OP-ELM: Theory, Experiments and a Toolbox

Yoan Miche[1,2], Antti Sorjamaa[1], and Amaury Lendasse[1]

[1] Department of Information and Computer Science, HUT, Finland
[2] Gipsa-Lab, INPG, France

Abstract. This paper presents the Optimally-Pruned Extreme Learning Machine (OP-ELM) toolbox. This novel, fast and accurate methodology is applied to several regression and classification problems. The results are compared with widely known Multilayer Perceptron (MLP) and Least-Squares Support Vector Machine (LS-SVM) methods. As the experiments (regression and classification) demonstrate, the OP-ELM methodology is considerably faster than the MLP and the LS-SVM, while maintaining the accuracy in the same level. Finally, a toolbox performing the OP-ELM is introduced and instructions are presented.

1 Introduction

The amount of information is increasing rapidly in many fields of science. It creates new challenges for storing the massive amounts of data as well as to the methods, which are used in the data mining process. In many cases, when the amount of data grows, the computational complexity of the used methodology also increases.

Feed-forward neural networks are often found to be rather slow to build, especially on important datasets related to the data mining problems of the industry. For this reason, the nonlinear models tend not to be used as widely as they could, even considering their overall good performances. The slow building of the networks comes from a few simple reasons; many parameters have to be tuned, by slow algorithms, and the training phase has to be repeated many times to make sure the model is proper and to be able to perform model structure selection (number of hidden neurons in the network, regularization parameters tuning...).

Guang-Bin Huang *et al.* in [1] propose an original algorithm for the determination of the weights of the hidden neurons called Extreme Learning Machine (ELM). This algorithm decreases the computational time required for training and model structure selection of the network by hundreds. Furthermore, the algorithm is rather simplistic, which makes the implementation easy.

In this paper, a methodology called Optimally-Pruned ELM (OP-ELM), based on the original ELM, is proposed. The OP-ELM methodology, presented in Section 2, is compared in Section 3 using several experiments and two well-known methods, the Least-Squares Support Vector Machine (LS-SVM) and the Multilayer Perceptron (MLP). Finally, a toolbox for performing the OP-ELM is introduced in Appendix.

V. Kůrková et al. (Eds.): ICANN 2008, Part I, LNCS 5163, pp. 145–154, 2008.

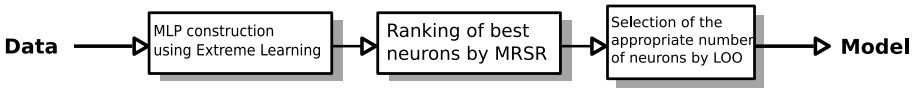

Fig. 1. The three steps of the OP-ELM algorithm

2 OP-ELM

The OP-ELM is made of three main steps summarized in Figure 1.

2.1 Extreme Learning Machine (ELM)

The first step of the OP-ELM algorithm is the core of the original ELM: the building of a single-layer feed-forward neural network. The idea of the ELM has been proposed by Guang-Bin Huang *et al.* in [1], even the idea of such network was already proposed in [2].

In the context of a single hidden layer perceptron network, let us denote the weights between the hidden layer and the output by $\mathbf{b}$. Activation functions proposed in the OP-ELM Toolbox differ from the original ELM choice since linear, sigmoid and gaussian functions are proposed here. For the output layer, a linear function is used.

