INTERPRETING EXTREME LEARNING MACHINE AS AN APPROXIMATION TO AN INFINITE NEURAL NETWORK

Elina Parviainen, Jaakko Riihimäki
Dept. of Biomedical Engineering and Computational Science (BECS)
Aalto University School of Science and Technology, Finland

Yoan Miche, Amaury Lendasse
Dept. of Information and Computer Science (ICS)
Aalto University School of Science and Technology, Finland

Keywords: extreme learning machine (ELM), neural network kernel

Abstract: Extreme Learning Machine (ELM) is a neural network architecture in which hidden layer weights are randomly chosen and output layer weights determined analytically. We interpret ELM as an approximation to a network with infinite number of hidden units. The operation of the infinite network is captured by neural network kernel (NNK). We compare ELM and NNK both as part of a kernel method and in neural network context. Insights gained from this analysis lead us to strongly recommend model selection also on the variance of ELM hidden layer weights, and not only on the number of hidden units, as is usually done with ELM. We also discuss some properties of ELM, which may have been too strongly interpreted in previous works.

1 INTRODUCTION

Extreme Learning Machine (Huang et al., 2006) (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.

In this work we point out that ELM, 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 training data. ELM uses a fixed mapping from data to feature space. This is similar to a kernel method, except that instead of some theoretically derived kernel, the mapping ELM uses is random. Therefore, the individual weights of the ELM hidden layer have little meaning, and essential information about the weights is captured by their variance.

This thought is met also in derivation of the neural network kernel (NNK) (Williams, 1998), which is widely used in Gaussian process prediction. The network in the derivation has infinite number of hidden units, and when the weights are integrated out, the resulting function is parameterized in terms of weight variance.

We interpret ELM as an approximation to this infinite neural network. This leads us to study connections between NNK and ELM from two opposite points of view.

On the one hand, we can use ELM hidden layer to compute a kernel, and use it in any kernel method. This idea has been suggested for Support Vector Machines (SVM) in (Frénay and Verleysen, 2010), which has been the main inspiration for our work. We try out the same idea in Gaussian process classification.

On the other hand, we can use NNK to replace the hidden layer computations in ELM. This is done by first computing a similarity-based representation for data points using NNK, 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.

Our experiments show that the theoretically derived NNK can replace ELM when ELM is used as a kernel. NNK can also perform computations of the ELM hidden layer, albeit at higher computational cost. 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. We argue, and support the argument by experiments,

that the variance of hidden layer weights is an important tuning parameter of ELM. Model selection of weight variance should therefore be considered.

We first present in Section 2 some tools (ELM, NNK, data sets) which we will need in subsequent sections. Section 3 discusses similarities and differences between fully trained neural networks, ELM, and kernel methods, and reports the experiments with ELM kernel and computing ELM hidden layer results using NNK. Results of the experiments and properties of ELM are commented in length in Section 4.

2 METHODS

2.1 Extreme Learning Machine (ELM)

The following, including Algorithm 1, is an abridged and slightly modified version of ELM introduction in (Miche et al., 2010).

The ELM algorithm was originally proposed in (Huang et al., 2006) and it makes use of the Single Layer Feedforward Neural Network (SLFN). The main concept behind the ELM lies in the random choice of the SLFN hidden layer weights and biases. The output weights are determined analytically, thus the network is obtained with very few steps and with low computational cost.

Consider a set of N distinct samples $(\mathbf{x}_i, \mathbf{y}_i)$ with $\mathbf{x}_i \in \mathbb{R}^{d_1}$ and $\mathbf{y}_i \in \mathbb{R}^{d_2}$; then, a SLFN with H hidden units is modeled as the following sum

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

with f being the activation function, $\mathbf{w}_i$ the input weights, b_i the biases and β_i the output weights.

