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/

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 $\mathbf{H}$ 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 $\mathbf{H}$ 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 $\mathbf{H}$ 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 λ 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 $\mathbf{d} = \text{diag}(\mathbf{CU}^T)$, 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 $\text{MSE}^{\text{TR}-\text{PRESS}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\varepsilon}_i^2$

9: **end while**

10: Keep the best $\text{MSE}^{\text{TR}-\text{PRESS}}$ and the associated λ 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 $\mathbf{w}$ and $\mathbf{x}$. Now, if the weights $\mathbf{w}$ have diverse directions, the angles θ between $\mathbf{w}$ and $\mathbf{x}$ – 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* $\leq M$ 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.

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

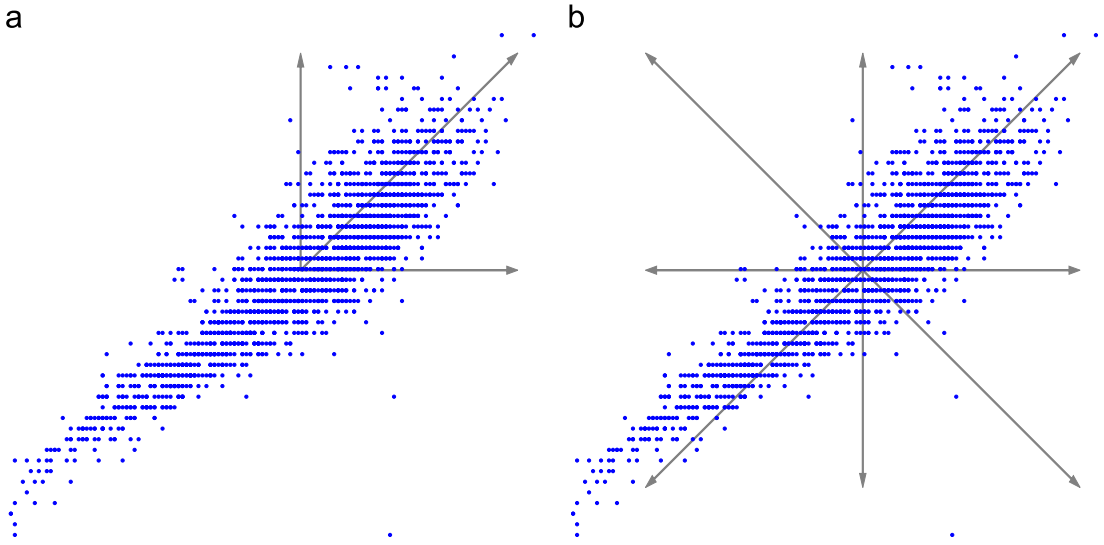

Fig. 1. Illustration of possible weights (*arrows*) for binary (a) and ternary (b) weight scheme, in a 2D subspace of normalized Abalone data (*blue dots*). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

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.

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

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.

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.

