MDL Regularizer: A New Regularizer based on the MDL Principle

Kazumi Saito and Ryohei Nakano
NTT Communication Science Laboratories
2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan
{saito,nakano}@cslab.kecl.ntt.jp

Abstract

This paper proposes a new regularization method based on the MDL (Minimum Description Length) principle. An adequate precision weight vector is trained by approximately truncating the maximum likelihood weight vector. The main advantage of the proposed regularizer over existing ones is that it automatically determines a regularization factor without assuming any specific prior distribution with respect to the weight values. Our experiments using a regression problem showed that the MDL regularizer significantly improves the generalization error of a second-order learning algorithm and shows a comparable generalization performance to the best tuned weight-decay regularizer.

1. Introduction

It has been found that adding some regularization (penalty) term to an objective function in the learning of neural networks can lead to significant improvements in network generalization. Such terms have been proposed on the basis of several viewpoints such as weight-decay [7], function-smoothing [11, 3], weight-pruning [6, 8], and Bayesian priors [9, 17]. Some are calculated by using simple arithmetic operations, while others utilize higher-order derivatives.

Most of these methods require an adequate regularization factor be determined by using some method which can reasonably estimate the generalization performance. Resampling methods such as cross-validation [16] or bootstrap [4] have been widely used for this purpose, but they require a large amount of computation even for mid-scale problems. On the other hand, Bayesian methods can automatically determine a regularization factor (hyperparameter value) by assuming some specific prior distribution with respect to the weight values; however, we generally cannot know a suitable prior distribution in advance.

Standard statistical criteria such as AIC [1] or MDL [12] cannot be applied directly to determine the regularization factor, because they only compromise on the number of model parameters. Quite recently, however, it has been shown in the context of the VC-dimension that the model complexity of neural networks increases when we allow a wider range of weight values [2]. Thus, it would be expected that some statistical criterion can be used to automatically determine a value of the regularization factor without assuming any specific prior distribution.

This paper proposes a new regularization method based on the MDL principle. Section 2 explains the proposed MDL regularizer and how to embed it in a second-order learning algorithm. Section 3 shows experimental results for a regression problem and evaluates the generalization performance of the MDL regularizer in comparison to a weight-decay regularizer.

2. Regularization technique based on MDL principle

2.1. Background

Let $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_m, y_m)\}$ be a set of examples, where $\mathbf{x}_t$ denotes an n-dimensional input vector and y_t a target value corresponding to $\mathbf{x}_t$. In a three-layer neural network, let h be the number of hidden units, $\mathbf{w}_j$ $(j = 1, \cdots, h)$ be the weight vector between all the input units and the hidden unit j, and $\mathbf{w}_0 = (w_{00}, \cdots, w_{0h})^T$ be the weight vector between all the hidden units and the output unit; w_{j0} means a bias term and x_{t0} is set to 1. Note that $\mathbf{a}^T$ denotes the transposed vector of $\mathbf{a}$. Hereafter, a vector consisting of all parameters, $(\mathbf{w}_0^T, \cdots, \mathbf{w}_h^T)^T$, is simply expressed as $\mathbf{\Phi} = (\phi_1, \cdots, \phi_N)^T$, where $N(= nh + 2h + 1)$ denotes the dimension of $\mathbf{\Phi}$. Then, the output value of

the three-layer neural network can be described as

$$z(\mathbf{x}_t; \mathbf{\Phi}) \quad = \quad w_{00} + \sum_{j=1}^{h} w_{0j} f(\mathbf{w}_j^T \mathbf{x}_t), \qquad (1)$$

where

$$f(v) \quad = \quad \frac{1}{1+e^{-v}}.$$

Now, we assume that each value of y was corrupted by Gaussian noise with a mean of 0 and an unknown standard deviation of σ. Then, the learning problem in neural networks can be formalized as the maximum likelihood problem which maximizes the logarithmic likelihood

$$\begin{aligned} &\sum_{t=1}^{m} \log P(y_t|\mathbf{x}_t; \mathbf{\Phi}, \sigma^2) \\ &\quad = \quad -\frac{m}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{t=1}^{m} \left(y_t - z(\mathbf{x}_t; \mathbf{\Phi})\right)^2 . \quad (2) \end{aligned}$$

Here, since the maximum likelihood weight vector, $\hat{\mathbf{\Phi}}$, which maximizes (2), is nothing but the least-squares estimate for the second term of (2), it can be calculated by using some learning algorithm.