In the case where the SLFN perfectly approximates the data, the errors between the estimated outputs $\hat{\mathbf{y}}_i$ and the actual outputs $\mathbf{y}_i$ are zero and the relation is

$$\sum_{i=1}^{H} \beta_i f(\mathbf{w}_i \mathbf{x}_j + b_i) = \mathbf{y}_j, j \in [1,N], \tag{2}$$

which writes compactly as $\mathbf{H}\beta = \mathbf{Y}$, with

$$\mathbf{H} = \begin{pmatrix} f(\mathbf{w}_1\mathbf{x}_1 + b_1) & \cdots & f(\mathbf{w}_H\mathbf{x}_1 + b_H) \\ \vdots & \ddots & \vdots \\ f(\mathbf{w}_1\mathbf{x}_N + b_1) & \cdots & f(\mathbf{w}_H\mathbf{x}_N + b_H) \end{pmatrix}, \tag{3}$$

and $\beta = (\beta_1^T \ldots \beta_H^T)^T$ and $\mathbf{Y} = (\mathbf{y}_1^T \ldots \mathbf{y}_N^T)^T$.

Theorem 2.1 of (Huang et al., 2006) states that with randomly initialized input weights and biases for the SLFN, and under the condition that the activation function is infinitely differentiable, then the hidden layer output matrix can be determined and will provide an approximation of the target values as good as wished (non-zero training error).

The way to calculate the output weights β from the knowledge of the hidden layer output matrix $\mathbf{H}$ and target values, is proposed with the use of a Moore-Penrose generalized inverse of the matrix $\mathbf{H}$, denoted as $\mathbf{H}^\dagger$ (Rao and Mitra, 1972). Overall, the ELM algorithm is summarized as Algorithm 1.

Algorithm 1 ELM

s Given a training set $(\mathbf{x}_i, \mathbf{y}_i), \mathbf{x}_i \in \mathbb{R}^{d_1}, \mathbf{y}_i \in \mathbb{R}^{d_2}$, an activation function $f : \mathbb{R} \mapsto \mathbb{R}$ and the number of hidden nodes H.

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

Number of hidden units is an important parameter for ELM, and should be chosen with care. The selection can be done for example by cross-validation, information criteria or starting with a large number and pruning the network (Miche et al., 2010).

2.2 Neural Network Kernel

Neural network kernel is derived in (Williams, 1998) by letting the number of hidden units go to infinity. A Gaussian prior is set to hidden layer weights, which are then integrated out. The only parameters remaining after the integration are variances for weights. This leads to an analytical expression for expected covariance between two feature space vectors,

$$\mathrm{k_{NN}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{2}{\pi} \sin^{-1} \frac{2\tilde{\mathbf{x}}_i^T \Sigma \tilde{\mathbf{x}}_j}{\sqrt{(1 + 2\tilde{\mathbf{x}}_i^T \Sigma \tilde{\mathbf{x}}_i)(1 + 2\tilde{\mathbf{x}}_j^T \Sigma \tilde{\mathbf{x}}_j)}}. \tag{4}$$

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

NNK also arises as a special case of a more general arc-cosine kernel (Cho and Saul, 2009). NNK is not to be confused with tanh-kernel $\tanh((x_i \cdot x_j) + b)$, which is often also called MLP kernel (from Multi-Layer Perceptron).

2.3 Data Sets and Experiments

We perform two experiments. In "GP experiment" ELM kernel is used in a Gaussian Process classifier (Rasmussen and Williams, 2006). Results as function of H are compared with results of neural network kernel. In "ELM experiment" ELM classification accuracy (as function of number of hidden units) is compared to accuracy got by replacing 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 GP experiment is implemented by modifying `gpstuff` toolbox[1]. Expectation propagation (Minka, 2001) is used for Gaussian process inference. The ELM experiment uses the authors' matlab implementation.

We use six data sets from UCI machine learning repository (Asuncion and Newman, 2007). They are described in Table 1. For representative results, the data sets were chosen to have different sample sizes and different dimensionalities.