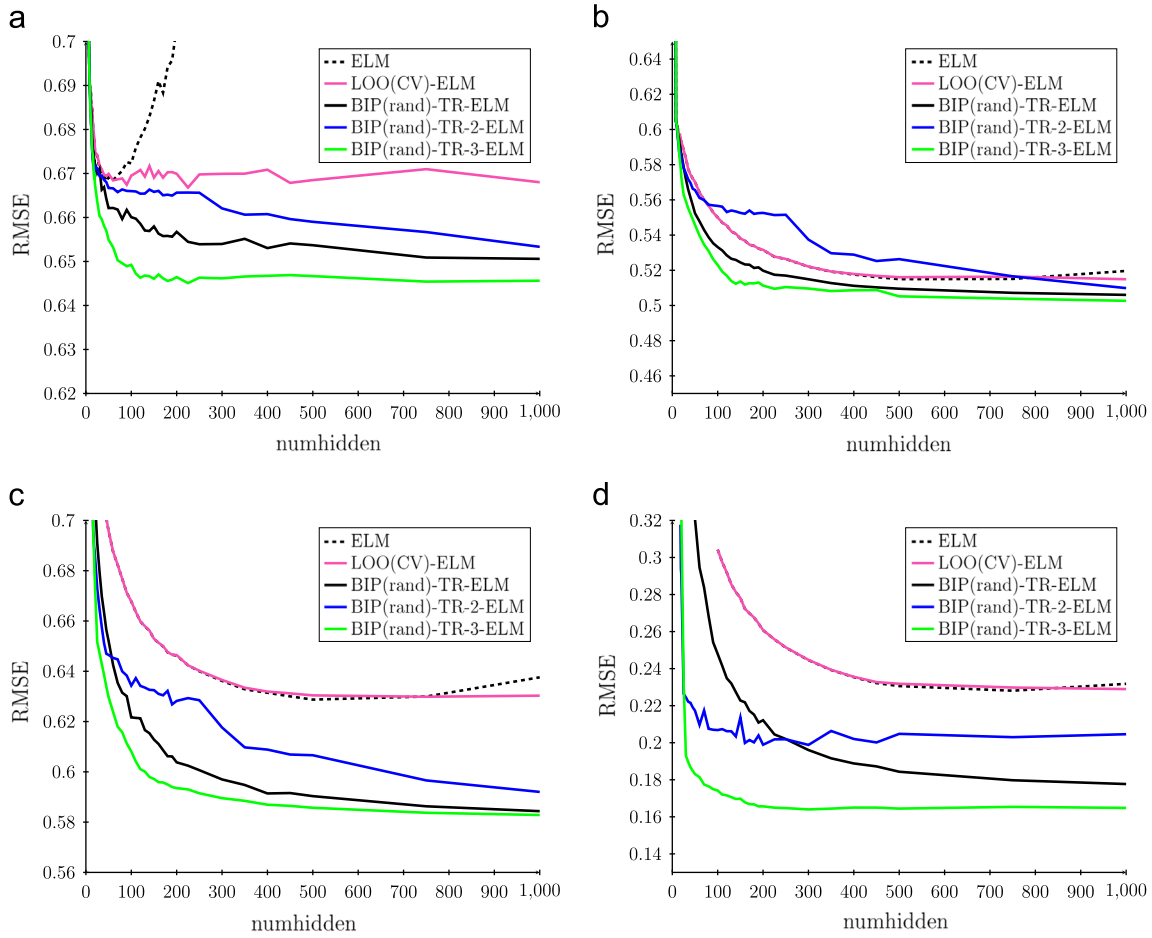

Fig. 2. Number of neurons vs. average achieved test RMSE for ELM (*black, dashed*), LOO(CV)-ELM (*purple*), BIP(rand)-TR-ELM with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme. (a) Abalone. (b) CaliforniaHousing. (c) CensusHouse8L. (d) ComputerActivity. (For interpretation of the 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

