FAST SAMPLING OF OPTIMAL PARAMETERS FOR ML MODELS

Natalia Uspenskaia
uspen.nat@gmail.com

February 2, 2020

ABSTRACT

Model's true parameters can be viewed as a vector that came from distribution $\mathcal{N}(\theta_{MLE}, \frac{1}{N}\mathcal{I}(\theta_{MLE})^{-1})$, where N 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, $\theta_{MLE} + \frac{1}{\sqrt{N}}\mathcal{I}(\theta_{MLE})^{-1/2}z$ has the desired distribution. To drastically reduce the complexity of computation of $\mathcal{I}(\theta_{MLE})^{-1/2}z$ Lanczos method and Pearlmutter trick are used.

1 Introduction

Let $\boldsymbol{y} = (y_1; y_2; \dots y_N)$ be a vector of i.i.d. random variables from one of a family of distributions on $\mathbb{R}$ indexed by a n-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

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

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:

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

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 $N \to \infty$, true parameters of the model can be viewed as a vector that came from distribution $\mathcal{N}(\theta_{MLE}, \frac{1}{N}\mathcal{I}(\theta_{MLE})^{-1})$, 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

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

then $\hat{\theta} \sim \mathcal{N}(\theta_{MLE}, \frac{1}{N}\mathcal{I}(\theta_{MLE})^{-1})$. 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/uspenat/optimal-sampling`

Although Pearlmutter trick suggests using both forward and backward mode differentiation, PyTorch only supports the latter. Calculation of $\nabla^2 F z$ 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.

This implementation consists of seven Python scripts and supports CUDA acceleration.

3.1 lanczos_method.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.

slq.py from [4] was taken as the basis for this script .

3.2 lanczos_hvp.py

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

3.3 hvp_operator.py

In this module a linear operator ModelHessianOperator(Operator) is defined to compute the hessian-vector product for a given pytorch model using subsampled data. The idea of hvp_operator_upd.py was taken from the [5].

3.4 sampling.py

The key functions in sampling.py are generate_weights and set_model_parameters. The first function generates parameters for trained ML model from the distribution described in 2. The set_model_parameters function sets this sampled parameters in the model.

3.5 CNNs_complexity

In this module create_model(a, b, c) function builds CNNs with a different number of parameters depending on the input parameters a, b, c which allows us to measure the running time of the algorithm. This model uses MNIST dataset. After that the desired tridiagonal matrix T and matrix V with orthogonal columns are calculated via the lanczos function from lanczos_method.py.

3.6 LogReg_py

Logistic Regression is considered for binary classification problem for sklearn.datasets.make_classification. The sample_scores function is used to generate parameters from the desired distribution and set them to the model. For model with sampled parameters accuracy and loss function are calculated for the test dataset.

3.7 ConvNN_MNIST_bin.py

Convolutional neural network is considered for binary classification problem for MNIST dataset. The same as for LogReg_py module for CNN with sampled parameters accuracy and loss function are calculated for the test dataset.

4 Experiments

First, MNIST dataset was taken to train CNNs. The dependence of time complexity by number of model parameters is shown in the Figure 1. Then two models were considered with sampled parameters from the distribution of optimal parameters for each model. One can see in Figure 2 and Figure 3 that accuracy is concentrated near accuracy corresponded to the model with trained parameters (MLE). The same tendency is observed for loss function.

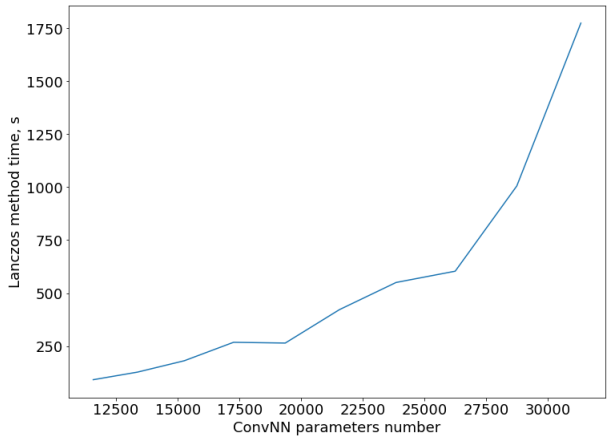
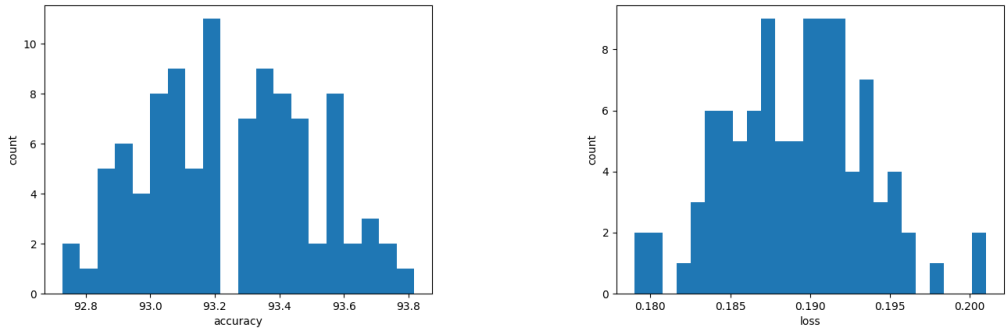
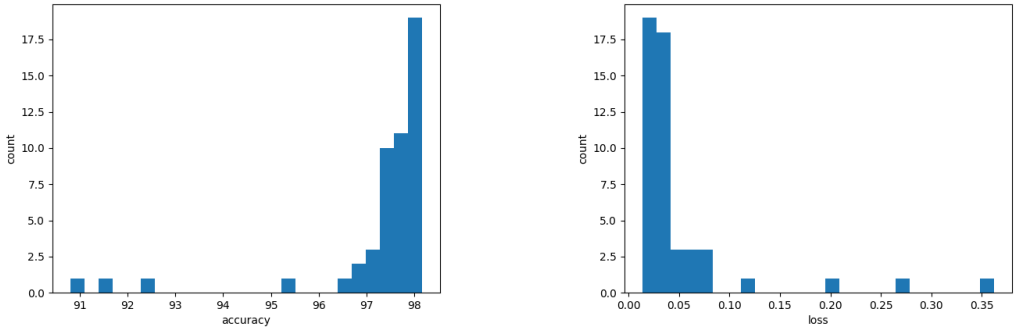


Figure 1: Complexity depending on the number of CNN parameters



(a) Accuracy (b) Loss

Figure 2: Accuracy and loss for logistic regression with 100 sampled parameters computed using 21 lanczos vectors of 42



(a) Accuracy (b) Loss

Figure 3: Accuracy and loss for convolutional neural network with 100 sampled parameters computed using 311 lanczos vectors of 4666

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.