WDS'08 Proceedings of Contributed Papers, Part I, 159–164, 2008. ISBN 978-80-7378-065-4 © MATFYZPRESS

Feedforward Neural Networks – Architecture Optimization and Knowledge Extraction

Z. Reitermanová

Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic.

Abstract. Feedforward neural networks represent a well-established computational model, which can be used for solving complex tasks requiring large data sets. When dealing with this kind of problems, the main requirements will be the speed of the learning process and the ability to generalize well the extracted knowledge. To satisfy these demands, adequate initial parameters of the model – like number of layers and number of neurons – are essential. For a given problem, especially the architecture of the model impacts its generalization capabilities.

Optimal network architecture speeds up recall and may also improve efficiency of further retraining. Some of the techniques also enable or simplify further knowledge extraction. The main goal of this article is to provide a survey of some existing techniques that optimize architecture of BP-networks.

Introduction

Artificial neural networks represent a well-established computational model, which can be used for solving complex tasks requiring large data sets. It is widely used in areas like data mining, web mining, bioinformatics or multimedia data processing. When solving these kinds of problems, we will concentrate especially on the speed of the learning and recall process and on the ability to generalize well the extracted knowledge. To satisfy these demands, adequate initial parameters of the model (e.g. learning rates, number of neurons and number of layers) are essential. For a given problem, especially the architecture of the model affects its generalization capabilities. Finding the optimal model consistent with the considered data thus constitutes one of the principle tasks of machine learning.

Nevertheless, searching for an optimal architecture has more objectives than just an improved generalization and computational efficiency. It can also help create a transparent structure of the network. The main criticism of standard feedforward neural networks concerns the inability to explain easily the knowledge they have extracted. Therefore, some of the techniques used to optimize the architecture of BP-networks are not focused on finding the minimum number of neurons and weights but rather on forming a clear and transparent network structure. This simplifies the following knowledge extraction.

BP-network model

To keep the paper self-contained, we will first describe briefly also the model of fully connected feed-forward neural networks (BP-networks) and the well-known back-propagation algorithm (BP). We adopted the following notation mainly from [Mrázová and Reitermanová, 2007]. BP-network is a model that consists of computational units, called neurons, which are arranged into multiple layers. Each neuron in one layer is connected to all neurons of the subsequent layer. A neuron with the weights $(w_1, \ldots, w_n)$, the threshold ϑ and the input vector $\vec{z} = (z_1, \ldots, z_n)$, computes its potential value ξ as $\xi = \sum_{i=1}^{n} z_i\, w_i + \vartheta$. The thresholds can be represented by weights coming from fictive neurons with a constant output value equal 1. Then, $\xi = \sum_{i=0}^{n} z_i\, w_i$, where $w_0 = \vartheta$ and $z_0 = 1$. To compute the output of a neuron, we will consider the sigmoidal transfer function $y = f(\xi) = \frac{1}{1+e^{-\lambda\xi}}$ having the derivative equal to: $f'(\xi) = \lambda\, y\,(1 - y)$, where λ is a positive constant representing the slope coefficient.

BP-training algorithm is a gradient method, which learns from examples. The training set T is a finite non-empty set of P ordered pairs of input/output patterns: $T = \{[\vec{x}_1, \vec{d}_1], \ldots, [\vec{x}_P, \vec{d}_P]\}$. The aim of the standard BP-training algorithm [Rumelhart et al., 1986] is to find a set of weights that ensure that for each input pattern the actual output produced by the network is the same as (or sufficiently close to) the output pattern. The desired behavior can be evaluated by the error function E:

$$E = \frac{1}{2} \sum_{p} \sum_{j} (y_{j,p} - d_{j,p})^2 \qquad (1)$$

where p is an index over all training patterns, j is an index over all output neurons, y is their actual and

d is their desired output value. Omitting the index p for the desired and actual neuron output values d and y, the weights of the network are adjusted iteratively after presenting each respective training pattern by:

$$w_{ij}(t+1) \;=\; w_{ij}(t) \;+\; \alpha\, \delta_j\, y_i. \tag{2}$$

The terms for δ_j correspond to:

