Neural Network Model of the Backpropagation Algorithm

Rudolf Jakša
Department of Cybernetics and Artificial Intelligence
Technical University of Košice
Letná 9, 041 20 Košice
Slovakia
jaksa@neuron.tuke.sk

Miroslav Katrák
Department of Cybernetics and Artificial Intelligence
Technical University of Košice
Letná 9, 041 20 Košice
Slovakia
bracek@mizu.sk

Abstract

We apply a neural network to model neural network learning algorithm itself. The process of weights updating in neural network is observed and stored into file. Later, this data is used to train another network, which then will be able to train neural networks by imitating the trained algorithm. We use backpropagation algorithm for both, for training, and for sampling the training process. We imitate the training of the network as whole. All the weights and weight changes of multilayer neural network are processed in parallel in order to model mutual dependencies between weights. Experimental results are provided.

Keywords: metalearning, learning to learn, error backpropagation.

1 Introduction

Adaptive or optimizing learning algorithms might be used in the neural network learning or in the machine learning domains. Instead of fixed learning algorithms, these algorithms improve their own learning performance over time, or they develop particular learning methods from scratch. This type of learning algorithms is known as the "metalearning" or the "learning to learn" approach. Works by Jürgen Schmidhuber and Sepp Hochreiter [2] [3] are representative of recent research in this area and more comprehensive overview is given by Sebastian Thrun in [4]. Thrun defines learning to learn as ability of algorithm to improve performance at each next task with experience from previous tasks [4]. Schmidhuber emphasizes on the ability of learner to evaluate and compare learning methods and course of learning, and using this evaluation to select proper learning strategy [2].

We can recognize following paradigms among metalearning approaches:

- similarity exploitation,
- learning parameters adaptation,
- discovery of learning algorithm.

Particular methods might be focused on any of these paradigms, or on all of them.

Similarity exploitation is the idea that a group of tasks shares some similarity, which once learned, might speed-up the learning of another tasks. We can simply learn the sequence of tasks to exploit their similarity, but some mechanism for distinguishing the task-specific knowledge versus common cross-task knowledge should improve the performance.

Adaptation of learning parameters might be done by some meta-learning algorithm above the learning algorithm. This paradigm might be based more on the "learning about learning", then on the "learning to learn" idea. However, knowledge about learning is the step to the "learning to learn".

Discovery of learning algorithm is the design of learning algorithm from scratch. This is more "learning the learning" then "learning to learn". Here we shift focus from adaptation into learning.

Metalearning algorithms might be based on the reinforcement learning or on supervised learning. In the reinforcement learning case, the learner through trial-and-error experience improves not only its performance on some particular task, but also its ability to learn. This can be achieved by treating learning algorithms as a part of solved task. It is: learning is one of actions of the learner. In the supervised learning scenario, learning and metalearning are usually treated as independent processes.

2 Backpropagation Imitation

In this section we will describe modelling of the error backpropagation algorithm. We want train neural network to train another neural network. A neural network model of error backpropagation algorithm should be able to train neural networks in the similar manner as the original backpropagation algorithm did. To obtain such a model we will sample the training process of backpropagation learning and then try to imitate it. This is simple, while also a general approach to metalearning. It consists of the following sequence:

1. train arbitrary neural network with error backpropagation algorithm and sample the learning process,
2. train the learning network to imitate original learning algorithm,
3. train arbitrary neural network using the learning network.

Consider multilayer neural network with neuron activations x_i, link weights w_{ij}, biases θ_i, and neuron activation functions $f_i(in_i)$:

$$x_i = f_i(in_i) \qquad in_i = \sum_{j=1}^{M} w_{ij}x_j + \theta_i \tag{1}$$

The in_i is input into i-th neuron and M is the number of links connecting into i-th neuron. The error J in the supervised learning mode is defined:

$$J^p = \frac{1}{2}\sum_{i=1}^{N_0}(ev_i^p - x_i^p)^2 \tag{2}$$

The p is the index of data pattern, N_0 is the number of output neurons of neural network, and ev_i^p is the expected output of i-th neuron on p-th pattern. For simplicity, we will omit pattern index p later. Gradient based error minimizing adaptation of weights follows:

$$\Delta w_{ij} = -\gamma\frac{\partial J}{\partial w_{ij}} = -\gamma\frac{\partial J}{\partial in_i}\frac{\partial in_i}{\partial w_{ij}} = \gamma\delta_i x_j \tag{3}$$

The weight w_{ij} links the j-th neuron into i-th neuron, the γ is learning rate constant, and δ_i is defined as:

$$\delta_i = -\frac{\partial J}{\partial in_i} = -\frac{\partial J}{\partial x_i}\frac{\partial x_i}{\partial in_i} = -\frac{\partial J}{\partial x_i}f'(in_i) \tag{4}$$