The main novelty introduced by the ELM is in the determination of the kernels, initialized randomly. While the original ELM used only sigmoid kernels, gaussian, sigmoid and linear are proposed in OP-ELM: gaussian ones have their centers taken randomly from the data points and a width randomly drawn between percertile 20 percent and percentile 80 percent of the distance distribution of the input space; sigmoids weights are drawn at random from a uniform distribution in the interval $[-5, 5]$. A theorem proposed in [1] states that with the additional hypothesis of infinite differentiability of the activation functions, output weights $\mathbf{b}$ can be computed from the hidden layer output matrix $\mathbf{H}$: the columns $\mathbf{h_i}$ of $\mathbf{H}$ are computed by $\mathbf{h_i} = \mathtt{Ker}(\mathbf{x_i}^T)$, where `Ker` stands for either linear, sigmoid or gaussian activation functions (including multiplication by first layer weights). Finally, the output weights $\mathbf{b}$ are computed by $\mathbf{b} = \mathbf{H}^\dagger\mathbf{y}$, where $\mathbf{H}^\dagger$ stands for the Moore-Penrose inverse [3] and $\mathbf{y} = (y_1, \ldots, y_M)^T$ is the output.

The only remaining parameter in this process is the number of neurons N of the hidden layer. From a practical point of view, it is advised to set the number of neurons clearly above the number of the variables in the dataset, since the next step aims at pruning the useless neurons from the hidden layer.

2.2 Multiresponse Sparse Regression (MRSR)

For the removal of the useless neurons of the hidden layer, the Multiresponse Sparse Regression proposed by Timo Similä and Jarkko Tikka in [4] is used. It is mainly an extension of the Least Angle Regression (LARS) algorithm [5] and hence is actually a variable ranking technique, rather than a selection one. The

main idea of this algorithm is the following: denote by $\mathbf{T} = [\mathbf{t}_1 \dots \mathbf{t}_p]$ the $n \times p$ matrix of targets, and by $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_m]$ the $n \times m$ regressors matrix. MRSR adds each regressor one by one to the model $\mathbf{Y}^k = \mathbf{X}\mathbf{W}^k$, where $\mathbf{Y}^k = [\mathbf{y}_1^k \dots \mathbf{y}_p^k]$ is the target approximation by the model. The $\mathbf{W}^k$ weight matrix has k nonzero rows at kth step of the MRSR. With each new step a new nonzero row, and a new regressor to the total model, is introduced.

An important detail shared by the MRSR and the LARS is that the ranking obtained is exact in the case, where the problem is linear. In fact, this is the case, since the neural network built in the previous step is linear between the hidden layer and the output. Therefore, the MRSR provides the exact ranking of the neurons for our problem.

Details on the definition of a cumulative correlation between the considered regressor and the current model's residuals and on the determination of the next regressor to be added to the model can be found in the original paper about the MRSR [4].

MRSR is hence used to rank the kernels of the model: the target is the actual output y_i while the "variables" considered by MRSR are the outputs of the kernels h_i.

2.3 Leave-One-Out (LOO)

Since the MRSR only provides a ranking of the kernels, the decision over the actual best number of neurons for the model is taken using a Leave-One-Out method. One problem with the LOO error is that it can get very time consuming if the dataset tends to have a high number of samples. Fortunately, the PRESS (or PREdiction Sum of Squares) statistics provide a direct and exact formula for the calculation of the LOO error for linear models. See [6,7] for details on this formula and implementations:

$$\epsilon^{\text{PRESS}} = \frac{y_i - \mathbf{h}_i\mathbf{b}}{1 - \mathbf{h}_i\mathbf{P}\mathbf{h}_i^T}, \tag{1}$$

where $\mathbf{P}$ is defined as $\mathbf{P} = (\mathbf{H}^T\mathbf{H})^{-1}$ and $\mathbf{H}$ the hidden layer output matrix defined in subsection 2.1.

The final decision over the appropriate number of neurons for the model can then be taken by evaluating the LOO error versus the number of neurons used (properly ranked by MRSR already).

In the end, a single-layer neural network possibly using a mix of linear, sigmoid and gaussian kernels is obtained, with a highly reduced number of neurons, all within a small computational time (see section 3 for comparisons of performances and computational times between MLP, LSSVM and OP-ELM).

2.4 Discussion on the Advantages of the OP-ELM

In order to have a very fast and still accurate algorithm, each of the three presented steps have a special importance in the whole OP-ELM methodology.

