CRONE 2018
Conference Proceedings

A Comparison of Stochastic Gradient MCMC using Multi-Core and GPU Architectures

Sergio Hernández[1*] **and José Valdés**[1]

***For correspondence:**
shernandez@ucm.cl (SH)

[1]Laboratorio de Procesamiento de Información Geoespacial. Universidad Católica del Maule. Chile; [2]Centro de Innovación en Ingeniería Aplicada.

Abstract Deep learning models are traditionally used in big data scenarios. When there is not enough training data to fit a large model, transfer learning repurpose the learned features from an existing model and re-train the lower layers for the new task. Bayesian inference techniques can be used to capture the uncertainty of the new model but it comes with a high computational cost. In this paper, we compare the run time performance of an Stochastic Gradient Markov Chain Monte Carlo method using different architectures. As opposed to the widely usage of GPUs for deep learning, we found significant advantages from using modern CPU architectures.

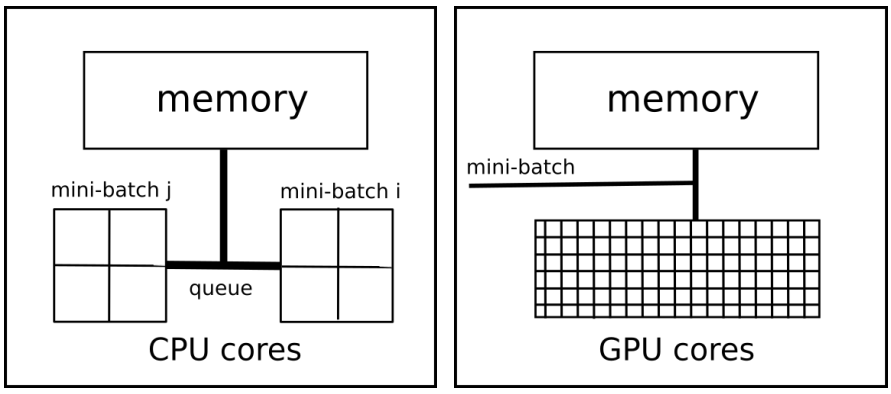
Introduction

As the amount of stored data is increased. analytical techniques for processing large-scale datasets are also required. In the other hand, big datasets are not only described by the number of observations but also through its dimensionality (number of features). Therefore. modern scalable Bayesian inference techniques must be able to improve performance on a single machine (scale up) as well as distributing performance across several machines or CPU cores (scale out) [1].

In particular. Bayesian inference is used for computing posterior distributions of the model parameters. In order to obtain such posterior estimates. Markov Chain Monte Carlo (MCMC) methods approximate intractable integrals as an expectation using a finite number of samples. However, this expectation requires the full dataset in order to guarantee convergence. Splitting a big dataset into smaller batches and running parallel chains is a simple alternative for scaling out Bayesian inference. Nevertheless, each chain will have its own posterior approximation and there is no standard method for recombining the results into a single posterior distribution.

The Stochastic Gradient Langevin Monte Carlo (SGLD) algorithm is an approximation technique that fully exploits data subsets [7]. The method replaces gradients and likelihood evaluations required by standard MCMC by stochastic gradients based on data subsets (mini-batches). Some implementations can be found in software libraries such as Edward [6], ZhuSuan [5] and the SGMCMC package [2]. All of these implementations are based on TensorFlow[1] as a backend and therefore inherit the scalability of GPU computation. However, there is little research done on the scalability of SGLD using modern CPU architectures. Therefore, in this paper we present a novel comparison of SGLD implementations on GPUs and CPUs.

[1]https://www.tensorflow.org/



(a) CPU implementation. **(b)** GPU implementation.

Figure 1. SG-MCMC implementation on CPU and GPU. In CPU, a data queue feeds multiple Markov chains. In GPU, a single mini-batch is fed to a Markov chain that updates model parameters.

Stochastic Gradient MCMC

Stochastic Gradient Descent (SGD) is an optimization technique traditionally used for out-of-core training deep learning models. For example, if we represent a dataset $\mathbf{D} = \{\mathbf{d_1}, \dots, \mathbf{d_N}\}$ as a set of tuples $\mathbf{d} = (\mathbf{x}, y)$ that contains features $\mathbf{x} \in \mathbb{R}^D$ and the labels $y = \{1, \dots, K\}$. An stochastic optimization procedure iterates over mini-batches of size $B \ll N$ and compute a point estimate $\theta^* = \text{argmax}\, p(\mathbf{D}|\theta)$.

Deep learning models such as convolutional neural networks may exploit model parallelism for fully connected layers and data parallelism for convolutional layers. The former training technique applies parallelism across the model dimension (number of features). The latter technique applies parallelism across the data dimension (number of examples). In a model parallel setting, different workers are trained on the same batch and they recombine results as soon as possible. In the other hand, in a data parallel setting, each worker is trained on different batches of data.

In the Bayesian framework, we consider the unknown parameter θ as a random variable. The Stochastic Gradient Langevin Monte Carlo (SGLD) algorithm uses an stochastic gradient approximation to generate samples from the posterior distribution $p(\theta|\mathbf{D})$. The SGLD algorithm generate proposals using:

$$\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{B}\sum_i^B \nabla \log p(\mathbf{d_i}|\theta_t)\right) + \eta_t \tag{1}$$

where $\eta_t \sim \mathcal{N}(0, \epsilon_t)$ and $\epsilon_t \mapsto 0$ is a time-decaying step size.