2.2. MDL regularizer

The intuitive idea of the MDL principle [12] can be explained as a communication problem where a sender wishes to transmit a data set to a receiver using a message of the shortest possible length. In regression problems, we suppose that the input data values are already known to the receiver, since we are trying to predict the output data, using the input data.

Now, by sending an adequate weight vector, the receiver can estimate some approximated output values. Thus, by sending only the difference between the actual and approximated values, the receiver can obtain the complete output data values. In the case of the maximum likelihood weight vector $\hat{\mathbf{\Phi}}$, the message length for the difference values can be optimally given by $-\sum_{t=1}^{m} \log P(y_t|\mathbf{x}_t; \hat{\mathbf{\Phi}}, \sigma^2)$; however, the message length for $\hat{\mathbf{\Phi}}$ usually requires a large number of bits, due to its high precision. To compromise these two message lengths, Rissanen [12] proposed a criterion for deriving the standard MDL criterion, by introducing a truncated weight vector $\bar{\mathbf{\Phi}}$ for $\hat{\mathbf{\Phi}}$. Here, Rissanen's criterion is almost equivalent to the following one.

$$\min_{\bar{\mathbf{\Phi}}, \sigma^2} \left\{ -\sum_{t=1}^{m} \log P(y_t|\mathbf{x}_t; \bar{\mathbf{\Phi}}, \sigma^2) - \sum_{k=1}^{N} \log |\hat{\phi}_k - \bar{\phi}_k| \right\} \qquad (3)$$

Indeed, the second term goes down if we use a coarser precision, while the first term generally increases because the coarser truncated vector can deviate more from $\hat{\mathbf{\Phi}}$.

On the basis of the criterion (3), we propose a new regularization technique. For any truncated weight vector $\bar{\mathbf{\Phi}}$, there exists an N-dimensional vector $\mathbf{u} = (u_1, \cdots, u_N)^T$ such that $\bar{\phi}_k = f(u_k)\hat{\phi}_k$. Here, recall that $f(u_k)$ represents a sigmoidal function, $f(u_k) = 1/(1+e^{-u_k})$, and $0 < f(u_k) < 1$. Conversely, for any vector $\mathbf{u}$, $f(u_k)\hat{\phi}_k$ gives an approximation to some truncated weight value $\bar{\phi}_k$. Thus, the following criterion can be regarded as a continuous version of (3).

$$\begin{aligned} &\min_{\mathbf{u}, \sigma^2} \left\{ -\sum_{t=1}^{m} \log P(y_t|\mathbf{x}_t; \bar{\mathbf{\Phi}}(\mathbf{u})), \sigma^2) \right. \\ &\qquad \left. -\sum_{k=1}^{N} \log |\hat{\phi}_k - f(u_k)\hat{\phi}_k| \right\} \\ &\equiv \quad \min_{\mathbf{u}, \sigma^2} \left\{ \frac{m}{2} \log 2\pi\sigma^2 + \frac{1}{2\sigma^2} \sum_{t=1}^{m} \left(y_t - z(\mathbf{x}_t; \bar{\mathbf{\Phi}}(\mathbf{u}))\right)^2 \right. \\ &\qquad \left. -\sum_{k=1}^{N} \log(1 - f(u_k)) - \sum_{k=1}^{N} \log |\hat{\phi}_k| \right\} \qquad (4) \end{aligned}$$

Note that the fourth term, $\sum_{k=1}^{N} \log |\hat{\phi}_k|$, is constant and independent of the minimization process. Incidentally, directly minimizing (3) with respect to $\bar{\mathbf{\Phi}}$ will be a tough constrained minimization problem, because we must always assure that $\bar{\mathbf{\Phi}}$ is a truncation of $\hat{\mathbf{\Phi}}$.

