Fast Learning for Problem Classes using a Knowledge Based Network Initialization

Michael Hüsken, Christian Goerick

2000

Preprint:

This is an accepted article published in IJCNN2000, Proceedings of the International Joint Conference on Artificial Neural Networks. The final authenticated version is available online at: https://doi.org/[DOI not available]

Published in:
Proceedings of the International Joint Conference on Neural Networks (IJCNN 2000),
volume VI, pages 619-624, 2000

Fast learning for problem classes using knowledge based network initialization

Michael Hüsken and Christian Goerick*

Institut für Neuroinformatik
Ruhr-Universität Bochum
44780 Bochum, Germany
Phone: ++49 (0)234 / 32-25558, Fax: ++49 (0)234 / 32-14209
{huesken,goerick}@neuroinformatik.ruhr-uni-bochum.de

Abstract

The success of learning as well as the learning speed of an artificial neural network (ANN) strongly depends on the initial weights. If problem or domain specific knowledge exists, it can be transferred to the ANN by means of a special choice of the initial weights. In this paper, we focus on the choice of a set of initial weights, well suited to fast and robust learning of all particular problems out of a class of related problems. Our evolutionary approach particularly takes the learning algorithm into consideration in the design of the initial weights. The superior properties of the initial weights resulting from this algorithm are corroborated using a class defined by solving a differential equation with variable boundary conditions.

Keywords: problem class, neural network, adaptability, evolutionary algorithm, differential equation, weight initialization

1 Introduction

Initializing the weights is one important aspect in designing artificial neural networks (ANNs). There are two established ways for the initialization, differing in the use of problem dependent knowledge. Without such knowledge, initial weights are chosen small and at random in order to introduce as small a bias as possible and to allow for symmetry breaking between the neurons (LeCun, Bottou, Orr, & Müller 1998). When problem or domain specific knowledge is to be used for the initialization, some kind of information is transferred into the ANN prior to learning. See Reed & Marks (1999) for an overview of these methods. They all can be characterized as follows: They either focus on the acceleration of learning the solution of one particular problem or they try to improve the generalization of the resulting network.

We will present a different approach in this paper. Driven by an application where the efficient multiple solution of several related problems must be achieved, we propose a method for the initialization that pursues different goals: It should be well suited to accelerate the learning of a whole class of problems and therefore in accordance with the learning method that is being used. Both goals have not been addressed in the literature so far.

We will develop our method starting from results being presented in Hüsken, Goerick, & Vogel (2000), where it was shown that using the result of a previous training for a closely related task can be a good starting point for a new task. The new method will be applied to a class of problems: the solution of several differential equations by means of ANNs.

The remaining paper is organized as follows. In section 2 we will summarize the method to solve DEs by means of ANNs and introduce the problem class, we perform our experiments with. Our evolutionary approach is introduced in section 3 and the experiments as well as the results will be presented in the following section. The paper will close with some conclusions and an outlook.

* Current address: HONDA R&D Europe (Deutschland) GmbH, Carl-Legien-Str. 30, 63073 Offenbach, Germany

2 Problem class: Solving differential equations using ANNs

Solving ordinary and partial differential equations can be learned to a good degree by an ANN. The basic ideas as introduced by Lagaris, Likas, & Fotiadis (1998) are summarized in the first paragraphs of this section. Let the DE to be solved be given by

$$G\left(\boldsymbol{x}, \psi(\boldsymbol{x}), \nabla\psi(\boldsymbol{x}), \nabla^2\psi(\boldsymbol{x}), \ldots\right) = 0 \quad \boldsymbol{x}\epsilon\tilde{D} \subseteq \mathbb{R}^n \quad , \tag{1}$$

where $\psi(\boldsymbol{x})$ denotes the solution, G the functional defining the structure of the DE, and ∇ some differential operator. The basic idea, called collocation method, is to discretise the domain $\tilde{D}$ over a finite set of points D. Thus (1) becomes a system of equations. An approximation of the solution $\psi(\boldsymbol{x})$ is given by the trail solution $\psi_t(\boldsymbol{x})$. As a measure for the degree of fulfillment of the original DE (1) an error function similar to the mean squared error is defined:

$$E = \frac{1}{|D|}\sum_{\boldsymbol{x}_i\epsilon D}\left[G\left(\boldsymbol{x}_i, \psi_t, \nabla\psi_t, \nabla^2\psi_t, \ldots\right)\right]^2 \quad . \tag{2}$$

