Optimal Use of Regularization and Cross-validation in Neural Network Modeling

Dingding Chen, cding@okstate.edu, Oklahoma State University
Martin T. Hagan, mhagan@master.ceat.okstate.edu, Oklahoma State University

Abstract

This paper proposes a new framework for adapting regularization parameters in order to minimize validation error during the training of feedforward neural networks. A second derivative of validation error based regularization algorithm (SDVR) is derived using the Gauss-Newton approximation to the Hessian. The basic algorithm, which uses incremental updating, allows the regularization parameter α to be recalculated in each training epoch. Two variations of the algorithm, called convergent updating and conditional updating, enable α to be updated over a variable interval according to the specified control criteria. Simulations on a noise-corrupted parabolic function with two-inputs and a single output are investigated. The results demonstrate that the SDVR framework is very promising for adaptive regularization and can be cost-effectively applied to a variety of different problems.

1. Introduction

Many applications have shown that training a feedforward neural network with the regularized performance function $F = \boldsymbol{e}^T\boldsymbol{e} + \alpha\boldsymbol{w}^T\boldsymbol{w}$ can improve the generalization performance of the network if the regularization parameter α is appropriately estimated. However, how to determine the parameter α is still an open question. There are several different approaches to this problem. MacKay's Bayesian framework automatically adapts the regularization parameter to maximize the evidence of the training data [1]. The computation overhead in updating the regularization parameter can be reduced when the Gauss-Newton approximation to the Hessian matrix is employed [2].

A different approach to adaptive regularization is to minimize validation error. In this case, a validation data set, which is independent of the training data, is withheld for decision making. The motivation behind this approach is based on the assumption that the selected validation set is a good representative of new data. Therefore, the model with minimum validation error will have a better chance to generalize well on novel inputs. The simplest application of this method is to train the neural network with a number of different α values, and then choose the model having the smallest validation error.

A more attractive approach to validation-set-based regularization is to use an optimization algorithm to adapt the parameter α automatically. Consider that the validation error is a function of network weights, and the network weights are affected by the α value through the regularized performance function. Therefore, the validation error is an implicit function of α. These relations can be used to solve the optimization problem. A gradient descent scheme was proposed by Larsen et al. [3] using a single validation set, and extended to multi-fold validation sets [4]. In both approaches, an updated regularization parameter is calculated after the network has been trained to convergence with the previous regularization parameter. After each parameter update, the network is again trained to convergence. However, the use of gradient descent with convergent updating has always suffered from the problem that both the resulting model performance and the computation cost are sensitive to the initial α and the initial learning rate. An improved algorithm that applies the conjugate gradient technique was reported in [5], but it also uses convergent updating.

In the following section, we will propose a new framework for updating the regularization parameter. It uses second derivative of validation error based regularization (SDVR). We will show how the the basic SDVR algorithm and its two variations can be used to reestimate the optimum value of α before the training algorithm has converged. Although our discussion is concentrated on a single validation set with a single parameter α, the extension to multi-fold validation sets and multiple regularization parameters is straightforward. In the next section we will test this new framework with several numerical experiments and will show how to make optimal use of the SDVR framework.

2. SDVR Framework

The SDVR framework has three different implementations, which are distinguished by the way in which the parameter α is updated. The basic SDVR algorithm is derived using an *incremental updating* method, which recalculates α in each training epoch. The *convergent updating* method optimizes the training performance function with fixed α until the network converges, then it determines the next estimate. The *conditional updating* method recomputes the regularization parameter only if the validation error increases, or if the trajectory of the validation error is descending but very flat.

2.1 Incremental Updating

In the incremental updating, we assume that the network is trained with a batch training algorithm on the training data set. Its regularized performance index at iteration k is

$$F_t(\boldsymbol{w}_k) = (\boldsymbol{e}_t(\boldsymbol{w}_k))^T\boldsymbol{e}_t(\boldsymbol{w}_k) + \alpha_k\boldsymbol{w}_k{}^T\boldsymbol{w}_k, \qquad (1)$$

with $\boldsymbol{w}_k$ being an $N \times 1$ weight vector and $\boldsymbol{e}_t(\boldsymbol{w}_k)$ an $n \times 1$ error vector. For next training epoch, the weight vector $\boldsymbol{w}_{k+1}$ is computed to minimize the performance index by using the Gauss-Newton method

