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].

In this paper a different method of performing the model structure selection is used. Namely, leave-one-out (LOO) cross-validation, which is a special case of k-fold cross-validation, where k is equal to the number of samples in the training set (i.e. $k=M$). In LOO cross-validation, the models are trained on M training sets, in each of which exactly one of the samples has been left out. The left-out sample is used for testing, and the final estimation of the generalization error is the mean of the M obtained errors. Due to the fact that maximum use is made of the training set, the LOO cross-validation gives a reliable estimate of the generalization error, which is important for performing good model structure selection.

The amount of computation for LOO cross-validation might seem excessive, but for linear models one can compute the LOO error E_{loo} without retraining the model M times by using PRESS statistics [26]. Since ELMs are essentially linear models of the outputs of the hidden layer, the PRESS approach can be applied here as well:

$$E_{loo} = \frac{1}{M}\sum_{i=1}^{M}\frac{y_i - \hat{y}_i}{1 - hat_{ii}}, \tag{5}$$

where y_i and $\hat{y}_i$ are respectively the ith training sample, and its approximation by the trained model, and hat_{ii} is the ith value on the diagonal of the HAT-matrix, which is the matrix which transforms $\mathbf{Y}$ into $\hat{\mathbf{Y}}$:

$$\hat{\mathbf{Y}} = \mathbf{H}\beta = \mathbf{H}\mathbf{H}^{\dagger}\mathbf{Y} = \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y} = \mathrm{HAT}\cdot\mathbf{Y}. \tag{6}$$

From the above equation, it can be seen that a large part of the HAT-matrix consists of $\mathbf{H}^{\dagger}$, the Moore–Penrose generalized inverse of the matrix $\mathbf{H}$. Therefore, combined training and model structure selection of the ELM can be optimized by using a method that explicitly computes $\mathbf{H}^{\dagger}$. The $\mathbf{H}^{\dagger}$ computed during training can then be reused in the computation of the LOO error.

Furthermore, since only the diagonal of the HAT-matrix is needed, it suffices to compute the row-wise dot-product between $\mathbf{H}$ and $\mathbf{H}^{\dagger T}$, and it is not needed to compute $\mathbf{H}\mathbf{H}^{\dagger}$ in full.

In summary, the algorithm for efficient training and LOO-based model structure selection of ELM then becomes:

Algorithm 2. Efficient ELM training and model structure selection.

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 $\aleph = \{n_1, n_2, \ldots, n_{max}\}$ defining set of possible numbers of hidden neurons.
1: Generate the weights for the largest ELM:
2: – Randomly generate input weights $\mathbf{w}_i$ and biases b_i, $i \in [1, n_{max}]$;
3: **for** all $n_j \in \aleph$ **do**
4: Train the ELM:
5: – Take the input weights and biases for the first n_j neurons;
6: – Calculate the hidden layer output matrix $\mathbf{H}$;
7: – Calculate $\mathbf{H}^{\dagger}$ by solving it from $(\mathbf{H}^T\mathbf{H})\mathbf{H}^{\dagger} = \mathbf{H}^T$;
8: – Calculate output weights matrix $\beta = \mathbf{H}^{\dagger}\mathbf{Y}$;
9: Compute E_{loo}:
10: – Compute diag(HAT) (row-wise dot-product of $\mathbf{H}$ and $\mathbf{H}^{\dagger T}$);
11: – $E_{loo,j} = \frac{1}{M}\sum_{i=1}^{M}\frac{y_i - \hat{y}_i}{1 - hat_{ii}}$;
12: **end for**
13: As model structure, select the ELM with that number of hidden neurons $n_j \in \aleph$, which minimizes $E_{loo,j}$;

In Fig. 1, the running times for training and combined training and model structure selection are compared. It can be seen that by explicitly computing $\mathbf{H}^{\dagger}$, the training procedure becomes somewhat slower, but due to the re-use of $\mathbf{H}^{\dagger}$ in the model structure selection, combined training and model structure selection became a lot faster. In practice, one can of course use the fastest function, depending on whether the model just needs to be trained or the model structure also needs to be selected.