Table 1: Data sets used in the experiments.

name	# samples	# dims	data types
Arcene[†]	200	10000	cont.
US votes	435	16	bin.
WDBC	569	30	cont.
Pima	768	8	cont.
Tic Tac Toe	958	27	categ.
Internet ads	2359	1558	cont., bin.

[†] (Guyon et al., 2004)

For ELM experiment 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 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. Maximum number of hidden units for ELM experiment is 250, to make sure to cover the sensible operating range of ELM (up to N hidden units) for all data sets.

In the GP experiment, we are more interested in ELM behavior as the function of hidden units than the prediction accuracy. We therefore only consider variation from random weights, drawing 30 repetitions of random weights. The data is split into train and test sets (50 % / 50 %). Zero-mean unit-variance normalization is used for the data.

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

ELM places few restrictions to the activation of hidden units. In this work we use the error function $\mathrm{Erf}(z) = 2/\sqrt{\pi}\int_0^z \exp -t^2$, since it is the sigmoid used in the derivation of NNK. For the same reason we use Gaussian distribution for weights, instead of the more usual uniform. ELM only requires the distribution to be continuous.

3 ANALYSIS OF ELM

Essential property of a fully trained neural network is its ability to learn features on data. Features extracted by the network 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 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 another kernel method only similarities between feature space vectors are used.

3.1 ELM Kernel

Authors of (Frénay and Verleysen, 2010) propose using ELM hidden layer to form a kernel to be used in SVM classification. They define ELM kernel function as

$$\mathrm{k}_{\mathrm{ELM}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{H} f(\mathbf{x}_i) \cdot f(\mathbf{x}_j), \qquad (5)$$

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.

When the number of hidden units grows, this kernel matrix approaches that given by NNK. Figure 1 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 clearly converges towards NNK.

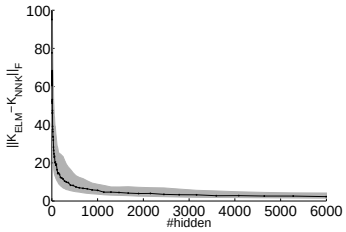

Figure 1: ELM kernel (K_{ELM}) approaches neural network kernel (K_{NN}) in Frobenius norm when number of hidden unit grows. Mean by black dots, 95% interval by shading. Variation is caused by randomness in weights. WDBC data set.

Results of GP experiment are shown in Figure 2. ELM kernel behavior in GP seems qualitatively similar to that observed in (Frénay and Verleysen, 2010) 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.

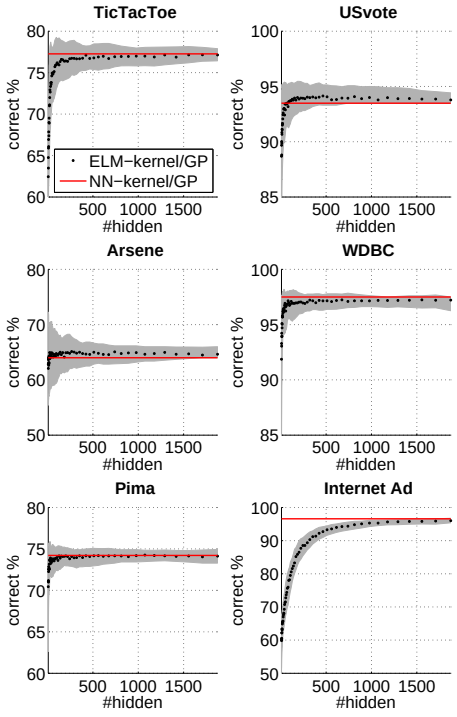

Figure 2: GP classification accuracy when using ELM kernel (black dots and shading) versus NKK accuracy (horizontal line).

3.2 NNK Replacing ELM Hidden Layer

3.2.1 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. Kernel matrix specifies the pairwise similarities. In order to use pairwise information from the NNK instead of ELM hidden layer, we must find a vectorial representation for the data.

