Neurocomputing 74 (2011) 2430–2437

Contents lists available at ScienceDirect

Neurocomputing

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

GPU-accelerated and parallelized ELM ensembles for large-scale regression

Mark van Heeswijk [a,*], Yoan Miche [a,b], Erkki Oja [a], Amaury Lendasse [a]

[a] Aalto University School of Science and Technology, Department of Information and Computer Science, P.O. Box 15400, FI-00076 Aalto, Finland
[b] Gipsa-Lab, INPG, 961 rue de la Houille Blanche, F-38402 Grenoble Cedex, France

ARTICLE INFO

Available online 13 May 2011

Keywords:
ELM
Ensemble methods
GPU
Parallelization
High-performance computing

ABSTRACT

The paper presents an approach for performing regression on large data sets in reasonable time, using an ensemble of extreme learning machines (ELMs). The main purpose and contribution of this paper are to explore how the evaluation of this ensemble of ELMs can be accelerated in three distinct ways: (1) training and model structure selection of the individual ELMs are accelerated by performing these steps on the graphics processing unit (GPU), instead of the processor (CPU); (2) the training of ELM is performed in such a way that computed results can be reused in the model structure selection, making training plus model structure selection more efficient; (3) the modularity of the ensemble model is exploited and the process of model training and model structure selection is parallelized across multiple GPU and CPU cores, such that multiple models can be built at the same time. The experiments show that competitive performance is obtained on the regression tasks, and that the GPU-accelerated and parallelized ELM ensemble achieves attractive speedups over using a single CPU. Furthermore, the proposed approach is not limited to a specific type of ELM and can be employed for a large variety of ELMs.

1. Introduction

Due to advances in technology, the size and dimensionality of data sets used in machine learning tasks have grown very large and continue to grow by the day. For this reason, it is important to have efficient computational methods and algorithms that can be applied on very large data sets, such that it is still possible to complete the machine learning tasks in reasonable time.

Meanwhile, video cards' performances have been increasing more rapidly than typical desktop processors and they now provide large amounts of computational power—compared again with typical desktop processors [1].

With the introduction of NVidia CUDA [2] in 2007, it has become easier to use the GPU for general-purpose computation, since CUDA provides programming primitives that allow you to run your code on highly parallel GPUs without needing to explicitly rewrite the algorithm in terms of video card operations. Examples of successful applications of CUDA include examples from biotechnology, linear algebra [3], molecular dynamics simulations and machine learning [4]. Depending on the application, speedups of up to 300 times are possible by executing code on a single GPU instead of a typical CPU, and by using multiple GPUs it is possible to obtain even higher speedups. The introduction of CUDA has lead to the development of numerous libraries that use the GPU in order to accelerate their execution by several orders of magnitude. An overview of software and libraries using CUDA can be found on the CUDA zone web site [2].

In this work, one of these libraries is used, namely CULA [5], which was introduced in October 2009 and provides GPU-accelerated LAPACK functions (see [6] for the original LAPACK). Using this library the training and model structure selection of the models can be accelerated. The particular models used in this work are a type of feedforward neural network, called extreme learning machine (ELM) [7–10] (see [11–14] for recent developments based on ELM).

The ELM is well-suited for regression on large data sets, since it is relatively fast compared with other methods [11,15] and it has been shown to be a good approximator when it is trained with a large number of samples [16]. Even though ELMs are fast, there are several reasons to implement them on GPU and reduce their running time. First of all, because the ELMs are applied to large data sets the running time is still significant. Second, large numbers of neurons are often needed in large-scale regression problems. Finally, model structure selection needs to be performed (and thus multiple models with different structures need to be executed) in order to avoid under- or overfitting the data.

By combining multiple ELMs in an ensemble model, the test error can be greatly reduced [10,17,18]. In order to make it feasible to apply an ensemble of ELMs to regression on large data sets, in this paper various strategies are explored for reducing the

* Corresponding author.
E-mail address: mark.van.heeswijk@tkk.fi (M. van Heeswijk).

0925-2312/$ - see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.neucom.2010.11.034

computational time. First, the training and model structure selection of the ELMs is accelerated by performing these steps largely on GPU. Second, the training of the ELM is performed in such a way that values computed during training can be reused for very efficient model structure selection through leave-one-out cross-validation. Finally, the process of building the models is parallelized across multiple GPUs and CPU cores in order to further speed up the method.

Experiments are performed on two large regression data sets: the first one is the well-known Santa Fe Laser data set [19] for which the regression problem is based on a time series; the second one is the data set 3 from the ESTSP'08 competition [19], which is also a time series, but consists of a particularly large number of samples, and needs a large regressor [20,21].

Results of the experiments show competitive performance on the regression task, and validate our approach of using a GPU-accelerated and parallelized ensemble model of multiple ELMs: by adding more ELM models to the ensemble, the accuracy of the model can be improved; model training and structure selection of the individual ELM models can be effectively accelerated; and due to the modularity of the ensemble model, the process of building all models can be parallelized across multiple GPUs and CPU-cores. Therefore, the proposed approach is very suitable for application in large-scale regression tasks.

