International Journal on Artificial Intelligence Tools
Vol. 29, No. 5 (2020) 2050010 (10 pages)

DOI: 10.1142/S0218213020500104

World Scientific
www.worldscientific.com

Variance Counterbalancing for Stochastic Large-scale Learning

Pola Lydia Lagari
School of Nuclear Engineering, Purdue University
West Lafayette, IN 47907, USA
plagari@purdue.edu

Lefteri H. Tsoukalas
School of Nuclear Engineering, Purdue University
West Lafayette, IN 47907, USA
tsoukala@purdue.edu

Isaac E. Lagaris
Department of Computer Science and Engineering, University of Ioannina
Ioannina 45500, Greece
lagaris@cs.uoi.gr

Received 18 March 2020
Accepted 20 April 2020
Published 21 August 2020

Stochastic Gradient Descent (SGD) is perhaps the most frequently used method for large scale training. A common example is training a neural network over a large data set, which amounts to minimizing the corresponding mean squared error (MSE). Since the convergence of SGD is rather slow, acceleration techniques based on the notion of "Mini-Batches" have been developed. All of them however, mimicking SGD, impose diminishing step-sizes as a means to inhibit large variations in the MSE objective.

In this article, we introduce random sets of mini-batches instead of individual mini-batches. We employ an objective function that minimizes the average MSE and its variance over these sets, eliminating so the need for the systematic step size reduction. This approach permits the use of state-of-the-art optimization methods, far more efficient than the gradient descent, and yields a significant performance enhancement.

Keywords: Large-scale training; neural networks; stochastic gradient descent; variance reduction.

1. Introduction

The stochastic approximation method of Robbins and Monro,[1] laid the foundation for the development of SGD, which together with its variants are the preferred approaches for large-scale training. In this article, we examine the problem of training a neural network over a large data set, which nowadays is a frequent task in a host of

machine-learning applications. Several modifications to the original SGD algorithm, targeting to accelerate convergence, have appeared in the literature recently; see for example ADAM,[2,3] AMSGrad,[4] RADAM[5] and ADABound and AMSBound[6] among others. Since the deterministic gradient descent is lagging behind in performance compared to Quasi-Newton methods, the question of the possibility to extend the latter appropriately for the stochastic case, naturally arises. In this direction several techniques have emerged,[7–9] which use a quasi-Newton direction in place of the negative gradient, in an attempt to hasten convergence. SGD-based techniques have been recently detailed and analyzed in an extensive review article by Bottou *et al.*[10] All of these SGD–alternatives, operate on randomly selected subsets of the training set, called "Mini–Batches". Each Mini–Batch contains a relatively small number of training points and the associated algorithms, in order to avoid over-training and at the same time keep the MSE variance low, resort to using steps of diminishing size, retarding therefore the convergence.

In Section 2 we present the rationale and the framework of the proposed approach, along with justification for the algorithmic choices. The Variance Counterbalancing (VCB) algorithm is detailed in Section 2.1. In Section 3, implementation details are given about the platforms used and the coding language, while in Section 3.1, a brief description of the functionally weighted neural network used, is laid out. Numerical experiments are described in Section 4, where VCB is compared to the full-batch BFGS approach and with the widely used ADAM[3] method. Finally in Section 5, conclusions are drawn along with some remarks and suggestions for further investigation.

2. Variance Counterbalancing

Let S be a training set

$$S = \{(x_1, y_1), \ldots, (x_N, y_N)\} \quad \text{with } x \in R^n,\ y \in R \tag{1}$$

and let $N(x, w)$ be a neural network with weights w, that is to be trained over S.