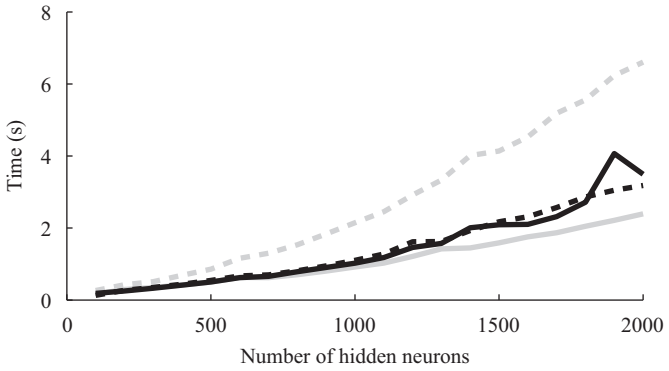


Fig. 1. Comparison of running times of ELM training (*solid lines*) and ELM training + model structure selection (*dotted lines*), with (*black lines*) and without (*gray lines*) explicitly computing and reusing $\mathbf{H}^{\dagger}$.

3. Ensemble model of ELM

A common way to achieve reduced error in a certain task is by building multiple models and average (or take a linear combination of) of their outputs. This is what is called an ensemble model. The idea behind it is that the individual models make different errors (in different directions), and that these errors tend to cancel each other out, resulting in a reduced error.

In order to determine the optimal combination of the models, the individual models have to be evaluated on a subset of the data (say, a calibration set) for which the target values are known. After evaluation, each model's predictions of the target values in the calibration are known. Now, using these predictions, the linear combination of these predictions that best fits the true target values can be determined. Computing this linear combination is done with positivity constraints on the weights.

Alternatively, instead of the outputs of the models, their leave-one-out outputs can be used for determining the optimal linear combination of the models. This way, a separate calibration set is not needed, and the ensemble method can be build using just the training set. Also, using leave-one-out output prevents overfitting the linear combination to the data on which it is optimized. For more information on this particular method of creating ensemble models, see [18].

Since ELMs are partially random non-linear models, they provide a set of quasi-independent models. For that reason, it is possible to use an ensemble methodology in order to achieve better generalization performance. The independence between the ELMs is increased by using a random subset of variables for the training of each ELM. A total of 100 ELM models are build, and for each ELM individually, the number of hidden neurons is tuned by performing the LOO cross-validation as described in Section 2.2.

After model structure selection, the ELMs can be combined into an ensemble model. In a previous work [17], a calibration set (separate from the training set) was used to determine the ensemble weights (i.e. the linear combination of the ELMs). However, since in this paper during the model structure selection, the leave-one-out error is computed on the training set, the leave-one-out output on the training set is already computed, and can be used to determine the optimal linear combination of models.

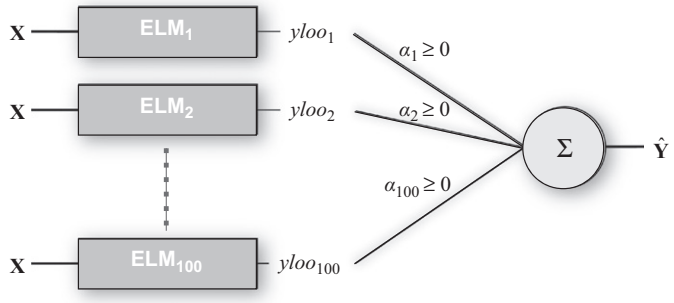


Fig. 2. Block diagram showing the overall setup of the ensemble of ELMs.

Added advantage of this approach is that there is no need to sacrifice part of the training set for the calibration of the ensemble, and the models can thus be trained more effectively.

Once the ensemble weights are calibrated using the LOO output of the ELMs, the calibrated ensemble is evaluated on a test set. The output of the ensemble is computed as the linear combination of the outputs of the individual models. Fig. 2 summarizes the overall implementation.

Further parallelization possibilities can clearly be seen from Fig. 2: every ELM can be constructed independently from the other ELMs and therefore the creation of the ELMs is parallelized over multiple GPUs and CPUs. Also, the ELMs themselves can be accelerated. These optimizations will be discussed in detail in the next section.

4. GPU-acceleration of ELMs and parallelization

4.1. Motivation

Many techniques have been developed in the field of machine learning to analyze data, and to extract useful information from it, which can be used to gain insight in the data or perform a task like prediction. However, due to advances in technology, the size and dimensionality of the data sets used in machine learning continue to grow by the day. Therefore, it is important to have efficient computational methods and algorithms that are able to handle these large data sets, such that the model selection and learning can still be performed in reasonable time.