The $f'(in_i)$ is the derivative of activation function $f(in_i)$. For output neurons we get:

$$\delta_i = -\frac{\partial J}{\partial x_i}f'(in_i) = (ev_i - x_i)f'(in_i) \tag{5}$$

For neurons in hidden layers we get:

$$\delta_i = -f'(in_i)\sum_{h=1}^{N_h}\frac{\partial J}{\partial in_h}\frac{\partial in_h}{\partial x_i} =$$

$$= -f'(in_i)\sum_{h=1}^{N_h}\frac{\partial J}{\partial in_h}\frac{\partial}{\partial x_i}\sum_{l=1}^{N_l}w_{hl}x_l =$$

$$= -f'(in_i)\sum_{h=1}^{N_h}\frac{\partial J}{\partial in_h}w_{hi} = f'(in_i)\sum_{h=1}^{N_h}\delta_h w_{hi} \tag{6}$$

The N_h is number of links coming from i-th neuron and h is index of these links and corresponding neurons. The N_l is number of neurons which have connections into h neurons (see Fig.1). The rule (6) is the error backpropagation rule, defining the backward propagation of error through network. Rule (3) defines weight changes minimizing this error, and rule (5) sets the base for error minimization.

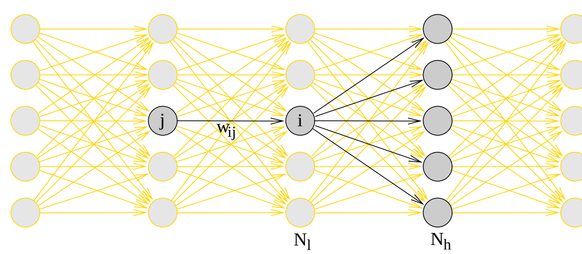

Figure 1: Neuron indices for rule (6).

The error backpropagation algorithm is defined by rules (1), (2), (3), (5), and (6). To model this algorithm we may sample variables: Δw, w, θ, δ, x, ev, J, in, and $f'(in)$. Some of these variables can be derived from others, so not full set of them is necessary. We can model either, the rule (6), or the full set of rules. When modelling full set of rules, interactions in the whole network may be processed in model. When modelling rule (6) only, only neighborhood of particular link is considered.

The number of inputs and outputs of learning network is equal to the number of sampled variables. Outputs are Δw changes, and $\Delta\theta$ possibly. To use learning network, rules (6), (5), and (3) from original backpropagation algorithm have to be replaced with outputs of this learning network.

3 Experiments

Consider the neural network on the Fig.2 with two inputs, one hidden neuron, and one output. It has three weights and two biases, which changes we will try to approximate with learning network. Thus, we will sample these changes while learning with the backpropagation algorithm. Then, we will train learning network to approximate them. Besides these changes, we will sample all three weights and two biases, two inputs, one output, and one expected value on output. This is: 9 inputs and 5 outputs for the learning network. Such a learning network with two hidden neurons is on the Fig.3. The number of hidden neurons is arbitrary, it might depend on the tasks learned and on the complexity of original training algorithm, in our case – error backpropagation.

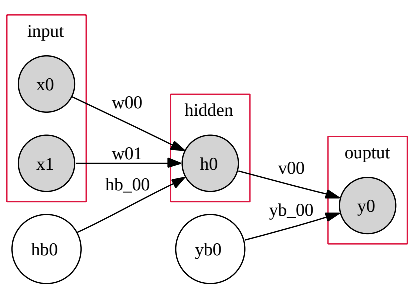

Figure 2: Simple network to be trained. The x_0, and x_1 are inputs; y_0 is output; w_{00}, w_{01}, and v_{00} are weights; hb_{00}, and yb_{00} are biases; h_0 is hidden neuron activation, and y_0 is the output.

In the 1st experiment we will train the network from Fig.2 to approximate boolean function AND (Tab.1). This is simple task and networks learn quickly. Parameters of training of basic network are: $\gamma = 0.23$, number of training cycles is 5000. Parameters of training of learning network are: $\gamma =$ 0.198, number of training cycles is 200, number of hidden neurons is 2. Network topology is the same as on the Fig.3.