Therefore, finding an approximation of the solution of (1) is equal to finding a function which minimizes the error E.

Due to multilayer feedforward neural networks being universal approximators (Hornik, Stinchcombe, & White 1989), the trail solution $\psi_t(\boldsymbol{x})$ can be represented by such an ANN. In case of a given network architecture (number of neurons, layers, connections, kind of activation function) the problem is reduced to finding a configuration of weights that minimizes (2). As E is differentiable with respect to the weights for most DEs, efficient gradient-based learning algorithms for ANNs can be employed for minimizing (2). Of course, it is possible to apply other optimization methods without the need for calculating the gradient of E, like evolutionary algorithms.

Apart from equations (1) or (2), the solution of a DE has to meet additional constraints like initial and boundary conditions. In principle, there exist different ways to find solutions under these constraints. One way, proposed by Lagaris, Likas, & Fotiadis (1998), is to use an *ansatz*, satisfying the constraints by definition, thus yielding an unconstrained optimization task.

Unfortunately, this is only applicable to problems with simply shaped boundaries and the ansatz has to be constructed problem dependent. In case of more complex problems, e. g. real world optimization tasks, the shape is usually more complicated and mostly given by a set of points S. To overcome these problems, a more time-consuming algorithm based on the combination of a multilayer feedforward ANN and a radial basisfunction network (RBFN) is proposed (Lagaris, Likas, & Papageorgiou 1998). The parameters of the RBFN are chosen in such a way that the boundary conditions are exactly met.

As this approach is very time-consuming and dependents on many additional parameters and the scope of this paper is not to study constraint optimization but to investigate the consequences of different kinds of weight initializations, we apply the straightforward method of using a penalty term to enforce the boundary conditions:

$$E_{\mathrm{bc}} = \frac{1}{|S|}\sum_{\boldsymbol{x}_i\epsilon S}\left[\psi_t(\boldsymbol{x}_i) - \psi_{\mathrm{bc}}(\boldsymbol{x}_i)\right]^2 \quad . \tag{3}$$

The function $\psi_{\mathrm{bc}}(\boldsymbol{x})$ represents the boundary condition, here evaluated at a set of points S. The same approach applies to constraints like fixed values of the derivatives of $\psi(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ on the boundary.

Now, one is left with the minimization of the weighted sum of (2) and (3):

$$E' = E + \eta E_{\mathrm{bc}} \quad . \tag{4}$$

The weight parameter η can either be chosen as a fixed value or be determined by cross validation (Bishop 1995). The experiments in this paper are performed using $\eta = 10$; additional experiments have shown that the results are insensitive to this choice.

As in Hüsken, Goerick, & Vogel (2000) we utilize the one-dimensional second order DE

$$\psi''(x) = -\omega^2\psi(x) \quad , \tag{5}$$

which describes the harmonic oscillator. The real constant ω can be interpreted as the natural frequency of the system. In particular, we deal with the boundary value problem, given by

$$\psi_{\mathrm{bc}}(-1) = A \quad \text{and} \quad \psi_{\mathrm{bc}}(1) = B \quad . \tag{6}$$

The problem class for our investigations is given by changes of the boundary condition B ($0 \leq B \leq 1$), keeping