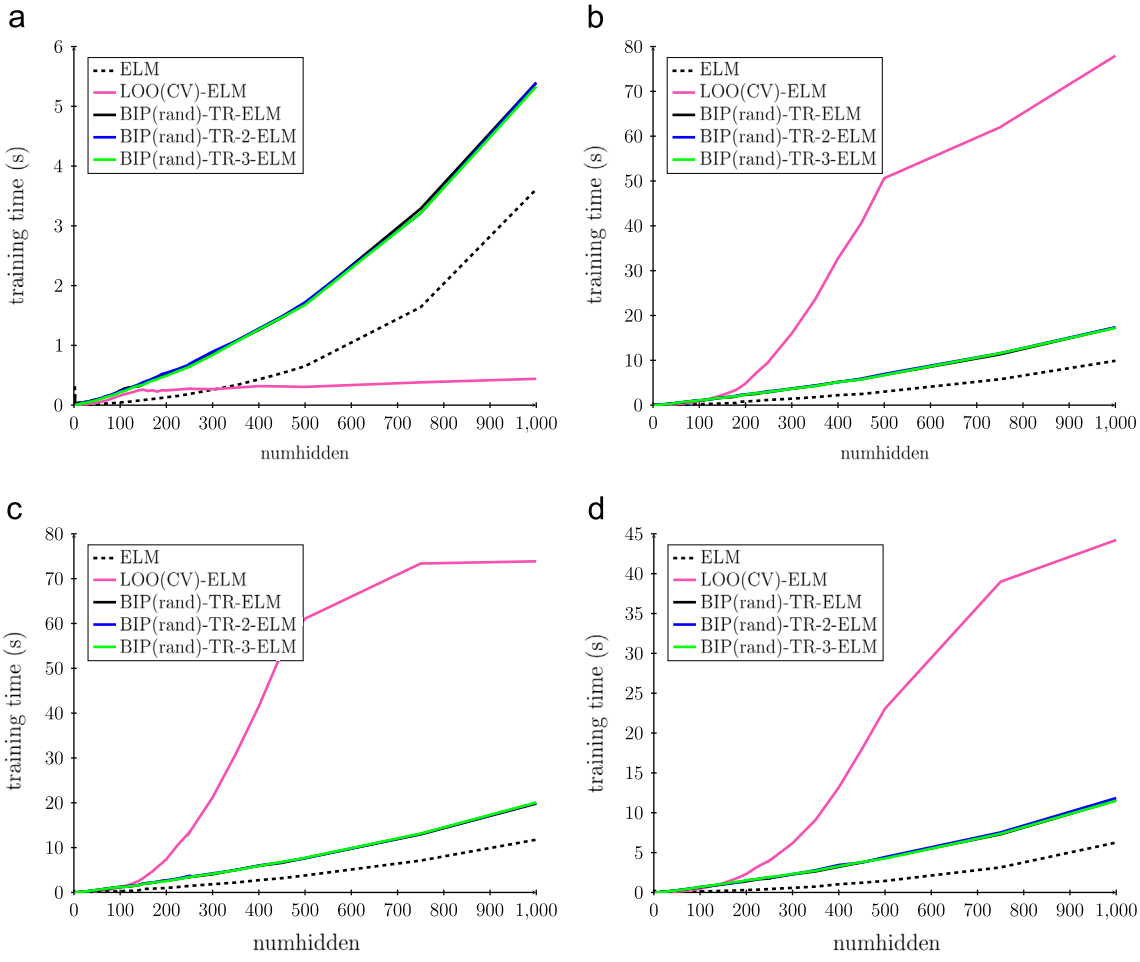

Fig. 3. Number of neurons vs. average training time for ELM (*black, dashed*), LOO(CV)-ELM (*purple*), BIP(rand)-TR-ELM with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme. (a) Abalone. (b) CaliforniaHousing. (c) CensusHouse8L. (d) ComputerActivity. (For interpretation of the references 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).

