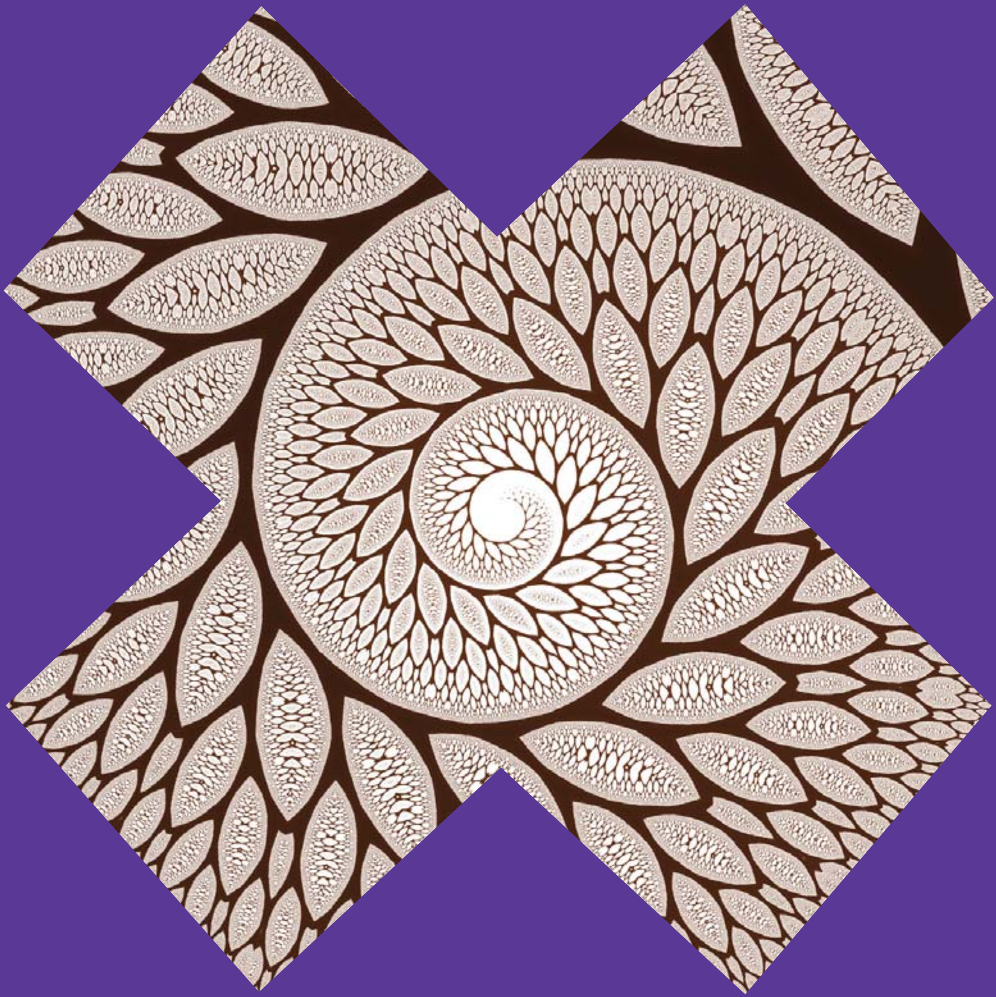
Department of Information and Computer Science

Advances in Extreme Learning Machines

Mark van Heeswijk



Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 43/2015

Advances in Extreme Learning Machines

Mark van Heeswijk

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T2 at the Aalto University School of Science (Espoo, Finland) on the 17th of April 2015 at 12 noon.

Aalto University
School of Science
Department of Information and Computer Science
Environmental and Industrial Machine Learning Group

Supervising professor
Aalto Distinguished Prof. Erkki Oja

Thesis advisor
Dr. Yoan Miche

Preliminary examiners
Prof. Guang-Bin Huang, Nanyang Technological University, Singapore
Prof. Jonathan Tapson, University of Western Sydney, Australia

Opponent
Prof. Donald C. Wunsch, Missouri University of Science & Technology, United States

Aalto University publication series
DOCTORAL DISSERTATIONS 43/2015

© Mark van Heeswijk

ISBN 978-952-60-6148-1 (printed)
ISBN 978-952-60-6149-8 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-6149-8

Unigrafia Oy
Helsinki 2015

Finland

NORDIC ECOLABEL
441 697
Printed matter

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Mark van Heeswijk

Name of the doctoral dissertation
Advances in Extreme Learning Machines

Publisher School of Science

Unit Department of Information and Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 43/2015

Field of research Information and Computer Science

Manuscript submitted 19 January 2015 **Date of the defence** 17 April 2015

Permission to publish granted (date) 9 March 2015 **Language** English

☐ **Monograph** ☒ **Article dissertation (summary + original articles)**

Abstract

Nowadays, due to advances in technology, data is generated at an incredible pace, resulting in large data sets of ever-increasing size and dimensionality. Therefore, it is important to have efficient computational methods and machine learning algorithms that can handle such large data sets, such that they may be analyzed in reasonable time. One particular approach that has gained popularity in recent years is the Extreme Learning Machine (ELM), which is the name given to neural networks that employ randomization in their hidden layer, and that can be trained efficiently. This dissertation introduces several machine learning methods based on Extreme Learning Machines (ELMs) aimed at dealing with the challenges that modern data sets pose. The contributions follow three main directions.

Firstly, ensemble approaches based on ELM are developed, which adapt to context and can scale to large data. Due to their stochastic nature, different ELMs tend to make different mistakes when modeling data. This independence of their errors makes them good candidates for combining them in an ensemble model, which averages out these errors and results in a more accurate model. Adaptivity to a changing environment is introduced by adapting the linear combination of the models based on accuracy of the individual models over time. Scalability is achieved by exploiting the modularity of the ensemble model, and evaluating the models in parallel on multiple processor cores and graphics processor units. Secondly, the dissertation develops variable selection approaches based on ELM and Delta Test, that result in more accurate and efficient models. Scalability of variable selection using Delta Test is again achieved by accelerating it on GPU. Furthermore, a new variable selection method based on ELM is introduced, and shown to be a competitive alternative to other variable selection methods. Besides explicit variable selection methods, also a new weight scheme based on binary/ternary weights is developed for ELM. This weight scheme is shown to perform implicit variable selection, and results in increased robustness and accuracy at no increase in computational cost. Finally, the dissertation develops training algorithms for ELM that allow for a flexible trade-off between accuracy and computational time. The Compressive ELM is introduced, which allows for training the ELM in a reduced feature space. By selecting the dimension of the feature space, the practitioner can trade off accuracy for speed as required.

Overall, the resulting collection of proposed methods provides an efficient, accurate and flexible framework for solving large-scale supervised learning problems. The proposed methods are not limited to the particular types of ELMs and contexts in which they have been tested, and can easily be incorporated in new contexts and models.

Keywords Extreme Learning Machine (ELM), high-performance computing, ensemble models, variable selection, random projection, machine learning

ISBN (printed) 978-952-60-6148-1 **ISBN (pdf)** 978-952-60-6149-8

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2015

Pages 192 **urn** http://urn.fi/URN:ISBN:978-952-60-6149-8

Preface

This work has been carried out at the Department of Information and Computer Science at the Aalto University School of Science and was made possible thanks to the funding and support of the Adaptive Informatics Research Centre (AIRC), the Department of Information and Computer Science, the Helsinki Graduate School in Science and Engineering (Hecse) and the Finnish Cultural Foundation (SKR). In addition, I would like to thank the Nokia Foundation for its financial support.

I am very grateful to my supervisor Erkki Oja for his support and excellent supervision. Thank you for always having an open door and some words of advice and encouragement, whenever it was needed. Furthermore, I would like to thank my instructor Yoan Miche and former instructor Amaury Lendasse for their expertise and guidance in the daily research and helping me develop and find my way as a researcher.

I am thankful to Guang-Bin Huang and Jonathan Tapson for their careful pre-examination of the thesis. Thank you for your valuable comments and suggestions for improvements. Furthermore, I would like to thank Donald C. Wunsch for the honor of having him as my opponent.

In addition to my instructors, I would like to thank Juha Karhunen, who is currently leading the Environmental and Industrial Machine Learning Group, as well as current and former members of our group: Francesco Corona, Federico Montesino Pouzols, Antti Sorjamaa, Qi Yu, Elia Liitiäinen, Dušan Sovilj, Emil Eirola, Alexander Grigorievskiy, Luiza Sayfullina, Ajay Ramaseshan, Anton Akusok and Li Yao and Zhangxing Zhu (in no particular order). It is impossible to thank all my colleagues from the department by name, but let me at least thank here: Mats Sjöberg, Kyunghyun Cho, Xi Chen, Nicolau Gonçalves, Ricardo Vigário, Oskar Kohonen, Mari-Sanna Paukkeri, Tiina Lindh-Knuutila, Nima Reyhani, Onur Dikmen, Zaur Izzatdust and Paul Wagner (also, in no particular order). Thank you for the many interesting discussions, pre-Christmas parties and trips throughout the years. Special thanks as well to Minna Kauppila, Leila Koivisto and Tarja Pihamaa, who on many an occasion helped organize practical matters for conference travels.

I would like to thank my friends and family for their support. I am extremely grateful to my parents as well as my brother Rob for their support and always believing in me, no matter what path I chose to pursue. Thank you! Finally, my love and deepest gratitude go to Gosia for her support, love and patience during these past years. You make everything better.

Espoo, March 2015,

Mark van Heeswijk

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Mark van Heeswijk, Yoan Miche, Tiina Lindh-Knuutila, Peter A.J. Hilbers, Timo Honkela, Erkki Oja, and Amaury Lendasse. Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction. In *LNCS 5769 - Artificial Neural Networks, ICANN'09: International Conference on Artificial Neural Networks*, pp. 305-314, September 2009.

II Mark van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, 74 (16): pp. 2430-2437, September 2011.

III Benoît Frenay, Mark van Heeswijk, Yoan Miche, Michel Verleysen, and Amaury Lendasse. Feature selection for nonlinear models with extreme learning machines. *Neurocomputing*, 102, pp. 111-124, February 2013.

IV Alberto Guillén, Maribel García Arenas, Mark van Heeswijk, Dušan Sovilj, Amaury Lendasse, Luis Herrera, Hector Pomares and Ignacio Rojas. Fast Feature Selection in a GPU Cluster Using the Delta Test. *Entropy*, 16 (2): pp. 854-869, 2014.

V Mark van Heeswijk, and Yoan Miche. Binary/Ternary Extreme Learning Machines. *Neurocomputing*, 149, pp. 187-197, February 2015.

VI Mark van Heeswijk, Amaury Lendasse, and Yoan Miche. Compressive ELM: Improved Models Through Exploiting Time-Accuracy Trade-offs. In *CCIS 459 - Engineering Applications of Neural Networks*, pp. 165-174, 2014.

Author's Contribution

Publication I: "Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction"

This publication introduces an ensemble of ELMs for one-step-ahead time series prediction, with ensemble weights adapting after each prediction step, depending on the error of the individual models. This allows the ensemble to adapt to nonstationarities in the time series. Furthermore, various retraining strategies are explored for retraining the models on a sliding or growing window. The proposed method is tested on stationary and nonstationary time series. Experiments show that the adaptive ensemble model has low computational cost, and achieves a test error comparable to the best methods, while keeping adaptivity. The present author defined the problem together with the other authors, and was responsible for most of the coding, experiments, and writing of the article.

Publication II: "GPU-accelerated and parallelized ELM ensembles for large-scale regression"

This publication presents an ensemble of GPU-accelerated ELMs, which are trained in parallel on multiple GPUs, such that regression on large data sets can be performed in reasonable time. Furthermore, an efficient method based on PRESS statistics is exploited for model selection. 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. Finally, the proposed approach is not limited to a specific type of ELM and can be employed for a large variety of ELMs. The present author was responsible for the proposal of the topic, coding, experiments, and most of the writing of the paper.

Publication III: "Feature selection for nonlinear models with extreme learning machines"

This publication explores a feature selection method based on Extreme Learning Machines, which returns a complete feature selection path representing the trade-off between the best feature subset for each subset size and the corresponding generalisation error. The present author contributed extensively to the definition of the problem and was responsible for the coding and write-up of the baseline experiments, consisting of various feature selection methods using the mutual information criterion.

Publication IV: "Fast Feature Selection in a GPU Cluster Using the Delta Test"

This publication proposes a 'genetic algorithm'-based feature selection method. The proposed algorithm is designed to be applied with very large datasets which could otherwise not be evaluated due to memory or time limitations. The workload is distributed over multiple nodes using the classical island approach. Furthermore, the main computational bottleneck (evaluation of the fitness function/Delta Test) is parallelized and evaluated across multiple GPUs. The present author was responsible for the initial implementation of the (multi-)GPU-accelerated Delta Test and writing the related parts of the paper.

Publication V: "Binary/Ternary Extreme Learning Machines"

This publication proposes two new ELM variants: Binary ELM, with a weight initialization scheme based on (0,1)-weights; and Ternary ELM, with a weight initialization scheme based on (-1,0,1)-weights. The motivation behind this approach is that these features will be from very different subspaces and therefore each neuron extracts more diverse information from the inputs than neurons with completely random features traditionally used in ELM. Experiments show that indeed ELMs with ternary weights generally achieve lower test error, and additionally are more robust to irrelevant and noisy variables. Since only the weight generation scheme is adapted, the computational time of the ELM is unaffected, and the improved accuracy, added robustness and the implicit variable selection of Binary ELM and Ternary ELM come for free. The present author was responsible for the proposal of the topic, coding, experiments, and most of the writing of the paper.

Publication VI: “Compressive ELM: Improved Models Through Exploiting Time-Accuracy Trade-offs”

This publication investigates the trade-off between the time spent optimizing and training several variants of the Extreme Learning Machine, and their final performance. Ideally, an optimization algorithm finds the model that has best test accuracy from the hypothesis space as fast as possible, and this model is efficient to evaluate at test time as well. However, in practice, there exists a trade-off between training time, testing time and testing accuracy, and the optimal trade-off depends on the user’s requirements. The proposed model in this publication, the Compressive Extreme Learning Machine, allows for a time-accuracy trade-off by training the model in a reduced space. Experiments indicate that this trade-off is efficient in the sense that on average more time can be saved than accuracy lost and therefore might provide a mechanism for obtaining better models in less time. The present author was responsible for the proposal of the topic, coding, experiments, and most of the writing of the paper.

List of Abbreviations

AIC	Akaike Information Criterion
BIC	Bayesian Information Criterion
BIP	Batch Intrinsic Plasticity
BW	Backward
CV	Cross-Validation
DT	Delta Test
ELM	Extreme Learning Machine
FSP	Feature Selection Path
FW	Forward
FWBW	Forward-Backward
GPU	Graphics Processing Unit
LARS	Least Angle Regression
LOO	Leave One Out
MI	Mutual Information
MSE	Mean Square Error
MRSR	Multiresponse Sparse Regression
PRESS	Predictive Sum of Squares
RMSE	Root Mean Square Error
SLFN	Single-Layer Feedforward Network
SET	Sparsity Error Trade-off
SVD	Singular Value Decomposition
TR	Tikhonov-Regularized

List of Notations

$\mathbf{x}$, $\mathbf{y}$	vectors
$\mathbf{X}$, $\mathbf{Y}$	matrices
$\hat{y}$	estimation of y
$\mathbf{x}_i$	i^{th}sample
$\mathbf{x}_{ij}$	i^{th}sample, j^{th}entry
N	number of samples
d	dimension of input samples
M	number of hidden neurons
$\mathbf{w}_i$	hidden layer weights of neuron i
b_i	hidden layer bias of neuron i
$f(\cdot)$	(transfer) function
$\mathbf{H}$	hidden layer output matrix
$\mathbf{H}^\dagger$	Moore-Penrose pseudo-inverse of $\mathbf{H}$
$\boldsymbol{\beta}$	hidden layer output weights
$\lvert\cdot\rvert$	absolute value
$\lVert\cdot\rVert$	L2 norm
δ	Delta Test

1. Introduction

1.1 Motivation and scope

Due to technological advances, nowadays data gets generated at an ever-increasing pace and the size and dimensionality of data sets continue to grow by the day. Therefore, it is important to develop efficient and effective machine learning methods, that can be used to analyze this data and extract useful knowledge and insights from this wealth of information.

In recent years, Extreme Learning Machines (ELMs) have emerged as a popular framework in machine learning. ELMs are a type of feed-forward neural networks characterized by a random initialization of their hidden layer weights, combined with a fast training algorithm. The effectiveness of this random initialization and their fast training makes them very appealing for large data analysis.

Although in theory ELMs have been proven to be universal approximators and the random initialization of the hidden neurons should be sufficient to solve any approximation problem, in practice it matters greatly how many samples are available for training; whether there are any outliers in the data; and which variables are used as inputs. Therefore, proper care needs to be taken to obtain a robust and accurate model, and prevent overfitting. Furthermore, even though ELMs have efficient training algorithms, due to the size of modern data sets, ELMs can benefit from strategies for accelerating their training.

The focus of this thesis therefore is on developing efficient, and effective ELM-based methods that are specifically suited for handling the challenges posed by modern data sets. The contributions of the dissertation are along three directions, described in the following section.

1.2 Contributions of the thesis

Firstly, **ELM-based ensemble methods** are developed, which adapt to context and can scale to large data. The stochastic nature of ELMs makes them particularly suited for ensembling, since each ELM tends to make different errors when modeling data. By combining them in an ensemble model, these errors are averaged out, resulting in a more accurate model. In particular, Publication I introduces an adaptive ensemble of ELMs, which allows for adapting to nonstationarities in the data by adjusting the linear combination of the models based on their accuracy over time. Publication II on the other hand, is aimed at reducing the computational time of the ensemble model, such that it may scale to larger data. Scalability is achieved by exploiting the modularity of the ensemble model, and evaluating its constituent models in parallel on multiple processor cores and by accelerating their training by performing it on graphical processing units (GPUs). Furthermore, an efficient method (based on PRESS-statistics) is exploited for fast model selection.

Secondly, **variable selection approaches based on ELM and Delta Test** are developed for reducing the dimensionality of the data by selecting only the relevant variables. This, in turn, results in more accurate and efficient models. In particular, Publication III introduces a new variable selection method based on ELM, which is shown to be a competitive alternative to traditional variable selection methods. Publication IV focuses on variable selection with a genetic algorithm using the Delta Test criterion for estimating the accuracy a nonlinear model can achieve for a given variable subset. The scalability of variable selection using Delta Test is achieved by accelerating it on GPU, and by parallelizing the workload over multiple cluster nodes. Finally, besides these explicit variable selection methods, Publication V develops a new weight initialization scheme for ELM consisting of binary and ternary sparse weights. As a result, the hidden neurons extract more diverse information from the data, which results in more accurate and effective models. This weight scheme is shown to perform implicit variable selection. Since only the weight scheme is adapted, the resulting increased robustness and accuracy come for free and at no increase in computational cost.

Finally, training algorithms for ELM are developed that allow for a **flexible trade-off between accuracy and computational time**. In particular, Publication VI introduces the Compressive ELM, which provides a way to reduce the computational time by performing the training of ELM in a reduced feature space. This allows for a flexible time-accuracy trade-off (and might provide a way to obtain more accurate models in less time).

Overall, the resulting collection of proposed methods provides an efficient, accurate and flexible framework for solving large-scale supervised learning problems. The developed methods are not limited to the particular types of ELMs and contexts in which they have been tested, and may readily be adapted to new contexts and models.

1.3 Structure of the thesis

The remainder of this thesis gives an introduction to topics and theory relevant to the thesis, and highlights results from the included publications. In particular, chapter 2 discusses the general machine learning background relevant to the thesis. Chapter 3 introduces Extreme Learning Machines and some of its variants. Chapter 4 discusses ensemble models and contributions to ensembles of ELMs. Chapter 5 gives an overview of feature selection and related contributions. Chapter 6 discusses the compressive ELM and finally, Chapter 7 provides conclusions and future work.

2. Machine learning

"All models are wrong, but some are useful."

– George Box

Machine learning is a challenging field, which is concerned with the problem of building models that can extract useful information or insights from given data. As mentioned in the introduction already, the size and dimensionality of the data sets become larger by the day, and it is therefore important to develop efficient computational methods and algorithms that are able to handle these large data sets, such that the machine learning tasks can still be performed in reasonable time.

This chapter gives an overview of the basic concepts of machine learning relevant to this thesis, and on supervised learning in particular.

2.1 Unsupervised learning

In machine learning, at least two different types of learning can be distinguished: supervised learning and unsupervised learning (Bishop, 2006; Murphy, 2012; Alpaydin, 2010). In unsupervised learning, no target variables are given, and the task is to extract useful patterns or information from just $(\mathbf{x_i})_{i=1}^{N}$, where $\mathbf{x}_i$ refers to the i^{th} sample in a data set of N samples (e.g. corresponding to images). Examples of unsupervised learning include clustering and principal component analysis (PCA), where the algorithm tries to discover latent structure in the data. Other uses of unsupervised learning are visualization or exploration of the data.

2.2 Supervised learning

In supervised learning on the other hand, the goal is to model the relationship between a set of explanatory variables $\mathrm{x_i}$ and the corresponding target variable (or target variables) y_i, where subscript i indicates the sample. That is, given a set of data $(\mathrm{x_i}, y_i)_{i=1}^N$, model the relationship between inputs $\mathrm{x_i}$ and outputs y_i as a function f, such that $f(\mathrm{x_i})$ matches y_i as closely as possible. This is often referred to as functional approximation.

In case the target variable $y_i \in \mathbb{R}$, this is known as regression. In case y_i corresponds to a category or class, this is known as classification.

2.2.1 Functional approximation

An example of a functional approximation problem is time series prediction, where the task is to predict future values of a particular time series based on its past values. One possibility for using past data to predict the future would be to model the next value of the time series (at time $t+1$) as a function of the values in the previous d time steps. Having recast the task of time series prediction as a functional approximation (or regression) problem, the problem of one-step ahead time series prediction can be described as follows

$$\hat{y}_i = f(\mathbf{x}_i, \boldsymbol{\beta}) \tag{2.1}$$

where $\mathbf{x}_i$ is a $1 \times d$ vector $[x(t-d+1), \ldots, x(t)]$ with d the number of past values that are used as input, and $\hat{y}_i$ the approximation of $x(t+1)$. Note the difference between $\mathbf{x}_i$ and $x(t)$.

Depending on what kind of relation is expected to exist between the input variables and output variables of a given problem, the regression is performed on either the input variables themselves or nonlinear transformations of them, e.g. like in neural networks which perform linear regression on nonlinear transformations of the input variables (i.e. the outputs of the hidden layer) and the target variables.

2.2.1.1 Linear regression

In linear regression, as the name suggests, the function f becomes a linear combination of the input variables, i.e.

$$f(\mathbf{x}_i, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}. \tag{2.2}$$

Given a number of training samples $(\mathrm{x}_i, y_i)_{i=1}^{N}$, the inputs $\mathrm{x_i}$ and targets y_i can be gathered in matrices, such that the linear system can be written as

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{Y} \tag{2.3}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \tag{2.4}$$

d is the number of inputs and N the number of training samples. The matrix $\mathbf{X}$ is also know as the regressor matrix and each row contains an input and a column of ones corresponding to β_0, while the corresponding row in $\mathbf{Y}$ contains the target to approximate.

The weight vector $\boldsymbol{\beta}$ which results in the least mean square error (MSE) approximation of the training targets $\mathbf{Y}$ given input $\mathbf{X}$ can now be computed as follows (Bishop, 2006):

$$\begin{aligned} \mathbf{X}\boldsymbol{\beta} &= \mathbf{Y} \\ \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{Y} \\ (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \\ \boldsymbol{\beta} &= (\mathbf{X^T X})^{\mathbf{-1}}\mathbf{X^T} = \mathbf{X}^{\dagger}\mathbf{Y} \end{aligned}$$

where $\mathbf{X}^{\dagger}$ is known as the pseudo-inverse or Moore-Penrose inverse (Rao and Mitra, 1971).[1]

Furthermore, since the approximation of the output for given $\mathbf{X}$ and $\boldsymbol{\beta}$ is defined as $\hat{\mathbf{Y}} = \mathbf{X}\boldsymbol{\beta}$

$$\begin{aligned} \hat{\mathbf{Y}} &= \mathbf{X}\boldsymbol{\beta} \\ &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \\ &= \mathrm{HAT} \cdot \mathbf{Y} \end{aligned}$$

where the HAT-matrix is the matrix that transforms the output $\mathbf{Y}$ into the approximated output $\hat{\mathbf{Y}}$. It is defined as $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ and plays an important role in this thesis, as it provides an efficient way to estimate the

[1]The matrix $\mathbf{X}^T\mathbf{X}$ is invertible (non-singular) exactly when its rank equals dimension d, which is usually the case if $N \geq d$. In case $N < d$, $\mathbf{X}^{\dagger} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$

expected performance of a linear model, and therefore an efficient way to perform model selection.

2.2.1.2 Linear basis function models

Instead of doing regression on the input variables, one can also perform regression on non-linear transformations of the input variables. These nonlinear transformations are often referred to as *basis functions*, and the approach as a whole as *basis function* expansion. This approach is more powerful than linear regression and can, given enough basis functions, approximate any given function under the condition that these basis functions are infinitely differentiable. In other words, they are universal approximators (Hornik et al., 1989; Cybenko, 1989; Funahashi, 1989).

2.2.2 Model structure selection

In supervised learning, a model tries to learn the relationship between a set of inputs and a set of outputs. This could for example be a set of images that needs to be classified into a number of categories, or a time series prediction problem, in which future values of that time series need to be predicted given its past values and possibly other external information.

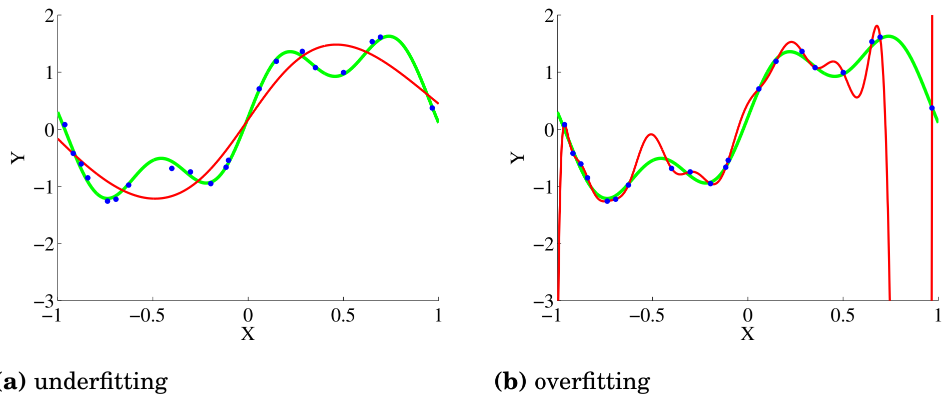
The model that is used to represent and learn this relationship has a certain structure determined by its parameters and a corresponding learning algorithm with its hyper-parameters. The class of possible models is sometimes known as the hypothesis space (Alpaydin, 2010), and it is up to the learning algorithm to find the best model from the hypothesis space (in terms of some criterion like e.g. accuracy) that models the relationship between input and output data best, and can consequently be used to accurately predict the output for future unseen inputs.

In optimizing the structure of a model, many models with different structure and parameters are evaluated according to some criteria. For example, in case of neural networks the models could differ in terms of the number or type of neurons in the hidden layer; how many and which variables are taken as input; and the algorithm and parameters used to train the neural network.

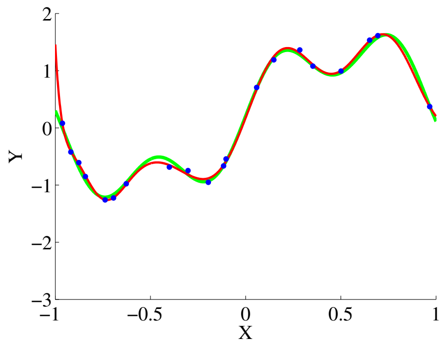
A commonly used criterion is the accuracy of the model. However, since the future samples are not necessarily the same as the currently available samples (e.g. due to noise or other changes in the environment), it is important that the model generalizes to this new unseen data: i.e. it is not enough to perfectly model the training data.

2.2.2.1 Motivation

In selecting the right model structure, one of the aspects often optimized is the model complexity (e.g. the number of hidden neurons in a neural network). If the model is too complex, it will perfectly fit the training data, but will have bad generalization on data other than the training data. On the other hand, if the model is too simple, it will not be able to approximate the training data at all. These cases are known as overfitting and underfitting, respectively, and are illustrated in Figure 2.1. If the model is too simple, it is not able to learn the functional mapping between the inputs and the outputs; if the model is too complex on the other hand, it perfectly approximates the points it was trained on, but exhibits poor generalization performance and does not approximate the underlying function of the data very well.



(a) underfitting **(b)** overfitting



(c) good fit

Figure 2.1. Output of various models (*red line*), trained on a number of points (*blue dots*) of the underlying function (*green line*) (van Heeswijk, 2009)

From these examples, it becomes clear that there is a trade-off between accuracy of the model on the training set, and the generalization performance of the model on the test set. Furthermore, there is an optimal complexity of the model, for which the trained model generalizes well to the unseen test set.

In order to determine the optimal complexity, the expected generalization error needs to be estimated, and it needs to be determined without using the test set. Here, three approaches are discussed: validation, k-fold cross-validation and leave-one-out cross-validation. See (Bishop, 2006) and (Efron and Tibshirani, 1993) for more detailed information on model (structure) selection methods.

2.2.3 Model selection methods

A good model performs well on the training set, and the input-output mapping that the model learned from the training set transfers well to the test set. In other words, the model approximates the underlying function of the data well and has good generalization.

How well a model generalizes can be measured in the form of the *generalization error*. In case of a functional approximation problem, and using an ℓ_2 loss function, the generalization error can be defined as

$$E_{gen}(\boldsymbol{\theta}) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i, \boldsymbol{\theta}))^2 \tag{2.5}$$

where N is the number of samples, $\mathbf{x}_i$ is the d-dimensional input, $\boldsymbol{\theta}$ contains the model parameters, and y_i is the output corresponding to input vector $\mathbf{x}_i$.

Of course, in reality there is no infinite number of samples, but only a limited amount of samples in the form of a *training set* and a *test set*, consisting of samples that the model will be trained on and samples that the model will be tested on, respectively. Therefore, the training set is to be used to estimate the generalization performance, and thus the quality, of a given model.

Below, three different methods are discussed that are often used in model selection and the estimation of the generalization error of a model.

Validation In validation, part of the training set is set aside in order to evaluate the generalization performance of the trained model. If the indices of the samples in the validation set are denoted by val and the indices of the samples in the full training set by $train$, then the estimation of the generalization error is defined as

$$\hat{E}_{gen}^{\mathrm{VAL}}(\boldsymbol{\theta}^*) = \frac{1}{|val|} \sum_{i\in val} (y_i - f(\mathbf{x}_i, \boldsymbol{\theta}^*_{train\setminus val}))^2 \tag{2.6}$$

where $\boldsymbol{\theta}^*_{train \setminus val}$ denotes the model parameters trained on all samples that are in the training set, but not in the validation set. Note that once the validation procedure and model selection is completed, the model is trained on the full training set.

The problem with this validation procedure is that it is not very reliable, since a small part of the data is held out for validation, and it is unknown how representative this sample is for the test set.

k-Fold cross-validation k-fold cross-validation is similar to validation, except that the training set is divided into k parts (typically $k = 10$), each of which is used as validation set once, while the rest of the samples are used for training. The final estimation of the generalization error is the mean of the generalization errors obtained in each of the k folds

$$\hat{E}^{kCV}_{gen}(\boldsymbol{\theta}^*) = \frac{1}{k}\sum_{s=1}^{k}\left[\frac{1}{|val_s|}\sum_{i \in val_s}(y_i - f(\mathbf{x}_i, \boldsymbol{\theta}^*_{train \setminus val_s}))^2\right] \tag{2.7}$$

where $\boldsymbol{\theta}^*_{train \setminus val_s}$ denotes the model parameters trained on all samples that are in the training set, but not in validation set val_s.

Although k-fold cross-validation gives a better estimation of the generalization error, it is computationally more intensive than validation, since the validation is performed k times.

Leave-one-out cross-validation Finally, Leave-one-out (LOO) cross-validation is a special case of k-fold cross-validation, namely the case where $k = N$. The models are trained on N training sets, each of which omits exactly one of the samples. The left-out sample is used for validation, and the final estimation of the generalization error is the mean of the N obtained errors

$$\hat{E}^{LOO}_{gen}(\boldsymbol{\theta}^*) = \frac{1}{N}\sum_{i=1}^{N}(f(\mathbf{x}_i, \boldsymbol{\theta}^*_{-i}) - y_i)^2 \tag{2.8}$$

where $\boldsymbol{\theta}^*_{-i}$ denotes the model parameters trained on all samples that are in the training set except on sample i.

Due to the fact that better use is made of the training set, the LOO cross-validation gives the more reliable estimate of the generalization error. Although the amount of computation for LOO cross-validation might seem excessive, for linear models, a closed-form formula exists that can compute all leave-one-out errors efficiently.

Leave-one-out computation using PRESS statistics Although it might seem like a lot of work to compute the leave-one-out errors (i.e. N models would need to be trained), the leave-one-out errors of a linear model can be computed efficiently from its residuals (i.e. the errors of the trained model on the training set) through PRESS (Prediction Sum of Squares) statistics (Allen, 1974; Myers, 1990)

$$\text{PRESS} = \sum_{i=1}^{N}(y_i - \mathbf{x}_i\boldsymbol{\beta}_{-i})^2 = \sum_{i=1}^{N}(y_i - \hat{y}_{i,-i})^2 = \sum_{i=1}^{N}(\epsilon_{i,-i})^2.$$

where $\epsilon_{i,-i}$ denotes the leave-one-out error when sample i is left out (also known as the PRESS residual); y_i denotes the target output specified by sample i from the training set, and $\boldsymbol{\beta}_{-i}$ denotes the weight vector obtained when training the linear model on the training set with sample i left out.

The PRESS residuals $\epsilon_{i,-i}$ can be computed efficiently as follows

$$\begin{aligned}\epsilon_{i,-i} &= \frac{\epsilon_i}{1 - \mathbf{x}_{i\cdot}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_{i\cdot}^T} \\ &= \frac{y_i - \hat{y}_i}{1 - \mathbf{x}_{i\cdot}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_{i\cdot}^T} \\ &= \frac{y_i - \mathbf{x}_i\boldsymbol{\beta}}{1 - \mathbf{x}_{i\cdot}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_{i\cdot}^T} \\ &= \frac{y_i - \mathbf{x}_i\boldsymbol{\beta}}{1 - h_{ii}} \end{aligned} \tag{2.9}$$

where $\mathbf{x}_{i\cdot}$ is the i^{th} row of matrix $\mathbf{X}$, h_{ii} is the i^{th} element on the diagonal of the HAT matrix $\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$, which was already encountered in Section 2.2.1.1. Therefore, the model only needs to be trained once on the entire training set in order to obtain $\boldsymbol{\beta}$, as well as the HAT matrix. Once the model is trained, all the PRESS residuals can easily be derived using Equation 2.9. Obviously, this involves a lot less computation than training the model for all N possible training sets.

Although PRESS statistics define an efficient way to compute the leave-one-out errors for linear models, this approach is not limited to models that are linear in the input variables: e.g. it can also be used in models that are linear in nonlinear transformations of the input variables, an important class of which is Extreme Learning Machines.

3. Extreme Learning Machines

“Not all those who wander are lost.”

– J.R.R. Tolkien, *Lord of the Rings*

Extreme Learning Machines (ELMs) (Huang et al., 2004, 2006b) is the name for a collection of neural network models, which employ randomization of the hidden layer weights and a fast training algorithm. Typically, instead of optimizing the hidden layer and output weights through an iterative algorithm like backpropagation (Rumelhart et al., 1986), ELMs initialize the hidden layer randomly and training consists of solving the linear system defined by the hidden layer outputs and the targets. Despite the hidden layer weights being random, it has been proven that the ELM is still capable of universal approximation of any non-constant piecewise continuous function (Huang et al., 2006a; Huang and Chen, 2007, 2008). Due to its speed and broad applicability, the ELM framework has become very popular in the past decade.

The goal of this chapter is not to give an exhaustive overview of the entire ELM literature, nor is the goal to include every single proposed ELM variant. Rather, the goal is to give a birds-eye view of Extreme Learning Machines; to put them in historical context; and to identify some of the learning principles used. For example, an ELM variant might include L1 regularization, L2 regularization, or might be pre-trained in some way. The amount of possible combinations of these learning principles (and thus the number of ELM variants) increases rapidly, yet the number of possible ways to optimize an ELM is relatively limited. The focus in this chapter will be mainly on variants related to the models developed in this thesis. For a more complete overview of ELM variants and applications, the reader is referred to Huang et al. (2011, 2015).

3.1 Historical context

The idea of randomization of the hidden layer of neural networks has become very popular under the name Extreme Learning Machines and the name has become associated with a vast assortment of different models and variants of neural networks with randomized weights, including Single-Layer Feedforward Networks (SLFNs) (Huang et al., 2006b), kernelized SLFNs (Frénay and Verleysen, 2010, 2011; Huang et al., 2010), and deep architectures (Kasun et al., 2013).

The idea of randomization of the hidden layer in neural networks has been proposed several times. For example, the Random Vector Functional Link (RVFL) network (Pao and Takefuji, 1992; Pao et al., 1994; Igelnik and Pao, 1995) incorporates random hidden layer weights and biases, and direct connections between the input layer and output layer. Furthermore, several authors (Schmidt et al., 1992; te Braake and van Straten, 1995; te Braake et al., 1996; Chen, 1996; te Braake et al., 1997) introduced neural networks with a randomly initialized hidden layer, trained using the pseudo-inverse. This approach has also been used in the past for initializing the weights of a neural network (Yam and Chow, 1995; Yam et al., 1997; Yam and Chow, 2000) before training it with e.g. backpropagation. Finally, more recently, (Widrow et al., 2013) proposed the No-Prop algorithm, which has a random hidden layer and uses the LMS algorithm for training the output weights, rather than the pseudo-inverse. For an overview of how ELM compares to other methods incorporating randomization, see (Wang and Wan, 2008; Huang, 2008, 2014).

Although the idea of randomization in neural networks appears elsewhere, it cannot be denied that with the development of the Extreme Learning Machine over the past decade, the idea and theory of using randomization in neural networks has really come to fruition, and has been developed into a framework (rather than a single method) covering many machine learning methods, the uniting factor being the fact that some sort of random basis expansion / randomized hidden layer is used. Along with these methods, many theoretical and empirical results have been developed regarding the effectiveness of randomized features (Huang et al., 2015).

3.2 Standard ELM algorithm

The basic ELM algorithm for training Single-Layer Feedforward Neural Networks (SLFN) was first described in (Huang et al., 2006b). As mentioned, the key idea of ELM is the random initialization of the hidden layer weights and the subsequent training consists of computing the least-squares solution to the linear system defined by the hidden layer outputs and targets. An overview of the structure of an ELM is given in Figure 3.1 and the algorithm for training this network, as described in (Huang et al., 2006b), can be summarized as follows.

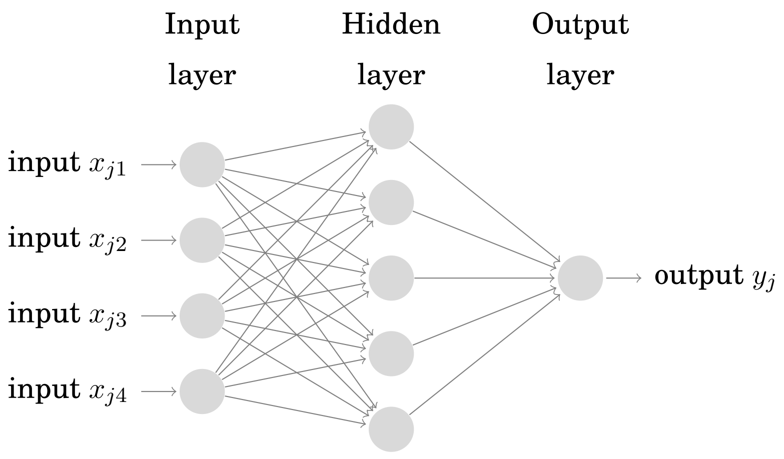


Figure 3.1. A schematic overview of an ELM

Consider a set of N distinct samples $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Then, the output of an SLFN with M hidden neurons can be written as

$$\hat{y_j} = \sum_{i=1}^{M} \beta_i f(\mathbf{w}_i \mathbf{x}_j + b_i), j \in [1, N], \tag{3.1}$$

where $\hat{y_j}$ is its approximation to y_j, f is the activation function, $\mathbf{w}_i$ the input weight vector, b_i the hidden layer bias and β_i the output weight corresponding to the i^{th} neuron in the hidden layer.

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

$$\sum_{i=1}^{M} \boldsymbol{\beta}_i f(\mathbf{w}_i \mathbf{x}_j + b_i) = y_j, j \in [1, N], \tag{3.2}$$

which can be written compactly as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{Y}, \tag{3.3}$$

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

$$\mathbf{H} = \begin{pmatrix} f(\mathbf{w}_1\mathbf{x}_1 + b_1) & \cdots & f(\mathbf{w}_M\mathbf{x}_1 + b_M) \\ \vdots & \ddots & \vdots \\ f(\mathbf{w}_1\mathbf{x}_N + b_1) & \cdots & f(\mathbf{w}_M\mathbf{x}_N + b_M) \end{pmatrix} \tag{3.4}$$

and $\boldsymbol{\beta} = (\beta_1 \dots \beta_M)^T$ and $\mathbf{Y} = (y_1 \dots y_N)^T$. See Algorithm 1 for a summary of the ELM algorithm.

Algorithm 1 Standard ELM

Given a training set $(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$, a probability distribution from which to draw random weights, an activation function $f : \mathbb{R} \mapsto \mathbb{R}$ and M the number of hidden nodes:

```
1: - Randomly assign input weights $\mathbf{w}_i$ and biases $b_i$, $i \in [1, M]$;
2: - Calculate the hidden layer output matrix $\mathbf{H}$;
3: - Calculate output weights matrix $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{Y}$.
```

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

3.3 Theoretical foundations

The strength of the Extreme Learning Machine is the fact that there is no need to iteratively tune of the randomly initialized network weights, which makes it very fast. Yet, despite the hidden neurons not being tuned, still an accurate network can be obtained.

Interpolation theory In particular, with the notations from Section 3.2, the Theorem presented in (Huang et al., 2006b) states that with randomly initialized input weights and biases for the SLFN, and under the condition that the activation function f is infinitely differentiable, then the hidden layer output matrix can be determined and will provide an approximation of the target values as good as wished (non-zero). Hence, the ELM can interpolate any set of samples as good as wished.

Theorem 1. *(Huang et al., 2006b). Given any small positive value $\epsilon > 0$, any activation function which is infinitely differentiable in any interval, and N arbitrary distinct samples $(\mathbf{x_i}, \mathbf{y_i}) \in \mathbb{R}^d \times \mathbb{R}^m$, there exists $M < N$ such that for any $\{\mathbf{w}_i, \mathbf{b}_i\}_{i=1}^{M}$ randomly generated from any interval of $\mathbb{R}^d \times \mathbb{R}$, according to any continuous probability distribution, with probability one, $\|\mathbf{H}\beta - \mathbf{T}\| < \epsilon$. Furthermore, if $M = N$, then with probability one, $\|\mathbf{H}\beta - \mathbf{T}\| = 0$.*

Universal approximation capability Besides being able to interpolate a finite set of samples, the ELM can also approximate any continuous target function f as good as wished.

Theorem 2. *(Huang et al., 2006a; Huang and Chen, 2007, 2008). Given any nonconstant piecewise continuous function $G : \mathbb{R}^d \to \mathbb{R}$, if span $\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in \mathbb{R}^d \times \mathbb{R}\}$ is dense in $L^2(\mathbb{R}^d)$ (i.e. the space of functions f on $\mathbb{R}^d$ which is a compact subset in the Euclidean space $\mathbb{R}^d$ such that $\int_{\mathbb{R}^d} |f(\mathbf{x})|^2 d\mathbf{x} < \infty$), then for any continuous target function f and any function sequence $\{G(\mathbf{w}_i, b_i, \mathbf{x})\}_{i=1}^{M}$ randomly generated according to any continuous sampling distribution, $\lim_{M\to\infty} \|f - f_M\| = 0$ holds with probability one if the output weights β_i are determined by ordinary least square to minimize $\left\| f(\mathbf{x}) - \sum_{i=1}^{M} \beta_i G(\mathbf{w}_i, b_i, \mathbf{x}) \right\|$.*

3.4 Building a sound and robust architecture

Although the details of how an ELM is generated and trained differ between ELM schemes, most of these schemes can in some way be considered a variant of the following Algorithm 2, with the exact details of each step varying between schemes. The goal of each of these schemes is optimization of the hidden layer, such that as good as possible performance is achieved in the context in which the ELM is applied.

Algorithm 2 General structure of ELM schemes

```
Generate the ELM
• while not ready:
    – Generate candidate neurons
    – Select those candidate neurons that give the best value of some criterion
Train the ELM
• determine optimal output weights training targets and outputs of the hidden neurons from the generation step, using an optimization criteria like least squares or a regularized version of it.
```

Three main approaches for optimizing the ELM structure can be identified: constructive approaches; pruning approaches; and regularization approaches, as well as combinations of them.

The next subsections give an overview of these main approaches for building a sound and robust architecture, as well as a method for pretraining the ELM in order to optimize the amount of information the hidden layer neurons extract.

3.4.1 Incremental approaches

The incremental approach starts from a small network, and incrementally grows the hidden layer by adding new neurons until a certain stopping criterion is reached.

For example, the *Incremental ELM (I-ELM)* (Huang et al., 2006a) adds neurons which reduce the residual error of the model so far obtained as much as possible. While doing so, it only needs to train the weight for the neuron added in the current step. The final network is the one which achieves a certain target training error, or if it does not achieve that error before reaching a specified network size, the network of that specified size.

The *Convex Incremental ELM (CI-ELM)* (Huang and Chen, 2007) improves on the convergence speed of the I-ELM towards low-error models through the use of a convex optimization method, and correcting (but not recomputing) the output weights with each incremental step.

As a final example, in the *Error-Minimized ELM (EM-ELM)* (Feng et al., 2009), more than one neuron can be added at the same time to grow the hidden layer. Additionally, the method has closed-form update rules for the weights when adding the new neurons, making the growing step fast.

3.4.2 Pruning approaches

Contrary to incremental approaches, pruning approaches first generate a larger than needed set of neurons. Given this set of candidate neurons, what remains is picking the best subset of M neurons for use in the SLFN.

In the Pruned ELM (Rong et al., 2008), a large set of candidate neurons is generated and ranked according to statistical relevance, using the χ^2 criterion or the information-gain criterion. An optimal threshold for this criterion is then determined using a separate validation set and the Akaike Information Criterion (AIC) (Akaike, 1974), after which the network is retrained on the entire training set.

The *Optimally Pruned ELM (OP-ELM)* (Miche et al., 2010) on the other hand, exploits the fact that the ELM is linear in the output of the hidden layer. This permits a fast and optimal ranking (in terms of training error) of the candidate neurons, using Least Angle Regression (LARS) (Efron et al., 2003), or Multiresponse Sparse Regression (MRSR) (Similä and Tikka, 2005). Once ranked, the optimal prefix of the sorted list of neurons is determined using the leave-one-out error, which can be efficiently computed using PRESS statistics (Allen, 1974; Myers, 1990).

Although the term 'pruned' suggest that the network architecture is being built starting from the largest network, and neurons are removed one-by-one, in fact the above approaches are quite similar to the incremental approach. The difference is that instead of randomly generating new neurons at each step, the entire candidate list of neurons is generated and ranked as a first step in the algorithm, and the neurons to be added are taken from that ranked candidate list of neurons. Therefore, the difference between the incremental and pruning approach is not that clear-cut.

For example, a recently proposed variant of the OP-ELM (which adds a number of regressors in each step of an MRSR-like algorithm, rather than a single one) was called the *Constructive Multi-output ELM* (Wang et al., 2014).

3.4.3 Regularization approaches

As an alternative to selecting the subset of hidden neurons, it is also possible to generate a large enough set of hidden neurons, and prevent overfitting by properly regularizing the network.

The *Regularized ELM (R-ELM)* (Deng et al., 2009) for example, is an approach in which the set of candidate neurons is fixed and taken large enough, while L2 regularization is used to prevent overfitting.

Finally, the *Tikhonov Regularized OP-ELM (TROP-ELM)* (Miche et al., 2011) is a variant of the OP-ELM, which efficiently incorporates the optimization of an L2 regularization parameter in the OP-ELM by integrating it in the SVD approach to computing pseudo-inverse $\mathbf{H}^{\dagger}$. This way, besides the advantage of sparsity, the output weights remain small and overfitting is prevented.

3.4.4 ELM pre-training

As it is extensively used in Publication V and Publication VI, in this section reviews intrinsic plasticity, as well as its adaptation to ELM (BIP-ELM) by (Neumann and Steil, 2011, 2013).

3.4.4.1 Motivation

Although ELMs are universal approximators, since often there are only limited training samples available. Therefore, it is important that the hidden layer neurons extract as much information as possible from the inputs.

A recently proposed pre-training method that achieves this is Batch Intrinsic Plasticity (BIP) (Neumann and Steil, 2011, 2013), which makes the ELM more robust by adapting the randomly generated hidden layer weights and biases such that each neuron achieves an exponential output distribution with a specified mean, and the amount of information that the hidden layer extracts from the limited amount of training samples is optimized.

Furthermore, the mechanism of intrinsic plasticity is one that is orthogonal to all the above-mentioned approaches. Namely, it generally takes place right after generating the random weights of the neurons, and its result is subsequently used in the further optimization, pruning and training of the ELM. As such, it can be used in combination with most other ELM approaches.

3.4.4.2 Intrinsic Plasticity

The concept of intrinsic plasticity has a biological background and refers to the fact that neurons adapt in such a way that they maximize their entropy (and thus the amount of information transmitted), while keeping the mean firing rate low. Intrinsic plasticity has been first used in papers regarding reservoir computing, recurrent neural networks, liquid state machines and echo state networks as a learning rule which maximizes information transmitted by the neurons (Triesch, 2005a,b; Verstraeten et al., 2007).

The information transmission of neurons is maximized by having the neuron outputs approximate an exponential distribution, which is the maximum entropy distribution among all positive distributions with fixed mean (Steil, 2007).

Furthermore, as (Verstraeten et al., 2007) notes in the context of reservoir computing, reservoirs are constructed in a stochastic manner, and the search for a method to construct a priori suitable reservoirs that are guaranteed or likely to offer a certain performance is an important line of research. Intrinsic plasticity is such a method which aims at constructing a network which is likely to give good performance. The recent study (Neumann et al., 2012) provides an in-depth analysis of intrinsic plasticity pre-training, and shows that it indeed results in well-performing networks with an impressive robustness against other network parameters like network size and strength of the regularization.

3.4.4.3 (Batch) Intrinsic Plasticity: BIP-ELM

In (Neumann and Steil, 2011, 2013; Neumann, 2013) the principle of intrinsic plasticity is transferred to ELMs and introduced as an efficient pre-training method, aimed at adapting the hidden layer weights and biases, such that the output distribution of the hidden layer is shaped like an exponential distribution. The motivation for this is that the exponential distribution is the maximum-entropy distribution over all distributions with fixed mean, maximizing the information transmission through the hidden layer. The only parameter of batch intrinsic plasticity is the mean of exponential distribution. This parameter determines the exact shape of the exponential distribution from which targets will be drawn, and can be set in various ways, as explained below.

Following (Neumann and Steil, 2011), the algorithm can be summarized as described as below.

Given the inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_N) \in \mathbb{R}^{N \times d}$ and input matrix $\mathbf{W}^{in} \in \mathbb{R}^{d \times M}$ (with N the number of samples in the training set, d the dimensionality of the data, and M the number of neurons), the synaptic input to neuron i is given by $s_i(k) = \mathbf{x}_k \mathbf{W}^{in}_{\cdot i}$. Now, it is possible to adapt slope a_i and bias b_i, such that the desired output distribution is achieved for neuron output $h_i = f(a_i s_i(k) + b_i)$. To this end, for each neuron random targets $\mathbf{t} = (t_1, t_2, \ldots, t_N)$ are drawn from the exponential distribution with a particular mean, and sorted such that $t_1 < \cdots < t_N$. The synaptic inputs to the neuron are sorted as well into vector $\mathbf{s_i} = (s_i(1), s_i(2), \ldots, s_i(N))$, such that $s_i(1) < s_i(2) < \cdots < s_i(N)$.