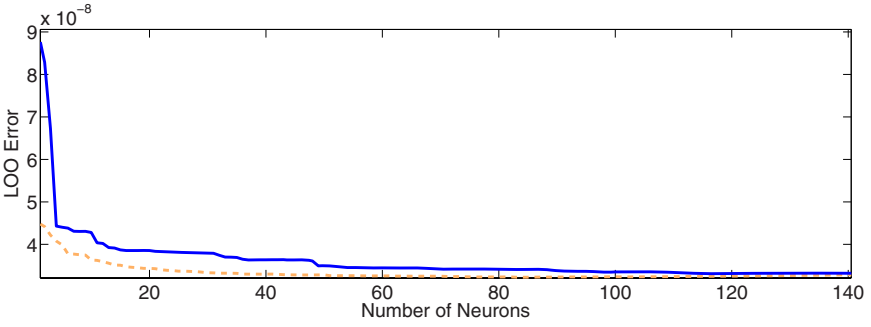

Fig. 2. Comparison of LOO error with and without the MRSR ranking. The solid blue line represents the LOO error without and the dashed orange one with the MRSR.

Indeed, the ELM is very fast, as can be seen in the original ELM paper [1] and in the experiments in Section 3. The ELM also has the advantage of being rather simple, for the process of training and initializing the neural network weights.

The variable ranking by the MRSR is also one of the fastest ranking methods providing the exact best ranking of the variables, since the problem is linear, when creating the neural network using ELM.

The linearity also enables the model structure selection step using the Leave-One-Out, which is usually very time-consuming. Thanks to the PRESS statistics formula for the LOO error calculation, the structure selection can be done in a reasonable time.

The final model structure selection for the OP-ELM model using the Ailerons dataset (see Section 3) is shown in Figure 2.

It can be seen from Figure 2 that the OP-ELM benefits greatly from the MRSR ranking step of its methodology. The convergence is faster and the LOO error gets smaller with respect to the number of neurons when the MRSR is used than when it is not.

3 Experiments

This section demonstrates the speed and accuracy of the OP-ELM method using several different regression and classification datasets. For the comparison, Section 3.2 provides also the performances using a well-known MultiLayer Perceptron (MLP) [8] and Least-Squares Support Vector Machine (LSSVM) [9] implementations. Following subsection shows a toy example to illustrate the performance of OP-ELM on a simple case that can be plotted.

3.1 Toy Example: Sum of Two Sines

A set of 1000 training points are generated following a sum of two sines. This gives a one-dimensional example, where no feature selection has to be performed. Figure 3 plots the obtained model on top of the training data.

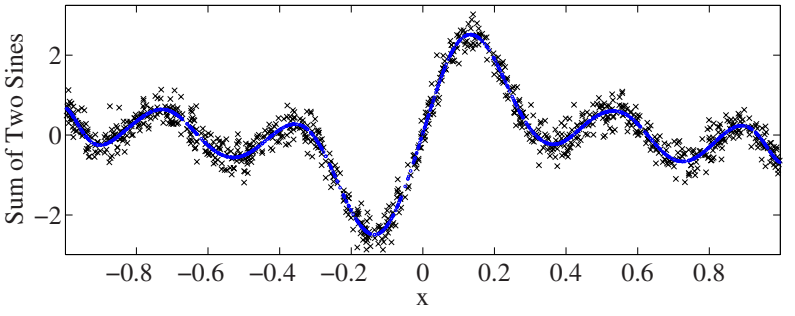

Fig. 3. Plot of a one dimensional sum of sines as black crosses and the model obtained by OP-ELM as blue circles

The model approximates the data very nicely.

3.2 Real Measurement Datasets

For the comparison of the three methods, we selected several different datasets: eight regression and four classification problems. For each dataset, all three methods are applied and the performances compared.

