Adaptive Regularization in Neural Network Modeling

Jan Larsen[1], Claus Svarer[2], Lars Nonboe Andersen[1] and Lars Kai Hansen[1]

[1] Department of Mathematical Modelling, Building 321
Technical University of Denmark
DK-2800 Lyngby, Denmark
emails: jl@imm.dtu.dk, lna@imm.dtu.dk, lkhansen@imm.dtu.dk
www: http://eivind.imm.dtu.dk
[2] Neurobiology Research Unit
Department of Neurology, Building 9201
Copenhagen University Hospital
Blegdamsvej 9
DK-2100 Copenhagen Ø, Denmark
email: csvarer@pet.rh.dk
www: http://neuro.pet.rh.dk

Abstract. In this paper we address the important problem of optimizing regularization parameters in neural network modeling. The suggested optimization scheme is an extended version of the recently presented algorithm [24]. The idea is to minimize an empirical estimate – like the cross-validation estimate – of the generalization error with respect to regularization parameters. This is done by employing a simple iterative gradient descent scheme using virtually no additional programming overhead compared to standard training. Experiments with feed-forward neural network models for time series prediction and classification tasks showed the viability and robustness of the algorithm. Moreover, we provided some simple theoretical examples in order to illustrate the potential and limitations of the proposed regularization framework.

1 Introduction

Neural networks are flexible tools for time series processing and pattern recognition. By increasing the number of hidden neurons in a 2-layer architecture any relevant target function can be approximated arbitrarily close [18]. The associated risk of overfitting on noisy data is of major concern in neural network design, which find expression in the ubiquitous bias-variance dilemma, see e.g., [9].

The need for regularization is two-fold: First, it remedies numerical problems in the training process by smoothing the cost function and by introducing curvature in low (possibly zero) curvature regions of cost function. Secondly, regularization is a tool for reducing variance by introducing extra bias. The overall objective of architecture optimization is to minimize the generalization error. The architecture can be optimized *directly* by stepwise selection procedures (including pruning techniques) or *indirectly* using regularization. In general, one would prefer a hybrid scheme; however, a very flexible regularization may substitute the need for selection procedures. The numerical experiments we consider mainly hybrid pruning/adaptive regularization schemes.

The trick presented in this communication addresses the problem of adapting regularization parameters.

> *The trick consists in formulating a simple iterative gradient descent scheme for adapting the regularization parameters aiming at minimizing the generalization error.*

We suggest to use an empirical estimate[3] of the generalization error, viz. the K-fold cross-validation [8], [38]. In [24] and [3] the proposed scheme was studied using the hold-out validation estimator.

In addition to empirical estimators for the generalization error a number of *algebraic* estimators like FPE [1], FPER [22], GEN [20], GPE [30] and NIC [32] have been developed in recent years. These estimates, however, depend on a number of statistical assumptions which can be quite hard to justify. In particular, they are $o(1/N_t)$ estimators where N_t is the number of training examples. However, for many practical modeling set-ups it is hard to meet the large training set assumption.

In [13] properties of adaptive regularization is studied in the simple case of estimating the mean of a random variable using an algebraic estimate of the average[4] generalization error and [14] proposed an adaptive regularization scheme for neural networks based on an algebraic estimate. However, experiments indicate that this scheme has a drawback regarding robustness. In addition, the requirement of a large training set may not be met.

The Bayesian approach to adapt regularization parameters is to minimize the so-called evidence [5, Ch. 10], [29]. The evidence, however, does not in a simple way relate to the generalization error which is our primary object of interest.

Furthermore [2] and [37] consider the use of a validation set to tune the amount of regularization, in particular when using the early-stop technique.

Section 2 considers training and empirical generalization assessment. In Section 3 the framework for optimization of regularization parameters is presented. The experimental section 4 deals with examples of feed-forward neural networks models for classification and time series prediction. Further, in order to study the theoretical potential/limitations of the proposed framework, we include some simulations on a simple toy problem.

2 Training and Generalization

Suppose that the neural network is described by the vector function $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})$ where $\boldsymbol{x}$ is the input vector and $\boldsymbol{w}$ is the vector of network weights and thresholds with dimensionality m. The objective is to use the network model for approximating the true conditional input-output distribution $p(\boldsymbol{y}|\boldsymbol{x})$, or some moments

[3] For further discussion on empirical generalization assessment, see e.g., [23].

[4] Average w.r.t. to different training sets.

hereof. For regression and signal processing problems we normally model the conditional expectation $E\{\boldsymbol{y}|\boldsymbol{x}\}$.

Assume that we have available a data set $\mathcal{D} = \{\boldsymbol{x}(k); \boldsymbol{y}(k)\}_{k=1}^{N}$ of N input-output examples. In order to both train and empirically estimate the generalization performance we follow the idea of K-fold cross-validation [8], [38] and split the data set into K randomly chosen disjoint sets of approximately equal size, i.e., $\mathcal{D} = \cup_{j=1}^{K} \mathcal{V}_j$ and $\forall\, i \neq j : \mathcal{V}_i \cap \mathcal{V}_j = \emptyset$. Training and validation is replicated K times, and in the j'th run training is done on the set $\mathcal{T}_j = \mathcal{D} \setminus \mathcal{V}_j$ and validation is performed on $\mathcal{V}_j$.

The cost function, $C_{\mathcal{T}_j}$, for network training on $\mathcal{T}_j$, is supposed to be the sum of a loss function (or training error), $S_{\mathcal{T}_j}(\boldsymbol{w})$, and a regularization term $R(\boldsymbol{w}, \boldsymbol{\kappa})$ parameterized by a set of regularization parameters $\boldsymbol{\kappa}$, i.e.,

$$C_{\mathcal{T}_j}(\boldsymbol{w}) = S_{\mathcal{T}_j}(\boldsymbol{w}) + R(\boldsymbol{w}, \boldsymbol{\kappa}) = \frac{1}{N_{tj}} \sum_{k=1}^{N_{tj}} \ell\left(\boldsymbol{y}(k), \widehat{\boldsymbol{y}}(k); \boldsymbol{w}\right) + R(\boldsymbol{w}, \boldsymbol{\kappa}) \qquad (1)$$

where $\ell(\cdot)$ measures the distance between the output $\boldsymbol{y}(k)$ and the network prediction $\widehat{\boldsymbol{y}}(k) = \boldsymbol{f}(\boldsymbol{x}(k); \boldsymbol{w})$. In section 4 we will consider log-likelihood and the square error loss function $\ell = |\boldsymbol{y} - \widehat{\boldsymbol{y}}|^2$. $N_{tj} \equiv |\mathcal{T}_j|$ defines the number of training examples in $\mathcal{T}_j$ and k is used to index the k'th example $[\boldsymbol{x}(k), \boldsymbol{y}(k)]$. Training provides the estimated weight vector $\widehat{\boldsymbol{w}}_j = \arg\min_{\boldsymbol{w}} C_{\mathcal{T}_j}(\boldsymbol{w})$.

The j'th validation set $\mathcal{V}_j$ consist of $N_{vj} = N - N_{tj}$ examples and the validation error[5] of the trained network reads

$$S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j) = \frac{1}{N_{vj}} \sum_{k=1}^{N_{vj}} \ell\left(\boldsymbol{y}(k), \widehat{\boldsymbol{y}}(k); \widehat{\boldsymbol{w}}_j\right) \qquad (2)$$

where the sum runs over the N_{vj} validation examples. $S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j)$ is thus an estimate of the generalization error, defined as the expected loss,

$$G(\widehat{\boldsymbol{w}}_j) = E_{\boldsymbol{x},\boldsymbol{y}}\{\ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}; \widehat{\boldsymbol{w}}_j)\} = \int \ell(\boldsymbol{y}, \widehat{\boldsymbol{y}}; \widehat{\boldsymbol{w}}_j) \cdot p(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x} d\boldsymbol{y} \qquad (3)$$

where $p(\boldsymbol{x}, \boldsymbol{y})$ is the unknown joint input-output probability density. Generally, $S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j) = G(\widehat{\boldsymbol{w}}_j) + O(1/\sqrt{N_{vj}})$ where $O(\cdot)$ is the Landau order function[6]. Thus we need large N_{vj} to achieve an accurate estimate of the generalization error. On the other hand, this leaves only few data for training thus the true generalization $G(\widehat{\boldsymbol{w}}_j)$ increases. Consequently there exist a trade-off among the two conflicting aims which calls for finding an optimal split ratio. The optimal split ratio[7] is an interesting open and difficult problem since it depends on the total algorithm in which the validation error enters. Moreover, it depends on the learning curve[8] [16].

