A Connection between Extreme Learning Machine and Neural Network Kernel

Eli Parviainen and Jaakko Riihimäki

BECS, Aalto University, P.O. Box 12200, FI-00076 AALTO, Finland

Abstract. We study a connection between extreme learning machine (ELM) and neural network kernel (NNK). NNK is derived from a neural network with an infinite number of hidden units. We interpret ELM as an approximation to this infinite network. We show that ELM and NNK can, to certain extent, replace each other. ELM can be used to form a kernel, and NNK can be decomposed into feature vectors to be used in the hidden layer of ELM. The connection reveals possible importance of weight variance as a parameter of ELM. Based on our experiments, we recommend that model selection on ELM should consider not only the number of hidden units, as is the current practice, but also the variance of weights. We also study the interaction of variance and the number of hidden units, and discuss some properties of ELM, that may have been too strongly interpreted previously.

Keywords: Extreme learning machine, ELM, Neural network kernel.

1 Introduction

Extreme Learning Machine [1] (ELM) is a currently popular neural network architecture based on random projections. It has one hidden layer with random weights, and an output layer whose weights are determined analytically. Both training and prediction are fast compared with many other nonlinear methods.

We study a connection between ELM and neural network kernel (NNK). NNK is a kernel that is often used with Gaussian processes [2]. It is derived by assuming a neural network with an infinite number of hidden units, and integrating out the network weights. We interpret ELM as an approximation to the infinite network used in this derivation.

We show that ELM and NNK can, to certain extent, replace each other in computations. The output of the hidden layer of ELM can be used to form a kernel [3] for kernel classifiers; this observation has been the main inspiration for our work. We experimentally show that this kernel approaches NNK when the number of hidden units grows. On the other hand, we can decompose the NNK matrix into a possible set of feature vectors, and use these vectors instead of the hidden layer of ELM.

The connection between ELM and NNK leads us to question the current practice of parameterizing ELM. Usually, only the number of hidden units is carefully selected, and a fixed variance or range parameter is used for the distribution from which the random weights are drawn. In NNK, the weights are integrated out, so that the only parameter left is the variance of the weights. In ELM, the individual weights have little meaning since they are random, but the variance of the distribution from which they are drawn should matter, just like it does in NNK. We study the effect of the weight variance parameter in ELM, finding that it affects the predictions. We therefore think that the model selection for ELM should consider both the variance and the number of hidden units.

Sect. 2 introduces ELM, NNK and the data sets we use. In Sect. 3 we motivate out work by noting how the random nature of ELM makes ELM resemble kernel methods, although it is usually thought as a neural network method. In Sect. 4 we discuss the connection between ELM and NNK in detail. Sect. 5 concentrates on the effects of variance parameter, and Sect. 6 studies the interaction of the weight variance and the number of hidden units. We finish by discussing our results and ELM in Sect. 7.

2 Methods and Data

2.1 Extreme Learning Machine

Extreme Learning Machine [1] is a feedforward neural network which consists of one hidden layer of sigmoid units, and one (linear) output layer. The key idea of ELM is the way the network weights are determined. The weights of the hidden layer are randomly chosen and not trained at all. The output weights are calculated analytically to predict a target variable from the hidden layer outputs. This makes training a single ELM network simple and fast. The price to pay is variation of results due to randomness in weights. For assessing average quality of results and their uncertainty, repeated runs are needed.

Several variants of ELM have been proposed, many of them incorporating some weak or indirect form of training in the first layer. For example, the least useful hidden units can be pruned [4], or the best of several ELM can be chosen for use in an evolutionary algorithm [5]. We use the plain ELM of [1] in this work.

The weights $\boldsymbol{w}_h$ and biases $\boldsymbol{b}_h$ of the hidden layer must be drawn from some continuous distribution [1]. Often a uniform distribution, spread over a fixed interval, is used. The width of the interval is not seen as a model parameter, but it is simply a constant that must be suitably fixed to guarantee that the sigmoid operation neither remains linear nor saturates to ± 1. In this work we use zero-mean Gaussian distributions, parameterized with variance σ^2. We use equal variance for all dimensions of input data. Gaussian was chosen because it is used in the derivation of the neural network kernel (Sect. 2.2).

ELM places few restrictions to the activation of hidden units. In this work we use the error function $f(z) = 2/\sqrt{\pi} \int_0^z \exp(-t^2) \mathrm{d}t$. Also this choice is motivated by its correspondence with the derivation of NNK. In [1] it is shown that, with a large enough number of hidden units, it is possible to achieve arbitrarily small training error, on the condition that the hidden unit sigmoids are infinitely differentiable. The error function fullfills this condition, so using it instead of the more common logistic or tanh sigmoids does not diminish the representational power of ELM.