Each dataset is divided into two sets, train and test sets. The trainset includes two thirds of the data, selected randomly without replacement, and the testset one third. Table 1 shows some key information about the datasets and the selected hyperparameters for the LS-SVM and the MLP methods.

Table 1. Key information about the selected datasets and the selected hyperparameters for the LS-SVM and the MLP. For the classification, the variables column also includes the number of classes in the dataset.

		Samples		LS-SVM		MLP
Regression	Variables	Train	Test	Gamma	Sigma	Neurons
Abalone	8	2784	1393	3.23	7.73	2
Ailerons	5	4752	2377	3.78	5.06	9
Elevators	6	6344	3173	2.2	7.27	10
Auto Price	15	106	53	621.84	5.43	7
Servo	4	111	56	211.94	4.19	7
Breast Cancer	32	129	65	18.99	4.94	2
Bank	8	2999	1500	1099	4.92	6
Stocks	9	633	317	67.39	2.3	12
Classification						
Iris	4 / 3	100	50	151.53	1.24	4
Wisconsin Breast Cancer	30 / 1	379	190	1169	6.07	1
Pima Indians Diabetes	8 / 1	512	256	57.88	2.38	1
Wine	13 / 3	118	60	14.54	7.21	7

The hyperparameters for the LS-SVM and the MLP are selected using a 10-fold Cross-Validation. The LS-SVM is performed using the LS-SVM toolbox [9] with the default settings for the hyperparameters and the grid search. The MLP is performed using a Neural Network toolbox, which is a part of the Matlab software from the Mathworks. The training of the MLP is performed using the Levenberg-Marquardt backpropagation.

In order to decrease the possibility of local minima with the MLP, the training is repeated 10 times for each fold and the best network according to the training error is selected for validation. For example, in order to validate the MLP network using 12 hidden neurons, we have to train a total of 100 MLP networks with 12 hidden neurons to evaluate the validation error. This procedure is done for each number of hidden nodes from 1 to 20 and the selected number according to the validation MSE is selected.

Table 2 shows the results of the validation for each method. Also included is the respective calculation time consumed when using similar computer systems in calculations.

Table 3 shows the results of the test for each method.

Table 2. Validation errors for each method and the calculation times in seconds

	Validation Error			Calculation Time		
Regression	LS-SVM	MLP	OP-ELM	LS-SVM	MLP	OP-ELM
Abalone	4,42	**4,16**	4,35	9,36E+04	2640	25
Ailerons	2,80E-08	**2,76E-08**	2,77E-08	1,40E+05	6360	500
Elevators	2,92E-06	**1,99E-06**	2,00E-06	1,04E+06	4080	1250
Auto Price	2,13E+07	8,46E+06	**5,21E+06**	660	720	0,015
Servo	0,598	**0,361**	0,506	480	480	0,34
Breast Cancer	1301	1514	**1164**	900	1500	0,22
Bank	2,45E-03	**8,93E-04**	1,00E-03	1,21E+06	3360	54
Stocks	**0,485**	0,878	0,819	720	1320	3,8
Classification						
Iris	0,923	**0,980**	0,950	300	540	0,13
Wisconsin Breast Cancer	0,953	**0,966**	0,958	960	2340	2,82
Pima Indians Diabetes	0,744	**0,777**	0,775	600	600	172
Wine	0,972	**0,983**	**0,983**	420	900	0,41

From Tables 3 and 3 we can see that in general, the OP-ELM is on the same performance level than the other methods. On some datasets, the method performs worse and on some, better than the LS-SVM or the MLP.

On all the datasets, however, the OP-ELM method is clearly the fastest, with several orders of magnitude. For example, in the Abalone dataset using the OP-ELM is more than 3700 times faster than the LS-SVM and roughly 100 times faster than the MLP.