In the minimization problem defined by (4), since there are two kinds of parameters, $\mathbf{u}$ and σ^2, we employ a coordinate descent strategy. Namely, for a fixed $\mathbf{u}$, by differentiating (4) with respect to σ^2 and setting it to 0, we obtain the relative minimum for σ^2 as follows:

$$\hat{\sigma}^2 \quad = \quad \frac{1}{m} \sum_{t=1}^{m} \left(y_t - z(\mathbf{x}_t; \bar{\mathbf{\Phi}}(\mathbf{u}))\right)^2 . \qquad (5)$$

Conversely, for a fixed $\hat{\sigma}^2$, by ignoring independent terms and multiplying $\hat{\sigma}^2$, (4) can be equivalently transformed to a problem which minimizes the following objective function.

$$\begin{aligned} F(\mathbf{u}) \;&=\; F_1(\mathbf{u}) + \hat{\sigma}^2 F_2(\mathbf{u}) \qquad (6) \\ \text{where} \qquad F_1(\mathbf{u}) \;&=\; \frac{1}{2} \sum_{t=1}^{m} \left(y_t - z(\mathbf{x}_t; \bar{\mathbf{\Phi}}(\mathbf{u}))\right)^2 , \\ F_2(\mathbf{u}) \;&=\; -\sum_{k=1}^{N} \log(1 - f(u_k)). \end{aligned}$$

Here, we can see that the second derivative of $F_2(\mathbf{u})$

with respect to u_k is always positive, since

$$\begin{aligned}\frac{\partial F_2(\mathbf{u})}{\partial u_k} &= f(u_k),\\ \frac{\partial^2 F_2(\mathbf{u})}{\partial u_k^2} &= f(u_k)(1-f(u_k)) > 0.\end{aligned}$$

Therefore, the second term of (6) can be regarded as a regularization term whose regularization factor is automatically determined by (5).

2.3. Second-order learning algorithm

In order to minimize the objective function defined by (6), we employ a newly invented second-order learning algorithm based on a quasi-Newton method [5], called BPQ [14], where the descent direction, $\Delta\mathbf{u}$, is calculated on the basis of a partial BFGS update and a reasonably accurate step-length, λ, is efficiently calculated as the minimal point of a second-order approximation. In first-order learning algorithms [13] which calculate the search direction as the gradient direction, a large number of iterations are often required until convergence. On the other hand, in existing second-order methods [5] which converge more quickly by using both gradient and curvature information, it is difficult to suitably scale up for large problems, and a large amount of computation is required for calculating the optimal step-length. The newly developed second-order algorithm called BPQ can be reasonably scaled up by introducing a storage space parameter, and the computational complexity for calculating the optimal step-length is reasonably small, almost equivalent to that of gradient vector evaluation. Incidentally, we can employ other learning algorithms, but BPQ has worked the most efficiently among several representative algorithms in our experiences [14].

In BPQ, the descent direction is calculated from the gradient vector, by directly applying the partial BFGS update. Here, the derivative of (6) with respect to u_k is calculated by

$$\begin{aligned}\frac{\partial F(\mathbf{u})}{\partial u_k} &= \frac{\partial F_1(\mathbf{u})}{\partial u_k} + \hat{\sigma}^2\frac{\partial F_2(\mathbf{u})}{\partial u_k}\\ \frac{\partial F_1(\mathbf{u})}{\partial u_k} &= \frac{\partial F_1(\bar{\mathbf{\Phi}})}{\partial \bar{\phi}_k}\frac{\partial \bar{\phi}_k(u_k)}{\partial u_k}\\ &= \frac{\partial F_1(\bar{\mathbf{\Phi}})}{\partial \bar{\phi}_k} f(u_k)(1-f(u_k))\hat{\phi}_k\\ \frac{\partial F_2(\mathbf{u})}{\partial u_k} &= f(u_k)\end{aligned}$$

where $\frac{\partial}{\partial \bar{\phi}_k}F_1(\bar{\mathbf{\Phi}})$ can be calculated by using the standard back propagation algorithm.