Given an invertible transfer function, the targets can now be propagated back through the hidden layer, and a linear model can be defined that maps the sorted $s_i(k)$ as closely as possible to the sorted t_k. To this end, a model $\Phi(\mathbf{s}_i) = (\mathbf{s}_i^T, (1 \ldots 1)^T)$ and parameter vector $\mathbf{v}_i = (a_i, b_i)^T$ are defined. Then, given the invertible transfer function f the optimal slope a_i and bias b_i for which each $s_i(k)$ is approximately mapped to t_k can be found by minimizing

$$||\Phi(\mathbf{s}_i) \cdot \mathbf{v}_i - f^{-1}(\mathbf{t})||$$

The optimal slope a_i and bias b_i can therefore, like in ELM, be determined using the Moore-Penrose pseudo-inverse:

$$\mathbf{v}_i = (a_i, b_i)^T = \Phi^{\dagger}(\mathbf{s}_i) \cdot f^{-1}(\mathbf{t})$$

This procedure is performed for every neuron with an invertible transfer function, and even though the target distribution can often not exactly be matched (due to the limited degrees of freedom in the optimization problem) it has been shown in (Neumann and Steil, 2011, 2013; Neumann, 2013) that batch intrinsic plasticity is an effective and efficient scheme for input-specific tuning of input weights and biases used in the non-linear transfer functions.

The stability of BIP-ELM combined with ridge regression like in the R-ELM (Deng et al., 2009) essentially removes the need to tune the amount of hidden neurons, and the only parameter of batch intrinsic plasticity is the mean of the exponential target distribution from which targets t are drawn, which is either set to a fixed value c, or randomly in the interval $[0, 1]$ on a per-neuron basis (Neumann and Steil, 2011, 2013; Neumann, 2013). These variants will be referred to as BIP(c)-ELM and BIP(rand)-ELM in this thesis.

3.5 Other ELM approaches

Besides the above-mentioned approaches, several other approaches have been developed over the past years, extending the ELM framework to different types of models, namely kernelized ELM (Frénay and Verleysen, 2010, 2011; Huang et al., 2010; Parviainen et al., 2010), multiple kernel ELM (Liu et al., 2015), representation learning using ELM (Kasun et al., 2013), ELMs with shaped input weights (Tapson et al., 2014; McDonnell et al., 2014), and semi-supervised and unsupervised ELM (Huang et al., 2014). More details on these methods can be found in the cited references, or in (Huang et al., 2015).

Furthermore, randomization ideas akin to ELM are increasingly being used in modern kernel methods, in order to let them scale to larger data, namely Random Kitchen Sinks (Rahimi and Recht, 2007, 2008), and Fastfood (Le et al., 2013).

3.6 ELM in practice

In theory, random initialization of the hidden layer and use of any non-constant piecewise continuous transfer function is sufficient for approximating any function, given enough neurons. In practice, however, there are a number of practical strategies that can be used for obtaining more accurate and effective ELMs. This section lists some of those practical tips for building a more effective ELM.

Normalization and pre-training As is well-known, data should be normalized such that each variable is zero-mean and unit-variance (or scaled to e.g. interval $[-1, 1]$. In practice, the former approach is more robust, since it is not as sensitive to outliers.

The range and number of input variables, together with the random weights of an ELM, will result in an expected activation at the input of each neuron, and one should make sure that e.g. the sigmoid neuron is not always operating in the saturated or linear region. For example, by letting the parameters of the probability distribution from which the random layer weights and biases are drawn depend on the number of inputs and transfer function, or by cross-validating them to optimize accuracy.

Another fast option is to use the Batch-Intrinsic Plasticity pre-training from Section 3.4.4, which automatically adapts the randomly drawn hidden layer weights and biases, such that each neuron operates in a useful regime.

Approximating the constant component In the non-kernel version of ELM, it might be helpful to include a bias in the output layer (i.e. achieved by concatenating the **H** matrix with a column of ones). Although this output bias is often not included in the description of the ELM since theoretically it is not needed, it allows the ELM to adapt to any non-zero mean in the targets at the expense of only a single extra parameter, namely the extra output weight.

Approximating the linear component Furthermore, in most problems, it is helpful to include a linear neuron for each input variable. This way, the rest of the nonlinear neurons can focus on fitting the nonlinear part of the problem, while the linear neurons take care of the linear part of the problem. Equivalently, an ELM could be trained on the residual of a linear model. This approach of decomposing the problem into a linear part and a nonlinear part has proven to be very effective in the context of deep learning (Raiko and Valpola, 2012).

4. Ensemble learning

"The only way of discovering the limits of the possible is to venture a little way past them into the impossible."

– Arthur C. Clarke

When discussing ensemble models it is helpful to look at a real-world example first. At fairs and exhibitions, sometimes there are these contests where the goal is to guess the number of marbles in a vase, and the person who makes the best guess wins the price. It turns out that while each individual guess is likely to be pretty far off, the average of all guesses is often a relatively good estimate of the real number of marbles in the vase. This phenomenon is often referred to as 'wisdom of the crowds'.

A similar strategy is employed in ensemble models: a number of individual models is built to solve a particular task, and these models are then combined into an ensemble model. Although the individual models might vary a lot in terms of accuracy, the combination gives a more accurate result.

This chapter introduces ensemble models, and the ELM-based ensemble models developed in this thesis, which make the ensemble adaptive to changes in the environment (Publication I) and allow them to scale to larger data (Publication II).

4.1 Ensemble Models

An ensemble model or committee (Bishop, 2006), combines multiple individual models, with the goal of reducing the expected error of the model. Commonly, this is done by taking the average or a weighted average of the individual models (see Figure 4.1).

Ensemble methods rely on having multiple good models with sufficiently uncorrelated errors. The simplest way to build an ensemble model is to

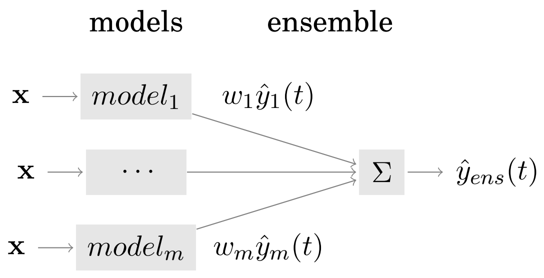


Figure 4.1. A schematic overview of how models can be combined in an ensemble (van Heeswijk, 2009)

take the average of the individual models (e.g. Figure 4.1, with $w_1 = \cdots = w_m = \frac{1}{m}$). In this case the output of the ensemble model becomes:

$$\hat{y}_{ens} = \frac{1}{m}\sum_{i=1}^{m} \hat{y}_i, \tag{4.1}$$

where $\hat{y}_{ens}$ is the output of the ensemble model, $\hat{y}_i$ are the outputs of the individual models and m is the number of models.

Now, following (Bishop, 2006), and assuming that the models are unbiased (i.e. absolute errors are zero-mean) and make independent errors, it can be shown that the variance of the ensemble model is lower than the average variance of all the individual models.

4.1.1 Error reduction by taking simple average of models

Suppose y denotes the true output to predict and $\hat{y}_i$ is the estimation of model i for this value. Then, the output $\hat{y}_i$ of model i can be written as the true value y plus some error term ϵ_i

$$\hat{y}_i = y + \epsilon_i, \tag{4.2}$$

and the expected error of the model is simply the mean square error

$$\mathbb{E}[\{\hat{y}_i - y\}^2] = \mathbb{E}[\epsilon_i^2]. \tag{4.3}$$

Now, define the average mean square error made by the models by

$$E_{avg} = \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}[\epsilon_i^2]. \tag{4.4}$$

Similarly, define the expected error of the ensemble as defined in Equation 4.1 by

$$E_{ens} = \mathbb{E}\Big[\Big\{\frac{1}{m}\sum_{i=1}^{m}(\hat{y} - y)\Big\}^2\Big] = \mathbb{E}\Big[\Big\{\frac{1}{m}\sum_{i=1}^{m}\epsilon_i\Big\}^2\Big]. \tag{4.5}$$

Then, assuming the errors ϵ_i are uncorrelated (i.e. $\mathbb{E}[\epsilon_i \epsilon_j] = 0$) and are zero-mean (i.e. $\mathbb{E}[\epsilon_i] = 0$), the expected ensemble error can be written as

$$E_{ens} = \frac{1}{m} E_{avg} = \frac{1}{m^2} \sum_{i=1}^{m} \mathbb{E}[\epsilon_i^2], \tag{4.6}$$

which suggests a great reduction of the error through ensembling. These equations assume completely uncorrelated errors between the models, while in practice errors tend to be highly correlated. Therefore, errors are often not reduced as much as suggested by these equations. It can be shown though that $E_{ens} < E_{avg}$ always holds (Bishop, 2006), so through ensembling, the test error of the ensemble is expected to be smaller than the average test error of the models.

Note however, that it is not a guarantee that the ensemble is more accurate than the best model in the ensemble, but only as accurate as the models, on average. Therefore, besides being as independent as possible, it is important that the models used to build the ensemble are sufficiently accurate.

4.1.2 Ensemble weight initialization

Besides taking a simple average of the models, it is also possible to take a weighted linear combination based on some criterion that measures the quality of the models.

Two different ensemble weight initializations are investigated in the publications in this thesis: uniform weight initialization (Publication I) and leave-one-out weight initialization (Publication II).

Uniform weight initialization For initialization of the ensemble model, each of the individual models is trained on a given training set, and initially each model contributes with the same weight to the output of the ensemble. This will be referred to as *uniform weight initialization*.

Leave-one-out weight initialization As an alternative to uniform weight initialization, the initial weights can be based on the leave-one-out output of the models on the training set, like in Breiman (1996b). This will be referred to as *leave-one-out weight initialization*.

Using Equation 2.9, for a any model that is linear in the parameters (like ELM), the leave-one-out errors can be efficiently computed. Therefore, the leave-one-out outputs (i.e. the estimations of the sample that is left out in each of the N folds) can be obtained efficiently as well, given the leave-one-out errors and the true targets.

Finally, the initial ensemble weights are obtained by fitting a non-negative linear combination of the leave-one-out outputs for all m models to the target outputs. Using this procedure, models that have bad generalization performance get relatively low weight, while models with good generalization performance get higher weights.

4.1.3 Ensembling strategies

It was shown in (Hansen and Salamon, 1990), that through combining multiple neural networks, accuracy can be improved as compared to the individual neural networks. Since, several strategies have been proposed for building ensembles. In *mixture of experts* (Jacobs et al., 1991), several models are built, each of which specializes on part of the problem domain. The weights of the ensemble model depend on the part of the domain in which a prediction is required. In *stacking* (Wolpert, 1992) and *boosting* (Freund and Schapire, 1996; Schapire et al., 1998), the models are built in sequence, taking into account the performance of the earlier built models, in order to improve on them. In bootstrap aggregating, or *bagging* (Breiman, 1996a), on the other hand, diversity between the models is obtained by training them on re-sampled versions of the training set, while in *stacked regressions* (Breiman, 1996b), leave-one-out cross-validation is used to obtain the ensemble weights. Finally, in (Liu and Yao, 1999) accuracy of a neural network ensemble is enhanced through negative correlation learning, which promotes diversity between the neural networks. For an overview of ensemble methods in general, see (Bishop, 2006; Murphy, 2012).

In the next section, adaptive ensemble models, and the contributions made in this thesis are discussed.

4.2 Adaptive ensemble models

When solving a particular regression or classification problem, it is often unknown in advance what the optimal model class and structure is. One alternative for selecting the optimal model class or structure would be through validation, cross-validation or leave-one-out validation, as discussed in Section 2.2.2. However, it is not guaranteed that the model selected based on a set of training samples will be the best model for newly obtained samples. For example, in a nonstationary context where the i.i.d. assumption does not hold and the information gathered from past samples can become inaccurate.

One strategy for handling nonstationarities would be to keep learning as new samples become available. For example, by retraining the model repeatedly on a finite window into the past such that it 'tracks' the nonstationarity.

Another strategy for adapting to nonstationarities is to use a strategy similar to (Jacobs et al., 1991), but instead of letting the linear combination of models depend on the part of the input space, let the linear combination directly depend on the accuracy of the models.

4.2.1 Adaptive ensemble model of ELMs

In Publication I, both strategies are investigated in one-step ahead prediction on both stationary and nonstationary time series, in which the next value of the time series is predicted, given all its past values.

Besides the already mentioned advantages of ensemble models over single models, this allows for adaptivity of the ensemble model to environmental changes.

Related Work The retraining of the ELMs in this ensemble is similar to the Online Sequential ELM (OS-ELM) (Liang et al., 2006), the important difference being that contrary to OS-ELMs, Publication I also provides a way to incrementally remove samples from the trained model. Furthermore, compared to the ensemble of OS-ELM (EOS-ELM) (Lan et al., 2009), which was introduced around the same time as Publication I, the adaptive ensemble adjusts the linear combination to optimize ensemble accuracy.

Both the ability to train on sliding windows, and the adaptive ensemble weights turn out to be an important contribution in the nonstationary environments, in which online sequential learning is typically applied.

Initializing the adaptive ensemble model The adaptive ensemble model consists of a number of randomly generated ELMs, which each have their own parameters. Because of the stochastic nature of the ELM, they are diverse in nature and will have different biases and input layer weights. To further increase diversity between the models, each ELM is built using different regressor variables; different regressor size; and different number of hidden neurons. Uniform weight initialization is used for the

Algorithm 3 Adaptive Ensemble of ELMs (Publication I)

```
Given $(\mathbf{x}(t), y(t))$, $\mathbf{x}(t) \in \mathbb{R}^d$, $y(t) \in \mathbb{R}$, and $m$ models:
 1: Create and train $m$ random ELMs: $(ELM_1 \dots ELM_m)$
 2: Initialize each $w_i$ to $\frac{1}{m}$
 3: while $t < t_{end}$ do
 4:     generate predictions $\hat{y}_i(t+1)$
 5:     $\hat{y}_{ens}(t+1) = \sum_i w_i \hat{y}_i(t+1)$
 6:     t = t + 1
 7:     compute all errors $\rightarrow \epsilon_i(t-1) = \hat{y}_i(t-1) - y(t-1)$
 8:     for i = 1 to #models do
 9:         $\Delta w_i = -\epsilon_i(t-1)^2 + mean(\epsilon(t-1)^2)$
10:         $\Delta w_i = \Delta w_i \cdot \alpha / (\#models \cdot \mathbf{var}(y))$
11:         $w_i = \max(0, w_i + \Delta w_i)$
12:         Retrain $ELM_i$
13:     end for
14:     renormalize weights $\rightarrow \mathbf{w} = \mathbf{w} / \|\mathbf{w}\|$
15: end while
```

ensemble weights.

Adapting the linear combination of models On the one hand, a number of different models are combined in a single ensemble model and the weights with which these models contribute to the ensemble are adapted based on their performance (see Algorithm 3 for details). The speed of the change can be controlled by a learning rate α.

The idea behind the algorithm is that as the time series changes, a different model will be more optimal to use in prediction. By monitoring the errors that the individual models in the ensemble make, a higher weight can be given to the models that have good prediction performance for the current part of the time series, and a lower weight can be given to the models that have bad prediction performance for the current part of the time series.

Figure 4.2 illustrates the resulting adaptation of the ensemble weights, during the task of one-step ahead prediction on two different time series.

Retraining the models On the other hand, Publication I explores the effect of different ways of retraining the models as new data becomes available: before making a prediction for time step t, each model is either retrained on a past window of n values $(\mathbf{x}_i, y_i)_{t-n}^{t-1}$ (sliding window), or on all values known so far $(\mathbf{x}_i, y_i)_1^{t-1}$ (growing window), using the recursive

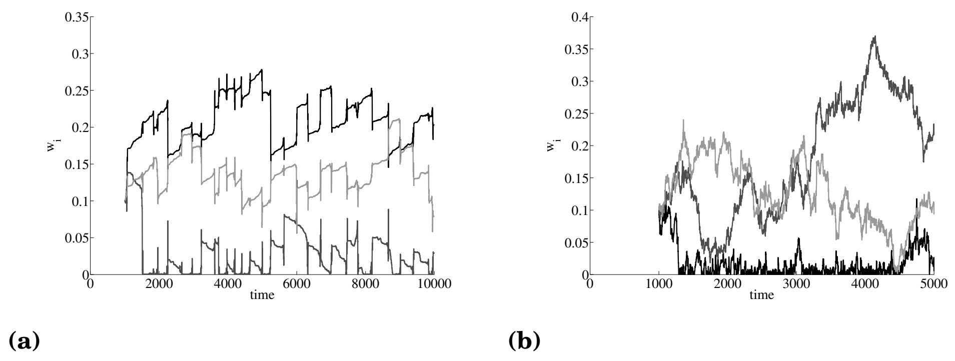


Figure 4.2. Plots showing part of the ensemble weights w_i adapting over time during sequential prediction on (a) Laser time series and (b) Quebec Births time series (learning rate=0.1, number of models=10) (Publication I)

least-squares algorithm as defined by (Bierman, 1977; Bontempi et al., 1998).

Through this algorithm samples can be incrementally added to an already trained linear model, which will result in the linear model that would have been obtained, had it been trained on the modified training set. Since an ELM is essentially a linear model of the responses of the hidden layer, it can be applied to (re)train the ELM quickly in an incremental way on a sliding window or a growing window and it can adapt.

Adding a sample to a linear model Suppose a linear model is trained on k samples of dimension d, with solution $\beta(k)$, and have $\mathbf{P}(k) = (\mathbf{X}^T\mathbf{X})^{-1}$, which is the $d \times d$ inverse of the covariance matrix based on k samples, then the solution of the model with added sample $(\mathbf{x}(k+1), y(k+1))$ can be obtained by

$$\begin{aligned}
&\mathbf{P}(k+1) = \mathbf{P}(k) - \tfrac{\mathbf{P}(k)\mathbf{x}(k+1)\mathbf{x}^T(k+1)\mathbf{P}(k)}{1+\mathbf{x}^T(k+1)\mathbf{P}(k)\mathbf{x}(k+1)},\\
&\gamma(k+1) = \mathbf{P}(k+1)\mathbf{x}(k+1),\\
&\varepsilon(k+1) = y(k+1) - \mathbf{x}^T(k+1)\hat{\boldsymbol{\beta}}(k),\\
&\hat{\boldsymbol{\beta}}(k+1) = \hat{\boldsymbol{\beta}}(k) + \gamma(k+1)\varepsilon(k+1)
\end{aligned} \tag{4.7}$$

where $\mathbf{x}(k+1)$ is a $1 \times d$ vector of input values, $\beta(k+1)$ is the solution to the new model and $\mathbf{P}(k+1)$ is the new inverse of the covariance matrix on the $k+1$ samples (Bierman, 1977; Bontempi et al., 1998).

Removing a sample from a linear model Similarly, a sample can be removed from the training set of a linear model giving the linear model that would have been obtained, had it been trained on the modified training set. In this case, the new model with removed sample $(\mathbf{x}(k), y(k))$ can be obtained by

$$\begin{aligned}
&\gamma(k-1) = \mathbf{P}(k)\mathbf{x}(k),\\
&\varepsilon(k-1) = y(k) - \tfrac{\mathbf{x}(k)\hat{\boldsymbol{\beta}}(k)}{1-\mathbf{x}^T(k)\mathbf{P}(k)\mathbf{x}(k)},\\
&\mathbf{P}(k-1) = \mathbf{P}(k) - \tfrac{\mathbf{P}(k)\mathbf{x}(k)\mathbf{x}^T(k)\mathbf{P}(k)}{1+\mathbf{x}^T(k)\mathbf{P}(k)\mathbf{x}(k)},\\
&\hat{\boldsymbol{\beta}}(k-1) = \hat{\boldsymbol{\beta}}(k) - \gamma(k)\varepsilon(k)
\end{aligned} \tag{4.8}$$

where $\boldsymbol{\beta}(k-1)$ is the solution to the new model and $\mathbf{P}(k-1)$ is the new inverse of the covariance matrix on the $k-1$ samples (Bierman, 1977; Bontempi et al., 1998; van Heeswijk, 2009).

4.2.2 Experiments

In the experiments of Publication I, the adaptive ensemble model of Extreme Learning Machines (ELMs) is applied to the problem of one-step ahead prediction on both stationary and nonstationary time series. It is verified that the method works on stationary time series, and the adaptivity of the ensemble model is verified on nonstationary time series.

Data The stationary data used in the experiments is the Santa Fe Laser Data time series (Weigend and Gershenfeld, 1993), which has been obtained from a far-infrared-laser in a chaotic state. This time series has become a well-known benchmark in time series prediction since the Santa Fe competition in 1991. It consists of approximately 10000 points. The non-stationary data used in the experiments is The Quebec Births time series[1], which records the number of daily births in Quebec over the period of January 1, 1977 to December 31, 1990. It consists of approximately 5000 points, is nonstationary and more noisy than the Santa Fe Laser Data.

Experimental parameters The adaptive ensemble model is trained on the first 1000 values of the time series, after which sequential one-step ahead prediction is performed on remaining values. This experiment is repeated for various combinations of learning rate α and number of models in the ensemble. Each ELM has a regressor size (of which a number of variables are randomly selected) and between 150 and 200 hidden neurons with a sigmoid transfer function. See Publication I for more details.

Effect of number of models and learning rate on accuracy

[1]`http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/misc/qbirths.dat`

Stationary data Figure 4.3 shows the effect of the number of models and the learning rate, which controls how quickly the ensemble weights adapt, on the prediction accuracy in the stationary case. It can be seen that the number of models strongly influences the prediction accuracy and that at least 40 models are needed to get good prediction accuracy. Furthermore, despite the data being stationary, it can be seen that a non-zero learning rate helps in adapting the linear combination of the models for optimal performance, presumably because the uniform initialization of the ensemble weights is sub-optimal.

Nonstationary data Figure 4.4 shows the effect of the number of models and the learning rate on the prediction accuracy in the nonstationary case. Again, it can be seen that the number of models strongly influences the prediction accuracy and that, compared to the stationary case, more models are needed to get good prediction accuracy.

Effect of retraining strategy on accuracy The influence of the various (re)training strategies can be found in Table 4.1.

As is to be expected, for the stationary data, optimal results are obtained for retraining on a growing window. For the nonstationary data, as expected, a sliding window is optimal.

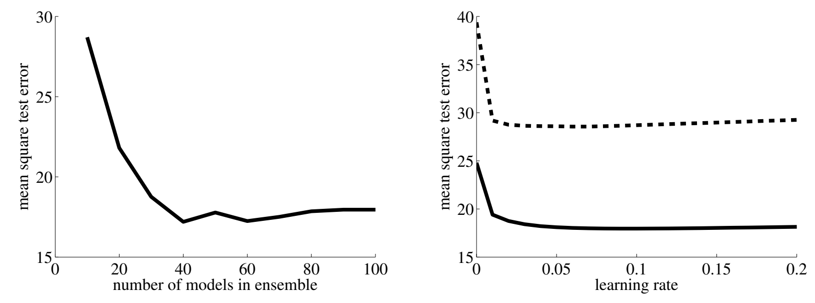
Discussion In general, the results of Publication I suggest the following strategy for obtaining robust models:

1. the more models are included in the ensemble, the more accurate it generally is (although there are diminishing returns).

2. a small learning rate of around 0.1 is optimal.

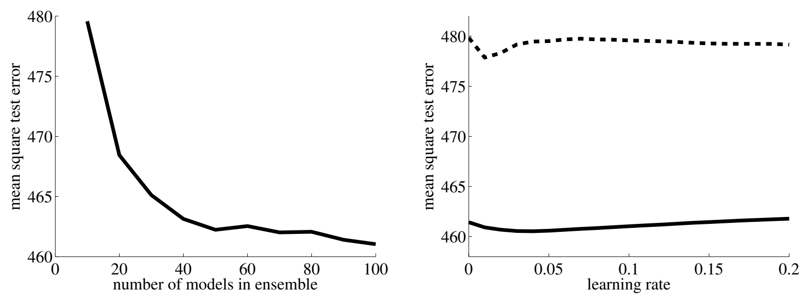
3. individual models should be retrained according to expectations/expertise.

The retraining strategy of the individual models affects the accuracy and often it is not known whether data is stationary or nonstationary, nor is the optimal sliding window size known. Future work could therefore include the investigation of ensembles consisting of models with varying retraining strategies and window sizes. Furthermore, to save computational resources, models that contribute little to the accuracy of the ensemble could be pruned. Vice versa, new models could be added to the ensemble when needed.



(a) **(b)**

Figure 4.3. MSE$_{test}$ of ensemble on **Santa Fe Laser Data** time series for varying number of models (no window retraining, learning rate 0.1), and as a function of learning rate (no window retraining), for 10 models (*dotted line*) and 100 models (*solid line*)



(a) **(b)**

Figure 4.4. MSE$_{test}$ of ensemble on **Quebec births** time series for varying number of models (no window retraining, learning rate 0.1), and as a function of learning rate (no window retraining), for 10 models (*dotted line*) and 100 models (*solid line*)

		retraining		
	learning rate	none	sliding	growing
laser	0.0	24.80	33.85	20.99
	0.1	17.96	27.30	**14.64**
qbirths	0.0	585.53	**461.44**	469.79
	0.1	567.62	**461.04**	468.51

Table 4.1. MSE$_{test}$ of ensemble of 100 ELMs for laser (training window size 1000)

4.3 GPU-accelerated and parallelized ELM ensembles

A big advantage of ensemble models is their modularity. Publication II exploits this modularity of ensembles and presents an ensemble of GPU-accelerated ELMs, that is accelerated in three distinct ways:

1. multiple individual models are trained in parallel across multiple graphics processor units (GPUs) and CPU cores.

2. the training and model structure selection procedures are accelerated by using the GPU.

3. the model structure selection is performed in an efficient way by use of PRESS statistics, while explicitly computing and reusing the pseudo-inverse of $\mathbf{H}^\dagger$, where $\mathbf{H}$is the hidden layer output matrix.

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. The next sections will highlight the contributions and experimental results of Publication II.

4.3.1 Parallelization across multiple cores

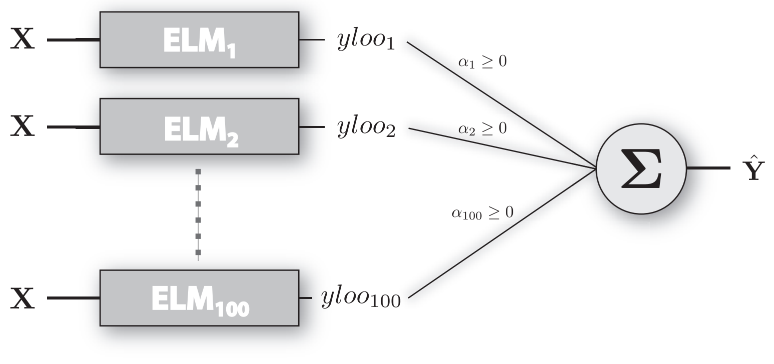


Figure 4.5. Block diagram showing the overall setup of the ensemble of ELMs. (Publication II)

Figure 4.5 illustrates the ELM ensemble of Publication II, and it can be seen that the ELMs in the ensemble can be built independently. Therefore, the running time of the ensemble can be optimized by dividing the

ELMs across multiple CPUs and GPUs and preparing them in parallel. This is achieved using MATLAB's Parallel Computing Toolbox (MathWorks, 2011), which allows the creation of 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, in case of an ensemble of 100 ELMs and 4 workers, each of the workers builds 25 ELMs.

Although in Publication II the parallelized ensemble model was not executed across multiple computers, it can be executed on multiple computers by using the MATLAB Distributed Computing Toolbox, for which at the time of writing Publication II no license was available.

4.3.2 GPU-acceleration of required linear algebra operations

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.

Available libraries There exist several software libraries aimed at speeding up a subset of the linear algebra functions found in LAPACK (Anderson et al., 1999):

- CULA Tools (Humphrey et al., 2010): a library introduced in October 2009, implementing a subset of LAPACK functions. It was the first widely available GPU-accelerated linear algebra package, and developed in cooperation with NVidia. Because of NVidia's investment in general-purpose GPU computing, this library is likely to remain well-supported. The free variant of this package contains functions for solving a linear system (`culaGesv`), and for computing the least-squares solution to a linear system (`culaGels`).

- MAGMA (Agullo et al., 2009): a linear algebra package, developed by the creators of the widely used LAPACK, aiming at running linear algebra operations on heterogeneous architectures (i.e. using both multicore CPU and multiple GPUs present on the system, in order to solve a single problem).

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

Table 4.2. An overview of the various functions used. (Publication II)

Solving linear systems on CPU vs GPU In Publication II, CULA Tools is used for accelerating the linear algebra operations. Specifically, the (`culaGesv`) and (`culaGels`) functions are 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.

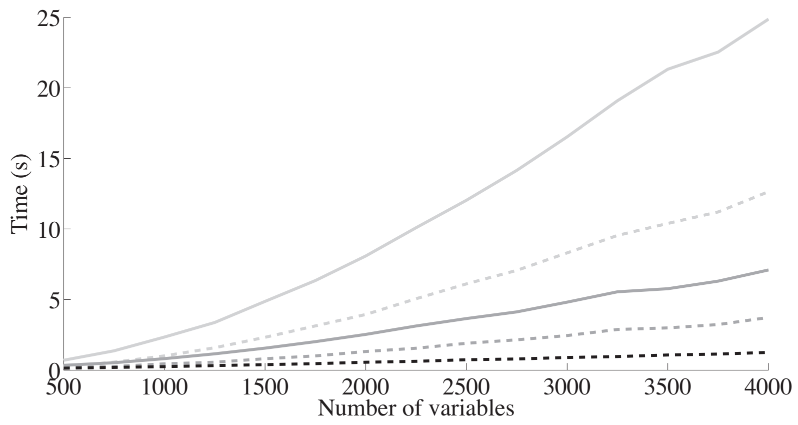
Relevant functions An overview of all the functions used in Publication II can be found in Table 4.2. Since in Publication II all linear systems are fully determined, and involve square regressor matrices, the functions give exactly the same result and only vary in running time.

Single vs. double-precision Even though double-precision calculations are possible on GPU, they are faster at performing single-precision calculations. In particular, for the NVidia GTX295 cards that were used in Publication II, the single-precision performance is 8 times higher than the double-precision performance. In NVidia's latest generation of video cards, this gap in performance is smaller, but still existent. Therefore, it is beneficial to use single-precision calculations wherever numerically possible.

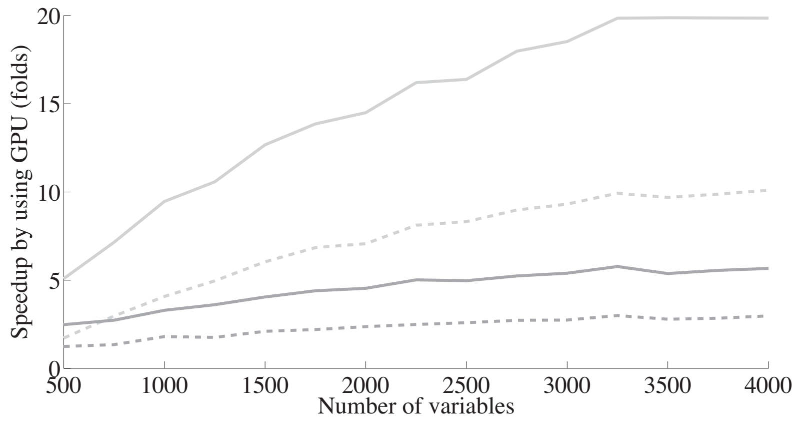
Performance comparison In order to get an idea of the running time of the function `culaGels`, it is compared with MATLAB's commonly used `mldivide` (also known as \), as well as with the `gels` function from MATLAB's underlying highly optimized multi-threaded LAPACK library[2]. For a fair comparison, and 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.

Figure 4.6a and Figure 4.6b summarize the results. As expected, it can be seen that the precision greatly affects the performance. Also, MAT-

[2]MATLAB r2009b was used, which utilizes the highly optimized MKL library by Intel on the Core i7 920 used for the experiments.



(a) Time (s) needed to solve linear system in double-precision (*solid lines*) and single-precision (*dashed lines*).



(b) Speedup of `culaGels` over other functions in double-precision (*solid lines*) and single-precision (*dashed lines*).

Figure 4.6. Performance comparison of functions for solving a linear system of 5000 samples and one target variable: `mldivide` (*light-gray lines*), `gels` (*gray lines*), `culaGels` (*black line*) (Publication II).

LAB's underlying LAPACK function `gels` perform much better than the commonly used `mldivide`. Finally, the GPU-accelerated function `culaGels` offers the fastest performance of all.

Finally, for square matrices, `culaGesv` and `gesv` (not pictured) are slightly faster than `gels` and `culaGels` and are therefore used in Publication II for slightly higher performance.

4.3.3 Efficient leave-one-out computation

Model structure selection allows for determining the optimal structure for the ELM model, where by optimal structure often the number of hidden neurons is meant. Besides the number of hidden neurons, also other parameters of the ELM and the used training algorithm can be cross-validated this way. This is done using a criterion which estimates the model generalization capabilities for the varying numbers of neurons in the hidden layer and the different other possible values for model parameters considered.

Minimal overhead in LOO computation by re-using pseudo-inverse Recall from Equation 2.9, that the HAT matrix needed in the computation of the leave-one-out error largely consists of the Moore-Penrose generalized inverse of the regressor matrix. Using the notations of ELM, HAT $= \mathbf{H}\mathbf{H}^\dagger$. Therefore, instead of just computing the solution to the linear system while training the ELM, combined training and leave-one-out computation 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 leave-one-out error.

Furthermore, since only the diagonal of the HAT-matrix is needed, it is sufficient to compute only the diagonal by taking 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. Therefore, the computation of the leave-one-out error then comes at a very low overhead once the pseudo-inverse is already computed.

Figure 4.7 illustrates this by comparing the running times for training and combined training and leave-one-out computation. It can be seen that although by explicitly computing $\mathbf{H}^\dagger$, the training procedure becomes somewhat slower, due to the re-use of $\mathbf{H}^\dagger$ in the leave-one-out-computation, combined training and leave-one-out computation can be done about as fast as just training the model.

Incidentally, because leave-one-out cross-validation virtually comes for free after training, it is a great alternative to using information criteria like AIC (Akaike, 1974) and BIC (Schwarz, 1978), which are often used in situations where cross-validation would be prohibitively slow.

The algorithm for fast training and leave-one-out-based model structure selection of ELM can then be summarized as in Algorithm 4. Although this particular example is for cross-validating the number of hidden neurons, the same approach can be used when cross-validating for other combinations of model parameters. Also, in case the neurons would be sorted by relevance first, the algorithm corresponds to OP-ELM (Miche et al., 2010). In case also an L2-regularization parameter is optimized at each number of neurons considered, this corresponds to TROP-ELM (Miche et al., 2011).

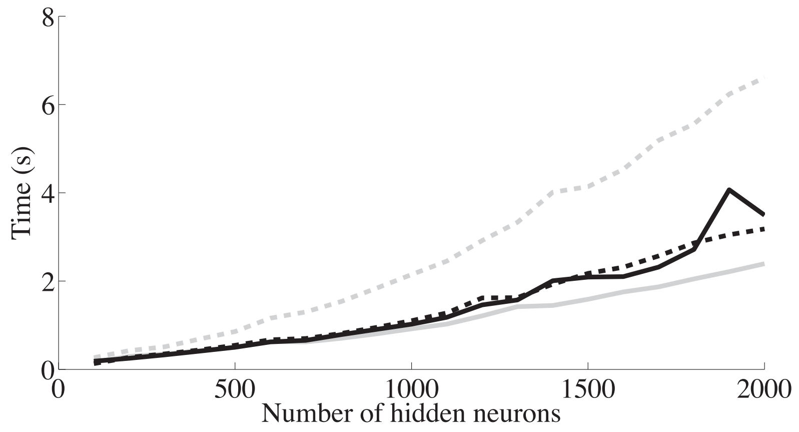


Figure 4.7. Comparison of running times of ELM training (*solid lines*) and ELM training + leave-one-out-computation (*dotted lines*), with (*black lines*) and without (*gray lines*) explicitly computing and reusing $\mathbf{H}^\dagger$ (Publication II)

Algorithm 4 Efficient LOO cross-validation of the number of neurons for an ELM (Publication II)

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:      - Given 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 $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{Y}$;
9:      Compute $E_{loo}$:
10:     - Compute $\mathrm{diag}(\mathrm{HAT})$ (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 $E_{loo,j}$;
```

4.3.4 Experiments

In Publication II, an ensemble model of ELMs is built for solving two relatively large regression problems based on one-step-ahead time series prediction. The model structure selection and training of the ELMs is accelerated using GPU, and the construction of the ensemble is parallelized by distributing the work over multiple CPUs and GPUs. The influence of the GPU-acceleration and parallelization is measured, and scalability of the approach is verified on two data sets.

Data The first data set is again the Santa Fe Laser Data time series (Weigend and Gershenfeld, 1993), which consists of approximately 10000 samples.

The second data set is the ESTSP'08 Competition data set number 3[3], which consists of approximately 30000 samples, and is computationally also more challenging due to the size of the regressor needed (Olteanu, 2008; Kourentzes and Crone, 2008).

Experimental parameters The samples from the time series are obtained using respectively 12, and 168 time steps for the Santa Fe Laser data and ESTSP'08 Competition data, after which the data is randomly divided into 85% for training and 15% for testing.

The ELMs have their number of neurons optimized using efficient leave-one-out cross-validation and for diversity, each ELM uses random variables from the regressor. The ensemble weights are determined through leave-one-out initialization and remain fixed.

The effect of GPU-acceleration and parallelization on the performance is consequently measured by varying the function used in model structure selection and training, as well as varying the number of MATLAB workers (i.e. threads). See Publication II for more details.

Effect of GPU-acceleration and used function Table 4.3 summarizes the results of the experiments performed in Publication II, and clearly shows the effect of the GPU-acceleration and the used function.

It can be seen that generally `mldiv` and `gesv` achieve similar performances, while both the use of single-precision and the use of GPU significantly speed up the ensemble.

Effect of number of workers on computational time Table 4.3 and Figure 4.8 summarize the influence of the number of MATLAB workers on the ensemble performance.

When not using any explicit parallelization through MATLAB workers (i.e. the line $N = 0$ in Table 4.3), the differences between using CPU and GPU are relatively modest. This is due to the fact that for `mldiv` and `gesv`, MATLAB automatically parallelizes the computation over the

[3]available from `http://research.ics.aalto.fi/eiml/datasets.shtml`

multiple CPU cores of the Intel Core i7 920, and can therefore keep up quite well with the single GPU being used for `culaGesv`.

However, when explicitly parallelizing the computation over N cores and GPUs, the difference between CPU and GPU significantly increases.

Overall, 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(`mldiv`$_{dp}$) / t(`culaGesv`$_{sp}$)).

Effect of number of models on accuracy Finally, Figure 4.9 shows the influence of the number of models in the ensemble on the accuracy of the ensemble. Similarly to Publication I, the results show that the more models are added to the ensemble, the more accurate it gets.

Discussion This fact alone (that more models is generally better in ensemble models), is additional justification for the approach proposed in Publication II. By parallelization and GPU-acceleration of the ensemble, it is possible to train more models and to train them faster, which results in a more accurate model that can be obtained in less time.

Some interesting questions to explore therefore would be: how fast can an ensemble model of a particular accuracy be obtained? Of course the easiest way to speed up the ensemble would be to evaluate each model on its own dedicated node, but even then, would it be possible to further speed up the ensemble model while retaining or improving accuracy?

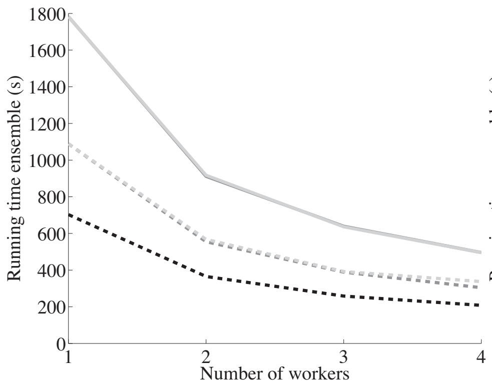
One possibility would be to parallelize other parts of the ELM, like the multiplication of the hidden layer weights and the inputs, as has been done in (He et al., 2011) and (He et al., 2013).

Another interesting direction to explore would be to make the ELMs more effective and accurate by an altered weight scheme (which is explored in Publication V), or by trading off speed for accuracy in the individual ELMs (which is explored in Publication VI) and ensembling more of them.

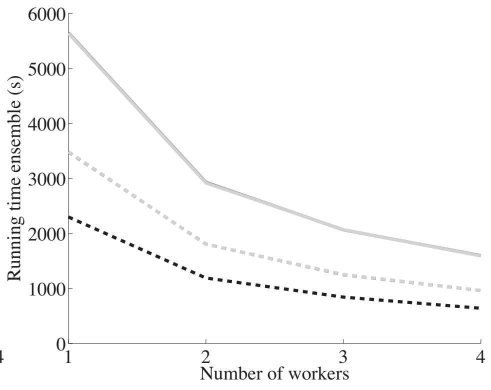
Before giving an overview of those publications though, the next chapter will highlight some contributions of the thesis to variable selection.

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

Table 4.3. 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) (Publication II).

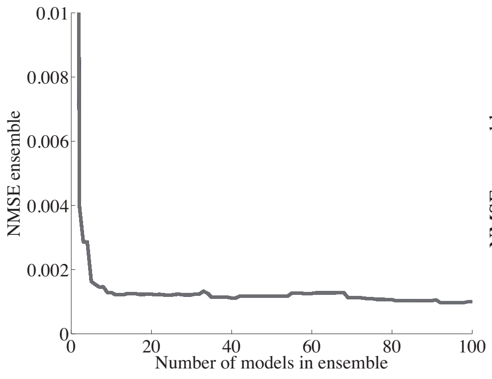


(a)

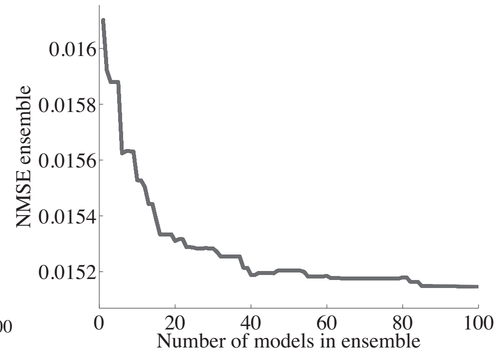


(b)

Figure 4.8. Running times (in seconds) for running the entire ensemble in parallel on (a) Santa Fe, (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*) (Publication II).



(a)



(b)

Figure 4.9. NMSE of an ensemble model with varying number of models, on (a) Santa Fe, (b) ESTSP'08 (Publication II).

5. Variable Selection

"And what is good, Phaedrus,
And what is not good–
Need we ask anyone to tell us these things?"

– Robert M. Pirsig,
Zen and the Art of Motorcycle Maintenance

Variable selection is a central issue in machine learning. The higher the dimensionality of the data, the more samples are needed to reliably train a model. This is sometimes referred to as the curse of dimensionality. Therefore, given the often limited number of training samples, it is important that the dimensionality is sufficiently reduced (without losing too much information) such that a reliable model can be trained.

This chapter introduces the basic concepts of variable selection, and highlights the related contributions of this thesis: an ELM-based method for variable selection (Publication III); a (multi-)GPU-accelerated Delta Test criterion used as criterion in a parallelized genetic algorithm for variable selection (Publication IV); and finally, a new weight scheme for ELM that results in more effective and efficient neural networks, and makes the ELM more robust to irrelevant and noisy variables (Publication V).

5.1 Variable selection

5.1.1 Motivation

Due to the technological developments of the past decades, it is easier than ever to gather large amounts of data. The grand challenge, then, is to extract relevant information from this data in order to gain useful knowledge.

Part of that challenge is being able to scale the methods to the size of modern data sets (which was partly the focus of Chapter 4). However, besides challenges in scaling due to the large amount of samples in modern data sets, another challenge is posed by their high dimensionality. Namely, the high dimensionality of modern data sets poses a problem when trying to train reliable and accurate models due to the curse of dimensionality (Bellman, 1961; Verleysen et al., 2003): the number of samples required to be able to train an accurate model scales exponentially with the dimensionality of the input space (i.e. the dimensionality of the data). Therefore, it is important to reduce the dimensionality as much as possible, so that more accurate models may be trained.

5.1.2 Dimensionality reduction

Feature extraction Although the data is high-dimensional, it is often not evenly spread throughout the input space. Instead, the samples will lie on some lower-dimensional manifold embedded in that high-dimensional space. Therefore, one strategy for handling high-dimensional data is by dimensionality reduction through *feature extraction*. Here, the goal is to find an alternate representation of the data by extracting its latent features, and representing the data in terms of these features. For example, if the data lies linear in some linear subspace, then Principal Component Analysis (PCA) can be used to obtain a more compact representation (Jolliffe, 2002). Note however, that this by itself does not take into account the relevance of a latent feature for predicting the output, as it is unsupervised.

Variable selection Contrary to unsupervised feature extraction, v*ariable selection* (Guyon and Elisseeff, 2003) is supervised and does take into account the relevance of a variable for predicting the output[1]. Through a search algorithm, the variable subset is optimized according to a statistical criterion measuring the quality of the variable subset.

Besides improving model accuracy, another motivation for performing variable selection may be interpretability of the models and gaining a better understanding about the problem at hand. For example, in gene

[1]Of course, feature extraction and variable selection can be combined, and extracted features can become input variables to some model. In this work, it is assumed that feature extraction has already taken place, and the focus is on selecting the best variable subset, given a set of input variables. The terms variable selection and feature selection will be used interchangeably.

expression analysis, variable selection can help identifying those genes that are relevant for predicting whether a patient is sick, and are therefore likely to play a role in the disease itself (Guyon and Elisseeff, 2003).

Finally, sometimes it is very expensive to obtain new samples and measure particular variables. Therefore, through variable selection, money and effort can be saved by only measuring and collecting the most relevant variables.

5.1.3 Variable selection methods

Approaches Several main approaches for variable selection are distinguished in (Guyon and Elisseeff, 2003): wrapper methods, filter methods, and embedded methods, which will be described shortly below.

Filter approach In filter approaches, as the name suggests, the variable subset is filtered before being passed on to a model for learning, and some criterion is used to evaluate the quality of a particular subset of variables.

For example, as a criterion, mutual information (MacKay, 2003; Kraskov et al., 2004) can be used to estimate how much information a particular variable subset contains about the targets (Rossi et al., 2006; François et al., 2007; Verleysen et al., 2009). The mutual information criterion was used in the baseline variable selection experiments of Publication III.

Another criterion that can be used for variable selection is the Delta Test (Eirola et al., 2008; Eirola, 2014; Sovilj, 2014). The Delta Test is a noise variance estimator, which indicates the performance a non-linear model can possibly attain, given particular data. The Delta Test criterion was used in Publication IV.

As a final example, a filter method based on Least Angle Regression (LARS) (Efron et al., 2003) could rank the variables according to relevance, and return a subset of the most relevant variables. This approach is included in the baseline experiments of Publication III.

Wrapper approach Whereas filter strategies use some statistical criterion for variable selection, in the wrapper strategy, the model is used directly for evaluating the quality of a variable subset. For example, an Extreme Learning Machine could be built with many different variable subsets, and have its generalization performance estimated using fast leave-one-out cross-validation. The variable subset that achieves minimum leave-one-out error will then be used for building the final ELM. Therefore, it can be seen as some sort of model structure selection. The advantage of wrapper strategies is that, since the actual model is used, the feature subset is likely to be better-optimized for that model. However, since the actual model needs to be trained, depending on the model, and the fact that there might not be a way to efficiently estimate the generalization performance, wrapper strategies can be computationally very expensive. Furthermore, care should be taken not to overfit during the feature selection process (Reunanen, 2003).

Embedded approach In the embedded approach, the learning machine itself incorporates variable selection as part of its training, and selection is not performed explicitly using a wrapper or filter approach. Although it could be argued that explicit variable selection in the training algorithm is actually model structure selection using a wrapper approach (rather than training and an embedded approach), it is useful to distinguish the embedded variable selection approach from filter and wrapper approaches, as the learning method might implicitly perform variable selection as a result of its structure (for example, like in the Binary ELM and Ternary ELM proposed in Publication V).

Search methods Besides a statistical criterion like mutual information, or an estimated performance of a model, a method is needed to explore the solution space of all possible variable selections in a systematic way, especially considering that the number of possible variable subsets is exponential in the number of variables.

One possibility is to use a local search method and, starting from a particular solution in the solution space, optimize the solution by repeatedly evaluating and jumping to neighboring candidate solutions, until an optimum is found. For example, Forward selection (Hastie et al., 2001) starts from an empty variable subset, and repeatedly adds the variable that improves the criterion the most. Similarly, Backward selection (Hastie et al., 2001) starts from the full variable subset, and repeatedly removes the variable that deteriorates the criterion the least. Finally, Forward-Backward selection allows both adding and removing variables at each step (Hastie et al., 2001). These local search methods are used in the baseline experiments of Publication III, in combination with the mutual information criterion.

Another possibility, which is explored in Publication IV, is to use a genetic algorithm to explore the space of possible solutions. Here, the candidate solutions are encoded in a population of individuals and consequently optimized through an evolution-like algorithm.

The remaining part of this chapter will give an overview of the contributions to variable selection methods that are made as part of this thesis.

5.2 ELM-FS: ELM-based feature selection

Publication III explores a feature selection method based on Extreme Learning Machine, which returns a complete feature selection path, representing the trade-off between the best feature subset for each subset size and the corresponding estimated generalization error. This allows the practitioner to make an informed decision about the feature subset that is best for the current context. For example, if sparsity is most important due to the difficulty of obtaining new samples, the smallest subset giving reasonable performance could be identified and selected. If, on the other hand, accuracy is more important, that feature subset giving best accuracy can be selected. The method is shown to be competitive with traditional feature selection methods, and can be used for ELM (as a wrapper method), or as an efficient filter method for more complicated non-linear machine learning methods.

5.2.1 Feature selection using the ELM

Introducing a scaling layer In order to use the Extreme Learning Machine for feature selection, it is first augmented with a scaling layer such that each input variable is multiplied with its own particular scale (i.e. for sample x_i, $x_{i1} \to s_1 x_{i1}, \dots, x_{id} \to s_d x_{id}$). Then, for regression, the feature selection problem for obtaining the feature subset that obtains minimum training error, can be written as

$$\min_{s,\theta} \frac{1}{n} \sum_{i=1}^{n} \left[y_i - f\left(s_1 x_{i1}, \dots, s_d x_{id} | \theta\right)\right]^2 s.t. \left\|\mathrm{s}\right\|_0 \leq d \tag{5.1}$$

where $f\left(s_1 x_{i1}, \dots, s_d x_{id} | \theta\right)$ stands for the ELM with parameters θ, and inputs $s_1 x_{i1}, \dots, s_d x_{id}$. Here, the scaling vector s is a vector of binary variables such that $s_i \in \{0, 1\}$, and $\|s\|_0$ is the L_0-norm of s (i.e. the number of non-zero scaling factors and therefore the number of selected variables).

Now, Equation 5.1 can be rewritten as a regularization, i.e.

$$\min_{s,\theta} \frac{1}{n} \sum_{i=1}^{n} \left[y_i - f\left(s_1 x_{i1}, \dots, s_d x_{id} | \theta\right)\right]^2 + C_0 \left\|s\right\|_0 , \tag{5.2}$$

for some regularization constant C_0, that can be used to control the size of the feature subset.

Relaxing the feature selection problem However, even though solving this optimization problem while gradually increasing C_0 would result in increasingly sparse optimal variable subsets, there are too many subsets to exhaustively evaluate all of them. Therefore, a common strategy in optimizing combinatorial problems is used. Namely, the feature selection problem is made easier through a technique known as relaxation (Burke and Kendall, 2005) and a regularization scheme is used to enforce sparsity: the binary scalings $s_i \in \{0,1\}$ are replaced by real $\tilde{s}_i \in [0,1]$, and the L_1-norm is used instead of the L_0-norm, which results in

$$\min_{\tilde{s},\theta} \frac{1}{n} \sum_{i=1}^{n} \left[t_i - f\left(\tilde{s}_1 x_{i1}, \ldots, \tilde{s}_d x_{id} | \theta\right)\right]^2 + C_1 \left\|\tilde{s}\right\|_1 . \tag{5.3}$$

This relaxed form of the optimization problem is easier to solve, since it is differentiable.

Solving the relaxed feature selection problem Since it is possible to obtain the gradient of the training error with respect to the scaling vector $\tilde{s}$, the algorithm can repeatedly take a small gradient descent step, updating $\tilde{s}$ such that the objective value of the optimization problem is improved. Therefore, for a given regularization parameter C_1, the problem can be efficiently solved.

However, there is the potential problem of plateaus where too small a step size would result in slow convergence towards the (local) optimum. Therefore, the scaling parameter $\tilde{s}$ is discretized and optimization of $\tilde{s}$ is performed on a hyper-grid instead. This guarantees termination of the algorithm, and limits the amount of steps the algorithm takes.

Now, optimal feature scalings of varying sparsity can be obtained by starting from a random initial $\tilde{s}$ and $C_1 = 0$, and repeatedly jumping to the neighbor pointed to by the gradient, until stuck in a local minimum.

At this point, the generalization error of that feature selection (each non-zero scaling indicating a feature being selected) can be estimated through fast leave-one-out error computation.

Obtaining a feature selection path and corresponding error estimates After getting stuck in a local optima, the regularization factor can be increased until the local minimum disappears, and minimization continues towards sparser scalings and eventually empty scalings.[2] Finally, by re-

[2]This sounds complicated, but it can be intuitively understood by seeing it as dropping a ball on a hilly landscape mountain, and letting it roll down. Whenever

peating this procedure many times and recording the optimal feature selections found for every feature subset size, a feature selection path, along with corresponding estimates of the generalization error is obtained. Algorithm 5 summarizes the full algorithm.

Algorithm 5 Local search algorithm for the relaxed feature selection problem (adapted from Publication III)

