Nondeterministic Discretization of Weights Improves Accuracy of Neural Networks

Marcin Wojnarski

Warsaw University, Faculty of Mathematics, Informatics and Mechanics
ul. Banacha 2, 02-097 Warszawa, Poland
mwojnars@ns.onet.pl

Abstract. The paper investigates modification of backpropagation algorithm, consisting of discretization of neural network weights after each training cycle. This modification, aimed at overfitting reduction, restricts the set of possible values of weights to a discrete subset of real numbers, leading to much better generalization abilities of the network. This, in turn, leads to higher accuracy and a decrease in error rate by over 50% in extreme cases (when overfitting is high).

Discretization is performed nondeterministically, so as to keep expected value of discretized weight equal to original value. In this way, global behavior of original algorithm is preserved. The presented method of discretization is general and may be applied to other machine-learning algorithms. It is also an example of how an algorithm for continuous optimization can be successfully applied to optimization over discrete spaces. The method was evaluated experimentally in WEKA environment using two real-world data sets from UCI repository.

Keywords: Generalization, Overtraining, Overfitting, Regularization.

1 Introduction

Multi-layer artificial neural networks [1,2] are well-established tools in machine learning, with proven effectiveness in many real-world problems. However, there are still many tasks in which they perform worse than other machine-learning systems [3]. One of the reasons is that neural networks contain usually thousands of real-valued adaptive parameters, and so they have strong tendency to get *overtrained* (*overfitted*), especially when the size of the training set is not large enough. Thus, methods to improve generalization abilities of neural networks are necessary.

Several such methods have been already proposed: early stopping [2] – the simplest and most commonly used – which consists of finishing the training process when the error on a validation set starts increasing; regularization [4,5,6] based on adding a regularization term to the error function; pruning [4,7], i.e. removing unnecessary weights or neurons during or after training; training with noise [5,8,9], i.e. disturbing training instances in a random way; weight sharing [10].

This paper introduces a novel method based on discretization of weights. The method is easy to implement and potentially more versatile than existing ones, yet it can lead to significant improvement in generalization abilities and accuracy of the neural network. It can be also used in conjunction with the above-mentioned algorithms.

Motivation which underlies the presented method is described in Sect. 2. It is followed by presentation of the algorithm in Sect. 3 and its experimental assessment in Sect. 4. Finally, Sect. 5 recaps main points of the paper and presents conclusions.

2 Motivation

Discretization of weights means restricting the set of possible values of neuron weights to a small discrete subset of real numbers. In this way, the decision model represented by a neural network gets simpler and can be described using fewer number of bits, since every weight may be represented, for instance, by a single byte instead of four or eight bytes. This in turn leads to better generalization abilities.

Theoretical justification of the method is provided by Rissanen's Minimum Description Length (MDL) Principle [11,12] which states that the best way to capture regularities in data and avoid overfitting is to choose a model that has short description. Thus, a neural network which uses only one byte to represent a weight value is better than a network requiring 4-byte-long description of every weight, even if the latter has slightly higher accuracy on training data than the former.

A more intuitive justification might be given by considering a system whose accuracy (measured during training on a training set) abruptly decreases when some weight is slightly disturbed, e.g. by 0.01. High accuracy of such a system is probably accidental and will not recur on test data. A system which is insensitive to such small perturbations of weight values would be much more trustworthy.

3 The Algorithm

Let us denote by Ω, $\Omega \subset \mathbb{R}$, the set of *permitted values* of weights. Assume that Ω is discrete (and usually $0 \in \Omega$). There is a training algorithm $\mathcal{A}$ given, e.g. backpropagation, which searches through a family $\mathcal{F}$ of models to find the one that achieves (approximately) minimum error rate on training data. Let us denote by $\mathcal{F}_\Omega$ the family of models from $\mathcal{F}$ whose weights belong to Ω (so $\mathcal{F}_\Omega \subset \mathcal{F}$). The goal is to create a discretized variant of algorithm $\mathcal{A}$, which finds a model in $\mathcal{F}_\Omega$ that minimizes the error rate over $\mathcal{F}_\Omega$.

