FAST SAMPLING OF OPTIMAL PARAMETERS FOR ML

Natalia Uspenskaia
uspen.nat@gmail.com

Vadim Selyutin
vadim.selyutin@skoltech.ru

Egor Gladin
egor.gladin@skoltech.ru

Semjon Glushkov
semyon.glushkov@skoltech.ru

December 21, 2019

ABSTRACT

Model's true parameters can be viewed as a vector that came from distribution $\mathcal{N}(\theta_{MLE}, \frac{1}{p}\mathcal{I}(\theta_{MLE})^{-1})$, where p is the number of samples, θ_{MLE} is maximum likelihood estimate and $\mathcal{I}(\theta)$ is Fisher information matrix. Therefore, sampling from this distribution is often required to estimate confidence level of predictions. If z has standard normal distribution, $\theta_{MLE} + \frac{1}{\sqrt{p}}\mathcal{I}(\theta_{MLE})^{-1/2}z$ has the desired distribution. We leverage Lanczos method and Pearlmutter trick to drastically reduce the complexity of computation of $\mathcal{I}(\theta_{MLE})^{-1/2}z$.

1 Introduction

Let $\boldsymbol{y} = (y_1; y_2; \dots y_p)$ be a vector of i.i.d. random variables from one of a family of distributions on $\mathbb{R}$ indexed by a p-dimensional parameter $\boldsymbol{\theta} = (\theta_1; \dots \theta_n)$ with density $f(y|\boldsymbol{\theta})$. Then the likelihood function of $\boldsymbol{\theta}$ is given by

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{p} f(y_i|\boldsymbol{\theta})$$

Maximum likelihood estimation of optimal parameters is a vector θ_{MLE} that maximizes $L(\boldsymbol{\theta})$. It is equivalent (but more convenient) to optimize log-likelihood function:

$$l(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta}) = \sum_{i=1}^{p} \log f(y_i|\boldsymbol{\theta})$$

Fisher information matrix is given by $\mathcal{I}(\theta) = -E(\nabla^2 l(\boldsymbol{\theta}))$, i.e. it equals the expectation of Hessian matrix of $l(\boldsymbol{\theta})$ taken over the distribution of $\boldsymbol{y}$ at a fixed point $\boldsymbol{\theta}$.

As $p \to \infty$, true parameters of the model can be viewed as a vector that came from distribution $\mathcal{N}(\theta_{MLE}, \frac{1}{p}\mathcal{I}(\theta_{MLE})^{-1})$, see [1] (page 185). This helps us estimate how far MLE parameters may deviate from the true parameters. However, we want to know the level of confidence of predictions of our model. Efficient sampling of parameters from the mentioned distribution and making predictions with these parameters would give us the desired knowledge.

2 Method description

Note that if $z \sim \mathcal{N}(0, I_n)$ (standard normal distribution) and

$$\hat{\theta} = \theta_{MLE} + \frac{1}{\sqrt{p}}\mathcal{I}(\theta_{MLE})^{-1/2}z$$

then $\hat{\theta} \sim \mathcal{N}(\theta_{MLE}, \frac{1}{p}\mathcal{I}(\theta_{MLE})^{-1})$. This gives us a way to sample parameters. However, straightforward calculation of $\mathcal{I}(\theta_{MLE})^{-1/2}z$ is computationally expensive. We further denote $A = \mathcal{I}(\theta_{MLE})$ for brevity.

2.1 Fast $A^{-1/2}z$

For fast computation we approximate $A^{-1/2}z$ in the Krylov subspace $K_m(A,z) = \langle z, Az, \dots, A^{m-1}z\rangle$. Consider orthonormal basis in $K_m(A,z)$ that was obtained via Lanczos method. Its vectors form left orthogonal matrix V_m, where first column is $\frac{z}{||z||_2} \Rightarrow z = ||z||_2 V_m e_1,\ e_1 = (1;0;\dots 0)$. Best approximation in the Krylov subspace is the solution of least squares:

$$A^{-1/2}z \simeq V_m(V_m^T V_m)^{-1} V_m^T A^{-1/2} z = V_m V_m^T A^{-1/2} z = \|z\| V_m V_m^T A^{-1/2} V_m e_1. \tag{1}$$

Lanczos method satisfies

$$AV_m = V_m T_m + t_{m,m-1} q_m e_m^T. \tag{2}$$

Using properties of intermediate terms in the procedure we obtain

$$T_m = V_m^T A V_m. \tag{3}$$

Matrix V_m has orthogonal columns, hence, T_m to the power $-1/2$ is

$$T_m^{-1/2} = V_m A^{-1/2} V_m^T. \tag{4}$$

Using this result we obtain final formula

$$A^{-1/2}z \simeq \|z\| V_m T_m^{-1/2} e_1, \tag{5}$$

Where T_m is tridiagonal. A function applied to the tridiagonal matrix can be computed much faster than for a dense matrix.

