

On the Robustness of Update Strategies for the Bayesian Hyperparameter α

Jan Poland

Nov 2, 2001

Abstract

Many practical realizations of Bayesian Regularization perform an update of the hyperparameters α and β after each training cycle. However, the most popular update algorithm fails to produce robust iterates if there is not much training data. This behavior is being studied and compared to an alternative update formula, which does not have this shortfall.

Keywords: Neural Networks, Weight Decay, Bayesian Regularization, Numerical Properties

1 Introduction

Bayesian Regularization is a powerful add-on to the training of Neural Networks which enables an automatic optimization of the weight decay parameter. Thus, both overfitting and oversmoothing are prevented efficiently. If used together with a Levenberg-Marquardt training algorithm, Bayesian Regularization implies very little additional computational costs, since it exploits the approximation to the Hessian (see [1]). Therefore, the weight decay can be updated easily after each training cycle. This variant is quite popular in practical implementations.

However, the most common update formula for the weight decay does not produce robust iterates if there is not much training data available. The aim of this work is demonstrating this effect. An alternative update formula which is likewise simple does not have this shortcoming, as we will see. Considering the fact that the common update formula is recommended in important publications and used in many implementations including the popular MATLAB Neural Networks Toolbox ([2]), we hope that

this presentation will be helpful for practitioners.

Bayesian Regularization for Neural Network training performs a splitting of the weight decay parameter δ into two positive numbers α and β , called *hyperparameters*. Here, β^{-1} is the estimated noise amplitude in the data and $\delta = \frac{\alpha}{\beta}$ is the weight decay parameter (see [3], [4], a very good introduction is in [5]). Network training then consists in minimizing the combined objective function

$$\begin{aligned} S(w) &= \frac{\beta}{2} \sum_{i=1}^N (y(x_i, w) - t_i)^2 + \frac{\alpha}{2} \sum_{i=1}^W w_i^2 \\ &= \beta \cdot SSE + \alpha \cdot SSW, \end{aligned}$$

where $(x_i, t_i)_{i=1}^N$ is the training data, $w \in \mathbb{R}^W$ contains the weights and biases and $y(x, w)$ is the network output if the input is x and the weight vector is w . Then, in a Bayesian framework, for the *optimal estimates* of the hyperparameters α and β the following equations are valid, if also w assumes its most probable value w_{MP} :

$$\begin{aligned} \alpha &= \frac{\gamma}{2 \cdot SSW} \quad \text{and} \\ \beta &= \frac{N - \gamma}{2 \cdot SSE}, \quad \text{where} \\ \gamma &= W - \alpha \cdot \text{trace}(H^{-1}) \end{aligned}$$

is the "effective number of well-determined parameters" and $H = \nabla \nabla S(w_{MP})$ is the Hessian of $S(w)$ at the optimum w_{MP} (see e.g. [5]).

2 The numerical problem

The common update strategy just employs the above equations after each training cycle in order to calculate γ using an approximation to the Hessian

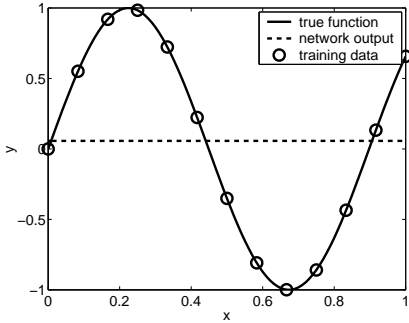


Figure 1: When there is not much training data, the usual update formula results in a problem.

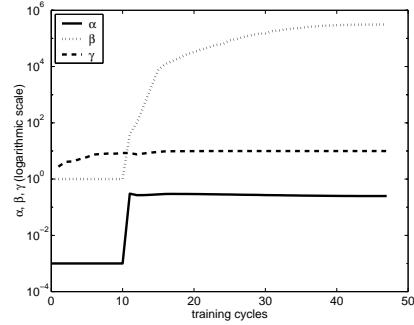


Figure 3: When update starts only after generation 10, the behaviour of the hyperparameters is nicer.