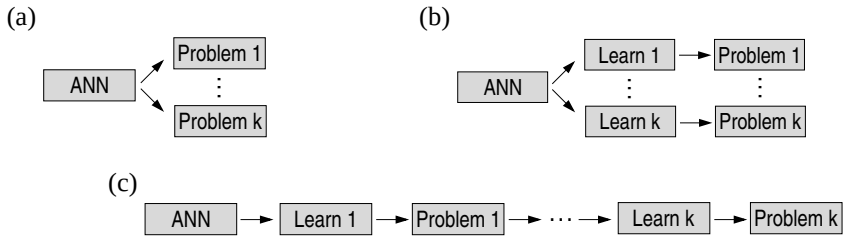

Figure 1: Different set-ups to simulate problem classes and/or changing environments

the other parameters $A = -1$ and $\omega = 1.2$ fixed.

3 Evolutionary algorithm for the design of second order adaptable ANNs

In Hüsken, Goerick, & Vogel (2000) it has been shown that learning can be accelerated, if the weights are initialized with the weights resulting from a previously learned closely related problem. Closely related problem means here a slightly varied boundary condition B or natural frequency ω. We call this method *pre-training*. However, it has also been pointed out that this method can not deliver optimal initial weights for the whole class of problems, as only one problem of the class is taken into consideration. Therefore, another speedup in multiple learning of related problems can be expected if the weights are chosen with respect to the problem class and the learning algorithm. The same line of argument applies to the approach proposed by Gegout (1995), where the suggested initialization does not account for the class of problems and the learning procedure.

One idea of doing so could be to train an ANN with respect to some of the problems of the problem class, using an error function like

$$e = \sqrt{\frac{1}{N}\sum_{\mathrm{i}=1}^{N} |e_i|^2} \quad . \tag{7}$$

Here, N is the number of problems taken into consideration and e_i is the error of the ANN when solving problem i.

As the data of the different problems will be contradictory, it is unlikely that this ANN performs good on the different problems (see figure 1 (a)) as only something like the expectation of the output will be learned. Since in this kind of design neither the demand of fast learning nor the learning algorithm itself is taken into account, no great improvement of performance compared to pre-trained ANNs is to be expected.

Therefore, we introduce a way to determine the initial weights taking into account the speed of adaptation to different problems. The ANNs resulting from this algorithm should have the property to fast learn different, but related, problems (see figure 1 (b)) and to adapt to changes in the problem (see figure 1 (c)). In analogy to the generalization of ANNs against different data sets stemming from the same system, we introduce a generalization against different problems, stemming from the same *class* of problems. Therefore, we call this property of ANNs *second order generalization* or *second order adaptability*.

The main point within the design of second order adaptable ANNs is an optimization of the initial weights with respect to the learning capacity. Of course, this can only be evaluated by means of a period of learning. Therefore, it becomes highly inefficient to apply a gradient-based optimization algorithm, as in this case something like the derivative of the quality of the learning period with respect to the initial weights has to be evaluated.

Therefore, we apply an *evolutionary algorithm*, a lowest order optimization algorithm, based on the principles of natural evolution. Each possible solution of the problem, i. e. each set of initial weights, is represented by one individual of a population. In each generation, the individuals of the parent population generate offsprings by means of recombination and mutation. These offsprings are evaluated with respect to their ability to solve the given problem, called *fitness*, and offsprings with a better fitness are selected with a higher probability as the parents of the next generation.

In principle, two methods, based on evolutionary algorithms, exist to find the weights of second order adaptable ANNs, depicted in figure 2. One possible way for the design is based on evolution in changing environments (see figure 2 (a)). The individuals in generation t are selected for the ability of learning problem i_t and in generation $t+1$ for the ability of learning a different problem i_{t+1}. As the second order adaptability of one individual is evaluated across successive generations, we call this approach *generation method*. Nolfi & Parisi (1997) and Sasaki & Tokoro (1997) have shown that in principle it is possible with such a method to generate individuals that can adapt faster to