The easiest way to do this is to discretize weights of the model found by algorithm $\mathcal{A}$, by simply rounding them to the closest values from Ω. This procedure is very simple, but it does not provide any control over the accuracy of the discretized model, so it cannot be beneficial for accuracy of the final model.

A better method is to interlace discretization with training process $\mathcal{A}$ by rounding weights every time they are updated. In this case, the process of searching for the best model is restricted to $\mathcal{F}_\Omega$ from the beginning, so it is directed by accuracy of *discretized* models and thus is able to find a better model than the previous method.

However, there is still another problem if the process of searching guided by algorithm $\mathcal{A}$ moves slowly through the space $\mathcal{F}$, which happens for example in backpropagation algorithm when the *learning rate* [1] is small. In this case, simple discretization – meant as deterministic rounding of every weight to the nearest value in Ω – may turn values of all updated weights back to the values from before the update. Consequently, the searching process guided by discrete variant of $\mathcal{A}$ may easily get stuck in some point of $\mathcal{F}_\Omega$ which is neither global nor even local minimum of error function.

This can be avoided by performing discretization in a *nondeterministic* way.

Let v denote the weight value to be discretized; $L_\Omega(v)$ is the greatest value in Ω not greater than v; $G_\Omega(v)$ is the least value in Ω not less than v. Value v is discretized by replacing it nondeterministically with either $L_\Omega(v)$ or $G_\Omega(v)$, according to the formula:

$$D_\Omega(v) = \begin{cases} L_\Omega(v) & \text{with probability } (G_\Omega(v) - v)/R_\Omega(v) \\ G_\Omega(v) & \text{with probability } (v - L_\Omega(v))/R_\Omega(v) \end{cases} , \tag{1}$$

where $D_\Omega(v)$ denotes discretized value of v and $R_\Omega(v) = G_\Omega(v) - L_\Omega(v)$. The above choice of probabilities makes the following important property hold:

$$\mathbb{E}(D_\Omega(v)) = v , \tag{2}$$

i.e. expected value of discretized weight is equal to the original value. In this way, discretization may be viewed as adding some zero-mean random fluctuations to weight values, without disturbing global behavior of the original algorithm. Note, however, that discretization is *not* equivalent to adding random fluctuations to weight values. General structure of the training algorithm which performs discretization of weights is presented in Figure 1.

```
for cycle := 1 to number of training cycles do
    pattern := GetNextPattern();
    CalculateResponse(network, pattern);
    UpdateWeights(network);    /* standard algorithm, e.g. backpropagation */
    for each weight in network do
        w := ValueOf(weight);
        d := $D_\Omega(w)$;    /* nondeterministic discretization of w, Eq. (1) */
        ValueOf(weight) := d;
    end
end
```

Fig. 1. Outline of the neural network training algorithm with discretization of weights

Some attention should be paid to the question of what set of permitted values Ω to use. Simple yet efficient choice is to take a set of evenly-spaced numbers containing zero:

$$\Omega = \{k\gamma : k \in \mathbb{Z}\} \ , \tag{3}$$

where $\gamma \in \mathbb{R}$ is a parameter that controls *granularity* of discretization.

4 Experimental Results

The presented modification was applied to standard backpropagation algorithm [2,1] used for training of multilayer neural networks. The modified algorithm was compared with the standard one on two real-world datasets from the UCI [13] machine-learning repository: *Labor* and *Image Segmentation*. Experiments were conducted in WEKA [14] environment, whose implementation of backpropagation algorithm was extended by the author to handle discretization of weights.

To enable thorough analysis and reliable comparison of the algorithms, different numbers of hidden neurons were tested: 5, 10, 20, 30, 50, 70, 100, 150, 200 and 250. In this way, it was possible to draw final conclusions that were independent from specific choice of training parameters.