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \boldsymbol{H}^{-1}(\boldsymbol{w}_k)\nabla F_t(\boldsymbol{w}_k), \quad (2)$$

where $\nabla F_t(\boldsymbol{w}_k)$ is the gradient vector of $F_t(\boldsymbol{w}_k)$ with respect to $\boldsymbol{w}_k$, $\boldsymbol{H}(\boldsymbol{w}_k)$ is the Gauss-Newton approximation to the Hessian matrix,

$$\boldsymbol{H}(\boldsymbol{w}_k) = \nabla^2 F_t(\boldsymbol{w}_k) \approx 2\boldsymbol{J}_t^T(\boldsymbol{w}_k)\boldsymbol{J}_t(\boldsymbol{w}_k) + 2\boldsymbol{I}_N(\alpha_k + \mu_w), \quad (3)$$

in which $\boldsymbol{J}_t(\boldsymbol{w}_k)$ is an $n \times N$ Jacobian matrix, $\boldsymbol{I}_N$ is an $N \times N$ identity matrix, and μ_w is a tunable parameter, as in the Levenberg-Marquardt implementation [6], [7].

In the basic SDVR algorithm, the updating equation for the parameter α can also be expressed in Newton's form:

$$\alpha_{k+1} = \alpha_k - H^{-1}(\boldsymbol{w}_{k+1}(\alpha_k))\nabla F_v(\boldsymbol{w}_{k+1}(\alpha_k)). \quad (4)$$

In Eq. (4), the validation error

$$F_v(\boldsymbol{w}_{k+1}(\alpha_k)) = (\boldsymbol{e}_v(\boldsymbol{w}_{k+1}))^T \boldsymbol{e}_v(\boldsymbol{w}_{k+1}) \quad (5)$$

is a function of weight vector $\boldsymbol{w}_{k+1}$, hence it is an implicit function of α_k. $\nabla F_v(\boldsymbol{w}_{k+1}(\alpha_k))$ and $H(\boldsymbol{w}_{k+1}(\alpha_k))$ are the gradient and Hessian of the validation error with respect to the regularization parameter α_k.

In order to update the parameter α_{k+1}, we assume that the weight vector $\boldsymbol{w}_{k+1}$ is updated first, using a fixed α_k. After $\boldsymbol{w}_{k+1}$ is computed, then α_{k+1} is updated. In the following derivation, we will refer to updating $\boldsymbol{w}_{k+1}$ as the inside loop training, and will refer to calculating α_{k+1} as the outside loop updating.

2.1.1 Incremental Gradient

As we see in Eq. (4), the mathematical implementation of the SDVR algorithm requires the gradient $\nabla F_v(\boldsymbol{w}_{k+1}(\alpha_k))$ and the Hessian $H(\boldsymbol{w}_{k+1}(\alpha_k))$. Since validation error is not an explicit function of α_k, we can use chain rule:

$$\nabla F_v(\boldsymbol{w}_{k+1}(\alpha_k)) = \frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1})^T \cdot \frac{\partial}{\partial \boldsymbol{w}_{k+1}} F_v(\boldsymbol{w}_{k+1}(\alpha_k)). \quad (6)$$

The first partial derivative term on the right side of Eq. (6) can be calculated from Eq. (2). Recall that $\boldsymbol{w}_k$ was computed before α_k was updated. From this view, $\boldsymbol{w}_k$ is not a function of α_k. Therefore, differentiating Eq. (2) with respect to α_k becomes

$$\frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1}) = -\left[\frac{\partial}{\partial \alpha_k}\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\right]\nabla F_t(\boldsymbol{w}_k) - \boldsymbol{H}^{-1}(\boldsymbol{w}_k)\frac{\partial}{\partial \alpha_k}\nabla F_t(\boldsymbol{w}_k). \quad (7)$$

To find the derivative matrix $\partial \boldsymbol{H}^{-1}(\boldsymbol{w}_k)/\partial \alpha_k$, we use one of properties of the inverse matrix:

$$\boldsymbol{H}(\boldsymbol{w}_k)\boldsymbol{H}^{-1}(\boldsymbol{w}_k) = \boldsymbol{I}_N. \quad (8)$$