The weights $\boldsymbol{\beta}$ of the output layer are determined by fitting a linear model to the outputs of the hidden layer. The outputs are collected into a matrix $\mathbf{F}$ with entries $[\mathbf{F}]_{ih} = f(\boldsymbol{w}_h^T \boldsymbol{x}_i + \boldsymbol{b}_h)$, with i indexing the data points $\boldsymbol{x}_i$ and h indexing the hidden units. The weights $\boldsymbol{\beta}$ are found from

$$\mathbf{F}\boldsymbol{\beta} = \mathbf{Y} \ , \tag{1}$$

where $\mathbf{Y}$ is the matrix or vector of target variables. We use ELM for binary classification, in which case $\mathbf{Y}$ is a vector with entries ± 1. A least squares estimate for $\boldsymbol{\beta}$ is obtained as

$$\boldsymbol{\beta} = \mathbf{F}^{\dagger}\mathbf{Y} \ , \tag{2}$$

where $\mathbf{F}^{\dagger}$ denotes the Moore-Penrose pseudoinverse [6] of $\mathbf{F}$.

The actual classification happens by thresholding the network output at zero. To our knowledge nothing would prevent use of sigmoid outputs, so the outputs would be class probabilities instead of binary labels. In such case, the linear model from the hidden layer to the targets would be replaced by a generalized linear model [7]. Sigmoid outputs are common in other neural network models. For simplicity, we stick to the thresholding that is common for ELM.

2.2 Neural Network Kernel

Neural network kernel is derived in [8] by letting the number of hidden units in the hidden layer of a neural network go to infinity. A Gaussian prior is set to the hidden layer weights, which are then integrated out. The only parameters remaining after the integration are the variances of the weight priors. This leads to an analytical expression for the expected covariance between two feature space vectors,

$$\mathrm{k}_{\mathrm{NN}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{2}{\pi} \sin^{-1} \left(\frac{2\tilde{\boldsymbol{x}}_i^T \Sigma \tilde{\boldsymbol{x}}_j}{\sqrt{(1 + 2\tilde{\boldsymbol{x}}_i^T \Sigma \tilde{\boldsymbol{x}}_i)(1 + 2\tilde{\boldsymbol{x}}_j^T \Sigma \tilde{\boldsymbol{x}}_j)}} \right) \ . \tag{3}$$

Above, $\tilde{\boldsymbol{x}}_i = [1 \ \boldsymbol{x}_i]$ is an augmented input vector and Σ is a diagonal matrix with variances of inputs. In this work all variances are assumed equal.

NNK also arises as a special case of a more general arc-cosine kernel [9]. NNK should not be confused with $\tanh$-kernel $\tanh((x_i \cdot x_j) + b)$, which is sometimes called MLP kernel (from Multi-Layer Perceptron) because of its connection to neural networks [10].

2.3 Data Sets

For comparing ELM and NNK we use six binary classification data sets from UCI machine learning repository [11]: Arcene, US votes, WDBC, Pima, TicTacToe and Internet ads. Their properties are summarized in Table 1. For representative results, the data sets were chosen to have different sample sizes and different dimensionalities.

In addition, we use an artificial data set Saturni, illustrated in Fig. 1. It consists of four spheres in random locations in a three-dimensional space. Each sphere is surrounded by a ring, whose points have a different class label than the points of the sphere. The data set is highly nonlinear, but easy to classify with a nonlinear model.

3 ELM Falls between Neural Networks and Kernels

Extreme learning machine, although introduced as a fast method for training a neural network, is in some sense closer to a kernel method in its operation. A fully trained neural network has learned a mapping such that the weights contain information about the

Table 1. Data sets for the experiments in Sects. 4 and 5

name	# samples	# dims	data types
Arcene [12]	200	10000	continuous
US votes	435	16	binary
WDBC	569	30	continuous
Pima	768	8	continuous
Tic Tac Toe	958	27	categorical
Internet ads	2359	1558	continuous, binary

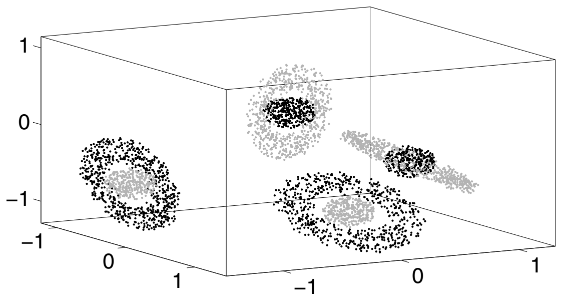

Fig. 1. The Saturni data set that is used in the experiments in Sect. 6

training data. ELM uses a fixed mapping from data to a feature space. This is similar to a kernel method, except that instead of some theoretically derived kernel, the mapping ELM uses is random.

The ability to learn features on data is an essential property of a fully trained neural network. Features should be good for predicting the target variable of a classification/regression task. In a network with one hidden and one output layer, the hidden layer learns the features, while the output layer learns a linear mapping. We can think of this as first non-linearly mapping the data into a feature space and then performing a linear regression/classification in that space.