Training is performed by minimizing w.r.t. w the relevant MSE, given by

$$E(w) = \frac{1}{N} \sum_{i=1}^{N} \left[N(x_i, w) - y_i\right]^2 . \tag{2}$$

Let also $S_i^{(M)}$ be a subset of S (a mini-batch), containing M points selected at random.

$$S_i^{(M)} = \{(x_{i_1}, y_{i_1}), \ldots, (x_{i_M}, y_{i_M})\} \subset S \tag{3}$$

with an associated MSE given by

$$E_i(w) = \frac{1}{M} \sum_{j=1}^{M} \left[N(x_{i_j}, w) - y_{i_j}\right]^2 . \tag{4}$$

Let the optimal weights be denoted by w^*, corresponding to a "proper" MSE minimum $E(w^*)$. The characterization "proper", refers to certain properties that should be satisfied at w^*. MSE objectives, in most cases, possess many local minima. Not every minimum is a suitable choice as far as learning is concerned. A proper minimum, that has the potential to generalize well, is one that maintains the fluctuations of the individual squared errors as small as possible. This may be expressed by the relation

$$E(w^*) \approx E_i(w^*), \ \forall\, i = 1, 2, \ldots \tag{5}$$

which is a quite plausible hypothesis, since otherwise either the training would be poor, or the neural network would be of limited capacity. If the requirement expressed in (5) could be imposed as a constraint to the optimization process, then the optimal w^* would be proper.

So if we consider a random set of K mini-batches, one may construct a suitable objective

$$F(w, \lambda) = E(w) + \lambda \frac{1}{K} \sum_{i=1}^{K} \left[E_i(w) - E(w)\right]^2, \quad \text{with } \lambda > 0 \tag{6}$$

that counterbalances $E(w)$ with a variance term to suppress the fluctuations. To avoid using the costly $E(w)$, we replace it in Eq. (6), by an average over the K mini-batches.

$$\bar{E}(w) = \frac{1}{K} \sum_{i=1}^{K} E_i(w)\,. \tag{7}$$

The new objective then becomes

$$\bar{F}(w, \lambda) = \bar{E}(w) + \lambda \frac{1}{K} \sum_{i=1}^{K} \left[E_i(w) - \bar{E}(w)\right]^2\,. \tag{8}$$

Apart from λ, the second term in Eq. (8) is the variance of $E_i(w)$

$$\sigma^2(w) \equiv \frac{1}{K} \sum_{i=1}^{K} \left[E_i(w) - \bar{E}(w)\right]^2\,. \tag{9}$$

The training proceeds in epochs, where each epoch is characterized by the random choice of the mini-batch set $\{S_i^{(M)}, i = 1, \ldots, K\}$. Note that the value of the penalty λ in each epoch, is crucial for the successful application of the VCB algorithm. Several approaches may apply, for instance the traditional one, where the penalty starts from a low value and is gradually enhanced, or inversely, starting from a high value and subsequently being reduced. The second approach aims to guide the training into the small variance region of the weight space from the very start. We have followed a "retarded penalty" approach analyzed in Appendix B.

2.1. *The VCB-algorithm*

To apply the above ideas to the problem of minimizing the MSE, one may proceed as follows

(1) Input:
S, the training set, containing N points (x, y). See Eq. (1).
K, the number of mini-batches $S_i^{(M)}$, each containing $M \geq 2$ points.
$\boldsymbol{\alpha}$, a lower bound for the penalty ($\alpha \approx 0.1$).
$\boldsymbol{\beta}$, an upper bound for the penalty ($\beta \approx 70$).
$\boldsymbol{w^{(0)}}$, the initial values for the weights.

(2) Set the iteration counter (epoch) $k = 0$, $w^* = w^{(0)}$.
Calculate $E(w^{(0)})$, the full MSE using all the training examples from Eq. (2).
Set $E^* = E(w^{(0)})$.

(3) Pick at random from S, a set of K mini-batches $S_{i=1,K}^{(M)}$, each containing M points.
A mini-batch is constructed by picking at random M out of N points from S, via an efficient algorithm listed in (Appendix A).