The idea of recovering points given their mutual relationships is old (Young and Householder, 1938), and is the basis of multidimensional scaling (Torgerson, 1952). Multidimensional scaling is used in psychometry for handling results of pairwise comparisons, and more generally as a dimension reduction method. When the data arises from real pairwise comparisons, like in psychometry, there is no guarantee of structure of the similarity matrix. In our case, on the contrary, the structure is known: NKK is derived as covariance, and is therefore positive semidefinite (PSD).

Any PSD matrix can be decomposed into a matrix and its Hermitian conjugate

$$\mathbf{C} = \mathbf{L}\mathbf{L}^H. \tag{6}$$

There are different methods for finding the factors (Golub and Van Loan, 1996). Matlab `cholcov` implements a method based on eigendecomposition. If we take $\mathbf{C}$ in (6) to be output of the NNK function (4), 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 $\mathbf{H}$ in ELM,

$$\beta = \mathbf{L}^\dagger \mathbf{Y}. \tag{7}$$

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 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 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. As positive definiteness cannot be guaranteed, we use the more general decomposition for PSD matrices in all cases.

The one remaining problem is the mapping of 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 fom test points to training point, as given by NNK (4). This means that NNK output for test data $\mathbf{C}_*$ is covariance matrix of the form

$$\mathbf{C}_* = \mathbf{L}\mathbf{L}_*^H. \tag{8}$$

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

$$\mathbf{L}_* = (\mathbf{L}^\dagger \mathbf{C}_*)^H = (\mathbf{L}^\dagger \mathbf{L}\mathbf{L}_*^H)^H, \tag{9}$$

and the predictions for test targets are computed as

$$\mathbf{Y}_* = \mathbf{L}_* \beta. \tag{10}$$

3.2.2 ELM Experiment

NNK-ELM results are shown in 3, as function of σ. Results for two data sets are clearly affected by the variance parameter, others are less sensitive.

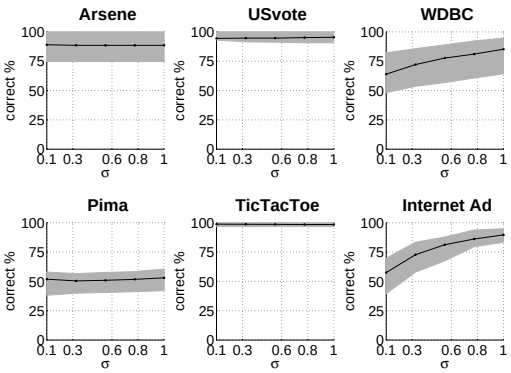

Figure 3: NNK-ELM results, mean and 95 % interval due to data variation.

Predictions given by ordinary ELM are shown in in Figure 4, and mean values for NNK-ELM predictions 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 level of guessing.

We also 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. In the following section we will look at variance effects in in more detail and discuss the reasons for importance of variance.

3.3 Variance of Weights

Importance of variance, or more often the range used for uniform distribution, is regarded important in ELM works (e.g. (Miche et al., 2010)). However, it is not seen as a model parameter, but simply a constant which must be suitably fixed to guarantee that the sigmoid operation neither remains linear nor too strongly saturates to ± 1.

Variance effects from Figure 4 are summarized in Figures 5 and 6.

Figure 4 shows the mean predictions of ELM 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.

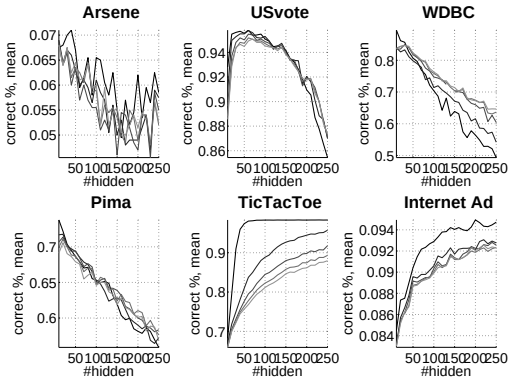

Figure 5: Effect of variance on mean of ELM predictions. Darker shade indicates smaller variance.