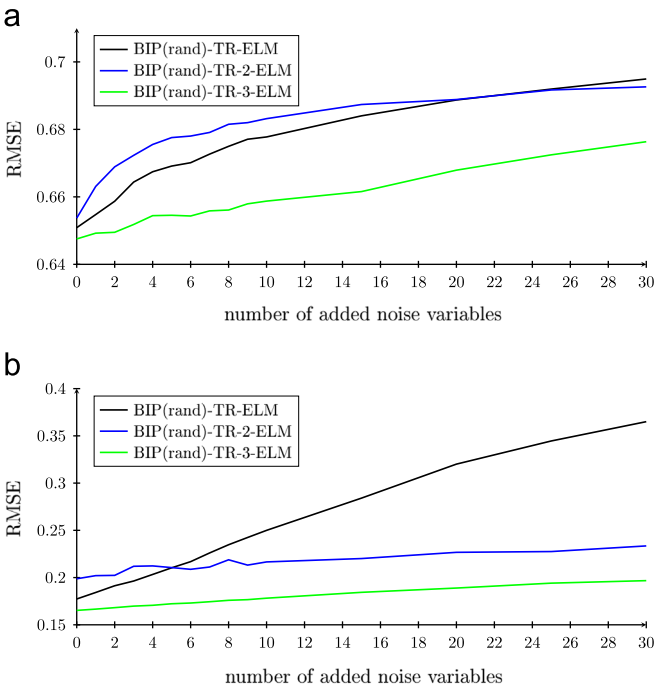

Fig. 4. Effect of adding irrelevant extra variables on RMSE for BIP(rand)-TR-ELM with 1000 hidden neurons and with Gaussian (*black*), binary (*blue*), ternary (*green*) weight scheme. (a) Abalone. (b) ComputerActivity. (For interpretation of the references to color in this figure caption, the reader 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 **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**