Finally, it should be noted that considering the very long computational time, which relates to the complexity of the problem in the LS-SVM case, some of the bad results obtained when using LS-SVM can be due to the fact that the

Table 3. Test errors for each method

Regression	LS-SVM	MLP	OP-ELM
Abalone	4,45	**4,34**	4,58
Ailerons	2,82E-08	**2,64E-08**	2,69E-08
Elevators	2,87E-06	**2,11E-06**	**2,11E-06**
Auto Price	2,25E+07	**1,02E+07**	5,91E+06
Servo	0,644	**0,565**	0,589
Breast Cancer	907	1033	**670**
Bank	2,47E-03	**9,05E-04**	1,00E-03
Stocks	**0,419**	0,764	0,861
Classification			
Iris	0,870	0,940	**0,980**
Wisconsin Breast Cancer	0,927	0,947	**0,968**
Pima Indians Diabetes	0,687	**0,769**	0,754
Wine	0,950	**0,967**	0,950

algorithm might not have converged properly. This might explain the results on Bank and Elevators datasets.

4 Conclusions

In this paper we have demonstrated the speed and accuracy of the OP-ELM methodology. Comparing to two well-known methodologies, the LS-SVM and the MLP, the OP-ELM achieves roughly the same level of accuracy with several orders of magnitude less calculation time.

Our goal is not to prove that the OP-ELM provides the best results in terms of the MSE, but instead to show that it provides very accurate results very fast. This makes it a valuable tool for applications, which need a small response time and a good accuracy. Indeed, the ratio between the accuracy and the calculation time is very good.

For further work, the comparisons with other methodologies are performed in order to verify the applicability and accuracy of the OP-ELM with different datasets.

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. Miller, W.T., Glanz, F.H., Kraft, L.G.: Cmac: An associative neural network alternative to backpropagation. Proceedings of the IEEE 70, 1561–1567 (1990)
3. Rao, C.R., Mitra, S.K.: Generalized Inverse of Matrices and Its Applications. John Wiley & Sons, Chichester (1972)
4. Similä, T., Tikka, J.: Multiresponse sparse regression with application to multidimensional scaling. In: Duch, W., Kacprzyk, J., Oja, E., Zadrożny, S. (eds.) ICANN 2005. LNCS, vol. 3697, pp. 97–102. Springer, Heidelberg (2005)

5. Efron, B., Hastie, T., Johnstone, I., Tibshirani, R.: Least angle regression. Annals of Statistics 32, 407–499 (2004)
6. Myers, R.: Classical and Modern Regression with Applications, 2nd edn. Duxbury, Pacific Grove (1990)
7. Bontempi, G., Birattari, M., Bersini, H.: Recursive lazy learning for modeling and control. In: European Conference on Machine Learning, pp. 292–303 (1998)
8. Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall, Englewood Cliffs (1998)
9. Suykens, J., Gestel, T.V., Brabanter, J.D., Moor, B.D., Vanderwalle, J.: Least-Squares Support-Vector Machines. World Scientific, Singapore (2002)
10. Lendasse, A., Sorjamaa, A., Miche, Y.: OP-ELM Toolbox, `http://www.cis.hut.fi/projects/tsp/index.php?page=research&subpage=downloads`
11. Whitney, A.W.: A direct method of nonparametric measurement selection. IEEE Transactions on Computers C-20, 1100–1103 (1971)

Appendix: Toolbox

A small overview of the OP-ELM Toolbox [10] is given in this appendix. Users can also refer to the toolbox documentation provided within the OP-ELM Toolbox package at the address: `http://www.cis.hut.fi/projects/tsp/index.php?page=research&subpage=downloads`. Main functions with their use and arguments are first listed, followed by a small example.

`gui_OPELM`

This command invokes the Graphical User Interface (GUI) for the OP-ELM Toolbox commands (quickly reviewed in the following). It can be used for typical problems. Figure 4 shows a snapshot of the GUI.