---

[5] That is, the loss function on the validation set.

[6] If $h(x) = O(g(x))$ then $|h(x)|/|g(x)| < \infty$ for $x \to 0$.

[7] For more elaborations on the split of data, see e.g., [2], [19], [23] and [25].

[8] Defined as the average generalization error as a function of the number of training examples.

The final K-fold cross-validation estimate is given by the average validation error estimates,

$$\widehat{\Gamma} = \frac{1}{K} \sum_{j=1}^{K} S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j). \tag{4}$$

$\widehat{\Gamma}$ is an estimate of the *average* generalization error over all possible training sets of size N_{tj},

$$\Gamma = E_{\mathcal{T}}\{G(\widehat{\boldsymbol{w}}_j)\}. \tag{5}$$

$\widehat{\Gamma}$ is an unbiased estimate of Γ if the data of $\mathcal{D}$ are independently distributed[9], see e.g., [15].

The idea is now to optimize the amount of regularization by minimizing $\widehat{\Gamma}$ w.r.t. the regularization parameters $\boldsymbol{\kappa}$. An algorithm for this purpose is described in Section 3. Furthermore, we might consider optimizing regularization using the hold-out validation estimate corresponding to $K = 1$. In this case one have to choose a split ratio. Without further ado, we will recommend a 50/50 splitting.

Suppose that we found the optimal $\boldsymbol{\kappa}$ using the cross-validation estimate. Replications of training result in K different weight estimates $\widehat{\boldsymbol{w}}_j$ which might be viewed as an ensemble of networks. In [15] we showed under certain mild conditions that when considering a $o(1/N)$ approximation, the average generalization error of the ensemble network $f_{\mathrm{ens}}(\boldsymbol{x}) = \sum_{j=1}^{K} \beta_j \cdot f(\boldsymbol{x}, \widehat{\boldsymbol{w}}_j)$ equals that of the network trained on all examples in $\mathcal{D}$ where β_j weights the contribution from the j'th network and $\sum_j \beta_j = 1$. If K is a divisor in N then $\forall j,\ \beta_j = 1/K$, otherwise $\beta_j = (N - N_{vj})/N(K-1)$. Consequently, one might use the ensemble network to compensate for the increase in generalization error due to only training on $N_{tj} = N - N_{vj}$ data. Alternatively, one might retrain on the full data set $\mathcal{D}$ using the optimal $\boldsymbol{\kappa}$. We use the latter approach in the experimental section.

A minimal necessary requirement for a procedure which estimates the network parameters on the training set and optimizes the amount of regularization from a cross-validation set is: the generalization error of the regularized network should be smaller than that of the unregularized network trained on the full data set $\mathcal{D}$. However, this is not always the case, and is the quintessence of various "no free lunch" theorems [11], [43], [45]:

- If the regularizer is parameterized using many parameters, $\boldsymbol{\kappa}$, there is a potential risk of over-fitting on the cross-validation data. A natural way to avoid this situation is to limit the number of regularization parameters. Another recipe is to impose constraints on $\boldsymbol{\kappa}$ (hyper regularization).
- The specific choice of the regularizers functional form impose prior constraints on the functions to be implemented by the network[10]. If the prior information is mismatched to the actual problem it might be better not to use regularization.

[9] That is, $[\boldsymbol{x}(k_1), \boldsymbol{y}(k_1)]$ is independent of $[\boldsymbol{x}(k_2), \boldsymbol{y}(k_2)]$ for all $k_1 \neq k_2$.

[10] The functional constraints are through the penalty imposed on the weights.

– The de-biasing procedure described above which compensate for training only on $N_{tj} < N$ examples might fail to yield better performance since the weights now are optimized using all data, including those which where left out exclusively for optimizing regularization parameters.
– The split among training/validation data, and consequently the number of folds, K, may not be chosen appropriately.

These problems are further addressed in Section 4.1.

3 Adapting Regularization Parameters

The choice of regularizer may be motivated by

– the fact that the minimization of the cost function is normally an ill- posed task. Regularization smoothens the cost function and thereby facilitates the training. The weight decay regularizer[11], originally suggested by Hinton in the neural networks literature, is a simple way to accomplish this task, see e.g., [34].
– a priori knowledge of the weights, e.g., in terms of a prior distribution (when using a Bayesian approach). In this case the regularization term normally plays the role of a log-prior distribution. Weight decay regularization may be viewed as a Gaussian prior, see e.g., [5]. Other types of priors, e.g., the Laplacian [12], [42] and soft weight sharing [33] has been considered. Moreover, priors have been developed for the purpose of restricting the number of weights (pruning), e.g., the so-called weight elimination [41].
– a desired characteristics of the functional mapping performed by the network. Typically, a smooth mapping is preferred. Regularizers which penalizes curvature of the mapping has been suggested in [4], [7], [31], [44].

In the experimental section we consider weight decay regularization and some generalizations hereof. Without further ado, weight decay regularization has proven to be useful in many neural network applications.

The standard approach for estimation of regularization parameters is more and less systematic search and evaluation of the cross-validation error. However, this is not viable for multiple regularization parameters. On the other hand, as will be demonstrated, it is possible to derive an optimization algorithm based on gradient descent.

Consider a regularization term $R(\boldsymbol{w}, \boldsymbol{\kappa})$ which depends on q regularization parameters contained in the vector $\boldsymbol{\kappa}$. Since the estimated weights $\widehat{\boldsymbol{w}}_j = \arg\min_{\boldsymbol{w}} C_{\mathcal{T}_j}(\boldsymbol{w})$ are controlled by the regularization term, we may in fact consider the cross-validation error Eq. (4) as an *implicit function* of the regularization parameters, i.e.,

$$\widehat{\Gamma}(\boldsymbol{\kappa}) = \frac{1}{K} \sum_{j=1}^{K} S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})) \qquad (6)$$

[11] Also known as ridge regression.

where $\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})$ is the $\boldsymbol{\kappa}$-dependent vector of weights estimated from training set $\mathcal{T}_j$. The optimal regularization can be found by using gradient descent[12],

$$\boldsymbol{\kappa}_{(n+1)} = \boldsymbol{\kappa}_{(n)} - \eta \frac{\partial \widehat{\Gamma}}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}(\boldsymbol{\kappa}_{(n)})) \tag{7}$$

where $\eta > 0$ is a step-size (learning rate) and $\boldsymbol{\kappa}_{(n)}$ is the estimate of the regularization parameters in iteration n.

Suppose the regularization term is linear in the regularization parameters,

$$R(\boldsymbol{w}, \boldsymbol{\kappa}) = \boldsymbol{\kappa}^\top \boldsymbol{r}(\boldsymbol{w}) = \sum_{i=1}^{q} \kappa_i r_i(\boldsymbol{w}) \tag{8}$$

where κ_i are the regularization parameters and $r_i(\boldsymbol{w})$ the associated regularization functions. Many suggested regularizers are linear in the regularization parameters, this includes the popular weight decay regularization as well as regularizers imposing smooth functions such as the Tikhonov regularizer [4], [5] and the smoothing regularizer for neural networks [31], [44]. However, there exist exceptions such as weight-elimination [41] and soft weight sharing [33]. In this case the presented method needs few modifications.