(4) Set the penalty value according to the "retarded penalty" policy described in (Appendix B)

$$\lambda = \min\left(\max\left(\frac{E^* - \bar{E}(w^{(k)})}{\sigma^2(w^{(k)})}, \alpha\right), \beta\right) . \tag{10}$$

(5) Minimize the objective $\bar{F}(w, \lambda)$ of Eq. (8), starting at $w = w^{(k)}$, and obtain $w^{(k+1)}$.

(6) Calculate $E(w^{(k+1)})$, i.e. the full MSE.
If $E(w^{(k+1)}) < E(w^*)$, Update: $w^* = w^{(k+1)}$ and $E^* = E(w^{(k+1)})$.

(7) Increment $k \to k + 1$, and repeat from step (3) until a stopping criterion is satisfied.

(8) Output:
The proper w^* and the corresponding MSE, $E^* = E(w^*)$.

Several stopping criteria may be used.

(i) Stop after a prescribed number of iterations.
This number could be determined, for example, by the available CPU-time budget.

(ii) Stop if for a preset number of consecutive iterations, progress is below a threshold.
This will prevent unnecessary iterations.

(iii) Stop if any of the above rules instructs so.
A combination in the sense of "whatever comes first", stopping rule.[a]

[a]This is the rule adopted in our numerical experiments.

Notice that following the minimization session in step (5), an evaluation of the full MSE takes place in step (6). This might at first seem to weigh on the CPU-time. However, and depending on the termination criteria, it may save iterations. Note also that the most expensive part, is not the full MSE itself but rather its gradient, which however is not being evaluated. Anyway, a rough estimate is that it adds, depending on the mini-batch number and size, approximately a 5–25% burden to the training time. In addition, if the full MSE is not evaluated, the resulting weights, that would be unconditionally updated, will not correspond to the least attained MSE value, and hence may be way too far from optimal.

3. Implementation

We have employed the "Merlin-3.0" optimization environment,[11] along with its programming language "MCL",[12] to implement the proposed VCB algorithm.

Merlin is a command driven environment, offering a host of robust optimization routines for bound constrained problems, and several auxiliary facilities. Merlin can be programmed at a high level via MCL (Merlin Control Language) to automate optimization strategies. Both Merlin and MCL are written in Fortran-77.

The objective $\bar{F}(w, \lambda)$ is programmed as a function, and its gradient as a subroutine, both in Fortran, in the format required by Merlin. The strategy followed in the VCB-Algorithm, is programmed in MCL.

Among the various types of Artificial Neural Networks (ANNs), we have chosen to use the recent "Functionally Weighted Neural Network" (FWNN),[13] a network with infinite number of nodes. Certainly, any type of ANN (MLP, RBF, etc.) could have been used instead as well.

3.1. *Brief description of FWNN*

The FWNN is given by the expression

$$N_{FW}(x, w) = \int_{-1}^{+1} \frac{ds}{\sqrt{1-s^2}} a(s) \exp\left(-\frac{||x-\mu(s)||^2}{2\sigma(s)^2}\right), \quad \text{with } x \text{ and } \mu(s) \in R^n \tag{11}$$

and

$$a(s) = \sum_{j=0}^{L_a} a_j s^j, \qquad \mu_i(s) = \sum_{j=0}^{L_\mu} \mu_{ij} s^j \quad (\forall\, i = 1, \ldots, n), \qquad \sigma(s) = \sum_{j=0}^{L_\sigma} \sigma_j s^j \tag{12}$$

where the weights are denoted collectively by

$$w = \left(\{a_j\}_{j=0}^{L_a},\ \{\mu_{ij}\}_{i=1,j=0}^{n,L_\mu},\ \{\sigma_j\}_{j=0}^{L_\sigma} \right).$$

The number of weights are given by the expression

$$L = (1 + L_a) + n(1 + L_\mu) + (1 + L_\sigma) = n + 2 + L_a + nL_\mu + L_\sigma\,.$$

The integral in (11), may be efficiently and accurately evaluated by the Gauss-Chebyshev quadrature.[14]