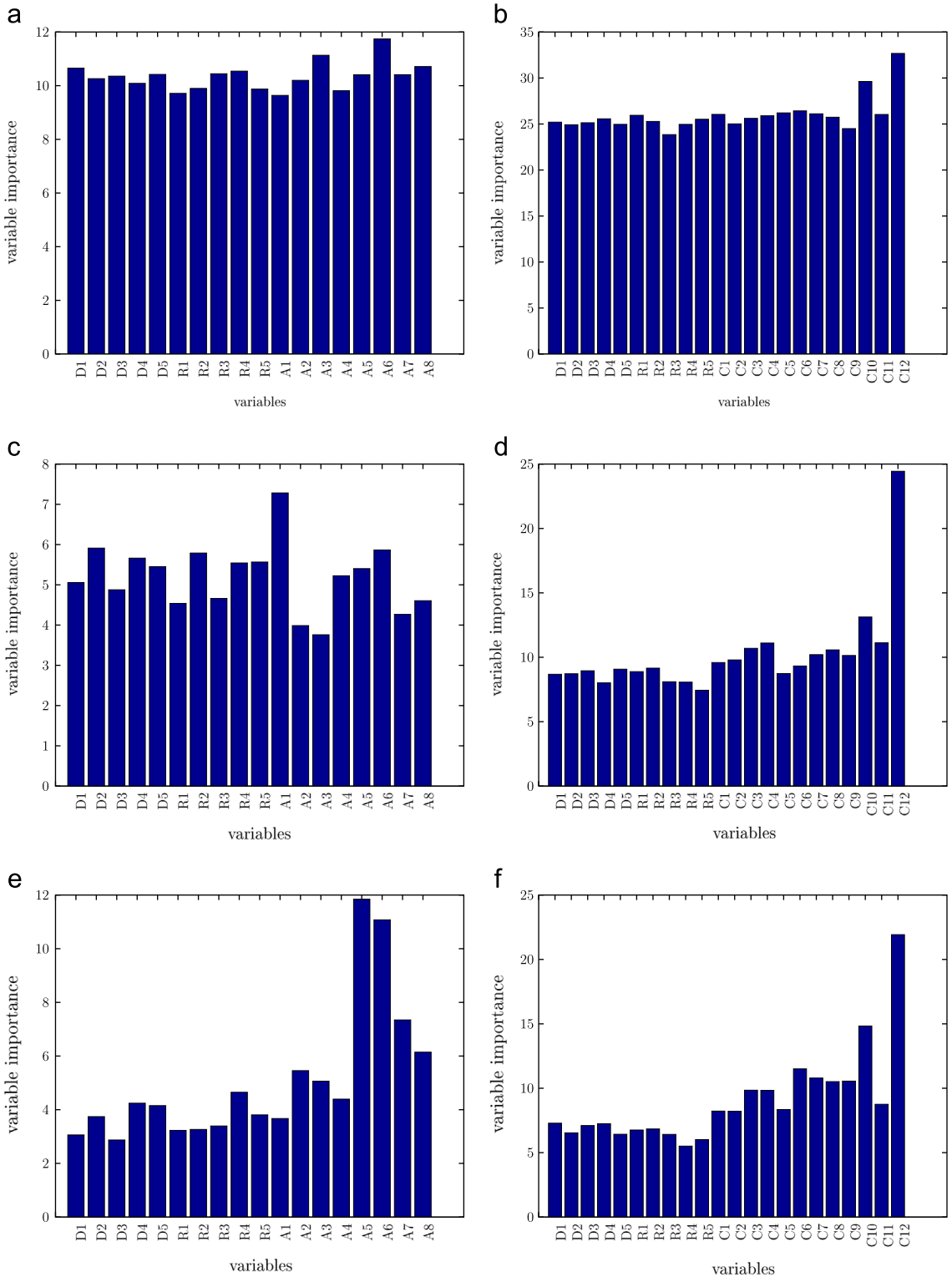

Fig. 5. Variable relevance measure for Abalone and ComputerActivity with 5 random noise variables R1, …, R5, 5 irrelevant variables D1, …, D5, and the original variables. (a) Abalone, Gaussian. (b) Computer Activity, Gaussian (c) Abalone, binary. (d) Computer Activity, binary. (e) Abalone, ternary. (f) 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 12^{th} 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	$5.29e3 \pm 3.60e4$	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	$1.33e2 \pm 1.02e2$	$2.11e-3 \pm 9.54e-5$	$5.24e5 \pm 3.57e6$	$2.22e6 \pm 3.48e6$	$1.34e6 \pm 1.13e6$
LOO(CV)-ELM	$2.20 \pm 1.10e-1$	$1.44e-3 \pm 1.38e-5$	$1.11e2 \pm 8.08e2$	$5.10e4 \pm 4.45e4$	$6.38e4 \pm 3.16e3$
BIP(rand)-TR-ELM	$2.11 \pm 3.60e-2$	$1.45e-3 \pm 1.30e-5$	$3.18 \pm 2.10e-1$	$3.16e4 \pm 5.27e2$	$5.85e4 \pm 1.03e3$
BIP(rand)-TR-2-ELM	$2.12 \pm 5.05e-2$	$1.45e-3 \pm 1.33e-5$	$3.58 \pm 4.21e-1$	$3.19e4 \pm 6.81e2$	$5.94e4 \pm 9.98e2$
BIP(rand)-TR-3-ELM	$2.09 \pm 3.86e-2$	$1.45e-3 \pm 1.36e-5$	$3.06 \pm 5.74e-2$	$3.14e4 \pm 5.94e2$	$5.78e4 \pm 9.15e2$

A.3. Denormalized RMSEs for zero mean, unit variance normalization

	Ab	De	Co	Ce	Ca
ELM	5.46 ± 1.37	1.68e−3 ± 2.13e−5	4.25 ± 1.24e−1	3.38e4 ± 4.23e2	5.99e4 ± 5.68e2
LOO(CV)-ELM	2.15 ± 4.36e−2	1.44e−3 ± 1.10e−5	4.20 ± 1.09e−1	3.34e4 ± 4.50e2	5.94e4 ± 5.40e2
BIP(rand)-TR-ELM	2.10 ± 3.08e−2	1.43e−3 ± 1.18e−5	3.26 ± 1.00e−1	3.10e4 ± 5.26e2	5.83e4 ± 1.01e3
BIP(rand)-TR-2-ELM	2.11 ± 3.87e−2	1.43e−3 ± 1.38e−5	3.75 ± 6.44e−1	3.14e4 ± 6.10e2	5.88e4 ± 8.52e2
BIP(rand)-TR-3-ELM	2.08 ± 3.88e−2	1.43e−3 ± 1.07e−5	3.02 ± 5.08e−2	3.09e4 ± 4.55e2	5.80e4 ± 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.