$$\delta_j \;=\; \begin{cases} \lambda\, y_j\ (1 \,-\, y_j\,)(d_j \,-\, y_j) & \text{for an output neuron} \\ \lambda\, y_j\ \ (1 \,-\, y_j\,)\ \sum_k \delta_k\, w_{jk} & \text{for a hidden neuron,} \end{cases} \tag{3}$$

w stands for weights and thresholds, respectively, i and k index neurons in the layers below and above the neuron j, respectively. d_j is the desired and y_j the actual output value of the neuron j. $t+1$, t and $t-1$ index next, present and previous weights, respectively, α is positive constant representing the learning rate.

Main drawbacks of BP-algorithm are low speed of the training process and high sensitivity to the choice of initial parameters. For a given task, performance of the model and its generalization capability depends highly on the network architecture – on the number of layers and neurons. Architecture of the network should correspond to the complexity of the analyzed data. When the network is too small, it is not able to learn the task properly. On the contrary, when it is too large, it has tendency to overtrain – it memorizes the training patterns and it is not able to recognize patterns outside the training set. Therefore, when training a larger network, we also need more training patterns to avoid overtraining.

Another big disadvantage of standard BP-networks is that the model doesn't explain how it reaches its decisions, that means, how it transforms inputs into the outputs. Contrary to traditional statistical models, it behaves more like a black-box. It is not clear, what is the relation between the training data and the weights and activities of hidden neurons. Extracting knowledge, especially rules, from a standard BP-network is therefore highly difficult.

Methods for architecture optimization try to solve the described problems. There are several reasons for searching optimal (i.e. minimal) structure of BP-networks. The main goal is to improve and speed up prediction and to achieve better generalization. We appreciate the effectiveness of computation especially when working with large data sets. Other objectives are to decrease sensitivity to noisy data and to detect and manage the overtraining problem. Smaller networks also need less training patterns.

Moreover, some of the techniques for architecture optimization are able to identify significant input parameters and relevant hidden neurons or weights. When we know, how the inputs impact the outputs and which inputs are more significant than others are, we can understand better the internal structure of a BP-network and we can easier explain the underlying process. Extracting knowledge from a smaller network with clear and simple structure is also much easier.

There are many different approaches for architecture optimization, which can be divided into categories based on their principles and goals:

- Brute-force.
- Pruning algorithms.
- Regularization techniques.
- Network construction techniques.
- Probability optimization techniques.
- Sensitivity analysis pruning algorithms.

Some of the methods just incrementally increase the number of neurons (e.g. network construction techniques), other techniques only decrease the network size (e.g. pruning algorithms), some adaptive methods enable both types of modifications (e.g. probability optimization techniques). In the following paragraphs, we will describe each approach in more detail.

Brute-force methods

Brute-force is the most common approach, how to find the optimal network size [Reed, 1993]. It is based on successive training of smaller networks, until the smallest topology is found, which still fits the data. This process is very time-consuming – many networks have to be trained. We also have to consider the remaining initial parameters, because the behavior of the trained network is very sensitive to them.

There are also other problems connected with training of networks with architecture, which is for a given task nearly optimal [Hagiwara, 1994]. For such a network, it is difficult to converge and to learn the correct function. The smaller the network is, the more often it falls into local minima during training. The training process is also usually much slower for networks with minimal architecture than for larger ones.

Pruning algorithms

Principle of the pruning algorithms [Reed, 1993] is to train a larger network than necessary and then remove redundant parts of the final network until a reasonable architecture is achieved. The usual

algorithm 1 Principle of pruning

```
1. Train a BP-network with a reasonable architecture.
2. While the network error is higher than a given threshold or while it decreases:
    (a) Compute the relevance of hidden neurons or weights using the chosen heuristic.
    (b) Remove the least relevant element.
    (c) Retrain the network.
```

training and pruning process can be described in few steps, as shown in Algorithm 1.