Since the derivative of the identity matrix $\boldsymbol{I}_N$ with respect to α_k is zero, we have

$$\left[\frac{\partial}{\partial \alpha_k}\boldsymbol{H}(\boldsymbol{w}_k)\right]\boldsymbol{H}^{-1}(\boldsymbol{w}_k) + \boldsymbol{H}(\boldsymbol{w}_k)\frac{\partial}{\partial \alpha_k}\boldsymbol{H}^{-1}(\boldsymbol{w}_k) = 0. \quad (9)$$

In Eq. (9), $\partial \boldsymbol{H}(\boldsymbol{w}_k)/\partial \alpha_k$ can be easily obtained from Eq. (3). After some algebraic manipulations, we get

$$\frac{\partial}{\partial \alpha_k}\boldsymbol{H}^{-1}(\boldsymbol{w}_k) = -2[\boldsymbol{H}^{-1}(\boldsymbol{w}_k)]^2. \quad (10)$$

Note that $\nabla F_t(\boldsymbol{w}_k)$ is a linear function of α_k,

$$\frac{\partial}{\partial \alpha_k}\nabla F_t(\boldsymbol{w}_k) = 2\boldsymbol{w}_k. \quad (11)$$

Substituting Eq. (10) and Eq. (11) into Eq. (7),

$$\frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1}) = -2\boldsymbol{H}^{-1}(\boldsymbol{w}_k)[-\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\nabla F_t(\boldsymbol{w}_k) + \boldsymbol{w}_k]. \quad (12)$$

The term in the bracket above is the new updated weight vector $\boldsymbol{w}_{k+1}$ from Eq. (2). Therefore, we can rewrite Eq. (12) as:

$$\frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1}) = -2\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}. \quad (13)$$

Now let's go back to Eq. (6),

$$\frac{\partial}{\partial \boldsymbol{w}_{k+1}}F_v(\boldsymbol{w}_{k+1}(\alpha_k)) = 2\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1}), \quad (14)$$

where $\boldsymbol{J}_v(\boldsymbol{w}_{k+1})$ is the Jacobian matrix of the validation data. The final expression of the incremental gradient now can be obtained by substituting Eq. (13) and Eq. (14) into Eq. (6):

$$\nabla F_v(\boldsymbol{w}_{k+1}(\alpha_k)) = -4[\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}]^T \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1}). \quad (15)$$

Comparing Eq. (15) with Larsen's gradient expression in [3], we can see that Larsen's expression is a special case of Eq. (15) when the weight vector converges, at which time $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k = \boldsymbol{w}$. However, our approach and Larsen's approach are based on different assumptions. In [3], the converged weight vector $\boldsymbol{w}$ is a function of fixed α_k, the derivative $\partial \boldsymbol{w}/\partial \alpha_k$ is derived using the convergent condition of the inside training loop, i.e., $\partial F_t/\partial \boldsymbol{w} = 0$ and $\partial(\partial F_t/\partial \boldsymbol{w})/\partial \alpha_k = 0$. While in our approach, the regularization parameter is updated in each training epoch. Only the weight vector $\boldsymbol{w}_{k+1}$, rather than $\boldsymbol{w}_k$, is a function of α_k. The derivative $\partial \boldsymbol{w}_{k+1}/\partial \alpha_k$ is computed from the weight updating equation directly. Treating $\boldsymbol{w}_k$ and α_k as independent variables is important in implementing the SDVR algorithm. As we will see next, this assumption also makes calculating the incremental Hessian convenient.

2.1.2 Incremental Hessian Approximation

The SDVR algorithm is characterized by the incremental Hessian $H(\boldsymbol{w}_{k+1}(\alpha_k))$, which can be computed by differentiating Eq. (15) with respect to α_k.

$$H(\boldsymbol{w}_{k+1}(\alpha_k)) = \frac{\partial}{\partial \alpha_k}\nabla F_v(\boldsymbol{w}_{k+1}(\alpha_k)) = -4\frac{\partial}{\partial \alpha_k}([\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}]^T \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})) \quad (16)$$

Since $[\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}]^T \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})$ is a scalar, it is equal to its transpose. Using the rule for differentiation of a product, we can rewrite Eq. (16) as

$$\frac{\partial}{\partial \alpha_k}\left[-4(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})\right] = -4\left[\frac{\partial}{\partial \alpha_k}[(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T]\right]\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1}) - 4\left[\frac{\partial}{\partial \alpha_k}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})]\right]\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1} \quad (17)$$