To obtain plausible results, 20 networks were trained for each algorithm and size of hidden layer, using different (random) split of data into training and test sets each time (percentage split: $50 + 50\%$ for *Labor* data; $25 + 75\%$ of training and test instances respectively for *Image Segmentation* data). Thus, 20 estimates of error rate on test set were obtained for each algorithm and size of hidden layer. Mean and standard deviation of these 20 values formed the basis of subsequent analysis.

Throughout all experiments, the learning rate [1] of neural networks (which controls magnitude of updates) was set to 0.1 and each network underwent 20 epochs of training. In discrete variant of the algorithm, all weights of networks – both in hidden and output layers – were discretized in the same way, with granularity $\gamma = 0.1$.

4.1 *Labor* Data

Labor data set[1] [13,15,16] contains information on final settlements in labor negotiations in Canadian industry. It is composed of 57 instances described by 16 attributes – mixed symbolic and numeric. In the experiment, symbolic attributes were turned into binary, which resulted in a data set described by 26 numeric or binary attributes. Then, all attributes were normalized. There were two classes with 37 and 20 instances respectively. Neural networks created during the experiment consisted of two layers of sigmoidal neurons: hidden one, of different size; and output one, containing two neurons, one for each class. Results of experiments are listed in Table 1 and presented graphically in Figure 2.

The results show that discretization substantially improves accuracy of neural networks trained on *Labor* data. The lowest error rate obtained with discretized

[1] ftp://ftp.ics.uci.edu/pub/machine-learning-databases/labor-negotiations .

Table 1. Error rates [%] and their standard deviations for neural networks trained with either standard or discrete backpropagation algorithm, evaluated on *Labor* data

No. of hidden neurons	Discrete backpropagation	Standard backpropagation
5	22.55 ± 9.83	35.10 ± 0.63
10	15.25 ± 7.93	35.10 ± 0.63
20	10.17 ± 5.42	34.06 ± 2.28
30	9.49 ± 4.99	28.08 ± 5.61
50	8.25 ± 5.37	26.00 ± 9.45
70	7.54 ± 4.98	22.62 ± 9.30
100	7.37 ± 3.94	18.26 ± 8.48
150	7.20 ± 3.33	21.39 ± 9.29
200	7.72 ± 4.37	21.25 ± 10.85
250	7.91 ± 4.11	21.96 ± 11.67

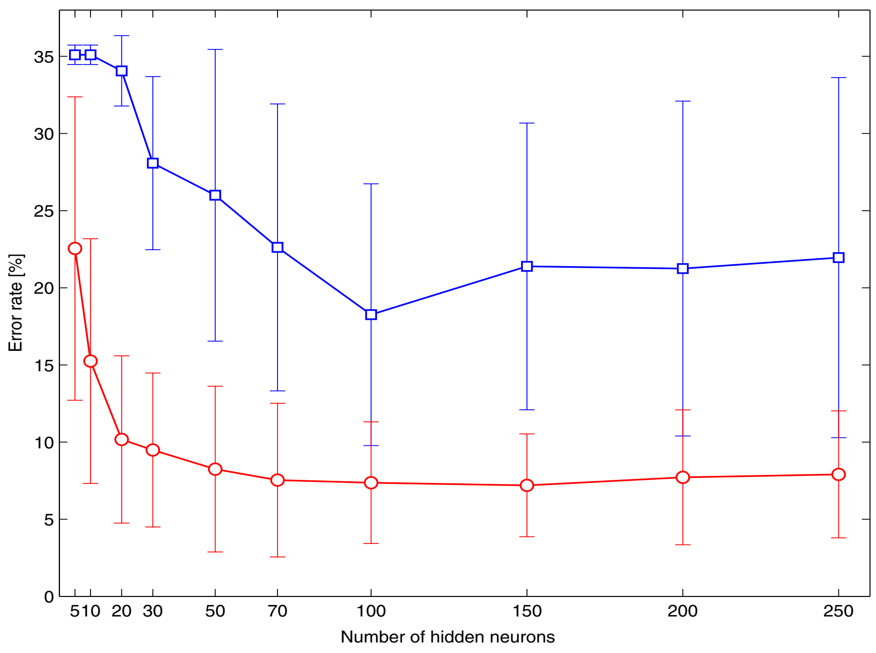

Fig. 2. Error rates [%] and their standard deviations (vertical bars) for neural networks trained with either standard (squares) or discrete (circles) backpropagation algorithm, evaluated on *Labor* data. Networks with different number of hidden neurons (horizontal axis) were checked.