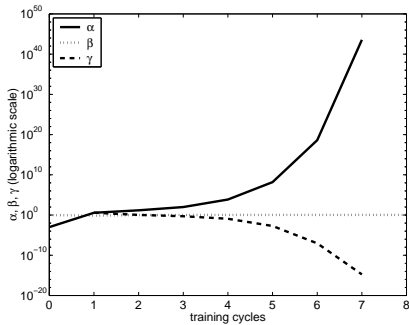


Figure 2: This is how the hyperparameters behave in the problem case.

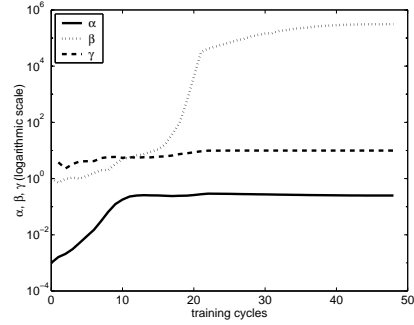


Figure 4: The hyperparameters with the alternative update formula.

and then update α and β . We used this method to train a two-layer network with three hidden tanh units and linear output. The training data consists of 13 points generated by a sine function shown in figure 1. The network output almost vanishes, although there are more observations than free parameters, since the network contains 10 trainable connections. Even if we duplicate the number of data points, the same effect is often observable, depending on the random initial weights.

Figure 2 displays the behaviour of the hyperparameters in this case. Before training, they have been initialized to $\alpha = 10^{-3}$ and $\beta = 1$. The weights have small initial values, so SSW is quite small. After the first training cycle, γ does not vanish, because at least the bias of the output unit is "well-determined" and α is not too large. Thus, the update $\alpha_{new} = \frac{\gamma}{2 \cdot SSW}$ yields a relatively large value. On the other hand, the update β_{new} remains

small, thus the next training cycle decreases the weights. If α remains sufficiently large and β is not too large, the training has the effect of minimizing $\sum w_i^2$, thus the weights converge to zero exponentially fast. Then the eigenvalues of A are almost constant, and even for large α we have in each step $\gamma \simeq \alpha^{-1}$. This implies $\alpha_{new} \approx (\alpha \cdot SSW)^{-1}$ after each training cycle and therefore $\alpha \rightarrow \infty$ exponentially fast.

Often it is recommended not to regularize the bias of the network output, since otherwise adding a constant to the target data t_i would alter the results of network training. However, instabilities occur also if the bias is not regularized, in particular the Hessian can become singular. Also in this case the same vanishing network output is observed.

A possibility to fix this problem consists in updating α only when the weights are already near their optimal values. Figure 3 shows the develop-

ment of the hyperparameters, if there is no update before training cycle 10. The training process now converges to a "sensible" network, the output of which is almost identical to the true function.

3 The alternative formula

Another strategy to cope with the problem is an alternative update formula for α . This can be derived by applying the EM (Expectation Maximization) algorithm (see [6]) or by using a free energy approach (see [7]). Yet another very simple derivation is the following: Consider the equation

$$\alpha = \frac{\gamma}{2 \cdot SSW} = \frac{W - \alpha \cdot \text{trace}(H^{-1})}{2 \cdot SSW}.$$

If we solve this equation, we get the alternative update formula

$$\alpha_{new} = \frac{W}{2 \cdot SSW + \text{trace}(H^{-1})}.$$

(Note that the "old" α is still part of the right hand side, since H depends on α .) Since $\text{trace}(H^{-1}) = \frac{W - \gamma}{\alpha}$, for small SSW we approximately have the update

$$\alpha_{new} = \alpha \cdot \frac{W}{W - \gamma}.$$

Thus, α does not "jump" as before, but behaves in a smoother manner. Figure 3 shows the situation if this update formula is used after each training cycle. Again the resulting network output is almost identical to the true function.

One can easily verify that the alternative update formula increases α if and only if the usual one increases α . Moreover, since

$$\frac{\gamma}{2 \cdot SSW} - \frac{W}{2 \cdot SSW + \text{trace}(H^{-1})} > 0$$

if and only if

$$\frac{\gamma}{2 \cdot SSW} - \alpha > 0$$

and the same holds for "<", the change of α is always more moderate under the alternative update formula. One can expect the alternative update formula therefore to have more robust but sometimes slower convergence properties.