```
1:  for all restarts do
2:    - $C_1 = 0$
3:    - initialize $\tilde{s}$ randomly
4:    while $\tilde{s} \neq 0$ do
5:      - estimate generalization error and update SET/FSP
6:      - compute gradient of training error w.r.t. $\tilde{s}$
7:      - evaluate direct neighbor pointed to by gradient and compute its
        training error
8:      - if training error has decreased, jump to neighbor
9:      - if training error has not decreased, increase $C_1$ until gradient
        points to neighbor with smaller $L_1$-norm and smaller training error,
        and jump to that neighbor
10:   end while
11: end for
```

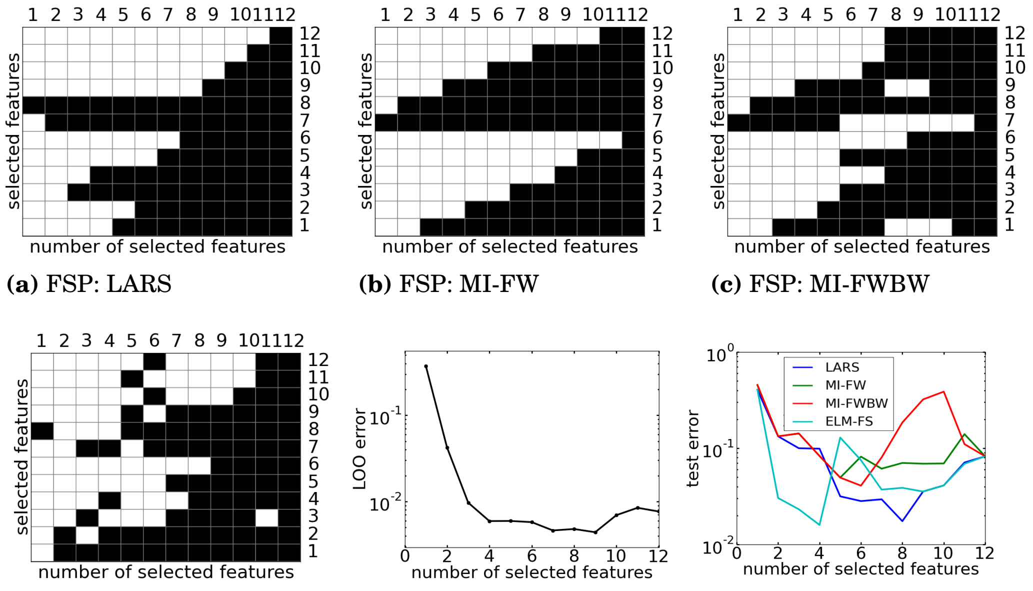
5.2.2 Feature selection path

During its execution, the ELM-FS algorithm (as described before and in detail in Publication III), keeps track of the optimal feature subsets it encounters for each feature subset size. Once finished, these results can be summarized in a plot depicting the Feature Selection Path (FSP). Apart from the optimal feature subset for each size, also the evolution of the feature subset can be seen, potentially giving insight into the problem at hand.

5.2.3 Sparsity-error trade-off curve

Besides the optimal feature selections encountered, the ELM-FS algorithm also keeps track of the corresponding estimated generalization error. These results can be summarized in the Sparsity-error trade-off curve (see Figure 5.1e for an example).

it gets stuck, increase the slope until it starts rolling again.



(a) FSP: LARS **(b)** FSP: MI-FW **(c)** FSP: MI-FWBW

(d) FSP: ELM-FS **(e)** SET curve: ELM-FS **(f)** test errors

Figure 5.1. Results on Santa Fe Laser Data set

Together, these plots can be used by the practitioner to make an informed decision about the optimal feature subset for the context.

5.2.4 Experiments

To evaluate the effectiveness of the ELM-FS algorithm, and to verify the soundness of the obtained variable selections, the ELM-FS algorithm is compared against several other methods for performing variable selection: (1) Least Angle Regression (LARS); (2) Forward Selection with Mutual Information criterion (MI-FW); (3) Forward-Backward Selection with Mutual Information criterion (MI-FWBW).

Once the feature selection paths for all methods are obtained, the test error is evaluated using an Optimally Pruned ELM (OP-ELM). This allows for verification that the SET curve indeed gives an indication of the optimal feature subset, and for comparison between the obtained variable selections of each method.

Data The data set used is again the Santa Fe Laser Data time series (Weigend and Gershenfeld, 1993), which consists of approximately 10000 samples, from which samples are obtained using a regressor size of 12.

Experimental parameters ELM-FS is performed using 100 repetitions. The 100 corresponding ELMs have between 1 and 100 randomly chosen neurons, drawn from a fixed set of 100 neurons. For more details see Publication III.

Comparison of feature selections The results of the experiment are summarized in Figure 5.1. It can be seen that in this experiment, especially for the smaller feature subsets, the ELM-FS is able to find better feature selections than the other methods. Presumably, this is due to its more thorough exploration of the sparser subsets, and the fact that ELM-FS allows for discontinuities in its feature selection path.

Comparison of computational times The computational efficiency of ELM-FS is comparable to MI-FW and MI-FWBW (see Table 2 of Publication III), and selected feature subsets are comparable or better in quality. Therefore, the ELM-FS provides an attractive alternative to existing feature selection methods.

5.3 Fast feature selection using a GPU-accelerated Delta Test

In Publication IV, a GPU-accelerated Delta Test criterion used as criterion in a parallelized genetic algorithm for variable selection. By using the GPU to accelerate the computation of the Delta Test, the main computational bottleneck of the algorithm is alleviated, and through GPU acceleration and parallelization variable selection with the Delta Test can efficiently be performed on large data sets, for which it would otherwise be prohibitively slow.

5.3.1 Parallelization of the Delta Test

Delta Test The Delta Test is a noise variance estimator which can be used to estimate, for a given data set $(x_i, y_i)_{i=1}^N$, the accuracy a non-linear model can possibly attain. Therefore, it can be used a criterion for evaluating the quality of different variable subsets for function approximation, and together with a search algorithm, be used for variable selection (Eirola, 2014; Sovilj, 2014). The Delta Test is defined as

$$\delta = \frac{1}{2N} \sum_{i=1}^{N} (y_i - y_{NN(i)})^2 \tag{5.4}$$

where y_i is the i^{th} sample in the output space, and $y_{NN(i)}$ is sample corresponding to the nearest neighbor of $\mathbf{x}_i$.

Computing the Delta Test From Equation 5.4, it can be seen that in order to compute the value of the Delta Test, for each sample $\mathbf{x}_i$, the nearest neighbor needs to be computed.

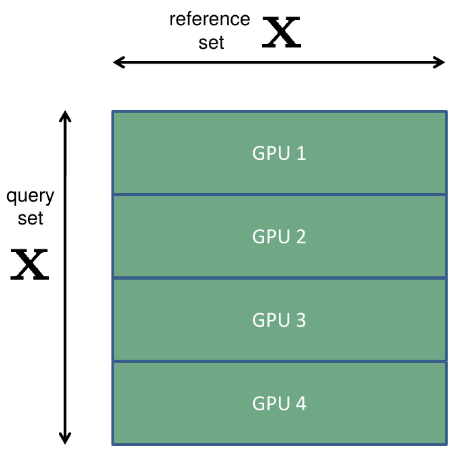


Figure 5.2. Division of nearest-neighbor computations in Multi-GPU-accelerated Delta Test

The approach used in Publication IV is the brute-force method for computing the nearest neighbors, which involves computing an $N \times N$ distance matrix. Once computed, for each x_i, the index of the nearest neighbor can be determined from the distance matrix, and the Delta Test criterion can be computed. Since the approach involves computing an $N \times N$ distance matrix, for modern data sets this can be quite challenging, and strategies are desired for making the computation feasible.

Delta Test at scale The Delta Test is implemented on GPU using the excellent GPU-accelerated nearest-neighbor library by (Garcia et al., 2008, 2010), and for e.g. 17000 samples and with a 1000 variables, a 40-50$\times$ speedup can be obtained over using the CPU.

Although not exploited in Publication IV, the nearest-neighbor problem is embarrassingly parallel (i.e. the nearest neighbor for each point can be determined independently). Therefore, the Delta Test computation can be further accelerated by dividing the nearest-neighbor computations over multiple GPUs, and let each GPU determine the nearest neighbor for part of the data set (see Figure 5.2). Experiments show that for a variable selection problem with 100k samples and 274 dimensions, on a single machine with 5 GPUs, an additional speedup of roughly ~4$\times$ can be achieved.

Discussion The (multi-)GPU-acceleration of the Delta Test makes it a very attractive criterion to use as a filter approach for ELM, either in combination with a local search method like Forward-Backward search as encountered in Publication III, or in combination with genetic algorithm like in Publication IV.

5.4 Binary/Ternary ELM

Although in theory, ELM is a universal approximator, in practice, there are a limited number of samples available, and there is a risk of overfitting. Therefore, the functional approximation should use as limited number of neurons as possible, and the hidden layer should extract and retain as much information as possible from the input samples. The question then becomes, which neurons work well together to extract as much useful information as possible?

Publication V proposes two new ELM variants: Binary ELM, with a weight initialization scheme based on {0,1}-weights; and Ternary ELM, with a weight initialization scheme based on {-1,0,1}-weights. This weight initialization scheme results in features from very different subspaces and therefore, each neuron extracts more diverse information from the inputs than neurons with completely random features traditionally used in ELM. Experiments show that indeed ELMs with ternary weights generally achieve lower test error, and additionally are more robust to irrelevant and noisy variables. Since only the weight generation scheme is adapted, the computational time of the ELM is unaffected, and the improved accuracy, added robustness and the implicit variable selection of Binary ELM and Ternary ELM come for free.

5.4.1 Improved hidden layer weights

Traditionally, the hidden layer weights of the ELM are initialized randomly, with weights and biases drawn from a continuous probability distribution. For example, a uniform distribution on interval [-3, 3], or a Gaussian distribution with certain variance σ. These hidden layer weights, together with the transfer function and the data, result in particular activations of the hidden layer.

A typical transfer function (like sigmoid) takes the inner product of the weight and a sample, adds a bias, and transforms the result in a nonlinear way, i.e. it looks like $f\left(\langle \mathrm{w_i}, \mathrm{x}\rangle + b_i\right)$,where $\langle \mathrm{w_i}, \mathrm{x}\rangle$ is the inner product of weight w_i and x is the input vector. Since$\langle \mathrm{w_i}, \mathrm{x}\rangle = |\mathrm{w_i}||\mathrm{x}|\cos\theta$, it can be seen that the typical activation of f depends on:

1. the expected length of $\mathrm{w_i}$
2. the expected length of x
3. the angles θ between the weights and the samples

5.4.1.1 Orthogonal weights?

An important factor affecting the diversity of the hidden neuron activations consists of the angles between the hidden layer weights and the samples. Therefore, the primary strategy that comes to mind for improving the diversity of the hidden neuron activations is to improve the diversity by taking weights that are mutually orthogonal (e.g. M d-dimensional basis vectors, randomly rotated in the d-dimensional space, where d is the dimension of the input space).

However, experiments suggested that this strategy does not significantly improve accuracy. Presumably, for the tested cases, the random weight scheme of ELM already covers the possible weight space pretty well (furthermore, randomly drawn zero-mean vectors are close to orthogonal in the first place).

5.4.1.2 Binary weight scheme

Another way to improve the diversity of the weights is by having each of them work in a different subspace (e.g. each weight vector has different subset of variables as input). This strategy turns out to significantly improve accuracy, at no extra computational cost.

In the binary weight scheme (see Algorithm 6, Figure 5.3 and Figure 5.4), binary weights are generated and added, starting from the sparsest subspace, until the desired amount of weights is reached. Weights within a subspace are added in random order to avoid bias towards particular variables. Once generated, the weights are normalized or adapted through Batch-Intrinsic Plasticity (BIP) pre-training (see Section 3.4.4 and Section 5.4.2) to ensure the neurons are activated in the right region (neither completely in the linear part, nor in the saturated part).

5.4.1.3 Ternary weight scheme

The ternary weight scheme (see Figure 5.3 and 5.4) is identical to the binary weight scheme, except that due to the sign of the weights there are more possible weights for each possible subspace, rather than a single one, allowing for richer weights.

5.4.2 Motivation for BIP pre-training

Since for given weight w and input x, the expected value of |w||x| determines which part of the transfer function is activated most, the norm of the weights is important and affects the performance of ELM. Of course,

Algorithm 6 Binary weight scheme, with M the desired number of hidden neurons, n the dimension of the subspaces in which to generate weights, and d the number of inputs

```
1: Generate ELM:
2: $n = 1$;
3: while number of neurons $\leq M$ and $n \leq d$ do
4:     - Generate the $\binom{d}{n}$ possible assignments of $n$ ones to $d$ positions
5:     - Shuffle the order of the generated weights to avoid bias to certain
       inputs due to the scheme used to generate the $\binom{d}{n}$ assignments
6:     - Add the generated weights (up to a maximum of $M$ neurons)
7:     - $n = n + 1$;
8: end while
9: - Normalize the norm of the weights, such that they are unit length.
```

$$\begin{array}{c|ccccc|}
 & \mathbf{1} & 0 & 0 & 0 & 0 \\
 & 0 & \mathbf{1} & 0 & 0 & 0 \\
1\,var & 0 & 0 & \mathbf{1} & 0 & 0 \\
 & 0 & 0 & 0 & \mathbf{1} & 0 \\
 & 0 & 0 & 0 & 0 & \mathbf{1} \\ \hline
 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\
 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
 & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 \\
2\,vars & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
 & \vdots & \vdots & \vdots & \vdots & \vdots \\
 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\ \hline
3\,vars & \multicolumn{5}{c|}{etc.}
\end{array}$$

(a) binary weight scheme

$$\begin{array}{c|cccc|}
 & \mathbf{+1} & 0 & 0 & 0 \\
 & \mathbf{-1} & 0 & 0 & 0 \\
1\,var & 0 & \mathbf{+1} & 0 & 0 \\
 & 0 & \mathbf{-1} & 0 & 0 \\
 & \vdots & \vdots & \vdots & \vdots \\ \hline
 & \mathbf{+1} & \mathbf{+1} & 0 & 0 \\
 & \mathbf{+1} & \mathbf{-1} & 0 & 0 \\
 & \mathbf{-1} & \mathbf{+1} & 0 & 0 \\
2\,vars & \mathbf{-1} & \mathbf{-1} & 0 & 0 \\
 & \vdots & \vdots & \vdots & \vdots \\
 & 0 & 0 & \mathbf{-1} & \mathbf{-1} \\ \hline
3\,vars & & & &
\end{array}$$

(b) ternary weight scheme

Figure 5.3. Illustration of the binary and ternary weight schemes. Note that weights are added starting from the sparsest subspace, and from each subspace they are added in random order to avoid bias towards particular variables.

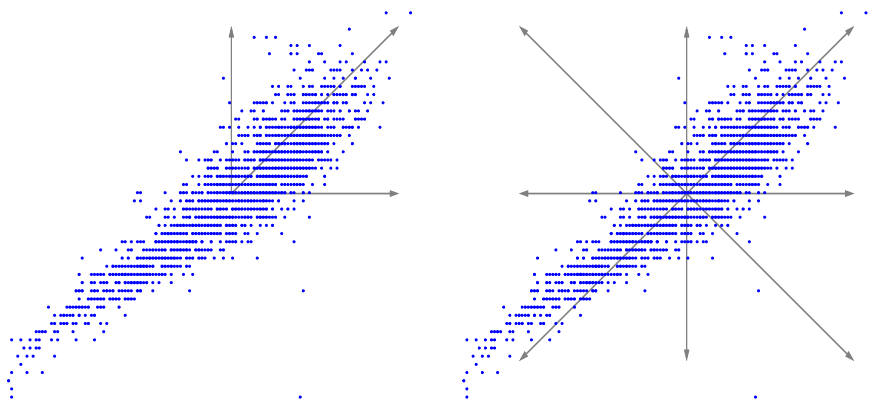

(a) possible binary weights **(b)** possible ternary weights

Figure 5.4. Illustration of possible weights (*arrows*) for binary (*a*) and ternary (*b*) weight scheme, in a 2D subspace of normalized Abalone data (*blue dots*)

the weights could be normalized to be e.g. unit length, but the question remains what is the optimal length for the given data. Therefore, to ensure that the weights are properly scaled, the ELMs are pre-trained using Batch Intrinsic Plasticity (BIP) pre-training. In particular, the BIP(rand) variant (Neumann and Steil, 2011, 2013) is used, since it offers an attractive balance between computational time and accuracy.

An added advantage of using BIP pre-training is that when comparing ELMs with varying weight schemes, any differences in performance must come from the differences in the direction of the weights and are not a result of the different scaling of the weights.

Furthermore, since BIP pre-training adapts the neurons to operate in their non-linear regime, as many linear neurons are included as there are input variables. This ensures good performance of the ELM, even if the problem is completely linear.

5.4.3 Fast L2 regularization through SVD

With limited data, the capability of ELM to overfit the data increases with increasing number of neurons, especially if those neurons are optimized to be well-suited for the function approximation problem. Therefore, to avoid overfitting, Tikhonov regularization with an efficiently cross-validated regularization parameter is used.

Using the SVD decomposition of $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, it is possible to obtain all needed information for computing the PRESS statistic without recomputing the pseudo-inverse for every λ:

$$\begin{aligned}
\hat{\mathbf{Y}} &= \mathbf{H}\beta \\
&= \mathbf{H}(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{Y} \\
&= \mathbf{H}\mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y} \\
&= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y} \\
&= \mathrm{HAT} \cdot \mathbf{Y}
\end{aligned}$$

where $\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}$ is a diagonal matrix with $\frac{d_{ii}^2}{d_{ii}^2+\lambda}$ as the i^{th} diagonal entry. Now

$$\begin{aligned}
\mathrm{MSE}^{\text{TR-PRESS}} &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - hat_{ii}}\right)^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - \mathbf{h}_{i\cdot}(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{h}_{i\cdot}^T}\right)^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - \mathbf{u}_{i\cdot}\left(\frac{d_{ii}^2}{d_{ii}^2+\lambda}\right)\mathbf{u}_{i\cdot}^T}\right)^2
\end{aligned}$$

where $\mathbf{h}_{i\cdot}$ and $\mathbf{u}_{i\cdot}$ are the i^{th} row vectors of $\mathbf{H}$ and $\mathbf{U}$, respectively. Now, the Tikhonov-regularized PRESS and corresponding λ can be computed using Algorithm 7, where $\mathbf{A} \circ \mathbf{B}$ refers to the element-wise product of matrices $\mathbf{A}$ and B (Schur product). Due to the convex nature of criterion $\mathrm{MSE}^{\text{TR-PRESS}}$ with respect to regularization parameter λ, the Nelder-Mead procedure used for optimizing λ converges quickly in practice (Nelder and Mead, 1965; Lagarias et al., 1998). This efficient optimization of the regularization parameter for ELM, by incorporating it in the SVD decomposition first appeared (in slightly different form) in (Miche et al., 2011).

5.4.4 Experiments

In Publication V, for evaluating the performance of the gaussian, binary and ternary weight schemes, their resulting performance in terms of accuracy is compared on various regression tasks. Since the weight range determines the typical activation of the transfer function (remember $\langle \mathbf{w_i}, \mathbf{x}\rangle = |\mathbf{w_i}||\mathbf{x}|\cos\theta$), the ELMs are pre-trained using Batch Intrinsic Plasticity pre-training. Any performance difference between weight schemes will be a result of the different directions of the weights. Furthermore, since BIP pre-training adapts the neurons to operate in their non-linear regime, as many linear neurons are included as there are input variables. This ensures good performance of the ELM, even if the problem is completely

Algorithm 7 Tikhonov-regularized PRESS. In practice, the **while** part of this algorithm (convergence for λ) is solved using by a Nelder-Mead approach (Nelder and Mead, 1965) (Publication V)).

```
1: Decompose $\mathbf{H}$ by SVD: $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^T$
2: Precompute $\mathbf{B} = \mathbf{U}^T\mathbf{y}$
3: while no convergence on $\lambda$ achieved do
4:     - Precompute $\mathbf{C} = \mathbf{U} \circ \begin{pmatrix} \frac{d_{11}^2}{d_{11}^2+\lambda} & \cdots & \frac{d_{11}^2}{d_{11}^2+\lambda} \\ \vdots & \ddots & \vdots \\ \frac{d_{NN}^2}{d_{NN}^2+\lambda} & \cdots & \frac{d_{NN}^2}{d_{NN}^2+\lambda} \end{pmatrix}$
5:     - Compute $\hat{\mathbf{y}} = \mathbf{C}\mathbf{B}$, the vector containing all $\hat{y}_i$
6:     - Compute $\mathbf{d} = \text{diag}\left(\mathbf{C}\mathbf{U}^T\right)$, the diagonal of the HAT matrix, by
       taking the row-wise dot-product of $\mathbf{C}$ and $\mathbf{U}$
7:     - Compute $\varepsilon = \frac{\mathbf{y}-\hat{\mathbf{y}}}{\mathbf{1}-\mathbf{d}}$, the leave-one-out errors
8:     - Compute $\text{MSE}^{\text{TR-PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$
9: end while
10: Keep the best $\text{MSE}^{\text{TR-PRESS}}$ and the associated $\lambda$ value
```

linear. Finally, to prevent overfitting, the models incorporate L2 regularization with efficiently optimized regularization parameter (as described in Section 5.4.3). Therefore, in summary, the models that are compared are the BIP(rand)-TR-ELM; the BIP(rand)-TR-2-ELM and the BIP(rand)-TR-3-ELM.

Data The data sets used for the experiments are the Abalone (8 input variables, 4177 samples) and ComputerActivity (12 input variables, 8192 samples) data sets from the UCI repository (Asuncion and Newman, 2007). Roughly half of the samples is used for training, and the remaining half is used for testing.

Relative performance of weight schemes In this experiment, the relative performance of the weight schemes is compared for varying number of hidden neurons. As mentioned, the ELMs are pre-trained through BIP, and use L2 regularization with efficiently optimized regularization parameter. As further baseline, a standard ELM and an ELM which has its number of neurons cross-validated are included as well. The results of this experiment are summarized in Figure 5.5.

There, it can be seen that for increasing number of neurons, the standard ELM starts to overfit at some point, resulting in an increase in the RMSE on the test set. Performing leave-one-out cross-validation to limit the number of used hidden neurons prevents this overfitting. Finally, the proposed methods generally achieve much better RMSE than the basic ELM variants and generally, ternary weights outperform weights drawn from a Gaussian distribution, while binary weights perform worse than ternary and Gaussian weights for large number of neurons.

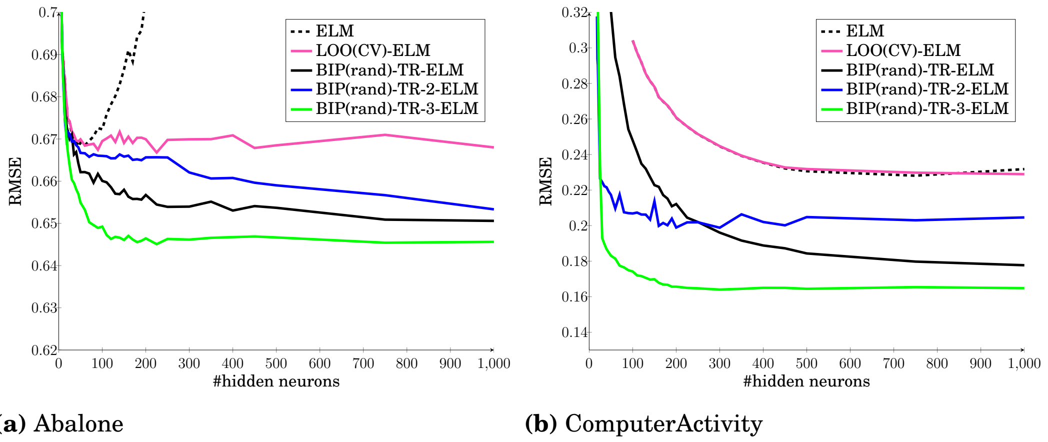


Figure 5.5. number of neurons vs. average achieved test RMSE for ELM (*black, dashed*), LOO(CV)-ELM (*purple*), BIP(rand)-TR-ELM with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme (Publication V).

Robustness against irrelevant variables In this experiment, the robustness of the different weight schemes against irrelevant variables is investigated by measuring the effect of up to 30 added noise variables on the RMSE. The results are summarized in Figure 5.5 and Table 5.1, where it can be seen that the proposed weight schemes are more robust against irrelevant variables. The difference is especially large for the Computer-Activity Data set.

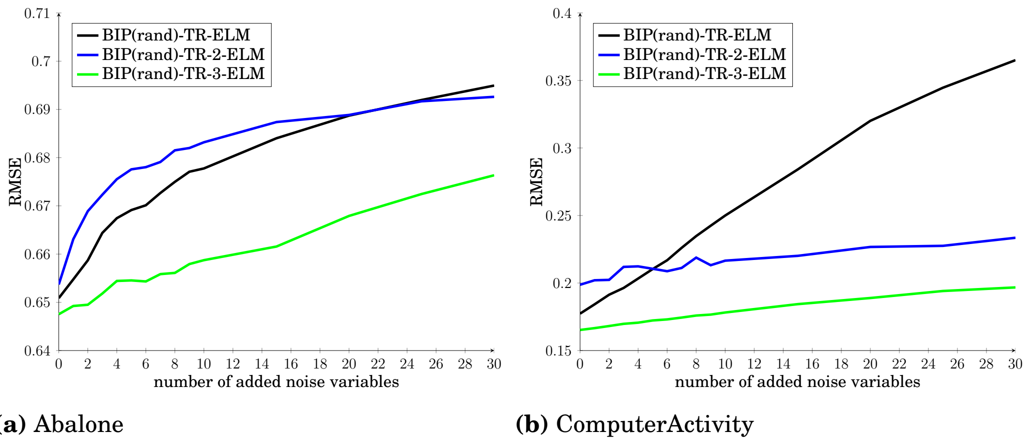


Figure 5.6. Effect of adding irrelevant extra variables on RMSE for BIP(rand)-TR-ELM with 1000 hidden neurons and with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme (Publication V).

	Ab			Co		
	gaussian	binary	ternary	gaussian	binary	ternary
RMSE with original variables	0.6497	0.6544	0.6438	0.1746	0.1785	0.1639
RMSE with 30 added irr. vars	0.6982	0.6932	0.6788	0.3221	0.2106	0.1904
RMSE loss	0.0486	**0.0388**	**0.0339**	0.1475	**0.0321**	**0.0265**

Table 5.1. Effect of adding irrelevant extra variables on RMSE for BIP(rand)-TR-ELM with 1000 hidden neurons and with gaussian, binary, and ternary weight scheme (Publication V).

Implicit variable selection Finally, to get more insight into why the weight schemes perform the way they do, the BIP(rand)-TR-ELM, BIP(rand)-TR-2-ELM and the BIP(rand)-TR-3-ELM are trained on the ComputerActivity data set, where 5 irrelevant variables (taken from the DeltaElevators UCI data) and 5 noise variables have been added.

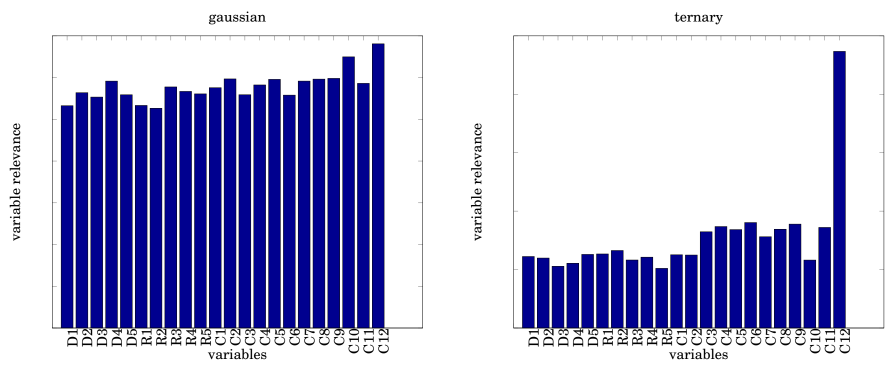
After training, the relevance of each input variable is quantified as

$$\mathbf{relevance} = \sum_{i=1}^{M} |\beta_i \times \mathbf{w}_i|,$$

where M is the number of hidden neurons; β_i is the output weight; $\mathbf{w}_i$ is the input weight corresponding to neuron i, and **relevance** is the d-dimensional vector containing a measure of relevance for each of the d input variables. If a variable j has a large value of $\mathbf{relevance}_j$, compared to other variables, this can be interpreted as that variable being implicitly selected by the ELM (i.e. the ELM favors neurons that extract information from that variable).

The results are summarized in Figure 5.7, and suggest qualitatively different behaviour of the ELMs with the different weight schemes. While the BIP(rand)-TR-ELM does not favor any neurons that employ a particular input variable, the BIP(rand)-TR-3-ELM favors neurons that employ a specific input variable. Similar results hold for BIP(rand)-TR-2-ELM.

These results suggests that the improved performance and robustness of the binary and ternary weight scheme might come from implicit variable selection, and that through the modified weight schemes, they are able to focus on those variables that are important to the problem, while ignoring the variables that are not.



(a) Gaussian weight scheme **(b)** Ternary weight scheme

Figure 5.7. Relevance assigned by the weight schemes to different input variables, as measured by $\sum_{i=1}^{M} |\beta_i \times \mathbf{w}_i|$ (Publication V).

Discussion Two new weight initialization schemes have been proposed and robust ELM variants using these weight schemes are introduced and evaluated: the BIP(rand)-TR-2-ELM and BIP(rand)-TR-3-ELM.

The motivation behind these schemes is that weights picked in this way will be from very different subspaces, and therefore improve the diversity of the neurons in the hidden layer.

Experiments show that Ternary ELM generally achieves lower test error and that both the binary and ternary weight schemes improve robustness of the ELM against irrelevant and noisy variables, which might be due to the schemes being able to perform implicit variable selection. Since only the weight generation scheme is changed, the computational time of ELM remains unchanged compared to ELMs with traditional random weights. Therefore, the better performance, added robustness and implicit variable selection in Binary ELM and Ternary ELM come for free.

Although this work only investigates the robustness of the weight schemes, an additional advantage of the proposed weight schemes is that they result in sparse weight matrices. Especially for large ELMs, this can result in significantly reduced memory requirements, since the weight matrices can be saved as sparse matrices with binary or ternary entries[3]. Furthermore, algorithms designed for sparse matrix multiplication might be used to speed up the multiplication of the inputs and the hidden layer weights. This could potentially provide a further way to speed up existing MapReduce schemes for accelerating the hidden layer computations (Catanzaro et al., 2008; He et al., 2011, 2013).

[3]Although the weights of each neuron are normalized, all weights of a neuron are scaled with the same scalar, which can be incorporated in e.g. the slope of the sigmoid transfer function.

6. Trade-offs in Extreme Learning Machines

"There are all kinds of interesting questions that come from a knowledge of science, which only adds to the excitement and mystery and awe of a flower. It only adds. I don't understand how it subtracts."

– Richard P. Feynman

In the training of neural networks, there often exists a trade-off between the time spent optimizing the model under investigation, and its final performance. Ideally, an optimization algorithm finds the model that has best test accuracy from the hypothesis space as fast as possible, and this model is efficient to evaluate at test time as well. However, in practice, there exists a trade-off between training time, testing time and testing accuracy, and the optimal trade-off depends on the user's requirements.

This chapter gives an overview of some of those trade-offs within the context of ELMs and highlights the results from Publication VI, which introduces the Compressive ELM and forms an initial investigation into these trade-offs within Extreme Learning Machines.

Whereas the other contributions are mostly aimed at directly improving the accuracy of the ELM-based methods, either by ensembling them (Publication I, Publication II), by obtaining better variable selections (Publication III, Publication IV), or by improved weight initialization schemes (Publication V), the Compressive ELM takes a contrary approach.

Instead of directly improving the accuracy, the Compressive ELM focuses on improving the computational time by providing a time-accuracy trade-off and training the model in a reduced space, while trying to retain accuracy as much as possible. Experiments indicate that potentially more time can be saved than accuracy lost and therefore the compressive training mechanism may provide an avenue towards obtaining more accurate models in less time.

6.1 Trade-offs between computational time and accuracy

When choosing a model for solving a machine learning problem, which model is most suitable depends a lot on the context and the requirements of the application. For example, it might be the case that the model is trained on a continuous stream of data, and therefore has some restrictions on the training time. On the other hand, the computational time in the testing phase might be restricted, like in a setting where the model is used as the controller for an aircraft or a similar setting that requires fast predictions. Alternatively, the context in which the model is applied might not have any strong constraints on the computational time and, above all, accuracy or interpretability could be considered most important regardless of the computational time.

6.1.1 Time-accuracy curves

This chapter focuses on time-accuracy trade-offs in Extreme Learning Machines, and on trade-offs between training time and accuracy in particular, which can be affected in two ways:

- by **improving the accuracy** through spending more time optimizing the model,

- or vice-versa, by **reducing the computational time** of the model, without sacrificing accuracy too much.

Each type of model has its own ways of balancing computational time and accuracy, and has an associated training time-accuracy curve that expresses the efficiency of the model in achieving a certain accuracy (the closer the curve is to the bottom left, the better). Thus, given a collection of models, the question becomes: which model produces the best accuracy the fastest?

6.1.2 Examples from Extreme Learning Machines

In order to illustrate what time-accuracy trade-offs exist within Extreme Learning Machines, this section presents time-accuracy trade-offs of several Extreme Learning Machine variants.

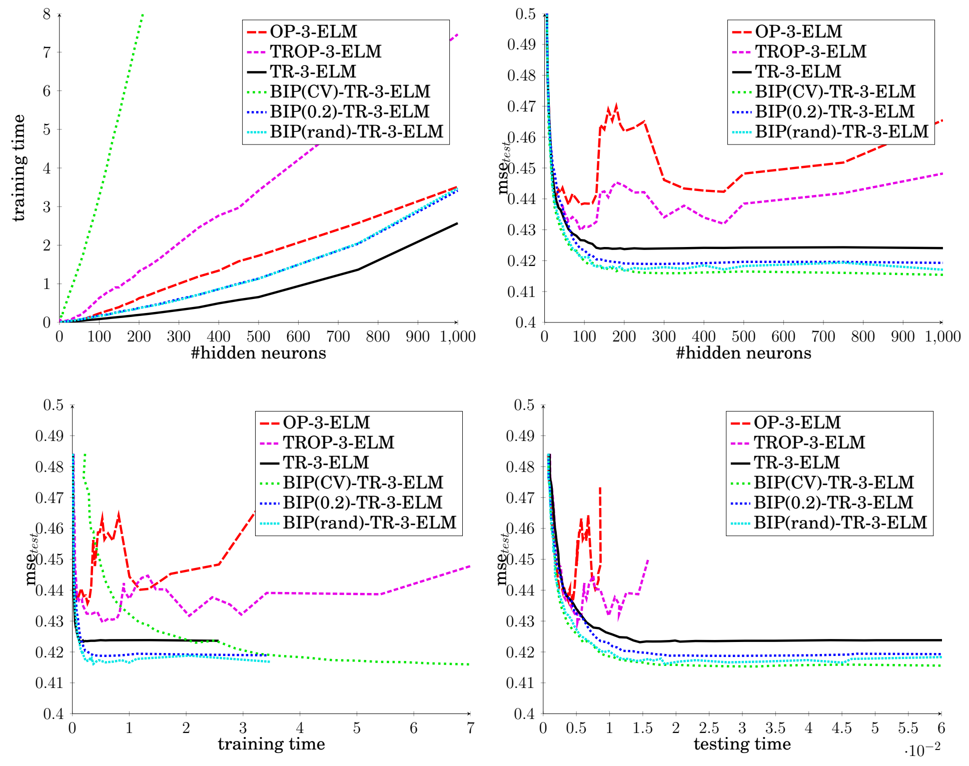


Figure 6.1. Results for various ELM variants on Abalone UCI data set (Publication V).

Extreme Learning Machine variants The following variants of Extreme Learning Machine are compared in terms of their training time, testing time, and obtained accuracy on the Abalone UCI data set (Asuncion and Newman, 2007). For all of the variants, the ternary weight scheme is used, because of the benefits shown in Publication V, and Section 5.4. The Tikhonov-regularized variants all use efficient optimization of regularization parameter λ, using the SVD approach to computing $\mathbf{H}^{\dagger}$ (see Section 5.4.3).

OP-ELM The Optimally Pruned ELM (Miche et al., 2010), with neurons ranked by relevance, and then pruned based on the leave-one-out error.

TROP-ELM The Tikhonov-regularized OP-ELM (Miche et al., 2011).

TR-ELM The Tikhonov-regularized ELM (Deng et al., 2009).

BIP(CV)/BIP(0.2)/BIP(rand) TR-ELM The Tikhonov-regularized ELM pre-trained using Batch Intrinsic Plasticity mechanism (Neumann and Steil, 2011, 2013; Neumann, 2013), adapting the hidden layer weights and biases, such that they retain as much information as possible. The BIP parameter is either fixed, randomized, or cross-validated over 20 possible values.

Most desirable model depends on requirements The results are summarized in Figure 6.1, and show that it depends on the user's criteria which model is most suitable for that context.

Training time most important If it is most important that the model can be trained fast, then it can be argued that the BIP(rand)-TR-3-ELM offers the best trade-off. It does take slightly longer to train than the TR-3-ELM, however, for that modest overhead the test accuracy is generally improved.

Test error most important If accuracy is all that matters, then the results suggest it might be worth it to cross-validate the BIP parameter, instead of setting it randomly per-neuron. Although the training time for BIP(CV)-TR-3-ELM is many times higher due to the cross-validation of the BIP parameter, it generally results in the most accurate model.

Testing time most important If the speed of the model at test time is most important, then, surprisingly, the results suggest that the BIP(rand)-TR-3-ELM is the most suitable model. This result is unexpected, since a model that has its irrelevant neurons pruned (like OP-ELM and TROP-ELM) is generally faster at test time. However, for this context, the results reveal that, even though the OP-ELM and TROP-ELM are faster, they tend to slightly overfit while pruning the neurons, resulting in slightly worse test accuracy.

From Figure 6.1, it can be seen that because of this, the TR-3-ELM variants are more attractive when it comes to testing time (i.e. for a given testing time, they are always able to provide the better accuracy).

This surprising result shows the importance of analyzing ELM algorithms in terms of their time-accuracy trade-off. Furthermore, this time-accuracy trade-off analysis suggests further research directions into the prevention of overfitting in model structure selection (Reunanen, 2003).

An initial work along these lines is (Wang et al., 2014), which provides a variant of OP-ELM that computes an approximate ranking of the neurons, rather than an exact ranking, and is shown to achieve better accuracy. Presumably, this is because the approximate ranking prevents the model structure selection from overfitting.

Since TR-3-ELM offers attractive trade-offs between speed and accuracy, this model is used for the Compressive Extreme Learning Machine, which will be discussed next.

6.2 Compressive ELM

In the previous section on time-accuracy trade-offs, two possible strategies have been discussed that can affect this trade-off: (1) improving the accuracy of the models, and (2) reducing the computational time of the model. In terms of training time-accuracy plots, this would be "pushing the curve down" and "pushing the curve to the left", respectively.

Whereas the other contributions discussed in this thesis mainly focus on improving the accuracy of the models by ensembling them (Publication I, Publication II), by obtaining better variable selections (Publication III, Publication IV), or by improved weight initialization schemes (Publication V), the Compressive ELM focuses on reducing computational time by performing the training in a reduced space, while retaining accuracy as much as possible.

6.2.1 Low-distortion embeddings

To achieve this, the dimensionality of the hidden layer output matrix is reduced by creating a low-distortion embedding of it in a lower-dimensional space through Johnson-Lindenstrauss-like embeddings (Johnson and Lindenstrauss, 1984; Achlioptas, 2003; Matoušek, 2008), which approximately preserve the distances between the points and retain the relevant structure as much as possible. These embeddings occur in approximate distance-based machine learning algorithms like approximate nearest-neighbors (Indyk and Motwani, 1998), and are extensively used in the field of Randomized numerical linear algebra (Martinsson, 2009).

6.2.2 Randomized numerical linear algebra

Part of many algorithms in randomized numerical linear algebra is the embedding (or sketching) of the data into a lower dimension. In comparison to distance-based methods though, in linear algebra, the requirement for preserving the distances is not as strict, and as long as the distances are roughly preserved (within some factor) it is useful (Martinsson, 2009).

One aspect of linear algebra in which low-distortion embeddings can be used is in approximate matrix decomposition. Given a matrix, an approximate matrix decomposition can be achieved by first embedding the rows of the matrix into a lower-dimensional space (through one of many available low-distortion Johnson-Lindenstrauss-like embeddings), solving the decomposition, and then projecting back to the full space. If such an embedding (or sketch) is accurate, then this allows for solving the problem with high accuracy in reduced time. An example of such an algorithm is the approximate SVD.

Approximate SVD The algorithm for approximate SVD is summarized in Algorithm 8. More background on the algorithm can be found in (Halko et al., 2011).

Algorithm 8 Approximate SVD (Halko et al., 2011) (Publication V).