algorithm (7.20%) is by 60% smaller than with standard algorithm (18.26%), which is a huge difference. Moreover, standard deviation of the error rate among networks with the same size of hidden layer is also significantly lower when

Table 2. Error rates [%] and their standard deviations for neural networks trained with either standard or discrete backpropagation algorithm, evaluated on *Image Segmentation* data

No. of hidden neurons	Discrete backpropagation	Standard backpropagation
5	24.41 ± 3.84	37.72 ± 5.94
10	15.18 ± 3.02	27.88 ± 3.94
20	12.12 ± 1.43	19.94 ± 2.26
30	11.15 ± 1.12	19.61 ± 2.10
50	10.91 ± 1.25	17.45 ± 1.50
70	10.34 ± 0.97	16.59 ± 1.21
100	10.24 ± 0.99	16.24 ± 1.29
150	10.87 ± 1.40	16.24 ± 1.68
200	10.77 ± 1.75	16.76 ± 2.11
250	10.63 ± 1.90	17.04 ± 2.41

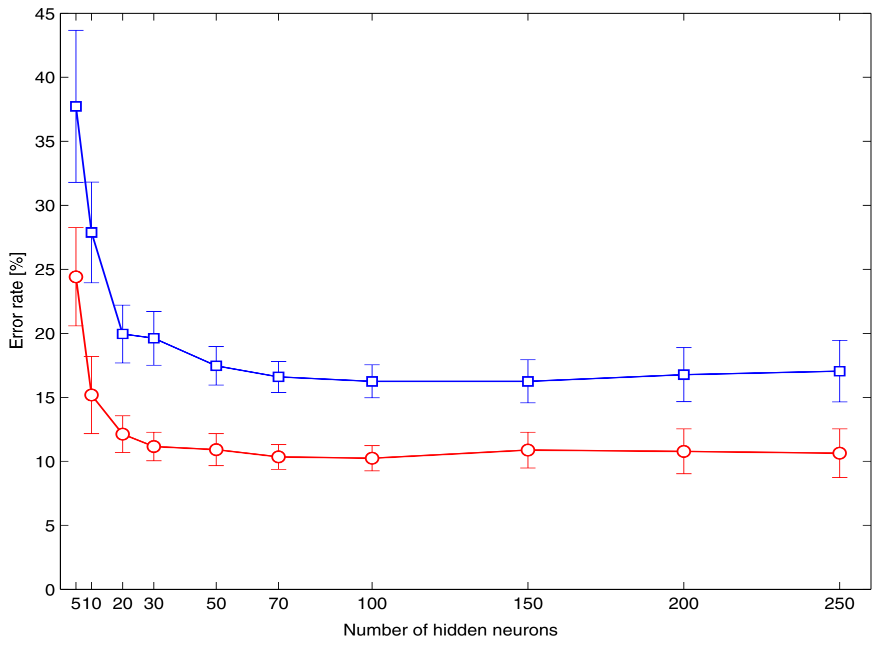

Fig. 3. Error rates [%] and their standard deviations (vertical bars) for neural networks trained with either standard (squares) or discrete (circles) backpropagation algorithm, evaluated on *Image Segmentation* data. Networks with different number of hidden neurons (horizontal axis) were checked.

discretization is used. These large differences indicate that standard backpropagation highly overtrains on *Labor* data and discretization of weights is an efficient way to reduce this overtraining.

4.2 *Image Segmentation* Data

Image Segmentation data set[2] [13,3] contains 2310 instances, described by 19 numeric attributes and uniformly distributed in 7 classes. The attributes were normalized before training. Neural networks created during the experiment consisted two layers of sigmoidal neurons: hidden one, of different size; and output one, containing 7 neurons, one for each class. Results of experiments are listed in Table 2 and presented graphically in Figure 3.