2.2 Fast Hessian-vector multiplication

A key point in the previous subsection is that we need a way for a fast computation of Az. When $A = \nabla^2 F$ is a Hessian of some function, it is possible to speed up matvec using a concept of automatic differentiation [2].

Given the function $F : \mathbb{R}^n \to \mathbb{R}$ and the vector z, one can accomplish this by first computing the directional derivative $\nabla F \cdot z$ through the forward mode and then applying the reverse mode on this result to get $\nabla^2 F z$ [3]. This computes Az with $O(n)$ complexity, even though A is a $n \times n$ matrix.

3 Implementation

GitHub: `https://github.com/vadimselyutin/Fast_sampling_of_optimal_parameters_for_ML`

Google Colab: `https://colab.research.google.com/drive/1FGEjZ_OG-F9EXEBE9bOk2tPx-a5gj26c`

Although Pearlmutter trick suggests using both forward and backward mode differentiation, PyTorch only supports the latter. Calculation of $\nabla^2 z$ can still be done in $O(n)$ by firstly computing ∇F through a backward mode differentiation, and then applying backward mode differentiation to ∇F initialized with z. This is the approach we use in our implementation.

Our project implementation consists of three Python scripts and supports CUDA acceleration .

3.1 slq_upd.py

Lanczos method itself is implemented in this module. The key function _lanczos_m_upd computes symmetric $m \times m$ tridiagonal matrix T and matrix V with orthogonal rows constituting the basis of the Krylov subspace $K_m(A,x)$, where x is an arbitrary starting unit vector.

The main function parameters are the following:

- A – Scipy Linear Operator from which we want to obtain the Krylov subspace basis vectors;
- m – the number of basis vectors to find;
- matrix_shape – the shape of the linear operator matrix A;
- orthtol – orthogonality tolerance for computation of the resulting vectors.

As the basis for this script we've taken slq.py from [4]. Our main contribution here is that we updated it in such a way, that now the _lanczos_m_upd function is able to process Scipy Linear Operator as the input parameter instead of the usual numpy matrix A as it was before.

3.2 lanczos_upd.py

The main purpose of lanczos_upd.py is to transform the input ModelHessianOperator object to the ScipyLinearOperator object and then call the _lanczos_m_upd function.

3.3 hvp_operator_upd.py

This is the key script to be launched for the computations. In this module first we define a linear operator ModelHessianOperator(Operator) to compute the hessian-vector product for a given pytorch model using subsampled data. Second, we upload data (i.e. the MNIST dataset), set training parameters and build pytorch model to be trained. After that the desired tridiagonal matrix T and matrix V with orthogonal rows are calculated via the lanczos function from lanczos_upd.py.

One might notice that the part of building an ML model is a little bit different on the Colab version of our project implementation. To be able to measure the running time of our algorithm, we implemented the create_model(a, b, c) function there which builds CNNs with a different number of parameters depending on the imput parameters a, b, c.

The idea of hvp_operator_upd.py was taken from the [5].

4 Experiments

First, we validated the described algorithm to compute $A^{-1/2}z$ for random $n \times n$ matrices (we considered $n = 1000$) s.t. $A \succ 0$. We computed a relative accuracy of our approximation depending on the number of Lanczos vectors m in the Krylov subspace setting the orthtol parameter $= 10^{-5}$. This dependence is shown in the figure 1. Fig 1.1 displays that it is not required to compute all Lanczos vectors to obtain enough accuracy which decreases the computation time. The number of vectors m can be determined by the desired accuracy. Fig 1.2 reveals the time complexity depending on the number of computed basis vectors m.