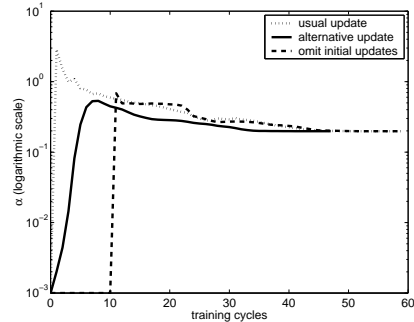


Figure 5: The development of α with more, more complex and noisy training data.

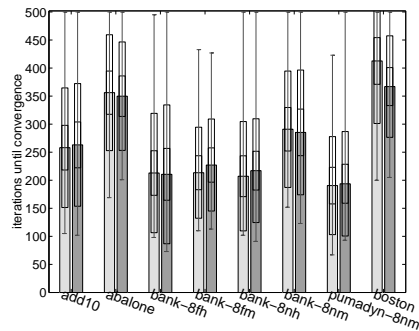


Figure 6: Benchmark problems: training time until convergence of the alternative update strategy (light bars) compared to the common one (dark bars).

4 Experimental verifications

An example with a larger data set is shown in figure 5. A two-layer network with six hidden tanh units, linear outputs and shortcut connections was trained with 441 data points generated by a more complex function with two inputs and one output. We can observe that with the usual update formula the first guess of α is definitely too large. If the updates before training cycle 10 are omitted, the guess is better. On the other hand, with the alternative update formula the curve is smoother and the training converges faster (after 47 instead of 59 training cycles).

Next, we consider eight different benchmark problems from the DELVE suite [8]. Figure 6 compares the numbers of training cycles until con-

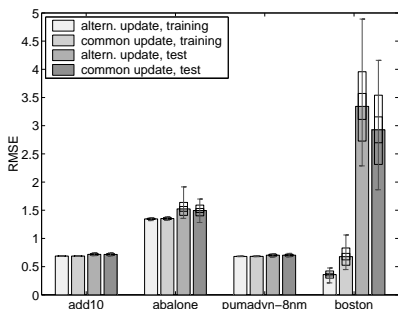


Figure 7: Comparison of training and test errors.

vergence (maximum 500 cycles) of the usual and the alternative update formula. The data has been splitted in 90% for training and 10% for test. For each update strategy, 30 independent networks have been trained, the figure displays the mean, the standard deviations, the 95%-confidence intervals and the minimum and maximum values. There is no significant difference between common and alternative update formula. The same is true for the training and test errors for all data sets except "boston", wich is the smallest one containing 506 patterns (figure 7 shows the values for only four cases).

5 Conclusions

The conclusion of this study is a definite recommendation to use the alternative formula for updating the hyperparameter α in Bayesian Regularization instead of the common formula. In particular when there is not very much training data, the alternative update formula is clearly better. But even with much training data, there is no advantage using the common update formula. Although the common formula works if initial updates are omitted, this strategy is not optimal since a new parameter (the number of steps after which the updates start) has to be introduced.

This recommendation is in contrast to the case of RBF networks, where [6] prefers the common formula to the alternative one because of convergence speed. Further, we recommend to use one weight decay parameter to regularize all weights including the output bias, if there is not very much training data available, since this yields a robust training.

The problem with the output bias stated above can be addressed by rescaling the target data t_i to mean zero and variance one before training.

References

- [1] D. Foresee and M. Hagan. Gauss-newton approximation to bayesian learning. In *Proceedings of the 1997 International Joint Conference on Neural Networks*, 1997.
- [2] H. Demuth and M. Beale. *Neural Network Toolbox User's Guide, Sixth Printing Revised for Version 4*. The Math Works Inc., 3 Apple Hill Drive, Natick, MA 01760-2098, 2000.
- [3] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.
- [4] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472, 1992.
- [5] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- [6] M. J. L. Orr. Recent advances in radial basis function networks. Technical report, Institute for Adaptive and Neural Computation, Edinburgh University, UK, June 1999.
- [7] D. J. C. MacKay. Comparison of approximate methods for handling hyperparameters. *Neural Computation*, 11(5):1035–1068, 1999.
- [8] C. E. Rasmussen et al. DELVE - data for evaluating learning in valid experiments. <http://www.cs.toronto.edu/~delve>.