Particular pruning algorithms differ in the way, how they solve the following important tasks: First question is how to evaluate the significance or relevance of single network elements (i.e. weights or neurons). Second question is how to successively detect and remove the unimportant elements based on the chosen relevance measure. Third difficult question is when to stop pruning. Another important question is which elements of the network to remove. Most of the pruning techniques target the pruning of edges (weights) or hidden neurons. More difficult task is to remove also the irrelevant input neurons. Pruning of network inputs corresponds to identification of the input variables, which are not needed for training. By their elimination we can obtain a network, which will generalize better [Zurada et al., 1994].

Particular pruning methods use different strategies for measuring relevance. One of most common relevance measures for weights is called weight saliency [Mozer and Smolensky, 1989; LeCun et al., 1990]. Saliency is the sensitivity of error function to removal or change of a single weight. Actually, removal of a weight correspond to setting the weight equal zero. The sensitivity of weight w_i is formally defined as: $S(w_i) = E(w_i = 0) - E(w_i = w_i^f)$ where w_i^f is the final value of weigh w_i after training. Weight saliency is a very sophisticated and precise relevance measure, which is however highly inefficient [Mozer and Smolensky, 1989; Karnin, 1990].

Several pruning methods approximate weight saliency in order to improve computational effectivity [Mozer and Smolensky, 1989], [Karnin, 1990], optimal brain damage (OBD) [LeCun et al., 1990] and optimal brain surgeon (OBS) [Hassibi et al., 1993]. In the case of OBD, the saliency is approximated by the second derivative of the error function w.r.t. the weight, where OBD assumes that the error function is quadratic and that the Hessian is diagonal. OBS method results from the OBD, but differ in the way, in which the saliency is approximated. It iteratively computes the full Hessian, which leads to a more exact approximation of the error function. Main weakness of the OBD and OBS techniques is their relatively low computational efficiency. Nevertheless, many modifications and approximations have been introduced – e.g. in [Attik et al., 2005], which try to deal with this problem.

The previous methods targeted pruning of weights. Hagiwara [1994] suggests three simple and effective strategies for detecting both redundant hidden neurons and weights (called Goodness factor, Consuming energy and Weights power). These measures are much less sophisticated and precise than OBD or OBS, but require significantly less computational time. Goodness factor is defined as: $G_i = \sum_p \sum_j (y_i w_{ij})^2$ where p is an index over all training patterns, i is an index over all neurons in a fixed hidden layer, y_i is their output, j is an index over all neurons in the next layer, and w_{ij} is weight from the i-th to the j-th neuron. Goodness factor favors neurons with high weights and high activities for most patterns. Consuming energy is defined as follows: $E_i = \sum_p \sum_j y_i w_{ij} y_j$ where y_j is the output of the j-th neuron in the next layer. Consuming energy prefers neurons, which excites frequently and at the same time as neurons in the next layer. Weight power subsumes only weights and is defined as follows: $W_i = \sum_j (w_{ij})^2$. Hagiwara [1994] has shown on the mirror symmetry problem, that all the three strategies massively reduce the network size and lead to better generalization, while the simplest strategy, Weight power, reaches best results. However, methods based on weight magnitude often remove also important parts of the network [Hassibi et al., 1993].

Regularization techniques

Regularization techniques try to eliminate redundant weights already during training. A penalty term added to the error function usually enforces a decrease of absolute weight values during training. Weights smaller than a given threshold are regarded as useless and removed. The best-known representative of this approach is called weight decay [Werbos, 1988]. Penalty term added to error function has the form $E' = \beta \sum_i w_i^2$, where β is a positive constant. Other variants of the penalty terms – e.g. [Garson, 1991; Tsaih, 1999] – yield varying results. Their main weakness consists in their relatively high computational costs and worse generalization [Hanson and Pratt, 1989].

Network construction techniques

Network construction techniques start with a small network and incrementally add hidden neurons and weights until a reasonable architecture is achieved. Similar approach is to split existing neurons if the performance of the network is not sufficient [Lee, 1991]. Network construction methods are usually not based on training from examples, as they require different training paradigms, such as genetic algorithms. Therefore, they are hardly comparable to the standard pruning or regularization techniques.