In order to explain the step-length calculation, we introduce Pearlmutter's operators [10] defined by

$$\begin{aligned}\Re_{\Delta\mathbf{u}}\{F(\mathbf{u})\} &= \frac{\partial}{\partial\lambda}F(\mathbf{u}+\lambda\Delta\mathbf{u})_{|\lambda=0},\\ \Re^2_{\Delta\mathbf{u}}\{F(\mathbf{u})\} &= \frac{\partial^2}{\partial\lambda^2}F(\mathbf{u}+\lambda\Delta\mathbf{u})_{|\lambda=0}.\end{aligned}$$

Then, when $\Re_{\Delta\mathbf{u}}\{F(\mathbf{u})\} < 0$ and $\Re^2_{\Delta\mathbf{u}}\{F(\mathbf{u})\} > 0$, the optimal step-length of a second-order approximation to the objective function is expressed by

$$\lambda = -\frac{\Re_{\Delta\mathbf{u}}\{F(\mathbf{u})\}}{\Re^2_{\Delta\mathbf{u}}\{F(\mathbf{u})\}}.$$

Here, the method for coping with the other cases is exactly the same as described in [14].

In the case of the objective function defined by (6), the first-order derivative is calculated as follows:

$$\begin{aligned}\Re_{\Delta\mathbf{u}}\{F(\mathbf{u})\} &= \Re_{\Delta\mathbf{u}}\{F_1(\mathbf{u})\} + \hat{\sigma}^2\Re_{\Delta\mathbf{u}}\{F_2(\mathbf{u})\}\\ \Re_{\Delta\mathbf{u}}\{F_1(\mathbf{u})\} &= \left(\nabla_{\bar{\mathbf{\Phi}}}F_1(\bar{\mathbf{\Phi}}(\mathbf{u}))\right)^T\bar{\mathbf{\Phi}}'(\mathbf{u})\\ \Re_{\Delta\mathbf{u}}\{F_2(\mathbf{u})\} &= \sum_{k=1}^{N} f(u_k)\Delta u_k\end{aligned}$$

On the other hand, the second-order derivative is calculated as follows:

$$\begin{aligned}&\Re^2_{\Delta\mathbf{u}}\{F(\mathbf{u})\}\\ &\quad= \Re^2_{\Delta\mathbf{u}}\{F_1(\mathbf{u})\} + \hat{\sigma}^2\Re^2_{\Delta\mathbf{u}}\{F_2(\mathbf{u})\}\\ &\Re^2_{\Delta\mathbf{u}}\{F_1(\mathbf{u})\}\\ &\quad= \left(\Re_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}\right)^T\left(\nabla^2_{\bar{\mathbf{\Phi}}}F_1(\bar{\mathbf{\Phi}})\right)\Re_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}\\ &\qquad+\left(\nabla_{\bar{\mathbf{\Phi}}}F_1(\bar{\mathbf{\Phi}})\right)^T\Re^2_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}\\ &\quad= \Re^2_{\Re_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}}\{F_1(\bar{\mathbf{\Phi}})\}\\ &\qquad+\left(\nabla_{\bar{\mathbf{\Phi}}}F_1(\bar{\mathbf{\Phi}})\right)^T\Re^2_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}\\ &\Re^2_{\Delta\mathbf{u}}\{F_1(\mathbf{u})\}\\ &\quad= \sum_{k=1}^{N} f(u_k)(1-f(u_k))(\Delta u_k)^2\end{aligned}$$