```
Given an $m \times n$ matrix $\mathbf{A}$, compute k-term approximate SVD $\mathbf{A} \approx \mathbf{U}\mathbf{D}\mathbf{V}^T$ as follows:
1: - Form the $n \times (k+p)$ random matrix $\mathbf{\Omega}$. (where $p$ is small oversampling parameter)
2: - Form the $m \times (k+p)$ sampling matrix $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$. ("sketch" it by applying $\mathbf{\Omega}$)
3: - Form the $m \times (k+p)$ orthonormal matrix $\mathbf{Q}$, such that range($\mathbf{Q}$) = range($\mathbf{Y}$).
4: - Compute $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
5: - Form the SVD of $\mathbf{B}$ so that $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^T$
6: - Compute the matrix $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$
```

Application to ELM Through the approximate SVD, the time it takes to train the ELM can be reduced. Furthermore, the efficient L2 regularization from Section 5.4.3 can be easily integrated in it as well. The main question now is, whether it is actually possible to obtain more accurate models in less time.

Before giving an overview of the experiments investigating this, the next section gives a short overview of fast sketching algorithms that can be used as part of the approximate SVD algorithm.

6.2.3 Faster Sketching

Typically, the bottleneck in Algorithm 8 is the time it takes to sketch the matrix. Rather than using a class of random matrices of Gaussian variables for sketching A, one can also use random matrices that are sparse or structured in some way (Achlioptas, 2003; Matoušek, 2008), and for which the matrix-vector product can be computed more efficiently. Furthermore, (Ailon and Chazelle, 2006) introduced the Fast Johnson-Lindenstrauss Transform (FJLT), which uses a class of random matrices that permit application to vectors from an $n \times n$ matrix to a vector in $\mathcal{O}(n \log(n))$, rather than the usual $\mathcal{O}(n^2)$. Besides this obvious speedup, this class of matrices is also more successful in creating a low-distortion embedding when applied to a sparse matrix. These transforms consist of the application of three easy-to-compute matrices

$$\left(\mathbf{P} \right)_{k \times n} \left(\mathbf{W} \right)_{n \times n} \left(\mathbf{D} \right)_{n \times n}$$

where $\mathbf{P}$, $\mathbf{W}$, and $\mathbf{D}$ vary depending on the exact scheme. Generally, $\mathbf{D}$ is a diagonal matrix with random Rademacher variables $\{-1, +1\}$ on the diagonal. In this work, the following transforms are considered for faster sketching:

- **Fast Johnson Lindenstrauss Transform (FJLT)**, introduced in (Ailon and Chazelle, 2006) for which $\mathbf{P}$ is a sparse matrix of random Gaussian variables, and $\mathbf{W}$ encodes the Discrete Walsh-Hadamard Transform.

- **Subsampled Randomized Hadamard Transform (SRHT)**, for which $\mathbf{P}$ is a matrix selecting k random columns from $\mathbf{W}$, and $\mathbf{W}$ encodes the Discrete Walsh-Hadamard Transform.

Both sketching methods were implemented for the Compressive ELM by adapting the excellent Blendenpik library (Avron et al., 2010).

6.2.4 Experiments

The experiments of Publication V investigate the trade-off between computational time (both training and test), and the accuracy of the Compressive ELM in relation to the dimensionality of the space into which the problem is reduced, using the sketch. For sketching, Compressive TR-3-ELMs with a Gaussian, FJLT, and SRHT sketching scheme are considered, and compared with the standard TR-3-ELM.

Data The experiments are performed using the CaliforniaHousing data set (Asuncion and Newman, 2007), and the data is divided randomly into 8000 random samples for training and the remaining 12640 samples for testing.

Experimental parameters The number of hidden neurons in each model is varied between 2 and 1000, and parameter $k = [50, 100, 200, 400, 600]$.

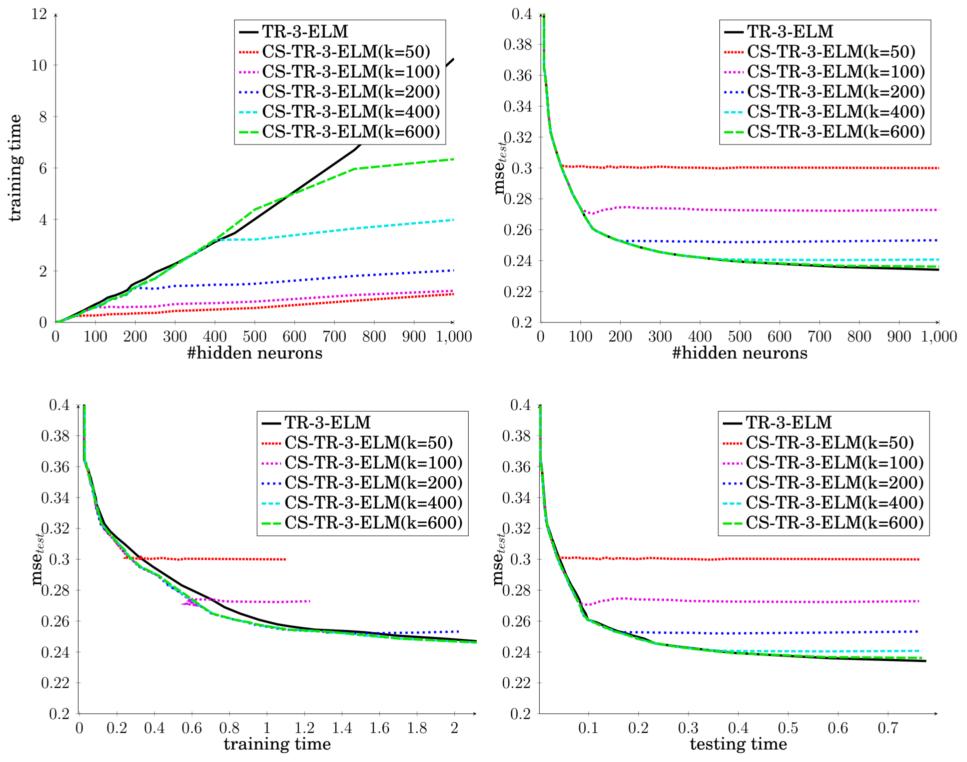


Figure 6.2. Results for Compressive ELMs using FJLT sketching with varying k on CaliforniaHousing UCI data set (Publication V).

Experiments are repeated with 200 random realizations of the training and test set, and average results over those 200 runs are reported.

Effect of embedding dimension Figure 6.2 shows the time-accuracy trade-offs achieved by the various methods, and shows the effect of the embedding dimension. Although only the results of FJLT sketching are presented here, similar results hold for the other sketching methods. It can be seen that

- setting k lower than the number of neurons results in faster training times (which makes sense since the problem solved is smaller).
- as long as parameter k is chosen large enough, the method is not losing efficiency (i.e. there is no model that achieves better error in the same computational time), and it is potentially gaining efficiency (as shown by the bottom-left plot of Figure 6.2.

Effect of embedding scheme Finally, the experiments showed that sketches with Gaussian matrices are generally the fastest, and for the tested problem sizes, the SRHT is slightly faster than the FJLT.

Discussion Surprisingly, the experiments did not show substantial differences in terms of computational time for the different embedding schemes. Therefore, for the datasets tested, it seems that although FJLT permits faster matrix multiplications, no advantage could be taken of it in the tested problems. More research is required to gain more insight into this behaviour, and to understand in what contexts the FJLT can effectively be exploited.

Furthermore, the results indicate that although the Compressive ELM provides an effective trade-off between time and accuracy, in this case, little compression could be obtained for the information in the hidden layer: as soon as k is set lower than the number of neurons, the accuracy decreases accordingly. This challenge to significantly compress the information in the hidden layer might be due to the quality of the features used in the Tikhonov-regularized Ternary Compressive ELM: they might result in a full-rank matrix that has no redundancies. However, more research is needed to better understand in what architectures and applications the Compressive ELM can effectively compress the randomly extracted features.

In summary, the Compressive ELM provides a way for trading off computational time and accuracy, by performing the training in a reduced space, using low-distortion embeddings and approximate matrix decompositions based on them. Experiments indicate that through this mechanism, potentially more time can be saved than accuracy lost and therefore the compressive training mechanism may provide a way to obtain more accurate models in less time.

Finally, developing low-distortion embeddings and related theoretical results is currently a very active field of research, and new developments in this area can be readily integrated to improve the performance of Compressive ELM.

7. Conclusions and Discussion

7.1 Contributions

The thesis develops several ensembling strategies based on ELM that can be applied in nonstationary environments and to large-scale learning problems. Due to the fine-grained structure of the ensemble models, they are very flexible, and can be evaluated in a parallel fashion. The individual models can be built and evaluated efficiently by performing the training and model structure selection on GPUs. Finally, the (adaptive) combination of diverse individual models results in a more accurate model. The models themselves can be relatively simple, while due to ensembling the overall model is still accurate. The results suggest this to be an accurate, yet flexible approach that can be used for analyzing large-scale potentially nonstationary streams of data.

Furthermore, several very different variable selection strategies have been explored. Firstly, in the ELM-FS, the use of the ELM architecture for variable selection, through relaxation of the variable selection problem is investigated. Besides providing a good subset of variables, the presented approach determines for each subset size, which variable subset would be best, and how the generalization error is expected to change depending on the number of variables selected. This provides insight into the problem, and allows for an informed decision on the variable subset that is best for the current context. Secondly, a filter method is presented for variable selection, which allows for variable selection in large-scale problems through the use of (multi-)GPU-acceleration of the Delta Test criterion, used in a parallelized genetic algorithm. Finally, new binary and ternary random weight schemes for ELM are developed, which result in more accurate and compact models. Experiments suggest that the changed weight schemes make the ELM more robust to noisy and irrelevant variables, and are therefore a promising target for future research.

Finally, an approach was presented for reducing the computational time needed for training the ELM by embedding the random feature expansions in a lower-dimensional space. This allows for a flexible trade-off between accuracy and speed of the ELM, and could potentially reduce redundancy in the hidden neurons, resulting in more effective models. Experiments suggest that this approach may provide a path towards obtaining more accurate models in less time.

Overall, the collection of proposed methods provides an efficient, accurate and flexible framework for solving large-scale supervised learning problems. Several distinct strategies are explored for obtaining faster and more accurate models that are particularly suited for handling modern large-scale data sets. The developed methods are not limited to the particular types of ELMs and contexts in which they have been tested, and they may be readily combined, adapted and integrated for application in different contexts and models.

7.2 Future directions

Of course research is a continuing process, and every new development and result brings new questions. To conclude, this section gives a short overview of some of those possible future research directions.

Better training and regularization strategies Several results from the thesis suggest the importance of proper regularization for obtaining an accurate model and preventing overfitting. Furthermore, recent results in deep learning (Ba and Caruana, 2014), suggest that the success of deep neural networks may be due to the way they are trained, rather than just their architecture: i.e. it is possible to train a shallow mimic network based on an already trained deep network, which attains better accuracy than when a shallow network is trained directly. Perhaps what is missing for shallow networks, in order to obtain as good or similar performance as deep networks, is the right algorithm?

Therefore, an important line of research is exploring alternative training and regularization strategies in ELM-based architectures. What is the best strategy in training and performing model structure selection, such that the model obtains high accuracy and does not suffer from overfitting?

Improved data-agnostic random features The results from the Binary and Ternary ELM suggest that improved data-agnostic random features can be obtained by using weight initialization schemes other than the ones typically used in hidden layer initialization. Why exactly does one weight scheme work better than another, and can insights into this inform on even better weight schemes?

Overall then, aside from the question “what is the best accuracy a model can achieve?”, in my opinion the more interesting question is “what is the most effective way to achieve this accuracy?”. What routes exist towards obtaining optimal performance, and what effective shortcuts exist along the way for getting there faster?

Bibliography

D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, June 2003. ISSN 00220000. doi: 10.1016/S0022-0000(03)00025-4.

E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180:012037, 2009. ISSN 1742-6596. doi: 10.1088/1742-6596/180/1/012037.

N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing - STOC '06*, New York, New York, USA, 2006. ACM Press. ISBN 1-59593-134-1. doi: 10.1145/1132516.1132597.

H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. ISSN 00189286. doi: 10.1109/TAC.1974.1100705.

D. M. Allen. The relationship between variable selection and data agumentation and a method for prediction. *Technometrics*, pages 125–127, Dec. 1974.

E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2nd edition, 2010.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. W. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' guide*. Society for Industrial Mathematics, Philadelphia, PA, third edition, Apr. 1999. ISBN 0-89871-447-8 (paperback).

A. Asuncion and D. J. Newman. UCI Machine Learning Repository, 2007.

H. Avron, P. Maymounkov, and S. Toledo. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010.

J. Ba and R. Caruana. Do Deep Nets Really Need to be Deep? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662. Curran Associates, Inc., 2014.

R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

G. J. Bierman. *Factorization Methods for Discrete Sequential Estimation*. Academic Press, New York, NY, 1977.

C. M. Bishop. *Pattern recognition and machine learning*. Springer, Secaucus, 2006. ISBN 0-387-31073-8.

G. Bontempi, M. Birattari, and H. Bersini. Recursive Lazy Learning for Modeling and Control. In *European Conference on Machine Learning*, pages 292–303, 1998.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug. 1996a. ISSN 0885-6125. doi: 10.1007/BF00058655.

L. Breiman. Stacked regressions. *Machine Learning*, 24:49–64, 1996b. ISSN 0885-6125. doi: 10.1007/BF00117832.

E. K. Burke and G. Kendall. Search methodologies: Introductory tutorials in optimization and decision support techniques. pages 1–620, 2005. ISSN 1873-2518. doi: 10.1007/0-387-28356-0.

B. C. Catanzaro, N. Sundaram, and K. Keutzer. Fast Support Vector Machine Training and Classification on Graphics Processors. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 104–111, Helsinki, Finland, 2008. ACM.

C. P. Chen. A rapid supervised learning neural network for function interpolation and approximation. *IEEE Transactions on Neural Networks*, 7(5):1220–30, Jan. 1996. ISSN 1045-9227. doi: 10.1109/72.536316.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, pages 303–314, 1989.

W. D. W. Deng, Q.-H. Zheng, and L. C. L. Chen. Regularized Extreme Learning Machine. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, number 60825202 in 2009 IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, pages 389–395, MOE KLINNS Lab and SKLMS Lab, Xi'an Jiaotong University, Xi'an, China, 2009. IEEE. ISBN 978-1-4244-2765-9. doi: 10.1109/CIDM.2009.4938676.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*, volume 57. Chapman & Hall/CRC, May 1993. ISBN 0-412-04231-2.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least Angle Regression. *The Annals of Statistics*, 32(2):1–44, 2003.

E. Eirola. *Machine learning methods for incomplete data and variable selection*. Ph.D. Thesis, Aalto University, 2014.

E. Eirola, E. Liitiäinen, A. Lendasse, F. Corona, and M. Verleysen. Using the Delta test for variable selection. In *Proc. of ESANN 2008 European Symposium on Artificial Neural Networks*, pages 25–30, 2008. ISBN 2930307080.

G. Feng, G.-B. Huang, Q. Lin, and R. Gay. Error Minimized Extreme Learning Machine With Growth of Hidden Nodes and Incremental Learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, 2009. ISSN 10459227. doi: 10.1109/TNN.2009.2024147.

D. François, F. Rossi, V. Wertz, and M. Verleysen. Resampling methods for parameter-free and robust feature selection with mutual information. *Neurocomputing*, 70:1276–1288, 2007. ISSN 09252312. doi: 10.1016/j.neucom.2006.11.019.

B. Frénay and M. Verleysen. Using SVMs with randomised feature spaces: an extreme learning approach. *ESANN 2010: 18th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, (April):315–320, 2010.

B. Frénay and M. Verleysen. Parameter-insensitive kernel in extreme learning for non-linear support vector regression. *Neurocomputing*, 74(16):2526–2531, Sept. 2011. ISSN 09252312. doi: 10.1016/j.neucom.2010.11.037.

Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996. ISBN 1558604197. doi: 10.1.1.133.1040.

K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989. ISSN 0893-6080.

V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using GPU. *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–6, 2008. ISSN 2160-7508. doi: 10.1109/CVPRW.2008.4563100.

V. Garcia, E. Debreuve, F. Nielsen, and M. Barlaud. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *Proceedings - International Conference on Image Processing, ICIP*, pages 3757–3760, 2010. ISBN 9781424479948. doi: 10.1109/ICIP.2010.5654017.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. ISSN 15324435. doi: 10.1162/153244303322753616.

N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, pages 1–74, Sept. 2011.

L. K. Hansen and P. Salamon. Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(10):993–1001, Oct. 1990. ISSN 0162-8828.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2001.

Q. He, C. Du, Q. Wang, F. Zhuang, and Z. Shi. A parallel incremental extreme SVM classifier. *Neurocomputing*, 74(16):2532–2540, 2011. ISSN 09252312 (ISSN).

Q. He, T. Shang, F. Zhuang, and Z. Shi. Parallel extreme learning machine for regression based on MapReduce. *Neurocomputing*, 102(0):52–58, Feb. 2013. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2012.01.040.

K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

G. Huang, S. Song, J. N. D. Gupta, and C. Wu. Semi-Supervised and Unsupervised Extreme Learning Machines. *IEEE Transactions on Cybernetics*, 44(12): 2405–17, Dec. 2014. ISSN 21682267. doi: 10.1109/TCYB.2014.2307349.

G. Huang, G.-B. Huang, S. Song, and K. You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48, Jan. 2015. ISSN 08936080. doi: 10.1016/j.neunet.2014.10.001.

G.-B. Huang. Reply to "Comments on "The extreme learning machine"". *IEEE Transactions on Neural Networks*, 19(8):1495–1496, 2008. ISSN 10459227 (ISSN).

G.-B. Huang. An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels. *Cognitive Computation*, 6(3):376–390, Apr. 2014. ISSN 1866-9956. doi: 10.1007/s12559-014-9255-2.

G.-B. Huang and L. Chen. Convex incremental extreme learning machine. *Neurocomputing*, 70(16-18):3056–3062, Oct. 2007. ISSN 09252312. doi: 10.1016/j.neucom.2007.10.008.

G.-B. Huang and L. Chen. Enhanced random search based incremental extreme learning machine. *Neurocomputing*, 71(16-18):3460–3468, Oct. 2008. ISSN 09252312. doi: 10.1016/j.neucom.2007.10.008.

G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2 of *2004 IEEE International Joint Conference on Neural Networks - Proceedings*, pages 985–990, Sch. of Elec. and Electron. Eng., Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore, 2004. doi: 10.1109/IJCNN.2004.1380068.

G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006a. ISSN 10459227 (ISSN). doi: 10.1109/TNN.2006.875977.

G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, Dec. 2006b. ISSN 09252312. doi: 10.1016/j.neucom.2005.12.126.

G.-B. Huang, X. Ding, and H. Zhou. Optimization method based extreme learning machine for classification. *Neurocomputing*, 74(1-3):155–163, Dec. 2010. ISSN 09252312. doi: 10.1016/j.neucom.2010.02.019.

G.-B. Huang, D. H. Wang, and Y. Lan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2(2):107–122, May 2011. ISSN 1868-8071. doi: 10.1007/s13042-011-0019-y.

J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis. CULA: hybrid GPU accelerated linear algebra routines. In *Defense*, volume 7705, pages 770502–770502–7, 2010. ISBN 9780819481696. doi: 10.1117/12.850538.

B. Igelnik and Y.-H. Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6(6):1320–9, Jan. 1995. ISSN 1045-9227. doi: 10.1109/72.471375.

P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 126:604–613, 1998. ISSN 00123692. doi: 10.4086/toc. 2012.v008a014.

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87, Feb. 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.1.79.

W. Johnson and J. Lindenstrauss. Extension of Lipschitz maps into a Hilbert space. *Contemp. Math.*, 26:189–206, 1984.

I. T. Jolliffe. Principal Component Analysis, Second Edition. *Encyclopedia of Statistics in Behavioral Science*, 30:487, 2002. ISSN 00401706. doi: 10.2307/ 1270093.

L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong. Representational Learning with Extreme Learning Machine for Big Data. *IEEE Intelligent Systems*, 28(6):31–34, 2013.

N. Kourentzes and 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*, pages 117–127, Porvoo, Finland, Sept. 2008.

A. Kraskov, H. Stögbauer, and P. Grassberger. Estimating mutual information. *Physical Review E*, 69(6), June 2004. ISSN 1539-3755. doi: 10.1103/PhysRevE. 69.066138.

J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998. ISSN 1052-6234. doi: 10.1137/ S1052623496303470.

Y. Lan, Y. C. Soh, and G.-B. Huang. Ensemble of online sequential extreme learning machine. *Neurocomputing*, 72(13-15):3391–3395, Aug. 2009. ISSN 09252312. doi: 10.1016/j.neucom.2009.02.013.

Q. Le, T. Sarlós, and A. J. Smola. Fastfood - Approximating Kernel Expansions in Loglinear Time. *International Conference on Machine Learning*, 28, 2013.

N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, Nov. 2006. ISSN 1045-9227. doi: 10.1109/TNN.2006.880583.

X. Liu, L. Wang, G.-B. Huang, J. Zhang, and J. Yin. Multiple kernel extreme learning machine. *Neurocomputing*, 149:253–264, Feb. 2015. ISSN 09252312. doi: 10.1016/j.neucom.2013.09.072.

Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12:1399–1404, 1999. ISSN 0893-6080. doi: 10.1016/S0893-6080(99)00073-8.

D. J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. ISBN 0521642981.

P.-G. Martinsson. Randomization: Making Very Large-Scale Linear Algebraic Computations Possible. Technical report, 2009.

MathWorks. MATLAB Parallel Computing Toolbox, 2011. URL `http://www.mathworks.com/products/parallel-computing`.

J. Matoušek. On variants of the Johnson-Lindenstrauss lemma. *Random Structures & Algorithms*, pages 142–156, 2008. doi: 10.1002/rsa.

M. D. McDonnell, M. D. Tissera, A. van Schaik, and J. Tapson. Fast, simple and accurate handwritten digit classification using extreme learning machines with shaped input-weights. *arXiv preprint arXiv:1412.8307*, Dec. 2014.

Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse. OP-ELM: optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, Oct. 2010. ISSN 10459227 (ISSN). doi: 10.1109/TNN.2009.2036259.

Y. Miche, M. van Heeswijk, P. Bas, O. Simula, and A. Lendasse. TROP-ELM: A double-regularized ELM using LARS and Tikhonov regularization. *Neurocomputing*, 74(16):2413–2421, Sept. 2011. ISSN 09252312. doi: 10.1016/j.neucom.2010.12.042.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

R. H. Myers. *Classical and Modern Regression with Applications, 2nd edition*. Duxbury, Pacific Grove, CA, USA, 1990.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

K. Neumann. *Reliability of Extreme Learning Machines*. Ph.D. Thesis, Bielefeld University, 2013.

K. Neumann and J. J. Steil. Batch intrinsic plasticity for extreme learning machines. In *Artificial Neural Networks and Machine Learning - ICANN 2011*, volume 6791 LNCS of *21st International Conference on Artificial Neural Networks, ICANN 2011*, pages 339–346, Research Institute for Cognition and Robotics (CoR-Lab.), Bielefeld University, Universit{ä}tsstr. 25, 33615 Bielefeld, Germany, 2011. Springer. ISBN 978-3-642-21734-0.

K. Neumann and J. J. Steil. Optimizing Extreme Learning Machines via Ridge Regression and Batch Intrinsic Plasticity. *Neurocomputing*, 102(0):555–560, Feb. 2013. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2012.01.041.

K. Neumann, C. Emmerich, and J. J. Steil. Regularization by Intrinsic Plasticity and its Synergies with Recurrence for Random Projection Methods. *Journal of Intelligent Learning Systems and Applications*, 4(3):230–246, 2012. ISSN 2150-8402.

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*, pages 139–148, Porvoo, Finland, Sept. 2008.

Y.-H. Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, May 1992. ISSN 0018-9162. doi: 10.1109/2.144401.

Y.-H. Pao, G.-H. Park, and D. J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180, Apr. 1994. ISSN 09252312. doi: 10.1016/0925-2312(94)90053-1.

E. Parviainen, J. Riihimäki, Y. Miche, and A. Lendasse. Interpreting extreme learning machine as an approximation to an infinite neural network. In *KDIR 2010 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval*, International Conference on Knowledge Discovery and Information Retrieval, KDIR 2010, pages 65–73, Dept. of Biomedical Engineering and Computational Science (BECS), Aalto University School of Science and Technology, Aalto, Finland, 2010. ISBN 978-989-8425-28-7 (ISBN).

A. Rahimi and B. Recht. Random Features for Large-Scale Kernel Machines. *Advances in Neural Information Processing Systems*, 2007.

A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. *Advances in Neural Information Processing Systems*, 2008.

T. Raiko and H. Valpola. Deep Learning Made Easier by Linear Transformations in Perceptrons. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, pages 924–932, 2012.

C. R. Rao and S. K. Mitra. *Generalized Inverse of the Matrix and Its Applications*. John Wiley & Sons Inc, Jan. 1971.

J. Reunanen. Overfitting in Making Comparisons Between Variable Selection Methods. *Journal of Machine Learning Research*, 3:1371–1382, 2003. ISSN 15324435. doi: 10.1162/153244303322753715.

H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu. A fast pruned-extreme learning machine for classification problem. *Neurocomputing*, 72(1-3):359–366, Dec. 2008. ISSN 09252312. doi: 10.1016/j.neucom.2008.01.005.

F. Rossi, A. Lendasse, D. François, V. Wertz, and M. Verleysen. Mutual information for the selection of relevant variables in spectrometric nonlinear modelling. *Chemometrics and Intelligent Laboratory Systems*, 80(2):215–226, Feb. 2006. doi: doi:10.1016/j.chemolab.2005.06.010.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. ISSN 0028-0836. doi: 10.1038/323533a0.

R. E. Schapire, Y. Freund, P. L. Bartlett, and W. S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26:322–330, 1998.

W. Schmidt, M. Kraaijveld, and R. Duin. Feedforward neural networks with random weights. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4. IEEE Comput. Soc. Press, 1992. ISBN 0-8186-2915-0. doi: 10.1109/ICPR.1992.201708.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978.

T. Similä and J. Tikka. Multiresponse sparse regression with application to multidimensional scaling. In *Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005*, volume 3697, pages 97–102, 2005.

D. Sovilj. *Learning Methods for Variable Selection and Time Series Prediction*. Ph.D. Thesis, Aalto University, 2014.

J. J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20 (3):353–364, Apr. 2007. ISSN 0893-6080.

J. Tapson, P. de Chazal, and A. van Schaik. Explicit Computation of Input Weights in Extreme Learning Machines. *Proceedings of ELM-2014 Volume 1: Algorithms and Theories*, pages 41–49, 2014.

H. A. B. te Braake and G. van Straten. Random activation weight neural net (RAWN) for fast non-iterative training. *Engineering Applications of Artificial Intelligence*, 8(1):71–80, Feb. 1995. ISSN 0952-1976.

H. A. B. te Braake, H. J. L. van Can, G. van Straten, and H. B. Verbruggen. Regulated activation weights neural network (RAWN). *ESANN 1996, European Symposium on Artificial Neural Networks, Bruges, Belgium*, (April):19–24, 1996.

H. A. B. te Braake, H. J. L. van Can, G. van Straten, and H. B. Verbruggen. Two-step approach in the training of regulated activation weight neural networks (RAWN). *Engineering Applications of Artificial Intelligence*, 10(2):157–170, Apr. 1997. ISSN 0952-1976.

J. Triesch. A Gradient Rule for the Plasticity of a Neuron's Intrinsic Excitability. In W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, editors, *Artificial Neural Networks: Biological Inspirations - ICANN 2005*, volume 3696 of *Lecture Notes in Computer Science*, pages 65–70. Springer Berlin / Heidelberg, 2005a. ISBN 978-3-540-28752-0. doi: 10.1007/11550822_11.

J. Triesch. Synergies between intrinsic and synaptic plasticity in individual model neurons. In *NIPS*, 2005b.

M. van Heeswijk. *Adaptive Ensembles of Extreme Learning Machines for Time Series Prediction*. M.Sc Thesis, Eindhoven University of Technology, 2009.

M. Verleysen, D. François, G. Simon, and V. Wertz. On the effects of dimensionality on data analysis with neural networks. In *Artificial Neural Nets Problem Solving Methods*, volume 2687, pages 105–112. 2003. ISBN 978-3-540-40211-4. doi: 10.1007/3-540-44869-1_14.

M. Verleysen, F. Rossi, and D. François. Advances in feature selection with mutual information. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5400, pages 52–69, 2009. ISBN 9783642018046. doi: 10.1007/978-3-642-01805-3_4.

D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, Apr. 2007. ISSN 0893-6080.

L. P. Wang and C. R. Wan. Comments on "The extreme learning machine". *IEEE Transactions on Neural Networks*, 19(8):1494–1495, Aug. 2008. ISSN 10459227 (ISSN). doi: 10.1109/TNN.2008.2002273.

N. Wang, M. Han, N. Dong, and M. J. Er. Constructive multi-output extreme learning machine with application to large tanker motion dynamics identification. *Neurocomputing*, 128:59–72, Mar. 2014. ISSN 09252312. doi: 10.1016/j.neucom.2013.01.062.

A. S. Weigend and N. A. Gershenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison-Wesley, Reading, 1993.

B. Widrow, A. Greenblatt, Y. Kim, and D. Park. The No-Prop algorithm: A new learning algorithm for multilayer neural networks. *Neural Networks*, 37:182–188, 2013. ISSN 08936080. doi: 10.1016/j.neunet.2012.09.020.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992. ISSN 08936080. doi: 10.1016/S0893-6080(05)80023-1.

J. Y. Yam and T. W. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219–232, Jan. 2000. ISSN 09252312. doi: 10.1016/S0925-2312(99)00127-7.

Y. F. Yam and T. W. S. Chow. Determining initial weights of feedforward neural networks based on least squares method. *Neural Processing Letters*, 2(2):13–17, Mar. 1995. ISSN 1370-4621. doi: 10.1007/BF02312350.

Y. F. Yam, T. W. S. Chow, and C. T. Leung. A new method in determining initial weights of feedforward neural networks for training enhancement. *Neurocomputing*, 16:23–32, 1997. ISSN 09252312. doi: 10.1016/S0925-2312(96)00058-6.

Publication I

Mark van Heeswijk, Yoan Miche, Tiina Lindh-Knuutila, Peter A.J. Hilbers, Timo Honkela, Erkki Oja, and Amaury Lendasse. Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction. In *LNCS 5769 - Artificial Neural Networks, ICANN'09: International Conference on Artificial Neural Networks*, pp. 305-314, September 2009.

Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction*

Mark van Heeswijk[1,3,**], Yoan Miche[1,2], Tiina Lindh-Knuutila[1,4], Peter A.J. Hilbers[3], Timo Honkela[1], Erkki Oja[1], and Amaury Lendasse[1]

[1] Adaptive Informatics Research Centre, Helsinki University of Technology
P.O. Box 5400, 02015 TKK - Finland
heeswijk@cis.hut.fi
[2] INPG Grenoble - Gipsa-Lab, UMR 5216
961 rue de la Houille Blanche, Domaine Universitaire, 38402 Grenoble - France
[3] Eindhoven University of Technology
Den Dolech 2, P.O. Box 513, 5600 MB Eindhoven - The Netherlands
[4] International Computer Science Institute of University of California,
947 Center Street, Suite 600, Berkeley, CA 94704 - USA

Abstract. In this paper, we investigate the application of adaptive ensemble models of Extreme Learning Machines (ELMs) to the problem of one-step ahead prediction in (non)stationary time series. We verify that the method works on stationary time series and test the adaptivity of the ensemble model on a nonstationary time series. In the experiments, we show that the adaptive ensemble model achieves a test error comparable to the best methods, while keeping adaptivity. Moreover, it has low computational cost.

Keywords: time series prediction, sliding window, extreme learning machine, ensemble models, nonstationarity, adaptivity.

1 Introduction

Time series prediction is a challenge in many fields. In finance, experts predict stock exchange courses or stock market indices; data processing specialists predict the flow of information on their networks; producers of electricity predict the load of the following day [1,2]. The common question in these problems is how one can analyze and use the past to predict the future.

A common assumption in the field of time series prediction is that the underlying process generating the data is stationary and that the data points are independent and identically distributed (IID). Under this assumption, the training data is generally a good indication for what data to expect in the test phase.

However, a large number of application areas of prediction involve nonstationary phenomena. In these systems, the IID assumption does not hold since the

* This work was supported by the Academy of Finland Centre of Excellence, Adaptive Informatics Research Centre.

** Corresponding author.

system generating the time series changes over time. Therefore, contrary to the stationary case, one cannot assume that one can use what has been learned from past data and one has to keep learning and adapting the model as new samples arrive. Possible ways of doing this include: 1) retraining the model repeatedly on a finite window of past values and 2) using a combination of different models, each of which is specialized on part of the state space.

Besides the need to deal with nonstationarity, another motivation for such an approach is that one can drop stationarity requirements on the time series. This is very useful, since often we cannot assume anything about whether or not a time series is stationary.

To construct the ensemble model presented in this paper, a number of Extreme Learning Machines (ELMs) [3] of varying complexity are generated, each of which is individually trained on the data. After training, these individual models are combined in an ensemble model. The output of the ensemble model is a weighted linear combination of the outputs of the individual models. During the test phase, the ensemble model adapts this linear combination over time with the goal of minimizing the prediction error: whenever a particular model has bad prediction performance (relative to the other models) its weight in the ensemble is decreased, and vice versa. A detailed description can be found in Section 2.3.

In the first experiment, we test the performance of this adaptive ensemble model in repeated one-step ahead prediction on a time series that is known to be stationary (the Santa Fe A Laser series [4]). The main goal of this experiment is to test the robustness of the model and to investigate the different parameters influencing the performance of the model. In the second experiment, the model is applied to another time series (Quebec Births [5]) which is nonstationary and more noisy than the Santa Fe time series.

Ensemble methods have been applied in various forms (and under various names) to time series prediction, regression and classification. A non-exhaustive list of literature that discusses the combination of different models into a single model includes *bagging* [6], *boosting* [7], *committees* [8], *mixture of experts* [9], *multi-agent systems for prediction* [10], *classifier ensembles* [11], among others. Out of these examples, our work is most closely related to [10], which describes a multi-agent system prediction of financial time series and recasts prediction as a classification problem. Other related work includes [11], which deals with classification under concept drift (nonstationarity of classes). The difference is that both papers deal with classification under nonstationarity, while we deal with regression under nonstationarity.

In Section 2, the theory of ensemble models and the ELM are presented, as well as how we combine both of them in the adaptive ensemble method. Section 3 describes the experiments, the datasets used and discusses the results.

2 Methodology

2.1 Ensemble Models

In ensemble methods, several individual models are combined to form a single new model. Commonly, this is done by taking the average or a weighted average of the individual models, but other combination schemes are also possible [11]. For example, one could take the best n models and take a linear combination of those. For an overview of ensemble methods, see [8].

Ensemble methods rely on having multiple good models with sufficiently uncorrelated error. The individual models are typically combined into a single ensemble model as follows:

$$\hat{y}_{ens}(t) = \frac{1}{m}\sum_{i=1}^{m}\hat{y}_i(t)), \tag{1}$$

where $\hat{y}_{ens}(t)$ is the output of the ensemble model, $\hat{y}_i(t)$ are the outputs of the individual models and m is the number of models.

Following [8], it can be shown that the variance of the ensemble model is lower than the average variance of all the individual models:

Let $y(t)$ denote the true output that we are trying to predict and $\hat{y}_i(t)$ the estimation for this value of model i. Then, we can write the output $\hat{y}_i(t)$ of model i as the true value $y(t)$ plus some error term $\epsilon_i(t)$:

$$\hat{y}_i(t) = y(t) + \epsilon_i(t). \tag{2}$$

Then the expected square error of a model becomes

$$\mathbb{E}[\{\hat{y}_i(t) - y(t)\}^2] = \mathbb{E}[\epsilon_i(t)^2]. \tag{3}$$

The average error made by a number of models is given by

$$E_{avg} = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}[\epsilon_i(t)^2]. \tag{4}$$

Similarly, the expected error of the ensemble as defined in Equation 1 is given by

$$E_{ens} = \mathbb{E}\Big[\Big\{\frac{1}{m}\sum_{i=1}^{m}\hat{y}_i(t) - y(t)\Big\}^2\Big] = \mathbb{E}\Big[\Big\{\frac{1}{m}\sum_{i=1}^{m}\epsilon_i(t)\Big\}^2\Big]. \tag{5}$$

Assuming the errors $\epsilon_i(t)$ are uncorrelated (i.e. $E[\epsilon_i(t)\epsilon_j(t)] = 0$) and have zero mean ($E[\epsilon_i(t)] = 0$), we get

$$E_{ens} = \frac{1}{m}E_{avg}. \tag{6}$$

Note that these equations assume completely uncorrelated errors between the models, while in practice errors tend to be highly correlated. Therefore, errors are often not reduced as much as suggested by these equations, but can be improved by using ensemble models. It can be shown that $E_{ens} < E_{avg}$ always holds. Note that this only tells us that the test error of the ensemble is smaller than the average test error of the models, and that it is not necessarily better than the best model in the ensemble. Therefore, the models used in the ensemble should be sufficiently accurate.

2.2 ELM

The ELM algorithm is proposed by Guang-Bin Huang *et al.* in [3] and makes use of Single-Layer Feedforward Neural Networks (SLFN). The main concept behind ELM lies in the random initialization of the SLFN weights and biases. Under the condition that the transfer functions in the hidden layer are infinitely differentiable, the optimal output weights for a given training set can be determined analytically. The obtained output weights minimize the square training error. The trained network is thus obtained in very few steps and is very fast to train, which is why we use them in the adaptive ensemble model.

Below, we review the main concepts of ELM as presented in [3]. 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

$$\sum_{i=1}^{N} \beta_i f(\mathbf{w}_i \mathbf{x}_j + b_i), j \in [1, M], \tag{7}$$

with f being the activation function, $\mathbf{w}_i$ the input weights to the i^{th} neuron in the hidden layer, b_i the 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

$$\sum_{i=1}^{N} \beta_i f(\mathbf{w}_i \mathbf{x}_j + b_i) = y_j, j \in [1, M], \tag{8}$$

which can be written compactly as

$$\mathbf{H}\beta = \mathbf{Y}, \tag{9}$$

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

$$\mathbf{H} = \begin{pmatrix} f(\mathbf{w}_1\mathbf{x}_1 + b_1) & \cdots & f(\mathbf{w}_N\mathbf{x}_1 + b_N) \\ \vdots & \ddots & \vdots \\ f(\mathbf{w}_1\mathbf{x}_M + b_1) & \cdots & f(\mathbf{w}_N\mathbf{x}_M + b_N) \end{pmatrix}, \tag{10}$$

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

Given the randomly initialized first layer of the ELM and the training inputs $\mathbf{x}_i \in \mathbb{R}^d$, the hidden layer output matrix $\mathbf{H}$ can be computed. Now, given $\mathbf{H}$ and the target outputs $y_i \in \mathbb{R}$ (i.e. $\mathbf{Y}$), the output weights β can be solved from the linear system defined by Equation 9. This solution is given by $\beta = \mathbf{H}^\dagger\mathbf{Y}$, where $\mathbf{H}^\dagger$ is the Moore-Penrose generalized inverse of the matrix $\mathbf{H}$ [12]. This solution for β is the unique least-squares solution to the equation $\mathbf{H}\beta = \mathbf{Y}$. Overall, the ELM algorithm then is:

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 $\beta = \mathbf{H}^{\dagger}\mathbf{Y}$.
```

Theoretical proofs and a more thorough presentation of the ELM algorithm are detailed in the original paper [3].

2.3 Adaptive Ensemble Model of ELMs

When creating a model to solve a certain regression or classification problem, it is unknown in advance what the optimal model complexity and architecture is. Also, we cannot always assume stationarity of the process generating the data (i.e. in cases where the IID assumption does not hold). Therefore, since the information that has been gathered from past samples can become inaccurate, it is needed to keep learning and keep adapting the model once new samples become available. Possible ways of doing this include: 1) retraining the model repeatedly on a finite window into the past and 2) use a combination of different models, each of which is specialized on part of the state space. In this paper, we employ both strategies in repeated one-step ahead prediction on (non)stationary time series. On the one hand, we use diverse models and adapt the weights with which these models contribute to the ensemble. On the other hand, we retrain the individual models on a limited number of past values (sliding window) or on all known values (growing window).

Adaptation of the Ensemble. The ensemble model consists of a number of randomly initialized ELMs, which each have their own parameters (details are discussed in the next subsection). The model ELM_i has an associated weight w_i which determines its contribution to the prediction of the ensemble. Each ELM is individually trained on the training data and the outputs of the ELMs contribute to the output $\hat{y}_{ens}$ of the ensemble as follows: $\hat{y}_{ens}(t+1) = \sum_i w_i \hat{y}_i(t+1)$.

Once initial training of the models on the training set is done, repeated one-step ahead prediction on the 'test' set starts. After each time step, the previous predictions $\hat{y}_i(t-1)$ are compared with the real value $y(t-1)$. If the square error $\epsilon_i(t-1)^2$ of ELM_i is larger than the average square error of all models at time step $t-1$, then the associated ensemble weight w_i is decreased, and vice versa. The rate of change can be scaled with a parameter α, called the learning rate. Furthermore, the rate of change is normalized by the number of models and the variance of the time series, such that we can expect similar behaviour on time series with different variance and ensembles with a different number of models. The full algorithm can be found in Algorithm 2.

Adaptation of the Models. As described above, ELMs are used in the ensemble model. Each ELM has a random number of input neurons, random number of hidden neurons, and random variables of the regressor as input.

Besides changing the ensemble weights w_i as a function of the errors of the individual models at every time step, the models themselves are also retrained. Before making a prediction at time step t, each model is either retrained on a past window of n values $(\mathbf{x}_i, y_i)_{t-n}^{t-1}$ (sliding window), or on all values known so far $(\mathbf{x}_i, y_i)_1^{t-1}$ (growing window). Details on how this retraining fits in with the rest of the ensemble can be found in Algorithm 2.

As mentioned in Section 2.2, ELMs are very fast to train. In order to further speed up the retraining of the ELMs, we make use of PRESS statistics, which allow you to add and remove samples from the training set of a linear model and give you the linear model that you would have obtained, had you trained it on the modified training set. Since an ELM is essentially a linear model of the responses of the hidden layer, PRESS statistics can be applied to (re)train the ELM in an incremental way. A detailed discussion of incremental training of ELMs with PRESS statistics falls outside the scope of this paper, but details on PRESS statistics can be found in [13].

Algorithm 2. Adaptive Ensemble of ELMs

```
Given a set $(\mathbf{x}(t), y(t)), \mathbf{x}(t) \in \mathbb{R}^d, y(t) \in \mathbb{R}$, and $m$ models,
 1: Create $m$ random ELMs: $(ELM_1 \ldots ELM_m)$
 2: Train each of the ELMs individually on the training data
 3: Initialize each $w_i$ to $\frac{1}{m}$
 4: while $t < t_{end}$ do
 5:     generate predictions $\hat{y}_i(t+1)$
 6:     $\hat{y}_{ens}(t+1) = \sum_i w_i \hat{y}_i(t+1)$
 7:     t = t + 1
 8:     compute all errors $\rightarrow \epsilon_i(t-1) = \hat{y}_i(t-1) - y(t-1)$
 9:     for i = 1 to #models do
10:         $\Delta w_i = -\epsilon_i(t-1)^2 + mean(\epsilon(t-1)^2)$
11:         $\Delta w_i = \Delta w_i \cdot \alpha / (\#models \cdot \mathbf{var}(y))$
12:         $w_i = \max(0, w_i + \Delta w_i)$
13:         Retrain $ELM_i$
14:     end for
15:     renormalize weights $\rightarrow \mathbf{w} = \mathbf{w} / ||\mathbf{w}||$
16: end while
```

3 Experiments

3.1 Experiment 1: Stationary Time Series

The Santa Fe Laser Data time series [4] has been obtained from a far-infrared-laser in a chaotic state. This time series has become a well-known benchmark

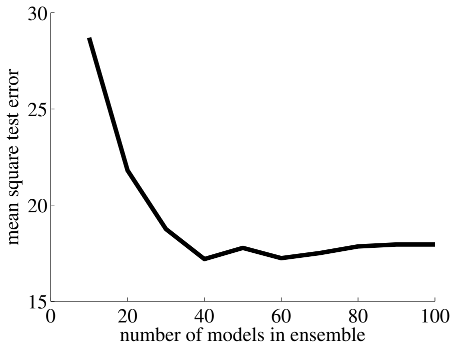


Fig. 1. MSE$_{test}$ of ensemble on laser time series for varying number of models (no window retraining, learning rate 0.1)

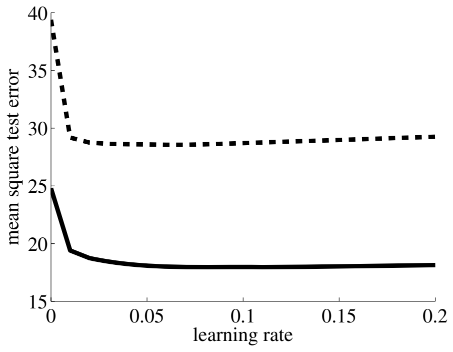


Fig. 2. MSE$_{test}$ of ensemble on laser time series as a function of learning rate (no window retraining), for 10 models (*dotted line*) and 100 models (*solid line*)

in time series prediction since the Santa Fe competition in 1991. It consists of approximately 10000 points and the time series is known to be stationary.

The adaptive ensemble model is trained on the first 1000 values of the time series, after which sequential one-step ahead prediction is performed on the following 9000 values. This experiment is repeated for various combinations of learning rate α and number of models in the ensemble. Each ELM has a regressor size of 8 (of which 5 to 8 variables are randomly selected) and between 150 and 200 hidden neurons with a sigmoid transfer function.

Figure 1 shows the effect of the number of models on the prediction accuracy. It can be seen that the number of models strongly influences the prediction accuracy and that at least 60 models are needed to get good prediction accuracy. Figure 2 shows the effect of the learning rate on the prediction accuracy. The influence of the various (re)training strategies can be found in Table 1. This table also shows that the method is able to achieve a prediction accuracy comparable to the best methods [14].

Table 1. MSE$_{test}$ of ensemble for laser (training window size 1000)

		retraining		
learning rate	#models	none	sliding	growing
0.0	10	39.39	58.56	34.16
0.1	10	28.70	37.93	18.42
0.0	100	24.80	33.85	20.99
0.1	100	17.96	27.30	**14.64**

Figures 3 and 4 show the adaptation of some of the ensemble weights over the length of the entire prediction task.

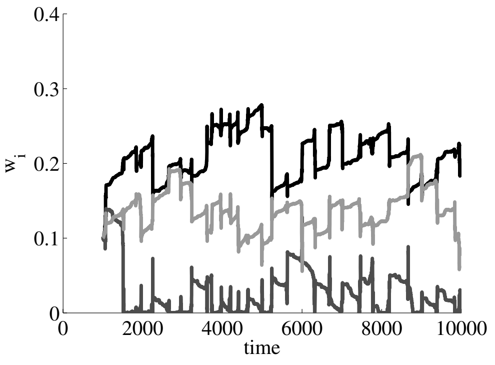


Fig. 3. plot showing part of the ensemble weights w_i adapting over time during sequential prediction on laser time series (#models=10, learning rate=0.1, no window retraining)

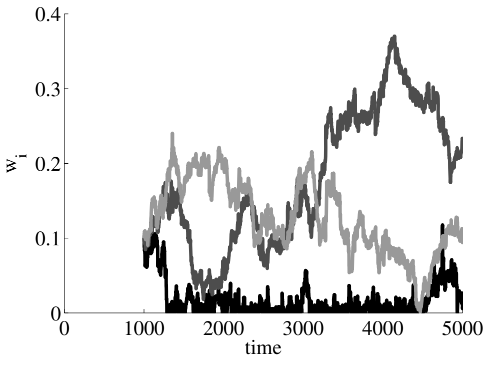


Fig. 4. plot showing part of the ensemble weights w_i adapting over time during sequential prediction on qbirths time series (#models=10, learning rate=0.1, no window retraining)

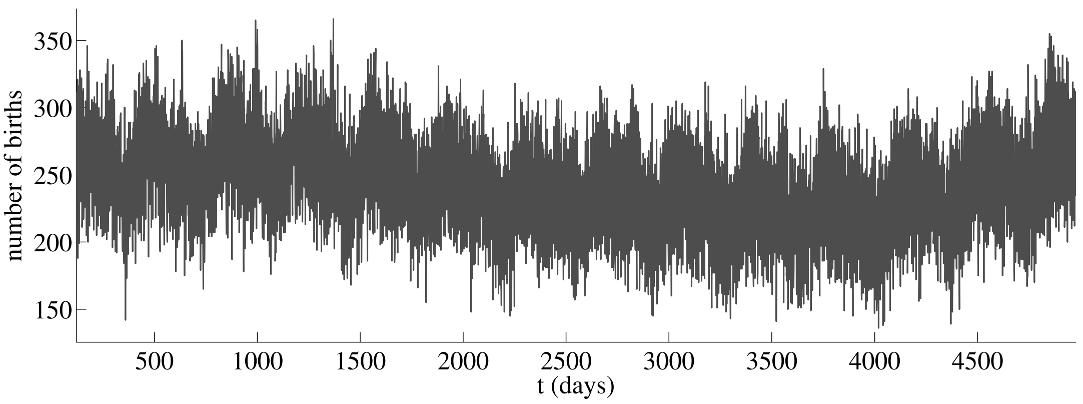


Fig. 5. The Quebec Births time series

3.2 Experiment 2: Nonstationary Time Series

The Quebec Births time series [5] consists of the number of daily births in Quebec over the period of January 1, 1977 to December 31, 1990. It consists of approximately 5000 points, is nonstationary and more noisy than the Santa Fe Laser Data.

The adaptive ensemble model is trained on the first 1000 values of the time series, after which sequential one-step ahead prediction is performed on the following 5000 values. This experiment is repeated for varying learning rates α and numbers of models. Each ELM has a regressor size of 14 (of which 12 to 14 variables are randomly selected) and between 150 and 200 hidden neurons.

Figure 6 shows the effect of the number of models on the prediction accuracy. It can be seen that the number of models strongly influences the prediction accuracy, as was the case with the Santa Fe time series. However, we need more

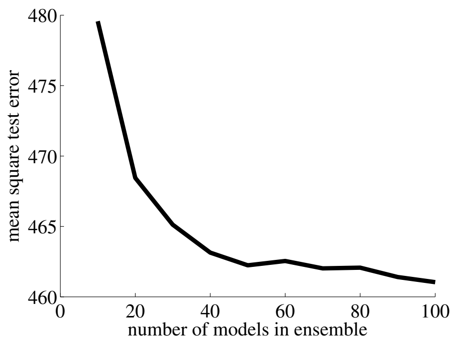


Fig. 6. MSE_{test} of ensemble on Quebec Births time series for varying number of models (sliding window retraining, learning rate 0.1)

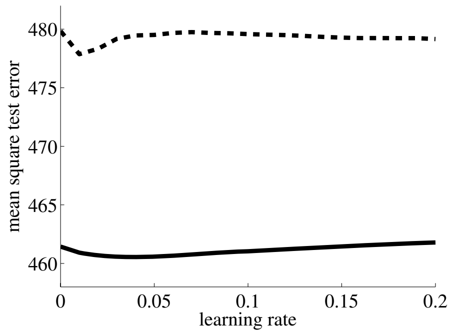


Fig. 7. MSE_{test} of ensemble on Quebec Births time series as a function of learning rate (retraining with sliding window of size 1000), for 10 models (*dotted line*) and 100 models (*solid line*)

Table 2. MSE_{test} of ensemble for Quebec Births (training windows size 1000)

		retraining		
learning rate	#models	none	sliding	growing
0.0	10	594.04	479.84	480.97
0.1	10	582.09	479.58	476.87
0.0	100	585.53	**461.44**	469.79
0.1	100	567.62	**461.04**	468.51

models in order to get a good prediction accuracy. Figure 7 shows the effect of the learning rate on the prediction accuracy. The influence of the various (re)training strategies can be found in Table 2.

3.3 Discussion

The experiments clearly show that it is important to have a sufficient number of models (more is generally better). Furthermore, the shape of the learning rate graph is independent of the number of models, which means that these parameters can probably be optimized independently from each other. We are currently performing a more thorough statistical analysis for determining the best strategy for optimizing the parameters. However, the results suggest that choosing the number of models high and choosing a sufficiently large learning rate (i.e. $\alpha = 0.1$) is a good and robust strategy.

The results also show that the proposed adaptive ensemble method is able to achieve a prediction accuracy comparable to the best methods [14], and is able to do so for both stationary and nonstationary series. An added advantage of the method is that it is adaptive and has low computational cost.

4 Conclusions

We have presented an adaptive ensemble model of Extreme Learning Machines (ELMs) for use in repeated one-step ahead prediction. The model has been tested on both stationary and nonstationary time series, and these experiments show that in both cases the adaptive ensemble method is able to achieve a prediction accuracy comparable to the best methods. An added advantage of the method is that it is adaptive and has low computational cost. Furthermore, the results suggest that we can make good guesses for the parameters of the method (i.e. choose number of models sufficiently large and choose learning parameter $\alpha = 0.1$). We are currently performing more thorough statistical analysis of the model, in order to determine the best strategy for optimizing the parameters. In addition, we would like to extend the model to include other models like OP-ELM [15] and investigate how we can guide adding new models to the ensemble in an online fashion, in order to introduce an extra degrees of adaptivity.

References

1. Sorjamaa, A., Hao, J., Reyhani, N., Ji, Y., Lendasse, A.: Methodology for long-term prediction of time series. Neurocomputing 70(16-18), 2861–2869 (2007)
2. Simon, G., Lendasse, A., Cottrell, M., Fort, J.-C., Verleysen, M.: Time series forecasting: Obtaining long term trends with self-organizing maps. Pattern Recognition Letters 26(12), 1795–1808 (2005)
3. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: Theory and applications. Neurocomputing 70(1-3), 489–501 (2006)
4. Weigend, A., Gershenfeld, N.: Time Series Prediction: Forecasting the Future and Understanding the Past. Addison-Wesley, Reading (1993)
5. Quebec Births Data, `http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/misc/qbirths.dat`
6. Breiman, L.: Bagging predictors. In: Machine Learning, pp. 123–140 (1996)
7. Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the margin: a new explanation for the effectiveness of voting methods. The Annals of Statistics 26, 322–330 (1998)
8. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, Secaucus (2006)
9. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural Computation 3, 79–87 (1991)
10. Raudys, S., Zliobaite, I.: The multi-agent system for prediction of financial time series. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 653–662. Springer, Heidelberg (2006)
11. Kuncheva, L.I.: Classifier ensembles for changing environments. MCS, 1–15 (2004)
12. Rao, C.R., Mitra, S.K.: Generalized Inverse of Matrices and Its Applications. John Wiley & Sons Inc., Chichester (1972)
13. Myers, R.H.: Classical and Modern Regression with Applications, 2nd edn. Duxbury, Pacific Grove (1990)
14. Suykens, J.A.K., Gestel, T.V., Brabanter, J.D., Moor, B.D., Vandewalle, J.: Least Squares Support Vector Machines. World Scientific, Singapore (2002)
15. Miche, Y., Sorjamaa, A., Lendasse, A.: OP-ELM: Theory, experiments and a toolbox. In: Kůrková, V., Neruda, R., Koutník, J. (eds.) ICANN 2008, Part I. LNCS, vol. 5163, pp. 145–154. Springer, Heidelberg (2008)

Publication II

Mark van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, 74 (16): pp. 2430-2437, September 2011.

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) $y_i \in \mathbb{R}, 1 \le i \le M$ (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.

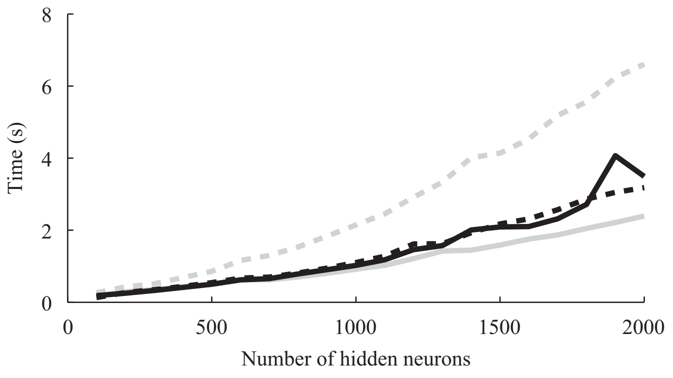


Fig. 1. Comparison of running times of ELM training (*solid lines*) and ELM training + model structure selection (*dotted lines*), with (*black lines*) and without (*gray lines*) explicitly computing and reusing $\mathbf{H}^{\dagger}$.

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.

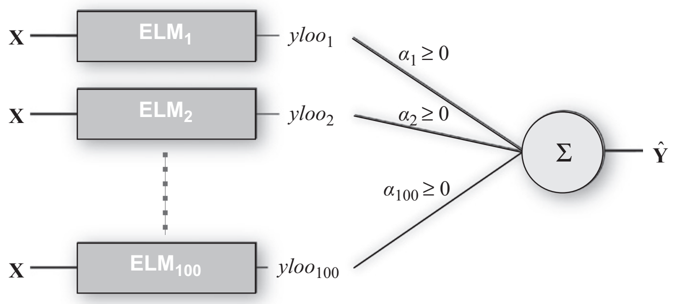


Fig. 2. Block diagram showing the overall setup of the ensemble of ELMs.

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

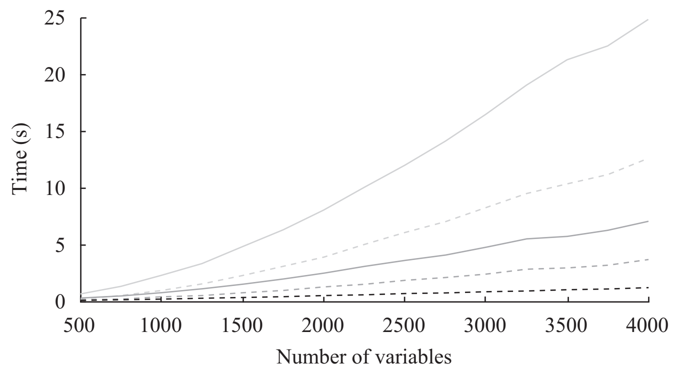


Fig. 3. Time (s) needed to solve a linear system of 5000 variables and one target variable, using `mldivide` (*light-gray lines*), `gels` (*gray lines*), `culaGels` (*dashed black line*) for double precision (*solid lines*) and single precision (*dashed lines*).

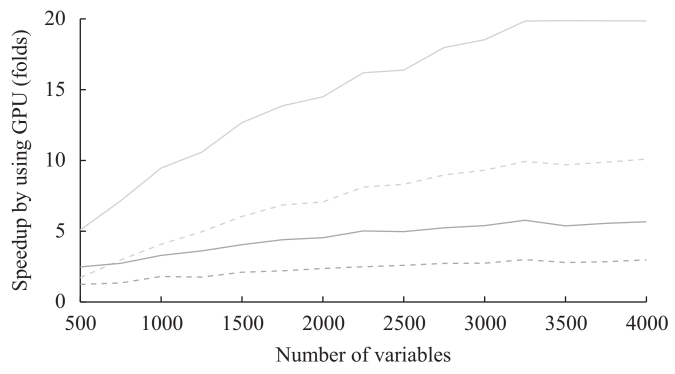


Fig. 4. Speedup achieved in solving system of 5000 variables and one target variable, by using `culaGels` instead of `mldivide`, (*light-gray lines*), `gels` (*gray lines*) for double precision (*solid lines*) and single precision (*dashed lines*).

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(`mldiv`$_{dp}$)/t(`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 (`mldiv`$_{dp}$) (s)	t (`gesv`$_{dp}$) (s)	t (`mldiv`$_{sp}$) (s)	t (`gesv`$_{sp}$) (s)	t (`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**

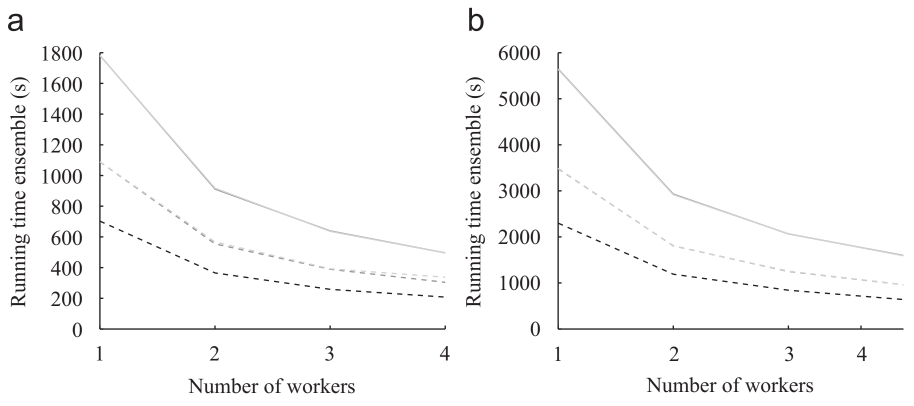


Fig. 5. Running times (in seconds) for running the entire ensemble in parallel on (a) Santa Fe and (b) ESTSP'08, for varying numbers of workers, using `mldivide` (*light-gray lines*), `gesv` (*gray lines*), `culaGesv` (*black line*) for double precision (*solid lines*) and single precision (*dashed lines*).

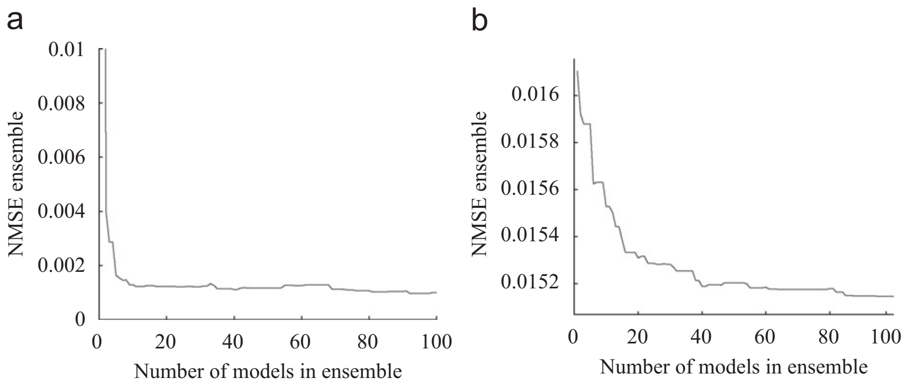


Fig. 6. NMSE of an ensemble model with varying number of models on (a) Santa Fe and (b) ESTSP'08.

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.