The ELM is well-suited for application on large data sets, since it is relatively fast compared with other methods 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 reduce their running time. First of all, because the ELMs are applied to large data sets, the running time is still significant. Second, on large data sets, typically large numbers of neurons are needed, which increases the running time of ELM. Third, in order to avoid under- and overfitting the data, one has to perform model structure selection, and thus compute multiple models with different structure.

In the next subsections, the methods used to reduce the running time of the ensemble of ELMs are discussed.

4.2. GPU-acceleration of ELM

Since the running time of the ELM algorithm largely consists of a single operation (solving the linear system), it is the prime target for optimizing the running time of the ELM. If this operation can be accelerated, then the running time of each ELM (and thus of the ensemble) can be greatly reduced. In this work, this operation is performed on the GPU.

Currently, there are several libraries in development aimed at speeding up a subset of the linear algebra functions found in LAPACK [6]:

- CULA tools [5]: A library introduced in October 2009, implementing a subset of LAPACK functions. The free variant of this package contains functions for solving a linear system (`culaGesv`), and performing a least-squares solve (`culaGels`).
- MAGMA [27]: A recently introduced linear algebra package aiming at running linear algebra operations on heterogeneous architectures (i.e. using both multi-core CPU and multiple GPUs present on the system, in order to solve a single problem).[1]

In this work, CULA Tools is used, which was the first widely available GPU-accelerated linear algebra package, and was developed in cooperation with NVidia. Therefore, it is likely to be well-supported. Specifically, the (`culaGesv`) and (`culaGels`) functions were used, and wrappers around these functions were written, such that they can be used from MATLAB in the training and model structure selection of the ELM.

Similar functions are offered by MATLAB and its underlying LAPACK library. An overview of all functions used in this paper can be found in Table 1. Since in our application of these functions all linear systems are fully determined, they give exactly the same result and only vary in running time.

Something worth noting about computations on GPU, is that even though double precision calculations are possible, GPUs perform much better in single precision [1]. In the NVidia GTX295 cards that were used in this work, the single precision performance is eight times higher than the double precision performance.[2] Therefore, one should use single precision calculations wherever possible.

A second reason for using single precision calculations wherever possible is that the way only half as much memory is needed, and the amount of needed memory determines how far the method will scale. In our experiments, each GPU has 896 MiB of video card memory at its disposal. This means that the part of our algorithm that is executed on GPU (i.e. line 7 in Algorithm 2) can use at most this amount of memory. For a training problem of 25 000 samples, approximately 100 MiB is needed, and the amount of memory needed scales linearly with the number of samples. Therefore, on the used hardware, the approach scales to approximately 200 000 samples. If one would use the NVidia Tesla C2070, which has 6 GiB of memory, the approach would scale to approximately 1.5 M samples.

In order to get an idea of the running time of the function `culaGels`, it is compared with MATLAB's commonly used `mldivide` (also known as \), as well as with the `gels` function from MATLAB's underlying highly optimized multi-threaded LAPACK library.[3]

Since on the CPU the performance in single precision is about twice the double precision performance, the functions are compared in both single precision and double precision.[4]

[1] It should be noted that this library is being developed by the creators of the widely used LAPACK.

[2] In NVidia's latest generation of video cards, the double precision performance has been greatly increased and operates at half the speed of single precision.

[3] Used MATLAB is version R2009b, which on our Intel i7 920 machine uses the highly optimized MKL library by Intel.

[4] The functions compared here are the functions typically used in the general case of training an ELM (i.e. the case with non-square **H**). In our optimized implementation as explained in Algorithm 2, we are dealing with a square matrix on the left-hand side of the equation (line 7). Therefore, we actually use the `culaGesv` and `gesv` functions for slightly higher performance.

Table 1
An overview of the various functions used.