Using the results of the Appendix, the gradient of the cross-validation error equals

$$\frac{\partial \widehat{\Gamma}}{\partial \boldsymbol{\kappa}}(\boldsymbol{\kappa}) = \frac{1}{K} \sum_{j=1}^{K} \frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j), \tag{9}$$

$$\frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j) = -\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{w}^\top}(\widehat{\boldsymbol{w}}_j) \cdot \boldsymbol{J}_j^{-1}(\widehat{\boldsymbol{w}}_j) \cdot \frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{w}}(\widehat{\boldsymbol{w}}_j). \tag{10}$$

where $\boldsymbol{J}_j = \partial^2 C_{\mathcal{T}_j} / \partial \boldsymbol{w} \partial \boldsymbol{w}^\top$ is the Hessian of the cost function. As an example, consider the case of weight decay regularization with separate weight decays for two group of weights, e.g., the input-to-hidden and hidden-to output weights, i.e.,

$$R(\boldsymbol{w}, \boldsymbol{\kappa}) = \kappa^I \cdot |\boldsymbol{w}^I|^2 + \kappa^H \cdot |\boldsymbol{w}^H|^2 \tag{11}$$

where $\boldsymbol{\kappa} = [\kappa^I, \kappa^H]$, $\boldsymbol{w} = [\boldsymbol{w}^I, \boldsymbol{w}^H]$ with $\boldsymbol{w}^I$, $\boldsymbol{w}^H$ denoting the input-to-hidden and hidden-to output weights, respectively. The gradient of the validation error then yields,

$$\frac{\partial S_{\mathcal{V}_j}}{\partial \kappa^I}(\widehat{\boldsymbol{w}}_j) = -2(\widehat{\boldsymbol{w}}_j^I)^\top \cdot \boldsymbol{g}_j^I, \quad \frac{\partial S_{\mathcal{V}_j}}{\partial \kappa^H}(\widehat{\boldsymbol{w}}_j) = -2(\widehat{\boldsymbol{w}}_j^H)^\top \cdot \boldsymbol{g}_j^H \tag{12}$$

where $\boldsymbol{g}_j$ is the vector

$$\boldsymbol{g}_j = [\boldsymbol{g}_j^I, \boldsymbol{g}_j^H] = \boldsymbol{J}_j^{-1}(\widehat{\boldsymbol{w}}_j) \cdot \frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{w}}(\widehat{\boldsymbol{w}}_j). \tag{13}$$

In summary, the algorithm for adapting regularization parameters consists of the following 8 steps:

[12] We have recently extended this algorithm incorporating second order information via the Conjugate Gradient technique [10].

1. Choose the split ratio; hence, the number of folds, K.
2. Initialize $\boldsymbol{\kappa}$ and the weights of the network[13].
3. Train the K networks with fixed $\boldsymbol{\kappa}$ on $\mathcal{T}_j$ to achieve $\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})$, $j = 1, 2, \cdots, K$. Calculate the validation errors $S_{\mathcal{V}_j}$ and the cross-validation estimate $\widehat{\Gamma}$.
4. Calculate the gradients $\partial S_{\mathcal{V}_j}/\partial \boldsymbol{\kappa}$ and $\partial \widehat{\Gamma}/\partial \boldsymbol{\kappa}$ cf. Eq. (9) and (10). Initialize the step-size η.
5. Update $\boldsymbol{\kappa}$ using Eq. (7).
6. Retrain the K networks from the previous weight estimates and recalculate the cross-validation error $\widehat{\Gamma}$.
7. If no decrease in cross-validation error then perform a bisection of η and go to step 5; otherwise, continue.
8. Repeat steps 4–7 until the relative change in cross-validation error is below a small percentage or, e.g., the 2-norm of the gradient $\partial \widehat{\Gamma}/\partial \boldsymbol{\kappa}$ is below a small number.

Compared to standard neural network training the above algorithm does generally not lead to severe computational overhead. First of all, the standard approach of tuning regularization parameters by, more or less systematic search, requires a lot of training sessions. The additional terms to be computed in the adaptive algorithm are: 1) the derivative of the regularization functions w.r.t. the weights, $\partial \boldsymbol{r}/\partial \boldsymbol{w}$, 2) the gradient of the validation errors, $\partial S_{\mathcal{V}_j}/\partial \boldsymbol{w}$, and 3) the inverse Hessians, $\boldsymbol{J}_j^{-1}$. The first term is often a simple function of the weights[14] and computationally inexpensive. In the case of feed-forward neural networks, the second term is computed by one pass of the validation examples through a standard back-propagation algorithm. The third term is computationally more expensive. However, if the network is trained using a second order scheme, which requires computation of the inverse Hessian[15], there is no computational overhead.

The adaptive algorithm requires of the order of $K \cdot itr_{\kappa} \cdot itr_{\eta}$ weight retrainings. Here itr_{κ} is the number of iterations in the gradient descent scheme for $\boldsymbol{\kappa}$ and itr_{η} is the average number of bisections of η in step 7 of the algorithm. In the experiments carried out the number of retrainings is approx. 100–300 times K. Recall, since we keep on retraining from the current weight estimate, the number of training epochs is generally small.

The number of weight retrainings is somewhat higher than that involved when optimizing the network by using a pruning technique like validation set based Optimal Brain Damage (vOBD) [24], [26]. vOBD based on K-fold cross-validation requires of the order of $K \cdot m$ retrainings, where $m = \dim(\boldsymbol{w})$. The adaptive regularization algorithm is easily integrated with the pruning algorithm as demonstrated in the experimental section.

[13] In Sec. 4.1 a practical initialization procedure for $\boldsymbol{\kappa}$ is described.

[14] For weight decay, it is $2\boldsymbol{w}$.

[15] Often the computations are reduced by using a Hessians approximation, e.g., the Gauss-Newton approximation. Many studies have reported significant training speed-up by using second order methods, see e.g., [21], [34].

4 Numerical Experiments

4.1 Potentials and Limitations in the Approach

The purpose of the section is to demonstrate the potential and limitations of the suggested adaptive regularization framework. We consider the simple linear data generating *system*, viz. estimating the mean of a Gaussian variable,

$$y(k) = w^\circ + \varepsilon(k) \tag{14}$$

where w° is the true mean and the noise $\varepsilon(k) \sim \mathcal{N}(0, \sigma_\varepsilon^2)$.

We employ 2-fold cross-validation, i.e., $\mathcal{D} = \mathcal{T}_1 \cup \mathcal{T}_2$, where $\mathcal{T}_j$, $j = 1, 2$ denote the two training sets in the validation procedure containing approximately half the examples[16]. The linear *model* $y(k) = w + e(k)$ is trained using the mean square cost function augmented by simple weight decay, as shown by

$$C_{\mathcal{T}_j}(w) = \frac{1}{N_{tj}} \sum_{k=1}^{N_{tj}} (y(k) - w)^2 + \kappa \cdot w^2 \tag{15}$$

where k runs over examples of the data set in question. The estimated weights are $\widehat{w}_j = \bar{y}_j/(1+\kappa)$ where $\bar{y}_j = N_{tj}^{-1} \sum_{k=1}^{N_{tj}} y(k)$ are the estimated mean. For this simple case, the minimization of the cross-validation error given by,

$$\widehat{\Gamma}(\kappa) = \frac{1}{2} \sum_{j=1}^{2} S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j(\kappa)), \qquad S_{\mathcal{V}_j}(\widehat{\boldsymbol{w}}_j(\kappa)) = \frac{1}{N_{vj}} \sum_{k=1}^{N_{vj}} (y(k) - \widehat{w}_j)^2, \tag{16}$$

can be done exactly. The optimal κ is given by

$$\kappa_{\mathrm{opt}} = \frac{\bar{y}_1^2 + \bar{y}_2^2}{2\bar{y}_1\bar{y}_2} - 1. \tag{17}$$

Assuming N to be even, the ensemble average of the estimated weights[17], $\widehat{w}_j(\kappa_{\mathrm{opt}})$, leads to the final estimate

$$\widehat{w}_{\mathrm{reg}} = \frac{1}{2}\left(\widehat{w}_1(\kappa_{\mathrm{opt}}) + \widehat{w}_2(\kappa_{\mathrm{opt}})\right) = \frac{\bar{y}_1\bar{y}_2(\bar{y}_1 + \bar{y}_2)}{\bar{y}_1^2 + \bar{y}_2^2}. \tag{18}$$

Notice two properties: First, the estimate is self-consistent as $\lim_{N\to\infty} \widehat{w}_{\mathrm{reg}} = \lim_{N\to\infty} \widehat{w}_{\mathcal{D}} = w^\circ$ where $\widehat{w}_{\mathcal{D}} = N^{-1} \sum_{k=1}^{N} y(k) = (\bar{y}_1 + \bar{y}_2)/2$ is the unregularized estimate trained on all data. Secondly, it is easy to verify that $\bar{y}_j \sim \mathcal{N}(w^\circ, 2\sigma_\varepsilon^2/N)$. That is, if the *normalized true weight* $\theta \equiv w^\circ/A$ where $A = \sqrt{2/N} \cdot \sigma_\varepsilon$ is large then $\bar{y}_j \approx w^\circ$ which means, $\widehat{w}_{\mathrm{reg}} \approx \widehat{w}_{\mathcal{D}}$.