ELM has no feature learning ability. It projects the input data into whatever feature space the randomly chosen weights happen to specify, and learns a linear mapping in that space. Parameters affecting the feature space representation of a data point are the type and number of neurons, and the variance of hidden layer weights. Training data can affect these parameters through model selection, but not directly through any training procedure.

This is similar to what a support vector machine does. A feature space representation for a data point is derived, using a kernel function with a few parameters, which are typically chosen by some model selection routine. Features are not learned from data, but dictated by the kernel. Weights for linear classification or regression are then learned in the feature space. The biggest difference is that where ELM explicitly generates the feature space vectors, in SVM or in another kernel method only similarities between feature space vectors are used.

Seeing ELM as a neural network method with kernel-like behavior raises a question about a connection between ELM and the neural network kernel. In the following section we will interpret ELM as an approximation to NNK.

4 ELM Feature Space Approximates NNK Feature Space

NNK is derived from a neural network with an infinite number of hidden units. We interpret ELM as an approximation to this infinite neural network. In this section to study the connection of NNK and ELM from two opposite points of view. First, we use ELM as a kernel in a kernel classifier. Second, we show how to decompose the NNK matrix and use the results to replace the hidden layer of ELM.

4.1 ELM Kernel Replacing NNK

Authors of [3] propose using ELM hidden layer to form a kernel to be used in SVM classification. They define ELM kernel function (the notation is ours) as

$$\mathrm{k}_{\mathrm{ELM}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{H} \sum_{h=1}^{H} f(\boldsymbol{w}_h^T \boldsymbol{x}_i + \boldsymbol{b}_h) f(\boldsymbol{w}_h^T \boldsymbol{x}_j + \boldsymbol{b}_h) \ , \tag{4}$$

that is, the data is fed trough the ELM hidden layer to obtain the feature space vectors, and their covariance is then computed and scaled by the number of hidden units H.

When the number of hidden units grows, this kernel matrix approaches the NNK matrix. Fig. 2 shows the approach, measured by Frobenius norm, as function of H. Especially for small H the ELM kernel varies due to random weights, but it converges towards NNK.

Experiment. We use ELM kernel in a Gaussian Process classifier [2]. The experiment is implemented by modifying the GPstuff toolbox[1]. Expectation propagation [13] is used for Gaussian process inference. The data is split into train and test sets (50 % / 50 %). Zero-mean unit-variance normalization is used for the data. We are more interested in ELM behavior as function of hidden units than the prediction accuracy. We therefore only consider variation from the random weights (using 30 repetitions) and do not repeat over splits of data.

Results of the experiment are shown and compared to NNK results in Fig. 3. ELM kernel behavior in GP seems qualitatively similar to that observed in [3] for SVM. The classification accuracy first rises rapidly and then sets as a fixed level. Variation due to random weights remains, but NNK result stays inside the 95% interval of ELM.

4.2 NNK Replacing ELM Hidden Layer

We use NNK to replace the hidden layer computations in ELM. This is done by first forming the NNK matrix, and then deriving a possible set of explicit feature space vectors by matrix decomposition. This corresponds to using ELM with an infinite number of hidden units.

[1] http://www.lce.hut.fi/research/mm/gpstuff/

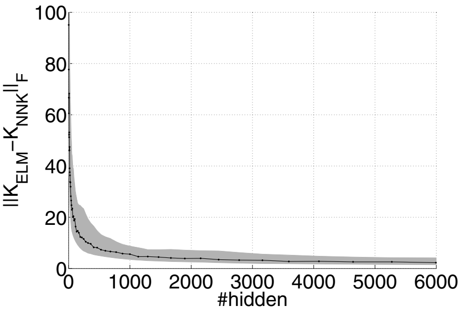

Fig. 2. ELM kernel (K_{ELM}) approaches neural network kernel (K_{NN}) in Frobenius norm when number of hidden unit grows. Mean by black dots, 95% interval by shading. Variation is caused by randomness in weights. WDBC data set.

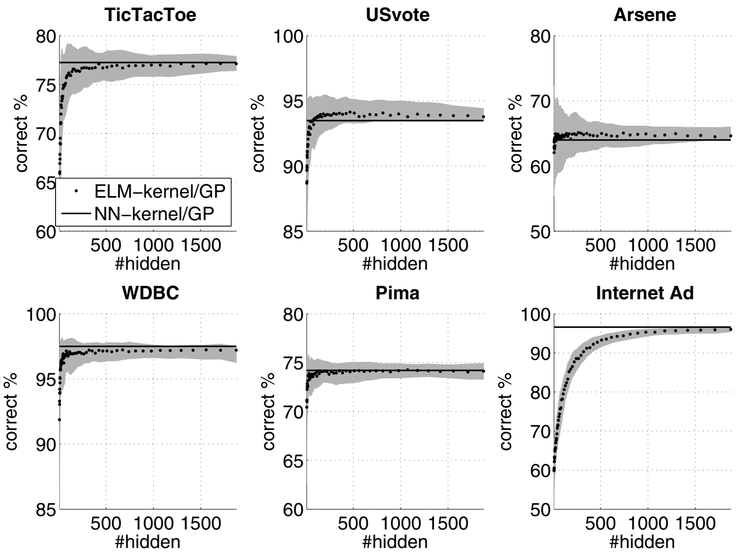

Fig. 3. Mean accuracy of GP classification when using ELM kernel (black dots and shading) versus NNK accuracy (horizontal line). The shading indicates variability due the repeated initializations of the ELM kernel (95 % interval).