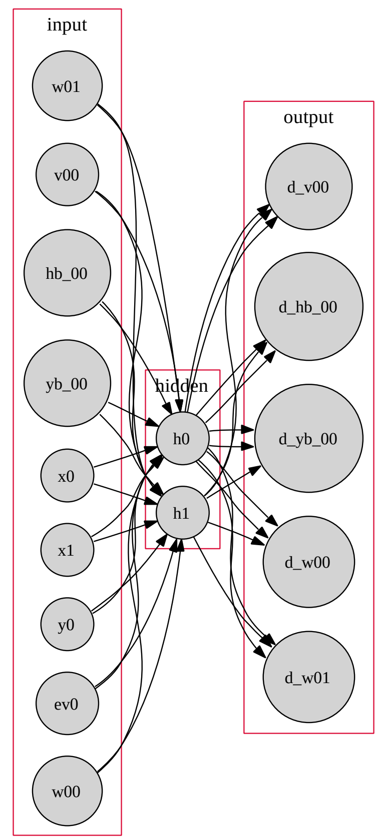

Figure 3: Learning network with two hidden neurons for training of network from Fig.2. Inputs are the variables describing the state of neural network on Fig.2 and outputs are changes of them provided by the learning algorithm.

AND

x_0	x_1	y_0
0	0	0
0	1	0
1	0	0
1	1	1

OR

x_0	x_1	y_0
0	0	0
0	1	1
1	0	1
1	1	1

Table 1: Training data for the boolean AND and OR functions for network on Fig.2.

The training history for learning network is on the Fig.4. Comparison of training of basic network using backpropagation algorithm and using learning network is on the Fig.5. The learning network achieved better convergence then the original backpropagation algorithm. However, the performance of learning network depends also on its training – it is prone to overfitting. Also note, that implementations of training using learning network and error backpropagation algorithm may differ in speed. In our case, in this experiment, backpropagation training took 0.408 seconds, while learning network training took 3.781 seconds.

In the 2nd experiment we will use basic network without a hidden layer. This is sufficient for the AND-function approximation. We will have 2 inputs, 1 output, and 0 hidden neurons in the basic network; and 7 inputs, 3 outputs, and 0 hidden neurons in the learning network. The training history for this learning network is on the Fig.6. Comparison of training of basic network without hidden neurons using backpropagation algorithm and using learning network is on the Fig.7. Performance of learning network in this setup is comparable to performance of backpropagation algorithm, although it is slightly worse then in the 1st experiment with hidden neurons.

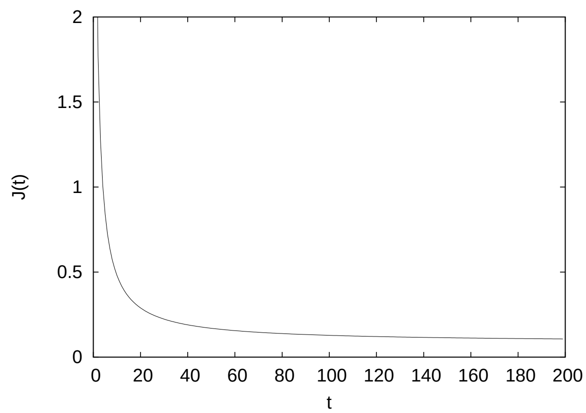

Figure 4: Error of the training of learning network for the basic network from Fig.2.

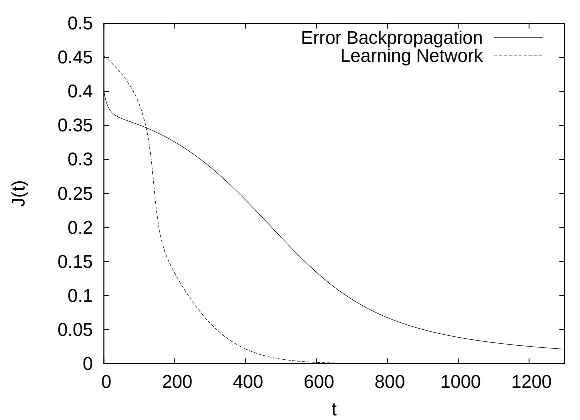

Figure 5: Error of the training of basic network from Fig.2 using error backpropagation algorithm and using the learning network from Fig.3.

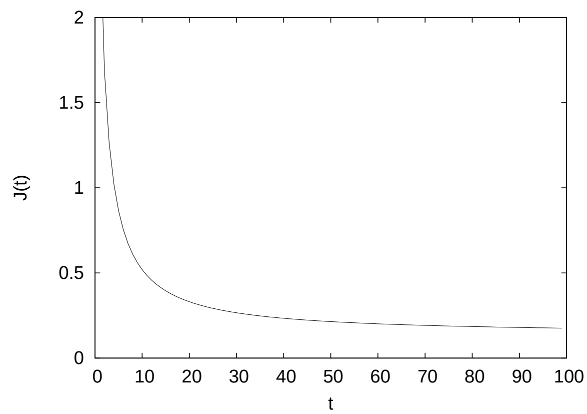

Figure 6: Error of the training of learning network for the basic network without hidden neurons.