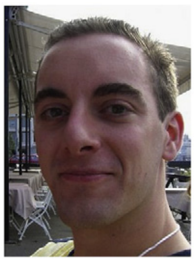
7. Conclusion and future work

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.

Publication III

Benoît Frenay, Mark van Heeswijk, Yoan Miche, Michel Verleysen, and Amaury Lendasse. Feature selection for nonlinear models with extreme learning machines. *Neurocomputing*, 102, pp. 111-124, February 2013.

Neurocomputing 102 (2013) 111–124

Contents lists available at SciVerse ScienceDirect

Neurocomputing

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

Feature selection for nonlinear models with extreme learning machines

Frénay Benoît [a,b,*], Mark van Heeswijk [b], Yoan Miche [b], Michel Verleysen [a], Amaury Lendasse [b,c,d]

[a] *Machine Learning Group, ICTEAM Institute, Université catholique de Louvain, BE 1348 Louvain-la-Neuve, Belgium*
[b] *Aalto University School of Science, Department of Information and Computer Science, P.O. Box 15400, FI-00076 Aalto, Finland*
[c] *IKERBASQUE, Basque Foundation for Science, 48011 Bilbao, Spain*
[d] *Computational Intelligence Group, Computer Science Faculty, University Of The Basque Country, Paseo Manuel Lardizabal 1, Donostia/San Sebastián, Spain*

ARTICLE INFO

Available online 7 June 2012

Keywords:
Extreme learning machines
Regression
Feature selection
Regularisation

ABSTRACT

In the context of feature selection, there is a trade-off between the number of selected features and the generalisation error. Two plots may help to summarise feature selection: the feature selection path and the sparsity-error trade-off curve. The feature selection path shows the best feature subset for each subset size, whereas the sparsity-error trade-off curve shows the corresponding generalisation errors. These graphical tools may help experts to choose suitable feature subsets and extract useful domain knowledge. In order to obtain these tools, extreme learning machines are used here, since they are fast to train and an estimate of their generalisation error can easily be obtained using the PRESS statistics. An algorithm is introduced, which adds an additional layer to standard extreme learning machines in order to optimise the subset of selected features. Experimental results illustrate the quality of the presented method.

1. Introduction

Feature selection is an important issue in machine learning. On the one hand, if not enough features are selected, prediction may be impossible. On the other hand, using all features may reveal impossible since the amount of available training data is usually small with respect to dimensionality. Aside from generalisation concerns, feature selection may also help experts to understand which features are relevant in a particular application. For example, in cancer diagnosis, feature selection may help to understand which genes are oncogenic. In industry, it is interesting to know which measures are actually useful to assess the quality of a product, since it allows reducing the measurement costs.

Usually there exists a trade-off between the number of selected features and the generalisation error [1]. Indeed, more features means more information, so an ideal model should perform better. However, the curse of dimensionality and the finite number of samples available for learning may harm this ideal view when too many features are considered. Another issue is that the best generalisation error is often not the only objective; interpretability of the selected features may also be a major requirement. Therefore there is often a need for the user to select the number of features by hand, with the help of appropriate tools.

For each fixed number of selected features, one may find (at least in principle) the optimal subset of features, giving the best generalisation error. However choosing between the subsets created in this way for various sizes might be difficult. Two plots may help to summarise feature selection: the feature selection path and the sparsity-error trade-off curve. The feature selection path shows the best feature subset for each subset size, whereas the sparsity-error trade-off curve shows the corresponding generalisation errors. From these plots, experts can choose suitable feature subsets and extract useful domain knowledge. Notice that the feature selection path and the sparsity-error trade-off curve are strongly related, for the latter allows choosing a feature subset in the former.

In real learning situations, the feature selection path and the sparsity-error trade-off curve can only be estimated, since both the target function and the data distribution are unknown. For linear regression problems, the LARS algorithm [2] is an efficient tool for finding the best features for linear models. However, the problem remains open for nonlinear regression problems and models.

For nonlinear problems, ranking methods can be used to rank features using e.g. mutual information [3,4]. Thereafter, feature subsets are built by adding features in the order defined by the ranking. However, feature subsets can evolve discontinuously for nonlinear problems: the best feature subset of size $d+1$ does not necessarily contain the best subset of size d [1]. Methods like

[*] Corresponding author at: Machine Learning Group, ICTEAM Institute, Université catholique de Louvain, BE 1348 Louvain-la-Neuve, Belgium. Tel.: +321 04 78 133; fax: +321 04 72 598.
E-mail address: benoit.frenay@uclouvain.be (F. Benoît).

forward or backward search [1] allow searching through the space of possible feature subsets, but they can only select or drop one feature at a time. Moreover, many possible feature selections must be considered at each iteration by such methods based on greedy search.

This paper proposes a new algorithm to build the feature selection path and the sparsity-error trade-off curve for nonlinear problems. Contrarily to e.g. forward search, the proposed iterative algorithm considers only one neighbour at each iteration. Yet, multiple features can enter or leave the current feature subset at each step of the search. Extreme learning machines are used since they are very fast to train and an estimate of their generalisation error can easily be computed [5–10]. The proposed method is theoretically and experimentally compared with other feature selection methods. Experiments show that the proposed algorithm obtains reliable estimates of the two plots: the feature selection path and the sparsity-error trade-off curve. In some cases, the proposed algorithm obtains (i) optimal test errors using less features and (ii) feature selection paths with more information, with respect to the paths obtained by the other feature selection algorithms used here for comparison.

The following of this paper is organised as follows. Section 2 discusses feature selection. Section 3 introduces the feature selection path and the sparsity-error trade-off curve and discusses how they can be used in practice. Section 4 proposes an algorithm and compares it theoretically with existing methods. Section 5 assesses the proposed algorithm experimentally and conclusions are drawn in Section 6.

2. Domain analysis and feature selection

In many applications, feature selection is necessary. Indeed, the number of available samples is usually small with respect to the data dimensionality. In that case, the curse of dimensionality prevents us from using all the features, since the necessary number of training samples grows exponentially with the dimensionality. Therefore, feature selection consists of choosing a trade-off between the number of selected features and the adequacy of the learned model. However, it is not always obvious what is a *good* feature subset.

A common criterion for assessing the quality of a subset of features is the generalisation error, i.e. the expected error for new samples. This criterion relates to the capacity of the model to generalise beyond training data. Sometimes, experts simply want to minimise the generalisation error. However, in some contexts, experts are searching for sparse feature subsets with only a few features because interpretability is a major concern. In such cases, the number of features is chosen in order to achieve sufficient generalisation. Limiting the number of features may also be necessary because of e.g. measurement costs. In conclusion, feature selection requires flexible tools which are able to adapt to specific user needs.

The next section discusses two strongly related tools for addressing common questions in feature selection situations: the feature selection path and the sparsity-error trade-off curve. Section 4 proposes an algorithm to estimate both of them in the case of nonlinear regression problems. In this paper, the focus is set on regression and the mean square error (MSE)

$$\frac{1}{n}\sum_{i=1}^{n}[t_i-\hat{f}(x_i^1,..,x_i^d|\theta)]^2 \tag{1}$$

is used, where $x_i=(x_i^1,\dots,x_i^d)$ is instance i, t_i is the target value, n is the number of samples and $\hat{f}$ is a function approximator with parameters θ.

3. Feature selection path and sparsity-error trade-off curve

Given a set of features, a feature selection path (FSP) shows the best feature subset for each subset size. Here, best feature subsets are selected in terms of generalisation error. Fig. 1 shows an estimate of the feature selection path (FSP) for an artificial problem, called here the XOR-like problem. The artificial dataset is built using six random features which are uniformly distributed in [0,1]. For each sample $x_i=(x_i^1,\dots,x_i^6)$, the target is

$$f(x_i)=x_i^1+(x_i^2>0.5)(x_i^3>0.5)+\epsilon_i \tag{2}$$

where (i) $(x>0.5)$ is equal to 1 when $x>0.5$ and is equal to 0 otherwise and (ii) ϵ_i is a noise with distribution $\mathcal{N}(0,0.1)$. This regression problem is similar to the XOR problem in classification: the product term can only be computed using both features 2 and 3. In order to have a sufficient number of data for the feature selection, 1000 training samples were generated. Fig. 1 is obtained with the approach proposed in this paper (see Sections 4 and 5). Each column corresponds to a subset size, where black cells correspond to selected features. Rows correspond to features. In essence, a feature selection path is very similar to the plots in Efron et al. [2], which show estimates of regression coefficients for different coefficient sparsities.

Each feature subset corresponds to a generalisation error. Indeed, for each subset size, one can estimate how well the selected features allow generalising to new samples. These generalisation errors are required in order to choose one of the feature subsets in the FSP. Therefore, one obtains a sparsity-error trade-off (SET) curve, which shows the best achievable generalisation error for the different feature set sizes. Here, sparsity refers to the size of the feature subset itself: sparse feature subsets contain less features.

Fig. 2 shows an estimate of the sparsity-error trade-off (SET) curve for the XOR-like problem, where the generalisation errors correspond to the feature subsets given in Fig. 1. The SET curve shows that the generalisation error is large when only a few features are selected, i.e. when the feature subset is too sparse. The generalisation error improves quickly as sparsity decreases and achieves its optimum for three features. Then, the generalisation error starts to increase, because of the curse of dimensionality. Indeed, the number of training samples becomes too small with respect to the dimensionality.

Using the feature selection path and the sparsity-error trade-off curve, experts can answer many questions. It is possible to see e.g. which features are useful, which features are necessary to achieve correct results, which features do not seem to be worth collecting, etc. These questions cannot be answered if one only

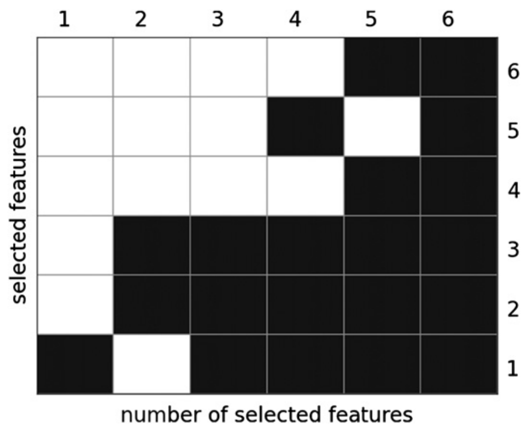


Fig. 1. Estimate of the feature selection path for the XOR-like problem. Columns and rows correspond to subset sizes and features, respectively.

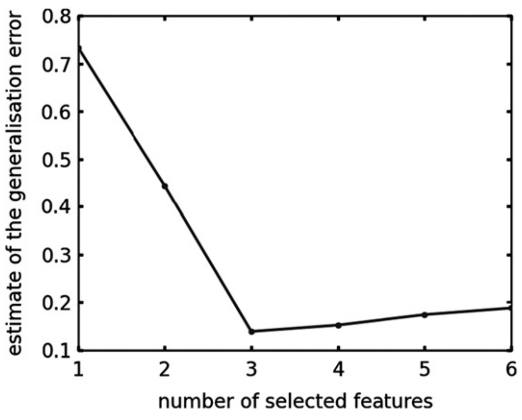


Fig. 2. Estimate of the sparsity-error trade-off curve for the XOR-like problem.

has the best feature subset: the path of feature subsets is necessary, as well as the corresponding generalisation errors.

Let us shortly discuss the XOR-like problem using the FSP and the SET curve in Figs. 1 and 2. Here, three features are sufficient to achieve optimal models. Indeed, the estimate of the generalisation error has reached its minimum value. Notice that the selected features are the relevant features in Eq. (2).

The FSP provides important additional information: features 2 and 3 should be selected together. Indeed, when only one feature is selected, the feature subset is {1}. But when two feature are selected, feature 1 is no longer used. Instead, features 2 and 3 are selected jointly. This cannot be seen when looking only at the optimal feature subset { 1,2,3}. The FSP reflects Eq. (2), where the target depends on a nonlinear combination of features 2 and 3.

4. Estimating FSPs and SET curves

In practice, the true FSP and the true SET curve are impossible to obtain. Indeed, both the true approximated functional and the true data distribution are unknown. Instead, one has to rely on estimates. This section reviews existing approaches and introduces a new algorithm in order to overcome their weaknesses.

4.1. *Estimating the generalisation error*

In order to estimate the SET curve, it is necessary to choose an estimator of the generalisation error. The generalisation error corresponds to the expected value of the error on new, unknown samples. Hence, techniques like e.g. cross-validation or bootstrap can be used [11,12]. Namely, these methods use the available data to build a training set and a test set. A model is trained using training data and tested on test data. The resulting error gives an estimate of the generalisation error, since none of the test samples have been used for training. The process can be repeated to obtain reliable estimates.

It should be pointed out that both cross-validation and bootstrap estimate the generalisation of a given model, not the best possible generalisation error. Therefore, using a good model is necessary to obtain a reliable estimate of the SET curve. A problem might be that the choice of the feature subsets may be biased by the model. Indeed, it is possible for optimal feature subsets to differ with respect to the model. However, it seems reasonable to think that the problem will not be too important for sparse feature subsets, which are precisely the feature subsets which are looked for by experts.

In this paper, leave-one-out (LOO) cross-validation [13] is used to estimate the generalisation error. First, a single sample is removed from the dataset and a model is built using the remaining data. Then, the prediction error on the unused sample is computed. The process is repeated for each sample; the average result gives an estimate of the generalisation error.

4.2. *Optimising feature subsets*

In practice, it is impossible to test all possible feature subsets, since the number of tests grows exponentially with the dimensionality of data. Instead, one typically starts with an arbitrary feature subset, which is iteratively improved. Examples of such methods include LARS and forward-backward search. The latter can e.g. use mutual information to guide the search.

LARS [2] is an algorithm which solves efficiently the LASSO problem [14], i.e. an L_1-regularised linear regression. The constraint on the L_1-norm enforces sparsity: the number of selected features increases as the regularisation decreases. LARS can be used for feature selection and the path of its solutions can be converted into a FSP. However, LARS is optimal for linear problems but not necessarily for nonlinear ones.

Mutual information [3,4] is a measure of the statistical dependency between a set of features and a target variable. It can be used to choose a subset of features using the strength of the statistical link between the subset and the output. A simple example of feature selection method based on mutual information consists of (i) ranking features according to their mutual information with respect to the output and (ii) adding features to the feature subset in the order defined by the ranking. In such a case, only d features subsets need to be considered, where d is the dimensionality. Such procedures are simple, but they cannot deal efficiently e.g. with XOR-like problems, where features must be considered together to establish statistical dependencies. An alternative consists of using multivariate greedy methods, like e.g. forward or backward search.

Forward search [1] starts from an empty set of features and iteratively selects a feature to add. Backward search [1] is similar, but its starts with all features and iteratively removes them. At each step, every feature which is not yet selected has to be considered, which means that a total of $\mathcal{O}(d^2)$ feature subsets are considered. Mutual information or validation error can be e.g. used to choose feature subsets and guide the search. Since features are added (or removed) one at a time, successive feature subsets can only differ by one feature, which may not be optimal in practice.

In the above methods, it is impossible to add or remove several features simultaneously. It means that for problems like the XOR-like problem of Section 3, the FSP may not be optimal and may not highlight the fact that some features must be selected together. Indeed, Fig. 1 shows that when the number of selected features changes from one to two, three features must be changed. This cannot be achieved with e.g. forward search. Moreover, for the above methods, a lot of possible feature subsets have to be considered at each iteration.

In the rest of this section, a new algorithm is introduced to overcome the weaknesses of the above methods. Namely, the proposed algorithm allows obtaining FSP with significant differences in successive feature subsets. In Section 5, experiments show that in some situations, the proposed algorithm obtains (i) optimal test errors using less features and (ii) FSPs with more information than the FSPs obtained by LARS and two other greedy search algorithms.

4.3. *Relaxing the feature selection problem*

The generalisation error is seldom used to guide the search for feature subsets. Indeed, this error is usually very costly to estimate, since one needs to rely on e.g. cross-validation. Instead, the heuristic methods described above use other objective functions like e.g. regularised training error or mutual information. Here, a similar approach to LARS is proposed. The feature selection problem is firstly relaxed and a regularisation scheme is used to enforce feature sparsity.

In order to approximate the FSP and the SET curve, let us focus on finding good feature subsets and good models for each feature subset size. Using Eq. (1), the corresponding problem can be stated for regression as

$$\min_{\beta,\theta} \frac{1}{n}\sum_{i=1}^{n}[t_i - \hat{f}(\beta_1 x_i^1,\ldots,\beta_d x_i^d|\theta)]^2 \quad \text{s.t. } \|\beta\|_0 = d_s \le d \tag{3}$$

where β is a vector of binary variables s.t. $\beta_i \in \{0,1\}$, $\|\beta\|_0$ is the L$_0$-norm of β, i.e. the number of non-zero components β_i, and d_s is the size of the feature subset. Here, each binary variable β_i indicates whether the ith feature is selected or not. The constraint limits the number of active features. Notice that the generalisation error is replaced by the training error in (3).

Because of the L$_0$-norm constraint, the above optimisation problem is still combinatorial and difficult to solve. In order to simplify the optimisation problem, let us first rewrite Eq. (3) as a regularisation, i.e.

$$\min_{\beta,\theta} \frac{1}{n}\sum_{i=1}^{n}[t_i - \hat{f}(\beta_1 x_i^1,\ldots,\beta_d x_i^d|\theta)]^2 + C_0\|\beta\|_0 \tag{4}$$

for some regularisation constant $C_0 \in \Re^+$. It is now possible to use a common approach in machine learning, which consists of replacing the L$_0$-norm with an L$_1$-norm [15]. Indeed, it has been shown e.g. for linear models [2] and support vector machines [16,17] that regularising with respect to the L$_1$-norm decreases the number of features actually used by the model. Moreover, the L$_1$-norm is easier to optimise than the L$_0$-norm. The same idea is used in LARS: [2] shows that a linear regression with an L$_1$ regularisation can be used to reduce the number of selected features. Notice that the above approach is similar to a common approach in integer programming which is called relaxation [18]. Eq. (4) becomes

$$\min_{\tilde{\beta},\theta} \frac{1}{n}\sum_{i=1}^{n}[t_i - \hat{f}(\tilde{\beta}_1 x_i^1,\ldots,\tilde{\beta}_d x_i^d|\theta)]^2 + C_1\|\tilde{\beta}\|_1 \tag{5}$$

for some regularisation constant $C_1 \in \Re^+$. Vector $\tilde{\beta}$ no longer defines a feature subset. Instead, Eq. (5) is related to *feature scaling*, a problem similar to feature selection where ones tries to find coefficients giving a different importance to each feature.

Eq. (5) is easier to solve than Eq. (4) since it is differentiable. Yet, solutions of Eq. (5) can be converted into approximated solutions of Eq. (4). Indeed, a non-zero $\tilde{\beta}_i$ variable can be considered to mean that the corresponding feature is selected, i.e. $\beta_i = 1$. Indeed, even for small values of $\tilde{\beta}_i$, feature i is still used by the model. The next subsection proposes an algorithm to build the FSP and the SET curve using Eq. (5).

Notice that the C_1 constant is controlling the regularisation on $\tilde{\beta}$. Indeed, the resulting feature scaling becomes sparser and sparser as C_1 increases. In general, an L$_1$-norm regularisation on a vector of coefficients causes the coefficients to become zero one after another, until none of them remains [14,2,19,1]. Indeed, using the L$_1$-norm regularisation is equivalent to setting a Laplacian prior on $\tilde{\beta}$ [19]. Using the L$_2$-norm, sparsity would be lost [19,1], which explains why the L$_1$-norm is used here. The L$_1$-norm regularisation behaviour is illustrated by Efron et al. in the case of LARS [2].

4.4. *Solving the relaxed feature selection problem*

For various values of C_1, the solutions of Eq. (5) have different degrees of sparsity. The algorithm which is proposed here uses this fact to span the different sizes of feature subsets. Indeed, if C_1 is progressively increased, the sparsity of resulting feature subsets will increase as well. In a nutshell, the proposed algorithm therefore simply solves Eq. (5) for increasing C_1 values.

Solving Eq. (5) is not trivial. Indeed, the objective function may be non-convex and many local minima may exist. A possible approach is gradient descent with multiple restarts. However, gradient descent on continuous variables can be very slow, e.g. if the minimised function has many plateaux. Moreover, it is difficult to reach exact values like e.g. $\tilde{\beta}_i = 0$ or $\tilde{\beta}_i = 1$.

In this paper, feature scalings are discretised to overcome the above problems. Indeed, exact solutions are not necessary, since they are converted into binary feature subsets afterwards. The space of all possible feature scaling $[0,1]^d$ becomes a hypergrid $\{0,1/k,\ldots,1\}^d$ with $k+1$ non-zero values in each dimension. Next, the gradient of the regularised training error is used to guide the search. At each step, the search only considers the direct neighbour pointed to by the gradient. Here, a direct neighbour of the feature scaling $\tilde{\beta}$ is a feature scaling $\tilde{\beta}'$ s.t. $\max_i|\tilde{\beta}_i' - \tilde{\beta}_i| \le 1/k$. According to that definition, several feature scalings can change at each step. In this paper, k is equal to 10 for the experiments.

The proposed procedure is detailed in Algorithm 1. A fast implementation based on extreme learning machines is proposed in Section 4.5. For each repetition of the main loop, the feature scaling $\tilde{\beta}$ is randomly initialised and C_1 is set to zero, i.e. no regularisation is initially performed. The current solution and the current model are used to update the FSP and the SET curve for $\|\tilde{\beta}\|_0$ features, if necessary. Given the current solution $\tilde{\beta}$ and the current value of C_1, the gradient of the regularised training error is used to find a candidate $\tilde{\beta}_{new}$ in the direct neighbourhood of $\tilde{\beta}$. If $\tilde{\beta}_{new}$ is actually better than $\tilde{\beta}$ in terms of regularised training error, then $\tilde{\beta}_{new}$ becomes the new, current solution. Otherwise, a local minimum has been found; C_1 is increased and the algorithm searches for a sparser solution with a smaller regularised training error (with respect to the new C_1 constant). The algorithm stops when C_1 is so large that the L$_0$-norm $\|\tilde{\beta}\|_0$ becomes zero, i.e. when the feature subset becomes empty.

Algorithm 1. Local search algorithm for the relaxed feature selection problem

```
for all restarts do
  $C_1 \leftarrow 0$
  initialise $\tilde{\beta}$ randomly
  find the vector of parameters $\theta$ corresponding to $\tilde{\beta}$ (train a model)
  compute the regularised training error

  while $\|\tilde{\beta}\|_0 > 0$ do
    estimate the generalisation error obtained using $\tilde{\beta}$ and $\theta$
    convert the feature scaling $\tilde{\beta}$ into a feature subset $\beta$
    update the FSP and the SET curve, if necessary

    compute the gradient of the regularised training error
    find the direct neighbour $\tilde{\beta}_{new}$ pointed by the gradient
    find the vector of parameters $\theta_{new}$ corresponding to $\tilde{\beta}_{new}$ (train a model)

    compute the new regularised training error
    if the regularised error has not decreased then
      increase $C_1$ until the gradient points to $\tilde{\beta}_{new}$ s.t. $\|\tilde{\beta}_{new}\|_1 < \|\tilde{\beta}\|_1$
```

```
            increase $C_1$ until the regularised training error for $\tilde{\beta}_{new}$
      becomes lower

            find the vector of parameters $\theta_{new}$ corresponding to $\tilde{\beta}_{new}$
            compute the new regularised training error
         end if

         update the current solution $\tilde{\beta}$ with $\tilde{\beta}_{new}$
         update the vector of parameters $\theta$ with $\theta_{new}$
         update the regularised training error
      end while
   end for
```

In Algorithm 1, θ is the vector of model parameters introduced in Eq. (1). The procedure to obtain θ depends on the type of model which is used. For example, in the case of linear regression, instances can be first multiplied by scaling coefficients $\tilde{\beta}$. Then, the weights θ of the linear regression are obtained as usual using the scaled instances and the target values. The case of non-linear models is illustrated in Section 4.5, which proposes a fast implementation of Algorithm 1 based on extreme learning machines.

Since (i) each local minimum is reached in a finite number of steps and (ii) C_1 is increased whenever a local minimum of the regularised training error is reached, Algorithm 1 is guaranteed to terminate in a finite amount of steps. Eventually, the feature subset becomes empty and the algorithm terminates. Feature scalings are converted into feature subsets by simply assuming that features with non-zero scalings $\tilde{\beta}_i$ are selected. Indeed, simply rounding the scalings toward 0 or 1 could not be sufficient, as even features which correspond to small scalings may nevertheless be used by the model.

Algorithm 1 is not guaranteed to find the optimal solution for each feature subset size. However, by slowly increasing the regularisation on $\|\beta\|_1$, the proposed algorithm spans the whole spectrum of feature subsets sizes. Multiple restarts are performed to decrease the influence of local minima.

Compared with e.g. forward search and backward elimination, Algorithm 1 has several advantages. Firstly, the gradient information is used to consider only one neighbour at each iteration. Secondly, multiple features can be updated simultaneously. Moreover, Algorithm 1 can select unselected features or remove selected features, which is impossible in simple forward or backward search.

4.5. Fast implementation of the proposed algorithm

Algorithm 1 requires (i) models which are fast to train and (ii) a fast estimator of the generalisation error. Extreme learning machines (ELMs) meet both these requirements [5–8]. Firstly, their training is very fast, since it only requires solving a linear system. Secondly, the LOO error of an ELM can be computed quickly and exactly using the PRESS statistics [13,10]. The LOO error is a special case of the cross-validation error, an estimator of the generalisation error. This subsection firstly reviews ELMs, then shows how to use ELMs in order to implement Algorithm 1.

ELMs are feed-forward neural networks with one hidden layer (see Fig. 3). In traditional feed-forward neural networks, the weights of both hidden and output weights are simultaneously optimised through gradient descent. This learning procedure is called back-propagation in the case of the popular multi-layer perceptron [20]. However, gradient descent has many drawbacks. In particular, it is slow and can get stuck in one of the many local minima of the objective function [5].

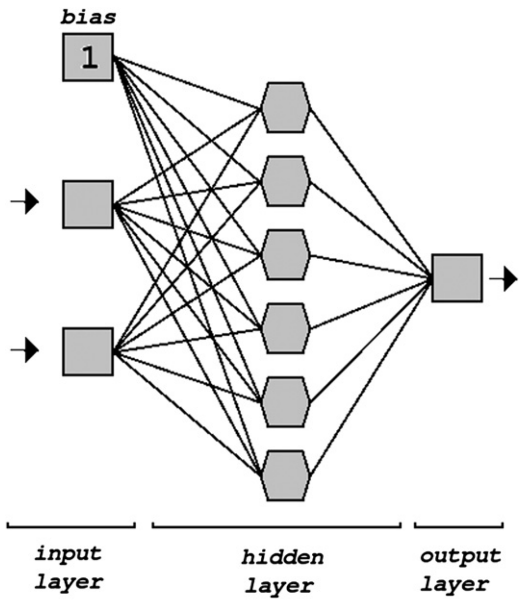


Fig. 3. Feed-forward neural network with one hidden layer.

Extreme learning machines [5–7] provide an interesting alternative to train feed-forward neural networks, which solves the above problems. Firstly, the weights and biases in the hidden layer are set randomly and remain fixed during the training process. Then, the hidden layer output matrix of the ELM with m hidden neurons is computed as

$$H = \begin{bmatrix} \sigma\left(\sum_{i=1}^{d} W_{i1}X_{1i}+b_1\right) & \cdots & \sigma\left(\sum_{i=1}^{d} W_{im}X_{1i}+b_m\right) \\ \vdots & \ddots & \vdots \\ \sigma\left(\sum_{i=1}^{d} W_{i1}X_{ni}+b_1\right) & \cdots & \sigma\left(\sum_{i=1}^{d} W_{im}X_{ni}+b_m\right) \end{bmatrix} \tag{6}$$

where σ is the activation function of the hidden units, W is the $d \times m$ matrix of random hidden layer weights, X is a $n \times d$ matrix where each row corresponds to a training instance and b is the m-dimensional vector of random hidden layer biases. Usually, σ is the hyperbolic tangent tanh, but any infinitely differentiable function can be used [5]. For example, radial basis functions are also considered in [21].

Since the output of an ELM is a linear combination of the m hidden layer neuron outputs, the output weights are found by solving the linear problem

$$\min_{w} \|T - Hw\|_2^2 \tag{7}$$

where T is an n-dimensional vector containing the target values and w is the m-dimensional vector of output weights. It is well known that the unique solution of Eq. (7) is

$$w = H^{\dagger}T \tag{8}$$

where $H^{\dagger}$ is the Moore–Penrose pseudo-inverse [22] of H. Using e.g. singular value decomposition, $H^{\dagger}$ can be computed efficiently.

In the seminal paper [5], it is shown that ELMs achieve good performances in terms of error, with respect to other state-of-the-art algorithms. Moreover, ELMs are shown to be much faster than traditional machine learning models. For example, they can be trained up to thousands times faster than support vector machines. Notice that there exist a significant number of variants of ELMs. In particular, other activation functions can be used [21] and ELMs can be trained incrementally [9]. The universal approximation capability of ELMs is discussed in [23].

Another advantage of ELMs is that it is possible to obtain an analytical expression for an estimate of their generalisation error

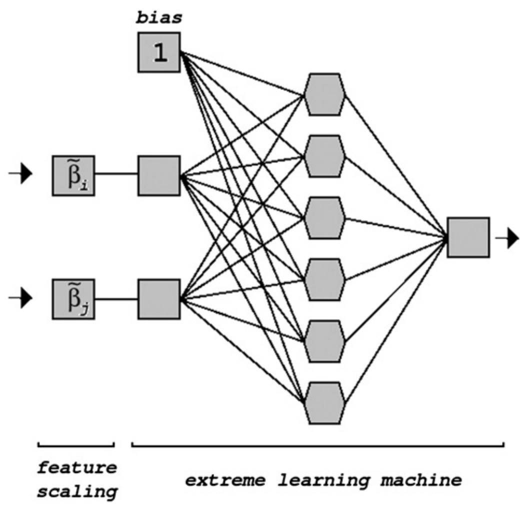


Fig. 4. Extreme learning machine with integrated feature scaling.

[10]. Indeed, the LOO error for an ELM can be obtained using the PRESS statistics [13], i.e.

$$\text{PRESS} = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{e_i}{1-z_{ii}}\right]^2 \tag{9}$$

where e_i is the error for the ith training instance and z_{ii} is the ith diagonal term of

$$Z = HH^{\dagger}. \tag{10}$$

Since ELMs are fast to train and a fast estimator of their generalisation error exists, they are perfectly fitted to implement Algorithm 1. Intuitively, as shown in Fig. 4, the feature scaling can be seen as an extra layer put in front of the ELM. In the following, the feature scaling is directly plugged into ELMs to make the development easier. The hidden layer output matrix of the new ELM becomes

$$\tilde{H} = \begin{bmatrix} \sigma\left(\sum_{i=1}^{d} W_{i1}\tilde{\beta}_i X_{1i}+b_1\right) & \cdots & \sigma\left(\sum_{i=1}^{d} W_{im}\tilde{\beta}_i X_{1i}+b_m\right) \\ \vdots & \ddots & \vdots \\ \sigma\left(\sum_{i=1}^{d} W_{i1}\tilde{\beta}_i X_{ni}+b_1\right) & \cdots & \sigma\left(\sum_{i=1}^{d} W_{im}\tilde{\beta}_i X_{ni}+b_m\right) \end{bmatrix} \tag{11}$$

and the optimal output weights of the new ELM are now given by

$$\tilde{w} = \tilde{H}^{\dagger}T \tag{12}$$

Using the above definitions, the gradient of the regularised training error with respect to the scaling vector $\tilde{\beta}$ becomes

$$\nabla_{\tilde{\beta}}\text{MSE} = \begin{bmatrix} -\frac{2}{n}\sum_{i=1}^{n} e_i \sum_{j=1}^{m} w_j W_{1j} X_{i1} \tilde{H}'_{ij} \\ \vdots \\ -\frac{2}{n}\sum_{i=1}^{n} e_i \sum_{j=1}^{m} w_j W_{dj} X_{id} \tilde{H}'_{ij} \end{bmatrix} \tag{13}$$

where e_i is the error for the ith training instance and $\tilde{H}'$ is defined as

$$\tilde{H}' = \begin{bmatrix} \sigma'\left(\sum_{i=1}^{d} W_{i1}\tilde{\beta}_i X_{1i}+b_1\right) & \cdots & \sigma'\left(\sum_{i=1}^{d} W_{im}\tilde{\beta}_i X_{1i}+b_m\right) \\ \vdots & \ddots & \vdots \\ \sigma'\left(\sum_{i=1}^{d} W_{i1}\tilde{\beta}_i X_{ni}+b_1\right) & \cdots & \sigma'\left(\sum_{i=1}^{d} W_{im}\tilde{\beta}_i X_{ni}+b_m\right) \end{bmatrix}$$

$$= \begin{bmatrix} 1-\tilde{H}_{11}^2 & \cdots & 1-\tilde{H}_{1m}^2 \\ \vdots & \ddots & \vdots \\ 1-\tilde{H}_{n1}^2 & \cdots & 1-\tilde{H}_{nm}^2 \end{bmatrix} \tag{14}$$

since σ is here the hyperbolic tangent tanh whose derivative is $\tanh'(z) = 1-\tanh(z)^2$.

Algorithm 1 can be implemented using (i) Eq. (12) to train an ELM, (ii) Eq. (9) to estimate its generalisation error and (iii) Eq. (13) to compute the gradient guiding the search. Notice that the vector of model parameters θ which appears in both Eq. (1) and Algorithm 1 corresponds here to the vector of output weights $\tilde{w}$. In theory, one should optimise the ELM size m before starting the scaling search. However, there is no guarantee that the optimal ELM size is identical for different numbers of selected features. Therefore, the solution chosen here is simply to choose a random ELM size at each restart. Indeed, only ELMs with correct sizes (with respect to the feature subset size) will eventually be taken into account, since they are precisely the ELMs which will be used to build the FSP and the SET curve.

In the rest of this paper, the proposed implementation of Algorithm 1 is called ELM-FS, for ELM-based feature selection.

4.6. *Remarks on the estimated SET curve*

In the proposed approach, the SET curve is estimated by selecting the best feature subsets among those which are considered during the search. The resulting SET curve can be used to select a feature subset, e.g. the one with the lowest generalisation error. However, two remarks hold here. Firstly, the estimate of the generalisation error provided by cross-validation (and in particular LOO) tends to be less reliable when more and more features are added, because of the curse of dimensionality. This could lead experts to choose too large feature subsets. Secondly, the estimated generalisation error is not valid any more as soon as a particular feature selection is chosen. Indeed, since the estimate was used to select a particular feature subset, it is biased for this particular solution. An additional, independent set of instances should be used to estimate the final generalisation error. Yet, the estimated SET curve can be used to select a subset size.

5. Experiments

In this section, two goals are pursued through experiments. Firstly, it is necessary to assess whether the proposed algorithm obtains feature subsets which are either equivalent or better than those obtained using standard feature selection methods. Secondly, since the proposed algorithm naturally provides a FSP, it is important to assess whether the FSP provides useful information or not, with respect to methods which only provide a best feature subset.

The following of this section is organised as follows. Section 5.1 describes the experimental settings. Sections 5.2 and 5.3 show the results for artificial and real datasets, respectively.

5.1. *Experimental settings*

ELM-FS is compared with three other methods, in terms of feature subsets and test error: LARS, forward search with mutual information (MI-FW) and forward-backward search with mutual information (MI-FWBW). LARS searches for linear relationships in data [2], whereas MI-FW and MI-FWBW search for more general, possibly nonlinear relationships. Whereas the features and the output are compared in terms of correlation for LARS, mutual information estimates their statistical dependency. Each feature is normalised using the mean and the standard deviation computed on training samples.

MI-FW starts with an empty subset of features. At each iteration, MI-FW computes the mutual information between the current subset of features and the output. Then, the feature which increases the most this mutual information is added to the current subset of features. The algorithm continues until all features have been added. MI-FW is not repeated, since it always starts with the same, empty subset of features. The implementation of MI-FWBW is similar, except that features can be either added or removed at each step. Moreover, MI-FWBW is repeated 100 times with random initial feature subsets in order to (i) reduce the effect of local minima and (ii) obtain a complete FSP. Mutual information is estimated using a k-nearest neighbours approach introduced by Kraskov et al. [3], where k is chosen using cross-validation [24].

ELM-FS is performed using 100 repetitions. The neurons of the 100 corresponding ELMs are chosen in a fixed set of 100 neurons. For each repetition, (i) a random number of neurons is chosen between 1 and 100 and (ii) the corresponding number of neurons are chosen in the fixed set of neurons.

The test errors are computed as follows. For each dataset, an ELM is initialised with 100 neurons. Then, for each feature selection algorithm and each feature subset size, the output weights are optimised using the feature subset of the corresponding size, the training samples and OP-ELM, a state-of-the-art method in extreme learning [10]. Eventually, the predictions of

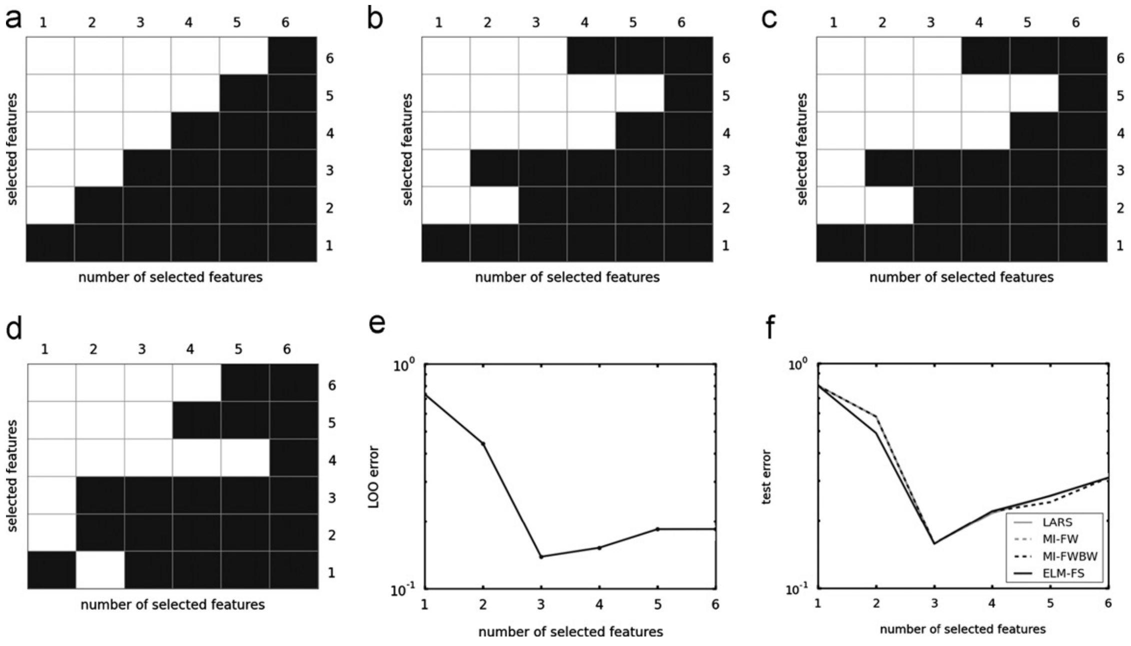


Fig. 5. Results for the XOR-like dataset: (a–d) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (e) the SET curve for ELM-FS and (f) the test errors for the four compared methods. Notice the logarithmic scales for errors. (a) FSP: LARS. (b) FSP: MI-FW. (c) FSP: MI-FWBW. (d) FSP: ELM-FS. (e) SET curve: ELM-FS. (f) test errors.

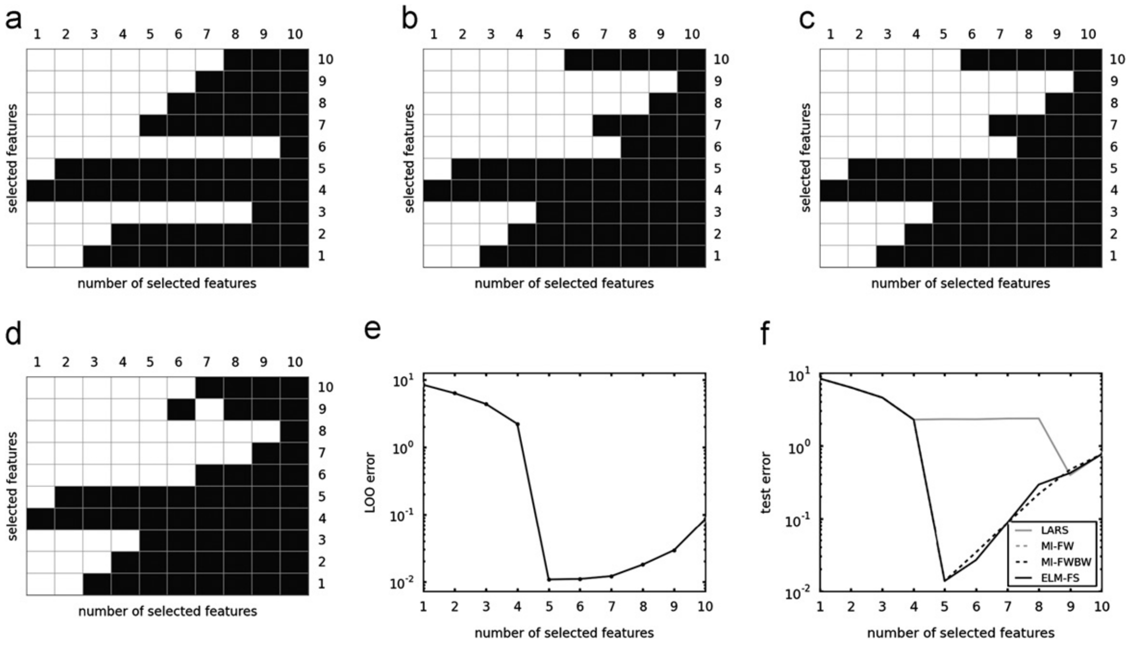


Fig. 6. Results for the functional dataset: (a-d) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (e) the SET curve for ELM-FS and (f) the test errors for the four compared methods. Notice the logarithmic scales for errors. (a) FSP: LARS. (b) FSP: MI-FW. (c) FSP: MI-FWBW. (d) FSP: ELM-FS. (e) SET curve: ELM-FS. (f) test errors.

the resulting ELM are compared on the test samples in order to produce the test error. In order to be able to compare the different feature selection algorithms, the test errors for a given dataset are obtained using the same initial ELM. Therefore, identical feature subsets correspond to identical test errors.

5.2. *Results on artificial datasets*

In this subsection, two artificial toy problems are used to compare ELM-FS with LARS, MI-FW and MI-FWBW: (i) the XOR-like problem introduced in Section 3 and (ii) a complex, nonlinear functional [24]. For convenience, the definition of the XOR-like problem is repeated below.

For the XOR-like problem, the artificial dataset is built using six random features which are uniformly-distributed in [0,1]. For each sample $x_i = (x_i^1, \dots, x_i^6)$, the target is

$$f(x_i) = x_i^1 + (x_i^2 > 0.5)(x_i^3 > 0.5) + \epsilon_i \quad (15)$$

where (i) $(x > 0.5)$ is equal to 1 when $x > 0.5$ and is equal to 0 otherwise and (ii) ϵ_i is a noise with distribution $\mathcal{N}(0,0.1)$. This regression problem is similar to the XOR problem in classification: the product term can only be computed using both features 2 and 3.

For the functional problem, the artificial dataset is built using ten random features which are uniformly-distributed in [0,1]. For each sample $x_i = (x_i^1, \dots, x_i^{10})$, the target is

$$f(x_i) = 10\,\sin(x_i^1)x_i^2 + 20(x_i^3 - 0.5)^2 + 10x_i^4 + 5x_i^5 + \epsilon_i \quad (16)$$

where ϵ_i is a noise with distribution $\mathcal{N}(0,0.1)$.

Table 1
Computation times in seconds of the different feature selection algorithms for the XOR-like problem and the functional problem, including the search of the k parameter for the Kraskov estimator.

	LARS	MI-FW	MI-FWBW	ELM-FS
XOR-like	1.2e−2	1.0e+2	2.6e+2	3.1e+2
functional	1.3e−2	1.7e+2	5.1e+2	4.2e+2

For both artificial problems, 1000 training samples were generated in order to have a sufficient amount of data for the feature selection. Each test set consists of 9000 samples, so that the test error accurately estimates the generalisation error.

For the XOR-like dataset, Fig. 5 shows (i) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (ii) the SET curve for ELM-FS and (iii) the test errors for the four methods. The SET curve recommends to use three features. In this case, the four methods choose the correct feature subset, i.e. $\{1,2,3\}$. However, the FSP obtained using ELM-FS provides additional information: features 2 and 3 should be selected together. Indeed, when ELM-FS selects only one feature, feature 1 is selected. But when ELM-FS selects two features, feature 1 is no longer used. Instead, features 2 and 3 are selected jointly. This information cannot be seen on the FSPs of LARS, MI-FW and MI-FWBW: they successively select feature 1 and either feature 2 or feature 3. In conclusion, the FSP obtained using ELM-FS reflects well Eq. (15), where the target depends on a nonlinear combination of features 2 and 3. Notice that when only two features are selected, ELM-FS obtains a slightly smaller test error, which supports its choice of features.

For the functional dataset, Fig. 6 shows (i) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (ii) the SET curve for ELM-FS and (iii) the test errors for the four methods. ELM-FS recommends to use the five features which are actually the ones used to compute the target. Identical feature subsets and test errors are obtained using the other algorithms, except LARS which includes feature 3 only for large feature subset sizes and achieves larger test errors.

According to results for the XOR-like and functional datasets, ELM-FS is able to cope with nonlinearities and obtains sound feature subsets. Moreover, for both datasets, the feature subset which corresponds to the minimum of the SET curve also obtains the minimum test error. In other words, the SET curve estimated by ELM-FS using the PRESS statistics is a valuable tool for choosing the size of the optimal feature subset.

An important difference between ELM-FS and the other methods, i.e. LARS, MI-FW and MI-FWBW, is that the obtained FSP highlight features which must be selected together. Indeed, ELM-FS is able to drop a feature when the feature subset size increases,

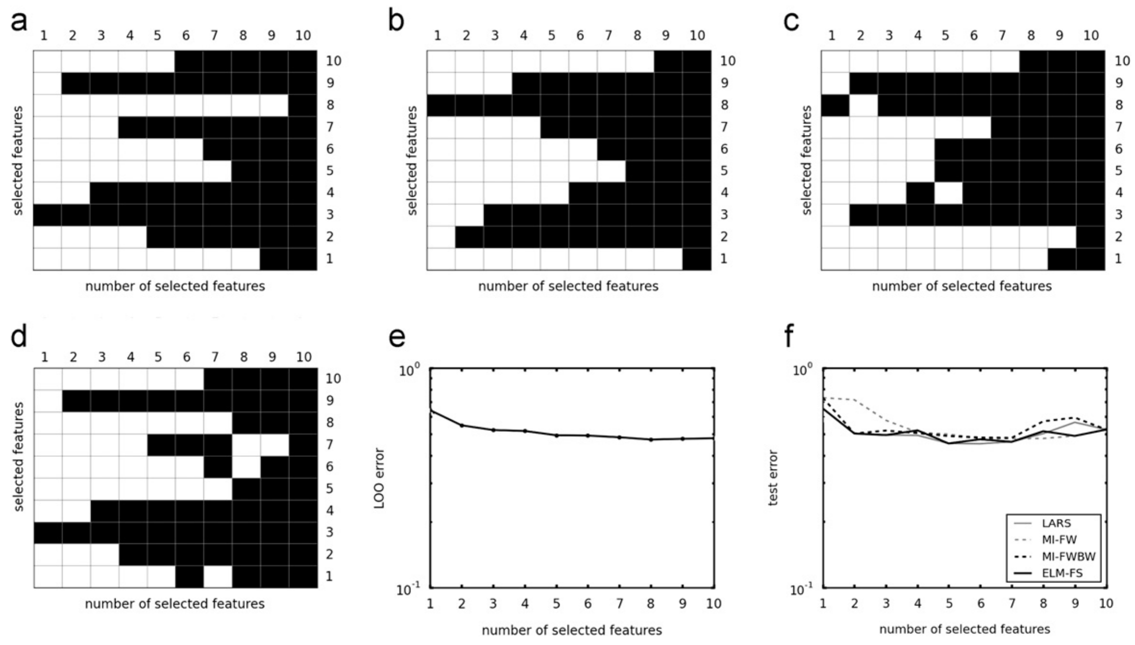


Fig. 7. Results for the diabetes dataset: (a-d) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (e) the SET curve for ELM-FS and (f) the test errors for the four compared methods. Notice the logarithmic scales for errors. (a) FSP: LARS. (b) FSP: MI-FW. (c) FSP: MI-FWBW. (d) FSP: ELM-FS. (e) SET curve: ELM-FS. (f) test errors.

in order to add two new features which must be used jointly. This provides an insightful information about the target function.

Table 1 shows the computation times for the different feature selection algorithms, including the computation time for the selection of the k parameter used by the Kraskov estimator of the mutual information. In terms of computation time, ELM-FS is comparable to MI-FWBW, whereas LARS and MI-FW are faster. However, it should be highlighted that (i) LARS only searches for linear relationships and (ii) MI-FW searches through a much smaller space of possible feature subsets.

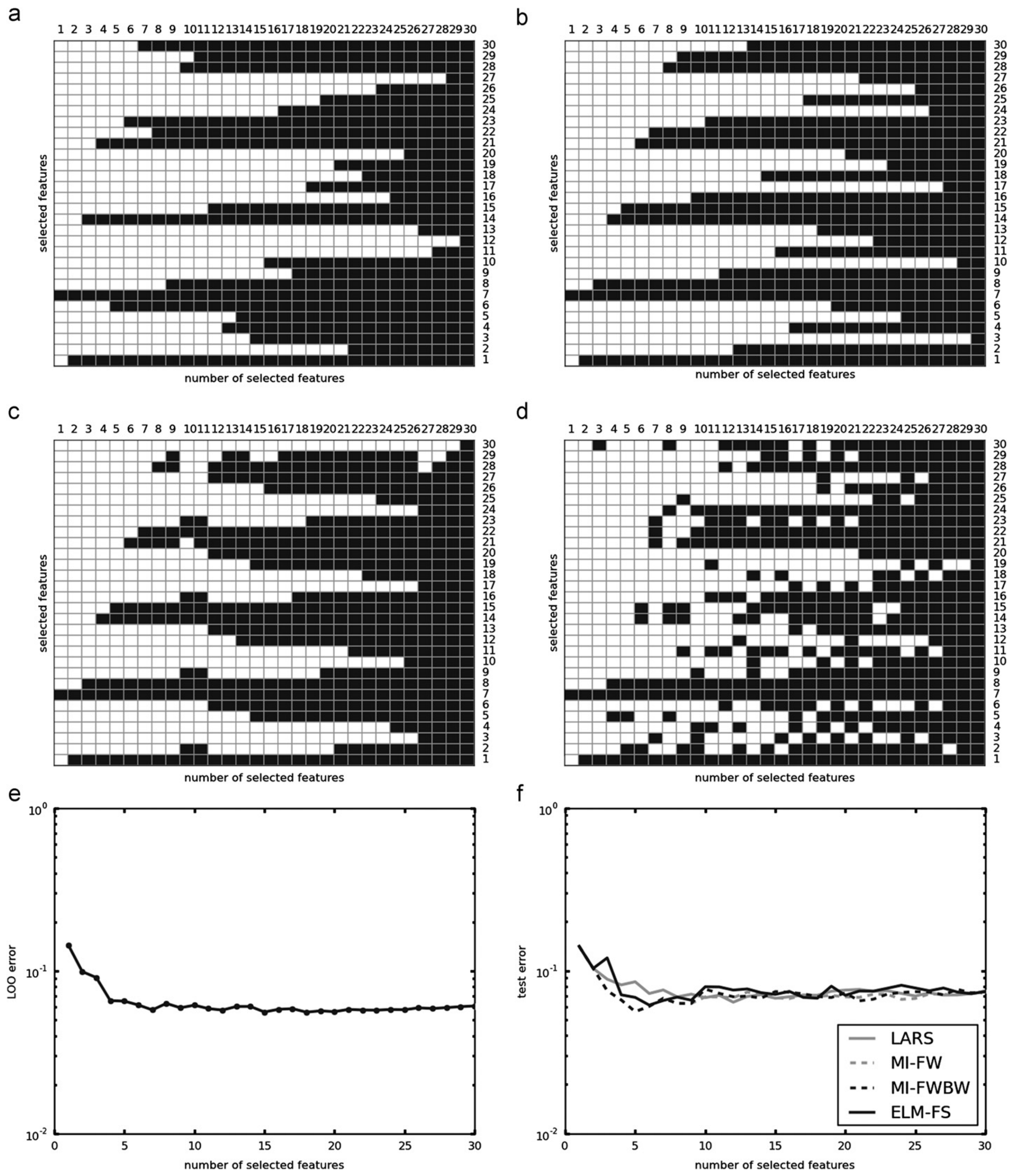


Fig. 8. Results for the Poland electricity load dataset: (a-d) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (e) the SET curve for ELM-FS and (f) the test errors for the four compared methods. Notice the logarithmic scales for errors.