The first partial derivative of the product within the bracket on the right side of Eq. (17) can be expanded as

$$\frac{\partial}{\partial \alpha_k}[(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T] \quad . \quad (18)$$
$$= \left[\left[\frac{\partial}{\partial \alpha_k}\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\right]\boldsymbol{w}_{k+1} + \boldsymbol{H}^{-1}(\boldsymbol{w}_k)\frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1})\right]^T$$

Substituting Eq. (10) and Eq. (13) into Eq. (18), we get

$$\frac{\partial}{\partial \alpha_k}[(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T] = -4[[\boldsymbol{H}^{-1}(\boldsymbol{w}_k)]^2\boldsymbol{w}_{k+1}]^T . \quad (19)$$

Note that in Eq. (17), $\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})$ is not a explicit function of α_k. Its derivative with respect to α_k can also be calculated using the chain rule:

$$\frac{\partial}{\partial \alpha_k}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})] \quad . \quad (20)$$
$$= \frac{\partial}{\partial \alpha_k}(\boldsymbol{w}_{k+1})^T \cdot \frac{\partial}{\partial \boldsymbol{w}_{k+1}}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})]$$

In Eq. (20), only $\partial[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})]/\partial \boldsymbol{w}_{k+1}$ is unknown, which is the Hessian matrix of the validation data. As with the Hessian matrix of the training data, we use the Gauss-Newton method to obtain the approximate expression

$$\frac{\partial}{\partial \boldsymbol{w}_{k+1}}(\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})) \cong \boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1}) . \quad (21)$$

Then, Eq. (20) becomes

$$\frac{\partial}{\partial \alpha_k}[\boldsymbol{e}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})] \quad (22)$$
$$= -2[\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}]^T\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})$$

The incremental Hessian can be obtained by putting Eq. (19) and Eq. (22) into Eq. (17):

$$H(\boldsymbol{w}_{k+1}(\alpha_k)) \quad (23)$$
$$\cong 16(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})$$
$$+ 8(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})$$

Now we have completed the derivation of both the gradient and the Hessian in Eq. (4) for the incremental updating of the SDVR algorithm. It appears to involve significant computation. However, since $\boldsymbol{H}^{-1}(\boldsymbol{w}_k)$ and $\boldsymbol{w}_{k+1}$ are available from the inside loop training, and the validation gradient vector $\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})$ and Hessian approximation $\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})$ can be calculated simply by passing the validation data through the same conventional subroutine as used in computing the gradient and Hessian matrix for the training data, the additional computation overhead in the SDVR algorithm is limited.

One more comment should be made on Eq. (23). Recall that in the inside loop training, the Hessian approximation given in Eq. (3) can be made positive definite with the Levenberg-Marquardt implementation, which guarantees that the weight increment is always in a descent direction. However, in Eq. (23), the second sequential product term has a quadratic form, but the first sequential product term is non-quadratic, which can be either positive or negative. In order to keep the increment of the regularization parameter in a descent direction, we will force the incremental Hessian to be equal to the quadratic term when the value obtained from Eq. (23) is negative. In this case, we get a larger Hessian, which corresponds to a reduced learning rate and will not cause any problem during the training.

In addition, a tunable positive parameter μ_α can be added to Eq. (23) to make the incremental Hessian invertible in any case and can be used to adjust the effective learning rate. This is similar to the use of μ_w in Eq. (3) with the Levenberg-Marquardt implementation. A small μ_α corresponds to a second derivative dominated approach, while a large μ_α indicates a transition to the gradient descent method. Another usage of μ_α is to force α to be positive if the decrement of the regularization parameter suggested by Eq. (4) is too large. Adding μ_α into Eq. (23), we have

$$H(\boldsymbol{w}_{k+1}(\alpha_k)) \quad (24)$$
$$\cong 16(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{e}_v(\boldsymbol{w}_{k+1})$$
$$+ 4(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1})^T\boldsymbol{H}_v(\boldsymbol{w}_{k+1})(\boldsymbol{H}^{-1}(\boldsymbol{w}_k)\boldsymbol{w}_{k+1}) + \mu_\alpha$$

with $\boldsymbol{H}_v(\boldsymbol{w}_{k+1}) = 2\boldsymbol{J}_v^T(\boldsymbol{w}_{k+1})\boldsymbol{J}_v(\boldsymbol{w}_{k+1})$.