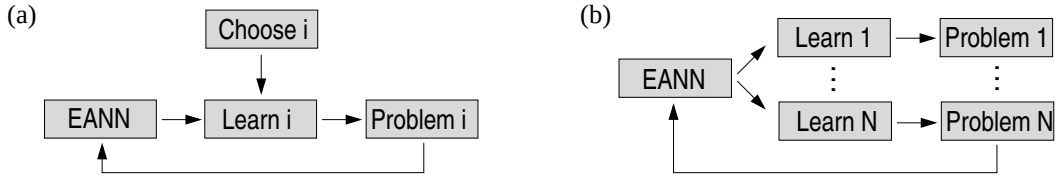

Figure 2: Different strategies to develop 2^{nd} order adaptive ANNs which can cope with changing environments

two different tasks. However, some preliminary experiments have shown this approach to have some fundamental disadvantages, which render it unsuitable for our purpose.

Due to these difficulties, we introduce a different method, called *ensemble method*, depicted in figure 2 (b). The basic idea is to select the individuals in each generation with respect to their second order adaptability. This value is estimated by measuring the adaptability towards a number of problems from the problem class. For the estimation, we use equation (7), but e_i is given by the error *after* performing a limited number of learning cycles towards the problem i. To optimize the initial weights with respect to other goals like high generalization ability or the reduction of the final error after an infinite adaptation process, only e_i has to be changed accordingly. It can be seen that this kind of evaluation of the initial weights depends on both the learning speed and the learning algorithm.

In many applications of the combination of ANNs and evolutionary algorithms for one problem, it has turned out to be beneficial to modify the individuals' weights not only by means of mutations, but also with respect to the weight modification through learning, known as Lamarckian inheritance. As in our case learning takes place multiple times with respect to different problems, this scheme can not be carried over directly. Therefore, we introduce a heuristic approach, which can be called *averaged Lamarckian inheritance*. The idea is to analyze similarities and differences of the weights resulting from learning the different problems i. If one particular weight turns out to have similar values after adaptation independent of i, this value seems to be a more convenient starting point for learning all problems of the problem class. Otherwise, if the weights turn out to be very different after learning, especially with different signs, a small value should be chosen as the starting point. One straightforward way to realize this idea is to inherit the arithmetic mean of the weights resulting from learning the different problems.

In different experiments, this algorithm has turned out superior compared with the generation method as well as compared with the ensemble method, using Darwinian inheritance. This is a remarkable result, as robustness against changing environments is achieved by a kind of Lamarckian inheritance (in contrast to other investigations (Sasaki & Tokoro 1997; Sasaki & Tokoro 1998)). However, in the following experiments we will apply the ensemble method to find initial weights of an ANN with a very high second order adaptability, compared to other methods.

4 Experiments and results

We perform two different experiments, referring to the two demands described above and depicted in figure 1 (b) and (c). Therefore, we utilize the problem class (5) and (6) with $0 \leq \psi(1) = B \leq 1$. In these experiments, we compare the learning process of second order adaptable ANNs, with the learning process of ANNs with randomly initialized weights or with the solution of a modified problem as initialization. The second order adaptable ANNs are generated with respect to five problems with the boundary values $B = 0, 0.25, 0.5, 0.75, 1$. This number of problems has turned out to be sufficient for representing knowledge about the whole problem class; but even for a lower number of problems, the ANNs are second order adaptable. Pre-training is performed till the error E' falls below 10^{-5}.

In both experiments we use fully connected feedforward multilayer perceptrons with one layer of 10 hidden neurons with sigmoidal activation functions and linear output neurons. We use 20 equidistant points as the set D of points to evaluate the fulfillment of the DE. Due to the superior generalization ability of ANNs, these are here enough points to find a good overall solution. Learning is performed using the RPROP-algorithm (Riedmiller 1994) with standard parameters.