Best-known example of this approach is cascade correlation [Fahlman and Lebiere, 1990]. This method has several advantages: the training process is very quick, the network determines size and topology without outside interventions, and the algorithm is resistant to changes of training set during training. Key problems, crucial for the success of cascade correlation and other network construction techniques, are to decide, when to stop adding new neurons and when and where add a new neuron or edge. Wrong treatment of these questions can lead to overtraining and strong slow-down of the construction process.

Probability optimization techniques

Probability optimization techniques dynamically alter the neural network topology already during training. They are usually based on genetic algorithms or simulated annealing. Particular methods differ in the way, in which they code the solution. Method introduced by Whitley et al. [1990] trains a fully connected network, then prunes weights using genetic operators and finally retrains the network. Pruned networks compete for survival and are awarded for fewer parameters and better generalization. Advantage of these techniques is in the fact, that they usually lead to good generalization, even if the training set is relatively small [Hancock, 1992]. Main disadvantage is that the pruning process is very time-consuming.

Sensitivity analysis

The main objective of sensitivity analysis is to identify important input patterns and their features. Previously discussed pruning methods (OBD, OBS) determined in principle the sensitivity of the applied error function to single weights. In sensitivity analysis, however, we will evaluate the impact of network inputs on its output. Tchaban et al. [1998] suggested a simple yet less precise sensitivity measure called weight product. For a neuron i and a neuron j from the immediately following layer that are connected by the weight w_{ij}, the weight product S_{ij} is defined by: $S_{ij} = \frac{x_i w_{ij}}{y_j}$, where x_i denotes the i-th input element of the neuron j and y_j corresponds to its output value. Empirical studies [Wang et al., 2000; Garson, 1991] have, however, shown that the approach of Tchaban et al. [1998] is ineffective when examining the influence of the inputs on outputs. A different definition of sensitivity in BP-networks has been suggested by Zurada et al. [1994], namely $S_{ij} = \frac{\partial y_j}{\partial x_i} = f'(\xi_j)\, w_{ij}$ for the sensitivity of a single neuron j on its input i and $S_{ij} = \frac{\partial y_j}{\partial y_i} = \sum_k S_{kj} f'(\xi_k) w_{ik}$ for the sensitivity of the output neuron j on the input i with the hidden neurons indexed by k.

Several variants of sensitivity analysis use then the computed sensitivity coefficient for pruning. In this respect, the key questions refer to the detection of redundant neurons and to the stop criteria for pruning. Engelbrecht et al. [1995], Zurada et al. [1994], and Engelbrecht and Cloete [1996] use heuristics, which remove inputs and hidden neurons with the smallest sensitivity, Engelbrecht [2001] suggests to remove neurons with minimum variance of sensitivity. Unfortunately, the sensitivity criteria alone are not capable of detecting all redundant neurons, because they assume that inputs and activities of hidden neurons are mutually independent and numerical. Montaño and Palmer [2003] presents an extension called Numeric sensitivity analysis, which handles also quantitative or discrete input parameters.

Other techniques

Besides architecture optimization techniques, other methods can be used to improve the generalization, create simpler and smoother network function and thus enable simpler knowledge extraction. Some of them can be easily combined with some of the techniques for architecture optimization to achieve better results.

One of such techniques is learning from hints. This method was proposed originally by Abu-Mostafa [1993a] and it is very powerful in incorporating prior knowledge as a learning aid into the learning-from-examples paradigm. A hint is any piece of information about the unknown function. It may take the form of naturally or even artificially generated examples or the form of a global constraint of the unknown function. A hint may reduce the number of functions that are candidates to be the wanted function. Abu-Mostafa [1993b] has shown that restricting the space of candidate hypotheses for the wanted network function by learning from hints can reduce the VC-dimension of the final network structures.

Yu and Simmons [1990], Gallmo and Carlstrom [1995], and Suddarth and Kergosien [1990] proposed the extra output hint method. It consists in supplying the network with additional target outputs during training which express some knowledge about the problem. Unlike for most other hint methods, no modification of the network algorithm or error criterion is involved in this case and the extra-added outputs can be removed after training is finished. Mrázová and Reitermanová [2007] successfully combine this method with other techniques that enforce transparent network structure to achieve better generalization and more transparent internal network structure.

