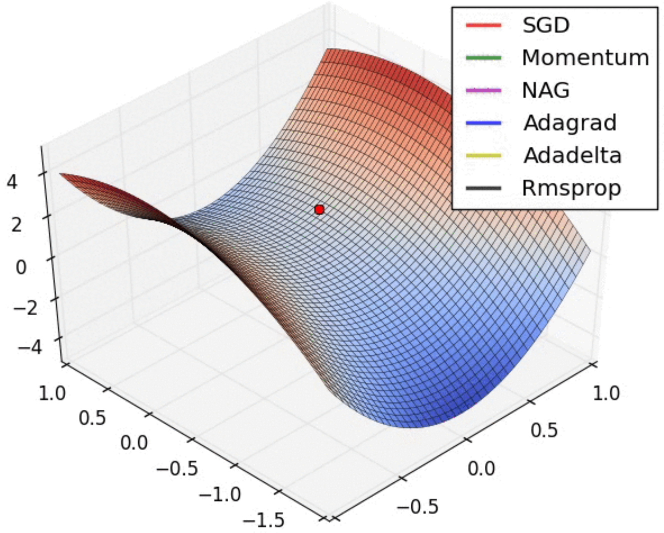
- AI models become increasingly complex
- Synergy between *AI* and *math libraries*

First-order methods

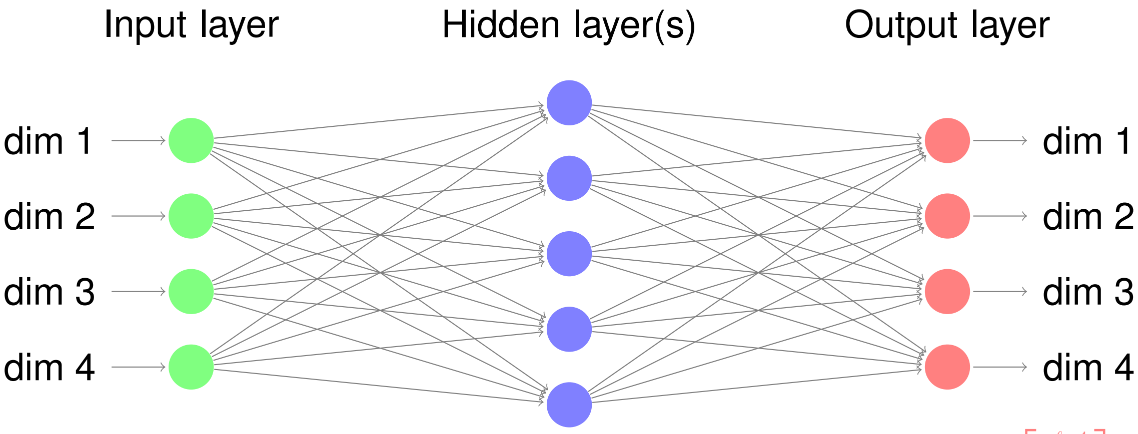
- Require fine hyper-parameter tuning
- No curvature $\rightarrow$ sensitive to ill-conditioning
- Often unable to fully exploit the power of distributed computing architectures

Higher-order methods (Newton, Quasi-Newton, etc)

- Mitigate effects of ill-conditioning
- Enough concurrency to exploit distributed computing architectures



Multilayer Perceptron Model



$$x_i^{\ell} = s(W_\ell x_i^{\ell-1})$$

W $= [W_{\ell=0}, W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}},\ ...]$

Loss function f:

mean squared error

$$f\left(W, x^{\ell=0}, y\right) = \left\|x^{L_{max}} - y\right\|^2$$

$$\min_W F := \sum_{i=1}^{n} f\left(W, x^{\ell=0}, y_i\right)$$

Optimize weight tensor W

$$\begin{bmatrix} x_0^{l=0} \\ x_1^{\ell=0} \\ x_2^{\ell=0} \\ x_3^{\ell=0} \end{bmatrix}$$

$$\begin{bmatrix} x_0^{l=1} \\ x_1^{\ell=1} \\ x_2^{\ell=1} \\ x_3^{\ell=1} \\ x_4^{\ell=1} \end{bmatrix} = s(W_{\ell=1} \cdot \begin{bmatrix} x_0^{\ell=0} \\ x_1^{\ell=0} \\ x_2^{\ell=0} \\ x_3^{\ell=0} \end{bmatrix})$$

$$W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$$

$$[x]^{\ell=2} = s(W_{\ell=2} \cdot \begin{bmatrix} x_0^{\ell=1} \\ x_1^{\ell=1} \\ x_2^{\ell=1} \\ x_3^{\ell=1} \\ x_4^{\ell=1} \end{bmatrix})$$