Function name	Description	Runs on
`mldivide`, \	Solve linear system (MATLAB)	CPU
`gesv`	Solve linear system (LAPACK)	CPU
`gels`	Least-square solve (LAPACK)	CPU
`culaGesv`	Solve linear system (CULA)	GPU
`culaGels`	Least-square solve (CULA)	GPU

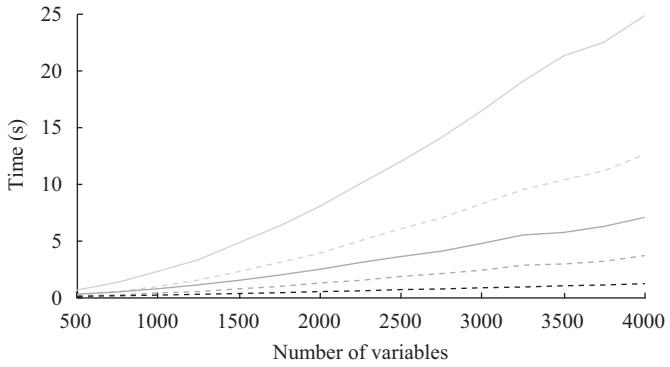


Fig. 3. Time (s) needed to solve a linear system of 5000 variables and one target variable, using `mldivide` (*light-gray lines*), `gels` (*gray lines*), `culaGels` (*dashed black line*) for double precision (*solid lines*) and single precision (*dashed lines*).

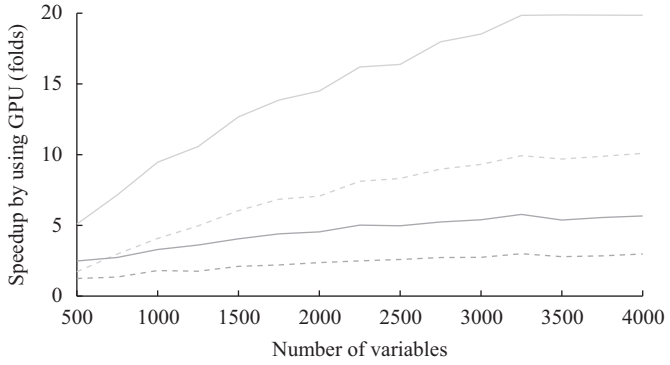


Fig. 4. Speedup achieved in solving system of 5000 variables and one target variable, by using `culaGels` instead of `mldivide`, (*light-gray lines*), `gels` (*gray lines*) for double precision (*solid lines*) and single precision (*dashed lines*).

In Fig. 3, the running times of the various functions for solving a linear system are shown. In Fig. 4, the speedup by using `culaGels` over the other algorithms can be seen (i.e. the lines from Fig. 3, divided by the black line from Fig. 3).

From these figures, it can be seen that the precision greatly affects the performance. Also, MATLAB's underlying LAPACK function `gels` perform much better than the commonly used `mldivide`. Finally, `culaGels` offers the fastest performance of all.

4.3. Parallellization across CPUs/GPUs

Looking at Fig. 2, one can see that the ELMs that are part of the ensemble model can be prepared and trained in a completely independent way. Therefore, running time can be optimized by dividing the preparation of all models across multiple CPU cores, and multiple GPUs.

This is achieved using MATLAB's parallel computing toolbox [28], which allows to create a pool of so-called MATLAB workers. Each of the workers runs its own thread for executing the program, and gets its own dedicated GPU assigned to it, which is used to accelerate the training and model structure selection that has to be performed for each model. As an example, consider the case of an ensemble of 100 ELMs, and four workers. In this case, each of the workers builds 25 ELMs.

Although in this paper, the parallelized ensemble model was not executed across multiple computers, the current code could be executed on multiple computers by using the MATLAB distributed computing toolbox.

5. Experiments and results

Experiments are performed on two relatively large regression data sets. The first one is the full Santa Fe Laser data set [19] for which the regression problem is based on a time series. The second data set is the ESTSP'08 competition data set number 3 [19] which is also a regression problem based on a time series computationally more challenging due to the size of the regressor used [20,21]. Sizes of the data sets are given in Table 2: 85% of the data is used for training, and the remaining 15% for test.

The ensemble model built in the experiments consists of 100 ELMs. In order to increase diversity between the ELMs, we randomly select which input variables from the regressor it uses. The ELMs have between 100 and 1000 neurons with sigmoid (`tanh`) transfer functions, and contain a linear neuron for every input they have,[5] such that they perform well on linear problems. Furthermore, in our implementation of ELM, an output bias is trained in addition to the output weights. Adding this feature has minimal overhead, and cross-validation experiments show this has no negative impact. However, it allows the ELM to adapt to changing properties of the data on retraining like, for example, a shift in the mean of the target data.

The ELMs are trained on 85% of the data and have their structure selected through the earlier discussed efficient LOO cross-validation on the training set.

Once the ELMs have been build, the ensemble weights are computed based on the LOO output of the ELMs on the training set. Finally, the ensemble is tested on the test set. See Table 3 for a summary of the parameters.