Techniques, which enforce smooth network function and transparent network architecture, e.g. [Mrázová and Wang, 2007], [Mrázová and Reitermanová, 2007], also significantly support better generalization capability and subsequent pruning of redundant edges and neurons. Obviously, neurons providing always the same internal representation can be more easily recognized and subsequently also removed with minor changes of the network – e.g. using the approach described by Sietsma and Dow [1991]. The same holds for neurons yielding for all the patterns identical or complementary internal representation to another hidden neuron.

Analysis of the discussed methods

The above-discussed approaches to architecture optimization differ in many ways – they employ different principles, characteristics, and even goals. Some of the methods just try to decrease network size and speed up the recall – e.g. regularization techniques. Other techniques are capable of improving generalization and robustness to noisy data – e.g. pruning algorithms and probability optimization techniques. More sophisticated strategies – e.g. sensitivity analysis – rather identify significant network elements and facilitate knowledge extraction.

Each of the described methods has its advantages but also limitations. No one can be denoted as the best in any situation and for all reasons. For example, regularization techniques can be combined only with gradient-based training algorithms, such as back-propagation. On the other hand, network construction techniques usually require different training paradigms, such as genetic algorithms. Some of the pruning algorithms (OBS, weight magnitude, and sensitivity analysis) can be used with both types of learning algorithms.

A significant drawback of most standard methods consists in their low efficiency. Real-world applications therefore prefer simpler and more efficient methods, such as weight magnitude. Even though more sophisticated methods reach better results, precision is usually compensated by unproportional increase in computation time. Best results are thus often obtained by a combination of several applicable methods.

Conclusions and further research

In this article, we have discussed the main approaches to the optimization of BP-networks – namely pruning algorithms, regularization techniques and network construction techniques. In particular, the methods based on sensitivity analysis are capable of improving the network generalization and simplify the following knowledge extraction from the network. Unfortunately, sensitivity analysis is not guaranteed to detect all redundant network elements as it assumes both the inputs of the network and the outputs of its hidden neurons to be mutually independent.

For this reason, we plan to focus our further research on a combination of sensitivity analysis with other methods leading to an improved generalization and simplified knowledge extraction. We expect that especially learning with hints together with an enforced condensed internal representation [Mrázová and

Reitermanová, 2007] and pruning based on the formed internal representation will improve generalization and enable an easy extraction of transparent rules from the final network.

Acknowledgments. The author would like to thank her advisor Doc. RNDr. Iveta Mrázová, CSc. for her useful advices and overall help. The present work was supported by the Czech Science Foundation under the contract no. 201/05/H014 and by the Charles University Grant Agency under the contract no. 8176/2008 (Grant no. 258010).

References

Abu-Mostafa, Y. S., A method for learning from hints, in *Advances in NIPS*, vol. 5, pp. 73–80, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993a.

Abu-Mostafa, Y. S., Hints and the VC dimension, *Neural Computing*, *5*, 278–288, 1993b.

Attik, M., Bougrain, L., and Alexandre, F., Neural network topology optimization, *ICANN'05*, *3697*, 53–58, 2005.

Engelbrecht, A. and Cloete, I., A sensitivity analysis algorithm for pruning feedforward neural networks, *IEEE ICNN'96*, *2*, 1274–1277, 1996.

Engelbrecht, A. P., A new pruning heuristic based on variance analysis of sensitivity information, *IEEE Transactions on Neural Networks*, *12*, 1386–1399, 2001.

Engelbrecht, A. P., Cloete, I., and Zurada, J. M., Determining the significance of input parameters using sensitivity analysis, in *IWANN*, pp. 382–388, 1995.

Fahlman, S. E. and Lebiere, C., The cascade-correlation learning architecture, in *Advances in NIPS*, edited by D. S. Touretzky, vol. 2, pp. 524–532, Morgan Kaufmann, San Mateo, 1990.