16 That is, $N_{t1} = \lfloor N/2 \rfloor$ and $N_{t2} = N - N_{t1}$. Note that these training sets are also the two validation sets, $\mathcal{V}_1 = \mathcal{T}_2$, and vice versa.

17 The ensemble average corresponds to retraining on all data using κ_{opt}. The weighting of the two estimates is only valid for N even (see Sec. 2 for the general case).

The objective is now to test whether using $\widehat{w}_{\text{reg}}$ results in lower generalization error than employing the unregularized estimate $\widehat{w}_{\mathcal{D}}$. The generalization error associated with using the weight w is given by

$$G(w) = \sigma_\varepsilon^2 + (w - w^\circ)^2. \tag{19}$$

Further define the generalization error improvement,

$$Z = G(\widehat{w}_{\mathcal{D}}) - G(\widehat{w}_{\text{reg}}) = (\widehat{w}_{\mathcal{D}} - w^\circ)^2 - (\widehat{w}_{\text{reg}} - w^\circ)^2. \tag{20}$$

Note that Z merely is a function of the random variables $\bar{y}_1$, $\bar{y}_2$ and the true weight w°, i.e., it suffices to get samples of $\bar{y}_1$, $\bar{y}_2$ when evaluating properties of Z. Define the normalized variables

$$\widetilde{y}_j = \frac{\bar{y}_j}{A} \sim \mathcal{N}\left(\frac{w^\circ}{\sigma_\varepsilon} \cdot \sqrt{\frac{N}{2}}, 1\right) = \mathcal{N}(\theta, 1). \tag{21}$$

It is easily shown that the normalized generalization error improvement Z/A^2 is a function of $\widetilde{y}_1$, $\widetilde{y}_2$ and θ; hence, the distribution of Z/A^2 is parameterized solely by θ.

As a quality measure we consider the *probability of improvement* in generalization error given by $\text{Prob}\{Z > 0\}$. Note that $\text{Prob}\{Z > 0\} = 1/2$ corresponds to equal preference of the two estimates. The probability of improvement depends only on the normalized weight θ since $\text{Prob}\{Z > 0\} = \text{Prob}\{Z/A^2 > 0\}$.

Moreover, we consider the *relative generalization error improvement*, defined as

$$\text{RGI} = 100\% \cdot \frac{Z}{G(\widehat{w}_{\mathcal{D}})}. \tag{22}$$

In particular, we focus on the probability that the relative improvement in generalization is bigger than[18] x, i.e., $\text{Prob}(\text{RGI} > x)$. Optimally $\text{Prob}(\text{RGI} > x)$ should be close to 1 for $x \le 0\%$ and slowly decaying towards zero for $0\% < x \le 100\%$. Using the notation $\widetilde{w}_{\text{reg}} = \widehat{w}_{\text{reg}}/A$, $\widetilde{w}_{\mathcal{D}} = \widehat{w}_{\mathcal{D}}/A$, RGI can be written as

$$\text{RGI} = 100\% \cdot \frac{(\widetilde{w}_{\mathcal{D}} - \theta)^2 - (\widetilde{w}_{\text{reg}} - \theta)^2}{N/2 + (\widetilde{w}_{\mathcal{D}} - \theta)^2}. \tag{23}$$

Thus, the distribution of RGI is parameterized by θ and N.

The quality measures are computed by generating Q independent realizations of $\widetilde{y}_1$, $\widetilde{y}_2$, i.e., $\{\widetilde{y}_1^{(i)}, \widetilde{y}_2^{(i)}\}_{i=1}^{Q}$. E.g., the probability of improvement is estimated by $P_{\text{imp}} = Q^{-1} \sum_{i=1}^{Q} \mu(Z^{(i)})$ where $\mu(x) = 1$ for $x > 0$, and zero otherwise.

The numerical results of comparing $\widehat{w}_{\text{reg}}$ to the unregularized estimate $\widehat{w}_{\mathcal{D}}$ is summarized in Fig. 1.

[18] Note that, $\text{Prob}(\text{RGI} > 0) = \text{Prob}(Z > 0)$.

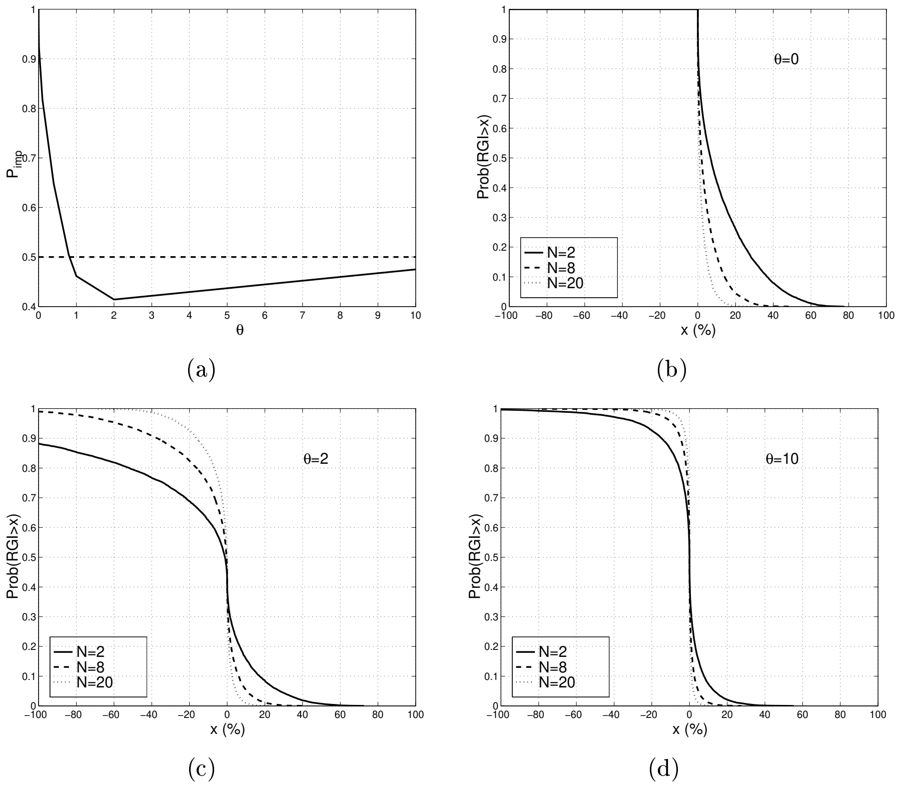

Fig. 1. Result of comparing the optimally regularized estimate $\widehat{w}_{\text{reg}}$ of the mean of a Gaussian variable to the unregularized estimate $\widehat{w}_{\mathcal{D}}$. The results are based on $Q = 10^5$ independent realizations. The probability of improvement P_{imp}, shown in panel (a), is one for when the normalized true weight $\theta = \sqrt{N/2} \cdot w^\circ / \sigma_\varepsilon = 0$, and above 0.5 for $\theta \lesssim 0.8$. That is, when the prior information of the weight decay regularizer is correct (true weight close to zero), when N is small or when σ_ε is large. As θ becomes large P_{imp} tends to 0.5 due to the fact that $\widetilde{w} \approx \widehat{w}_{\mathcal{D}}$. Panel (b)–(d) display Prob(RGI $> x$) for $\theta \in \{0, 2, 10\}$. The ideal probability curve is 1 for $x < 0$ and a slow decay towards zero for $x > 0$. The largest improvement is attained for small θ and small N. Panel (c) and (d) indicate that small N gives the largest probability for $x > 0$; however, also the smallest probability for negative x. That is, a higher chance of getting a good improvement also increases the change of deterioration. Notice, even though $P_{\text{imp}} < 0.5$ for $\theta = 2, 10$ there is still a reasonable probability of getting a significant improvement.