The used hardware consists of a desktop computer with Intel Core i7 920 CPU and NVidia GTX295 GPUs.

The experiments have been repeated several times for both data sets. Table 4 gives the total running times of the ensemble for the various functions used to build the ELMs (see Table 1 for a description of the functions). The functions are both evaluated in *single precision* and *double precision* (indicated by subscript *sp* and *dp* respectively).

Table 4 and Fig. 5 also show how the running time scales with the number of MATLAB workers.

The ensembles are also evaluated by their normalized mean square error (NMSE), where NMSE is defined as

$$\mathrm{NMSE} = \frac{\mathrm{MSE}}{var(\mathbf{Y})} = \frac{1/M\sum_{i=1}^{M}(y_i-\hat{y}_i)^2}{var(\mathbf{Y})}, \tag{7}$$

where M is the number of samples. Table 5 gives the NMSE of the ensembles on the test set.

Fig. 6 shows how the number of ELMs in the ensemble affects the NMSE of the ensemble. It can be seen that the more models are added to the ensemble, the lower the NMSE of the ensemble becomes.

[5] The neurons in the hidden layer are ordered such that the linear ones come first. Therefore, the linear neurons are always selected by the model structure selection procedure.

6. Discussion

The experiments show a 3.3 times speedup over the typical double precision implementation of an ensemble of ELMs, by using the GPU to speed up the slowest part of the algorithm, and parallelizing across multiple CPU cores and GPUs (i.e. $t(\texttt{mldiv}_{dp})$/ $t(\texttt{culaGesv}_{sp})$).

Table 2
Sizes of the used data sets. First column gives original total size of the data, while the other columns only mention the number of samples used in each type of set (training, test).

	Total size (samples × variables)	Training	Test
Santa Fe	10 081 × 12	8569	1512
ESTSP'08	31 446 × 168	26729	4717

Table 3
Parameters used in the experiments.

Parameter	Santa Fe	ESTSP'08
Regressor size	12	168
# Randomly selected variables	2–12	2–168
#Hidden neurons	100:100:1000	
Crit. for model struct. selection	LOO error on training set	
Trained on	Random 85% of the data	
Tested	Remaining 15% of the data	
Ensemble weights	Based on LOO output of ELMs	

Even if the parallelized GPU implementation is compared with the fastest parallelized CPU implementation, still a significant speedup is observed.

An unexpected result is the fact that the `gesv` functions have approximately the same running time as the `mldivide` functions, contrary to the observations in the earlier benchmarks in Section 4.2. We expect this to be the case, because the functions are applied in a different situation (i.e. in the case with multiple columns on the right-hand side). However, the result of the GPU variants of the functions being faster than the CPU variants of the functions always holds.

Another unexpected result was the fact that running a job in the MATLAB parallel toolbox with 1 worker (i.e. not parallelized), is much slower than running the job without the Parallel Toolbox. It turns out this is due to the fact that every worker limits its execution to a single thread. Therefore, the code running within that worker runs on a single core, and no speedups are achieved by MATLAB's multi-threaded LAPACK (which normally uses multiple cores). Therefore, one has to take care to load the machine with enough workers, such that all CPU cores can be effectively used.

Table 5
Results for both data sets: normalized mean square test error and standard deviation (in parenthesis).

	Santa Fe	ESTSP'08
NMSE (std.)	$1.87e-3$ ($4.61e-4$)	$1.55e-2$ ($6.57e-4$)

Table 4
Results for both data sets: running times (in seconds) for running the entire ensemble in parallel on N workers, using the various functions in single precision (*sp*) and double precision (*dp*).

	N	t ($\texttt{mldiv}_{dp}$) (s)	t ($\texttt{gesv}_{dp}$) (s)	t ($\texttt{mldiv}_{sp}$) (s)	t ($\texttt{gesv}_{sp}$) (s)	t ($\texttt{culaGesv}_{sp}$) (s)
Santa Fe	0	674.0	672.3	515.8	418.4	401.0
	1	1781.6	1782.4	1089.3	1088.8	702.9
	2	917.5	911.5	567.5	554.7	365.3
	3	636.1	639.0	392.2	389.3	258.7
	4	495.7	495.7	337.3	304.0	**207.8**
ESTSP	0	2145.8	2127.6	1425.8	1414.3	1304.6
	1	5636.9	5648.9	3488.6	3479.8	2299.8
	2	2917.3	2929.6	1801.9	1806.4	1189.2
	3	2069.4	2065.4	1255.9	1248.6	841.9
	4	1590.7	1596.8	961.7	961.5	**639.8**