As in the case of *Labor* data, also for *Image Segmentation* data the performance of neural networks can be improved by discretization of weights. The best result achieved with discretization (10.24% error rate) is by 37% better than the best result of standard algorithm (16.24%). The improvement is smaller than for *Labor* data, probably due to significantly bigger size of the training set and thus smaller degree of overfitting.

5 Conclusions

A novel method that reduces overfitting of neural networks has been presented. The method is based on non-deterministic discretization of weights after every training cycle, which restricts the set of possible weight values to a discrete subset of real numbers and enables much shorter description of the network. This, in turn, improves generalization abilities and performance of the system, according to Minimum Description Length principle. Thanks to non-determinism, there is no risk that discretization would force the training process to stop in some far-from-optimal point of parameter space. The method was evaluated on two real-world data sets from UCI repository, exhibiting high effectiveness in preventing neural network from overfitting: the use of discretization enabled decrease in error rate by 60% and 37% respectively. Importantly, it was better to use discretization than to decrease the number of hidden neurons, so discretization appeared to be more effective than the most straightforward method of avoiding overtraining.

It should be emphasized that the presented method is potentially very versatile. Although the paper covers only neural networks and backpropagation algorithm, discretization of parameters of a model could be applied to many other algorithms and systems, as different as evolutionary algorithms, Bayesian networks or Gaussian mixture models, to name just a few.

There are also two more general conclusions which follow from the presented study. They may seem strange at first sight but have deep consequences. Firstly, it appears that methods of continuous optimization – like gradient descend, which lies in the basis of backpropagation algorithm – can be successfully applied to optimization over *dis*continuous (e.g. discrete) spaces, as well.

Secondly, all decision systems built from data and described by real-valued parameters might probably benefit from some kind of restriction imposed on possible parameter values.

These conclusions definitely need more investigation.

[2] ftp://ftp.ics.uci.edu/pub/machine-learning-databases/image

Acknowledgement

The author thanks anonymous reviewers for their helpful remarks.

References

1. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
2. Ripley, B.D.: Pattern recognition and neural networks. Cambridge University Press, Cambridge (1996)
3. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine Learning, Neural and Statistical Classification. Elis Horwood, London (1994)
4. Rychetsky, M., Ortmann, S., Glesner, M.: Pruning and regularization techniques for feed forward nets applied on a real world data base. In: Heiss, M. (ed.) International Symposium on Neural Computation, pp. 603–609 (1998)
5. Bishop, C.M.: Training with noise is equivalent to Tikhonov regularization. Neural Computation 7(1), 108–116 (1995)
6. Burger, M., Neubauer, A.: Analysis of tikhonov regularization for function approximation by neural networks. Neural Networks 16(1), 79–90 (2003)
7. Wojnarski, M.: LTF-C: Architecture, training algorithm and applications of new neural classifier. Fundamenta Informaticae 54(1), 89–105 (2003)
8. Sietsma, J., Dow, R.J.F.: Creating artifical neural networks that generalize. Neural Networks 4(1), 67–79 (1991)
9. Holmström, L., Koistinen, P.: Using additive noise in back-propagation training. IEEE Transactions on Neural Networks 3(1), 24–38 (1992)
10. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
11. Rissanen, J.: Modeling by shortest data description. Automatica 14, 465–471 (1978)
12. Grünwald, P., Myung, I.J., Pitt, M.: Advances in Minimum Description Length. MIT Press, Cambridge (2005)
13. Newman, D.J., Hettich, S., Merz, C.B.: UCI repository of machine learning databases (1998)
14. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
15. Bergadano, F., Matwin, S., Michalski, R.S., Zhang, J.: Measuring quality of concept descriptions. In: EWSL, pp. 1–14 (1988)
16. Bergadano, F., Matwin, S., Michalski, R.S., Zhang, J.: Representing and acquiring imprecise and context-dependent concepts in knowledge-based systems. In: ISMIS, pp. 270–280 (1988)