4.2 Classification

We test the performance of the adaptive regularization algorithm on a vowel classification problem. The data are based on the Peterson and Barney database [35]. The classes are vowel sounds characterized by the first four formant frequencies. 76 persons (33 male, 28 female and 15 children) have pronounced $c = 10$ different vowels (IY IH EH AE AH AA AO UH UW ER) two times. This results in a data base of totally 1520 examples. The database is the verified database described in [40] where all data[19] are used, including examples where utterance failed of unanimous identification in the listening test (26 listeners). All examples were included to make the task more difficult.

The regularization was adapted using a hold-out validation error estimator, thus the examples were split into a data set, $\mathcal{D}$, consisting of $N = 760$ examples (16 male, 14 female and 8 children) and an independent test set of the remaining 760 examples. The regularization was adapted by splitting the data set $\mathcal{D}$ equally into a validation set of $N_v = 380$ examples and a training set of $N_t = 380$ examples (8 male, 7 female and 4 children in each set).

We used a feed-forward 2-layer neural network with hyperbolic tangent neurons in the hidden layer and modified SoftMax normalized outputs, $\widehat{y}_i$, see e.g., [5], [17], [3]. Thus, the outputs estimates the posterior class probabilities $p(\mathcal{C}_i|\boldsymbol{x})$, where $\mathcal{C}_i$ denotes the i'th class, $i = 1, 2, \cdots, c$. Bayes rule (see e.g., [5]) is used to assign $\mathcal{C}_i$ to input $\boldsymbol{x}$ if $i = \arg\max_j p(\mathcal{C}_j|\boldsymbol{x})$. Suppose that the network weights are given by $\boldsymbol{w} = [\boldsymbol{w}^I, \boldsymbol{w}^I_{\text{bias}}, \boldsymbol{w}^H, \boldsymbol{w}^H_{\text{bias}}]$ where $\boldsymbol{w}^I$, $\boldsymbol{w}^H$ are input-to-hidden and hidden-to-output weights, respectively, and the bias weights are assembled in $\boldsymbol{w}^I_{\text{bias}}$ and $\boldsymbol{w}^H_{\text{bias}}$. Suppose that the targets $y_i(k) = 1$ if $\boldsymbol{x}(k) \in \mathcal{C}_i$, and zero otherwise. The network is optimized using a log-likelihood loss function augmented by a weight decay regularizer using 4 regularization parameters,

$$\begin{aligned} C(\boldsymbol{w}) = & \frac{1}{N_t} \sum_{k=1}^{N_t} \sum_{i=1}^{c} y_i(k) \log(\widehat{y}_i(k, \boldsymbol{w})) \\ & + \kappa^I \cdot |\boldsymbol{w}^I|^2 + \kappa^I_{\text{bias}} \cdot |\boldsymbol{w}^I_{\text{bias}}|^2 + \kappa^H \cdot |\boldsymbol{w}^H|^2 + \kappa^H_{\text{bias}} \cdot |\boldsymbol{w}^H_{\text{bias}}|^2. \end{aligned} \quad (24)$$

We further define unnormalized weight decays as $\alpha \equiv \kappa \cdot N_t$. This regularizer is motivated by the fact that the bias, input and hidden layer weights play a different role, e.g., the input, hidden and bias signals normally have different scale (see also [5, Ch. 9.2]).

The simulation set-up is:

- Network: 4 inputs, 5 hidden neurons, 9 outputs[20].
- Weights are initialized uniformly over $[-0.5, 0.5]$, regularization parameters are initialized at zero. One step in a gradient descent training algorithm (see e.g., [28]) is performed and the weight decays are re-initialized at $\lambda_{\max}/10^2$, where $\lambda_{\max}$ is the max. eigenvalue of the Hessian matrix of the cost function. This initialization scheme is motivated by the following observations:
  - Weight decays should be so small that they do not reduce the approximation capabilities of the network significantly.

[19] The database can be retrieved from `ftp://eivind.imm.dtu.dk/dist/data/vowel/PetersonBarney.tar.Z`

[20] We only need 9 outputs since the posterior class probability of the 10th class is given by $1 - \sum_{j=1}^{9} p(\mathcal{C}_j|\boldsymbol{x})$.

  - They should be so large that the algorithm is prevented from being trapped in a local optimum and numerical instabilities are eliminated.
- Training is now done using a Gauss-Newton algorithm (see e.g., [28]). The Hessian is inverted using the Moore-Penrose pseudo inverse ensuring that the eigenvalue spread[21] is less than 10^8.
- The regularization step-size η is initialized at 1.
- When the adaptive regularization scheme has terminated 3% of the weights are pruned using a validation set based version of the Optimal Brain Damage (vOBD) recipe [24], [26].
- Alternation between pruning and adaptive regularization continues until the validation error has reached a minimum.
- Finally, remaining weights are retrained on all data using the optimized weight decay parameters.

Table 1. Probability of misclassification (pmc) and log-likelihood cost function (without reg. term, see Eq. (24)) for the classification example. The neural network averages and standard deviations are computed from 10 runs. In the case of small fixed regularization, weight decays were set at initial values equal to $\lambda_{\max}/10^6$ where $\lambda_{\max}$ is the largest eigenvalue of the Hessian matrix of the cost function. Optimal regularization refers to the case of optimizing 4 weight decay parameters. Pruning refers to validation set based OBD. KNN refers to k-nearest-neighbor classification.

Probability of Misclassification (pmc)

	NN small fixed reg.	**NN** opt. reg.+prun.	**KNN** ($k = 9$)
Training Set	0.075 ± 0.026	0.107 ± 0.008	0.150
Validation Set	0.143 ± 0.014	0.115 ± 0.004	0.158
Test Set	0.146 ± 0.010	0.124 ± 0.006	0.199
Test Set (train. on all data)	0.126 ± 0.010	0.119 ± 0.004	0.153

Log-likelihood Cost Function

	NN small fixed reg.	**NN** opt. reg.+prun.
Training Set	0.2002 ± 0.0600	0.2881 ± 0.0134
Validation Set	0.7016 ± 0.2330	0.3810 ± 0.0131
Test Set	0.6687 ± 0.2030	0.3773 ± 0.0143
Test Set (train. on all data)	0.4426 ± 0.0328	0.3518 ± 0.0096

Table 1 reports the average and standard deviations of the probability of

[21] Eigenvalue spread should not be larger than the square root of the machine precision [6].

misclassification (*pmc*) and log-likelihood cost function over 10 runs for pruned networks using the optimal regularization parameters. Note that retraining on the full data set decreases the test *pmc* slightly on the average. In fact, improvement was noticed in 9 out of 10 runs. The table further shows the gain of the combined adaptive regularization/pruning algorithm relative to using a small fixed weight decay. However, recall, cf. Sec. 4.1, that the actual gain is *very* dependent on the noise level, data set size, etc. The objective is not to demonstrate high gain for a specific problem, rather to demonstrate that algorithm runs fairly robust in a classification set-up. For comparison we used a k-nearest-neighbor (KNN) classification (see e.g., [5]) and found that $k = 9$ neighbors was optimal by minimizing *pmc* on the validation set. The neural network performed significantly better. Contrasting the obtained results to other work is difficult. In [36] results on the Peterson-Barney vowel problem are reported, but their data are not exactly the same; only the first 2 formant frequencies were used. Furthermore, different test sets have been used for the different methods presented. The best result reported [27] is obtained by using KNN and reach $pmc = 0.186$ which is significantly higher than our results.

Fig. 2 shows the evolution of the adaptive regularization as well as the pruning algorithm.

4.3 Time Series Prediction

We tested the performance of the adaptive regularization schemes on the Mackey-Glass chaotic time series prediction problem, see e.g., [21], [39]. The goal is to predict the series 100 steps ahead based on previous observations. The feed-forward net configuration is an input lag-space $\boldsymbol{x}(k) = [x(k), x(k-6), x(k-12), x(k-18)]$ of 4 inputs, 25 hidden hyperbolic tangent neurons, and a single linear output unit $\widehat{y}(k)$ which predicts $y(k) = x(k+100)$. The cost function is the squared error, $N_t^{-1} \sum_{k=1}^{N_t} (y(k) - \widehat{y}(k, \boldsymbol{w}))^2$, augmented by a weight decay regularizer using 4 different weight decays as described in Section 4.2.

