Training Large-Scale Neural Networks with a Newton Conjugate Gradient Method (Newton-CG)

Severin Reiz[§], Tobias Neckel[§], Hans-Joachim Bungartz[§]
[§] Technical University of Munich, reiz@in.tum.de, neckel@in.tum.de, bungartz@in.tum.de

Introduction

Approach: Newton-CG

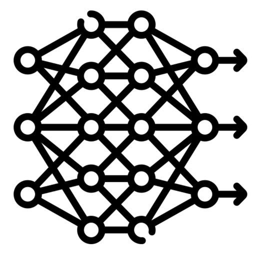
- **Methods** from scientific computing domain
  - Previous work on multilayer perceptron with GOFMM[1]
  - Approximation of neural network Hessians
- **HPC:** Exploit potential of supercomputers
  - **Concurrency:** Choose suitable algorithms for parallel computing
  - **Performance Portability:** For upcoming new GPU and CPU

Intro

Example of Machine Learning Tasks: Classification

Weight optimization in neural networks

Feedforward Neural Network



$$y = f(X, \mathbf{W}) = f^{(n)}(\dots f^{(2)}(f^{(1)}(x)))$$

Minimize function

$$\min_{\mathbf{W}\in\mathbb{R}^{n\times n}} L(X, Y, \mathbf{W})$$

Loss functions $L_{entr}(x, y, \mathbf{W}) = -\sum_{i=1}^{N} y_i \log(f^{(n)})$ or $L_{RMSE} = \sum_{i=1}^{N} ||f - y_i||_2$

(Stochastic) gradient descent $W_{k+1} = W_k - \sigma_k \nabla_w L(X, Y, wk)$

AdaGrad (popular in ML, similar to adam) $W_{k+1} = W_k - \alpha_k \frac{s_k}{\delta + \sqrt{r_k}}$

Newton-Method (problematic in ML due to size of inverse Hessian) $W_{k+1} = W_k - \mathbf{H_L}^{-1} \nabla L(W_k)$

Our approach: Efficient Hessian Newton

1. Fast Hessian **Matrix-Vector Multiply** (MatVec): Pearlmutter
2. A few iterative solvers require matvec only: do a few cg-steps
3. Tikhonov regularization with $\tilde{\mathbf{H}}_\mathbf{L} = \mathbf{H_L} + \tau\mathbf{I}$
4. Armijo feasibility check and update weights

Integration in TensorFlow for:
Regression, VAE, TensorFlow-Slim image classification model library , Transformer (loop unroll)

Algorithmic ingredients

$$H_L(\mathbf{W})s = \begin{pmatrix} \sum_{i=1}^{n} s_i \frac{\delta^2}{\delta w_1 \delta w_i} L(\mathbf{W}) \\ \sum_{i=1}^{n} s_i \frac{\delta^2}{\delta w_2 \delta w_i} L(\mathbf{W}) \\ \vdots \\ \sum_{i=1}^{n} s_i \frac{\delta^2}{\delta w_n \delta w_i} L(\mathbf{W}) \end{pmatrix} = \begin{pmatrix} \frac{\delta}{\delta w_1} \sum_{i=1}^{n} s_i \frac{\delta}{\delta w_i} L(\mathbf{W}) \\ \frac{\delta}{\delta w_2} \sum_{i=1}^{n} s_i \frac{\delta}{\delta w_i} L(\mathbf{W}) \\ \vdots \\ \frac{\delta}{\delta w_n} \sum_{i=1}^{n} s_i \frac{\delta}{\delta w_i} L(\mathbf{W}) \end{pmatrix} = \nabla_w(\nabla_w L(\mathbf{W}) \cdot s)$$

procedure PEARLMUTTER($X, Y, \mathbf{W}, s$)
Require: H_L, s, X, Y, W: Compute $H_L s = \nabla_w(\nabla_w L(\mathbf{W}) \cdot s)$
Require: W_0: Initial estimate for $\mathbf{W}$.
 $g_0 \leftarrow$ gradient($L(\mathbf{W})$) ▷ Back-Prop
 intermediate $\leftarrow$ matmul(g_0, s) ▷ Matrix-Multiplication
 $H_L s \leftarrow$ gradient(intermediate) ▷ Back-Prop
 return $H_L s$
end procedure

- no direct matrix access neccesary
- cg only needs matvecs

procedure CONJUGATE GRADIENTS($\tilde{H}_L, b$)
Require: $\tilde{H}_L, b$: Solve $\tilde{H}_L x = b$ for x.
Require: x_0: Initial estimate for x.
 $r_0 \leftarrow \tilde{H}_L x_0 - b,\ p_0 \leftarrow -r_0,\ k \leftarrow 0$
 while r_k too large **do**
  $\alpha_k \leftarrow \frac{r_k^\top r_k}{p_k^\top \tilde{H}_L p_k}$ Compute step size
  $x_{k+1} \leftarrow \alpha_k p_k$
  $r_{k+1} \leftarrow r_k + \alpha_k \tilde{H}_L p_k$
  $\beta_{k+1} \leftarrow \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$
  $p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$
  $k \leftarrow k + 1$
 end while
end procedure

procedure NEWTON-CG
Require: $L(\mathbf{W})$: Loss function with weights $\mathbf{W}$
Require: $\mathbf{W}_0$: Starting point
Require: τ: Tikhonov regularization/damping factor
 $k \leftarrow 0$
 while $\mathbf{W}_k$ not converged **do**
  $k \leftarrow k + 1$
  $p_k \leftarrow \mathtt{CG}((H_L + \tau I), -\nabla L(\mathbf{W}_k))$ Tikhonov
  if $\nabla L(\mathbf{W}_k)^\top p_k > \tau$ **then** $p_k \leftarrow -\nabla L(\mathbf{W}_k)$
  end if
  $\alpha_k \leftarrow \alpha$ Learning rate scheduler
  $\mathbf{W}_k \leftarrow \mathbf{W}_{k-1} + \alpha_k p_k$
 end while
end procedure