Here, by regarding $\Re_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}$ as the search direction, $\Re^2_{\Re_{\Delta\mathbf{u}}\{\bar{\mathbf{\Phi}}(\mathbf{u})\}}\{F(\bar{\mathbf{\Phi}})\}$ can be calculated as the second-order derivative with respect to a standard three-layer neural network; thus, we can use BPQ [14]. Therefore, by noting that

$$\begin{aligned}\Re_{\Delta u_i}\{\bar{\phi}_i(u_i)\} &= f(u_i)(1-f(u_i))\hat{\phi}_i\Delta u_i,\\ \Re^2_{\Delta u_i}\{\bar{\phi}_i(u_i)\} &= \Re_{\Delta u_i}\{\bar{\phi}_i(u_i)\}\\ &\quad\times(1-2f(u_i))\hat{\phi}_i(\Delta u_i).\end{aligned}$$

we can see that a reasonably accurate step-length is calculated efficiently.

2.4. Learning algorithm with MDL regularizer

In the coordinate method described above, $\hat{\sigma}^2$ can be updated by using (5) at any time while minimizing the objective function defined by (6). However, since BPQ is based on a quasi-Newton method, the best opportunity for updating $\hat{\sigma}^2$ is when an approximation to the inverse Hessian matrix is restarted by discarding the accumulated search information. Let s be the value of the partiality parameter used for the partial BFGS update; then, the proposed algorithm can be summarized as follows:

step 1: Calculate the maximum likelihood weight vector $\hat{\mathbf{\Phi}}$;
step 2: Initialize $\mathbf{u}$, calculate $\hat{\sigma}^2$ by using (5), and set $k = 1$;
step 3: Terminate the iteration if a stopping criterion is satisfied;
step 4: Calculate the current descent direction $\Delta\mathbf{u}_k$, calculate the step-length λ_k, and update the vector $\mathbf{u}$;
step 5: If $k \equiv 0 \pmod{s}$, update $\hat{\sigma}^2$ by using (5);
step 6: Set $k = k + 1$, and return to **step 3**;

3. Evaluation by experiments

3.1. Regression problem

By using a regression problem for a function

$$y \quad = \quad (1 - x + 2x^2)e^{-0.5x^2},$$

the performance of the proposed MDL regularizer was evaluated. In the experiment, the value of x was randomly generated in the range of $[-4, 4]$, and the corresponding value of y was calculated from x; each value of y was corrupted by adding Gaussian noise with a mean of 0 and a standard deviation of 0.2. The total number of training examples was set to 30, and the number of hidden units was set to 10.

In learning without the regularizer to obtain the maximum likelihood weight vector, the initial values for the weights between the input and hidden units were independently generated according to a normal distribution with a mean of 0 and a standard deviation of 1; the initial values for the weights between the hidden and output units were set to 0, but the bias value at the output unit was initially set to the average output value of all training examples. The iteration of learning was terminated when the gradient vector was sufficiently small,

$$\|\nabla F_1(\mathbf{\Phi})\|^2/N \quad < \quad 10^{-8}.$$

In learning with the MDL regularizer, the initial values for the vector $\mathbf{u}$ were set to the same value such that $f(u_k) = 0.9$. The iteration of learning was terminated when the gradient vector was sufficiently small,