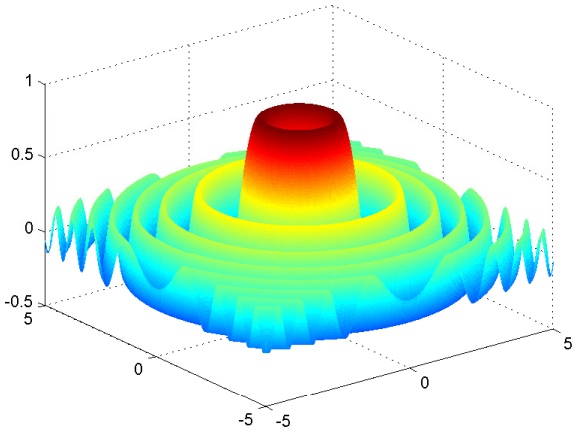

Fig. 1. The "Mexican Hat" function: $f(x_1, x_2) = \frac{\sin(x_1^2+x_2^2)}{\sqrt{x_1^2+x_2^2}}$.

4. Numerical Experiments

We have performed experiments with the two-dimensional function known as the "Mexican Hat", a 3-d plot of which is depicted in Fig. 1. We have considered a 2-dimensional grid of (200×200), i.e. a total of 40×10^3 points, for $(x_1, x_2) \in [-5, 5] \otimes [-5, 5]$. This training set of 40×10^3 points, is a large set, but not a very large one. The reason it was chosen is that the training of the MSE, $E(w)$ in Eq. (2), over the whole training set, is feasible. This enables a comparison of the standard approach to the method proposed in the present article.

Let τ_f and τ_g be the CPU-times required for a single evaluation of the neural network $N_{FW}(x, w)$ and of its gradient $\nabla_w N_{FW}(x, w)$ respectively. Then the required CPU-times for evaluating $\bar{F}(w, \lambda)$ in Eq. (8) and its gradient $\nabla_w \bar{F}(w, \lambda)$, are $KM\tau_f$, and $KM(\tau_f + \tau_g)$. If by N_f and N_g we denote the number of evaluations of $\bar{F}(w, \lambda)$ and $\nabla_w \bar{F}(w, \lambda)$ performed during the training, then the total training CPU-time is estimated by

$$T(K, M, N_f, N_g) = KM \left[N_f \tau_f + N_g \left(\tau_g + \tau_f \right) \right] . \tag{13}$$

Note that the MSE training over the whole training set, corresponds to setting $K = 1, M = N$. The FWNN used is described by $L_a = 8, L_\mu = 5, L_\sigma = 4$, with a total of $L = 26$ parameters. For this network in our computer system it turned out that $\tau_g \approx 2\tau_f$.

4.1. *Comparison with full-batch BFGS*

We report in Table 1 the averages over 20 experiments with different random initialization. The first row corresponds to the full-batch approach using the BFGS method with line search, while the rest list the results with the VCB algorithm. The last column (Percent), denotes the percentage of time spent for evaluating the full MSE in each epoch.

Table 1. Mini-batch set structure, and corresponding results.

K	M	N_f	N_g	MSE	T (in $10^6\tau_f$)	Percent
1	40 000	1252	1165	3.45×10^{-2}	190	×
20	50	4333	3873	3.26×10^{-2}	21	24
25	40	4479	3976	3.08×10^{-2}	22	25
40	25	4382	3910	3.28×10^{-2}	21	24
50	20	4607	4126	3.26×10^{-2}	22	24
50	50	3591	3271	3.34×10^{-2}	36	7

The FWNN weights were bounded in the range $[-100, 100]$. Note also that regularization techniques have not been used. For the optimization we have employed the Quasi-Newton approach with BFGS[15,16] updates.