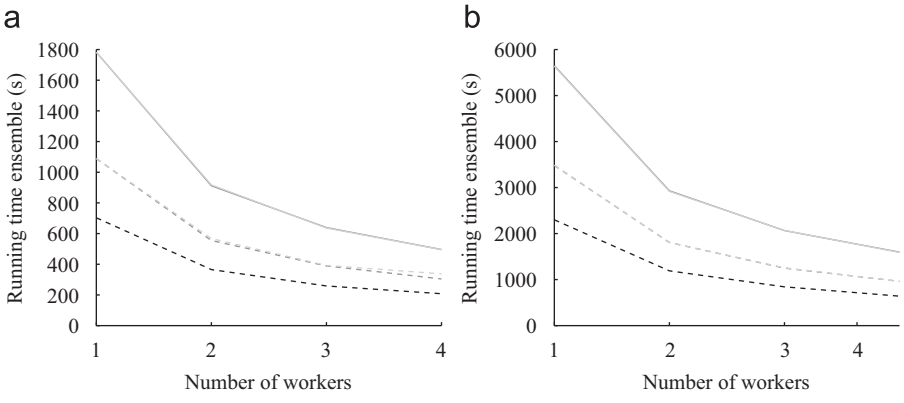


Fig. 5. Running times (in seconds) for running the entire ensemble in parallel on (a) Santa Fe and (b) ESTSP'08, for varying numbers of workers, using `mldivide` (*light-gray lines*), `gesv` (*gray lines*), `culaGesv` (*black line*) for double precision (*solid lines*) and single precision (*dashed lines*).

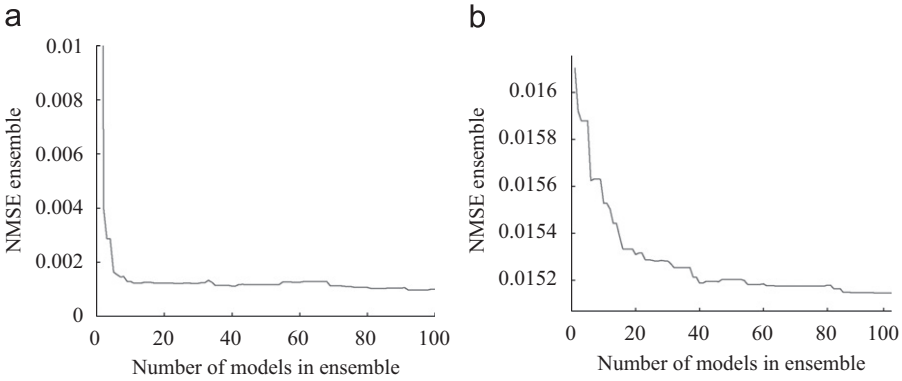


Fig. 6. NMSE of an ensemble model with varying number of models on (a) Santa Fe and (b) ESTSP'08.

Finally, Fig. 6 clearly shows how the number of ELMs in the ensemble affects the NMSE of the ensemble, and it can be seen that the more models are added to the ensemble, the lower the NMSE of the ensemble generally becomes.

Although results on the errors of the individual models compared with the errors of the ensemble model are not extensively reported here, it is important to mention that the test error achieved by the ensemble model is almost always lower than the test error of the best model in that ensemble, which provides a convincing argument for using an ensemble model.

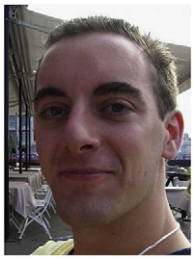
7. Conclusion and future work

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 effectively be parallelized across multiple GPUs and CPU-cores. Furthermore, the proposed approach is not limited to a specific type of ELM and can be employed for a large variety of ELMs.

Finally, in the future we would like to investigate the effect of running the ELM entirely on GPU, as well as explore the use of other types of ELMs, as well as other models such as reservoir computing methods [29], in the ensemble model.