The simulation set-up is:

- The data set, $\mathcal{D}$, has $N = 500$ examples and an independent test has 8500 examples.
- The regularization parameters are optimized using a hold-out validation error with an even split[22] of the data set into training and validation sets each having 250 examples.
- Weight decays are initialized at zero and one Gauss-Newton iteration is performed, then weight decays were re-initialized at $\lambda_{\max}/10^6$, where $\lambda_{\max}$ is the max. eigenvalue of the Hessian matrix of the cost function.
- The network is trained using a Gauss-Newton training scheme. The Hessian is inverted using the Moore-Penrose pseudo inverse ensuring that the eigenvalue spread is less than 10^8.
- The regularization step-size η is initialized at 10^{-2}.

[22] The sensitivity to different splits are considered in [24].

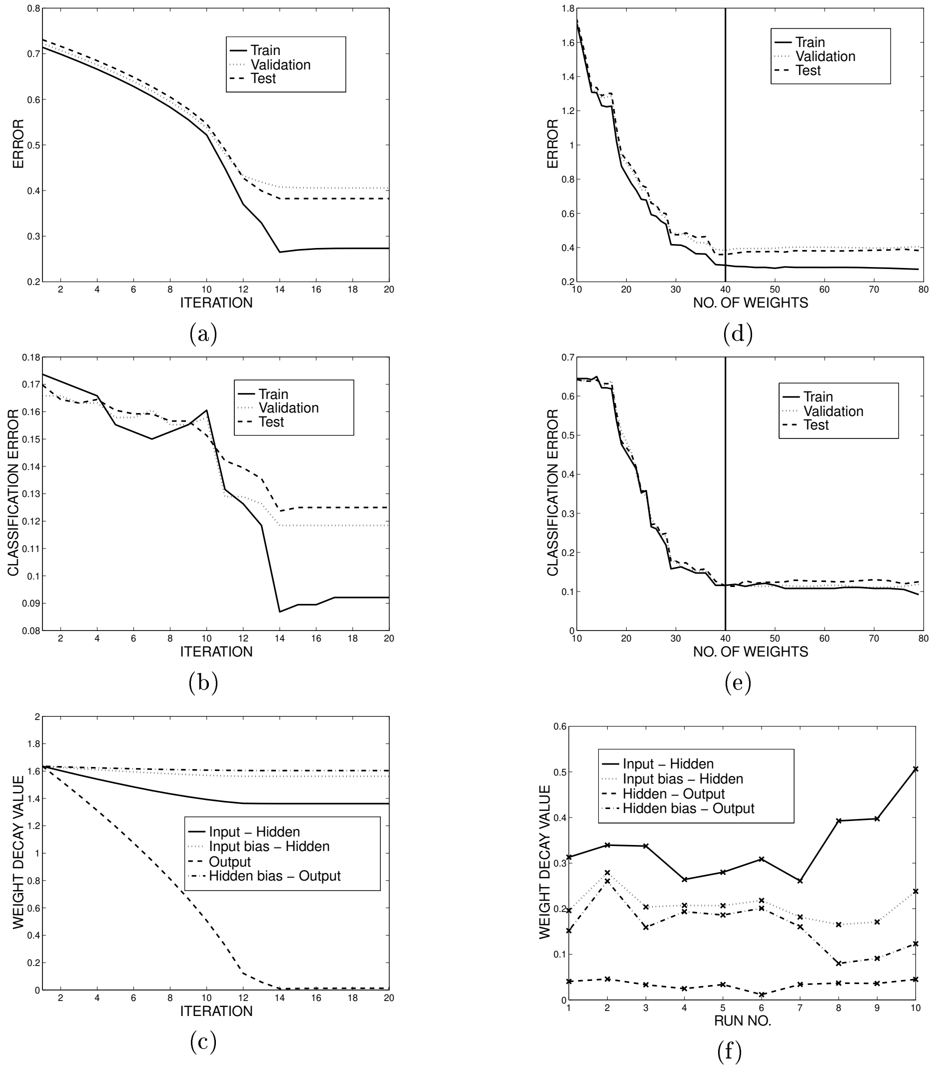

Fig. 2. Classification example. Panels (a), (b) and (c) show the evolution of the adaptive regularization algorithm in a typical run (fully connected network). The weight decays are optimized aiming at minimizing the validation error in panel (a). Note that also the test error decreases. This tendency is also evident in panel (b) displaying *pmc* even though a small increase noticed. In panel (c) the convergence unnormalized weight decays, $\alpha = \kappa \cdot N_t$, are depicted. Panels (d) and (e) show the evolution of errors and *pmc* during the pruning session. The optimal network is chosen as the one with minimal validation error, as indicated by the vertical line. There is only a marginal effect of pruning in this run. Finally, in panel (f), the variation of the optimal (end of pruning) α's in different runs is demonstrated. A clear similarity over runs is noticed.

– Alternation between adapting the 4 weight decays and validation set based pruning [24].
– The pruned network is retrained on all data using the optimized weight decay parameters.

Table 2 reports the average and standard deviations of the normalized squared error (i.e., the squared error normalized with the estimated variance of $x(k)$, denoted $\widehat{\sigma}_x^2$) over 10 runs for optimal regularization parameters. Retraining on the full data set decreases the test error somewhat on the average. Improvement was noticed in 10 out of 10 runs. We tested 3 different cases: small fixed regularization, small fixed regularization assisted by pruning and combined adaptive regularization/pruning. It turns that pruning alone does not improve performance; however, supplementing by adaptive regularization gives a test error reduction.

We furthermore tried a flexible regularization scheme, viz. individual weight decay where $R(\boldsymbol{w}, \boldsymbol{\kappa}) = \sum_{i=1}^{m} \kappa_i w_i^2$ and $\kappa_i \geq 0$ are imposed. In the present case it turned out that the flexible regularizer was not able to outperform the joint adaptive regularization/pruning scheme; possibly due to training and validation set sizes.

Table 2. Normalized squared error performance for the time series prediction examples. All figures are in units of $10^{-3}\widehat{\sigma}_x^2$ and averages and standard deviations are computed from 10 runs. In the case of small fixed regularization, weight decays were set at initial values equal to $\lambda_{\max}/10^6$ where $\lambda_{\max}$ is the largest eigenvalue of the Hessian matrix of the cost function. Optimal regularization refers to the case of optimizing 4 weight decay parameters. Pruning refers to validation set based OBD.

	NN small fixed reg.	**NN** small fixed reg.+prun.	**NN** opt. reg.+prun.
Training Set	0.17 ± 0.07	0.12 ± 0.04	0.10 ± 0.07
Validation Set	0.53 ± 0.26	0.36 ± 0.07	0.28 ± 0.14
Test Set	1.91 ± 0.68	1.58 ± 0.21	1.29 ± 0.46
Test Set (train. on all data)	1.33 ± 0.43	1.34 ± 0.26	1.17 ± 0.48

Fig. 3 demonstrates adaptive regularization and pruning in a typical case using 4 weight decays.

5 Conclusions

In this paper it was suggested to adapt regularization parameters by minimizing the cross-validation error or a simple hold-out validation error. We derived a simple gradient descent scheme for optimizing regularization parameters which has a small programming overhead and an acceptable computational overhead compared to standard training. Numerical examples with a toy linear model showed

