Neurocomputing 186 (2016) 74–89

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An optimized second order stochastic learning algorithm for neural network training

Shan Sung Liew [a,*], Mohamed Khalil-Hani [a], Rabia Bakhteri [b]

[a] VeCAD Research Laboratory, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
[b] Machine Learning Developer Group, Sightline Innovation, #202, 435 Ellice Ave, Winnipeg, Canada, MB R3B 1Y6

ARTICLE INFO

Article history:
Received 11 October 2015
Received in revised form
14 December 2015
Accepted 21 December 2015
Communicated by Jun Yu
Available online 7 January 2016

Keywords:
Stochastic diagonal Levenberg–Marquardt
Fast convergence
Hyperparameter overfitting
Computational efficiency
Distributed machine learning
Convolutional neural network

ABSTRACT

This paper proposes an improved stochastic second order learning algorithm for supervised neural network training. The proposed algorithm, named bounded stochastic diagonal Levenberg–Marquardt (B-SDLM), utilizes both gradient and curvature information to achieve fast convergence while requiring only minimal computational overhead than the stochastic gradient descent (SGD) method. B-SDLM has only a single hyperparameter as opposed to most other learning algorithms that suffer from the hyperparameter overfitting problem due to having more hyperparameters to be tuned. Experiments using the multilayer perceptron (MLP) and convolutional neural network (CNN) models have shown that B-SDLM outperforms other learning algorithms with regard to the classification accuracies and computational efficiency (about 5.3% faster than SGD on the *mnist-rot-bg-img* database). It can classify all testing samples correctly on the face recognition case study based on AR Purdue database. In addition, experiments on handwritten digit classification case studies show that significant improvements of 19.6% on MNIST database and 17.5% on *mnist-rot-bg-img* database can be achieved in terms of the testing misclassification error rates (MCRs). The computationally expensive Hessian calculations are kept to a minimum by using just 0.05% of the training samples in its estimation or updating the learning rates once per two training epochs, while maintaining or even achieving lower testing MCRs. It is also shown that B-SDLM works well in the mini-batch learning mode, and we are able to achieve 3.32× performance speedup when deploying the proposed algorithm in a distributed learning environment with a quad-core processor.

1. Introduction

Machine learning is a form of artificial intelligence that possesses the ability to learn from data. This usually comprises two main components: a parameterized and learnable model (e.g. neural network (NN)), and its corresponding learning algorithm. The structure of the model is often designed to provide nonlinear behavior, which is essential in performing classification or prediction.

Regardless of how good the learning capacity of a model is, its learning performance is still highly dependent on the effectiveness of the learning algorithm. A learning algorithm defines how the model can make use of the underlying information within the data, and learns from its statistics. In general, there are three main types of learning: unsupervised, semi-supervised, and supervised. Unsupervised learning aims to learn from the unlabeled data, which is often based on the probability of occurrence for data patterns. Semi-supervised learning utilizes small subset of the unlabeled data to pre-train the model. This provides a good intuition of the hidden structure of data before proceeding to training using the labeled data. Supervised learning infers a function by learning from input–output pairs of training samples [1]. The training is guided by the ground truth labels to tune the parameters. This paper focuses on the development of supervised learning algorithm for NN models.

Most learning algorithms are based on iterative methods. Given a parameterized model represented by a function, a learning algorithm aims to find a set of parameters that lead to an optimum solution for the function. In this context, the function refers to an NN with an error or loss function, the parameters are its weights and biases, and the solution is the state where the NN is deemed to have learned well. This is done by finding the suitable step sizes for these parameters (weights), and taking steps iteratively towards the direction until it reaches a desired solution:

$$W_{t+1} = W_t - \Delta W \tag{1}$$

* Corresponding author. Tel.: +60 168588653.
E-mail addresses: ssliew2@live.utm.my (S.S. Liew), khalil@fke.utm.my (M. Khalil-Hani), rbakhteri@sightlineinnovation.com (R. Bakhteri).

http://dx.doi.org/10.1016/j.neucom.2015.12.076
0925-2312/

where W_t and W_{t+1} are the weights at the tth and $(t+1)$th iterations respectively, and ΔW represents the optimal step sizes to be taken. The main objective is to find the best ΔW such that it reaches the solution in shortest number of iterations. In NN models, ΔW is usually computed by finding the error gradients of a loss function with respect to the weights, which can be done using the back-propagation algorithm.

Gradient descent is a first order learning algorithm which is most widely used in the NN training. The algorithm is simple, but often suffers from very slow convergence. Many other learning algorithms have been proposed to improve the learning convergence rate by utilizing both gradient and curvature information, or through adaptive mechanisms based on the current training status. As a result, these algorithms are able to produce a better learning curve, but with the expense of additional computations, which can be prohibitive to compute for larger and deeper neural network models. This suggests the necessity to develop a learning algorithm that can converge fast while requiring minimal computational overhead.