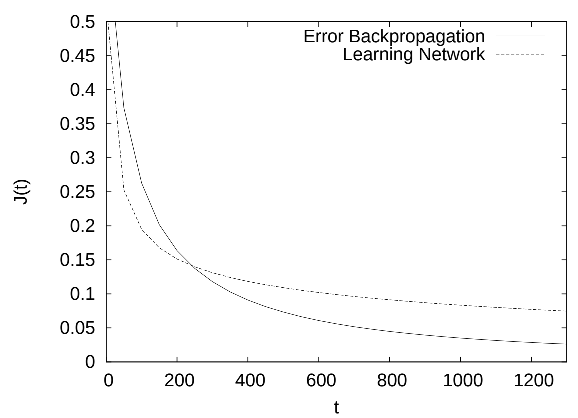

Figure 7: Error of the training of basic network without hidden neurons using error backpropagation algorithm and using the learning network.

In the 3rd experiment we will increase the number of hidden neurons. We will have 2 inputs, 1 output, and 4 hidden neurons in the basic network; and 21 inputs, 17 outputs, and 3 hidden neurons in the learning network. The task is OR-function approximation. The training history for this learning network is on the Fig.8. Comparison of training of basic network without hidden neurons using backpropagation algorithm and using learning network is on the Fig.9. Performance of learning network in this setup is again better then original error backpropagation algorithm.

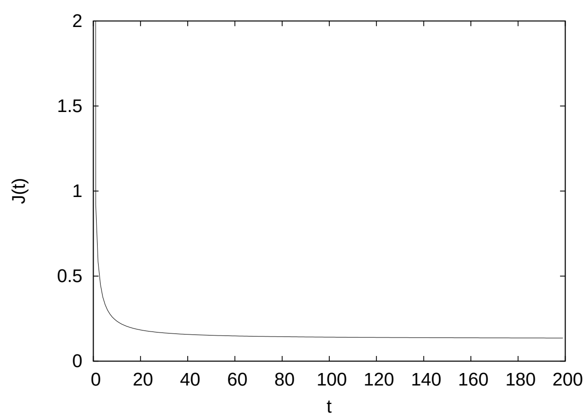

Figure 8: Error of the training of learning network for the basic network with 4 hidden neurons.

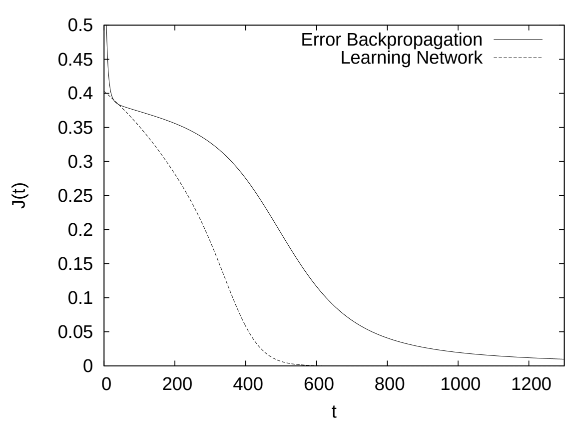

Figure 9: Error of the training of basic network with 4 hidden neurons using error backpropagation algorithm and using the learning network.

In the 4th experiment we will try more difficult classification task. The Fig.12 depicts training and testing data sets. The network topologies are: 2 inputs: x and y, 2 outputs: *innersquare* and *outersquare*, and 5 hidden neurons for the basic network; and 38 inputs, 27 outputs, and 3 hidden neurons for the learning network. The training history for this learning network is on the Fig.10. Comparison of training of basic network using backpropagation algorithm and using learning network is on the Fig.11. Training with the learning network in this task diverges, while training with backpropagation algorithm stopped on some level but did not diverge.

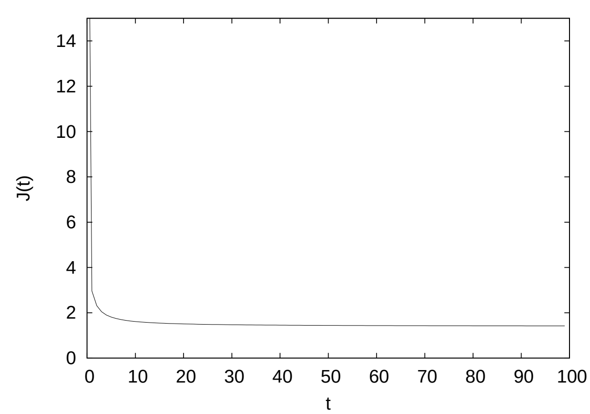

Figure 10: Error of the training of learning network for the basic network for square classification task.