5.3. Results on real datasets

In this subsection, four real datasets [25] are used to compare ELM-FS with LARS, MI-FW and MI-FWBW: (i) the diabetes dataset from Efron et al. [2], (ii) the Poland electricity load dataset [26], (iii) the Santa Fe laser dataset [27] and (iv) the anthrokids dataset [28]. The diabetes dataset consists of 442 samples with 10 continuous features. For comparison, a FSP is given for LARS in [2]. The Poland electricity load dataset consist of 1370 samples with 30 continuous features. The original time series is transformed into a regression problem, where the 30 past values are used to predict the electricity load of the next day. For example, the first feature corresponds to the last day. The Santa Fe laser dataset consists of 10,081 samples with 12 continuous features. The anthrokids dataset consists of 1019 samples with 53 features. For the experiments, the diabetes dataset, the Poland electricity load dataset and the anthrokids dataset are split into two parts: 70% of the instances are used for training and the remaining 30% of the instances are used for test. The Santa Fe laser dataset is split into a training set of 1000 instances and a test set of 9081 instances.

For the diabetes dataset, Fig. 7 shows (i) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (ii) the SET curve for ELM-FS and (iii) the test errors for the four methods. For feature subsets of at most three features, LARS and ELM-FS obtain lower test errors than MI-FW and MI-FWBW. The FSP of LARS and ELM-FS are identical for the three first subset sizes: feature 3 (body mass index), feature 9 (one of the serum measurements) and feature 4 (blood pressure). For larger feature subset sizes, the four algorithms achieve similar test errors. Here, ELM-FS has no advantage over other methods, but it achieves performances which are similar in terms of test error to those obtained by LARS, which it the best other method for this dataset. The SET curve provided by ELM-FS shows that using two or three features, almost optimal results can be achieved, which is confirmed by the test errors.

For the Poland electricity load dataset, Fig. 8 shows (i) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (ii) the SET curve for ELM-FS and (iii) the test errors for the four methods. According to the SET curve for ELM-FS, seven features are sufficient to achieve almost optimal generalisation error. For this subset size, LARS, MI-FW, MI-FWBW and ELM-FS choose the feature subsets {1,6,7,14,21, 23, 30} {1,7,8,14,15,21,22}, {1,7,8,14,15,21,22} and $\{1,3,7,8,21,22,23\}$, respectively. In other words, the four methods recommend to use the electricity load of yesterday (feature 1) and the electricity load of previous weeks on the same day (e.g. features 7, 14 or 21). Moreover, they recommend to use the electricity load around these days (e.g. features 6, 8, 15 or 22), which could e.g. be used to estimate the time series derivative. A few other features are used (e.g. features 3, 23 and 30), which may be explained by the important amount of redundancy in this regression problem. Test errors are similar for the four methods.

For the Santa Fe laser dataset, Fig. 9 shows (i) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (ii) the SET curve for ELM-FS and (iii) the test errors for the four methods. For ELM-FS, the SET curve shows that 4 features are sufficient to achieve almost optimal results. The FSP for ELM-FS shows that the corresponding subset is $\{1,2,4,7\}$. But the FSP also shows that features 3 and 8 seem to be important. Here, the FSP provide additional information: the analysis of the successive feature subsets for smaller subset sizes reveals other interesting features. This cannot be seen if only the selected feature subset is considered. LARS, MI-FW and MI-FWBW do not select features 1, 2, 4, and 7 together for small feature subsets. It explains that ELM-FS beats them in terms of test error for these subsets sizes. Here, LARS needs eight features to achieves a similar test error, whereas the methods based on mutual information are not able to compare to ELM-FS. Notice that the FSP obtained using ELM-FS has many discontinuities, which suggests redundancy or complex interactions between the features and the target function.

For the anthrokids dataset, Figs. 10, 11 and 12 show (i) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (ii) the SET curve for ELM-FS and (iii) the test errors for the four methods. For ELM-FS, the SET curve shows that nine features are sufficient to achieve almost optimal results. The test error achieves its minimum around this point for all methods. No method seems to be significantly better than the others. Yet, the FSP for ELM-FS is different from the three other FSPs: whereas LARS, MI-FW and MI-FWBW choose successive feature subsets which are very similar by design, ELM-FS does not

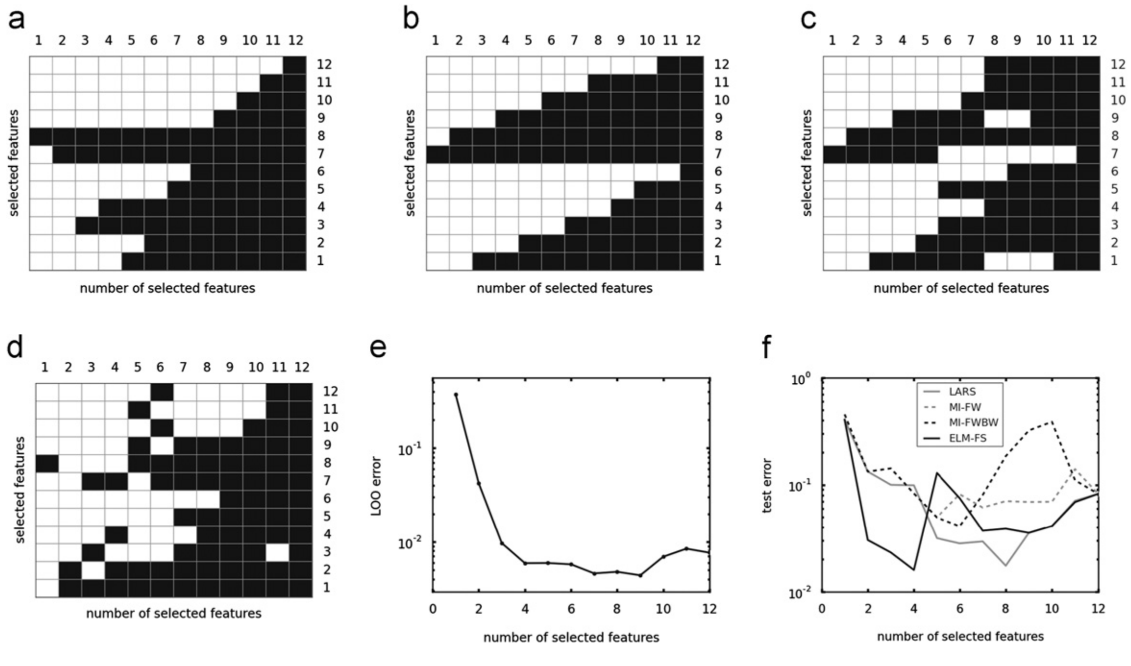
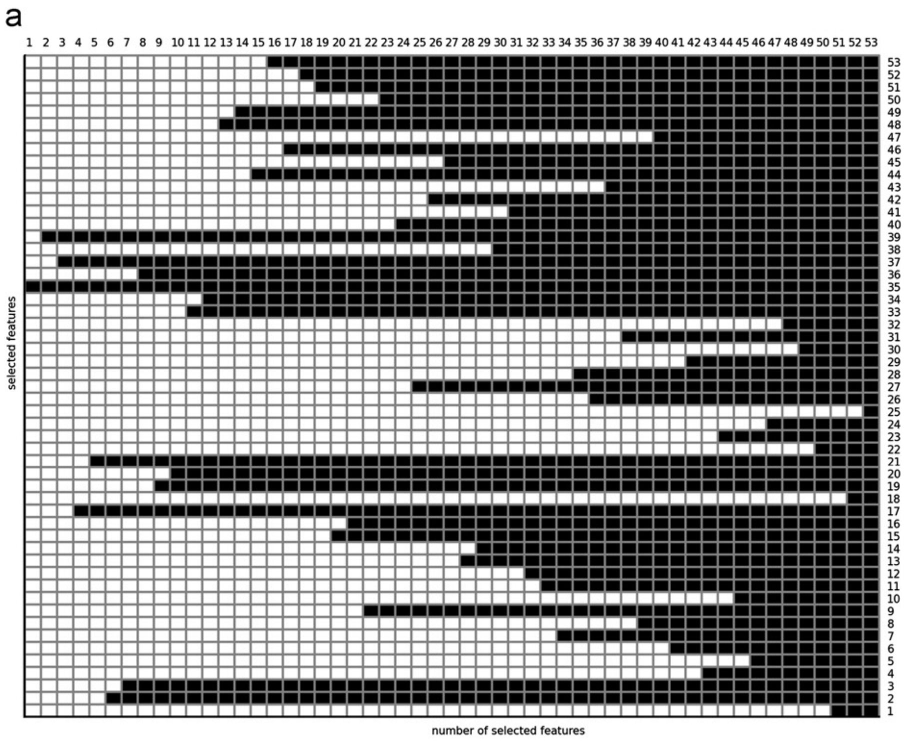


Fig. 9. Results for the Santa Fe laser dataset: (a-d) the FSPs for LARS, MI-FW, MI-FWBW and ELM-FS, (e) the SET curve for ELM-FS and (f) the test errors for the four compared methods. Notice the logarithmic scales for errors.



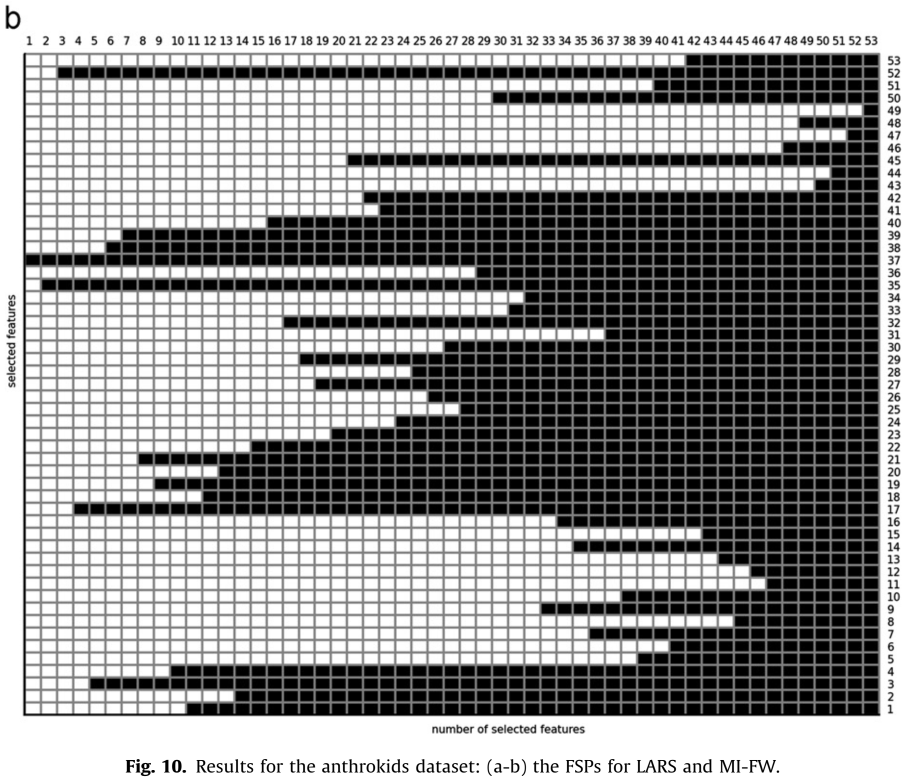


Fig. 10. Results for the anthrokids dataset: (a-b) the FSPs for LARS and MI-FW.

suffer from this constraint. The discontinuities in the FSP for ELM-FS indicate that there is an important amount of redundancy between features in this regression problem, what could not be seen with LARS, MI-FW and MI-FWBW. A closer analysis shows that three clusters of features are selected often in the nine first columns of the FSP for ELM-FS: features 1–3, 19–21 and 35–39. These three clusters are also found by the other feature selection methods. Notice that ELM-FS also selects e.g. features 8, 12 and 49 which are not selected by other methods.

Similarly to the case of artificial datasets, the results obtained in this subsection show that ELM-FS obtains sound feature subsets. For the diabetes dataset, the Poland electricity load dataset and the anthrokids dataset, ELM-FS is equivalent to the best methods in terms of test error. For all four datasets, the SET curve obtained by ELM-FS can be used to select the best feature subset size. For the Poland electricity load dataset and the Santa Fe laser dataset, the feature subset which corresponds to the minimum of the SET curve also obtains the minimum test error.

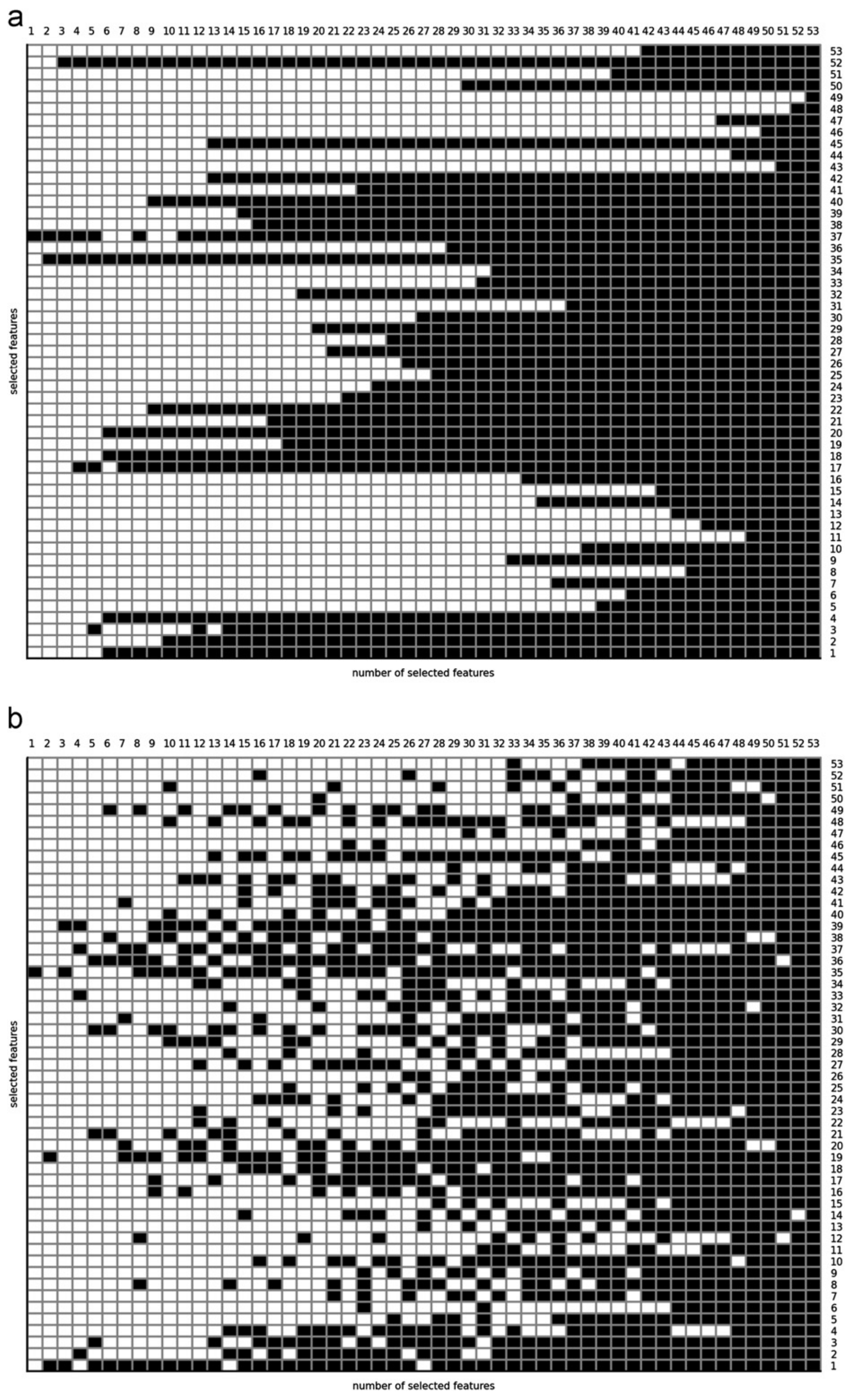


Fig. 11. Results for the anthrokids dataset: (a-b) the FSPs for MI-FWBW and ELM-FS.

For the diabetes dataset and the anthrokids dataset, the feature subset which corresponds to a sufficient LOO error in the SET curve almost obtains the minimum test error, with 3 and 9 features respectively.

The results for the Santa Fe laser dataset show that ELM-FS can be useful for problems with complex relationships between the features and the output. Firstly, the optimal test error is achieved with only four features, whereas LARS needs eight features to achieve a similar result. For the Santa Fe laser dataset, the small feature subsets obtained by ELM-FS allows reaching test errors which are significantly better (for the same subset sizes) than the test errors achieved by other methods. Secondly, the FSP obtained by ELM-FS reflects the complex relationships between the features and the target: there are many discontinuities in the FSP, which is also the case for the anthrokids dataset.

Table 2 shows the computation times for the different feature selection algorithms, including the computation time for the selection of the k parameter used by the Kraskov estimator of the mutual information. In terms of computation time, ELM-FS is comparable to MI-FWBW, whereas LARS and MI-FW are faster.

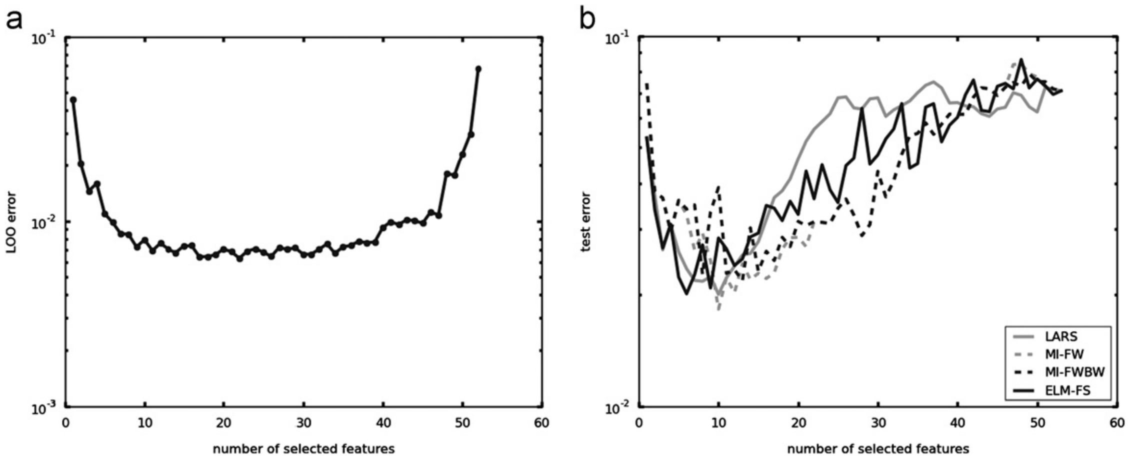


Fig. 12. Results for the anthrokids dataset: (a) the SET curve for ELM-FS and (b) the test errors for the four compared methods. Notice the logarithmic scales for errors.

Table 2
Computation times in seconds of the different feature selection algorithms for the diabetes dataset, the Poland electricity load dataset, the Santa Fe Laser dataset and the anthrokids dataset, including the search of the k parameter for the Kraskov estimator.

	LARS	MI-FW	MI-FWBW	ELM-FS
Diabetes	1.7e−3	1.2e+1	5.5e+1	6.0e+1
Poland electricity load	9.7e−3	4.9e+2	2.1e+3	7.1e+2
Santa Fe	2.9e−2	2.5e+2	7.0e+2	5.0e+2
Anthrokids	2.4e−2	4.1e+2	3.3e+3	4.5e+2

Again, it should be highlighted that (i) LARS only searches for linear relationships and (ii) MI-FW searches through a much smaller space of possible feature subsets.

6. Conclusion

This paper reviews two visual tools to help users and experts to perform feature selection and gain knowledge about the domain: the feature selection and the sparsity-error trade-off curve. The ELM-FS algorithm is proposed to build these two tools. A specific implementation using ELMs is used to analyse different datasets. The experimental results show that the proposed tools and the proposed algorithm can actually help users and experts. Indeed, they provide not only the optimal number of features but also the evolution of the estimation of the generalisation error, and which features are selected for different number of selected features. The proposed methodology allows making a trade-off between feature selection sparsity and generalisation error. This way, experts can e.g. reduce the number of features in order to design a model of the underlying process.

Acknowledgments

The authors would like to thank Gauthier Doquire (Université catholique de Louvain) who suggested the XOR problem for regression used in this paper.

References

[1] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction Springer Series in Statistics, 2nd ed., Springer, 2009.
[2] B. Efron, T. Hastie, L. Johnstone, R. Tibshirani, Least angle regression, Ann. Stat. 32 (2004) 407–499.
[3] A. Kraskov, H. Stögbauer, P. Grassberger, Estimating mutual information, Phys. Rev. E 69 (6) (2004) 066138, http://dx.doi.org/10.1103/PhysRevE.69.066138.
[4] F. Rossi, A. Lendasse, D. François, V. Wertz, M. Verleysen, Mutual information for the selection of relevant variables in spectrometric nonlinear modelling, Chemometr. Intell. Lab. Syst. 80 (2) (2006) 215–226, http://dx.doi.org/10.1016/j.chemolab.2005.06.010.
[5] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[6] G.-B. Huang, D. Wang, Y. Lan, Extreme learning machines: a survey, Int. J. Mach. Learn. Cyb. 2 (2011) 107–122.
[7] G.-B. Huang, D. Wang, Advances in extreme learning machines (elm2010), Neurocomputing 74 (16) (2011) 2411–2412.
[8] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cyb. B Cyb. 99 (2011) 1–17.
[9] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16-18) (2008) 3460–3468, advances in Neural Information Processing (ICONIP 2006)/ Brazilian Symposium on Neural Networks (SBRN 2006).
[10] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, OP-ELM: optimally-pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162, http://dx.doi.org/10.1109/TNN.2009.2036259.
[11] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the 14th international joint conference on Artificial intelligence, vol. 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995, pp. 1137–1143.
[12] B. Efron, R. Tibshirani, An Introduction to the Bootstrap, Monographs on Statistics and Applied Probability, Chapman & Hall, 1993.
[13] D.M. Allen, The relationship between variable selection and data augmentation and a method for prediction, Technometrics 16 (1) (1974) 125–127.
[14] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. B 58 (1994) 267–288.
[15] J.M.F. Bach, R. Jenatton, G. Obozinski, Convex optimization with sparsity-inducing norms, in: S.J.W.S. Sra, S. Nowozin (Eds.), Optimization for Machine Learning, MIT Press, 2011.
[16] P. Bradley, O. L. Mangasarian, Feature selection via concave minimization and support vector machines, in: Machine Learning Proceedings of the Fifteenth International Conference (ICML 98), Morgan Kaufmann, 1998, pp. 82–90.
[17] J. Zhu, S. Rosset, T. Hastie, R. Tibshirani, 1-norm support vector machines, in: S. Thrun, L.K. Saul, B. Schölkopf (Eds.), Advances in Neural Information Processing Systems, vol. 16, MIT Press, 2004.
[18] E.K. Burke, G. Kendall (Eds.), Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, 2006.
[19] C.M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), 1st ed., Springer, 2007.
[20] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1998.
[21] G.-B. Huang, C.-K. Siew, Extreme learning machine with randomly assigned RBF kernels, Int. J. Inf. Technol. 11 (1) (2005) 16–24.
[22] C. Rao, S. Mitra, Generalized Inverse of Matrices and its Applications, Wiley, 1971.
[23] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[24] M. Verleysen, F. Rossi, D. François, Advances in feature selection with mutual information, in: M. Biehl, B. Hammer, M. Verleysen, T. Villmann (Eds.), Similarity-Based Clustering, Lecture Notes in Computer Science, vol. 5400, Springer, Berlin/Heidelberg, 2009, pp. 52–69.
[25] Environmental and industrial machine learning group, http://research.ics.tkk.fi/eiml/datasets.shtml.

[26] A. Lendasse, J.A. Lee, V. Wertz, M. Verleysen, Forecasting electricity consumption using nonlinear projection and self-organizing maps, Neurocomputing 48 (1–4) (2002) 299–311.
[27] A.S. Weigend, N.A. Gershenfeld, Results of the time series prediction competition at the Santa Fe Institute, in: International Symposium on Neural Networks, 1993.
[28] A. Guillén, D. Sovilj, F. Mateo, I. Rojas, A. Lendasse, Minimizing the delta test for variable selection in regression problems, Int. J. High Perform. Syst. Archit. 1 (4) (2008) 269–281.

Benoît Frénay received an Engineer's degree from the Université catholique de Louvain (UCL), Belgium, in 2007. He is now a Ph.D. student at the UCL Machine Learning Group. His main research interests in machine learning include support vector machines, extreme learning, graphical models, classification, data clustering, probability density estimation and label noise.

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 the 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 recently received his Ph.D. degree in Computer Science and Signal and Image Processing from both the Aalto University School of Science and Technology (Finland) and the INPG (France). His main research interests are steganography/steganalysis and machine learning for classification/regression.

Michel Verleysen received the M.S. and Ph.D. degrees in electrical engineering from the Université catholique de Louvain (Belgium) in 1987 and 1992, respectively. He was an invited professor at the Swiss E.P.F.L. (Ecole Polytechnique Fédérale de Lausanne, Switzerland) in 1992, at the Université d'Evry Val d'Essonne (France) in 2001, and at the Université ParisI-Panthéon-Sorbonne from 2002 to 2011, respectively. He is now a Full Professor at the Université catholique de Louvain, and Honorary Research Director of the Belgian F.N.R.S. (National Fund for Scientific Research). He is an editor-in-chief of the Neural Processing Letters journal (published by Springer), a chairman of the annual ESANN conference (European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning), a past associate editor of the IEEE Transactions on Neural Networks journal, and member of the editorial board and program committee of several journals and conferences on neural networks and learning. He was the chairman of the IEEE Computational Intelligence Society Benelux chapter (2008–2010), and member of the executive board of the European Neural Networks Society (2005–2010). He is the author or co-author of more than 250 scientific papers in international journals and books or communications to conferences with reviewing committee. He is the co-author of the scientific popularization book on artificial neural networks in the series "Que Sais-Je?", in French, and of the "Nonlinear Dimensionality Reduction" book published by Springer in 2007. His research interests include machine learning, artificial neural networks, self-organization, time-series forecasting, nonlinear statistics, adaptive signal processing, and high-dimensional data analysis.

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 chief research scientist 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 140 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, nonlinear approximation in financial problems, functional neural networks and classification.

Publication IV

Alberto Guillén, Maribel García Arenas, Mark van Heeswijk, Dušan Sovilj, Amaury Lendasse, Luis Herrera, Hector Pomares and Ignacio Rojas. Fast Feature Selection in a GPU Cluster Using the Delta Test. *Entropy*, 16 (2): pp. 854-869, 2014.

Entropy **2014**, *16*, 854-869; doi:10.3390/e16020854

OPEN ACCESS

entropy

ISSN 1099-4300

www.mdpi.com/journal/entropy

Article

Fast Feature Selection in a GPU Cluster Using the Delta Test

Alberto Guillén [1,*], M. Isabel García Arenas [1], Mark van Heeswijk [2], Dusan Sovilj [2], Amaury Lendasse [2,3,4], Luis Javier Herrera [1], Héctor Pomares [1] and Ignacio Rojas [1]

[1] Department of Computer Architecture and Computer Technology, Universidad de Granada, Granada 18071, Spain; E-Mails: mgarenas@ugr.es (M.I.G.A.); jherrera@ugr.es (L.J.H.); hector@ugr.es (H.P.); irojas@ugr.es (I.R.)

[2] Department of Information and Computer Science, Aalto University School of Science, Espoo 02150, Finland; E-Mails: mark.van.heeswijk@aalto.fi (M.H.); dusan.sovilj@aalto.fi (D.S.); lendasse@hut.fi (A.L.)

[3] IKERBASQUE, Basque Foundation for Science, Bilbao 48011, Spain

[4] Arcada University of Applied Sciences, 00550 Helsinki, Finland

* Author to whom correspondence should be addressed; E-Mail: aguillen@ugr.es; Tel.: +34-958-240588.

Received: 13 October 2013; in revised form: 10 January 2014 / Accepted: 28 January 2014 / Published: 13 February 2014

Abstract: Feature or variable selection still remains an unsolved problem, due to the infeasible evaluation of all the solution space. Several algorithms based on heuristics have been proposed so far with successful results. However, these algorithms were not designed for considering very large datasets, making their execution impossible, due to the memory and time limitations. This paper presents an implementation of a genetic algorithm that has been parallelized using the classical island approach, but also considering graphic processing units to speed up the computation of the fitness function. Special attention has been paid to the population evaluation, as well as to the migration operator in the parallel genetic algorithm (GA), which is not usually considered too significant; although, as the experiments will show, it is crucial in order to obtain robust results.

Keywords: general-purpose computing on graphics processing units (GPGPU); feature selection; variable selection; big data

1. Introduction

The problem of variable selection is crucial for the time of the design models that classify or perform regression; so, the better the selection is, the more accurate that models can be designed [1–3]. Although classification and regression are both modeling problems, they should be treated separately. Thus, the work presented in this paper focuses on regression, also known as function approximation. Formally, the function approximation problem is to determine, given a set of input/output pairs $(\mathbf{x}_i, y_i) \in R^d \times R \;\; i = 1...N$ and an unknown function, f, such as $f(\mathbf{x}_i) \approx y_i$. From this, the problem of variable selection can be defined as the search for the subset of variables that make it possible to build a model that approximates the data as accurately as possible.

At first, the method to determine the most adequate subset of variables could seem trivial: build a model for each possible solution. This approach could be considered as the brute force approach and would be optimal; however, it has critical problems:

- It is not possible to compute all the combinations when the problem has a large number of variables, due to the computational cost;
- The validity criterion depends in the model, which usually has several parameters that affect the final results considerably.

These problems have encouraged the use of non-parametric metrics (which are model independent) to be more objective about the time of evaluating a solution. One of the mainstream ways is to use mutual information (MI) [4] as a criterion. MI is defined as the difference between the sum of the individual entropies and the joint entropy and of two events (in the variable selection problem, the events are the output, Y, and the inputs, X). The higher it is, the more relevant for the model the variables selected for X should be. This approach has been successfully applied using several estimators to compute it. Another non-parametric criterion that has been applied to variable selection is the delta test (DT) [5]. There are few papers that apply it, although its adequacy to perform variable selection has been theoretically demonstrated in [6,7]. Concretely, in [7], there are comparisons of DT *versus* MI using Kraskov's estimator, where DT shows better performance. A recent comparison between both approaches has been done in [8], where the MI showed a slightly better performance according to experts' opinions. However, in [9], MI estimation using the Parzen window and Kraskov's method were outperformed by DT, considering objective metrics as the approximation error using least squares support vector machines and experts' opinions. The interesting thing about this last paper is that, due to the small number of samples, an exhaustive search is performed, so that the global optimum for each metric is obtained. This encourages the research presented in this paper, because it emphasizes how important it is to analyze as many solutions as possible. Therefore, this work is focused on optimizing a parallel genetic algorithm, which is implemented on a novel high performance computing (HPC) architecture using clusters of computers with several graphical processing units (GPUs). Using this architecture, speed up can be obtained by using GPUs to compute the fitness (in this case, the DT) and CPUs to parallelize the genetic algorithm (GA). Moreover, the parallelization of optimization algorithms seems to be the only way to obtain solutions in large dataset problems, as there are computation and time limitations if a single machine is used. This work presents analyses, as well one of the most important parameters in parallel genetic algorithms: the migration operator.

The rest of the paper is organized as follows: Section 2 introduces the delta test. Afterwards, Section 3 describes the design of the parallel genetic algorithm, which will perform the optimizations presented in the experiments included in Section 4. Finally, conclusions are discussed.

2. Delta Test in Variable Selection

In order to evaluate the goodness of an individual, the delta test (DT) [5] value obtained using the combination of variables is used, as it has been shown to be an adequate criterion [6,10].

In function estimation, the main problem is to find the correct signal and noise terms. As mentioned, the relationship between input (signal) $\mathbf{x}_i$ and output y_i is represented as

$$y_i = f(\mathbf{x}_i) + r_i, i = 1, \ldots, N \tag{1}$$

where r_i is the noise term. The usual assumption in many domains behind r_i is that the term is independent and identically distributed following the normal distribution $r_i \sim \mathcal{N}(0, \sigma^2)$. In the training phase, the goal is to have a model with a good generalization ability that does not "learn" the noise, r_i. The DT provides a way to find out how much noise is present in the data *before* the modeling stage. That is, it is estimating the variance of the noise, σ^2, or the mean squared error (MSE), which can be obtained without overfitting.

The DT is computed using the nearest neighbor approach as:

$$\sigma^2 = \mathrm{Var}[r] \approx \frac{1}{2N}\sum_{i=1}^{N}(y_i - y_{NN(i)})^2 \tag{2}$$

where the first nearest neighbor of a point, $\mathbf{x}_i$, in the R^d space is $\mathbf{x}_{NN(i)}$ and $y_{NN(i)}$ is the output of $\mathbf{x}_{NN(i)}$. The DT is a special case of the Gamma Test [11], another noise variance estimator is based on nearest neighbor distributions. The difference is in the extra hyper-parameter present in the Gamma Test (the number of neighbors), while the DT uses only the first nearest neighbor, providing a fully non-parametric method. The reduction to only the closest neighbor still gives the unbiased estimator of the noise variance in the limit $N \to \infty$ [10].

2.1. Computation of Delta Test Using Pre-Calculated Distances

Computation of the nearest neighbor in the naive way involves calculating the distances between each pair of samples:

$$\mathbf{d}_{i,j} = \sum_{m=1}^{d}(x_i^{(m)} - x_j^{(m)})^2 \tag{3}$$

and returning the smallest $\mathbf{d}_{i,j}$ and the corresponding index of the Nearest Neigbor, $NN(i)$, for each sample. Since the focus is on examining non-empty subsets of variables that can share individual elements, a lot of time is wasted recomputing the squared differences to obtain $\mathbf{d}_{i,j}$. A simple solution to decrease the running time is to store that information into a $N(N-1)/2 \times d$ matrix, where each row contains precomputed squared differences for a pair of samples (x_i, x_j). Given this matrix, computing all pairwise distances for a given variable subset $I \subseteq \{1, 2, \ldots, d\}$ involves summing precomputed values for those I variables (*i.e.*, the I-th columns of the matrix).

2.2. Computation of the Delta Test on a GPU

The computation of the k nearest neighbors (KNN) requires great computational effort, since it has to compute the pairwise distances between all the points and, then, sort them to choose the closest ones. In [12], an implementation of the KNN algorithm on a GPU (the code is available at [13]) is presented. In this approach, brute force is used to compute the distances between the input vectors, as they are independent of each other; therefore, they are easily parallelized using the single instruction multiple data (SIMD) paradigm.

After computing the distances, they need to be sorted to determine the closest neighbors. To do so, the insertion sort algorithm used is Quicksort; due to its recursive nature, it cannot be implemented on GPUs.

All the processes require about 10 times faster speed than other algorithms implemented in libraries, such as ANN (Approximate Nearest Neighbor [14]) which uses kd-trees and box-decomposition trees to improve performance.

Another advantage of using the GPU is that, although the computation is repeated constantly, it allows one to handle large problems, unlike the pre-calculated distance approach, which might have memory limitations.

3. Design of the Genetic Algorithm

Genetic algorithms (GAs) are a well-known optimization tool that has been applied to many problems. This kind of algorithm is based on the principles of natural evolution, where the offspring of the new generations are supposed to be better than their parents. The main advantage of GAs is that they have the ability to explore the solution space globally, but at the same time, they can exploit the regions where the best solutions are.

When a GA is designed, the first decision to be made is how an individual will represent a solution, that is, the solution encoding. The variable selection problem has a straightforward encoding using a binary chromosome, whose length is equal to the number of variables. If a gene within the chromosome equals one, the variable is selected; if it is zero, then the variable is discarded.

Once the encoding of the solutions is defined, it is necessary to determine the operators that will affect the evolution of the individuals. These are introduced in the following subsections.

3.1. GA Operator Description

There are several types of operators that will modify the chromosome of an individual. The first one is the selection operator, which determines if an individual is going to reproduce and generate offspring. If the selection is very restrictive and only the best individuals reproduce, the convergence of the algorithm is accelerated. On the other hand, if random individuals are chosen, there is the risk of not converging to a good solution, as well as a lack of robustness. The chosen operator is "binary tournament selection", which consists of taking two couples of individuals and selecting from each couple the individual that has better fitness.

The second one is the crossover operator, which is responsible for combining the genes of the parents to produce the offspring. For the binary encoding, many operators have been proposed, although the two-points binary crossover seems to have a compromise between the exploration/exploitation of the solutions [15]. This operator defines two points in the chromosome where the individuals exchange genes, generating two new individuals.

Once the offspring are obtained, they might suffer a mutation in their chromosome. This mutation is the one that helps the algorithm to escape from local minima and jump to other regions of the solution space. The mutation selected was at the gene level, that is, if the individual is mutating, select-unselect a random variable.

Finally, to obtain the population $t + 1$, it has to be decided if the whole population, t, is going to be replaced (generational GA) or only some of the offspring replace their ancestors (stationary GA). As there is the need to obtain a high number of solutions, it is more adequate to use the newly generated individuals, but to maintain exploitation, the best individuals of the previous generation are kept. This is known as elitism [16].

The evolution continues until a stop criterion is satisfied. There are many possibilities and approaches proposed in the literature [17]; however, the proposed algorithm only considers an execution time limit of 600 s. This criterion affects all the previous operators, as well as their parameters (*i.e.*, crossover and mutation probabilities).

The reason for choosing 600 s is because this value is considered as the maximum time an operator is willing to wait before a solution is provided. Even though variable selection is an $off - line$ problem, a human operator might decide to recompute solutions with new collected data at any time. This time limit has been previously used in the literature for GAs [18,19], and it is a common time barrier used in many programs implemented by important companies, like Microsoft [20], Apple [21] , Apache [22] , *etc.*

Therefore, the limitation implies that the convergence should be fast, but diversity should still be reasonably maintained. These are the reasons to select the binary tournament (high selection pressure), two-point binary crossover with a high probability of obtaining a high exploitation of the solutions, mutation at a gene level, but with a high probability of maintain exploration, and elitism, to always keep the best chromosomes so far (high exploitation).

To summarize, the standard operators chosen for the algorithm are:

(1) Crossover: two-point binary with 0.8 probability.
(2) Mutation: gene level with 0.1 probability.
(3) Selection: binary tournament selection.
(4) Elitism: keep 10% of the best individuals from the previous generation.
(5) Stop criterion: 600-s time limit.

3.2. Parallelizing the GA

Apart from the optimization of the computation of the fitness function using GPUs, the available architecture allows the algorithm to be distributed through several machines in a classical cluster manner. GAs are intrinsically parallel in the sense that several operations can be done in parallel, because they are independent. However, modifications in the execution flow, like splitting the populations into sub-populations, also known as demes, lead to better results [23–26].

3.2.1. Island Model and Migration/Immigration Policy

Among the several approaches proposed in the literature to parallelize GAs [27], the multi-deme distributed GA (also known as the island model) is one of the most popular, due to its good behavior [26–28]. This implementation consists of evolving isolated populations on different islands and, in some cases, exchanging individuals. The implementation of this paradigm usually maps an island to a processor. In the available architecture for this work, since a processor might have several cores, populations are mapped to cores. Regarding the use of the GPUs to compute the DT for each individual, an island uses one GPU to compute the distance matrix.

The mechanism that exchanges individuals between the islands is known as the migration operator. The benefits of this procedure can be different, depending on the way it is applied and the criterion that selects the individuals [29]; it can help to maintain diversity by exploring new solutions provided by the new individuals, or it might help exploitation, because all islands will share some equal individuals. The communication between the islands requires one to consider the following parameters:

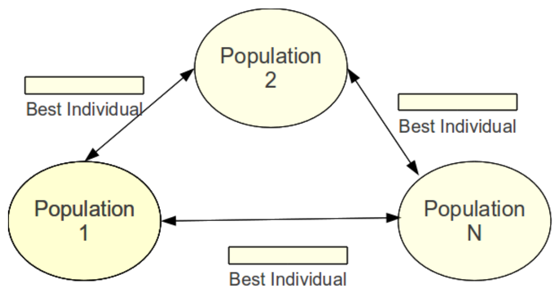
- Send policy: decides which individual or individuals should be sent.
- Acceptance policy: decides if one or several individuals coming from other islands are or are not included in the destination island.
- Replacement policy: if new individuals are incorporated into the population, this policy must determine which individuals of the current population should be replaced (in order to maintain the population size).
- Communication topology: defines the migration flow, that is, between which islands are individuals being exchanging.
- Communication frequency: determines when the migration step will be carried out.

The analysis of the GA behavior as a function of the migration process has not been studied deeply since some years ago [30,31], although recent papers study this parameter, as it has an important influence on the results [32,33]. Concretely, in [33], the authors propose the MultiKulti algorithm, whose main novelty is to introduce a criterion that determines if an island accepts a migration or does not. The results presented showed that the diversity and the number of individual evaluations before the algorithm finished increased using this migration filtering.

The migration rate is a parameter that can be random [34], fixed [26] or autoregulated, depending on the diversity of the population. However, to consider the population diversity to modulate the migration rate is equivalent to determining a replacement policy. The reason is that if the replacement policy determination is based as well on the current population and does not accept any replacement, it is as if the migration rate has been decreased.

The details on how the algorithm performs the selection of the individuals selected to be migrated and the ones to be replaced is analyzed in detail in Section 4.3. Regarding the migration topology, all the islands communicate with each other in a collective manner, as is depicted in Figure 1.

Figure 1. Fully connected island migration scheme.



3.2.2. Population Distribution in the Cluster

Since the available architecture is a heterogeneous grid of computers, it is obvious that the time to complete a generation during the run will be different on each computer [35]. As the distributed populations perform a collective communication broadcasting to individuals, the global performance of the algorithm would be injured if the fastest machines had to wait for the slower ones, wasting computing resources. In order to ameliorate this fact, the decision of setting different population sizes has been taken. Slower or overloaded machines will process less individuals, allowing these processes to require a shorter time to complete a generation. Therefore, during the collective communications, the waiting time for the synchronization will be reduced. Obviously, this approach is the same as increasing the predefined population size in the more powerful machines.

The question that arises from this policy is: how much should the size of the population be decreased/increased? The answer can be obtained empirically by measuring the time for one generation on each machine and obtaining the fraction between the fastest/slowest and the other time measurements. For example, if the time of $Machine_1$ is double that of $Machine_2$, the population size for $Machine_1$ should be half (or the population size for $Machine_2$ should be doubled).

Therefore, a communication step must be performed at the beginning of the algorithms: the root process (rank 0) is the reference, so each process with its corresponding GPU performs the evaluation of the same individual (*i.e.*, all variables selected) and waits for the message from the root process, indicating how much it took to perform the evaluation. Then, each process computes the number of individuals as:

$$round\left(sizePop * \frac{timeDT}{rootTimeDT}\right) \tag{4}$$

keeping in mind that the number of individuals must be even.

3.2.3. Optimizing the Computation of the Population Fitness

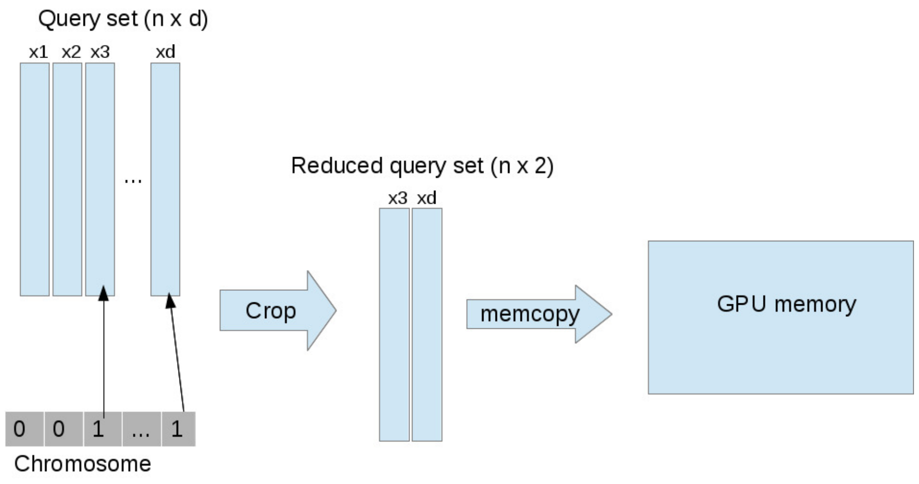
The implementation presented by [12] had to be adapted in order to use several GPUs within the same node. To do so, several functions from the NVIDIAlibrary have to be called, as is described in [36], to set the desired GPU by each process. For each time the KNN function is called, the whole reference dataset has to be copied to the GPU memory to compute the distances. Therefore, during the evaluation of the whole population, this process has to be repeated as many times as the number of individuals with the possibility of creating an overhead time that could be avoided, since a process will not use other

GPUs than the one selected and the reference dataset is the same. The implementation of the function was modified, so that the complete population was given to the mex-function, which computes the KNN neighbors. Thus, the kernels, when computing the distances just have to accumulate or not accumulate a distance in a certain dimension depending on the value of the gene in the chromosome.

However, the implementation splits the queries to fit in the GPU memory, and this depends in the dimension of the dataset, performing a "memcopy" from the host to the device for each split. Therefore, the higher the dimension of the problem is, the longer it will take to compute the distances.

In this paper, this computation has been optimized, as the dimension of the dataset defined by each chromosome is different. For example, a chromosome that encodes a selection of just two variables does not need the other $d - 2$ variables. Therefore, the function distance computation function is called as many times as there are individuals, but cropping the dimensionality of the input vectors on each call to match the variable selection encoded on each chromosome. This process is depicted in Figure 2 with a chromosome that encodes a solution with two variables.

Figure 2. The procedure to crop the query set based on the chromosome in order to make the computation of the distances and the data transfer faster.



This cropping is easily performed in MATLAB, due to its efficient matrix operations. Therefore, the computation of the fitness for individuals with a low number of variables becomes much faster, as the KNN function does not have to split the query to iterate through all the dimensions of the dataset.

This reduction in time is much higher than the one obtained by reducing the number of calls to select the GPU and to perform the copy of the reference set to the GPU memory.

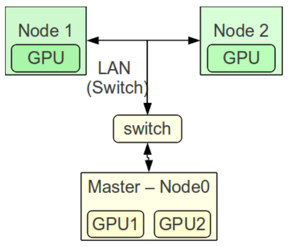
4. Experiments

This section shows the performance of the proposed algorithm when compared to previous approaches using small and large datasets. Afterwards, an analysis of the migration policy chosen is made.

4.1. Cluster Architecture

The cluster that was configured had the components described below that were interconnected, as Figure 3 shows.

Figure 3. The cluster of graphical processing units (GPUs) used in the experiments.



(1) One master node with two GPUs:
Processor:

- *Model name* : (26) Intel(R) Core(TM) i7 CPU 930 @ 2.80 GHz—*cache size*: 8192 KB
- *CPU cores*: 4; siblings: 8

Two GPUs:

- *Graphics processor*: GeForce GTS 450—*CUDA cores*: 192
- *Memory*: 1024 MB; *memory interface*: 128-bit
- *Bus Type*: PCI Express x16 Gen1 - *PCI-E Max link speed*: 2500

(2) Two Local network node with one GPU:
Processor:

- *Model name*: (23) Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83 GHz—*cache size*: 6144 KB
- *CPU cores*: 4; siblings: 4

GPU:

- *Graphics processor*: GeForce 9800 GTX — *CUDA cores*: 128
- *Memory*: 512 MB; *memory interface*: 256-bit
- *Bus type*: PCIExpress x16 Gen2; *PCI-E max link speed*: 5000

Processor:

- *Model name*: (15) Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40 GHz—*cache size*: 4096 KB
- *cpu cores*: 4; siblings: 4

GPU:

- *Graphics processor*: GeForce 8400 GS—*CUDA cores*: 16
- *Memory*: 512 MB; *memory interface*: 64-bit
- *Bus type*: PCIExpress x16; *PCI-E max link speed*: not available

4.2. Comparison with Previous Approaches

This section will analyze the performance and behavior of the proposed algorithm. First, small datasets are compared with a previous work; afterwards, the algorithm is applied to a large real-world dataset.

4.2.1. Small Datasets

The algorithm is compared with the approach presented in [37], which is a previous parallel version of the algorithm implemented in a regular cluster. For the sake of a fair comparison, the stop criteria will be the same: execution time. In [37], the time limit is set to 600 s, based on studies and recommendations from the industry arguing that that is the maximum time that an operator is willing to wait to see a solution.

The same population sizes were used (50,100 and 150), and the datasets processed were: (1) The Tecator dataset [38]: The Tecator dataset aims at performing the task of predicting the fat content of a meat sample on the basis of its near-infrared absorbance spectrum. The dataset contains 215 useful instances for interpolation problems, with 100 input channels and 3 outputs, although only one is going to be used (fat content). (2) The Anthrokids modified dataset [39]:This dataset consists of several measures to predict a child's weight. It has 1019 instances and 55 variables.

The results are shown in Table 1. As the table reflects, the performance of the proposed algorithm is not impressive in comparison with the previous algorithm. In fact, it does not outperform the previous approach. However, it is remarkable that the difference between the solutions is not too big, and in some cases, the new algorithm outperforms the previous one, demonstrating that the proposed approach is a good algorithm. The reason because the previous algorithm obtains, in general, better solutions for these datasets is because the GA is able to generate more populations, due to the pre-calculation of the distance matrix, as it was described in Section 2.

Table 1. Performance in terms of the delta test (DT) value of the cluster of GPUs (pGPU) against sequential (seq.) and parallel approaches (where the number of processes is notated as np and the number of GPUs as nGPU) .

Dataset	**Population**	**Measurement**	**seq.**	**parallel (np = 2)**	**parallel (np = 4)**	**pGPU (np = 4, nGPU = 4)**
Anthrokids	50	Mean (DT)	0.01278 (11.5 $\times10^{-4}$)	0.01269 (14.2 $\times10^{-4}$)	0.01204 (12.6 $\times10^{-4}$)	0.01587 (8.1 $\times10^{-3}$)
	100	Mean (DT)	0.01351 (11.6 $\times10^{-4}$)	0.01266 (86.4 $\times10^{-4}$)	0.01202 (17.4 $\times10^{-4}$)	0.014553 (5.6 $\times10^{-4}$)
	150	Mean (DT)	0.01475 (12.1 $\times10^{-4}$)	0.01318 (11.2 $\times10^{-4}$)	0.01148 (9.9 $\times10^{-4}$)	0.01556 (12 $\times10^{-4}$)
Tecator	50	Mean (DT)	0.13158 (7.9 $\times10^{-4}$)	0.14297 (7.7 $\times10^{-3}$)	0.13976 (7.8 $\times10^{-3}$)	0.123803 (3.7 $\times10^{-3}$)
	100	Mean (DT)	0.13321 (3.1 $\times10^{-3}$)	0.13587 (2.4 $\times10^{-3}$)	0.13914 (8.6 $\times10^{-3}$)	0.132501 (3 $\times10^{-3}$)
	150	Mean (DT)	0.13146 (8.5 $\times10^{-4}$)	0.1345 (2.4 $\times10^{-3}$)	0.13522 (6.9 $\times10^{-3}$)	0.13197 (9.9 $\times10^{-4}$)

4.2.2. Large Datasets

The analysis of the previous results might discourage the use of GPUs instead of using a pre-calculation of the distances; however, when the dataset starts becoming a little bit bigger, this second approach is not possible any more. The reason is because the application runs out of memory, so to use a cluster of GPUs, it is not a matter of performance in time or quality results, it is a matter of being able to provide a solution.

As an example, part of the dataset provided by the Spanish Institute of Statistics (Instituto Nacional de Estadística (INE)) that contains data about marital dissolutions in Spain is used. Several problems arise from this data, and one of them is predicting the dissolution process length, which is translated into the regression problem.

The data to be used consists of 19,967 input samples of 20 variables, which is divided into training (of 15,385) test (4568) sets. The size of the training set is too big to pre-calculate the distance matrix used in previous approaches [37].