2.1.3 Method of Application

In this paper, we will use a two-stage training method to apply the SDVR algorithm. In the first stage, our purpose is to determine an optimal regularization parameter. In the second stage, we will use a fixed α and perform the final training on a large data set consisting of the previous training and validation data. Since α is optimal, we will limit the complexity of the neural network so that it has less risk of overfitting.

Here are the general steps required for optimization of the regularization parameter with the incremental SDVR algorithm: 1. Divide the available data set into training and validation subsets using a proper splitting ratio and initialize the network weights $\boldsymbol{w}_0$ and α_0; 2. Take one step of the optimization on the training data set by calculating $\boldsymbol{w}_{k+1}$ with Eq. (2); 3. Evaluate the generalization performance on the validation data set. If the stop criterion is not satisfied, update the regularization parameter by using Eq. (4) to get α_{k+1}; 4. Reset $\boldsymbol{w}_k \Leftarrow \boldsymbol{w}_{k+1}$ and $\alpha_k \Leftarrow \alpha_{k+1}$ when the stop criterion is not met, then go back to step 2. Otherwise, terminate the first stage training; 5. Put the training data set and the validation data set together for the final training, using the latest updated regularization parameter.

For a demonstration of how the SDVR algorithm improves neural network model generalization, consider the parabolic function defined by the following equation:

$$Z = 2.5 + 5.0X(2 - X)Y(2 - Y)$$

where the input variable X and Y both range from 0 to 2. The true function surface is displayed in Figure 1. We use 211 data points for training and 130 data points for validation. The function targets are corrupted with normally distributed noise of zero mean and 0.04 variance, and a 2-10-10-1 feedforward neural network model with hyperbolic tangent activations on the hidden layers and linear activations on the output layer is applied to learn the parabolic function. Note that this complicated network (151 network parameters in total) will overfit the noisy training data if the unregularized performance function is used with the Levenberg-Marquardt algorithm. This is demonstrated in Figure 2, where we show two cross-sections of the fitted function, at $Y = 0.2$ and $Y = 0.4$.