Derivation. When using ELM, we only deal with vectorial data, with data space vectors transformed into feature space vectors by the hidden layer. Kernel methods rely on pairwise data, where only similarities from any point to all training points are considered. The kernel matrix specifies the pairwise similarities. In order to use pairwise information from the NNK instead of the ELM hidden layer, we must find a vectorial representation for the data. The problem of recovering points given their mutual relationships is old [14], and can be solved by a decomposition of the pairwise matrix.

NNK is derived as a covariance, and is therefore positive semidefinite (PSD). Any PSD matrix can be decomposed into a matrix and its Hermitian conjugate

$$\mathbf{C} = \mathbf{L}\mathbf{L}^H \ . \tag{5}$$

There are different methods for finding the factors [15]. Matlab `cholcov` implements a method based on eigendecomposition. If we take $\mathbf{C}$ in (5) to be the output of the NNK function (3), then $\mathbf{L}$ can be thought as one possible set of corresponding feature space vectors.

We use $\mathbf{L}$ to determine the output layer weights the same way we used the ELM features $\mathbf{F}$ in (2),

$$\boldsymbol{\beta} = \mathbf{L}^{\dagger}\mathbf{Y} \ . \tag{6}$$

The factors $\mathbf{L}$ are unique only up to a unitary transformation, but this is not a problem in ELM context, as the linear fitting of output weights is able to adapt to linear transformations.

With an infinite number of hidden units, the feature space is infinite-dimensional. Meanwhile, the data we have available is finite, and the N data points can span at most an N-dimensional subspace. Thus the maximum size of $\mathbf{L}$ is $N \times N$; the number of columns can be smaller if the data has linear dependencies.

If $\mathbf{C}$ is positive definite or close to it, a triangular $\mathbf{L}$ could be found using Cholesky decomposition, leading to fast and stable matrix operations when finding the output layer weights. Since positive definiteness cannot be guaranteed, we use the more general decomposition for PSD matrices in all cases.

The one remaining problem is to map the test points to the feature space. In ELM, the test data is simply fed through the hidden layer. In our case, the hidden layer does not physically exist, and we must base the calculations on similarities from the test points to the training points, as given by NNK (3). This means that NNK output for test data $\mathbf{C}_*$ is a covariance matrix of the form

$$\mathbf{C}_* = \mathbf{L}\mathbf{L}_*^H \ . \tag{7}$$

We already know the pseudoinverse of $\mathbf{L}$. Therefore $\mathbf{L}_*$ is recovered from

$$\mathbf{L}_* = (\mathbf{L}^{\dagger}\mathbf{C}_*)^H = (\mathbf{L}^{\dagger}\mathbf{L}\mathbf{L}_*^H)^H \ , \tag{8}$$

and the predictions for the test targets are computed as

$$\mathbf{Y}_* = \mathbf{L}_*\boldsymbol{\beta} \ . \tag{9}$$

Experiment. We compare ELM classification accuracy (as function of number of hidden units) to the accuracy obtained by replacing the ELM hidden layer with NNK. The latter variant of ELM is from now on referred to as NNK-ELM. The comparison is done for five different variances ($\sigma \in \{0.1, 0.325, 0.55, 0.775, 1\}$). The authors' Matlab implementation[2] is used.

The data is scaled to range $[-1, 1]$. Each data set is divided into 10 parts. Nine parts are used for training and one for testing, repeating this 10 times. This variation from data