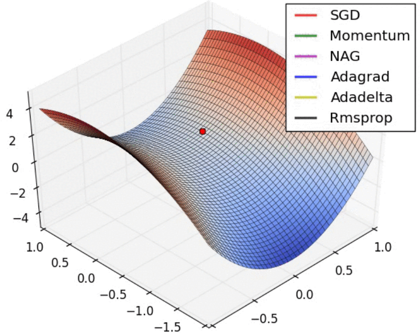
Optimize weight tensor W

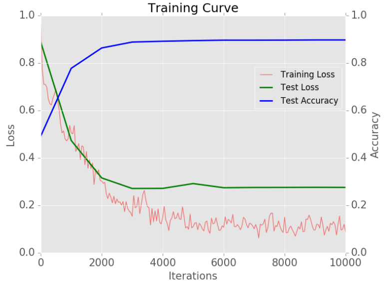
$$W = \left[\begin{bmatrix} w_o & w_1 & \dots & w_{d_0-1} \\ w_{d_0} & w_{d_0+1} & \dots & \\ \vdots & & & \end{bmatrix}_{\ell=0}, \begin{bmatrix} w_o & w_1 & \dots & w_{d_0-1} \\ w_{d_0} & w_{d_0+1} & \dots & \\ \vdots & & & \end{bmatrix}_{\ell=1}, \dots \right]$$

First-order methods

Gradient $\mathbf{g} = \frac{df}{dW}$ $\boxed{\rightarrow \mathbb{R}^{N\times 1} \text{ (Linear complexity)}}$

(Stochastic) Gradient Descent	$w_{k+1} = w_k - \alpha \mathbf{g}\|_{w_k}$
Momentum (2-step)	$v_{k+1} = \lambda v_k - \alpha_k \mathbf{g}\|_{w_k}$ $w_{k+1} = w_k + v_k$
NAG (Nesterov acc grad)	$v_{k+1} = \lambda v_k - \alpha_k \mathbf{g}\|_{w_k+\lambda v_k}$ $w_{k+1} = w_k + v_k$
AdaGrad	$r_{k+1} = r_k + \mathbf{g}\|_{w_k} * \mathbf{g}\|_{w_k}$ $w_{k+1} = w_k - \frac{alpha}{\delta I + \sqrt{diag(r_k+1)}} \mathbf{g}\|_{w_k}$
RMSprop	$r_{k+1} = \rho r_k + (1-\rho)\mathbf{g}\|_{w_k} * \mathbf{g}\|_{w_k}$





Second-order

Hessian: $H_f = \frac{d^2 f}{dW^2}$ $\boxed{\rightarrow \mathbb{R}^{N\times N} \text{(Quadratic complexity)}}$

Newton step: $w_{k+1} = w_k - (H_f)^{-1} g|_{w_k}$

- `[First-order]` Many synchronization barriers for first order (Broadcast w)
- `[Second-order]` Dense Hessian for 1M network: $\mathbb{R}^{1M\times 1M}$ **4 TB** memory

How dare you use second-order methods for AI?

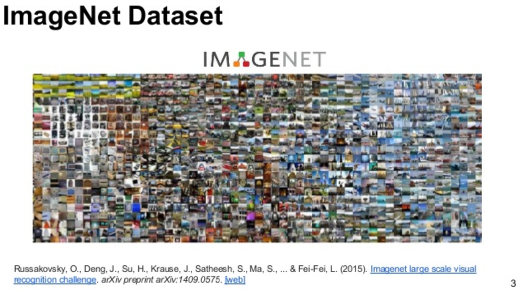
Beyond First Order Methods in ML

Workshop at NeurIPS 2019 (Dec 8-14)

- Donald Goldfarb: Economical use of second-order information in training machine learning models
- James Martens: K-FAC: Extensions, improvements, and applications
- . . .
- h-matrix approximation for Gauss-Newton Hessian

Interface for image classification

- Inhouse code for Auto-Encoder Network
- tensorflow.contrib.slim Link



Hierarchical (semi-separable) Matrix `H(SS)`

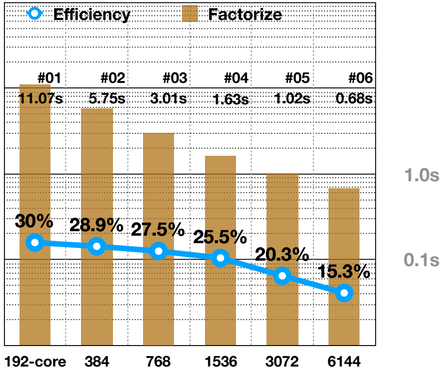
Hierarchical Off-Diagonal Low-Rank (HODLR) + sparse