$$\|\nabla F(\mathbf{u})\|^2/N \quad < \quad 10^{-8},$$

and the modification value to σ^2 was sufficiently small,

$$\|\sigma^2 - 2F_1(\mathbf{u})/m\| \quad < \quad 10^{-4}\sigma^2.$$

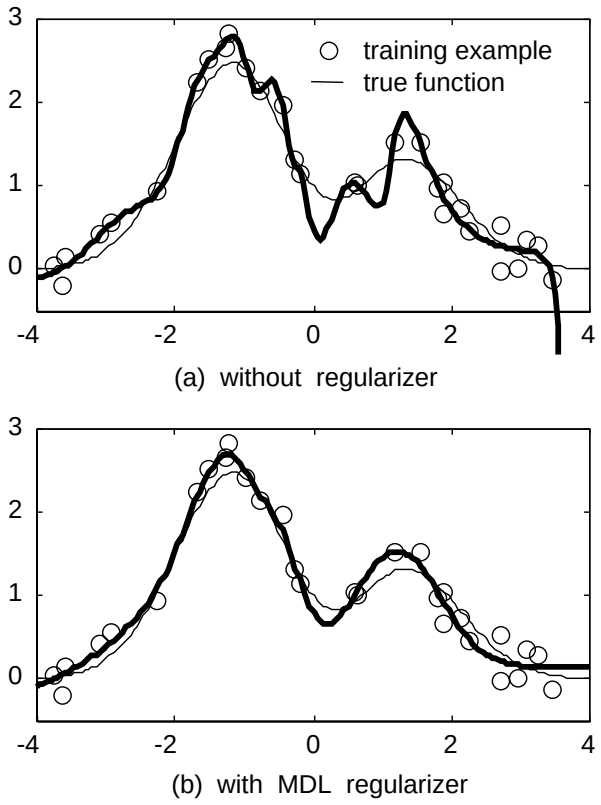

Figure 1. Curves obtained

3.2. Learning results

Figure 1(a) shows a function obtained through learning without the regularizer, together with the training examples and the true function. We can see that the result was over-fitted to the training examples to some degree. On the other hand, Figure 1(b) shows a function obtained through learning with the MDL regularizer. Clearly, the result with the regularizer more

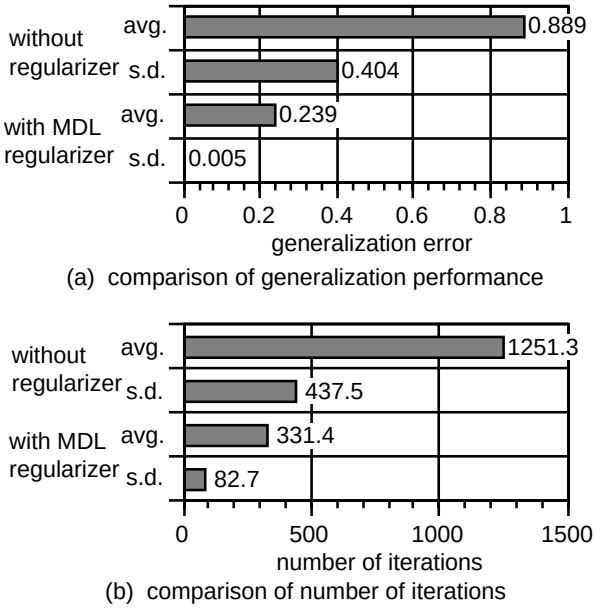

Figure 2. Performance of MDL regularizer

closely approximated to the true function than the result without it.

Figure 2(a) compares the generalization performance with and without the MDL regularizer, where trials were performed 10 times. Here, the generalization performance was evaluated by using the average RMSE (root mean squared error) for a set of $5,000$ test examples. The best possible RMSE level is 0.2 because each test example includes the same amount of Gaussian noise as given to a training example. The average generalization error of the MDL regularizer was 3.7 times as small as that without it. Moreover, by applying the proposed regularizer, the standard deviation got much smaller. This indicates that the MDL regularizer played a leading role in stabilizing the learning performance. Figure 2(b) compares the number of iterations required for the learning process with and without the regularizer. This figure shows that the MDL regularizer shortened the number of iterations by 3.8 times on average.