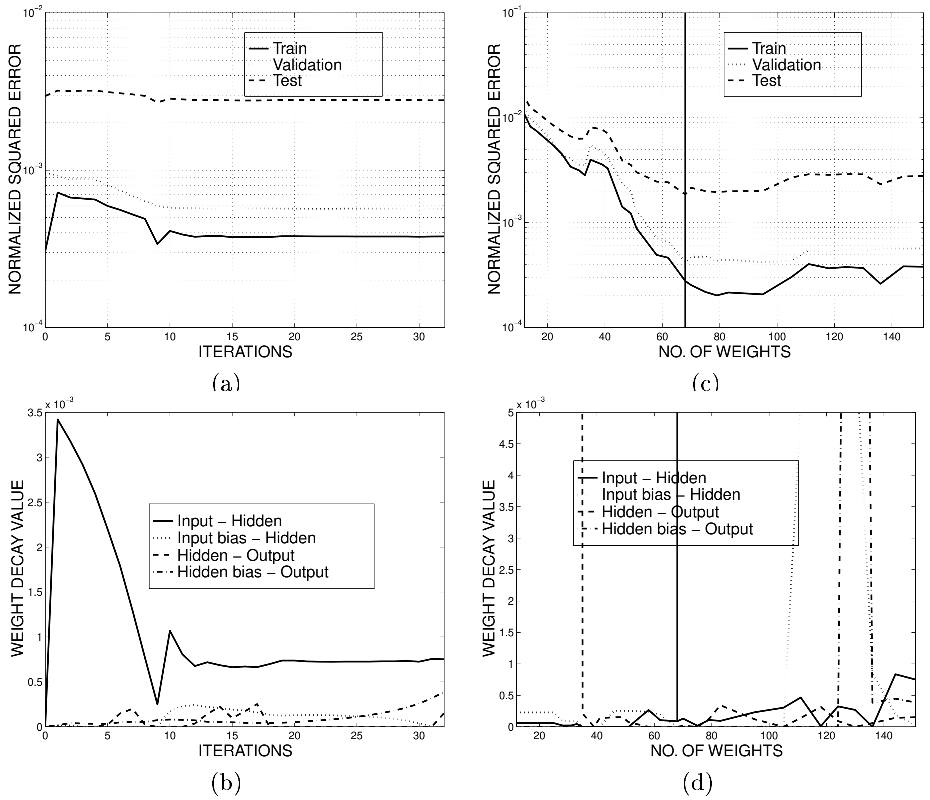

Fig. 3. Time series prediction example. Panels (a) and (b) show a typical evolution of errors and unnormalized weight decay values α when running the adaptive regularization algorithm using 4 weight decays. The normalized validation error drops approx. a factor of 2 when adapting weight decays. It turns out that some regularization of the input-to-hidden and output bias weights are needed whereas the other weights essentially requires no regularization[23]. In panel (c) and (d) it is demonstrated that pruning reduces the test error slightly. The optimal network is chosen as the one with minimal validation error, as indicated by the vertical line.

limitations and advantages of the adaptive regularization approach. Moreover, numerical experiments on classification and time series prediction problems successfully demonstrated the functionality of the algorithm. Adaptation of regularization parameters resulted in lower generalization error; however, it should be emphasized that the actual yield is very dependent on the problem and the choice of the regularizers functional form.

[23] Recall that if a weight decay κ is below $\lambda_{\max}/10^8$ it does not influence the Moore-Penrose pseudo inversion of the Hessian.

Acknowledgments

This research was supported by the Danish Natural Science and Technical Research Councils through the Computational Neural Network Center. JL furthermore acknowledge the Radio Parts Foundation for financial support. Mads Hintz-Madsen and Morten With Pedersen are acknowledged for stimulating discussions.

Appendix

Assume that the regularization term is linear in the regularization parameters, i.e.,

$$R(\boldsymbol{w}, \boldsymbol{\kappa}) = \boldsymbol{\kappa}^\top \boldsymbol{r}(\boldsymbol{w}) = \sum_{i=1}^{q} \kappa_i r_i(\boldsymbol{w}) \tag{25}$$

The gradient of the cross-validation error Eq. (4) is

$$\frac{\partial \widehat{\Gamma}}{\partial \boldsymbol{\kappa}}(\boldsymbol{\kappa}) = \frac{1}{K} \sum_{j=1}^{K} \frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})) \tag{26}$$

Using the chain rule the gradient vector of the validation error, $S_{\mathcal{V}_j}$, can be written as

$$\frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})) = \frac{\partial \boldsymbol{w}^\top}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})) \cdot \frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{w}}(\widehat{\boldsymbol{w}}_j(\boldsymbol{\kappa})) \tag{27}$$

where $\partial \boldsymbol{w}^\top / \partial \boldsymbol{\kappa}$ is the $q \times m$ derivative matrix of the estimated weights w.r.t. the regularization parameters and $m = \dim(\boldsymbol{w})$. In order to find this derivative matrix, consider the gradient of the cost function w.r.t. to the weights as a function of $\boldsymbol{\kappa}$ and use the following expansion around the current estimate $\boldsymbol{\kappa}_{(n)}$,

$$\frac{\partial C_{\mathcal{T}_j}}{\partial \boldsymbol{w}}(\boldsymbol{\kappa}) = \frac{\partial C_{\mathcal{T}_j}}{\partial \boldsymbol{w}}(\boldsymbol{\kappa}_{(n)}) + \frac{\partial^2 C_{\mathcal{T}_j}}{\partial \boldsymbol{w} \partial \boldsymbol{\kappa}^\top}(\boldsymbol{\kappa}_{(n)}) \cdot (\boldsymbol{\kappa} - \boldsymbol{\kappa}_{(n)}) + o(|\boldsymbol{\kappa} - \boldsymbol{\kappa}_{(n)}|). \tag{28}$$

Requiring $\widehat{\boldsymbol{w}}(\boldsymbol{\kappa}_{(n+1)})$ in the next iteration to be an optimal weight vector, i.e., $\partial C_{\mathcal{T}_j} / \partial \boldsymbol{w}(\boldsymbol{\kappa}_{(n+1)}) = \boldsymbol{0}$ implies

$$\frac{\partial^2 C_{\mathcal{T}_j}}{\partial \boldsymbol{w} \partial \boldsymbol{\kappa}^\top}(\widehat{\boldsymbol{w}}(\boldsymbol{\kappa}_{(n)})) = \boldsymbol{0}. \tag{29}$$

Recall that $\partial C_{\mathcal{T}_j} / \partial \boldsymbol{w}(\boldsymbol{\kappa}_{(n)}) = \boldsymbol{0}$ by assumption. Eq. (29) can be used for determining $\partial \boldsymbol{w}^\top / \partial \boldsymbol{\kappa}$. Recognizing that the cost function $C_{\mathcal{T}_j}(\widehat{\boldsymbol{w}}(\boldsymbol{\kappa})) = S_{\mathcal{T}_j}(\widehat{\boldsymbol{w}}(\boldsymbol{\kappa})) + R(\widehat{\boldsymbol{w}}(\boldsymbol{\kappa}), \boldsymbol{\kappa})$ depends *implicitly* (thorough $\widehat{\boldsymbol{w}}(\boldsymbol{\kappa})$) and *explicitly* on $\boldsymbol{\kappa}$ it is possible, by using Eq. (25), to derive the following relation[24]:

$$\frac{\partial \boldsymbol{w}^\top}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j) = -\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{w}^\top}(\widehat{\boldsymbol{w}}_j) \cdot \boldsymbol{J}_j^{-1}(\widehat{\boldsymbol{w}}_j) \tag{30}$$

[24] For convenience, here $\widehat{\boldsymbol{w}}$'s explicit $\boldsymbol{\kappa}$-dependence is omitted.

where $\boldsymbol{J}_j = \partial^2 C_{\mathcal{T}_j} / \partial \boldsymbol{w} \partial \boldsymbol{w}^\top$ is the Hessian of the cost function which e.g., might be evaluated using the Gauss-Newton approximation [28]. Finally, substituting Eq. (30) into (27) gives

$$\frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{\kappa}}(\widehat{\boldsymbol{w}}_j) = -\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{w}^\top}(\widehat{\boldsymbol{w}}_j) \cdot \boldsymbol{J}_j^{-1}(\widehat{\boldsymbol{w}}_j) \cdot \frac{\partial S_{\mathcal{V}_j}}{\partial \boldsymbol{w}}(\widehat{\boldsymbol{w}}_j) \tag{31}$$

$\partial S_{\mathcal{V}_j} / \partial \boldsymbol{w}$ is found by ordinary back- propagation on the validation set while $\partial \boldsymbol{r} / \partial \boldsymbol{w}^\top$ is calculated from the specific assumptions on the regularizer.