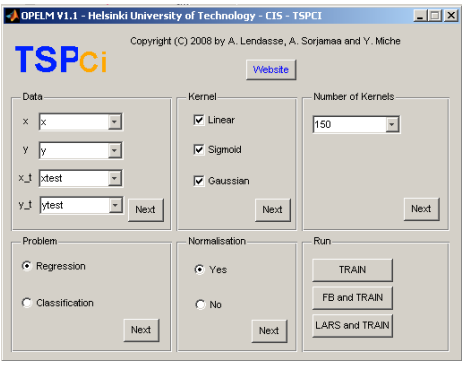

Fig. 4. Snapshot of the OP-ELM toolbox GUI

`train_OPELM`

This function trains a model using OP-ELM and the proper inputs.

```
function [model]=train_OPELM(data,[kernel],[maxneur],[problem],
                             [normal],[KM])
Inputs:   data       the data (can be multi-output)
          [kernel]   (optional) is the type of kernels to use.
                     Either 'l' (linear), 's' (sigmoid), 'g'
                     (gaussian), 'ls' (linear+sigmoid), 'lg'
                     (linear+gaussian) or 'lsg' (lin+sig+gauss).
          [maxneur]  (optional) maximum number of neurons model.
          [problem]  (optional) Either 'r' (regression) or 'c'
                     (classification).
          [normal]   (optional) Normalize data or not.
          [KM]       (optional) specifies a previously computed
                     Kernel Matrix to be used as initialization
                     of the model.
Output:   [model]    the obtained model.
```

`sim_OPELM`

This function uses the model computed by `train_OPELM` on a test dataset.

```
function [yh,error]=sim_OPELM(model,datatest) Inputs:   model
is the model previously obtained by the
                   train_OPELM function.
                   datatest  is the test data.
Outputs:  yh       the estimated output by the model.
          error    the mean square error (for regression problem)
                   or classification error with confusion matrix
                   (for classification problem).
                (if real output is known).
```

`LARS_Selection_OPELM`

This function uses the MRSR algorithm [4] with OP-ELM algorithm.

```
function myinputs=LARS_Selection_OPELM(data,[kernel],[maxneur],
                                       [problem],[normal])
Inputs:  same as train_OPELM Output:  myinputs   a 1xd matrix of
'1' (for selected variables)
                   and '0' (for the unselected).
```

`FB_OPELM`

This function uses the Forward-Backward [11] algorithm with OP-ELM algorithm.

```
function myinputs=FB_OPELM(data,[input_init],[kernel],[maxneur],
                                        [problem],[normal])
Inputs: same as train_OPELM
        [input_init] (optional) is an initialization of the input
                     selection to be used for the Forward-Backward
                     algorithm. Specified as a 1xd matrix of '0'
                     (if the considered variable is not to
                     be taken) and '1' (if it is to be taken).
Ouput:  myinputs     a 1xd matrix of '1' (for selected variables)
                     and '0' (for the unselected).
```

Example of use

```
>> data.x = (-1:0.001:1)';
>> data.y = 2*sin(8*data.x)+5*sin(3*data.x)+0.5*randn(2001,1);
>> datatest.x = (-1:0.0001:1)';
>> datatest.y = 2*sin(8*datatest.x)+5*sin(3*datatest.x)+0.5*randn(20001,1);
```

Create some data...

Train a model on the data with default parameters...

```
>> [model] = train_OPELM(data);
Warning: normalization unspecified...
-------> Switching to forced normalization.
Warning: problem unspecified...
-------> Switching to regression problem.
Warning: kernel type unspecified...
-------> Switching to lsg kernel.
Warning: maximum number of neurons unspecified...
-------> Switching to 100 maximum neurons.
```

Display some info about the built model...

```
>> show_model(model)
Model for regression, build on 2001x1 data, one dimensional
output. Uses 16 neurons; LOO error is 2.522819e-01.
```

Use the model on test data...

```
>> [yth,error] = sim_OPELM(model,datatest);
>> error
error = 0.2522
```