Variance also affects the uncertainty of ELM predictions, as witnessed by Figure 6, where the width of smoothed 95 % intervals of predictions is shown. WDBC data set exhibit the strongest effect, followed by Pima data. All data sets show some effect of variance. Number of hidden units interacts with effects of σ, but no general pattern appears. Also direction of the effect remains unspecified; for four data sets smaller σ gives larger uncertainty, but the opposite is true for the rest.

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. The operating point of the sigmoidal activation determines the flexibility of the model. When weights are small, the sigmoid produces a nearly linear mapping. Large weights result in a highly non-linear mapping. This is illustrated in

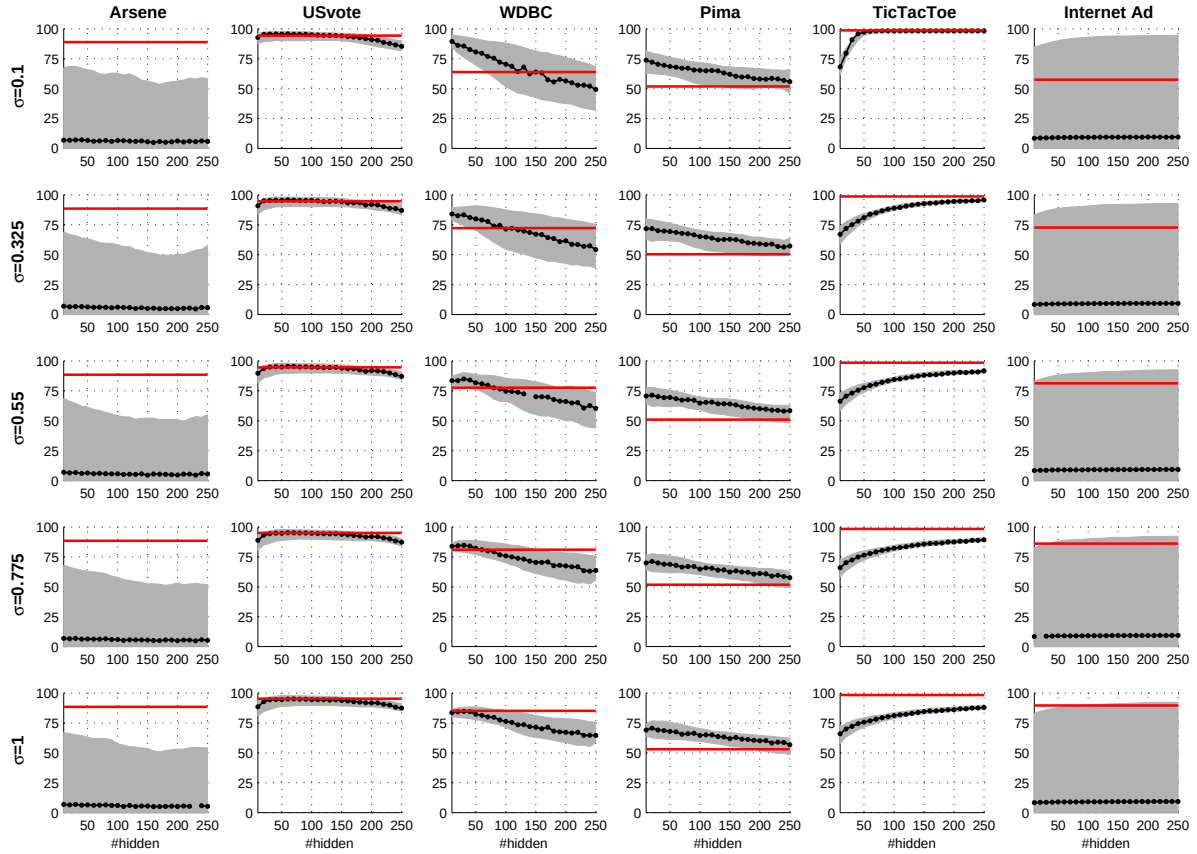

Figure 4: ELM results (mean as black dots, 95 % interval as shading) for different values of σ. Mean of NNK results (horizontal (red) line) are shown for comparison.