4.2. *Experiments with ADAM*

We have performed experiments with the ADAM method, since it is a reference method for stochastic optimization and therefore a comparison could be useful and interesting. ADAM calculates only the gradient of the MSE over a mini-batch, and unconditionally accepts the parameter updates. The path followed is not a monotonically descent one. There is no theoretically sound stopping rule to adopt, unless one evaluates from time to time either the MSE over the whole training set or its gradient. Hence there is an ambiguity regarding the termination criteria to be used in practice. In our experiments we have evaluated the full-MSE every 500 iterations, and stopped iterating when either $M \times N_g \geq L_t$ or MSE $\leq f_t$. We tried two different values for L_t. $L_t = 20 \times 10^6$ and $L_t = 30 \times 10^6$ for $M = 50$, and $L_t = 20 \times 10^6$ and $L_t = 40 \times 10^6$ for $M = 100, 150\&200$. The target value was set to $f_t = 3.3 \times 10^{-2}$, i.e. a little higher that the average attained by the VCB method. Note that the reported times in Table 2, do not contain the time spent for the periodic MSE evaluation, which is approximately equal to $N_g \times 10^{-4}$ in units of $10^6\tau_f$.

One may notice that for $M = 100$, the drop in the MSE value after an additional number of 163 067 iterations is 1.27×10^{-3}, showing that convergence is slow. The situation is similar for all values of M. For larger training sets, convergence is expected to be even slower. Inspecting Tables 1 and 2, notice that ADAM is roughly 1.6 to 3 times faster than the full-batch BFGS, and 2 to 6 times slower than VCB.

Table 2. ADAM mini-batch size, and corresponding results.

M	N_g	MSE	T (in $10^6\tau_f$)	N_g	MSE	T (in $10^6\tau_f$)
50	395 433	3.67×10^{-2}	59	568 767	3.63×10^{-2}	85
100	200 000	3.69×10^{-2}	60	363 067	3.56×10^{-2}	109
150	133 334	3.89×10^{-2}	60	253 422	3.63×10^{-2}	114
200	100 000	4.09×10^{-2}	60	196 167	3.63×10^{-2}	118

5. Remarks and Conclusions

In this article, a novel alternative to SGD-based methods is presented entitled "Variance Counterbalancing" a code name that points to its cornerstone concept. The numerical investigation for demonstrating its efficiency, although limited, clearly illustrates the capability of the approach to deal with large scale training problems. When compared to the full-batch approach, we have observed significant CPU-time reductions. Depending on the structure of the mini-batch set, VCB is from 5 to 10 times faster for the examined Mexican-hat data set. Moreover, the full-batch approach, since it lacks the stochastic element inherent in VCB, is trapped more often to undesirable local minima. Note also that on the average, VCB achieves lower MSE values as it can be verified by inspecting the MSE column of Table 1. The penalty term in Eq. (8), since it has a reducing effect on the variance, guides the optimization process to a minimum with promising generalization potential. Comparing to the ADAM method, we noticed an average 3–4 times acceleration and the extra advantage that unambiguous stopping rules may apply. We recommend that further tests, involving large multidimensional data sets preferably corresponding to real-world applications, should be carried out in order to definitively confirm and establish the practical utility of the proposed VCB approach.

Acknowledgments

This work has been supported by a donation to AI Systems Lab (AISL) by GS Gives, the Consortium for Nonproliferation Enabling Capabilities (CNEC) and the Office of Naval Research (ONR) under Grant No. N00014-18-1-2278.

References

1. H. Robbins and S. Monro, A stochastic approximation method, *Ann. Math. Statist.* **22** (1951) 400–407.
2. M. Li, T. Zhang, Y. Chen and A. J. Smola, Efficient mini-batch training for stochastic optimization, in *Proc. of the 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (*KDD '14*) (ACM, New York, NY, USA, 2014), pp. 661–670.
3. D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2014), published as a conference paper at the *3rd International Conference for Learning Representations* (San Diego, 2015).
4. S. J. Reddi, S. Kale and S. Kumar, On the convergence of Adam and beyond, in *Int. Conf. on Learning Representations* (2018).
5. L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao and J. Han, On the variance of the adaptive learning rate and beyond, *arXiv preprint 1908.03265* (2019).
6. L. Luo, Y. Xiong, Y. Liu and X. Sun, Adaptive gradient methods with dynamic bound of learning rate, in *Int. Conf. on Learning Representations* (2019).
7. P. Sunehag, J. Trumpf, S. Vishwanathan and N. Schraudolph, Variable metric stochastic approximation theory, in *Proc. of the Twelfth Int. Conf. on Artificial Intelligence and Statistics*, eds. D. van Dyk and M. Welling, in *Proc. of Machine Learning Research*, Vol. 5 (PMLR, 16–18 April 2009), pp. 560–566.