[2] Available from http://www.becs.tkk.fi/∼eiparvia/matlabcode.html

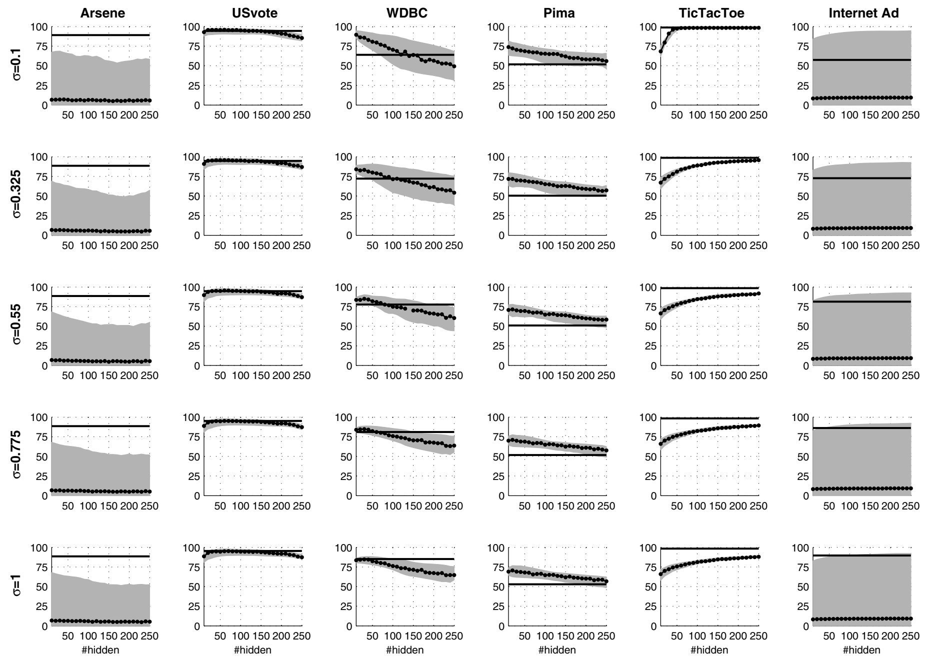

Fig. 4. ELM results (mean as black dots, 95 % interval as shading) for different values of σ. Mean of NNK results (horizontal line) are shown for comparison.

is shown in figures. ELM results have another source of variation, the random weights. This is handled by repeating the runs 10 times, each time drawing random weights, and averaging over results. The maximum number of hidden units is 250, to make sure to cover the sensible operating range of ELM (up to N hidden units) for all data sets.

The predictions given by the ordinary ELM are shown in in Fig. 4, and the mean predictions of NNK-ELM are included for comparison. We notice that when the variance is properly chosen, using NNK gives equal or better results than ELM for most data sets. Pima data set is an exception, ELM has some predictive power whereas NNK-ELM performs almost at the level of guessing.

5 Variance of Weights Affects ELM Predictions

An infinite network performing equally well or often better than ELM raises a question about meaningfulness of choosing model complexity based on hidden units only, as is traditionally done with ELM. NNK is parameterized by the variance of weights. The network has an infinite number of hidden units, and when the weights are integrated out, the values of the individual weights become meaningless. Also in ELM, the essential information about the weights is captured by their variance, since the individual, random weights are not meaningful as such.

We argue, and support the argument by experiments, that the variance of hidden layer

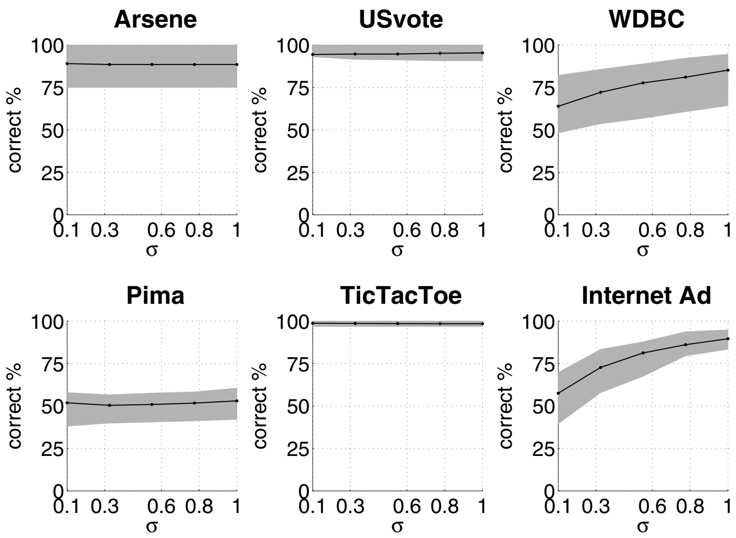

Fig. 5. NNK-ELM results, mean and 95 % interval due to data variation

weights is an important parameter in ELM as well as in NNK. It should therefore be considered when doing model selection on ELM.

NNK-ELM results are shown in Fig. 5, as function of σ. We notice that the choice of variance has a marked effect on two and some effect on other data sets, both for ordinary ELM and the NNK variant.

Fig. 6 summarizes the variance effects from Fig. 4. The mean predictions of ELM are shown as function of H. For TicTacToe and WDBC data sets the predictions are clearly affected by the variance parameter. For Internet ad data the overall effect of both H and σ is very small. In that scale, the smallest variance nonetheless gives results clearly different than larger values. Results for other data sets are not very sensitive to the variance values that were tried. For TicTacToe and Internet ads smaller variance gives better predictions, for WDBC the biggest one. Clearly no fixed sigma can be used for all data sets.

When thinking about the mechanism by which the variance parameter affects the results, differences between data sets are to be expected. Variance affects model complexity, and obviously different models fit different data sets. Variance and distribution of the data together determine the magnitude of values seen by the activation function. This is illustrated in Fig. 7. One-dimensional data points, spread over range [-1,1] (the x-axis), are given random weights drawn from a zero-mean Gaussian distribution and then fed through an error function sigmoid (denoted *erf* in the figure), repeating this 10000 times. Mean output and 95 % interval are depicted. On average, the sigmoid produces a zero response, but the distribution of responses is determined by the variance used. Small variance means mostly small weights, and linear operation. Large variance produces many large weights, which increase the proportion of large responses by the network, allowing nonlinear mappings.