Gallmo, O. and Carlstrom, J., Some experiments using extra output learning to hint multi layer perceptrons, in *SCC'95*, edited by L. Niklasson and M. Boden, pp. 179–190, 1995.

Garson, G. D., Interpreting neural-network connection weights, *AI Expert*, *6*, 46–51, 1991.

Hagiwara, M., A simple and effective method for removal of hidden units and weights, *Neurocomputing*, *6*, 207–218, 1994.

Hancock, P. J. B., Pruning neural nets by genetic algorithm, in *ICANN*, edited by I. Aleksander and J. Taylor, pp. 991–994, Elsevier, Brighton, 1992.

Hanson, S. J. and Pratt, L., Comparing biases for minimal network construction with back-propagation, *Advances in NIPS*, *1*, 177–185, 1989.

Hassibi, B., Stork, D. G., and Wolf, G. J., Optimal brain surgeon and general network pruning, *IEEE ICNN'93*, *1*, 293–299, 1993.

Karnin, E. D., A simple procedure for pruning back-propagation trained neural networks, *IEEE ICNN'90*, *1*, 239–242, 1990.

LeCun, Y., Denker, J., Solla, S., Howard, R. E., and Jackel, L. D., Optimal brain damage, in *Advances in NIPS*, edited by D. S. Touretzky, vol. 2, Morgan Kauffman, San Mateo, CA, 1990.

Lee, T.-C., *Structure Level Adaptation for Artificial Neural Networks*, Kluwer Academic Publishers, Norwell, MA, USA, 1991.

Montaño, J. J. and Palmer, A., Numeric sensitivity analysis applied to feedforward neural networks, *Neural Computing and Applications*, *12*, 119–125, 2003.

Mozer, M. C. and Smolensky, P., Skeletonization: A technique for trimming the fat from a network via relevance assessment, *Advances in NIPS*, *1*, 107–115, 1989.

Mrázová, I. and Reitermanová, Z., Enforced knowledge extraction with BP-networks, in *Intelligent Engineering Systems through Artificial Neural Network*, vol. 17, pp. 285–290, ASME Press, New York, USA, 2007.

Mrázová, I. and Wang, D., Improved generalization of neural classifiers with enforced internal representation, *Neurocomputing*, *70*, 2940–2952, 2007.

Reed, R., Pruning algorithms - a survey, *IEEE Trans. on Neural Networks*, *4*, 740–747, 1993.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Learning internal representations by error propagation, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, *1*, 318–362, 1986.

Sietsma, J. and Dow, R. J. F., Creating artificial neural networks that generalize, *Neural Networks*, *4*, 67–79, 1991.

Suddarth, S. C. and Kergosien, Y. L., Rule-injection hints as a means of improving network performance and learning time, in *EURASIP Workshop 1990 on Neur. Netw.*, vol. 412, pp. 120–129, Springer London, 1990.

Tchaban, T., Taylor, M. J., and Griffin, J. P., Establishing impacts of the inputs in a feedforward neural network, *Neural Computing and Applications*, *7*, 309–317, 1998.

Tsaih, R., Sensitivity analysis, neural networks, and the finance, in *IJCNN'99*, vol. 6, pp. 3830–3835, 1999.

Wang, W., Jones, P., and Partridge, D., Assessing the impact of input features in a feedforward neural network, *Neural Computing and Applications*, *9*, 101–112, 2000.

Werbos, P. J., Back-propagation: Past and future, in *IEEE ICNN'88*, vol. 1, pp. 343–353, IEEE Press, New York, USA, 1988.

Whitley, D., Starkweather, T., and Bogart, C., Genetic algorithms and neural networks: Optimizing connections and connectivity, in *Parallel Computing*, vol. 14, pp. 347–361, 1990.

Yu, Y.-H. and Simmons, R. F., Extra output biased learning, *IJCNN'90*, *3*, 161–166, 1990.

Zurada, J. M., Malinowski, A., and Cloete, I., Sensitivity analysis for minimization of input data dimension for feedforward neural network, in *ISCAS'94*, vol. 6, pp. 447–450, 1994.