Although a particular type of ELM is used in this paper (i.e. an ELM with conventional additive nodes), the proposed approach is not limited to this specific type of ELM. Indeed, the proposed approach can be employed for ELMs with a much wider type of hidden nodes, which need not necessarily be 'neuron-alike' [22,16,12].

The organization of this paper is as follows. Section 2 discusses the models used in this work and how to select an appropriate model structure. Section 3 gives an overview of the whole algorithm. Specifically, how multiple individual models are combined into an ensemble model and what parts are currently accelerated using GPU. Section 4 shows the results of using this approach on the two mentioned large data sets. Finally, the results are discussed and an overview of the work in progress is given.

2. Extreme learning machine for large-scale regression

The problem of regression is about establishing a relationship between a set of output variables (continuous) $y_i \in \mathbb{R}, 1 \leq i \leq M$ (single-output here) and another set of input variables $\mathbf{x}_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$. In the regression cases studied in the experiments, the number of samples M is large: 10 000 for one case and 30 000 for the other.

2.1. Extreme learning machine (ELM)

The ELM algorithm is proposed by Huang et al. in [8] and uses single-layer feedforward neural networks (SLFN). The key idea of ELM is the random initialization of a SLFN weights. Below, the main concepts of ELM as presented in [8] are reviewed.

Consider a set of M distinct samples $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Then, a SLFN with N hidden neurons is modeled as the following sum:

$$\sum_{i=1}^{N} \beta_i f(\mathbf{w}_i \mathbf{x}_j + b_i), \quad j \in [1, M], \tag{1}$$

with f being the activation function, $\mathbf{w}_i$ the input weights to the ith neuron in the hidden layer, b_i the hidden layer biases and β_i the output weights.

In the case where the SLFN would perfectly approximate the data (meaning the error between the output $\hat{y}_i$ and the actual value y_i is zero), the relation is

$$\sum_{i=1}^{N} \beta_i f(\mathbf{w}_i \mathbf{x}_j + b_i) = y_j, j \in [1, M], \tag{2}$$

which can be written compactly as

$$\mathbf{H}\beta = \mathbf{Y}, \tag{3}$$

where $\mathbf{H}$ is the hidden layer output matrix defined as

$$\mathbf{H} = \begin{pmatrix} f(\mathbf{w}_1 \mathbf{x}_1 + b_1) & \cdots & f(\mathbf{w}_N \mathbf{x}_1 + b_N) \\ \vdots & \ddots & \vdots \\ f(\mathbf{w}_1 \mathbf{x}_M + b_1) & \cdots & f(\mathbf{w}_N \mathbf{x}_M + b_N) \end{pmatrix} \tag{4}$$

and $\beta = (\beta_1 \ldots \beta_N)^T$ and $\mathbf{Y} = (y_1 \ldots y_M)^T$.

With these notations, the theorem presented in [8] states that with randomly initialized input weights and biases for the SLFN, and under the condition that the activation function f is infinitely differentiable, the hidden layer output matrix can be determined and will provide an approximation of the target values as good as wished (non-zero) [8,16].

The output weights β can be computed from the hidden layer output matrix $\mathbf{H}$ and target values $\mathbf{Y}$ by using a Moore–Penrose generalized inverse of $\mathbf{H}$, denoted as $\mathbf{H}^\dagger$ [23]. Overall, the ELM algorithm is then:

Algorithm 1. ELM

Given a training set $(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$, an activation function $f : \mathbb{R} \mapsto \mathbb{R}$ and N the number of hidden nodes,
1: - Randomly assign input weights $\mathbf{w}_i$ and biases b_i, $i \in [1, N]$;
2: - Calculate the hidden layer output matrix $\mathbf{H}$;
3: - Calculate output weights matrix $\beta = \mathbf{H}^\dagger \mathbf{Y}$.

The proposed solution to the equation $\mathbf{H}\beta = \mathbf{Y}$ in the ELM algorithm, as $\beta = \mathbf{H}^\dagger \mathbf{Y}$ has three main properties making it a rather appealing solution:

1. It is one of the least-squares solutions to the mentioned equation, hence the minimum training error can be reached with this solution;
2. It is the solution with the smallest norm among the least-squares solutions;
3. The smallest norm solution among the least-squares solutions is unique and is $\beta = \mathbf{H}^\dagger \mathbf{Y}$.

Theoretical proofs and a more thorough presentation of the ELM algorithm are detailed in the original paper in which Huang et al. present the algorithm and its justifications [8]. Furthermore, as described in [22,16,12], the hidden nodes need not be 'neuron-alike'.

The only parameter of the ELM algorithm is the number of neurons N to use in the SLFN. The optimal value for N can be determined by performing model structure selection, using an information criterion like BIC, or through a cross-validation procedure.

2.2. Model structure selection by efficient LOO computation

Model structure selection enables one to determine an optimal number of neurons for the ELM model. This is done using some criterion which estimates the model generalization capabilities for varying numbers of neurons in the hidden layer. One such possibility is the classical Bayesian information criterion (BIC) [24,25], which is used in [17].