4.1 Learning one particular problem

In our first experiment, we investigate the ability of the different initialized ANNs to learn the solution of different problems of the problem class, i. e. solutions of the DE with different boundary conditions $\psi(1) = B$. This experiment is related with the demand, depicted in figure 1(b). The pre-trained ANN is generated by learning the problem with $B = 0.5$. This seems to be the best initialization possible using this method, as experiments have shown that the

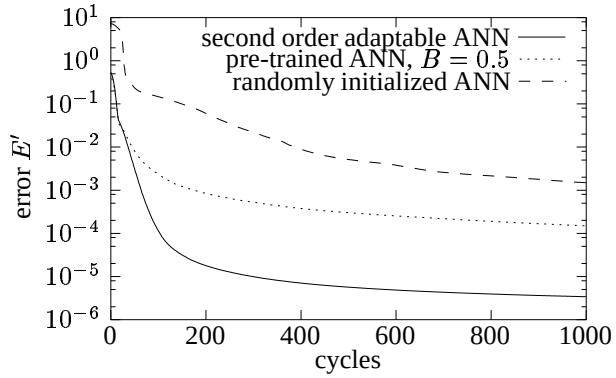

Figure 3: Learning curves, averaged over the boundary problems $0 \leq \psi(1) = B \leq 1$

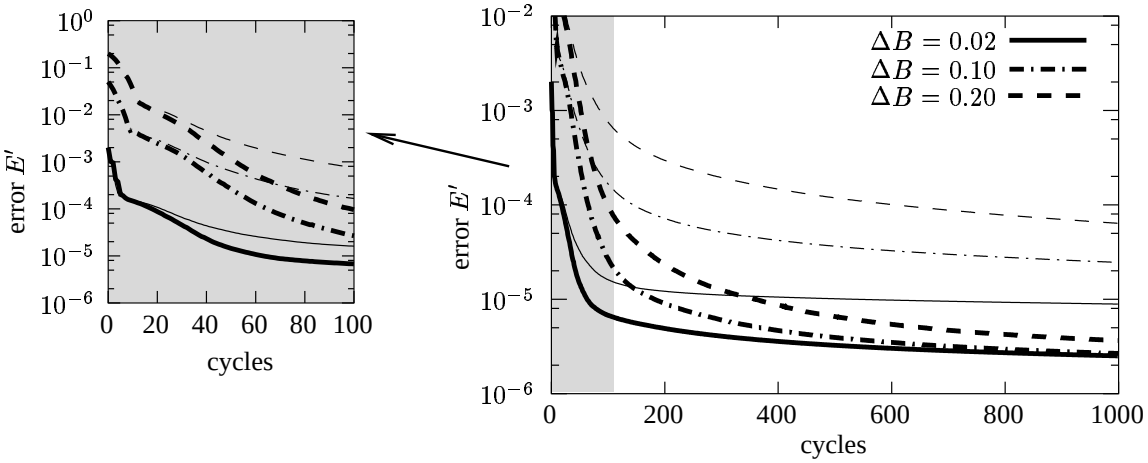

Figure 4: Ability to track the solution of the changing boundary value problem of a pre-trained second order adaptable ANN (thick lines) and randomly initialized and pre-trained ANN (thin lines), the amount of the change is given by ΔB

knowledge becomes less supportive the more the problem has changed (Hüsken, Goerick, & Vogel 2000). Therefore, this problem is supposed to be of average type.

The results, averaged across 100 different values of $0 \leq B \leq 1$ are given in figure 3. Obviously, the second order adaptable ANN performs best in terms of the learning speed on the different problems. Additionally, it is to be mentioned that the best and the worst result as well as the variance within the 100 runs is smallest for the second order adaptable ANN.

4.2 Tracking the solution of changing problems

In this section, we investigate the ability of fast adaptation to changes in the problem, as depicted in figure 1 (c). Therefore, problem B_1 is learned till the error falls below 10^{-5}. Afterwards, the problem is changed by $\Delta B = |B_2 - B_1|$ and learning starts towards the new problem B_2. The initial learning period either starts from randomly chosen weights or from a second order adaptable ANN.