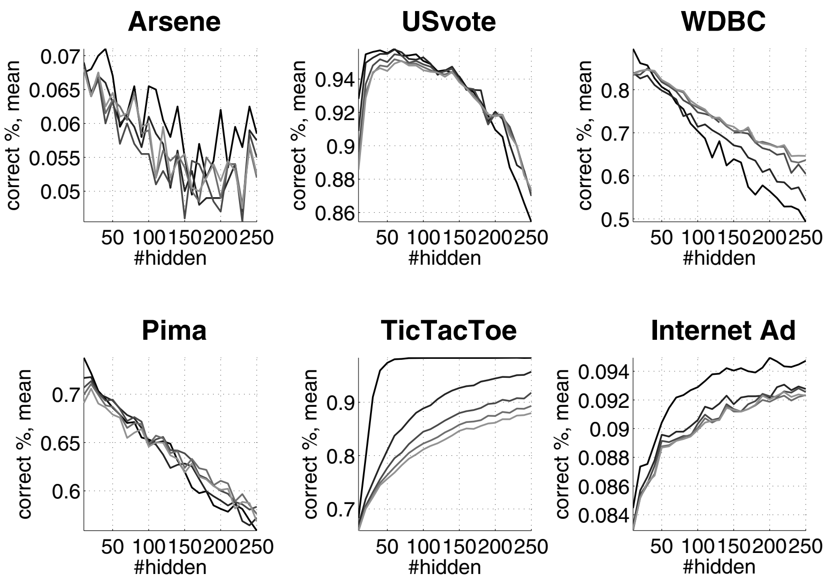

Fig. 6. Effect of variance on mean of ELM predictions. Darker shade indicates smaller variance.

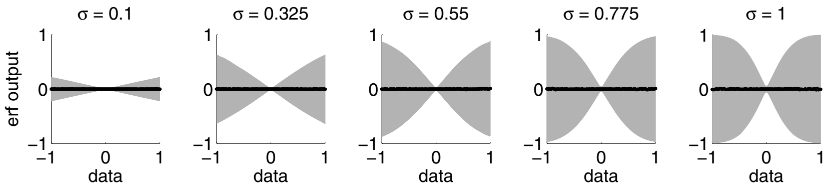

Fig. 7. Distributions of predictions of an error function sigmoid for different σ

6 Variance and the Number of Hidden Units Interact

In [16] we recognized the importance of weight variance as a parameter of ELM, and noted, that we now have two parameters that both affect the nonlinearity of the model. We did not, however, study their relative importance or joint effects. In this section we look at the interaction of the two parameters.

We use the artificial Saturni data set, from which we can easily draw repeated samples of different sizes. The size of the training set varies, and the test data always has 500 samples. The results are averaged over 10 ELM initializations and 10 samples drawn from the Saturni.

Fig. 8 shows the prediction accuracy of ELM for data sets of 250, 500, 750 and 1000 points, as a contour plot over a grid of weight variances and numbers of hidden units.

In the direction of hidden units, the plots seem to divide in three regions with qualitatively different behavior for the variance parameter. Left, where the number of hidden units is small, the variance has very little effect. Right, with a large number of hidden units, very small and very large variances give better accuracy than the medium-sized ones. The area in between has more complicated behavior of the variance. Generally, a large variance gives best results, but the area of optimal or almost optimal accuracy stretches towards lower variances for some values of the number of hidden units. The variance also seems to have an upper limit beyond which the accuracy decreases.

These differences can be related to network capacity in relation to the phenomenon to be modeled. Following complicated class boundaries requires a sufficient number of hidden units. On the other hand, if there are too many hidden units, the model becomes so flexible that it can overfit to the training data.

The left region is an area, where the number of hidden units is too small to capture the nonlinearities of the data. On the right, the model has started to overfit. This explains, why a small variance improves performance: it causes most of the weights to be small, making most of the hidden units fairly linear. This achieves a similar effect as regularizers have in traditional neural network models. Our tentative interpretation for why also high variances improve performance over the middle-sized ones is, that a large variance makes many hidden units saturate to -1 or +1. If such a unit happens to be located near the outskirts of the data, it gives equal outputs for most or all data points. Such a hidden unit is useless in classification, and therefore the effective number of hidden units is reduced by large variances.

The region of optimal predictions, between these two extremal areas, lies in a certain range of numbers of hidden units, but its exact location is determined by the variance. This calls for model selection of both the number of hidden units and the variance in this region.