The proposed approach is executed during 600 s using the same configuration of the previous section (detailed in Section 3.1), being able to finish only one generation, with 50 individuals providing a DT value of 0.001642 using 9 variables. Due to the high memory requirements, one of the computers with the oldest GPU model was delaying the other two, because of the synchronization step in the migration. Therefore, the experiments are repeated only with two nodes instead of three. The performance increases significantly, allowing the algorithm to evolve a mean of 7.6 generations, obtaining a DT value of 0.001589 using only 4 variables.

To test the validity of the selection provided, a model (a radial basis function neural network with 15 neurons) is designed using the methodology proposed in [40] without local search optimization. The experiments are done using all the variables and using the ones selected by the algorithm, obtaining the results shown in Tables 2 and 3. The approximation errors provided by the neural networks prove how critical the necessity of performing variable selection before modeling a real world problem is. The network maintains a good generalization capability, as the differences between the test error and the training error are small.

Table 2. Delta test values for the large-sized dataset (standard deviation in brackets).

Running Time	DT value (std)	# variables	# generations (std)
600 s (3 nodes, 4 GPUs)	0.001629 (2×10^{-4})	9	1 (0)
600 s (2 nodes, 3 GPUs)	0.001592 (1×10^{-4})	4	7.6 (0.5)

Table 3. Approximation errors (Normalized Root Mean Square Error - NRMSE) of the large dataset with and without variable selection using an RBFNN (Radial Basis Function Neural Network) with 15 neurons.

	Train Error	Test Error
with variable selection	0.4846	0.5017
without variable selection	1.3086	1.3197

4.3. Comparison Between Different Migration Policies

Another experiment carried out consists of a comparison of several approaches regarding how the communication between the islands is performed. In order to isolate as much as possible the effect of the migration parameters, the seeds of the random function used on each run is the same, so that the unique element that could modify the results is the migration step. The setting for the experiments is:

- **Send policy**:
  Send best (SB): selects the best individual of the current population;
  Send random (SR): selects a random individual of the current population.
- **Acceptance policy**:
  Always accept (AA): always includes the incoming individuals to the population;
  Accept if different (AiD): includes the individuals if they fulfill a predefined criterion.
- **Replacement policies**:
  Replace worst (RW): substitutes the worst individual by the incoming one.
- **Communication topology**:
  fully connected (send to all, receive from all).
- **Communication frequency**:
  synchronous fixed rate each 5 generations.

Regarding the send policy, the two selected options are considered, as are the most popular in the literature. The acceptance policy consists of always accepting the incoming individuals or only accepting them if they are *different* from the best individual in the population. In the implementation, an individual $\vec{i} = i_1, i_2, ..., i_d$ with $i_k \in 0, 1$ is considered different from another individual if the Hamming distance between them is not higher than $d/2$.

As the tackled problem requires huge computational effort, only the worst individual replacement policy is chosen with the aim of converging to a good solution fast. Other approaches could consider other replacement policies.

4.3.1. Diversity of the Population

The value of the average Hamming distance between each pair of chromosomes is chosen to be the metric that determines the diversity of the population. For each run, the evolution of this parameter is recorded and, then, the average of the evolutions is computed; these values are graphically depicted in Figure 4. It is easy to see how the population maintains diversity when the conditional acceptance criterion is applied, and the configuration, SB-AA-RW, is the one that has less diversity in the population, confirming previous results.

4.3.2. Accuracy and Robustness

The reduction of the diversity by not using a conditional acceptance criterion could seem a disadvantage; however, as Table 4 shows, it is totally the opposite case, because it allows the algorithm to exploit the less divergent individuals more and to achieve better performance. This table also shows that the use of the conditional criterion does not affect the final result at all.

Figure 4. Diversity of the population using different communication parameters. SB, send best; AA, always accept; RW, replace worst; AiD, accept if different; SR, send random.

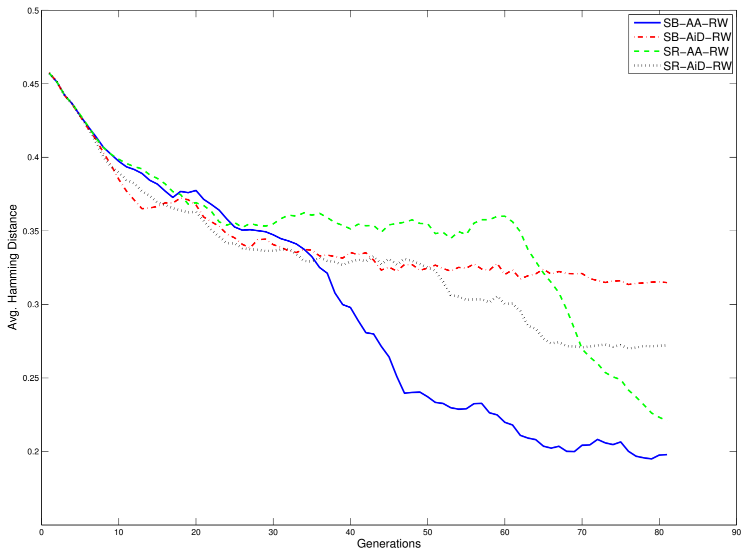


Table 4. Average results and the standard deviation of the best individual obtained.

Migration policy	DT value
SB-AA-RW	1.668×10^{-3} (8.8×10^{-5})
SB-AiD-RW	1.763×10^{-3} (1.1×10^{-4})
SR-AA-RW	1.721×10^{-3} (1.3×10^{-4})
SR-AiD-RW	1.721×10^{-3} (1.3×10^{-4})

5. Conclusions

In this paper, we present a new algorithm that takes advantage of new high performance computing technologies. The main novelty is the use of a cluster, where the nodes have graphical processing units to compute the fitness function efficiently; this is a cluster of a cluster of GPUs. Another important contribution is the analysis of several migration policies, showing their influence on population diversity and result robustness. The performance of the algorithm for small datasets is acceptable when compared with previous methods, plus it is able to provide good results in a reasonable time. Even when the size of the dataset becomes too large, the proposed algorithm is able to provide a good solution, in contrast to the previous approach, which is not able to provide any.

Acknowledgments

This work was supported in part by the Consejería de Innovación, Ciencia y Empresa of the Spanish Junta de Andalucía, under Project TIC2906 and in part by the Spanish Ministry of Science and Innovation under Project SAF2010-20558.

Author Contributions

A. Guillén and L.J. Herrera have lead the ellaboration of the paper through all the stages. The rest of the authors have cooperated in the writting and revision of the paper.

Conflict of Interest

The authors declare no conflict of interest.

References

1. Bellman, R. *Adaptive Control Processes—A Guided Tour*; Princeton University Press: Princeton, NJ, USA, 1961.
2. Herrera, L.; Pomares, H.; Rojas, I.; Verleysen, M.; Guillén, A. Effective Input Variable Selection for Function Approximation. In *Artificial Neural Networks—ICANN 2006*, Proceedings of 16th International Conference, Athens, Greece, 10–14 September 2006; Part I, pp. 41–50.
3. Liitiäinen, E.; Corona, F.; Lendasse, A. On the curse of dimensionality in supervised learning of smooth regression functions. *Neural Process. Lett.* **2011**, *34*, 133–154.
4. Shannon, C.E. A mathematical theory of communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423.
5. Pi, H.; Peterson, C. Finding the embedding dimension and variable dependencies in time series. *Neural Comput.* **1994**, *6*, 509–520.
6. Eirola, E.; Liitiäinen, E.; Lendasse, A.; Corona, F.; Verleysen, M. Using the Delta Test for Variable Selection. In Proceedings of the ESANN 2008, European Symposium on Artificial Neural Networks, Bruges, Belgium, 23–25 April 2008; pp. 25–30.
7. Eirola, E. Variable Selection with the Delta Test in Theory and Practice. Master's Thesis, Helsinki University of Technology, Espoo, Finland, 2009.
8. Liébana-Cabanillas, F.J.; Nogueras, R.; Herrera, L.J.; Guillén, A. Analysing user trust in electronic banking using data mining methods. *Expert Syst. Appl.* **2013**, *40*, 5439–5447.
9. Del Moral, R.G.; Guillén, A.; Herrera, L.J.; Cañas, A.; Rojas, I. Parametric and Non-Parametric Feature Selection for Kidney Transplants. In Proceedings of The International Work-Conference on Artificial Neural Networks (IWANN), Tenerife, Spain, 12–14 June 2013; pp. 72–79.
10. Liitiäinen, E.; Verleysen, M.; Corona, F.; Lendasse, A. Residual variance estimation in machine learning. *Neurocomputing* **2009**, *72*, 3692–3703.
11. Jones, A.J. New tools in non-linear modeling and prediction. *Comput. Manag. Sci.* **2004**, *1*, 109–149.
12. Garcia, V.; Debreuve, E.; Barlaud, M. Fast k Nearest Neighbor Search Using GPU. In Proceedings of the CVPR Workshop on Computer Vision on GPU, Anchorage, AK, USA, 23–28 June 2008.
13. Source code for computing KNN using GPU. Available online: http://www.i3s.unice.fr/∼creative/KNN/ (accessed on 13 October 2013).
14. David M. Mount and Sunil Arya ANN: A Library for Approximate Nearest Neighbor Searching.Available online: http://www.cs.umd.edu/m̃ount/ANN/ (accessed on 13 October 2013).

15. Obaid, O.I.; Ahmad, M.; Mostafa, S.A.; Mohammed, M.A. Comparing performance of genetic algorithm with varying crossover in solving examination timetabling problem. *J. Emerg. Trends Comput. Inf. Sci.* **2012**, *3*, 1427–1434 .
16. Goldberg, D.E. Genetic and evolutionary algorithms come of age. *Commun. ACM* **1994**, *37*, 113–119.
17. Jong, K.A.D. *Evolutionary Computation—A Unified Approach*; MIT Press: Cambridge, MA, USA, 2006; pp. I–IX, 1–256.
18. Wang, L.; Kazmierski, T. VHDL-AMS Based Genetic Optimization of a Fuzzy Logic Controller for Automotive Active Suspension Systems. In Proceedings of the 2005 IEEE International Behavioral Modeling and Simulation Workshop, San Jose, CA, USA, 22–23 September 2005; pp. 124–127.
19. Zhang, J.; Li, S.; Shen, S. Extracting Minimum Unsatisfiable Cores with a Greedy Genetic Algorithm. In *AI 2006: Advances in Artificial Intelligence*, Proceedings of 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, 4–8 December 2006; pp. 847–856.
20. Maximum wait time for group policy scripts. Available online: http://technet.microsoft.com/enus/library/ cc940027.aspx (accessed on 1 December 2013).
21. Minimum acceptable timeout value. Available online: https://developer.apple.com/library/ios/DOCUMENTATION/UIKit/Reference/UIApplication_Class/Reference/Reference.html (accessed on 1 December 2013).
22. Maximum time an item in the search/bind cache remains valid. Available online: http://httpd.apache.org/ docs/current/ mod/mod_ldap.html (accessed on 1 December 2013).
23. Alba, E.; Tomassini, M. Parallelism and evolutionary algorithms. *IEEE Trans. Evol. Comput.* **2002**, *6*, 443–462.
24. Guillén, A.; Rojas, I.; González, J.; Pomares, H.; Herrera, L.; Paechter, B. Boosting the Performance of a Multiobjective Algorithm to Design RBFNNs through Parallelization. In *Adaptive and Natural Computing Algorithms*, Proceedings of 8th International Conference, Warsaw, Poland, 11–14 April 2007; pp. 85–92.
25. Guillén, A.; Pomares, H.; González, J.; Rojas, I.; Valenzuela, O.; Prieto, B. Parallel multiobjective memetic RBFNNs design and feature selection for function approximation problems. *Neurocomputing* **2009**, *72*, 3541–3555.
26. Guillén, A.; Rojas, I.; González, J.; Pomares, H.; Herrera, L.; Paechter, B. Improving the Performance of Multi-Objective Genetic Algorithm for Function Approximation through Parallel Islands Specialisation. In *AI 2006: Advances in Artificial Intelligence*, Proceedings of 19th Australian Joint Conference on Artificial Intelligence, Hobart, Australia, 4–8 December 2006; pp. 1127–1132.
27. Cantú-Paz, E. *Efficient and Accurate Parallel Genetic Algorithms*; Kluwer Academic Publishers: Norwell, MA, USA, 2000.
28. Herrera, F.; Lozano, M. Gradual distributed real-coded genetic algorithms. *IEEE Trans. Evol. Comput.* **2000**, *4*, 43–63.
29. Chiwiacowsky, L.; de Velho, H.; Preto, A.; Stephany, S. Identifying Initial Conduction in Heat Conduction Transfer by a Genetic Algorithm: A Parallel Approach. In Proceedings of the XXIV

iberian Latin-American Congress on Computacional Methods in Engineering, Ouro Preto, Brazil, 29–31 October. 2003; pp. 180–195.
30. Alba, E.; Troya, J.M. Influence of the migration policy in parallel distributed GAs with structured and panmictic populations. *Appl. Intell.* **2000**, *12*, 163–181.
31. Cantú-Paz, E. Migration Policies and Takeover Times in Parallel Genetic Algorithms. In Proceedings of Genetic and Evolutionary Computation Conference, Orlando, FL, USA, 13–17 July 1999.
32. Araujo, L.; Guervós, J.J.M. Diversity through multiculturality: Assessing migrant choice policies in an island model. *IEEE Trans. Evol. Comput.* **2011**, *15*, 456–469.
33. García-Sánchez, P.; Eiben, A.E.; Haasdijk, E.; Weel, B.; Guervós, J.J.M. Testing Diversity-Enhancing Migration Policies for Hybrid On-Line Evolution of Robot Controllers. In Proceedings of the EvoApplications, Málaga, Spain, 11–13 April 2012; pp. 52–62.
34. Miki, T.H.M.; Negami, M. Distributed Genetic Algorithms with Randomized Migration Rate. In Proceedings of the IEEE Conference Systems, Man and Cybernetics, Tokyo, Japan, 12–15 October 1999; pp. 689–694.
35. Alba, E.; Nebro, A.J.; Troya, J.M. Heterogeneous computing and parallel genetic algorithms. *J. Parallel Distrib. Comput.* **2002**, *62*, 1362–1385.
36. Guillén, A.; Arenas, M.G.; Herrera, L.; Pomares, H.; Rojas, I. GPU Cluster with MATLAB. In Proceedings of the The 2011 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, USA, 18–21 July 2011; pp. 379–384.
37. Guillén, A.; Sovilj, D.; Lendasse, A.; Mateo, F.; Rojas, I. Minimising the delta test for variable selection in regression problems. *Int. J. High Perform. Syst. Archit.* **2008**, *1*, 269–281.
38. StatLib: Data, Software and News from the Statistics Community. Available online: http://lib.stat.cmu.edu/datasets/tecator (accessed on 8 February 2014).
39. Environmental and Industrial Machine Learning Group. Anthropometric study of US Children Carried Out in 1977. Available online: http://research.ics.tkk.fi/eiml/datasets.shtml (accessed on 8 February 2014).
40. Guillén, A.; González, J.; Rojas, I.; Pomares, H.; Herrera, L.; Valenzuela, O.; Prieto, A. Improving clustering technique for functional approximation problem using fuzzy logic: ICFA algorithm. *Neurocomputing* **2007**, doi:10.1016/j.neucom.2006.06.017.

Publication V

Mark van Heeswijk, and Yoan Miche. Binary/Ternary Extreme Learning Machines. *Neurocomputing*, 149, pp. 187-197, February 2015.

Neurocomputing 149 (2015) 187–197

Contents lists available at ScienceDirect

Neurocomputing

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

Binary/ternary extreme learning machines

Mark van Heeswijk*, Yoan Miche

Aalto University School of Science, Department of Information and Computer Science, P.O. Box 15400, FI-00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 28 August 2013
Received in revised form
17 December 2013
Accepted 19 January 2014
Available online 28 September 2014

Keywords:
Extreme learning machine
Hidden layer initialization
Intrinsic plasticity
Random projection
Binary features
Ternary features

ABSTRACT

In this paper, a new hidden layer construction method for Extreme Learning Machines (ELMs) is investigated, aimed at generating a diverse set of weights. The paper proposes two new ELM variants: Binary ELM, with a weight initialization scheme based on $\{0,1\}$–weights; and Ternary ELM, with a weight initialization scheme based on $\{-1,0,1\}$–weights. The motivation behind this approach is that these features will be from very different subspaces and therefore each neuron extracts more diverse information from the inputs than neurons with completely random features traditionally used in ELM. Therefore, ideally it should lead to better ELMs. Experiments show that indeed ELMs with ternary weights generally achieve lower test error. Furthermore, the experiments show that the Binary and Ternary ELMs are more robust to irrelevant and noisy variables and are in fact performing implicit variable selection. Finally, since only the weight generation scheme is adapted, the computational time of the ELM is unaffected, and the improved accuracy, added robustness and the implicit variable selection of Binary ELM and Ternary ELM come for free.

1. Introduction

The core idea of the Extreme Learning Machine (ELM) [1,2] is that it creates a single-layer feedforward neural network (SLFN) consisting of a randomly initialized hidden layer which randomly projects the inputs into a high-dimensional space. These randomly projected inputs are then transformed in a nonlinear way using some (often) nonlinear transfer function like tanh. Finally, the training of the ELM consists of solving the linear system formed by these nonlinearly transformed outputs of the hidden layer, and their corresponding target values [1,2].

The fact that the hidden layer is not touched after initialization and training consists of solving a linear system, makes the ELM very fast compared to other learning methods based on for example back-propagation or gradient-descent [1,2]. However, an aspect of the ELM that has not received much attention so far is how to exactly initialize the hidden layer. Typically, some heuristics are used and the random layer weights and biases are drawn from a uniform distribution in interval $[-5,5]$ (assuming that the data is normalized to be zero mean and unit variance) [3], or from another probability distribution like the Gaussian distribution [4]. However, heuristics like these are not necessarily optimal for any given data set and it is possible to improve the hidden layer initialization by adapting it to the problem at hand.

One approach for adapting the hidden layer to the context is the mechanism of batch intrinsic plasticity (BIP) [5–7]. The idea of BIP is that it adapts the slope and bias of the hidden layer neurons such that their outputs are approximately exponentially distributed. Given that the exponential distribution is the maximum entropy distribution, the information transmission of the neurons is maximized, resulting in a better model [8].

However, given that a transfer function typically looks like $f(\mathbf{w}^T\mathbf{x}+b)$, and $\mathbf{w}^T\mathbf{x}$, the inner product between weight vector $\mathbf{w}$ and input $\mathbf{x}$, can be rewritten as $\mathbf{w}^T\mathbf{x} = |\mathbf{w}\|\mathbf{x}| \cos\theta$, where θ is the angle between vectors $\mathbf{w}$ and $\mathbf{x}$, it can be seen that the diversity of neuronal inputs is mostly affected by the diversity of the norms of vectors $\mathbf{w}$ and $\mathbf{x}$ and their angle θ. Although BIP adapts the scaling of the input weights (and with that, the expected value of $|\mathbf{w}\|\mathbf{x}|$) such that the neuron operates in a useful regime, BIP does not optimize the weight generation scheme itself. This suggests that in order to further improve the diversity of the information extracted by the hidden layer, the diversity of the angle θ between the weight vectors and the inputs could be optimized. In this paper, this is achieved by using a binary $\{0,1\}-$weight scheme, or a ternary $\{-1,0,1\}-$weight scheme. By using a weight scheme like this, each neuron in the hidden layer focuses on a particular subspace of the variables, and the diversity of the extracted information is improved. Furthermore, the binary and ternary weight schemes allow the ELM to perform implicit variable selection, because neurons that incorporate useful variables extract more useful information and receive higher weight, while neurons that incorporate bad variables extract less useful information and are given lower weight.

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

http://dx.doi.org/10.1016/j.neucom.2014.01.072
0925-2312/© 2014 Elsevier B.V. All rights reserved.

Experiments show that especially the ternary weight scheme can generally improve the achieved test error. Furthermore, it is shown that the Binary ELM and the Ternary ELM are more robust against irrelevant and noisy variables and are in fact performing implicit variable selection. These advantages come at no increase in computational cost in comparison to drawing the random weights from e.g. a uniform or Gaussian distribution, since only the weight generation scheme is adapted.

The rest of the paper is organized as follows. Section 2 of the paper discusses the background and theory of ELM, and gives a short overview of ELM variants as well as preliminaries and methods relevant for this paper. In particular, it is discussed how to perform efficient model selection and optimization of the L2 regularization parameter in ELM, which is important for training robust models. Furthermore, BIP is discussed, since it is useful to adapt the scaling of the hidden layer weights such that the neurons operate in an optimal regime. BIP is also important because it allows us to conclude that any observed differences in performance between ELMs are due to the different weight generation scheme. Section 3 discusses the proposed binary and ternary weight schemes. Finally, Section 4 contains the experiments and analysis which form the validation for the proposed approach.

2. Preliminaries

2.1. Regression/classification

In this paper, the focus is on the problem of regression, which is about establishing a relationship between a set of output variables (continuous) $y_i \in \mathbb{R},\ 1 \le i \le M$ (single-output here) and another set of input variables $\mathbf{x}_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$. Note that although in this paper the focus is on regression, the proposed pretraining approach can just as well be used when applying the ELM in a classification context.

2.2. Extreme Learning Machine (ELM)

The ELM algorithm is proposed by Huang et al. [2] 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 [2] are reviewed.

Consider a set of N 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 M hidden neurons is modeled as the following sum

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

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 $\boldsymbol{\beta}_i$ 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

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

which can be written compactly as

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

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

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

and $\boldsymbol{\beta} = (\beta_1 \ldots \beta_M)^T$. With these notations, the theorem presented in [2] states that with randomly initialized input weights and biases for the SLFN, and under the condition that the activation function f is infinitely differentiable, then the hidden layer output matrix can be determined and will provide an approximation of the target values as good as desired (non-zero).

Algorithm 1. Standard 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 $M$ hidden nodes:
1. Randomly assign input weights $\mathbf{w}_i$ and biases $b_i$, $i \in [1, M]$.
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. Among the least-squares solutions, it is the solution with the smallest norm.
3. This smallest norm solution among the least-squares solutions is unique and is $\beta = \mathbf{H}^{\dagger}\mathbf{Y}$.

The reason why the smallest norm solution is preferred, is because smaller norm solutions tend to have better generalization performance, as discussed in [9]. 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 [2]. Furthermore, the hidden nodes need not be 'neuron-alike' [10–12].

Finally, it is recommended to have a bias in the output layer (e.g. achieved by concatenating the **H** matrix with a column of ones). Although this output bias is often not included in the description of the ELM (since theoretically it is not needed), having the output bias allows the ELM to adapt to any non-zero mean in the output at the expense of only a single extra parameter, namely the extra output weight. This way, the rest of the nonlinear weights can focus on fitting the nonlinear part of the problem. In a different context of deep learning [13], decomposing the problem into a linear part and a nonlinear part has proven to be very effective.

Given a set of candidate neurons, what remains is optimizing the ELM's other parameters like the subset of M neurons to use or the regularization parameter. Approaches for picking a subset of M neurons include model structure selection using an information criterion like BIC and cross-validation using a criterion like the leave-one-out error (described in the next section). Other approaches include methods which first generate a larger than needed set of neurons, and consequently prune this set of neurons (for example OP-ELM [3], TROP-ELM [14]), or incremental ways for determining a set of hidden layer neurons (for example I-ELM [12], CI-ELM [10], EM-ELM [11]).

An optimization mechanism that is orthogonal to optimizing the subset of neurons is that of batch intrinsic plasticity (BIP) pretraining (see Section 2.5), which is a method for optimizing the output distribution of a given neuron, such that the amount of information encoded about the inputs is maximized. Also, the proposed binary and ternary weight schemes (see Section 3) can be considered as orthogonal to optimizing the subset of neurons, since – like batch intrinsic plasticity pretraining – it takes place before the training and optimization of ELM. Therefore, both BIP and the proposed binary and ternary weight schemes can be applied as a step in many different ELM variants, and are not restricted to a particular ELM.

2.3. Efficient LOO computation and model selection

Given a set of candidate ELM models and their corresponding parameters, model selection enables one to determine the optimal ELM and parameters. The set of candidate ELMs consists of ELMs that can for example vary in terms of the number of neurons in the hidden layer (most common), or the L2 regularization parameter. Given this set of candidate ELMs, the quality of each ELM is evaluated using some criterion which estimates its generalization capabilities.

The particular criterion used in this paper for the model selection is leave-one-out (LOO) cross-validation [15]. In LOO cross-validation, given a training set of N samples, the candidate models are trained on N different training sets each of which has exactly one of the samples left out (hence the name LOO cross-validation). The left-out sample is then used for evaluating the trained model, and the final estimation of the generalization error is the mean of the N obtained squared errors (MSE). 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 [15], which is important for performing accurate model selection.

The amount of computation for LOO cross-validation might seem excessive, but for linear models one can compute the LOO error $\mathrm{MSE}^{\mathrm{PRESS}}$, without retraining the model N times, by using PRESS statistics [16]. Since ELMs are essentially linear models of the outputs of the hidden layer, the PRESS approach can be applied here as well:

$$\mathrm{MSE}^{\mathrm{PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i-\hat{y}_i}{1-hat_{ii}}\right)^2 \tag{5}$$

where y_i and $\hat{y}_i$ are respectively the ith training targets, 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 that transforms $\mathbf{Y}$ into $\hat{\mathbf{Y}}$:

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

From Eq. (6), 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, by explicitly computing $\mathbf{H}^{\dagger}$, and reusing $\mathbf{H}^{\dagger}$ to compute the LOO error $\mathrm{MSE}^{\mathrm{PRESS}}$, model structure selection of the ELM comes at very low overhead. A detailed description of this approach can be found in [17]. In summary, the algorithm for training and LOO-based model structure selection of ELM is stated in Algorithm 2.

Algorithm 2. Efficient LOO cross-validation for ELM

```
Given an ELM and a set $\mathcal{H} = \{\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_{max}\}$of H matrices
    corresponding to ELMs with e.g.
different number of hidden neurons, different regularization
    parameters, etc.
1: for all $\mathbf{H}_i \in \mathcal{H}$
2:    Train the ELM:
3:     - Calculate $\mathbf{H}_i^{\dagger}$ by solving it from $(\mathbf{H}_i^T\mathbf{H}_i)\mathbf{H}_i^{\dagger} = \mathbf{H}_i^T$;
4:     - Calculate output weights matrix $\beta = \mathbf{H}_i^{\dagger}\mathbf{Y}$;
5:    Compute $\mathrm{MSE}^{\mathrm{PRESS}}$:
6:     - Compute diag(HAT) (row-wise dot-product of $\mathbf{H}_i$ and
    $\mathbf{H}_i^{\dagger T}$);
7: ssc  - $\mathrm{MSE}^{\mathrm{PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i-\hat{y}_i}{1-hat_{ii}}\right)^2$;
8: end for
9: As model structure, select the ELM corresponding to that
   $\mathbf{H}_i \in \mathcal{H}$ which minimizes $\mathrm{MSE}^{\mathrm{PRESS}}$;
```

With regard to Algorithm 2, in case an L2-regularization parameter is optimized and the $\mathbf{H}$ matrices correspond to $\mathbf{H}$ matrices with different regularization parameters, this is Regularized ELM [18], which is referred to as TR-ELM in this paper. In the next section an efficient way is discussed for cross-validating the L2-regularization parameter of Tikhonov regularization.

2.4. Efficient Tikhonov regularization

When applying Tikhonov regularization, the pseudo-inverse used in the ELM becomes $\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H}+\lambda\mathbf{I})^{-1}\mathbf{H}^T$, for some regularization parameter λ [18]. Each value of λ results in a different pseudo-inverse $\mathbf{H}^{\dagger}$, and it would be computationally expensive to recompute the pseudo-inverse for every λ. However, by incorporating the regularization in the singular value decomposition (SVD) approach to compute the pseudo-inverse, it becomes possible to obtain the various $\mathbf{H}^{\dagger}$'s with minimal re-computation. This scheme was first used in the context of TROP-ELM in [14], and is discussed next (with some minor optimizations).

Similar to Eq. (6) in Section 2.3, with Tikhonov regularization the HAT matrix consists for a large part of the pseudo-inverse $\mathbf{H}^{\dagger}$. Only now, the pseudo-inverse is dependent on λ. Using the SVD decomposition of $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, it is possible to obtain all needed information for computing the PRESS statistic without recomputing the pseudo-inverse for every λ:

$$\begin{aligned}\hat{\mathbf{Y}} &= \mathbf{H}\beta\\ &= \mathbf{H}(\mathbf{H}^T\mathbf{H}+\lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{Y}\\ &= \mathbf{H}\mathbf{V}(\mathbf{D}^2+\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y}\\ &= \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}(\mathbf{D}^2+\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y}\\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2+\lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y}\\ &= \mathrm{HAT}\cdot\mathbf{Y}\end{aligned}$$

where $\mathbf{D}(\mathbf{D}^2+\lambda\mathbf{I})^{-1}\mathbf{D}$ is a diagonal matrix with $d_{ii}^2/(d_{ii}^2+\lambda)$ as the ith diagonal entry. From the above equations it can now be seen that given $\mathbf{U}$:

$$\begin{aligned}\mathrm{MSE}^{\mathrm{TR-PRESS}} &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i-\hat{y}_i}{1-hat_{ii}}\right)^2\\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i-\hat{y}_i}{1-\mathbf{h}_{i\cdot}(\mathbf{H}^T\mathbf{H}+\lambda\mathbf{I})^{-1}\mathbf{h}_{i\cdot}^T}\right)^2\\ &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i-\hat{y}_i}{1-\mathbf{u}_{i\cdot}\left(\frac{d_{ii}^2}{d_{ii}^2+\lambda}\right)\mathbf{u}_{i\cdot}^T}\right)^2\end{aligned}$$

where $\mathbf{h}_{i\cdot}$ and $\mathbf{u}_{i\cdot}$ are the ith row vectors of $\mathbf{H}$ and $\mathbf{U}$, respectively. Now, the Tikhonov-regularized PRESS and the corresponding λ can be computed using Algorithm 3, where $\mathbf{A}\circ\mathbf{B}$ refers to the element-wise product of matrices $\mathbf{A}$ and $\mathbf{B}$ (Schur product). Due to the convex nature of criterion $\mathrm{MSE}^{\mathrm{TR-PRESS}}$ with respect to regularization parameter λ, the Nelder–Mead procedure used for optimizing λ converges quickly in practice [19,20].

Algorithm 3. Tikhonov-regularized PRESS. In practice, the **while** part of this algorithm (convergence for λ) is solved using by a Nelder–Mead approach [19], a.k.a. downhill simplex.

```
1: Decompose H by SVD: $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^T$
2: Precompute $\mathbf{B} = \mathbf{U}^T\mathbf{y}$
3: while no convergence on $\lambda$ achieved do
```

```
4:      - Precompute $\mathbf{C} = \mathbf{U} \circ \begin{pmatrix} \frac{d_{11}^2}{d_{11}^2+\lambda} & \cdots & \frac{d_{11}^2}{d_{11}^2+\lambda} \\ \vdots & \ddots & \vdots \\ \frac{d_{NN}^2}{d_{NN}^2+\lambda} & \cdots & \frac{d_{NN}^2}{d_{NN}^2+\lambda} \end{pmatrix}$
5:      - Compute $\hat{\mathbf{y}} = \mathbf{CB}$, the vector containing all $\hat{y}_i$
6:      - Compute $\mathbf{d} = \mathrm{diag}(\mathbf{CU}^T)$, the diagonal of the HAT matrix,
          by taking the row-wise dot-product of $\mathbf{C}$ and $\mathbf{U}$
7:      - Compute $\boldsymbol{\varepsilon} = \frac{\mathbf{y}-\hat{\mathbf{y}}}{\mathbf{1}-\mathbf{d}}$, the leave-one-out errors
8:      - Compute $\mathrm{MSE}^{\mathrm{TR-PRESS}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\varepsilon}_i^2$
9:  end while
10: Keep the best $\mathrm{MSE}^{\mathrm{TR-PRESS}}$ and the associated $\lambda$ value
```

2.5. *Intrinsic plasticity*

2.5.1. *Background*

The mechanism of intrinsic plasticity is one that is orthogonal to the earlier mentioned approaches. Namely, it generally takes place right after generating the random weights of the neurons, and its result is subsequently used in the further optimization, pruning and training of the ELM. As such, intrinsic plasticity can be used in combination with most other ELM approaches.

The concept of intrinsic plasticity has a biological background and refers to the fact that neurons adapt in such a way that they maximize their entropy (and thus the amount of information transmitted), while keeping the mean firing rate low. Intrinsic plasticity originally appeared in the neural networks literature in the context of reservoir computing, recurrent neural networks, liquid state machines and echo state networks [21–23], where it is used as a method to construct a priori suitable networks that are guaranteed or likely to offer a certain performance [23]. Like its biological equivalent, the goal of intrinsic plasticity in the context of neural networks is to maximize the information transmission of the neurons in the neural networks. The way it achieves this, is by shaping the neuron outputs such that they approximately follow an exponential distribution, which is the maximum entropy distribution among all positive distributions with fixed mean [8].

2.5.2. *Batch Intrinsic Plasticity (BIP) ELM*

In [5,7,6,24] the principle of intrinsic plasticity is transferred to ELMs and introduced as an efficient pretraining method, aimed at adapting the hidden layer weights and biases, such that the output distribution of the hidden layer is shaped like an exponential distribution. The only parameter of batch intrinsic plasticity is the mean μ_{exp} of the target exponential distribution.

Given the inputs $(\mathbf{x}_1, \ldots, \mathbf{x}_N) \in \mathbb{R}^{N \times d}$ and input matrix $\mathbf{W}^{in} \in \mathbb{R}^{d \times M}$ (with N being the number of samples in the training set, d the dimensionality of the data, and M the number of neurons), the synaptic input to neuron i is given by $s_i(k) = \mathbf{x}_k \mathbf{W}^{in}_{.i}$. Now, it is possible to adapt slope a_i and bias b_i, such that the desired output distribution is achieved for neuron output $h_i = f(a_i s_i(k) + b_i)$. To this end, for each neuron random targets $\mathbf{t} = (t_1, t_2, \ldots, t_N)$ are drawn from the exponential distribution with mean μ_{exp}, and sorted such that $t_1 < \cdots < t_N$. The synaptic inputs to neuron are sorted as well into vector $\mathbf{s}_i = (s_i(1), s_i(2), \ldots, s_i(N))$, such that $s_i(1) < s_i(2) < \cdots < s_i(N)$.

Now, in the case of an invertible transfer function, the targets can be propagated back through the hidden layer, and a linear model can be defined that maps the sorted $s_i(k)$ as closely as possible to the sorted $t_i(k)$. To this end, a model $\Phi(\mathbf{s}_i) = (\mathbf{s}_i^T, (1 \ldots 1)^T)$ and parameter vector $\mathbf{v}_i = (a_i, b_i)^T$ are defined. Then, given the invertible transfer function f the optimal slope a_i and bias b_i for which each $s_i(k)$ is approximately mapped to t_k can be found by minimizing

$$\| \Phi(\mathbf{s}_i) \cdot \mathbf{v}_i - f^{-1}(\mathbf{t}) \|$$

The optimal slope a_i and bias b_i can therefore, like in ELM, be determined using the Moore–Penrose pseudo-inverse:

$$\mathbf{v}_i = (a_i, b_i)^T = \Phi^{\dagger}(\mathbf{s}_i) \cdot f^{-1}(\mathbf{t})$$

This procedure is performed for every neuron with an invertible transfer function, and even though the target distribution cannot exactly be matched (due to the limited degrees of freedom in the optimization problem) it has been shown in [5,7] that batch intrinsic plasticity is an effective and efficient scheme for input-specific tuning of input weights and biases used in the non-linear transfer functions. Effectively, it makes the ELM insensitive to the original scaling of the inputs and generated weights, and automatically adapts the transfer function such that it operates in a meaningful regime.

There are several approaches to setting parameter μ_{exp} of the exponential target distribution from which targets $\mathbf{t}$ are drawn:

- setting μ_{exp} randomly in the interval $[0, 1]$ per-neuron [5,7]
- setting μ_{exp} to a specific value for all neurons [5,7]
- cross-validating μ_{exp} such that it adapts to the current context

In this paper, the first variant (denoted BIP(rand)-ELM) is used since it offers an attractive balance between computational time and accuracy.

As mentioned before, BIP is an optimization mechanism that is orthogonal to most other approaches to optimizing an ELM, and can therefore be incorporated in many existing ELM schemes like for example the ELM trained with L2 regularization. A nice advantage of this combination is that the stability of BIP-ELM combined with L2 regularization essentially removes the need to tune the amount of hidden neurons. Therefore, this is the approach used in the experiments, combined with weights generated from a Gaussian distribution, or generated using the proposed binary or ternary weight scheme that is discussed in the next section.

3. Binary/ternary ELM

3.1. *Motivation*

The main idea behind the ELM is the fact that theoretically it is sufficient to generate the hidden layer weights in a random way. As long as the weights are independent and the transfer functions are infinitely differentiable [1,2] the ELM is a universal approximator and can approximate any function, given enough data and given enough neurons. This is however, an asymptotic result, and in practice there is only limited data available. Therefore, proper care needs to be taken that the ELM does not overfit by e.g. using L1 regularization [3], L2 regularization [18], or L1 and L2 regularization [14] on the hidden layer.

Furthermore, as already discussed in the previous section, the biases and hidden layer weights can be adapted using batch intrinsic plasticity (BIP) pretraining such that they extract as much information as possible from the inputs. However, apart from scaling the weights and picking proper biases, BIP does not optimize the direction of the weights themselves. It is still perfectly possible that there exist redundancies between the neurons, and that although each neuron extracts the maximum amount of information, between neurons this information may be very similar. Therefore, the ELM may still benefit from a weight picking scheme that ensures the diversity of the extracted information.

To this end, in this paper a binary $\{0, 1\}-$weight and a ternary $\{-1, 0, 1\}-$weight scheme are proposed. The motivation for these schemes is that weights picked in this way lie in very different subspaces, and therefore result in diverse inputs to the hidden layer.

Another way to look at binary and ternary weight schemes is by looking at the weights in terms of their direction. Weights generated in this way point in very different directions, which is advantageous for the diversity of the hidden layer inputs (and thus the extracted information). This can be seen by rewriting the $\mathbf{w}^T\mathbf{x}$ that is part of $f(\mathbf{w}^T\mathbf{x}+b)$ as $\mathbf{w}^T\mathbf{x} = |\mathbf{w}||\mathbf{x}| \cos\theta$, where θ is the angle between vectors **w** and **x**. Now, if the weights **w** have diverse directions, the angles θ between **w** and **x** – and therefore the hidden layer inputs – will be diverse as well. Furthermore, combined with BIP pretraining (which essentially makes the ELM insensitive to the initial $|\mathbf{w}|$ and $|\mathbf{x}|$), any difference between ELMs that only differ in the way they generate the weights, will have to come from the diversity of $\cos\theta$, and hence the angles between the weights and the input data.

Finally, since the only thing that is adapted in the binary and ternary weight scheme is the way to generate the random weights, the computational time of the ELM is not affected compared to the traditional random weights, and any advantages that may result from the different weight generation scheme will come for free.

3.2. *Binary scheme*

In this scheme, the M d-dimensional weights of the hidden layer are binary $\{0,1\}^d$–weights, generated starting from the sparsest weight vectors. This way, first all $\binom{d}{1}$ subsets of 1 variable are included, then all $\binom{d}{2}$ subsets of 2 variables, etc., until there are M hidden neurons generated and each neuron employs a different subset of input variables. The weights are normalized to have unit length, such that the expected value of $\mathbf{w}^T\mathbf{x}$ is approximately the same for every neuron, regardless of the number of non-zero weights. Otherwise, the number of non-zero weights would strongly affect what part of the transfer function is activated. The procedure is summarized in Algorithm 4. In case $M > 2^d$ (the number of possible binary weights), also randomly rotated versions of the binary weights are added. However, this rarely happens since the number of possible weights grows exponentially with d.

Algorithm 4. Binary weight scheme, with M being the desired number of hidden neurons, n the dimension of the subspaces in which to generate weights, and d the number of inputs.

```
1: Generate ELM:
2: n=1;
3: while numneurons ≤ M and n ≤ d do
4:     - Generate the $\binom{d}{n}$ possible assignments of $n$ ones to $d$
       positions
5:     - Shuffle the order of the generated weights to avoid bias
       to certain inputs due to the scheme used to generate the $\binom{d}{n}$
       assignments
6:     - Add the generated weights (up to a maximum of $M$
       neurons)
7: - n=n+1;
8: end while
9: - Normalize the norm of the weights, such that they are unit
   length.
```

3.3. *Ternary scheme*

Whereas the binary weight scheme takes its motivation mostly from an increase in the diversity of extracted information by having each neuron use a different subset of variables, the ternary weight scheme is more geometrically motivated: the procedure is the same as for the binary weights, except that each position can be a -1 or a 1, and therefore each weight can be seen as pointing towards a corner of the hypercube. By including -1 as a possible weight, the number of possible directions of the weights is increased, while retaining the advantage of the neurons operating on a different subset of input variables.

In summary, in the ternary scheme, the M d-dimensional weights of the hidden layer are ternary $\{-1,0,1\}^d$–weights, generated starting from the sparsest weight vectors. This way, first all $2^1 \times \binom{d}{1}$ weights of 1 variable are included, then all $2^2 \times \binom{d}{2}$ weights of 2 variables, etc., until there are M hidden neurons generated. In case $M > 3^d$ (the number of possible ternary weights), also randomly rotated versions of the ternary weights are added. This rarely happens though, since the number of possible weights grows exponentially with d.

Fig. 1 illustrates the possible weights in both the binary and ternary weight scheme within the 2D subspace constituted by Abalone variables 2 and 4. Note that this is just one of the many 2D subspaces, and the weights of Binary ELM and Ternary ELM are

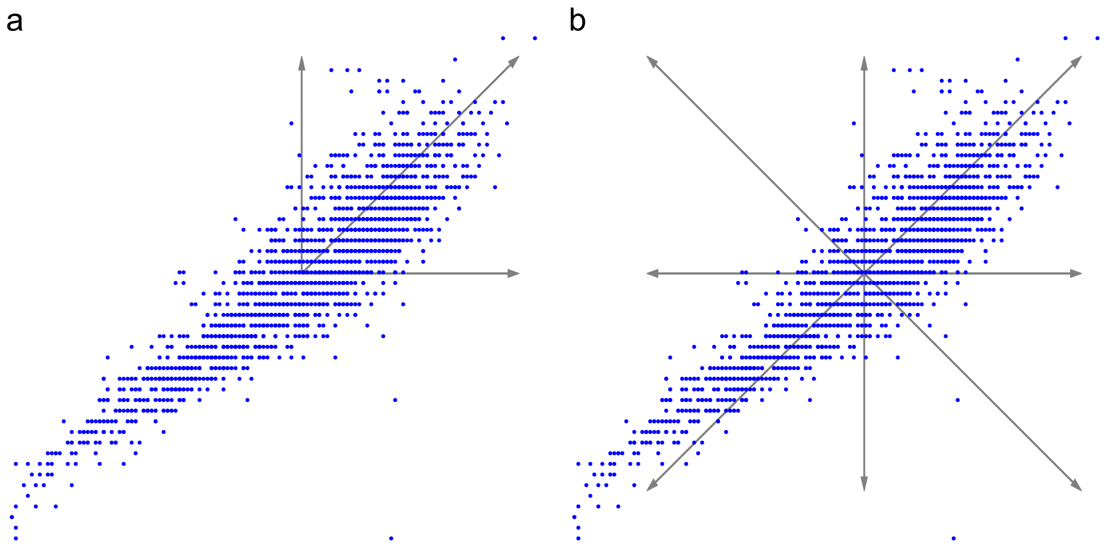


Fig. 1. Illustration of possible weights (*arrows*) for binary (a) and ternary (b) weight scheme, in a 2D subspace of normalized Abalone data (*blue dots*). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

drawn randomly from subspaces of increasingly higher dimension in the way described above, until the desired number of neurons is reached. Once the weights have been drawn, they may be normalized in some fashion, or as is done in this paper, scaled using BIP pretraining.

3.4. *Motivation for BIP pretraining*

Since for given weight $\mathbf{w}$ and input $\mathbf{x}$, the expected value of $|\mathbf{w}||\mathbf{x}|$ determines in which part of the transfer function is activated most, the norm of the weights is important and affects the performance of ELM. Of course, the weights could be normalized to be e.g. unit length, but the question remains what is the optimal length for the given data. Therefore, to ensure that the weights are properly scaled, the ELMs are pretrained using Batch Intrinsic Plasticity (BIP) pretraining. In particular, the BIP(rand) variant [5,7] is used, since it offers an attractive balance between computational time and accuracy. An added advantage of using BIP pretraining is that when comparing ELMs with varying weight schemes, any differences in performance must come from the differences in the direction of the weights and are not a result of the different scaling of the weights.

Since BIP pretraining adapts the neurons to operate in their non-linear regime, as many linear neurons are included as there are input variables. This ensures good performance of the ELM, even if the problem is completely linear.

3.5. *Motivation for Tikhonov regularization*

With limited data, the capability of ELM to overfit the data increases with increasing number of neurons, especially if those neurons are optimized to be well-suited for the function approximation problem. Therefore, to avoid overfitting, Tikhonov regularization with optimized regularization parameter as explained in Section 2.4, is used.

Table 1
Summary of the properties of the UCI data sets [25] used.

Task	Abbreviation	Number of variables	# Training	# Test
Abalone	Ab	8	2000	2177
CaliforniaHousing	Ca	8	8000	12,640
CensusHouse8L	Ce	8	10,000	12,784
DeltaElevators	De	6	4000	5517
ComputerActivity	Co	12	4000	4192

4. Experiments

This section describes the experiments that investigate the effectiveness of the Binary and Ternary weight scheme compared to the traditional random weights:

- the first experiment compares the average performances of each weight scheme on several UCI data sets.

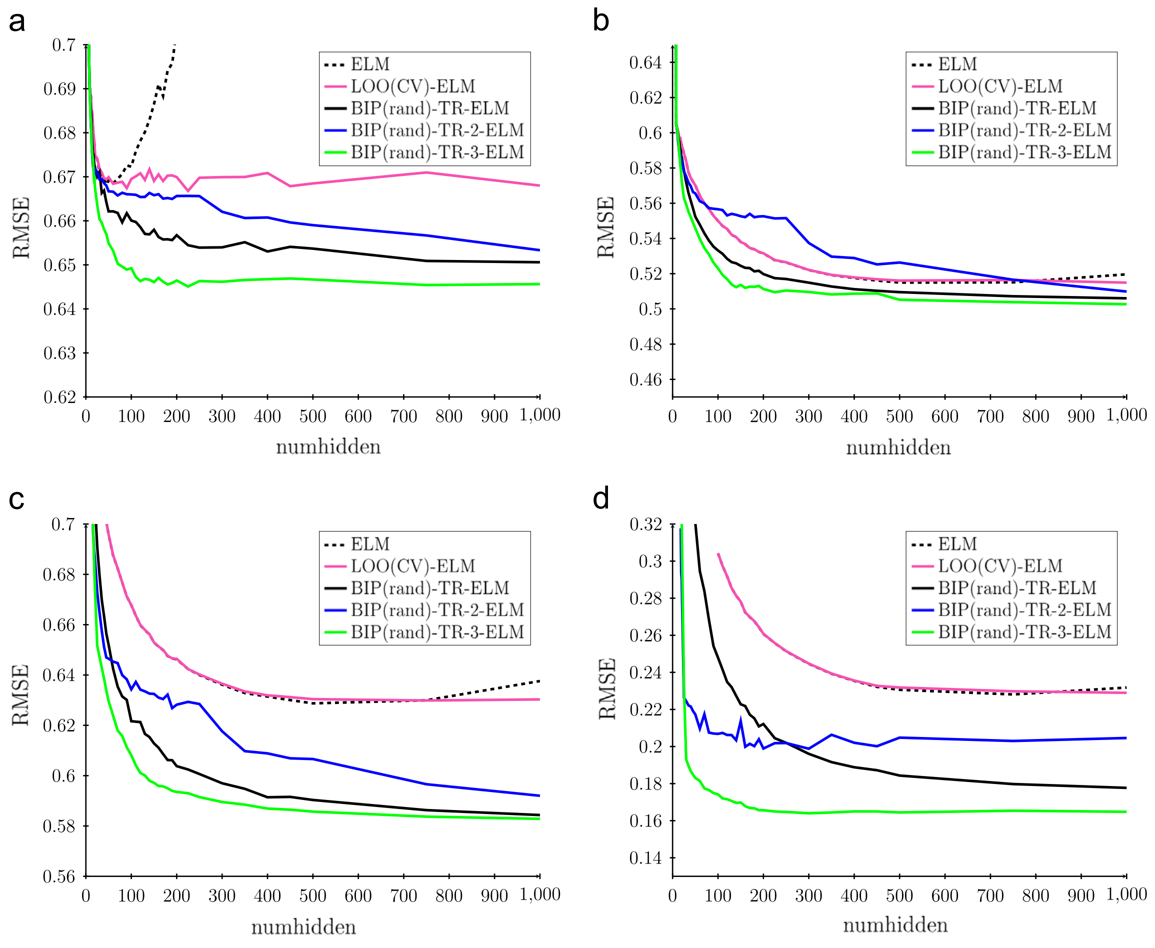


Fig. 2. Number of neurons vs. average achieved test RMSE for ELM (*black, dashed*), LOO(CV)-ELM (*purple*), BIP(rand)-TR-ELM with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme. (a) Abalone. (b) CaliforniaHousing. (c) CensusHouse8L. (d) ComputerActivity. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

- the second experiment compares the robustness of the various weight schemes to irrelevant and noisy input variables and investigates whether the weight schemes are performing implicit variable selection.

4.1. Data and preprocessing

As data sets, 5 different regression tasks from the UCI machine learning repository [25] are used, with the division of the data in training set and test set chosen in the same way as in [7], but drawn in a random way (without repetition) for each run of the experiment in order to control for the influence of the particular realization of the training and test set on the results. The specification of the data can be found in Table 1.

The data is preprocessed in such a way that each input and output variable is zero mean and unit variance. Note that this way of preprocessing makes it impossible to directly compare with papers that use a different way of preprocessing like rescaling variables to a specific interval. Results with different normalization, as well as denormalized versions of the RMSEs, can be found in Appendix A.

4.2. Average performances of each weight scheme

In this experiment, the average performances are compared for Binary ELMs, Ternary ELMs and ELMs with weights drawn from a Gaussian prior. As explained in Section 3, batch intrinsic plasticity pretraining with randomized μ_{exp} (BIP(rand)) is used to adapt the scaling of the weights to the current context. This controls for performance differences due to the scaling of the weights and

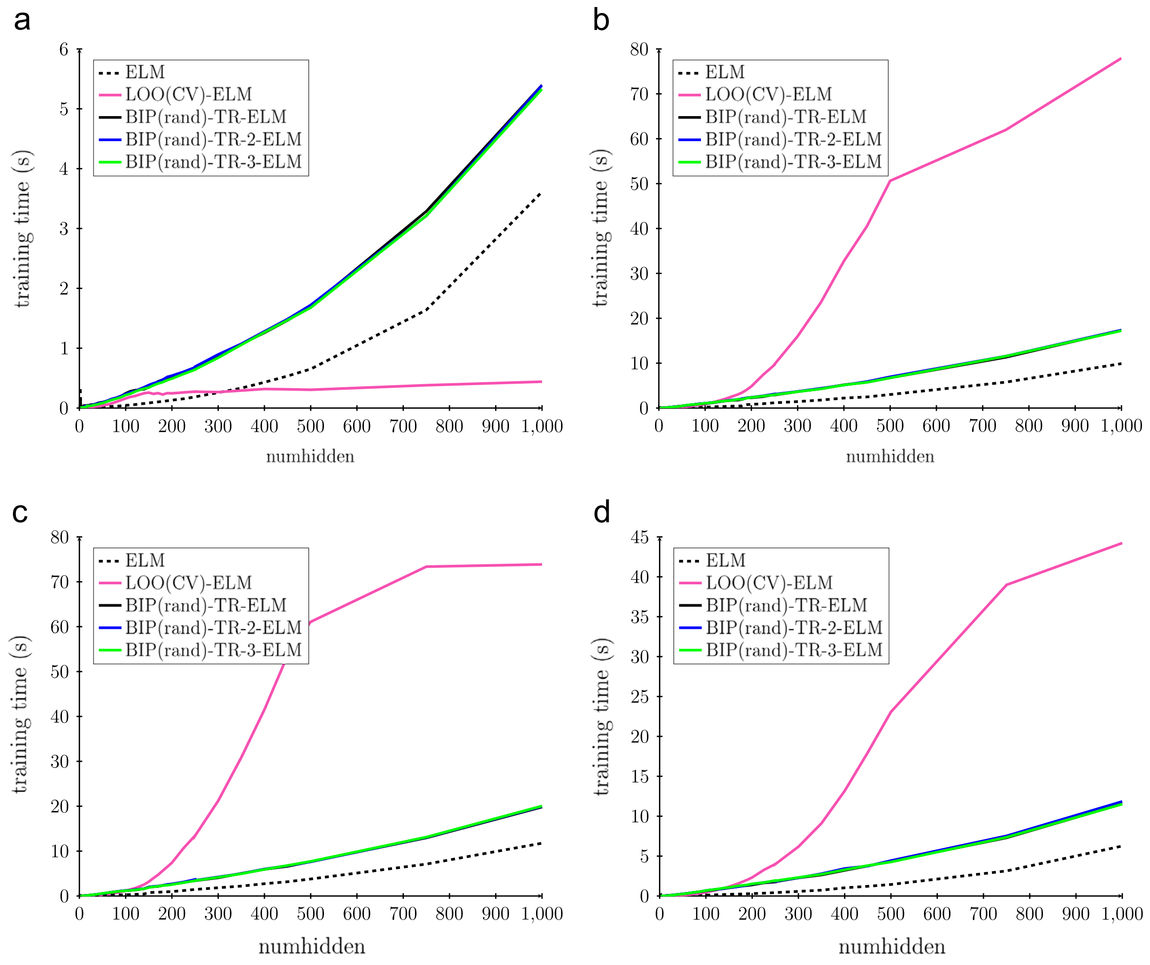


Fig. 3. Number of neurons vs. average training time for ELM (*black, dashed*), LOO(CV)-ELM (*purple*), BIP(rand)-TR-ELM with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme. (a) Abalone. (b) CaliforniaHousing. (c) CensusHouse8L. (d) ComputerActivity. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

Table 2
Average test RMSE achieved over 100 random divisions in training and test set, for ELMs with 1000 hidden neurons (data normalized to be zero mean and unit standard deviation).