References

[1] NVidia CUDA Programming Guide 3.1: 〈http://www.nvidia.com/object/cuda_get.html〉.
[2] NVidia CUDA Zone: 〈http://www.nvidia.com/object/cuda_home.html〉.
[3] V. Volkov, J.W. Demmel, Benchmarking GPUs to tune dense linear algebra, in: SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, no. November (Piscataway, NJ, USA), IEEE Press, 2008, pp. 1–11.
[4] B. Catanzaro, N. Sundaram, K. Keutzer, Fast support vector machine training and classification on graphics processors, in: Proceedings of the 25th International Conference on Machine Learning (ICML 2008), Helsinki, Finland, ACM, 2008, pp. 104–111.
[5] CULA (GPU-Accelerated LAPACK): 〈http://www.culatools.com/〉.
[6] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J.D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users'-guide, third ed., Society for Industrial Mathematics, Philadelphia, PA, 1999.
[7] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of the International Joint Conference on Neural Networks, 2004.
[8] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[9] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Real-time learning capability of neural networks, IEEE Transactions on Neural Networks 17 (July) (2006) 863–878.
[10] M. van Heeswijk, Y. Miche, T. Lindh-Knuutila, P.A. Hilbers, T. Honkela, E. Oja, A. Lendasse, Adaptive ensemble models of extreme learning machines for time series prediction, in: C. Alippi, M.M. Polycarpou, C.G. Panayiotou, G. Ellinas (Eds.), ICANN 2009, Part II, Heidelberg, Springer, 2009, pp. 305–314.
[11] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, IEEE Transactions on Neural Networks 17 (November) (2006) 1411–1423.
[12] G. Feng, G.-B. Huang, Q. Lin, R. Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, IEEE Transactions on Neural Networks 20 (8) (2009) 1352–1357.
[13] Y. Lan, Y.C. Soh, G.-B. Huang, Ensemble of online sequential extreme learning machine, Neurocomputing 72 (August) (2009) 3391–3395.
[14] G.-B. Huang, X. Ding, H. Zhou, Optimization method based extreme learning machine for classification, Neurocomputing 74 (1−3) (2010) 155–163, doi:10.1016/j.neucom.2010.02.019.
[15] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, OP-ELM: optimally pruned extreme learning machine, IEEE Transactions on Neural Networks 21 (October) (2010) 158–162.
[16] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Transactions on Neural Networks 17 (4) (2006) 879–892.
[17] M. van Heeswijk, Y. Miche, E. Oja, A. Lendasse, Solving large regression problems using an ensemble of GPU-accelerated ELMs, in: M. Verleysen (Ed.), ESANN 2010: 18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, d-side Publications, April 2010, pp. 309–314.
[18] Y. Miche, E. Eirola, P. Bas, O. Simula, C. Jutten, A. Lendasse, M. Verleysen, Ensemble modeling with a constrained linear system of leave-one-outputs, in: M. Verleysen (Ed.), ESANN 2010: 18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, d-side Publications, 2010, pp. 19–24.
[19] Santa Fe Laser and ESTSP'08 Competition Data available at: 〈http://www.cis.hut.fi/projects/eiml/research/downloads/datasets〉.
[20] M. Olteanu, Revisiting linear and non-linear methodologies for time series prediction-application to ESTSP'08 competition data, in: Proceedings of the 2nd European Symposium on Time Series Prediction, ESTSP'08, Porvoo, Finland, September 2008, pp. 139–148.
[21] N. Kourentzes, S.F. Crone, Automatic modeling of neural networks for time series prediction—in search of a uniform methodology across varying time frequencies, in: Proceedings of the 2nd European Symposium on Time Series Prediction, ESTSP 08, Porvoo, Finland, September 2008, pp. 117–127.
[22] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (October) (2007) 3056–3062.
[23] C. Rao, S. Mitra, Generalized Inverse of the Matrix and Its Applications, John Wiley & Sons Inc., 1971.
[24] G. Schwarz, Estimating the dimension of a model, The Annals of Statistics 6 (1978) 461–464.
[25] Y. Miche, A. Lendasse, A faster model selection criterion for OP-ELM and OP-KNN: Hannan-Quinn criterion, in: M. Verleysen (Ed.), ESANN 2009: European Symposium on Artificial Neural Networks, d-side publications, April 2009, pp. 177–182.
[26] R.H. Myers, Classical and Modern Regression with Applications, second ed., Duxbury, Pacific Grove, CA, USA, 1990.
[27] MAGMA: (Matrix Algebra on GPU and Multicore Architecture): 〈http://icl.cs.utk.edu/magma/〉.
[28] MATLAB Parallel Computing Toolbox: 〈http://www.mathworks.com/〉.
[29] D. Verstraeten, B. Schrauwen, M. D'Haene, D. Stroobandt, An experimental unification of reservoir computing methods, Neural Networks 20 (April) (2007) 391–403 (Echo State Networks and Liquid State Machines).