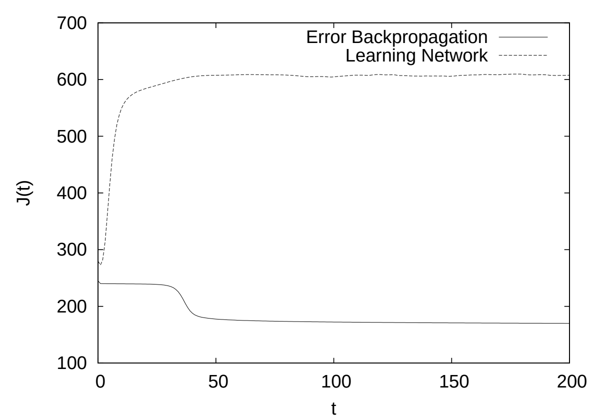

Figure 11: Error of the training of basic network for square classification task using error backpropagation algorithm and using the learning network.

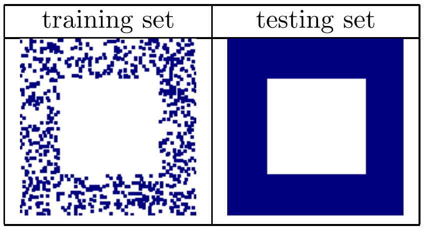

Figure 12: Training and testing sets for the classification task. The task is to classify points in space by their x and y coordinates, whether they will fit into inner square or not.

4 Analysis

The AND/OR-functions approximation tasks with hidden units show good results when trained with learning network. The ability of learning network to outperform backpropagation algorithm seems promising. In future, knowledge from several training algorithms might be used to train learning network in order to get even better performance by exploiting best of all of these algorithms. Trial and error mode might be further used when training learning network to further improve its performance, possibly beyond the reach of conventional learning algorithms.

Slightly worse performance in the experiments without hidden units points to the nonlinear character of either, learning rules of backpropagation algorithm, or the neural network which we train.

The problem with application of our approach to more complex classification task might be of similar character as instability of overtrained learning network. Chance of instability of learning with learning network is an inherent property of this approach. The better performance with simpler networks favorizes the modelling of the rule (6) only of backpropagation algorithm, instead of modelling all the rules, which we did in experiments. To further investigate the neural network modelling of backpropagation algorithm, rule (6) modelling, and deeper analysis of variable set for algorithm sampling might help.

5 Conclusion

Using neural network model of backpropagation algorithm to train neural networks is a viable approach. Novel methods of performance tuning of learning algorithm are possible when using this model. There is, however, a risk of learning instability with this approach, and actual modelling of backpropagation can be done in several different modes.

References

[1] M.Katrák, Metalearning methods for neural networks, (in Slovak), MS Thesis, Technical University Košice, (2005). neuron.tuke.sk/~jaksa/theses

[2] J.Schmidhuber, J.Zhao, and M.Wiering, Simple principles of metalearning, Technical Report IDSIA-69-96, IDSIA, (1996). citeseer.ist.psu.edu/schmidhuber96simple.html

[3] S.Hochreiter, A.S.Younger, and P.R.Conwell, Learning to Learn Using Gradient Descent, Lecture Notes in Computer Science, vol.2130, (2001). citeseer.ist.psu.edu/hochreiter01learning.html

[4] S.Thrun, Learning To Learn: Introduction. citeseer.ist.psu.edu/article/thrun96learning.html

[5] J.Schmidhuber, Evolutionary Principles in Self-Referential Learning, Diploma Thesis, Technische Universität München, (1987). www.idsia.ch/~juergen/diploma.html

[6] J.Schmidhuber, On Learning How to Learn Learning Strategies, Technical Report FKI-198-94, Fakultt für Informatik, Technische Universitt München, (1994). citeseer.ist.psu.edu/schmidhuber95learning.html

[7] J.Schmidhuber, A General Method for Incremental Self-Improvement and Multi-agent Learning in Unrestricted Environments, In X.Yao (Ed.), Evolutionary Computation: Theory and Applications, Scientific Publ. Co., Singapore, (1996). citeseer.ist.psu.edu/article/schmidhuber96general.html

[8] J.Schmidhuber, A Neural Network That Embeds Its Own Meta-Levels, In Proc. of the International Conference on Neural Networks '93, San Francisco. IEEE, (1993). citeseer.ist.psu.edu/schmidhuber93neural.html