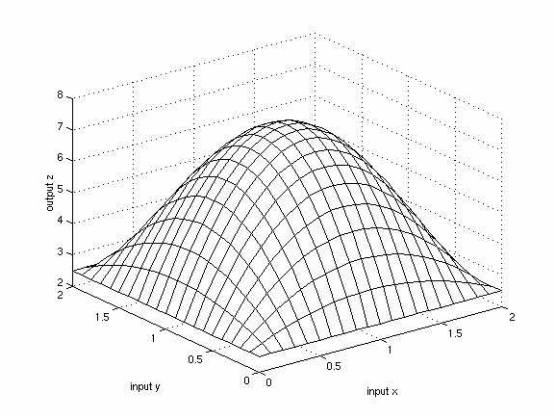

Figure 1 Parabolic Surface

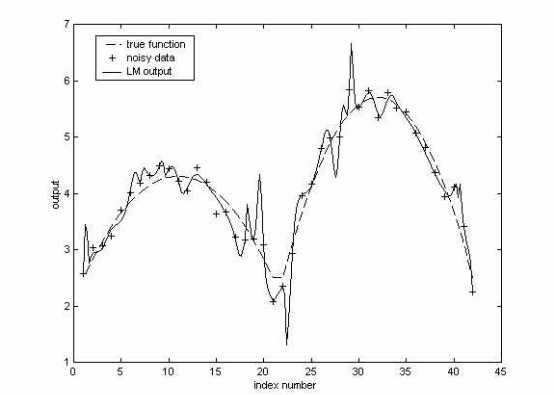

Figure 2 Training Results without Regularization

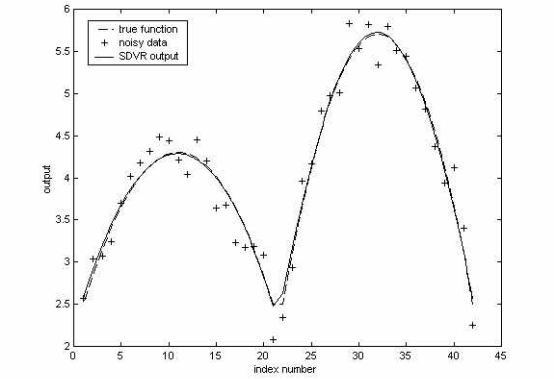

Figure 3 Training Results with SDVR Algorithm

In comparison, results obtained using the SDVR algorithm are displayed in Figure 3. We can see that no overfitting occurs because the appropriate regularization is determined during the adaptive training. Figure 4 shows how the parameter α changes with incremental updating. It starts from 0.01 and takes about 200 epochs to reach convergence. In this example, the parameter μ_α is initially set to 0.05, and is multiplied by 1.05 each time the validation error increases. The convergence of the regularization parameter is indirectly measured by the relative change of the validation error (RCVE) $\left|(F_v(\boldsymbol{w}_{k+1}) - F_v(\boldsymbol{w}_k))/F_v(\boldsymbol{w}_{k+1})\right|$. Here we use a 15-epoch sliding window to monitor the RCVE. The first stage training is terminated if the value of the RCVE between any two adjacent epochs within the window is less than 0.00005.

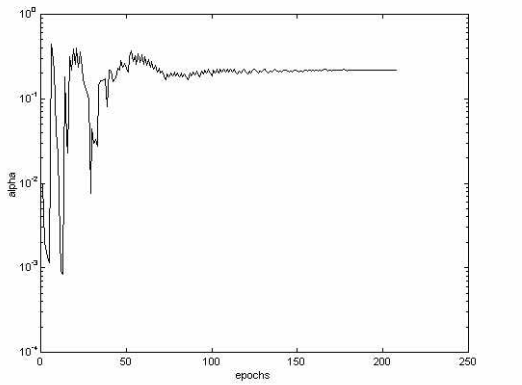

Figure 4 Regularization with Incremental Updating

2.2 Convergent Updating

The convergent updating can be considered as the opposite extreme from the incremental updating, with respect to the updating interval for α. While the incremental updating recalculates the regularization parameter in each training epoch, the convergent updating keeps training with fixed α until the optimization algorithm converges. After that, a new α will be estimated by using the same equation that we used in the incremental updating. In this approach, we consider the convergent updating as the special case of incremental updating initialized independently with the previous parameters. This procedure will repeat several times during the first stage training. The tunable parameter μ_α is set small at the beginning, but will be multiplied by a relatively large constant if the validation error increases at each switching point of α. The first stage training is usually terminated when the value of the RCVE between two adjacent updatings is smaller than a predetermined threshold. Since we train the network to the convergence for each fixed α, the final choice of the optimal regularization parameter will be the one with smallest validation error among all updatings.

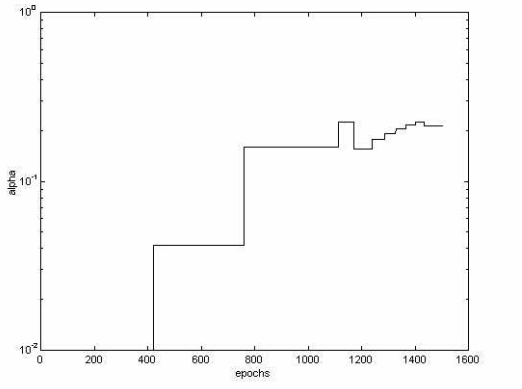

Figure 5 Regularization with Convergent Updating

Figure 5 shows the trajectory of α using convergent updating on our previous problem. This algorithm needs only a few iterations in the outside loop, but requires more training epochs in the inside loop. Starting from $\alpha = 0.01$, the training takes several hundred epochs to reach convergence for the first few updatings of the regularization parameter. As the parameter α approaches to the optimal value, the required number of training epochs with fixed α becomes smaller, since each retraining starts from the previous weight vector. In this example, the stable state of the inside loop training is controlled by a 30-epoch sliding window with 0.00005 RCVE threshold. The other specific settings include 0.05 for initial μ_α, and 10 for increment constant of μ_α.

2.3 Conditional Updating

The conditional updating is proposed as a compromise between the incremental updating and the convergent updating. In this implementation, we optimize the training performance with a fixed parameter α by iteratively updating the weight vector using Eq. (2) until the validation error increases, or the value of RCVE within a sliding window is small enough. After that, a new α is recalculated with Eq. (4), and the training is performed on the new objective function. Working in this way, the conditional updating does not need as many iterations as the incremental updating required in the outside loop. On the other hand, since the conditional updating uses a less expensive stop control in the inside loop, it does not require as many training epochs as the convergent updating needed with fixed α.