Mark van Heeswijk has been working as an exchange student in both the EIML (Environmental and Industrial Machine Learning, previously TSPCi) Group and Computational Cognitive Systems Group on his Master's Thesis on "Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction", which he completed in August 2009. Since September 2009, he started as a Ph.D. student in the EIML Group, ICS Department, Aalto University School of Science and Technology. His main research interest is in the field of high-performance computing and machine learning. In particular, how techniques and hardware from high-performance computing can be applied to meet the challenges one has to deal with in machine learning. He is also interested in biologically inspired computing, i.e. what can be learned from biology for use in machine learning algorithms and in turn what can be learned from simulations about biology. Some of his other related interests include: self-organization, complexity, emergence, evolution, bioinformatic processes, and multi-agent systems.

Yoan Miche was born in 1983 in France. He received an Engineer's Degree from Institut National Polytechnique de Grenoble (INPG, France), and more specifically from TELECOM, INPG, on September 2006. He also graduated with a Master's Degree in Signal, Image and Telecom from ENSERG, INPG, at the same time. He is currently finishing in both Gipsa-Lab, INPG, France and ICS Laboratory, Aalto University School of Science and Technology, Finland, his Ph.D. His main research interests are steganography/steganalysis and machine learning for classification/regression.

Erkki Oja (S'75-M'78-SM'90-F'00) received the D.Sc. degree from Helsinki University of Technology in 1977. He is Director of the Adaptive Informatics Research Centre and Professor of Computer Science at the Laboratory of Computer and Information Science, Aalto University (former Helsinki University of Technology), Finland, and the Chairman of the Finnish Research Council for Natural Sciences and Engineering. He holds an honorary doctorate from Uppsala University, Sweden. He has been a research associate at Brown University, Providence, RI, and visiting professor at the Tokyo Institute of Technology, Japan. He is the author or coauthor of more than 300 articles and book chapters on pattern recognition, computer vision, and neural computing, and three books: "Subspace Methods of Pattern Recognition" (New York: Research Studies Press and Wiley, 1983), which has been translated into Chinese and Japanese; "Kohonen Maps" (Amsterdam, The Netherlands: Elsevier, 1999), and "Independent Component Analysis" (New York: Wiley, 2001; also translated into Chinese and Japanese). His research interests are in the study of principal component and independent component analysis, self-organization, statistical pattern recognition, and applying artificial neural networks to computer vision and signal processing. Prof. Oja is a member of the Finnish Academy of Sciences, Fellow of the IEEE, Founding Fellow of the International Association of Pattern Recognition (IAPR), Past President of the European Neural Network Society (ENNS), and Fellow of the International Neural Network Society (INNS). He is a member of the editorial boards of several journals and has been in the program committees of several recent conferences including the International Conference on Artificial Neural Networks (ICANN), International Joint Conference on Neural Networks (IJCNN), and Neural Information Processing Systems (NIPS). Prof. Oja is the recipient of the 2006 IEEE Computational Intelligence Society Neural Networks Pioneer Award.

Amaury Lendasse was born in 1972 in Belgium. He received the M.S. degree in Mechanical Engineering from the Universite Catholique de Louvain (Belgium) in 1996, M.S. in control in 1997 and Ph.D. in 2003 from the same university. In 2003, he has been a post-doctoral researcher in the Computational Neurodynamics Lab at the University of Memphis. Since 2004, he is a senior researcher and a docent in the Adaptive Informatics Research Centre in the Aalto University School of Science and Technology (previously Helsinki University of Technology) in Finland. He has created and is leading the Environmental and Industrial Machine Learning (previously time series prediction and chemoinformatics) Group. He is chairman of the annual ESTSP conference (European Symposium on Time Series Prediction) and member of the editorial board and program committee of several journals and conferences on machine learning. He is the author or the coauthor of around 100 scientific papers in international journals, books or communications to conferences with reviewing committee. His research includes time series prediction, chemometrics, variable selection, noise variance estimation, determination of missing values in temporal databases, non-linear approximation in financial problems, functional neural networks and classification.