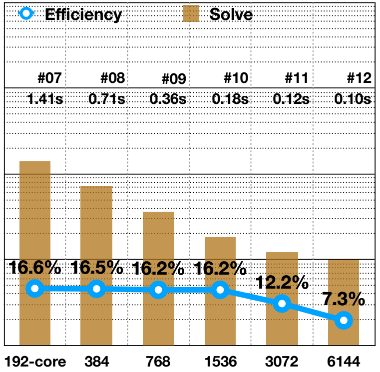
$$\tilde{K}_{\alpha\alpha} = \begin{bmatrix} \tilde{K}_{\mathtt{ll}} & 0 \\ 0 & \tilde{K}_{\mathtt{rr}} \end{bmatrix} + \begin{bmatrix} 0 & UV_{\mathtt{lr}} \\ UV_{\mathtt{rl}} & 0 \end{bmatrix} + \begin{bmatrix} 0 & S_{\mathtt{lr}} \\ S_{\mathtt{rl}} & 0 \end{bmatrix}$$

$$\left[\begin{array}{cc|cc} Z(\mathtt{l})_{\mathtt{CC}} & Z(\mathtt{l})_{\mathtt{CF}} & \Sigma(\alpha)^T & 0 \\ Z(\mathtt{l})_{\mathtt{FC}} & Z(\mathtt{l})_{\mathtt{FF}} & 0 & 0 \\ \hline \Sigma(\alpha) & 0 & Z(\mathtt{r})_{\mathtt{CC}} & Z(\mathtt{r})_{\mathtt{CF}} \\ 0 & 0 & Z(\mathtt{r})_{\mathtt{FC}} & Z(\mathtt{r})_{\mathtt{FF}} \end{array}\right]$$

- $\mathcal{O}(N \log N)$ entries K_{ij} and compression time

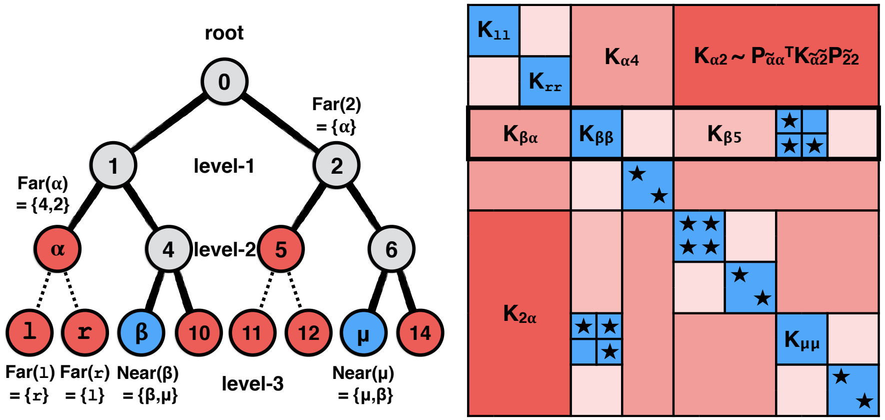
1. Shared-memory `MATVEC`
   SC17: Geometry-oblivious FMM for compressing dense SPD matrices
2. Asynchronous distributed `MATVEC`
   SC18: Distributed-memory hierarchical compression of dense SPD matrices
3. Distributed-memory `Linear Solver`
   MCSoC: Distributed O(N) Linear Solver for Dense Symmetric Hierarchical Semi-Separable Matrices





Hyper-parameters (Auto-Tuning)