Model	**Ab**	**De**	**Co**	**Ce**	**Ca**
ELM	1.694 ± 0.4185	0.708 ± 0.0115	0.232 ± 0.0088	0.638 ± 0.0171	0.520 ± 0.0070
LOO(CV)-ELM	0.668 ± 0.0203	0.605 ± 0.0095	0.229 ± 0.0076	0.630 ± 0.0179	0.515 ± 0.0069
BIP(rand)-TR-ELM	0.651 ± 0.0184	0.602 ± 0.0096	0.178 ± 0.0070	0.584 ± 0.0177	0.506 ± 0.0098
BIP(rand)-TR-2-ELM	0.653 ± 0.0202	0.602 ± 0.0104	0.205 ± 0.0360	0.592 ± 0.0187	0.510 ± 0.0090
BIP(rand)-TR-3-ELM	0.646 ± 0.0202	0.602 ± 0.0093	0.165 ± 0.0050	0.583 ± 0.0170	0.503 ± 0.0129

ensures that any differences in performance are actually due to the weight scheme used. Furthermore, since the better the neurons are the easier it will be to overfit, the ELMs also use Tikhonov regularization (TR) as described in Section 2.4.

To illustrate the advantages and trade-offs made in the proposed models, the models are also compared to standard ELM. Finally, since the basic ELM suffers from overfitting in case the number of neurons is large compared to the number of samples, another basic ELM variant is included which includes cross-validation of the number of neurons according to the LOO error. In this cross-validation procedure, the number of neurons is increased in steps of 10 (up to the currently tested number of hidden neurons), and an early stopping criterion is used, such that the optimization stops if there was no decrease in LOO error for 5 consecutive steps (i.e. 50 neurons).

Therefore, in summary, the ELMs tested are

- ELM
- LOO(CV)-ELM
- BIP(rand)-TR-ELM
- BIP(rand)-TR-Binary-ELM
- BIP(rand)-TR-Ternary-ELM

These ELMs have

- a trained output bias (achieved by adding a column of ones to the **H** matrix);
- as many linear neurons as inputs;
- Fermi neurons

to approximate respectively the constant, linear, and nonlinear component of the function. The number of hidden neurons is varied up to 1000 hidden neurons, and the ELMs are tested on 100 random partitions of each data set into training and test set (samples drawn without repetition).

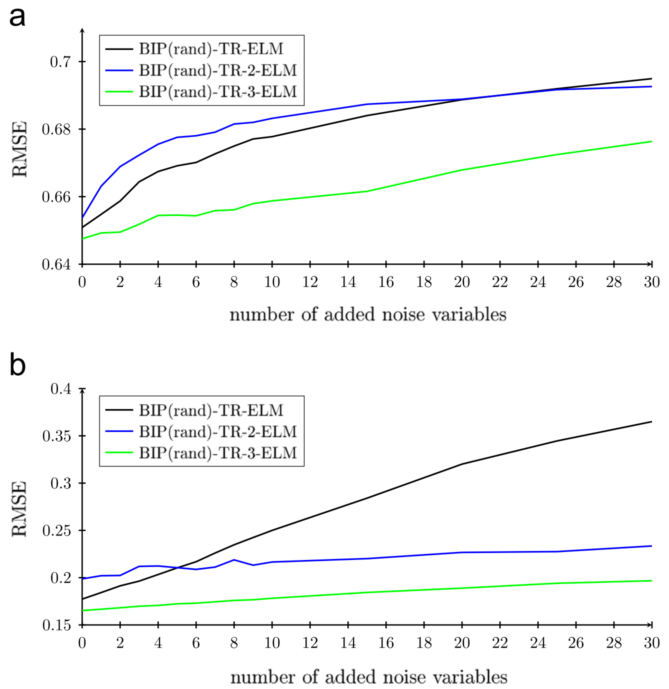


Fig. 4. Effect of adding irrelevant extra variables on RMSE for BIP(rand)-TR-ELM with 1000 hidden neurons and with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme. (a) Abalone. (b) ComputerActivity. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

4.2.1. Average RMSE

In Fig. 2 the average achieved RMSE on the test set is reported for the varying number of hidden neurons. As expected, for increasing number of neurons, the standard ELM starts to overfit at some point, resulting in an increase in the RMSE on the test set. Performing the LOO cross-validation to limit the number of used hidden neurons prevents this overfitting. Furthermore, the proposed methods generally achieve much better RMSE than the basic ELM variants. Finally, it can be seen that generally, ternary weights outperform weights drawn from a Gaussian distribution, and binary weights generally perform worse than ternary and Gaussian weights.

One possible hypothesis for why the binary weights perform worse than the ternary weights is that the binary weights result in less diverse activation of the hidden neurons with transfer function $f(\mathbf{w}^T\mathbf{x}+b)=f(|\mathbf{w}\|\mathbf{x}|\cos\theta+b)$. Indeed, considering Fig. 1, there are only 3 possible binary weights within a particular 2D subspace, covering $\pi/2$ radians of the circle (compared to 8 possible ternary weights, covering all 2π radians of the circle). Therefore, for a fixed sample $\mathbf{x}$, the binary weight scheme can potentially produce 3 different values of θ (and thus $\cos\theta$) that are $\pi/4$ radians apart, whereas the ternary weight scheme can potentially produce 8 different values that are $\pi/4$ radians apart. After removing symmetries (since $\cos(\theta+\pi)=-\cos\theta$), this leaves 4 different values for $\cos\theta$ that would add different information to the hidden layer, compared to the 3 different values of the binary weight scheme, which might give the ternary weight scheme its advantage. A further theoretical analysis of the relative performance of the binary and ternary weight scheme will be the subject of a future paper.

4.2.2. Average training time

The average training time for each model can be found in Fig. 3. It is interesting to see that the computational time of the LOO(CV)-ELM strongly depends on the used data set. For Abalone, the cross-validation procedure finds an optimal number of hidden neurons of about 50, after which the leave-one-out error quickly increases and further optimization is quickly halted due to the stopping criterion. Hence, the computational time remains low. However, for the other data sets, the number of optimal hidden neurons is much higher, and the cross-validation procedure becomes tedious. Furthermore, given the fact that it is possible to perform both BIP pretraining and optimization of the Tikhonov regularization parameter in less time than it takes to train a basic ELM (i.e. the computational time is not even doubled), cross-validation of the number of neurons becomes very unattractive.

The relatively low overhead on the computational time compared to the basic ELM, and the decreasing nature of the curves in

Table 3
Average RMSE loss of ELMs with 1000 hidden neurons, trained on the original data, and the data with 30 added irrelevant variables.

RMSE	Ab			Co		
	Gaussian	Binary	Ternary	Gaussian	Binary	Ternary
RMSE with original variables	0.6509	0.6537	0.6475	0.1773	0.1987	0.1653
RMSE with 30 added irr. vars	0.6949	0.6926	0.6763	0.3650	0.2335	0.1967
RMSE loss	0.0441	**0.0389**	**0.0288**	0.1877	**0.0348**	**0.0315**

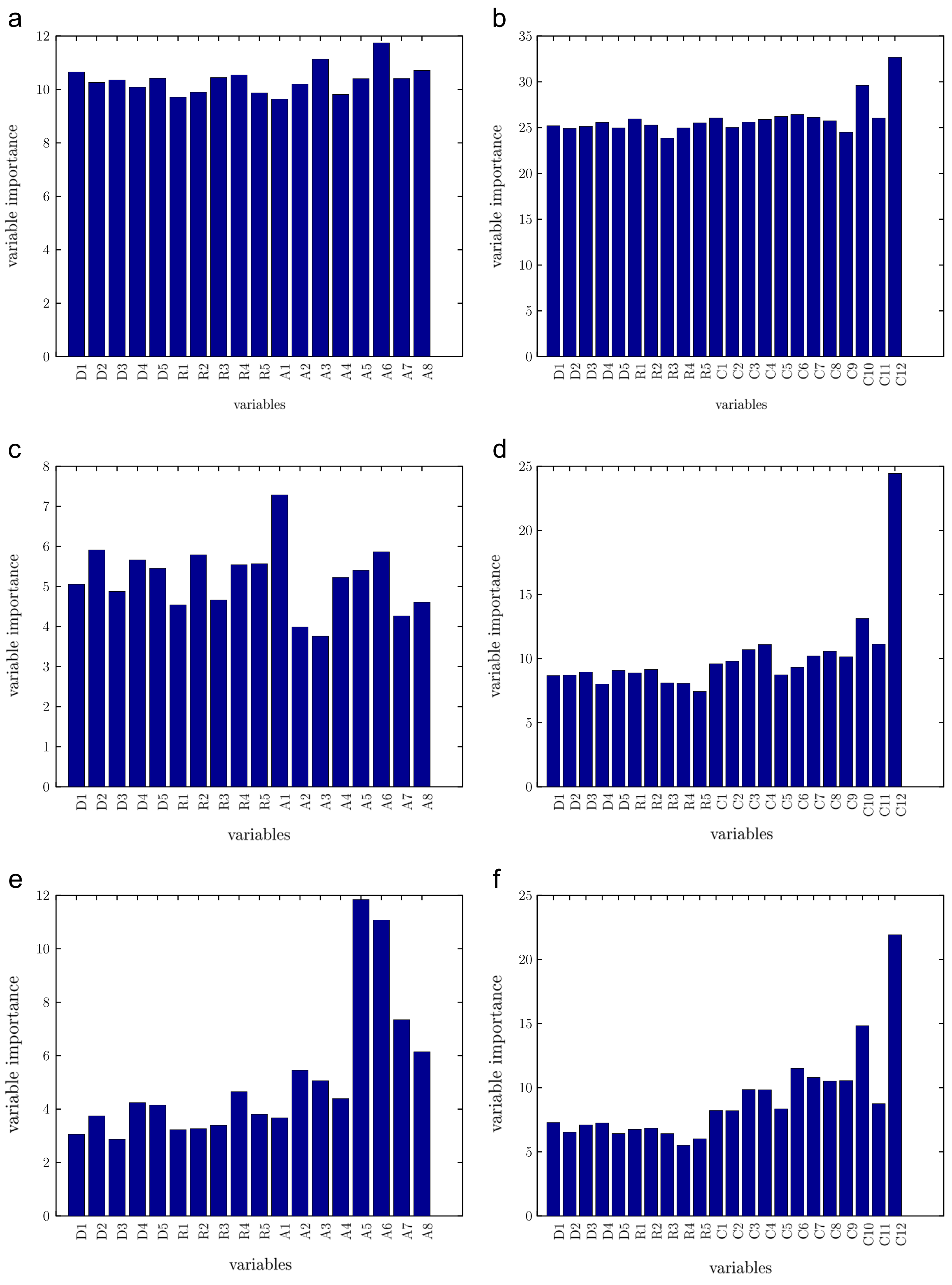


Fig. 5. Variable relevance measure for Abalone and ComputerActivity with 5 random noise variables R1, …, R5, 5 irrelevant variables D1, …, D5, and the original variables. (a) Abalone, Gaussian. (b) Computer Activity, Gaussian (c) Abalone, binary. (d) Computer Activity, binary. (e) Abalone, ternary. (f) Computer Activity, ternary

Fig. 2 therefore suggests that a robust and fast way to build a good ELM is to use L2 regularization and a large number of neurons. Table 2 summarizes the performance for the various weight schemes for ELMs with 1000 neurons (i.e. the most right points in the figures). Although for larger number of neurons the differences in terms of RMSE are smaller, the advantages of ternary weights over Gaussian weights are still present. Furthermore, the results show that the standard deviations of the RMSEs for the ternary weight scheme are consistently lower or equal than those for the Gaussian weight scheme.

4.3. *Effect of irrelevant variables*

In this experiment, the robustness against added irrelevant variables is evaluated, as well as a criterion showing that the binary and ternary ELMs are performing implicit variable selection.

4.3.1. *Robustness against added noise variables*

Both the binary and ternary weight schemes result in neurons operating on a diverse collection of subsets of the input variables. However, since these subsets might also include irrelevant variables in this experiment the robustness against irrelevant variables is tested. The various weight schemes are evaluated on the Abalone and ComputerActivity data with up to 30 irrelevant Gaussian noise variables added.

The results are summarized in Fig. 4 and Table 3. These results are again the averages over 100 random partitions of the data in training and test set. It can be seen that the ternary and binary weight schemes are more robust against irrelevant variables. The difference is especially large for the ComputerActivity data set.

4.3.2. *Implicit variable selection*

Considering the fact that the weights are sparse in the input variables, each neuron is in fact only extracting information from a certain subset of variables. Therefore, given a trained ELM, the output weights could be considered as an indication of how important or useful a specific neuron and variable subset is for the function approximation. In this experiment, the relevance of each input variable in the ELMs is quantified as

$$\textbf{relevance} = \sum_{i=1}^{M} |\beta_i \times \mathbf{w}_i|,$$

where M is the number of hidden neurons; β_{i} is the output weight; $\mathbf{w}_i$ is the input weight corresponding to neuron i, and **relevance** is the d-dimensional vector containing a measure of relevance for each of the d input variables. If a variable j has a large value of $relevance_j$, compared to other variables, this can be interpreted as that variable being implicitly selected by the ELM (i.e. the ELM favors neurons that extract information from that variable).

To test whether the ELMs perform implicit variable selection, the ELMs are trained on the Abalone and ComputerActivity data sets, where 5 irrelevant variables (taken from the DeltaElevators data) and 5 Gaussian noise variables have been added. The results for this experiment on Abalone and ComputerActivity are summarized in Fig. 5. There, it can be seen that for the Gaussian weights, the *relevance* measure indicates that the ELM does not favor any neurons that employ a particular input variable. However, for the Binary and Ternary ELM, the *relevance* measure clearly shows that the ELMs favor neurons that employ specific input variables. For example, the 12th input variable in ComputerActivity seems especially preferred. Finally, the *relevance* measure indicates that the irrelevant and noise variables are not given particularly high importance in general.

5. Conclusion

In this paper, Binary ELM and Ternary ELM have been described, which employ a weight initialization scheme based on $\{0,1\}$−weights and $\{-1,0,1\}$−weights respectively. The motivation behind these schemes is that weights picked in this way will be from very different subspaces, and therefore improve the diversity of the neurons in the hidden layer. Experiments show that Ternary ELM generally achieves lower test error. Furthermore, the experiments suggest that the binary and ternary weight schemes improve robustness against irrelevant variables and that the binary and ternary weight schemes perform implicit variable selection. Finally, since only the weight generation scheme is changed, the computational time of ELM remains unchanged compared to ELMs with traditional random weights. Therefore, the better performance, added robustness and implicit variable selection in Binary ELM and Ternary ELM come for free.

Appendix A

For comparison, RMSEs are included for another commonly-used normalization scheme (minmax), where input variables are rescaled to interval [−1,1], and output variables are rescaled to interval [0,1]. Denormalized versions of the RMSEs for both normalization schemes are included as well, which makes the RMSEs of both normalization schemes comparable and allows for evaluating the effect of the normalization on RMSE. All errors are avg. RMSEs achieved over 100 random divisions in training and test set, for ELMs with 1000 neurons.

A.1. *RMSEs for minmax normalization*

	Ab	**De**	**Co**	**Ce**	**Ca**
ELM	5.246 ± 4.3149	0.087 ± 0.0086	5.29e3 ± 3.60e4	4.433 ± 6.9604	2.764 ± 2.3442
LOO(CV)-ELM	0.086 ± 0.0089	0.059 ± 0.0056	1.123 ± 8.1590	0.102 ± 0.0890	0.132 ± 0.0066
BIP(rand)-TR-ELM	0.083 ± 0.0072	0.060 ± 0.0056	0.032 ± 0.0021	0.063 ± 0.0011	0.121 ± 0.0022
BIP(rand)-TR-2-ELM	0.083 ± 0.0074	0.060 ± 0.0056	0.036 ± 0.0042	0.064 ± 0.0014	0.123 ± 0.0022
BIP(rand)-TR-3-ELM	0.082 ± 0.0073	0.060 ± 0.0056	0.031 ± 0.0006	0.063 ± 0.0012	0.119 ± 0.0020

A.2. *Denormalized RMSEs for minmax normalization*

	Ab	**De**	**Co**	**Ce**	**Ca**
ELM	1.33e2 ± 1.02e2	2.11e−3 ± 9.54e−5	5.24e5 ± 3.57e6	2.22e6 ± 3.48e6	1.34e6 ± 1.13e6
LOO(CV)-ELM	2.20 ± 1.10e−1	1.44e−3 ± 1.38e−5	1.11e2 ± 8.08e2	5.10e4 ± 4.45e4	6.38e4 ± 3.16e3
BIP(rand)-TR-ELM	2.11 ± 3.60e−2	1.45e−3 ± 1.30e−5	3.18 ± 2.10e−1	3.16e4 ± 5.27e2	5.85e4 ± 1.03e3
BIP(rand)-TR-2-ELM	2.12 ± 5.05e−2	1.45e−3 ± 1.33e−5	3.58 ± 4.21e−1	3.19e4 ± 6.81e2	5.94e4 ± 9.98e2
BIP(rand)-TR-3-ELM	2.09 ± 3.86e−2	1.45e−3 ± 1.36e−5	3.06 ± 5.74e−2	3.14e4 ± 5.94e2	5.78e4 ± 9.15e2

A.3. Denormalized RMSEs for zero mean, unit variance normalization

	Ab	De	Co	Ce	Ca
ELM	5.46 ± 1.37	$1.68e-3 \pm 2.13e-5$	$4.25 \pm 1.24e-1$	$3.38e4 \pm 4.23e2$	$5.99e4 \pm 5.68e2$
LOO(CV)-ELM	$2.15 \pm 4.36e-2$	$1.44e-3 \pm 1.10e-5$	$4.20 \pm 1.09e-1$	$3.34e4 \pm 4.50e2$	$5.94e4 \pm 5.40e2$
BIP(rand)-TR-ELM	$2.10 \pm 3.08e-2$	$1.43e-3 \pm 1.18e-5$	$3.26 \pm 1.00e-1$	$3.10e4 \pm 5.26e2$	$5.83e4 \pm 1.01e3$
BIP(rand)-TR-2-ELM	$2.11 \pm 3.87e-2$	$1.43e-3 \pm 1.38e-5$	$3.75 \pm 6.44e-1$	$3.14e4 \pm 6.10e2$	$5.88e4 \pm 8.52e2$
BIP(rand)-TR-3-ELM	$2.08 \pm 3.88e-2$	$1.43e-3 \pm 1.07e-5$	$3.02 \pm 5.08e-2$	$3.09e4 \pm 4.55e2$	$5.80e4 \pm 1.30e3$

References

[1] 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, vol. 2, 2004, pp. 985–990. http://dx.doi.org/10.1109/IJCNN.2004.1380068.

[2] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501. http://dx.doi.org/10.1016/j.neucom.2005.12.126.

[3] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, OP-ELM: optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162.

[4] E. Parviainen, J. Riihimäki, Y. Miche, A. Lendasse, Interpreting extreme learning machine as an approximation to an infinite neural network, in: KDIR 2010—Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, 2010, pp. 65–73.

[5] K. Neumann, J.J. Steil, Batch intrinsic plasticity for extreme learning machines, in: Artificial Neural Networks and Machine Learning—ICANN 2011, vol. 6791, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2011, pp. 339–346. http://dx.doi.org/10.1007/978-3-642-21735-7_42.

[6] K. Neumann, C. Emmerich, J.J. Steil, Regularization by intrinsic plasticity and its synergies with recurrence for random projection methods, J. Intell. Learn. Syst. Appl. 4 (3) (2012) 230–246.

[7] K. Neumann, J.J. Steil, Optimizing extreme learning machines via ridge regression and batch intrinsic plasticity, Neurocomputing 102 (2013) 555–560 ⟨http://dx.doi.org/10.1016/j.neucom.2012.01.041.

[8] J.J. Steil, Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning, Neural Netw.: Off. J. Int. Neural Netw, Soc. 20 (3) (2007) 353–364.

[9] P.L. Bartlett, The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, IEEE Trans. Inf. Theory 44 (2) (1998) 525–536. http://dx.doi.org/10.1109/18.661502.

[10] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16–18) (2007) 3056–3062. http://dx.doi.org/10.1016/j.neucom.2007.10.008.

[11] G. Feng, G.-B. Huang, Q. Lin, R. Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, IEEE Trans. Neural Netw. 20 (8) (2009) 1352–1357. http://dx.doi.org/10.1109/TNN.2009.2024147.

[12] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892. http://dx.doi.org/10.1109/TNN.2006.875977.

[13] T. Raiko, H. Valpola, Deep learning made easier by linear transformations in perceptrons, in: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics, 2012, pp. 924–932.

[14] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, A. Lendasse, TROP-ELM: a double-regularized ELM using LARS and Tikhonov regularization, Neurocomputing 74 (16) (2011) 2413–2421. http://dx.doi.org/10.1016/j.neucom.2010.12.042.

[15] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, (2006). http://www.springer.com/computer/image+processing/book/978-0-387-31073-2, http://books.google.fi/books?id=kTNoQgAACAAJ.

[16] R.H. Myers, Classical and Modern Regression with Applications, 2nd ed., Duxbury, (1990). http://books.google.fi/books?id=LOHHKQAACAAJ.

[17] M. van Heeswijk, Y. Miche, E. Oja, A. Lendasse, GPU-accelerated and parallelized ELM ensembles for large-scale regression, Neurocomputing 74 (16) (2011) 2430–2437. http://dx.doi.org/10.1016/j.neucom.2010.11.034.

[18] W.-Y. Deng, Q.-H. Zheng, L. Chen, Regularized extreme learning machine, in: IEEE Symposium on Computational Intelligence and Data Mining, CIDM'09, IEEE, 2009, pp. 389–395. http://dx.doi.org/10.1109/CIDM.2009.4938676.

[19] J. Nelder, R. Mead, A simplex method for function minimization, Comput. J. 7 (4) (1965) 308–313.

[20] J.C. Lagarias, J.A. Reeds, M.H. Wright, P.E. Wright, Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions (1998). http://dx.doi.org/10.1137/S1052623496303470.

[21] J. Triesch, A gradient rule for the plasticity of a neuron's intrinsic excitability, in: W. Duch, J. Kacprzyk, E. Oja, S. Zadrozny (Eds.), Artificial Neural Networks: Biological Inspirations—ICANN 2005, vol. 3696, Springer, Berlin/Heidelberg, 2005, pp. 65–70. http://dx.doi.org/10.1007/11550822_11.

[22] J. Triesch, Synergies between intrinsic and synaptic plasticity in individual model neurons, in: L. Saul, Y. Weiss, L. Bottou (Eds.), Advances in Neural Information Processing Systems, vol. 17, MIT Press, 2005, pp. 1417–1424. http://dx.doi.org/10.1162/neco.2007.19.4.885.

[23] D. Verstraeten, B. Schrauwen, M. D'Haene, D. Stroobandt, An experimental unification of reservoir computing methods, Neural Netw. 20 (3) (2007) 391–403.

[24] K. Neumann, J.J. Steil, Intrinsic plasticity via natural gradient descent, in: ESANN 2012: 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2012, pp. 555–560.

[25] A. Asuncion, D.J. Newman, UCI Machine Learning Repository, 2007.

Mark van Heeswijk has been working as an exchange student in both the EIML (Environmental and Industrial Machine Learning, previously TSPCi) Group and the 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. His main research interests include high-performance computing, scalable machine learning methods, ensemble models and neural networks like extreme learning machines and deep belief networks.

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.

Publication VI

Mark van Heeswijk, Amaury Lendasse, and Yoan Miche. Compressive ELM: Improved Models Through Exploiting Time-Accuracy Trade-offs. In *CCIS 459 - Engineering Applications of Neural Networks*, pp. 165-174, 2014.

Compressive ELM: Improved Models through Exploiting Time-Accuracy Trade-Offs

Mark van Heeswijk[1], Amaury Lendasse[1,2,3], and Yoan Miche[1]

[1] Aalto University School of Science,
Department of Information and Computer Science,
P.O. Box 15400, FI-00076 Aalto, Finland
[2] Arcada University of Applied Sciences, Helsinki, Finland
[3] Department of Mechanical and Industrial Engineering, The University of Iowa,
Iowa City, IA 52242-1527, USA

Abstract. In the training of neural networks, there often exists a trade-off between the time spent optimizing the model under investigation, and its final performance. Ideally, an optimization algorithm finds the model that has best test accuracy from the hypothesis space as fast as possible, and this model is efficient to evaluate at test time as well. However, in practice, there exists a trade-off between training time, testing time and testing accuracy, and the optimal trade-off depends on the user's requirements. This paper proposes the Compressive Extreme Learning Machine, which allows for a time-accuracy trade-off by training the model in a reduced space. Experiments indicate that this trade-off is efficient in the sense that on average more time can be saved than accuracy lost. Therefore, it provides a mechanism that can yield better models in less time.

Keywords: Extreme Learning Machine, ELM, random projection, compressive sensing, Johnson-Lindenstrauss, approximate matrix decompositions.

1 Introduction

When choosing a model for solving a machine learning problem, which model is most suitable depends a lot on the context and the requirements of the application. For example, it might be the case that the model is trained on a continuous stream of data, and therefore has some restrictions on the training time. On the other hand, computational time in the testing phase might be restricted, like in a setting where the model is used as the controller for an aircraft or a similar setting that requires fast predictions. Alternatively, the context in which the model is applied might not have any strong constraints on the computational time, and above all, accuracy or interpretability is considered most important regardless of the computational time.

This paper focuses on time-accuracy trade-offs in a neural network architecture known as Extreme Learning Machine [1], and on trade-offs between training time and accuracy in particular. This trade-off can be affected in two ways:

– by improving the accuracy through spending more time optimizing the model,
– or vice-versa, by reducing the computational time of the model, without sacrificing accuracy too much.

Each type of model has its own ways of balancing computational time and accuracy, and has an associated curve (or set of points) on a "training time"-accuracy plot that expresses the efficiency of the model in achieving a certain accuracy (the closer the curve is to the bottom left, the better). Thus, given a collection of models, the question becomes: which model produces the best accuracy the fastest?

The remainder of this paper is organized as follows. Section 2 discusses the preliminaries and methods relevant for this paper and gives an example of the time-accuracy trade-offs that exist within several ELM variants. This illustrates the notion of 'efficiency' of a model, and motivates the choice of model that is studied in the rest of the paper. Section 3 proposes the Compressive ELM, a new model which allows trading off computational time and accuracy by performing the training in a reduced problem space rather than the original space. Finally, Section 4 contains the experiments and analysis which form the validation for the proposed approach.

2 Background

Regression / Classification. In this paper, the focus is on the problem of regression, which 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 $\mathbf{x}_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$. Note that although in this paper the focus is on regression, the proposed approach can just as well be used when applying the ELM in a classification context.

Extreme Learning Machine (ELM). The ELM algorithm is proposed by Huang *et al.* in [1] and uses Single-Layer Feedforward Neural Networks (SLFN). The key idea of ELM is that the hidden layer weights and hidden layer biases of the SLFN can be generated randomly, and do not need to be trained.

Consider a set of N distinct samples $(\mathbf{x}_i, y_i)$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. Then, an SLFN with M hidden neurons can be written as

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

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

Gathering the outputs of the transfer functions in an $N \times M$ matrix $\mathbf{H}$ and the targets in $\mathbf{Y}$, in case the network would perfectly approximate the targets this can be written compactly as

$$\mathbf{H}\beta = \mathbf{Y}, \tag{2}$$

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

$$\mathbf{H} = \begin{pmatrix} f(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & f(\mathbf{w}_M \cdot \mathbf{x}_1 + b_M) \\ \vdots & \ddots & \vdots \\ f(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & f(\mathbf{w}_M \cdot \mathbf{x}_N + b_M) \end{pmatrix} \tag{3}$$

and $\beta = (\beta_1 \dots \beta_M)^T$ and $\mathbf{Y} = (y_1 \dots y_N)^T$. Under the condition that the input weights and biases are randomly initialized, and the transfer function f is a bounded non-constant piecewise continuous activation function, [2] proves that the ELM is a universal approximator. Therefore, given enough neurons, the ELM can approximate a function or set of target values as good as desired. The optimal least-squares solution to the equation $\mathbf{H}\beta = \mathbf{Y}$ in the ELM algorithm is $\beta = \mathbf{H}^\dagger\mathbf{Y}$, where $\mathbf{H}^\dagger$ is the pseudo-inverse of $\mathbf{H}$. In summary then, the standard ELM algorithm can be described in Algorithm 1. Theoretical proofs and a more thorough presentation of the ELM algorithm can be found in [1].

Algorithm 1. Standard 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 $M$ hidden nodes:
1: - Randomly assign input weights $\mathbf{w}_i$ and biases $b_i$, $i \in [1, M]$;
2: - Calculate the hidden layer output matrix $\mathbf{H}$;
3: - Calculate output weights matrix $\beta = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}\mathbf{Y} = \mathbf{H}^\dagger\mathbf{Y}$.
```

Efficient Optimization of Regularization Parameter with SVD. Trained on a limited number of samples, the standard ELM is prone to overfitting the training data. One way of preventing overfitting is by applying Tikhonov Regularization, in which case pseudo-inverse used in the ELM becomes

$$\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T$$

for some regularization parameter λ [3]. Each value of λ results in a different pseudo-inverse $\mathbf{H}^\dagger$, and it would be computationally expensive to recompute the pseudo-inverse for every λ. However, by incorporating the regularization in the singular value decomposition (SVD) approach to compute the pseudo-inverse, it becomes possible to obtain the various $\mathbf{H}^\dagger$'s with minimal re-computation [4]. This scheme is first described in the context of ELM in [5], and is summarized next (with some minor optimizations). Suppose

$$\begin{aligned} \hat{\mathbf{Y}} &= \mathbf{H}\beta \\ &= \mathbf{H}(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{Y} \\ &= \mathbf{H}\mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y} \\ &= \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y} \\ &= \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^T\mathbf{Y} \\ &= \mathrm{HAT} \cdot \mathbf{Y} \end{aligned}$$

where $\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}$ is a diagonal matrix with $\frac{d_{ii}^2}{d_{ii}^2+\lambda}$ as the i^{th} diagonal entry. From the above equations it can now be seen that given $\mathbf{U}$:

$$\begin{aligned}
\text{MSE}^{\text{TR-PRESS}} &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - hat_{ii}}\right)^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - \mathbf{h}_{i\cdot}(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{h}_{i\cdot}^T}\right)^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \hat{y}_i}{1 - \mathbf{u}_{i\cdot}\left(\frac{d_{ii}^2}{d_{ii}^2+\lambda}\right)\mathbf{u}_{i\cdot}^T}\right)^2
\end{aligned}$$

where $\mathbf{h}_{i\cdot}$ and $\mathbf{u}_{i\cdot}$ are the i^{th} row vectors of $\mathbf{H}$ and $\mathbf{U}$, respectively. The optimal Tikhonov-regularized PRESS and corresponding λ can be determined efficiently using Algorithm 2. Due to the convex nature of criterion $\text{MSE}^{\text{TR-PRESS}}$ with respect to regularization parameter λ, the Nelder-Mead procedure used for optimizing λ converges quickly in practice [6,7].

Algorithm 2. Tikhonov-regularized PRESS. In practice, the **while** part of this algorithm (convergence for λ) is solved using by a Nelder-Mead approach [6], a.k.a. downhill simplex.

```
 1: Decompose $\mathbf{H}$ by SVD: $\mathbf{H} = \mathbf{UDV}^T$
 2: Precompute $\mathbf{B} = \mathbf{U}^T\mathbf{y}$
 3: while no convergence on $\lambda$ achieved do
 4:     - Precompute $\mathbf{C} = \mathbf{U} \cdot \text{diag}\left(\frac{d_{11}^2}{d_{11}^2+\lambda}, \ldots, \frac{d_{nn}^2}{d_{nn}^2+\lambda}\right)$
 5:     - Compute $\hat{\mathbf{y}} = \mathbf{CB}$, the vector containing all $\hat{y}_i$
 6:     - Compute $\mathbf{d} = \text{diag}\left(\mathbf{CU}^T\right)$, the diagonal of the HAT matrix, by taking the
        row-wise dot-product of $\mathbf{C}$ and $\mathbf{U}$
 7:     - Compute $\varepsilon = \frac{\mathbf{y}-\hat{\mathbf{y}}}{\mathbf{1}-\mathbf{d}}$, the leave-one-out errors
 8:     - Compute $\text{MSE}^{\text{TR-PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$
 9: end while
10: Keep the best $\text{MSE}^{\text{TR-PRESS}}$ and the associated $\lambda$ value
```

Example: Time-Accuracy Trade-offs for Several ELM Variants. In order to illustrate what time-accuracy trade-offs exist within ELM, and to motivate the choice of model studied later in this paper, this section presents time-accuracy trade-offs of several models:

– **ELM:** the basic ELM [1].
– **Optimally Pruned ELM (OP-ELM):** ELM trained by generating a set of neurons, ranking them by relevance, and then determining the optimal prefix of that sorted list of neurons in terms of leave-one-out error [8]
– **TROP-ELM:**OP-ELM with efficient optimization of the Tikhonov regularization integrated, using the SVD approach to computing $\mathbf{H}^\dagger$ [5]

- **TR-ELM:** Tikhonov-regularized ELM [3], with efficient optimization of regularization parameter λ, using the SVD approach. [9]
- **BIP(0.2), BIP(rand), BIP(CV):** ELMs pretrained using Batch Intrinsic Plasticity mechanism [10], aimed at adapting the hidden layer weights and biases, such that they retain as much information from the input as possible. The variants included here have the BIP parameter μ_{exp} fixed to a 0.2, randomized, or cross-validated over 20 possible values.

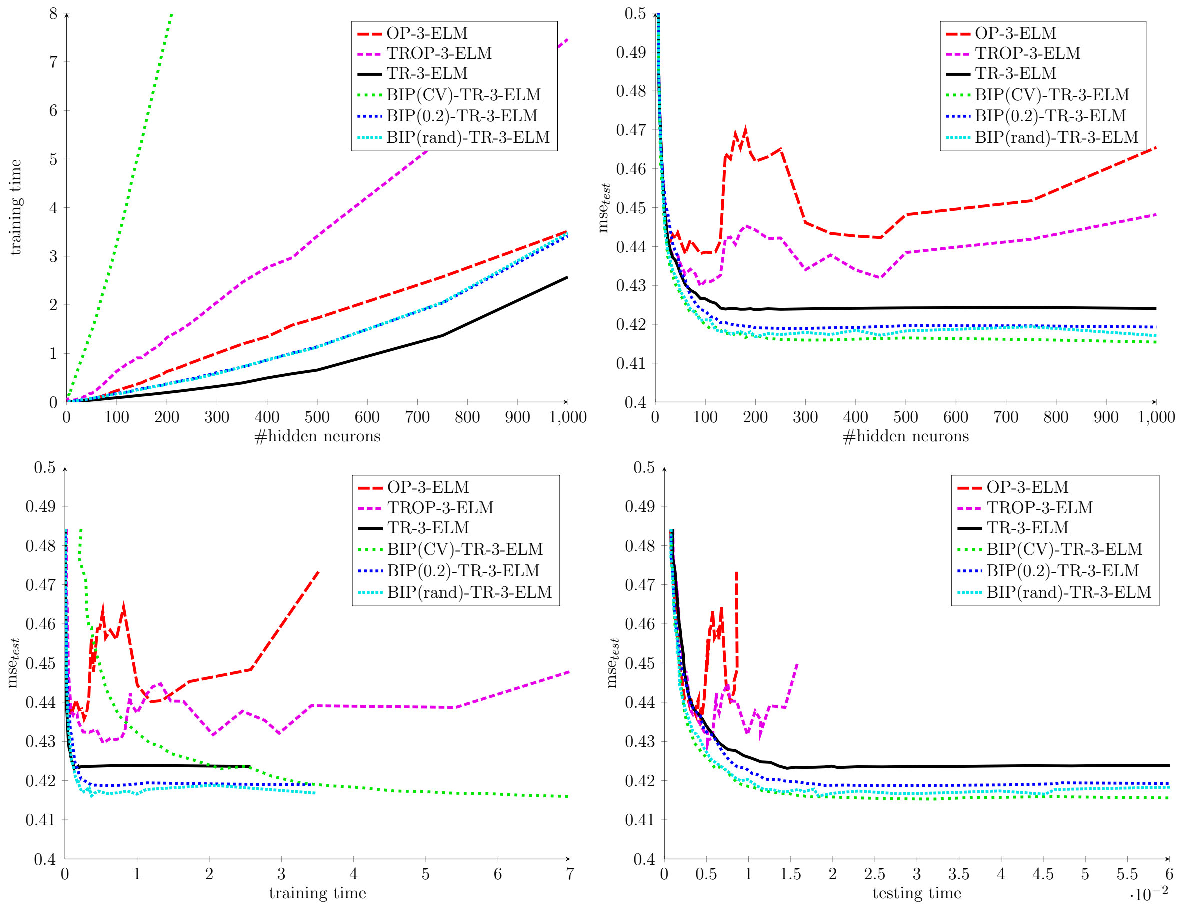


Fig. 1. Results for various ELM variants on Abalone UCI data set

All these models are trained and tested on the Abalone data set from the UCI repository [11] (see Section 4 for details), use ternary weights (see [9]), and have an initial number of hidden neurons varying between 2 and 1000. Each method trains and optimizes the ELM in its own way, with results as summarized in Figure 1. Depending on the users criteria, these results suggest:

- if *training time* most important, then BIP(rand)-TR-3-ELM is the obvious choice from all candidates as it provides almost optimal performance, while keeping training time low.
- if *test error* is most important, then BIP(CV)-TR-3-ELM is the best choice. However, since it cross-validates over 20 possible parameter values, the training time is 20 times as high, while only giving slightly better accuracy.

– if *testing time* is most important, then surprisingly TR-3-ELM is also the most attractive model. Even though OP-ELM and TROP-ELM tend to be faster in test, they suffer from slight overfitting as the number of initial hidden neurons increases. Therefore, the TR-3-ELM is the best choice, since it generally results in models with the best accuracy and lowest testing time.

Since TR-ELM offers attractive trade-offs between speed and accuracy, this model will be central in the rest of the paper. Furthermore, since due to the proper regularization the TR-ELM does not seem to overfit even for large number of neurons: more neurons generally means better accuracy. Naturally, this comes at an increase in training time, which is something that will be addressed in the next section, where the Compressive ELM is presented.

3 Compressive Extreme Learning Machine

Considering training time-accuracy trade-offs like in Figure 1, two possible strategies present itself to obtain models that are preferable over other models:

– reducing test error, using some efficient algorithm ("in terms of training time-accuracy plot: "pushing the curve down")
– reducing computational time, while retaining as much accuracy as possible ("in terms of training time-accuracy plot: "pushing the curve to the left")

The latter is the strategy that is taken in Compressive ELM: instead of performing the training in the original problem space, it performs the training in a reduced space, and then project the solution back to the original space.

Johnson-Lindenstrauss and Approximate Matrix Decompositions. Given an $m \times n$ matrix, an approximate matrix decomposition can be achieved by first embedding the rows of the matrix into a lower-dimensional space (through one of many available low-distortion Johnson-Lindenstrauss-like embeddings), solving the decomposition, and then projecting back to the full space. If such an embedding (or sketch) is accurate, then this allows for solving the problem with high accuracy in reduced time. The algorithm for Approximate SVD is summarized in Algorithm 3, and more background can be found in [12].

Algorithm 3. Approximate SVD [12]

```
Given an $m \times n$ matrix $A$, compute k-term approximate SVD $A \approx UDV^T$ as follows:
 1: - Form the $n \times (k+p)$ random matrix $\Omega$. (where p is small over sampling parameter)

 2: - Form the m $\times (k+p)$ sampling matrix $Y = A\Omega$. ("sketch" it by applying $\Omega$)
 3: - Form the m $\times (k+p)$ orthonormal matrix $Q$, such that $range(Q) = range(Y)$.
 4: - Compute $B = Q^* A$.
 5: - Form the SVD of $B$ so that $B = \hat{U}DV^T$
 6: - Compute the matrix $U = Q\hat{U}$
```

Faster Sketching. Typically, the bottleneck in Algorithm 3 is the time it takes to sketch the matrix. Rather than using a class of random matrices of Gaussian variables for sketching A, one can also use random matrices that are sparse or structured in some way [13,14], for which the matrix-vector product can be computed more efficiently. Furthermore, Ailon and Chazelle [15] introduced the Fast Johnson-Lindenstrauss Transform (FJLT), which uses a class of random matrices that allow application of an $n \times n$ matrix to a vector in $\mathcal{O}(n\log(n))$, rather than the usual $\mathcal{O}(n^2)$. Besides this obvious speedup, this class of matrices is also more successful in creating a low-distortion embedding when applied to a sparse matrix. These transforms consist of the application of three easy-to-compute matrices

$$\big(P\big)_{k\times n}\big(H\big)_{n\times n}\big(D\big)_{n\times n}$$

where P, H, and D vary depending on the exact scheme. Generally, D is a diagonal matrix with random Rademacher variables $(-1, +1)$ on the diagonal, H is encoding either the discrete Hadamard or discrete Fourier transform, and P is a sparse random matrix or a matrix sampling random columns from H. The D matrix can be applied to a vector x in $\mathcal{O}(n)$, The H matrix can be applied in $\mathcal{O}(n\log(n))$, and the P matrix adds a factor $nnz(P)$ or k, depending on the type.

4 Experiments

This section describes the experiments that investigate the trade-off between computational time (both training and test), and the accuracy of the Compressive ELM in relation to, the dimensionality of the space into which the problem is reduced, using the sketch. For sketching, TR-3-ELMs with the following sketching schemes are considered, and compared with the standard TR-3-ELM:

- Gaussian: sketching is performed using a $k \times n$ matrix of random Gaussian variables
- FJLT: the transform introduced in [15], for which P is a sparse matrix of random Gaussian variables, and H encodes the Discrete Hadamard Transform
- SRHT: a variant of the FJLT, for which P is a matrix selecting k random columns from H, and H encodes the Discrete Hadamard Transform

The number of hidden neurons in each model is varied between 2 and 1000, and parameter k is chosen from $[50, 100, 200, 400, 600]$. Experiments are repeated with 200 random realizations of the training and test set, and average results over those 200 runs are reported.

Data and Preprocessing. As data sets, different regression tasks from the UCI machine learning repository [11] are tested. Due to space restrictions only the results for CaliforniaHousing and FJLT sketching are presented here, but similar results hold for the other data sets and sketching methods. In each run, the data is divided randomly into 8000 random samples for training and and the remaining 12640 samples for testing. The data is preprocessed in such a way that each input and output variable is zero mean and unit variance.

Results. The results of the experiment are summarized in Figure 2. There, it can be seen that

– setting k lower than the number of neurons results in faster training times (which makes sense since the problem solved is smaller).
– as long as parameter k is chosen large enough, the method is not losing efficiency (i.e. there is no model that achieves better error in the same computational time), and it is potentially gaining efficiency (as shown by the bottom-left plot of Figure 2.

Finally, the experiments showed that sketches with Gaussian matrices are generally the fastest. Furthermore, for the tested problem sizes, the SRHT (which allows an efficient matrix multiplication) is generally faster than the FJLT (which uses sparse matrices). Although for this problem size the SRHT and FJLT are slower, they might still be needed in case the matrix to sketch is sparse [15].

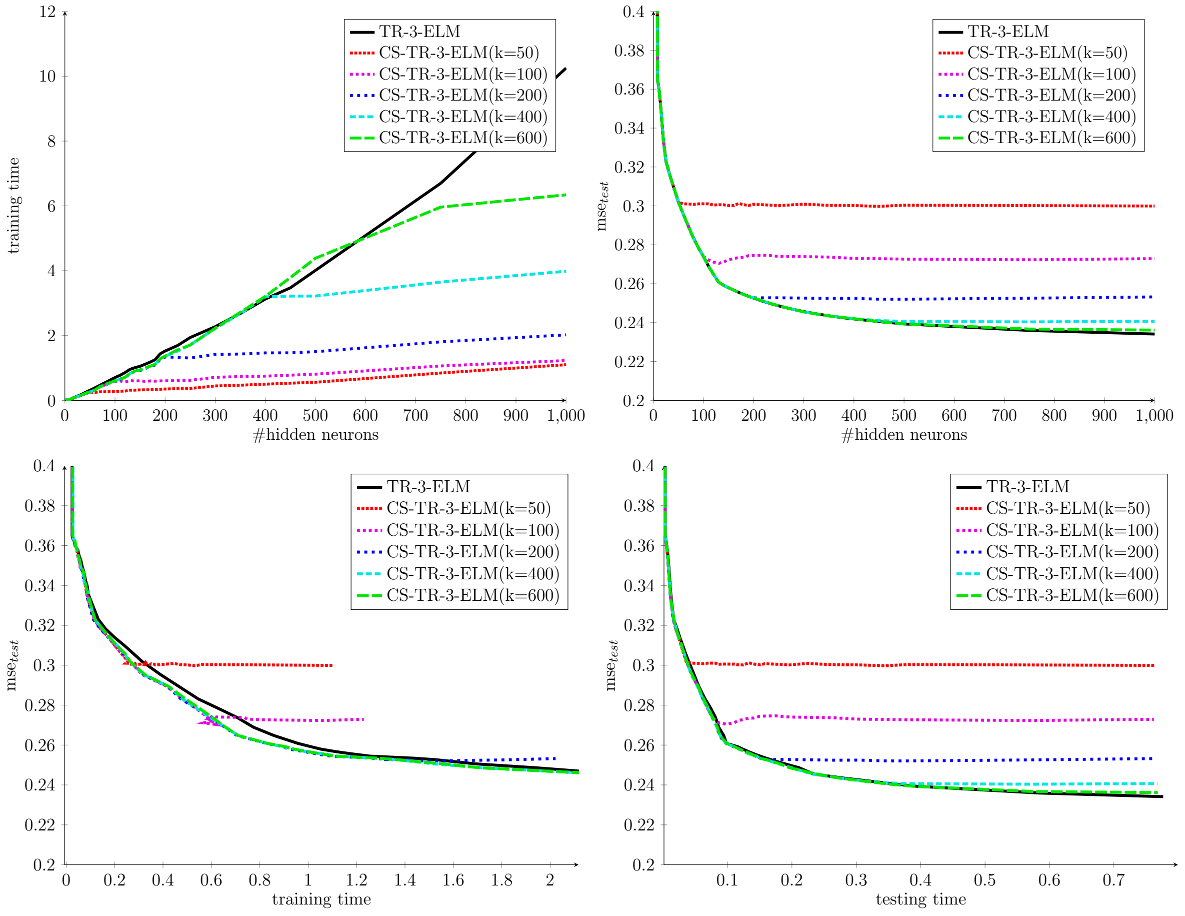


Fig. 2. Results for Compressive ELMs using FJLT sketching with varying k on CaliforniaHousing UCI data set

5 Conclusion

In this paper, the trade-off between computational time and test error has been investigated, in particular the trade-off between training time and test error. Having information about this trade-off for different models is useful information in selecting the most suitable model for a particular task.

The Compressive ELM proposed in this paper investigates a way to reduce training time by doing the optimization in a reduced space of k dimensions, and is shown to be efficient in the sense that (given k large enough), among the tested models the Compressive ELM achieves the best test error for each computational time (i.e. there are no models that achieve better test error and can be trained in the same or less time). A promising candidate for setting k such that it optimally reduces computational time (yet retains accuracy), would be to let k be informed by the theoretical bounds currently known for the sketching schemes. These theoretical bounds give lower bounds on k for which a low-distortion embedding of the given n points can be achieved with high probability. Although these bounds are typically not sharp (and therefore not optimal), in case the minimal k for successful embedding is lower than the number of neurons in the ELM, it can be exploited to reduce the training time.

Finally, developing low-distortion embeddings and sharpening their associated bounds is currently a hot topic of research, and any new developments in this area can easily be integrated to improve the performance of Compressive ELM.

References

1. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: Theory and applications. Neurocomputing 70(1-3), 489–501 (2006)
2. Huang, G.-B., Chen, L., Siew, C.-K.: Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Nodes. IEEE Transactions on Neural Networks 17(4), 879–892 (2006)
3. Deng, W.-Y., Zheng, Q.-H., Chen, L.: Regularized extreme learning machine. In: IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, pp. 389–395 (2009)
4. van Heeswijk, M., Miche, Y., Oja, E., Lendasse, A.: GPU-accelerated and parallelized ELM ensembles for large-scale regression. Neurocomputing 74(16), 2430–2437 (2011)
5. Miche, Y., van Heeswijk, M., Bas, P., Simula, O., Lendasse, A.: TROP-ELM: A double-regularized ELM using LARS and Tikhonov regularization. Neurocomputing 74(16), 2413–2421 (2011)
6. Nelder, J., Mead, R.: A simplex method for function minimization. The Computer Journal 7(4), 308–313 (1965)
7. Lagarias, J.C., Reeds, J.A., Wright, M.H., Wright, P.E.: Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions. SIAM Journal on Optimization 9, 112–147 (1998)
8. Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., Lendasse, A.: OP-ELM: optimally pruned extreme learning machine. IEEE Transactions on Neural Networks 21(1), 158–162 (2010)

9. van Heeswijk, M., Miche, Y.: Binary/Ternary Extreme Learning Machines. Neurocomputing (to appear)
10. Neumann, K., Steil, J.J.: Batch intrinsic plasticity for extreme learning machines. In: Honkela, T. (ed.) ICANN 2011, Part I. LNCS, vol. 6791, pp. 339–346. Springer, Heidelberg (2011)
11. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository (2007)
12. Halko, N., Martinsson, P.-G., Tropp, J.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions (September 2011) arXiv:0909.4061
13. Achlioptas, D.: Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of Computer and System Sciences 66(4), 671–687 (2003)
14. Matoušek, J.: On variants of the Johnson-Lindenstrauss lemma. Random Structures & Algorithms, 142–156 (2008)
15. Ailon, N., Chazelle, B.: Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In: Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing, STOC 2006, pp. 557–563. ACM Press, New York (2006)

Nowadays, due to advances in technology, the size and dimensionality of data sets used in machine learning 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 to large data sets, such that it is still possible to complete the machine learning task in reasonable time.

This dissertation introduces several machine learning methods based on Extreme Learning Machines (ELMs), meant to deal with these challenges. It focuses on developing efficient, yet accurate and flexible methods. These contributions take three main directions. Firstly, ensemble approaches based on ELM, that adapt to context and can scale to large data. Secondly, ELM-based variable selection approaches, that result in more accurate and efficient models. Finally, training algorithms for ELM that allow for a flexible trade-off between accuracy and computational time.

Aalto-DD 43/2015

ISBN 978-952-60-6148-1 (printed)
ISBN 978-952-60-6149-8 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Information and Computer Science
www.aalto.fi