Methods

Results

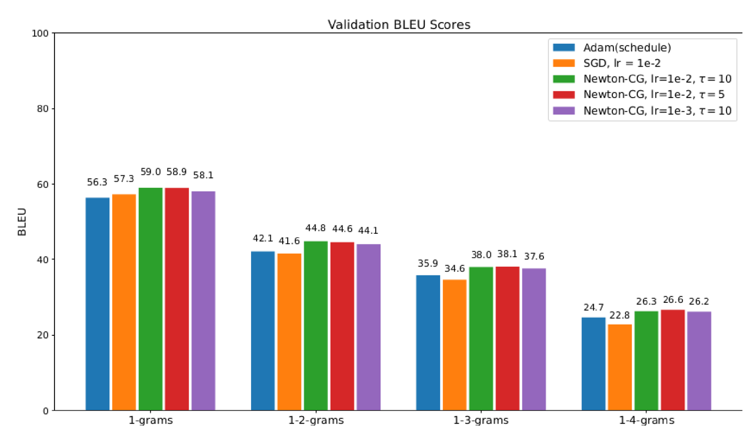
Comparison: Qualitative behavior of adam, SGD, and newton-cg.
Metrics: Loss, Stability and Epochs

scenario description	optimizer	adam	SGD	newton-cg
regression	life expectancy	- -	-	++
	boston housing	+	- -	++
variational autoencoder	mnist	+	+	○
bayesian nn	mnist	○	+	+
	wisc	++	+	○
	breakhis	-	-	++
image-classification	simple CNN mnist	○	○	+
	resnet-50 imagenet	○	○	○
	mobile-net imagenet	+	○	-
natural language	transformer	○	○	+

For all results and plots check github or see paper[2]
`https://github.com/severin617/Newton-CG`

Newton-CG Github

arXiv paper



BLEU scores (higher is better) for Portuguese-English translation
BLEU is measure for translating sentences (more than 1 word)

- Last-layer training suitable for transfer learning
- Pre-training with SGD
- Flexible Learning rate scheduler
- Bayesian Neural Networks using TF Probability
- Similar training behavior than literature[3,4]
- For natural language processing (transformer) validation scores of newton better than adam, sgd
- DGX-1 with horovod GPU parallelization (data parallelism)

Newton-CG runtimes per epoch with batch-size 512, ResNet-50 on ImageNet

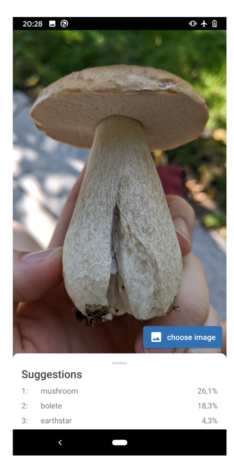
	1 GPU	2 GPUs	4 GPUs	8 GPUs
A100 runtime	238s	121s	65s	37s
A100 parallel efficiency	100%	98.3%	91.5%	80.4%

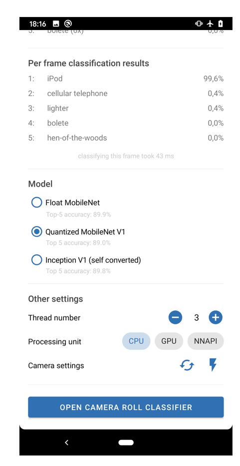
Results and comparison

Conclusion

- Method design
  - Run prominent models from current machine learning peers
  - Compare to current second-order literature, no claim to be superior
  - Newton-CG is a neat formulation for approximate newton
  - Very problem dependent (Convexity?). Less overfitting than SGD, adam (validation set accuracy often better with ncg, see BLEU scores)
  - Works well with data parallelism (80% parallel efficiency on 8 GPUs)
- Library design
  - **Community/reproducibility:** Newton-CG on github
  - For public outreach live **image classification** smartphone **app** TUM-Lens runs **locally** on your smartphone
    Use Newton-trained checkpoints from above
    Object detection, sign language recognition, model zoo

Google Play Store





Conclusion

References

[1] C. Chen, S. Reiz, C. D. Yu, H.-J. Bungartz, and G. Biros, “Fast approximation of the gauss–newton hessian matrix for the multilayer perceptron,” *SIAM Journal on Matrix Analysis and Applications*, vol. 42, no. 1, pp. 165–184, 2021.

[2] S. Reiz, T. Neckel, and H.-J. Bungartz, “Neural nets with a newton conjugate gradient method on multiple gpus,” in *accepted for publication*, PPAM, https://arxiv.org/abs/2208.02017, 2022.

[3] K. Osawa, Y. Tsuji, Y. Ueno, A. Naruse, R. Yokota, and S. Matsuoka, “Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12359–12367, 2019.

[4] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney, “Adahessian: An adaptive second order optimizer for machine learning,” *arXiv preprint arXiv:2006.00719*, 2020.