It is important to notice that using SGLD on a fully connected layer requires dense matrix multiplications for computing the log-likelihood and the log-prior. Therefore, we present two implementations using two different architectures.

GPU SG-MCMC

Graphic Processing Units (GPus) are many-core processors that offer massively parallel computation. GPUs are well suited for data parallel operations such as computing the gradient in convolutional neural networks and Monte Carlo methods [3]. In order to implement the SGLD algorithm we use CuPy, a Python library for GPU computation[2]. The library includes optimized GPU array manipulation and linear algebra routines, which are compatible with other CPU array manipulation routines such as Numpy.

[2] https://cupy.chainer.org/

CPU SG-MCMC

Modern CPUs are equipped with multiple cores which are well suited to single instruction multiple data operations. Python is built around a Global Interpreter Lock, which prevents the execution of multiple threads at once. Nevertheless, parallel Markov can be spawn using all available cores. Moreover, data parallelism can be achieved by sharing a data queue that feeds each process with a different data mini-batch. Figure 1 depicts the proposed GPU and CPU implementations of the SGLD algorithm.

Experimental Results

In this section, the goal is to demonstrate performance and scalability of the SGLD algorithm. The GPU and CPU implementations are tested using randomly generated multi-class classification problems. Different combinations on the number of features D and number of instances N were tested. The number of classes was set to $K = 3$, the number of epochs was set to $E = 2 \times 10^4$ and the size of the mini-batches was set to $B = 50$ for all examples.

Firstly, a serial version of the SGD algorithm is tested on the CPU and compared with a parallel implementation on the GPU. All experiments are run on a Server with an Intel Xeon E5-2620 CPU with 12 cores using a single fully connected softmax layer (which is widely used in transfer learning). The server is also equipped with an NVIDIA TITAN X GPU and both implementations were developed using Python 3.4.9. Table 1 shows the run time for the different values of D and N. According to [4], convolutional layers contain 90% of the computation and 5% of the parameters, while fully connected layers contain 95% of the parameters and $5\% - 10\%$ of the computation. Consistently, since there is a fully connected layer operating on a small batch of data, there is no speedup and the GPU implementation is about 10x slower than the CPU implementation.

D	N	CPU Time [s]	GPU Time [s]
10	1000	11.98	124.85
10	10000	119.90	1245.98
10	50000	594.71	6186.05
10	100000	1195.62	12261.71
50	1000	13.67	123.13
50	10000	135.96	1230.19
50	50000	689.89	6137.79
50	100000	1487.26	12240.06
100	1000	15.71	123.53
100	10000	156.94	1231.12
100	50000	806.78	6176.91
100	100000	1770.83	12394.13

Table 1. Run time comparison of CPU and GPU implementations of SGD for a softmax regression problem

Secondly, the run time behavior of a multi-core CPU implementation of SGLD is shown in Table 2. Similar to SGD, run time is scales with the number of data points N and features D. This is not the case of the GPU implementation of SGD, where the number of features seems not to affect the run time due to the massive parallel processing capabilities of the GPU.

References

[1] **Angelino E**, Johnson MJ, Adams RP. Patterns of Scalable Bayesian Inference. Foundations and Trends in Machine Learning. 2016; 9(2-3):119–247.

[2] **Baker J**, Fearnhead P, Fox EB, Nemeth C. sgmcmc: An R package for stochastic gradient Markov chain Monte Carlo. arXiv preprint arXiv:171000578. 2017; .

D	N	CPU Time [s]
10	1000	24.75
10	10000	245.22
10	50000	1186.81
10	100000	2380.18
50	1000	25.70
50	10000	246.64
50	50000	1230.02
50	100000	2496.15
100	1000	30.95
100	10000	279.28
100	50000	1450.57
100	100000	2796.20

Table 2. Run time comparison of CPU implementation of SGLD for a softmax regression problem

[3] **Lee A**, Yau C, Giles MB, Doucet A, Holmes CC. On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. Journal of computational and graphical statistics. 2010; 19(4):769–789.

[4] **Li X**, Zhang G, Li K, Zheng W. Chapter 4 - Deep Learning and Its Parallelization. In: Buyya R, Calheiros RN, Dastjerdi AV, editors. *Big Data* Morgan Kaufmann; 2016.p. 95 – 118.

[5] **Shi J**, Chen J, Zhu J, Sun S, Luo Y, Gu Y, Zhou Y. ZhuSuan: A Library for Bayesian Deep Learning. arXiv preprint arXiv:170905870. 2017; .

[6] **Tran D**, Hoffman MW, Moore D, Suter C, Vasudevan S, Radul A. Simple, Distributed, and Accelerated Probabilistic Programming. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. *Advances in Neural Information Processing Systems 31* Curran Associates, Inc.; 2018.p. 7598–7609.

[7] **Welling M**, Teh YW. Bayesian learning via stochastic gradient Langevin dynamics. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*; 2011. p. 681–688.