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 $\mathbf{U}$:

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

where $\mathbf{h}_{i\cdot}$ and $\mathbf{u}_{i\cdot}$ are the i^{th} row vectors of $\mathbf{H}$ and $\mathbf{U}$, respectively. The optimal Tikhonov-regularized PRESS and corresponding λ can be determined efficiently using Algorithm 2. 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 [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: $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^T$
2: Precompute $\mathbf{B} = \mathbf{U}^T\mathbf{y}$
3: while no convergence on $\lambda$ achieved do
4:     - Precompute $\mathbf{C} = \mathbf{U} \cdot \mathrm{diag}\left(\frac{d_{11}^2}{d_{11}^2+\lambda}, \ldots, \frac{d_{nn}^2}{d_{nn}^2+\lambda}\right)$
5:     - Compute $\hat{\mathbf{y}} = \mathbf{C}\mathbf{B}$, the vector containing all $\hat{y}_i$
6:     - Compute $\mathbf{d} = \mathrm{diag}\left(\mathbf{C}\mathbf{U}^T\right)$, 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 $\mathrm{MSE}^{\text{TR-PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$
9: end while
10: Keep the best $\mathrm{MSE}^{\text{TR-PRESS}}$ 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.

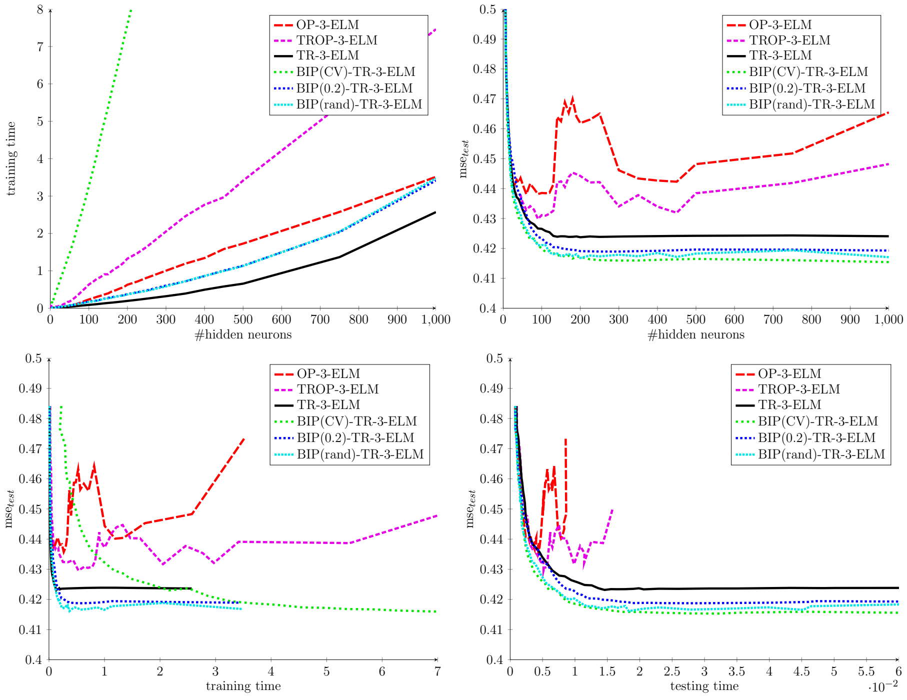

Fig. 1. Results for various ELM variants on Abalone UCI data set

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].

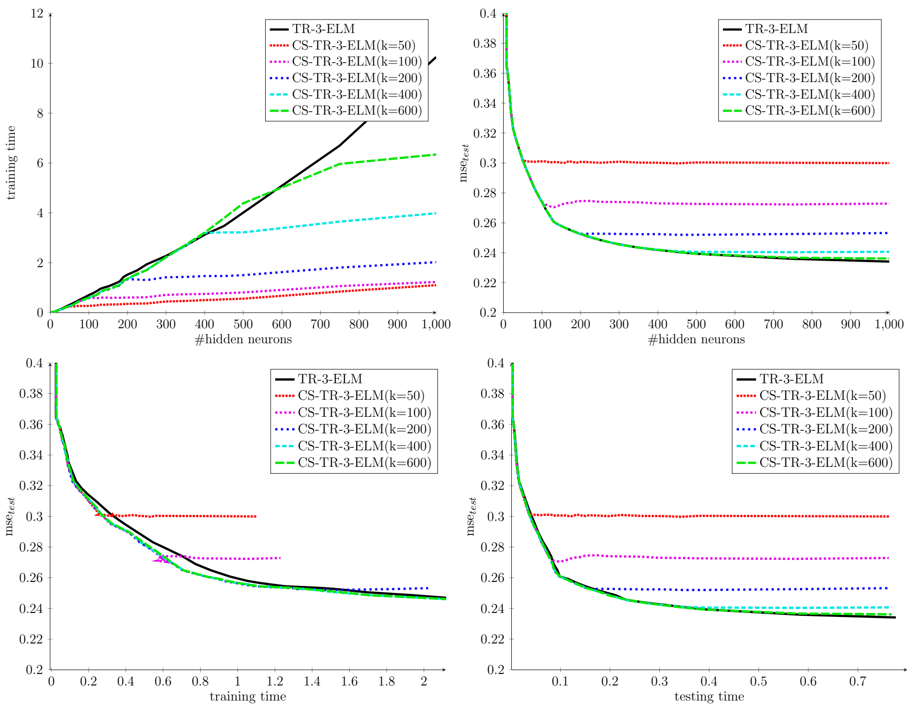

Fig. 2. Results for Compressive ELMs using FJLT sketching with varying k on CaliforniaHousing UCI data set

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)