References

1. H. Akaike: Fitting Autoregressive Models for Prediction. Annals of the Institute of Statistical Mathematics **21** (1969) 243–247
2. S. Amari, N. Murata, K.R. Müller, M. Finke & H. Yang: Asymptotic Statistical Theory of Overtraining and Cross-Validation. Technical report METR 95-06 (1995). Accepted for IEEE Transactions on Neural Networks. Available via `ftp://archive.cis.ohio-state.edu/pub/neuroprose/amari.overtraining.ps.Z`
3. L. Nonboe Andersen, J. Larsen, L.K. Hansen & M. Hintz-madsen: Adaptive Regularization of Neural Classifiers. In J. Principe *et al.* (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VII, Piscataway, New Jersey: IEEE, (1997) 24–33
4. C.M. Bishop: Curvature-Driven Smoothing: A Learning Algorithm for Feedforward Neural Networks. IEEE Transactions on Neural Networks **4**(4) (1993) 882–884
5. C.M. Bishop: Neural Networks for Pattern Recognition. Oxford, UK: Oxford University Press (1995)
6. J.E. Dennis & R.B. Schnabel: Numerical Methods for Unconstrained Optimization and Non-linear Equations. Englewood Cliffs, NJ: Prentice- Hall, (1983)
7. H. Drucker & Y. Le Cun: Improving Generalization Performance in Character Recognition. In B.H. Juang *et al.* (eds.), Neural Networks for Signal Processing: Proceedings of the 1991 IEEE-SP Workshop, Piscataway, New Jersey: IEEE (1991) 198–207
8. S. Geisser: The Predictive Sample Reuse Method with Applications. Journal of the American Statistical Association **50** (1975) 320–328
9. S. Geman, E. Bienenstock & R. Doursat: Neural Networks and the Bias/Variance Dilemma. Neural Computation **4** (1992) 1–58
10. C. Goutte & J. Larsen: Adaptive Regularization of Neural Networks using Conjugate Gradient, in Proceedings of ICASSP'98, Seattle USA **2** (1998) 1201-1204
11. C. Goutte: Note on Free Lunches and Cross-Validation. Neural Computation **9**(6) (1997) 1211–1215
12. C. Goutte: Regularization with a Pruning Prior. To appear in Neural Networks (1997)
13. L.K. Hansen and C.E. Rasmussen: Pruning from Adaptive Regularization. Neural Computation **6** (1994) 1223–1232
14. L.K. Hansen, C.E. Rasmussen, C. Svarer & J. Larsen: Adaptive Regularization. In J. Vlontzos, J.-N. Hwang & E. Wilson (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing IV, Piscataway, New Jersey: IEEE (1994) 78–87

15. L.K. Hansen & J. Larsen: Linear Unlearning for Cross-Validation. Advances in Computational Mathematics **5** (1996) 269–280
16. J. Hertz, A. Krogh & R.G. Palmer: Introduction to the Theory of Neural Computation. Redwood City, California: Addison-Wesley Publishing Company (1991)
17. M. Hintz-Madsen, M. With Pedersen, L.K. Hansen, & J. Larsen: Design and Evaluation of Neural Classifiers. In S. Usui, Y. Tohkura, S. Katagiri & E. Wilson (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VI, Piscataway, New Jersey: IEEE, (1996) 223–232
18. K. Hornik: Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks **4** (1991) 251–257
19. M. Kearns: A Bound on the Error of Cross Validation Using the Approximation and Estimation Rates, with Consequences for the Training-Test Split. Neural Computation **9**(5) (1997) 1143–1161
20. J. Larsen: A Generalization Error Estimate for Nonlinear Systems. In S.Y. Kung *et al.* (eds.), Neural Networks for Signal Processing 2: Proceedings of the 1992 IEEE-SP Workshop, Piscataway, New Jersey: IEEE (1992) 29–38
21. J. Larsen: Design of Neural Network Filters, Ph.D. Thesis, Electronics Institute, Technical University of Denamrk (1993). Available via `ftp://eivind.imm.dtu.dk/dist/PhD_thesis/jlarsen.thesis.ps.Z`
22. J. Larsen & L.K. Hansen: Generalization Performance of Regularized Neural Network Models. In J. Vlontzos *et al.* (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing IV, Piscataway, New Jersey: IEEE (1994) 42–51
23. J. Larsen & L.K. Hansen: Empirical Generalization Assessment of Neural Network Models. In F. Girosi *et al.* (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing V, Piscataway, New Jersey: IEEE (1995) 30–39
24. J. Larsen, L.K. Hansen, C. Svarer & M. Ohlsson: Design and Regularization of Neural Networks: The Optimal Use of a Validation Set. In S. Usui, Y. Tohkura, S. Katagiri & E. Wilson (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VI, Piscataway, New Jersey: IEEE, (1996) 62–71
25. J. Larsen *et al.* : Optimal Data Set Split Ratio for Empirical Generalization Error Estimates. In preparation.
26. Y. Le Cun, J.S. Denker & S.A. Solla: Optimal Brain Damage. In D.S. Touretzky (ed.), Advances in Neural Information Processing Systems 2, Proceedings of the 1989 Conference, San Mateo, California: Morgan Kaufmann Publishers (1990) 598–605
27. D. Lowe: Adaptive Radial Basis Function Nonlinearities and the Problem of Generalisation. Proc. IEE Conf. on Artificial Neural Networks, (1989) 171– 175
28. L. Ljung: System Identification: Theory for the User. Englewood Cliffs, New Jersey: Prentice-Hall (1987)
29. D.J.C. MacKay: A Practical Bayesian Framework for Backprop Networks. Neural Computation **4**(3) (1992) 448–472
30. J. Moody: Prediction Risk and Architecture Selection for Neural Networks. In V. Cherkassky *et al.* (eds.), From Statistics to Neural Networks: Theory and Pattern Recognition Applications, Berlin, Germany: Springer-Verlag Series F **136** (1994)
31. J. Moody, T. Rögnvaldsson: Smoothing Regularizers for Projective Basis Function Networks. In Advances in Neural Information Processing Systems 9, Proceedings of the 1996 Conference, Cambridge, Massachusetts: MIT Press (1997)
32. N. Murata, S. Yoshizawa & S. Amari: Network Information Criterion — Determining the Number of Hidden Units for an Artificial Neural Network Model. IEEE

Transactions on Neural Networks **5**(6) (1994) 865–872
33. S. Nowlan & G. Hinton: Simplifying Neural Networks by Soft Weight Sharing. Neural Computation **4**(4) (1992) 473–493
34. M. With Pedersen: Training Recurrent Networks. In Proceedings of the IEEE Workshop on Neural Networks for Signal Processing VII, Piscataway, New Jersey: IEEE, (1997)
35. G.E. Peterson & H.L. Barney: Control Methods Used in a Study of the Wowels. JASA **24** (1952) 175–184
36. R.S. Shadafan & M. Niranjan: A Dynamic Neural Network Architecture by Sequential Partitioning of the Input Space. Neural Computation **6**(6) (1994) 1202–1222
37. J. Sjöberg: Non-Linear System Identification with Neural Networks, Ph.D. Thesis no. 381, Department of Electrical Engineering, Linköping University, Sweden, (1995)
38. M. Stone: Cross-validatory Choice and Assessment of Statistical Predictors. Journal of the Royal Statistical Society B **36**(2) (1974) 111–147
39. C. Svarer, L.K. Hansen, J. Larsen & C. E. Rasmussen: Designer Networks for Time Series Processing. In C.A. Kamm *et al.* (eds.), Proceedings of the IEEE Workshop on Neural Networks for Signal Processing 3, Piscataway, New Jersey: IEEE (1993) 78–87
40. R.L. Watrous: Current Status of PetersonBarney Vowel Formant Data. JASA **89** (1991) 2459– 2460
41. A.S. Weigend, B.A. Huberman & D.E. Rumelhart: Predicting the Future: A Connectionist Approach. International Journal of Neural Systems **1**(3) (1990) 193–209
42. P.M. Williams: Bayesian Regularization and Pruning using a Laplace Prior. Neural Computation **7**(1) (1995) 117–143
43. D.H. Wolpert & W.G. Macready: The Mathematics of Search. Technical Report SFI-TR-95-02-010, Santa Fe Instute (1995)
44. L. Wu & J. Moody: A Smoothing Regularizer for Feedforward and Recurrent Neural Networks. Neural Computation **8**(3) 1996
45. H. Zhu & R. Rohwer. No Free Lunch for Cross Validation. Neural Computation **8**(7) (1996) 1421–1426

This article was processed using the LaTeX macro package with LLNCS style