The learning curves of this second period of learning, averaged across 100 different values of $0 \leq B_2 \leq 1$, are given in figure 4. In about the first 20 training cycles, the curves only depend on ΔB but not on the initial weight state in the first period. This is evident as learning has not really started yet and the error is only determined by the error of the solution of problem B_1 evaluated using the error function (4) belonging to problem B_2. Of course, this is a function of ΔB. In the succeeding learning cycles, learning of the second order adaptable ANNs is sped up due to the more beneficial initial weight state. As can be seen in figure 4, the results of the second order adaptable ANN are less sensitive to ΔB.

5 Discussion and outlook

The capacity of an artificial neural network (ANN) for learning depends strongly on the initial weights. We focused on finding initial weights not only for learning one particular problem, but for problems out of a class of related ones. The learning behavior of an ANN is also influenced by the learning algorithm, which is here taken into account during the optimization of the initial weights.

The proposed method was applied to the problem class of learning the solution of a differential equation with variable boundary conditions. The resulting initial weights were corroborated as being better suited, compared to other ways of initializing, for learning particular problems of the problem class as well as tracking changes of the problem, i. e. changes in the boundary condition. Especially, the error decreases faster, the variance within different runs becomes lower, i. e. the learning becomes more robust, and the ability to track larger changes of the problem was improved. These considerable advantages justify the increased computational cost for the design of the initial weights.

It is well known that not only the initial weights but also the ANN's architecture, i. e. the neurons and their connections, strongly influence the learning behavior of an ANN. In future, we will carry over the ideas of this paper to the optimization of both, architecture and initial weights. Another goal is the more stringent definition of problem classes.

Acknowledgment The authors would like to thank Jens Gayko and Bernhard Sendhoff for their support.

References

Bishop, C. M. (1995). Neural Networks for Pattern Recognition. Oxford: Oxford University Press.

Gegout, C. (1995). Improvement of Multilayer Perceptron Trainings with an Evolutionary Initialization. In ICANN95, Proceedings of the International Conference on Artificial Neural Networks, pp. 153–158.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural Networks, 2, 359–366.

Hüsken, M., Goerick, C., & Vogel, A. (2000). Fast adaptation of the solution of differential equations to changing constraints. In Proceedings of the Second International ICSC Symposium on Neural Computation (NC'2000). ICSC Academic Press.

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural network for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks 9(5), 987–1000.

Lagaris, I. E., Likas, A., & Papageorgiou, D. G. (1998). Neural network methods for boundary value problems defined in arbitrarily shaped domains. Technical Report 7-98, Department of Computer Science, University of Ioannina, Greece.

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient backprop. In G. B. Orr & K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade, pp. 9–50. Springer-Verlag.

Nolfi, S. & Parisi, D. (1997). Learning to adapt to changing environments in evolving neural networks. Adaptive Behavior 5(1), 75–98.

Reed, R. D. & Marks, R. J. (1999). Neural Smithing. Cambridge: The MIT Press / Bradford Books.

Riedmiller, M. (1994). Advanced supervised learning in multi-layer perceptrons – from backpropagation to adaptive learning algorithms. International Journal of Computer Standards and Interfaces 16(5), 265–278.

Sasaki, T. & Tokoro, M. (1997). Adaptation toward changing environments: Why Darwinian in nature? In P. Husbands & I. Harvey (Eds.), Fourth European Conference on Artificial Life (ECAL97), pp. 145–153. MIT Press.

Sasaki, T. & Tokoro, M. (1998). Adaptation under changing environments with various rates of inheritance of acquired characters: Comparison between Darwinian and Lamarckian evolution. In B. McKay, X. Yao, C. Newton, J.-H. Kim, & T. Furuhashi (Eds.), Simulated Evolution and Learning. Springer.