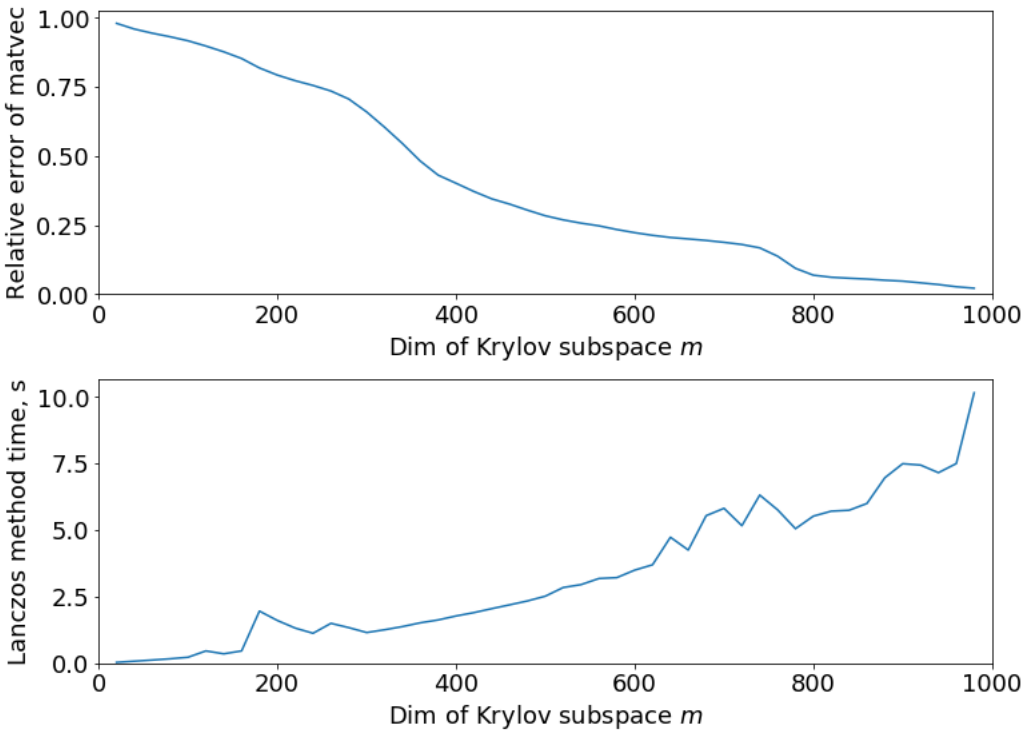

Figure 1: Accuracy and complexity for random matrix.

MNIST dataset was taken to train convolutional neural networks. The dependence of time complexity by number of model parameters is shown in the figure 2.

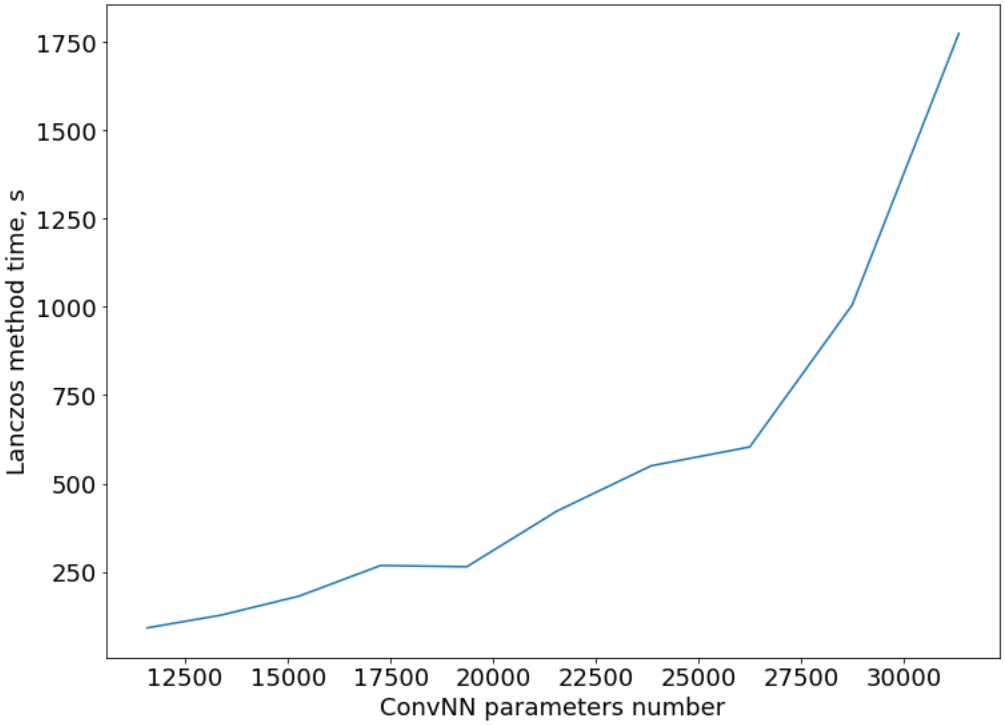

Figure 2: Complexity depending on the number of CNN parameters

5 Team members contribution

1. Scientific validation of the project: literature review, theoretical considerations (Egor, Natalia, Semyon, Vadim)
2. Implementation of a fast hessian-vector product algorithm (Egor, Semyon)
3. Implementation of Lanczos method with predetermined accuracy of orthogonality (Natalia, Vadim)
4. Time tests of the algorithm for neural networks with different number of weights (Semyon, Vadim)
5. Accuracy test of the implementation of Lanczos method (Egor, Natalia)
6. Presentation (Egor, Natalia, Semyon, Vadim)
7. Report (Egor, Natalia, Semyon, Vadim)

References

[1] Robert W Keener. *Theoretical statistics: Topics for a core course*, page 185. Springer, 2011.

[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153), 2018.

[3] Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

[4] Anton Tsitsulin. Intrinsic multi-scale evaluation of generative models. `https://github.com/xgfs/msid/blob/master/msid/slq.py`.

[5] Noah Golmant, Zhewei Yao, Amir Gholami, Michael Mahoney, Joseph Gonzalez. Pytorch-hessian-eigentings: efficient pytorch hessian eigendecomposition. `https://github.com/noahgolmant/pytorch-hessian-eigenthings`.

[6] EJ Allen, J Baglama, and SK Boyd. Numerical approximation of the product of the square root of a matrix with a vector. *Linear Algebra and its Applications*, 310(1-3):167–181, 2000.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.