- GOFMM is a tree-code
  Diminishing parallelism when approaching the root



m: leaf node size

s: compression rank for off-diag

stol: relative compression tolerance

nbs: number of neighbors

Hints

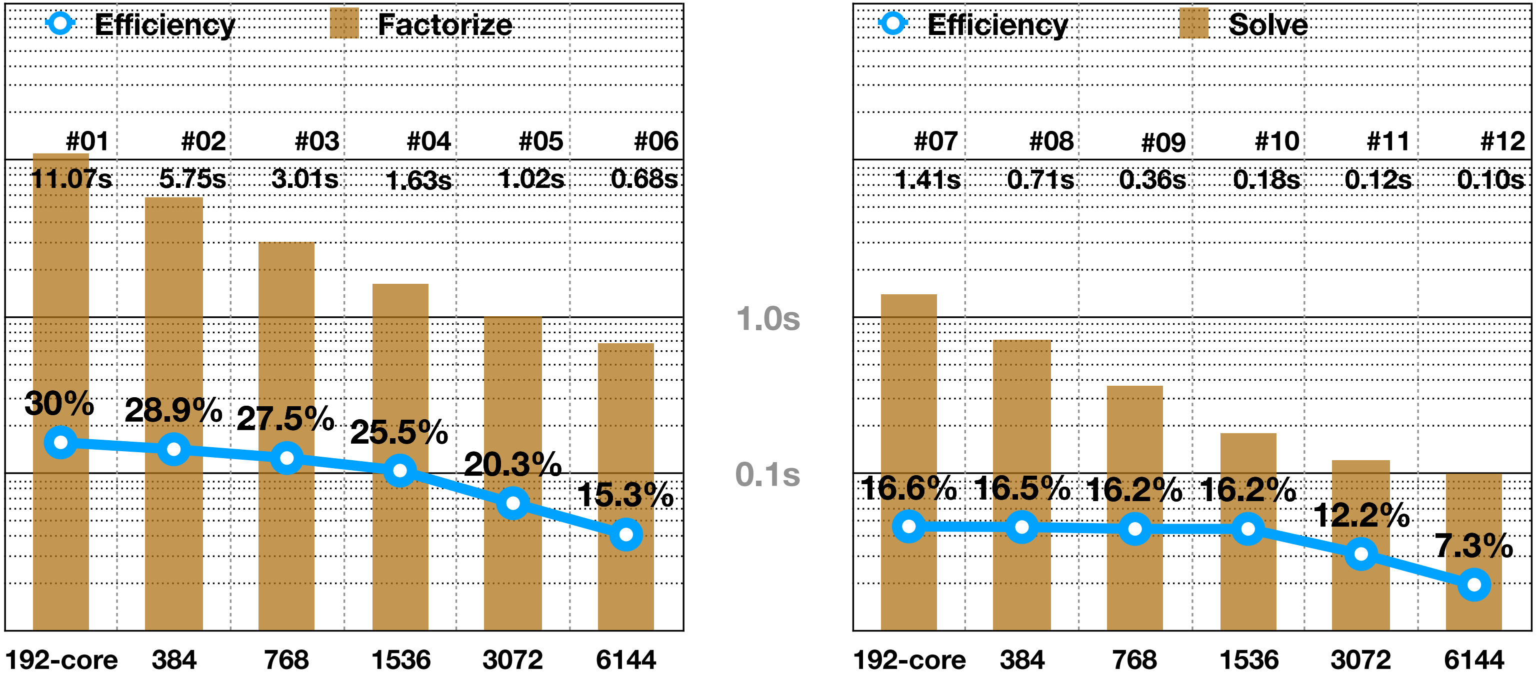
- m and s sufficiently large for BLAS
- Tune m and s from 64 to $2,048$

Comparison to other approximations

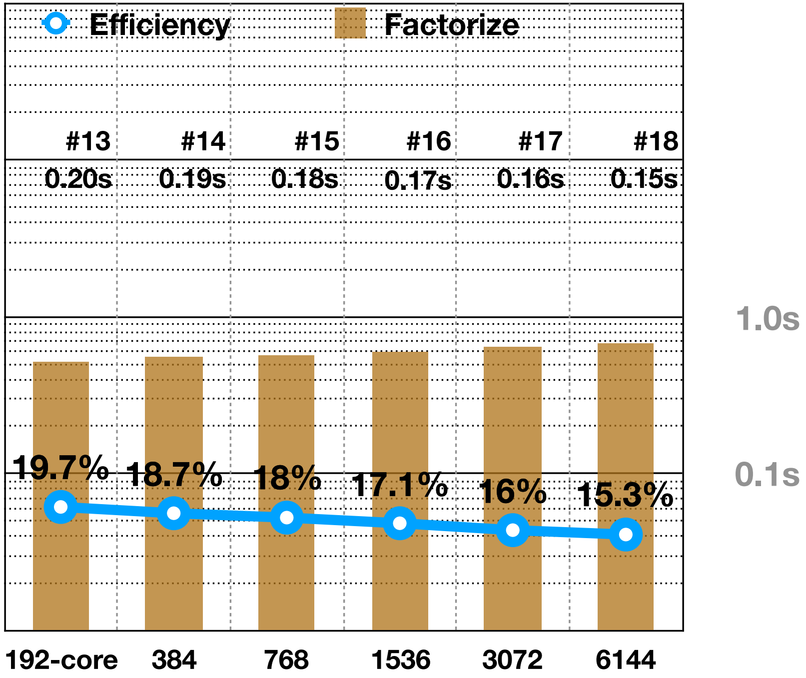
		HM-low	HM-high	RSVD-low	RSVD-high	KFAC-low	KFAC-high
AE (a)	%K	1.23%	11.77%	1.40%	12.14%	1.40%	12.14%
	ε_F	1.7E-1	4.7E-4	4.3E-1	5.1E-3	1.2E-1	7.3E-2
AE (b)	%K	0.53%	4.62%	0.62%	4.65%	0.62%	4.65%
	ε_F	5.7E-3	6.4E-4	8.4E-1	2.3E-1	1.7E-1	3.8E-2
AE (c)	%K	0.28%	2.31%	0.32%	2.38%	0.32%	2.38%
	ε_F	4.2E-3	4.9E-4	9.1E-1	2.1E-1	1.6E-1	4.1E-2