Generally, it seems that adjusting the variance and adjusting the network size can be alternative means of reaching the same goal. If the number of hidden units is correctly chosen, the variance may vary a lot, although an optimal region exists. And, if the variance is small enough, almost any number of hidden units will perform well, provided that the model has enough degrees of freedom for handling the nonlinearities in the data.

7 Discussion

7.1 ELM and NNK

We introduced NNK-ELM as a way for studying the effect of infinite hidden units in ELM, but it can also find its use as a practical method.

Computational complexity of NNK-ELM corresponds to that of N-hidden unit ELM. The matrix decomposition required in NNK-ELM scales as $O(N^3)$. In practice ELM training is much faster, since the optimal number of hidden units is usually much less than N. However, the optimal value is usually found by model selection, necessitating several ELM runs. If the selection procedure also considers large ELM networks, choosing ELM over NNK-ELM does not necessarily save time. Prediction is faster with ELM than with NNK-ELM, if $H \ll N$.

Furthermore, NNK-ELM has only one parameter, the variance of the distribution of the hidden layer weights. If the number of data points is reasonably small, NNK-ELM can thus result in considerable time savings when doing model selection. A factor

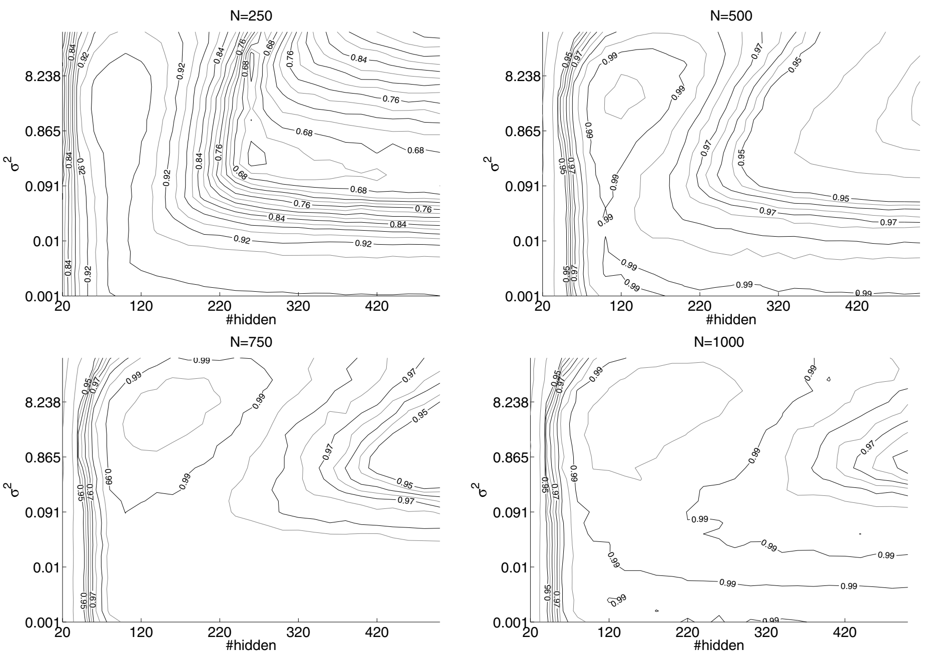

Fig. 8. ELM classification accuracy as function of the number of hidden units and of weight variance, for various N. Note the logarithmic scaling of the y-axis. The $N = 250$ plot has contour spacing of 0.02 throughout. In the other plots, contours are drawn at intervals of 0.1 below accuracy of 0.95 (this is to keep the narrow region of low accuracy a bit more readable), and at intervals of 0.005 above that.

adding to this is that, unlike ELM, NNK-ELM gives deterministic results, and only requires repetitions if variability due to training data is considered.

Use of ELM as a kernel, at least in a Gaussian process classifier, is likely to remain a curiosity. Classification performance seems to steadily increase as the number of hidden units grows, and, when considering the variation caused by ELM randomness, the performance does not exceed that of NNK. When an easy-to-compute, theoretically derived NNK function is available, we see no reason to favor a heuristical kernel, computation of which requires generating and storing random numbers and calculating an explicit mapping to a feature space.

7.2 Parameters of ELM

In our experiment with artificial data and large samples, repeating runs over both ELM initializations and data, it seemed that the weight variance and the number of hidden units may be alternative ways for controlling the nonlinearity of the model. This experiment probably captured the general principle of the interaction of the two parameters, although real data sets are usually not as easy to classify as the simple Saturni data.

If the number of hidden units is carefully chosen, the variance can vary a lot without the accuracy changing much. On the other hand, if a small variance is used, the number of hidden units may be almost arbitrary, provided it is large enough to capture the nonlinearities in the data. A small variance regularizes the model, and may prevent it from overfitting even if the number of hidden units would make overfitting possible.

Relying on a small variance and using an arbitrary but large number of hidden units has the problem, that an unnecessary large network size slows down the computations. Finding a model which is parsimonious with the hidden units and gives optimal accuracy required careful adjustment of both parameters with the artificial data. Also in the experiments with the real data sets, the weight variance had a noticeable effect on the results. We therefore think that ELM model selection should consider weight variance as an adjustable parameter.