Figure 6 illustrates the switchings of α during the first stage training with conditional updating. Beginning at $\alpha = 0.01$, the regularization is adapted quickly due to the increase of the validation error. Thus, the further unnecessary training with the initial α is avoided because it will lead to overfitting. During the transition of α from the less optimal to the optimal, the validation error may increase after a number of training epochs, or its trajectory may be descending but very flat. In the latter case, we can monitor the RCVE by using a shorter sliding window, 10-epochs long for example, to control the switching point of α. In this example, the initial μ_α is the same as that used in the other two cases, but the increment constant of μ_α is 2. The stop criterion in the outside loop for the conditional updating is similar to that for the incremental updating, but uses a shorter sliding window.

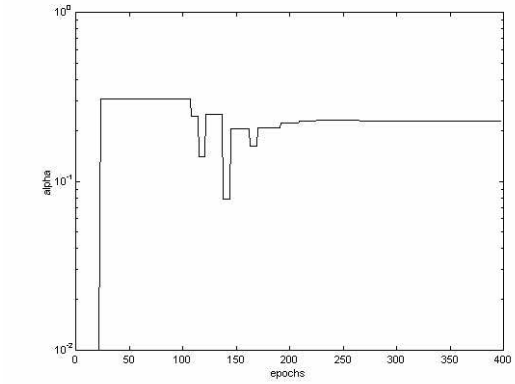

Figure 6 Regularization with Conditional Updating

3. Experiments

In this section, we test the SDVR algorithm under different initial conditions and different data sizes using the same training example of the noise-corrupted parabolic function with the same 2-10-10-1 network structure that we used in the previous section. Thirty trials with normally distributed initial weights of zero mean, 0.01 variance are analyzed under each test condition. The model output is calculated using the resulting weights from the final training with all data included. We use a large testing data set, 2601 samples from a 51 by 51 grid over the input plane, to measure the model performance. The performance index is the mean-squared error (mse) between the model outputs and the true function outputs. The results are averaged over the 30 trials. In addition, we use the number of float point operations (flops) to measure the computation load on the first stage training. We also take a measure of the variation in resulting regularization parameter to see if it is consistent under different test conditions.

The effect of the initial regularization parameter on the training results with the three implementations are summarized from Table 1 to Table 3. The item marked by the overline is the mean value averaged over 30 trials, and the prefix 'std' is the notation for the standard derivation of the item after the underscore. In this test, we use 211 patterns for the training set and 130 patterns for the validation set.

Results	α_0 =1.0	α_0 =0.01	α_0 =0.0001
$\overline{\text{mse}}$	2.57×10^{-3}	2.58×10^{-3}	2.62×10^{-3}
std_mse	1.27×10^{-5}	9.28×10^{-5}	2.04×10^{-4}
$\overline{\text{flops}}$	7.76×10^{9}	7.23×10^{9}	7.07×10^{9}
std_flops	1.28×10^{9}	1.67×10^{9}	1.39×10^{9}
$\overline{\text{alpha}}$	2.78×10^{-1}	2.76×10^{-1}	2.74×10^{-1}
std_alpha	1.20×10^{-2}	1.60×10^{-2}	1.78×10^{-2}

Table 1 SDVR Results with Incremental Updating

Results	α_0 =1.0	α_0 =0.01	α_0 =0.0001
$\overline{\text{mse}}$	2.65×10^{-3}	2.89×10^{-3}	2.92×10^{-3}
std_mse	3.14×10^{-4}	4.40×10^{-4}	4.16×10^{-4}
$\overline{\text{flops}}$	2.69×10^{10}	3.31×10^{10}	6.95×10^{10}
std_flops	4.79×10^{9}	9.17×10^{9}	4.67×10^{10}
$\overline{\text{alpha}}$	2.57×10^{-1}	2.28×10^{-1}	2.33×10^{-1}
std_alpha	2.56×10^{-2}	3.37×10^{-2}	4.45×10^{-2}

Table 2 SDVR Results with Convergent Updating

We can see that within each table, the model performances are consistent no matter what initial α the training starts from. The among-table variations in performance mean and α mean are also insignificant indicating that we can get similar results by using any of three SDVR implementations. However, the computation load for the convergent updating is sensitive to the value of initial α. For the given noise data, the computational cost is high if the training

begins at a very small regularization parameter.