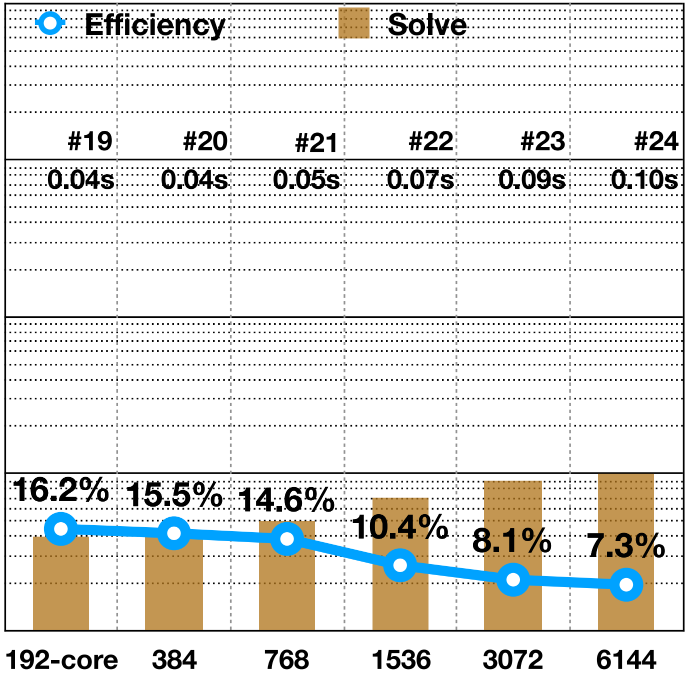
C. Chen, S. Reiz, C. Yu, H.-J. Bungartz, G. Biros: Fast Evaluation and Approximation of the Gauss-Newton Hessian Matrix for the Multilayer Perceptron, SIAM Journal on Mathematics of Data Science (SIMODS). SIAM, submitted OCt 2019. https://arxiv.org/abs/1910.12184

Strong scaling



Weak scaling





Conclusions and future work

Second-order optimization for AI using HSS Matrices on distributed-memory setup

- HSS $O(N)$ *enables* second-order ML

- Currently no (distributed) GPU support
  Now we only call `LAPACK` functions

Thank you for your attention!
Code available:
`https://github.com/severin617/hmlp-1`

C. Chen, S. Reiz, C. Yu, H.-J. Bungartz, G. Biros: Fast Evaluation and Approximation of the Gauss-Newton Hessian Matrix for the Multilayer Perceptron, SIAM Journal on Mathematics of Data Science (SIMODS). SIAM, submitted OCt 2019. https://arxiv.org/abs/1910.12184

Accuracy and Preconditioned conjugate gradient

#	Matrix	Spy	Pred	$\mathtt{MATVEC}_\varepsilon$	$\mathtt{SOLVE}_\varepsilon$	$\mathtt{SOLVE}_\mathtt{t}$	#Iter
1	COV	0%	True	1E-2	4E-11	0.43	0
2	COV	1%	True	7E-3	8E-4	0.43	940
3	COV	1%	False	7E-3	6E-2	-	2,532
4	K02	0%	True	1E-2	1E-6	0.13	0
5	K02	1%	True	8E-3	3E-5	0.13	0
6	K02	1%	False	8E-3	4E-4	-	1
7	K13	0%	True	5E-5	9E-7	0.13	0
8	K13	1%	True	4E-5	1E-6	0.13	0
9	K13	1%	False	4E-5	2E-5	-	1
10	K15	0%	True	5E-1	1E-6	0.05	0
11	K15	1%	True	1E-1	2E-4	0.05	3
12	K15	1%	False	1E-1	8E-4	-	3

Table : **Preconditioner experiments using precondition conjugate gradient (PCG).** All experiments were done on four Stampede-2 nodes (192 Skylake cores) for different matrices. Throughout the experiments we control the percentage of the sparse correction in and the type of the preconditioner (identity or HSS). The results are reported base on a mixed stopping