Model selection with two parameters is slower than with only one, but since the two control the same property (model nonlinearity), some simplification may be possible. If the behavior seen in our artificial data generalizes to natural data sets, a full grid search over the ranges of both parameters may not be necessary. It appears like a reasonably safe strategy to first select a good range for hidden units, and inside that range, to perform full model selection to find a good variance.

7.3 On Properties of ELM

Authors of [1] promote ELM by speed, dependence on a single parameter, small training error and good generalization performance. These claims have often been repeated by subsequent authors, but we have not come upon much discussion of them. Here we present some comments on these properties.

Training of a single ELM network is fast, provided the number of hidden units is small. *Speed of training* as the whole, however, depends also on the number of training runs. Model selection may require considerable number of repetitions, since all sensible parameter combinations should be considered. Further, due to the random nature of ELM, any runs must be repeated several times if average performance is to be assessed.

Complexity of model selection is determined by the *number of parameters*, and the ranges used for them. One parameter is the weight variance that we discussed above. Another is the number of hidden units. The only theoretically motivated upper limit for the number of hidden units to try is N (which is enough for zero training error). At that limit, computing pseudoinverse corresponds to ordinary inversion of an $N \times N$ matrix, with a complexity of $O(N^3)$. In practice, smaller upper limits are used.

Generally, *small training error* and good generalization may be contradictory goals. ELM has been proved [1] to be able to perfectly classify the training data if the number of hidden units equals or exceeds the number of data points. This behavior, though important in proving the computational power of ELM, is usually not desirable in modeling. A model should generalize, not exactly memorize the training data. This view is indirectly acknowledged in practical ELM work, where the number of hidden units is much smaller than N. This may prevent the ELM network from overfitting to the training data, a factor usually not discussed in ELM literature.

The *generalization ability* of ELM is attributed to the fact that computing output layer weights by pseudoinverse achieves a minimum norm solution. This is motivated by a work of Bartlett [17], who showed the generalization ability of a neural network to relate to small norm of weights. However, Bartlett's work considers the neural network as whole, not only the output layer. Although ELM minimizes the norm of output layer weights, the norm of the hidden layer weights depends on the variance parameter, and does not change in ELM training.

In the hidden layer, generalization ability is related to the operating point of the hidden unit activations, discussed in Sect. 5. A model with small hidden layer weights is nearly linear, and generalizes well. A highly non-linear model, produced by large weights, is more prone to overfitting. Therefore, conclusions about generalization ability of ELM should not be based on the output weights only.

References

1. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: Theory and applications. Neurocomputing 70, 489–501 (2006)
2. Rasmussen, C.E., Williams, C.K.I.: Gaussian processes for machine learning. MIT Press (2006)
3. Frénay, B., Verleysen, M.: Using SVMs with randomised feature spaces: an extreme learning approach. In: Proc. of ESANN, pp. 315–320 (2010)
4. 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, 158–162 (2010)
5. Zhu, Q.Y., Qin, A.K., Suganthan, P.N., Huang, G.B.: Evolutionary extreme learning machine. Pattern Recognition 38, 1759–1763 (2005)
6. Penrose, R.: A generalized inverse for matrices. Mathematical Proceedings of the Cambridge Philosophical Society 51, 406–413 (1955)
7. McCullagh, P., Nelder, J.A.: Generalized linear models, 2nd edn. Monographs on statistics and applied probability, vol. 37. Chapman & Hall (1989)
8. Williams, C.K.I.: Computation with infinite neural networks. Neural Computation 10, 1203–1216 (1998)
9. Cho, Y., Saul, L.K.: Kernel methods for deep learning. In: Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., Culotta, A. (eds.) Proc. of NIPS, vol. 22, pp. 342–350 (2009)
10. Vert, J.P., Tsuda, K., Schölkopf, B.: A primer on kernel methods. In: Schölkopf, B., Tsuda, K., Vert, J.P. (eds.) Kernel Methods in Computational Biology, pp. 35–70. MIT Press (2004)
11. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
12. Guyon, I., Gunn, S.R., Ben-Hur, A., Dror, G.: Result analysis of the NIPS 2003 feature selection challenge. In: Proc. of NIPS (2004)
13. Minka, T.: Expectation propagation for approximate Bayesian inference. In: Proc. of UAI (2001)
14. Young, G., Householder, A.S.: Discussion of a set of points in terms of their mutual distances. Psychometrika 3, 19–22 (1938)
15. Golub, G.H., Van Loan, C.F.: Matrix computations, 3rd edn. The Johns Hopkins University Press (1996)
16. Parviainen, E., Riihimäki, J., Miche, Y., Lendasse, A.: Interpreting Extreme Learning Machine as an approximation to an infinite neural network. In: Proc. of KDIR. INSTICC (2010)
17. Bartlett, P.L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Transactions on Information Theory 44, 525–536 (1998)