Results	α_0 =1.0	α_0 =0.01	α_0 =0.0001
$\overline{\text{mse}}$	2.71×10^{-3}	2.63×10^{-3}	2.68×10^{-3}
std_mse	3.18×10^{-4}	2.02×10^{-4}	3.13×10^{-4}
$\overline{\text{flops}}$	1.38×10^{10}	9.47×10^{9}	1.07×10^{10}
std_flops	3.75×10^{9}	4.65×10^{9}	4.25×10^{9}
$\overline{\text{alpha}}$	2.68×10^{-1}	2.73×10^{-1}	2.65×10^{-1}
std_alpha	3.14×10^{-2}	3.58×10^{-2}	4.33×10^{-2}

Table 3 SDVR Results with Conditional Updating

Table 4 summarizes how training results change with each implementation as the problem complexity varies. A reasonable index for the problem complexity is the size of the Jacobian matrix, which is the product of the number of training patterns and the number of network parameters. Since the network structure is fixed in this section, we choose three different data sizes to represent the different problem complexities. The data splitting ratio (validation data / training data) is 220/221 for the first data set, 480/481 for the second, and 1300/1301 for the third. The initial regularization parameter is set to 0.01. In Table 4, INC represents incremental updating, CVG refers to convergent updating and CDT is for conditional updating.

Method	No. Data	$\overline{\text{mse}}$	$\overline{\text{flops}}$	$\overline{\text{alpha}}$
INC	441	2.86×10^{-3}	1.28×10^{10}	1.20×10^{-1}
CVG	441	2.49×10^{-3}	3.03×10^{10}	1.27×10^{-1}
CDT	441	2.79×10^{-3}	1.60×10^{10}	1.20×10^{-1}
INC	961	1.82×10^{-3}	1.49×10^{10}	6.43×10^{-1}
CVG	961	1.89×10^{-3}	4.50×10^{10}	6.53×10^{-1}
CDT	961	1.83×10^{-3}	1.49×10^{10}	6.79×10^{-1}
INC	2601	9.97×10^{-4}	5.32×10^{10}	2.50×10^{-1}
CVG	2601	1.03×10^{-3}	6.87×10^{10}	2.61×10^{-1}
CDT	2601	1.04×10^{-3}	2.65×10^{10}	2.71×10^{-1}

Table 4 Comparison of Variations of SDVR Algorithm

It can be concluded from the table that the three SDVR implementations work equally well with respect to the model performance. However, the computation costs are quite different among them. The incremental updating is as efficient as the conditional updating for the first two data sets, but the conditional updating is more cost-effective for the third data set. The convergent updating is much slower than the other two methods for the moderate data size, but for the large data size, its computation cost is close to the incremental updating. Therefore, as a guide to the user, we suggest using either the incremental updating or the conditional updating if the size of the Jacobian matrix is not too large, and using the conditional updating method otherwise. If the computation load is not a big concern, then the convergent updating is always a useful method. The other factors, like data splitting ratio and noise level, may affect the comparison results, but they were not considered in this paper.

4. Conclusions

This paper presented a new framework for adaptation of the regularization parameter. We have introduced the basic SDVR algorithm and two variations. Tests on numerical examples demonstrate that the three SDVR implementations work equally well in providing good generalization under different initial conditions. The tests also indicate how to choose the best approach to reduce the computation load according to the problem complexity.

5. References

[1] D. J. C. MacKay, "Bayesian Interpolation," *Neural Computation*, vol. 4, pp. 415-447,1992.

[2] F. D. Foresee and M. T. Hagan, "Gaussian-Newton Approximation to Bayesian Learning," *Processings of the IEEE International Conference on Neural Networks (ICNN'97)*, vol.3, 1930-1935, Houston, Texas, June 1997.

[3] J. Larsen, L. K. Hansen, C. Svarer and M. Ohlsson, "Design and Regularization of Neural Networks: The Optimal Use of a Validation Set," *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, vol. 4, 62-71, Piscatway, New Jersey, 1996.

[4] J. Larsen, C. Svarer, L. N. Andersen and L. K. Hensen, "Adaptive Regularization in Neural Network Modeling," by fttp://eivind.mm.dtu.dk/dist/1997/larsen.bot.ps.Z.

[5] C. Goutte and J. Larsen, "Adaptive Regularization of Neural Networks Using Conjugate Gradient," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 2, 1201-1204, 1998.

[6] M. T. Hagan, H. B. Demuth and M. Beale, *Neural Network Design*, Boston: PWS Publishing Co., 1996.

[7] F. D. Foresee, *Generalization and Neural Networks*, Ph.D. Dissertation, Oklahoma State University, 1996.