An additional issue with most of these complex learning algorithms is that they have more hyperparameters than the conventional first order learning algorithms. The problem that can arise is hyperparameter overfitting [2], in which there are endless ways of configuring the learning algorithm, and this may end up selecting a combination of values that outperforms others purely by chance.

In this paper, we propose a second order learning algorithm, called the bounded stochastic diagonal Levenberg–Marquardt algorithm, B-SDLM for short. The proposed algorithm is an improved version of the original SDLM algorithm, which was first proposed by LeCun et al. as a fast and efficient method to train convolutional neural networks (CNNs) [3]. The key contributions of the proposed algorithm include:

1. B-SDLM encourages fast network convergence due to utilizations of both gradient and curvature information, while ensuring its learning stability through an additional boundary condition;
2. B-SDLM alleviates the problematic hyperparameter overfitting issue by having only a single hyperparameter to be tuned instead of many hyperparameters typically required in existing complex learning algorithms; and
3. compared to the conventional SGD algorithm, the computational overhead in B-SDLM is negligible, since it requires only a very small portion of training samples for Hessian estimation.

It should be noted here that, to our knowledge, this work is among the first attempts to run a stochastic second order learning algorithm in the mini-batch learning mode to realize parallel computation. Our experimental results also show the viability of executing a second order learning algorithm in a distributed machine learning environment to gain computation speedup.

The paper is organized as follows. Section 2 covers common learning algorithms for supervised neural network training. Section 3 presents the proposed algorithm. Section 4 describes the experimental design to verify the performance of the proposed algorithm. Section 5 presents the results and discussions on B-SDLM from various perspectives. The final section concludes the work and suggests possible future work.

2. Theoretical background: learning algorithms

Gradient descent (GD) is the most common first order optimization method to train NNs on supervised mode [4]. Also known as “steepest descent”, the idea in GD is to take steps proportional to the negative gradient of the loss function $E(W)$ at current point t to find the local minimum (possible solution) of the function:

$$W_{t+1} = W_t - \eta \frac{dE(W_t)}{dW_t} \tag{2}$$

where W_t and W_{t+1} are the weights at the tth and $(t+1)$th iterations respectively, $E(W_t)$ is the loss function, η is the learning rate which controls the steps to which a problem approaches its solution(s), and $\frac{dE(W_t)}{dW_t}$ is the error gradient that is computed through the back-propagation (BP) algorithm.

Conventional GD works in batch learning mode, hence the term batch GD (BGD). BGD works by summing the error gradients of all samples in training data. These gradients are then averaged to be used in updating the weights. BGD is simple and can be easily parallelized. However, its convergence towards an optimal solution is very slow. Certain important patterns of some training samples may not be well observed due to the gradient averaging. BGD can also be computationally intractable when dealing with large data.

Stochastic GD (SGD), on the other hand, performs weight update when each sample is presented to the NN. The learning curve may not be as smooth as in BGD due to the noisy updates, but it tends to reach convergence faster due to better chance of avoiding local minima [5]. However, for ill-conditioned problems (i.e. small errors near inputs can result in large errors near outputs), SGD can suffer from slow convergence near a local minimum. Nevertheless, SGD is, by far, the most common supervised learning algorithm used in NN training, and is applied in conjunction with the BP algorithm [6–8].

In NN training, the speed of learning is also highly affected by the value of the learning rate chosen. A learning rate dictates mainly the step sizes to be taken to reach an optimal solution. However, a fixed learning rate may not be suitable for all problems, since its value depends heavily on the condition of error surface and current search location of the weight space. Consequently, adaptive learning rate schemes have been proposed to find a suitable learning rate that accelerates the training speed. These schemes improve on the first order learning algorithms, and they can be categorized into two types: global and local adaptive algorithms.

2.1. Global adaptive algorithms

Global adaptive algorithms attempt to make use of the training state to adjust the global learning rate (a learning rate applied to all weight updates). For instance, the learning rate can be manually reduced after certain training iteration [9], but questions of when to reduce and to what value must be considered carefully. Another approach is to decay the learning rate over time based on a fixed schedule (annealing) [9]. The learning rate can also be decreased once the training process reaches a plateau, but with the expense of more hyperparameters.

All the aforementioned methods apply the same learning rate to all weights, which can be inappropriate. Each weight may require different step sizes to be optimally tuned. Table 1 (due to [9]) summarizes some of the global learning rate schedules: where η_0 is the initial global learning rate, η_t is the global learning rate at the tth iteration, M is usually set as total training samples, and c controls the magnitude.

Other algorithms utilize the difference of total errors between the current and previous training iterations to tune the learning rate. A notable example is the bold driver technique, where the learning rate is increased by a small proportion if the error decreases, or decreasing it and canceling the last weight changes whenever an increased error is observed [10,11]. However, it can only be applied in the batch learning mode and requires two additional hyperparameters.