3.3. Comparison with weight-decay regularizer

To evaluate how much the proposed regularizer improves the generalization performance, we compared it with a weight-decay regularizer which employs a squared penalty term Ω defined by

$$\Omega \quad = \quad \mu \sum_{k=1}^{N} \phi_k^2.$$

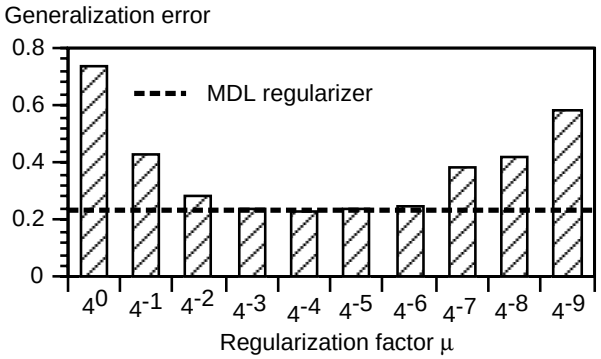

Figure 3. MDL regularizer vs. weight-decay regularizer

In our previous experience [15], this term gave the best result among other simple terms such as absolute penalty or normalized penalty. In the experiments, the regularization factor μ was changed from 4^0 to 4^{-9} by multiplying by 4^{-1}; trials were performed 10 times for each regularization factor.

Figure 3 compares the average generalization performance of MDL and weight-decay regularizers. This figure shows that the MDL regularizer has the best performance for any of the regularization factors of the weight-decay regularizer; only the weight-decay regularizer with adequate factors worked comparably. Since the performance of the weight-decay regularizer depends on its regularization factors, finding suitable factors mean a considerable amount of computation if we have little prior knowledge about the problems. Thus, it has been shown that the proposed technique is a promising approach for learning neural networks with a regularizer.

4. Conclusion

This paper proposed a new regularization method based on the MDL principle. In the MDL regularizer, an adequate precision weight vector is trained by approximately truncating the maximum likelihood weight vector. The main advantage of the proposed regularizer over existing ones is that it automatically determines a regularization factor achieving excellent generalization without assuming any prior distribution. Our experiments using a regression problem showed that the MDL regularizer significantly improves the generalization error of a second-order learning algorithm and shows a comparable performance to the best tuned weight-decay regularizer. In the future, we plan to do further evaluations using a wider variety of problems.

References

[1] H. Akaike. A new look at the statistical model identification. *IEEE Trans. Autom. Contor.*, AC-19:716–723, 1973.

[2] P. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. Technical report, Australian National University, 1996.

[3] C. Bishop. *Neural networks for pattern recognition*. Oxford Press, 1995.

[4] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. Chapman & Hall, New York, 1993.

[5] P. Gill, W. Murray, and M. Wright. *Practical optimization*. Academic Press, London, 1981.

[6] S. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 177–185, San Mateo, CA, 1989. Morgan Kaufmann.

[7] G. Hinton. Learning translation invariant recognition in massively parallel networks. In J. de Bakker, A. Nijman, and P. Treleaven, editors, *Proc. PARLE Conference on Parallel Architectures and Languages Europe*, pages 1–13, Berlin, 1987. Springer-Verlag.

[8] M. Ishikawa. A structural learning algorithm with forgetting of link weight. Technical report, Tech. Rep. TR-90-7, Electrotechnical Lab. Tsukuba-City, Japan, 1990.

[9] D. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[10] B. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

[11] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[12] J. Rissanen. *Stochastic complexity in statistical inquiry*. World Scientific, 1989.

[13] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClellandnd, editors, *Parallel Distributed Processing*, pages 318–362. MIT Press, 1986.

[14] K. Saito and R. Nakano. Partial BFGS update and efficient step-length calculation for three-layer neural networks. *Neural Computation*, 9(1):123–141, 1997.

[15] K. Saito and R. Nakano. Second-order learning algorithm with squared penalty term. In *Advances in Neural Information Processing Systems*, volume 9, 1997.

[16] M. Stone. Cross-validation: A review. *Operationsforsch. Statist. Ser. Statistics B*, 9(1):111–147, 1978.

[17] P. Williams. Bayesian regularization and pruning using a Laplace prior. *Neural Computation*, 7(1):117–143, 1995.