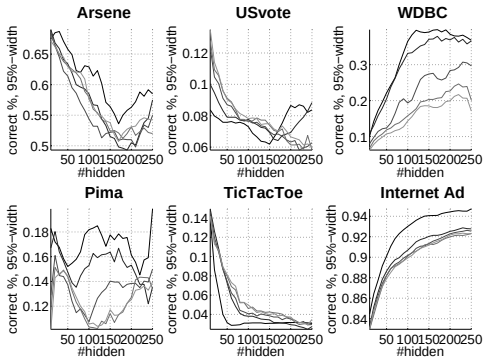

Figure 6: Effect of variance on width of the 95 % interval of ELM predictions. Darker shade indicates smaller variance.

Figure 7. One-dimensional data points, spread over range [-1,1] (the x-axis), are given random weights drawn from zero-mean Gaussian distribution and then fed through an error function sigmoid, 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.

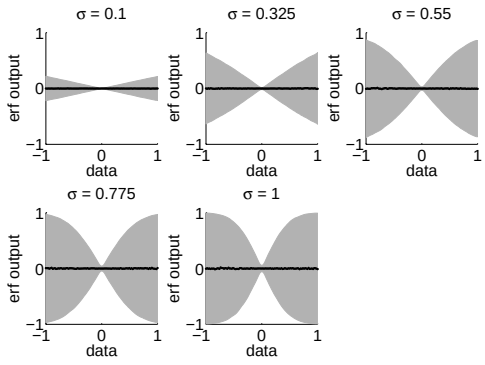

Figure 7: Distributions of predictions of an error function sigmoid for different σ.

Considering the effect of variance on complexity of the model, we think that variance should undergo model selection just as the number of hidden units does.

4 DISCUSSION

4.1 On Properties of ELM

Authors of (Huang et al., 2006) promote ELM by speed, dependence on a single tuning 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.

First factor to consider is the inherent randomness of ELM results. If average performance of ELM is to be assessed, any runs must be repeated several times, adding to the computational burden.

Complexity of model selection is determined by the *number of tuning parameters*, since all sensible combinations should be considered.

First parameter 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.

Traditionally, somewhat arbitrary fixed values have been used for weight variance, and model selection has only considered the number of hidden units. Our results show that also the weight variance has a noticeable effect on results, and should thus be considered a tuning parameter.

Generally, *small training error* and good generalization may be contradictory goals. ELM is proved (Huang et al., 2006) to be able to reproduce the training data exactly if the number of hidden units equals or exceeds the number of data points. This behavior, though important in proving 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 ELM network from overfitting to the training data, a factor usually not discussed in ELM literature.

Generalization ability of ELM is attributed to the fact that computing output layer weights by pseudoinverse achieves a minimum norm solution. Generalization ability of a neural network is in (Bartlett, 1998) shown 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, 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 hidden unit activations, discussed in Section 3.3. 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.

4.2 ELM as a Kernel

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 an explicit nonlinear feature space mapping.

4.3 NNK as ELM Hidden Layer

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 $O(N^3)$. ELM runs much faster than that, 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, on the other hand, is much faster with ELM, if $H \ll N$.

Furthermore, NNK-ELM has only one tuning parameter, the variance of hidden layer weights. Opposing the popular view, we argue that also ELM model selection should consider weight variance as parameter. If the number of data points is reasonably small, NNK-ELM can thus result in considerable time savings when doing model selection. A factor adding to this is that, unlike ELM, NNK-ELM gives deterministic results, and only requires repetitions if variability due to training data is considered.