8. R. M. Gower, D. Goldfarb and P. Richtárik, Stochastic block BFGS: Squeezing more curvature out of data, in *Proc. of the 33rd Int. Conf. on Machine Learning* (*ICML'16*), Vol. 48 (JMLR.org, 2016), pp. 1869–1878.
9. R. H. Byrd, S. L. Hansen, J. Nocedal and Y. Singer, A stochastic quasi-newton method for large-scale optimization, *arXiv e-prints*, arXiv:1401.7020 (2014).
10. L. Bottou, F. Curtis and J. Nocedal, Optimization methods for large-scale machine learning, *SIAM Review* **60**(2) (2018) 223–311.
11. D. G. Papageorgiou, I. N. Demetropoulos and I. E. Lagaris, Merlin-3.0 a multidimensional optimization environment, *Computer Physics Communications* **109** (1998) 227–249.
12. D. G. Papageorgiou, I. N. Demetropoulos and I. E. Lagaris, The Merlin control language for strategic optimization, *Computer Physics Communications* **109** (1998) 250–275.
13. K. Blekas and I. E. Lagaris, Artificial neural networks with an infinite number of nodes, *Journal of Physics*: *Conference Series* **915** (2017) 012006.
14. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions* (Dover, 1972).
15. R. Fletcher, A new approach to variable metric algorithms, *The Computer Journal* **13** (1970) 317–322.
16. M. J. D. Powell, A tolerant algorithm for linearly constrained optimization calculations, *Mathematical Programming* **45** (1989) 547–566.
17. D. E. Knuth, *The Art of Computer Programming*, Vol. 2 (3rd edn.): Seminumerical Algorithms (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997).
18. T. G. Jones, A note on sampling a tape-file, *Commun. ACM* **5** (1962) 343.
19. C. T. Fan, M. E. Muller and I. Rezucha, Development of sampling plans by using sequential (item by item) selection techniques and digital computers, *Journal of the American Statistical Association* **57**(298) (1962) 387–402.

Appendix A. Algorithm S

The problem

> Pick $k \leq n$ integers: $l_{j=1,k}$ at random, from the set $\{1, 2, \ldots, n\}$,

is being solved by "Algorithm S" described in Ref. 17 (p. 142), originally developed in Refs. 18 and 19, and listed below for convenience.

(1) Input: k, n
    Output: $l_j, j = 1, \ldots k$
(2) Initialize: $i = 0$, $m = 0$
(3) $x = \xi | \xi$ is a uniform random number in $(0, 1)$
(4) if $(n - i)x < k - m$, then
        $m \leftarrow m + 1, i \leftarrow i + 1$
        $l_m \leftarrow i$
    else
        $i \leftarrow i + 1$
    endif
(5) if $m = k$, Stop
    Repeat from step (3)

Appendix B. The Retarded Penalty Policy

At the end of kth epoch, the weight vector $w^{(k)}$, the mean $\bar{E}(w^{(k)})$, and the current best value of the full MSE E^* are known. The penalty for the next epoch is obtained by imposing the retarded requirement: $\bar{F}(w^{(k)}, \lambda_{k+1}) = E^*$. Using Eq. (8) and forcing $\lambda \in [\alpha, \beta]$

$$\lambda_{k+1} = \min\left(\max\left(\frac{E^* - \bar{E}(w^{(k)})}{\sigma^2(w^{(k)})}, \alpha\right), \beta\right) . \tag{B.1}$$