NNK-ELM can also naturally deal with non-standard data. NNK corresponds to an infinite network with error function sigmoids in hidden units. If a Gaussian kernel was used instead, the computation would imitate an infinite radial basis function network. Dropping the neural network interpretation, any positive semidefinite matrix can be used. This leaves us with the idea of using a kernel for nonlinear mapping, then returning to a vectorial representation of points, and applying a classical algorithm (as opposed to an inner-product formulation of algorithms needed for kernel methods). In the case of NNK-ELM, the algorithm is a simple linear regression, but the same idea could be used with arbitrary algorithms. This can serve as a way of applying classical, difficult-to-kernelize algorithms to non-standard data (like graphs or strings), for which kernels are defined.

4.4 Future Directions

When ELM is considered as an approximation of an infinite network, it becomes obvious that the variance of hidden layer weights is more important than the weights themselves. It should undergo rigorous model selection, as any other parameter. Also lessons already learned from other neural network architectures, like the effect of weight variance on the operating point of sigmoids, should be kept in mind when determining future directions for ELM development.

Questions about correct number and behavior of hidden units in ELM remain open. If the hidden units do not learn anything, what is their meaning in the network? Do they have a role besides increasing the variance of output?

If we fix the data x and draw weights w_i (including the bias) randomly and independently, then also the hidden layer outputs $a_i = f(w_i x)$ are independent random variables. They are combined into model output as $b = \sum_{i=1}^{H} \beta_i a_i$.

Variance of b is related to number and variance of a_i. This is seen by remembering that, for independent random variables F and G, $\mathrm{Var}[\mathrm{F}+\mathrm{G}] = \mathrm{Var}[\mathrm{F}] + \mathrm{Var}[\mathrm{G}]$. The more hidden units we use, the larger the variance of the model output.

Training of the output layer has opposite effect on variance. Output weights are not random, so variance of the model output b is related to that of a_i by rule $\mathrm{Var}[\mathrm{cF}] = c^2\mathrm{Var}[\mathrm{F}]$ (where c is a constant). That is, variance of b is formed as a weighted sum of variances of a_i. The weights β_i's are chosen to have minimal norm. Although minimizing the norm does not guarantee minimal variance, minimum norm estimators partially minimize the variance as well (Rao, 1972). Therefore, choice of output weights tends to cancel the variance-increasing effect of hidden units.

We have recognized the importance of variance, yet the roles and interactions of weight variance, number of hidden units (increases variance) and determination of output weights (decreases variance) are not clear, at least to the authors. If we are to understand how and why ELM works, the role of variance needs further study.

REFERENCES

Asuncion, A. and Newman, D. (2007). UCI machine learning repository.

Bartlett, P. L. (1998). 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(2):525–536.

Cho, Y. and Saul, L. K. (2009). Kernel methods for deep learning. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, *Proc. of NIPS*, volume 22, pages 342–350.

Frénay, B. and Verleysen, M. (2010). Using SVMs with randomised feature spaces: an extreme learning approach. In *Proc. of ESANN*, pages 315–320.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix computations*. The Johns Hopkins University Press.

Guyon, I., Gunn, S. R., Ben-Hur, A., and Dror, G. (2004). Result analysis of the nips 2003 feature selection challenge. In *Proc. of NIPS*.

Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501.

Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., and Lendasse, A. (2010). OP-ELM: Optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162.

Minka, T. (2001). Expectation propagation for approximate bayesian inference. In *Proc. of UAI*.

Rao, C. R. (1972). Estimation of variance and covariance components in linear model. *Journal of the American Statistical Association*, 67(337):112–115.

Rao, C. R. and Mitra, S. K. (1972). *Generalized Inverse of Matrices and Its Applications*. Wiley.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.

Torgerson, W. S. (1952). Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419.

Williams, C. K. I. (1998). Computation with infinite neural networks. *Neural Computation*, 10:1203–1216.

Young, G. and Householder, A. S. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3(1):19–22.