Use of Genetic Programming for the Search of a New Learning Rule for Neural Networks

Samy Bengio and Yoshua Bengio and Jocelyn Cloutier

***Abstract*— In previous work ([1, 2, 3]) we explained how to use standard optimization methods such as simulated annealing, gradient descent and genetic algorithms to optimize a parametric function which could be used as a learning rule for neural networks. To use these methods, we had to choose a fixed number of parameters and a rigid form for the learning rule. In this article, we propose to use genetic programming to find not only the values of rule parameters but also the optimal number of parameters and the form of the rule. Experiments on classification tasks suggest genetic programming finds better learning rules than other optimization methods. Furthermore, the best rule found with genetic programming outperformed the well-known backpropagation algorithm for a given set of tasks.**

I. Introduction

Learning mechanisms in neural networks are usually associated to changes in synaptic efficiency. In such models, a synaptic learning rule controls the variations of the parameters (synaptic weights) of the network. Researchers in neural networks have proposed learning rules based on mathematical principles (such as backpropagation [8]) or biological analogy (such as Hebbian rules [6]), but better learning rules may be needed to achieve human-like performance for many learning problems.

We proposed in [3] (see also [4] for a similar approach) a method to find new learning rules using standard optimization techniques. This method considers the learning rule of a neural network as a parametric function with local inputs, whose values vary for each synapse, and a fixed number of parameters, which are the same for all synapses in the network.

In this paper, we propose to use genetic programming to find not only the parameters of a learning rule but also its form. In section II., we introduce the idea of parametric learning rules. In the next section, we explain how the use of genetic programming for the optimization of parametric learning rules could simplify the design of such parametric functions. Finally in section IV., we show an experimental comparison between the best rules found by different optimization methods on a set of classification tasks and a comparison with the most used learning rule: backpropagation.

II. Parametric Learning Rules

We describe briefly in this section the basic idea of optimizing learning rules. More detailed treatment can be found for instance in [1, 2, 3], but also in [4]. The principle is straightforward: we consider the learning rule as a parametric function and we optimize its parameters using an optimization technique. Consider the following generic parametric learning rule:

$$\Delta w(i,j) = f(x_1, x_2, ..., x_n; \theta_1, \theta_2, ..., \theta_m) \quad (1)$$

where $\Delta w(i,j)$ is the weight change of the synapse from neuron i to neuron j and $f(\cdot)$ is a parametric function of n local variables x and m parameters θ.

Local variables represent informations and mechanisms that may influence a particular synapse, such as presynaptic activity and postsynaptic potential. Figure 1 shows the interaction between those elements. Parameters θ share the same values for all synapses and are found using optimization methods.

In order for a learning rule obtained through optimization to be useful, it must be successfully applicable in training networks for new tasks (i.e., tasks other than those used during optimization of the learning rule). This property of a learning rule is a form of *generalization*. We showed in [2] that this kind of generalization could be described with a formalism used to define the generalization property of learning systems, based on the notion of *capacity* [9]. This led to several important conclusions. For example, the expected error of a learning rule over new tasks should decrease when increasing the

Samy Bengio is from *Centre National d'Etudes des Télécommunications*, LAB/RIO/TNT, 22301 Lannion, France, while Yoshua Bengio and Jocelyn Cloutier are from *Université de Montréal*, Département IRO, Case Postale 6128, Succ. "A", Montréal (QC) Canada, H3C 3J7, e-mail: bengio@lannion.cnet.fr

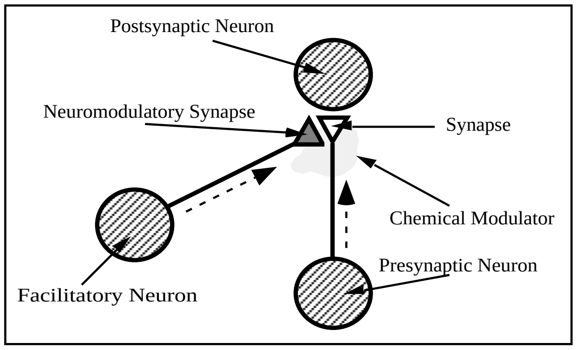

Figure 1: Elements found in the vicinity of a synapse, which can influence its efficacy.

number of tasks used for learning the parameters. However, it could increase with the number of parameters and the capacity of the learning rule class if an insufficient number or variety of training tasks are used during optimization.

These remarks help us in the design of such parametric learning rules. We need a rule class whose capacity matches the number and variety of the training tasks.

III. Use of Genetic Programming

When using standard optimization methods such as genetic algorithms, simulated annealing or gradient descent for the optimization of a synaptic learning rule, one needs to fix the exact form and number of parameters of the parametric learning rule in order to optimize it. In doing so, one could neglect a good solution just because its form has not been expected. A recent optimization technique, *genetic programming* [7], could help to correct this design problem.

Genetic programming is essentially based on the same principles as genetic algorithms ([5]): an initial *population* of *individuals*, each representing a solution, is randomly chosen. Then selecting the best individuals for *reproduction* and using *crossover* and *mutation* enables one to create a new generation. Repeating this process causes the whole population to improve.

With genetic algorithms, each individual is usually coded as a string of bits. Genetic programming uses a different representation scheme. Each individual is now represented as a tree where nodes are operators and leaves are constants or variables. Crossover now means to randomly swap subtrees from two parent individuals to create two new individuals and mutation means to randomly replace a subtree with a newly created one.

Using this optimization method for our problem of finding a parametric learning rule enables us to simplify the design. Now, we only need to select the appropriate local variables which can influence the synaptic change, the maximum number of parameters and the basic operators that can be used. We no longer have to fix in advance the exact form of the parametric learning rule.

IV. Experiments

We performed experiments to compare genetic programming to other optimization methods. Experiments were conducted under the following conditions:

- The tasks to solve with a new learning rule were two-dimensional classification problems. There were 20 different tasks. Some were linearly separable (10) while others where non-linearly separable (10).
- Each task was learned with 800 training examples and tested with 200 other examples. A task was said to be successfully learned when there was no classification error over the test set.
- We used a fully connected neural network with two input units, one hidden unit and one output unit. Furthermore, we added a backward path of modulator neurons to provide a measure of the error to each unit [1]. Figure 2 show how this backward path is created.

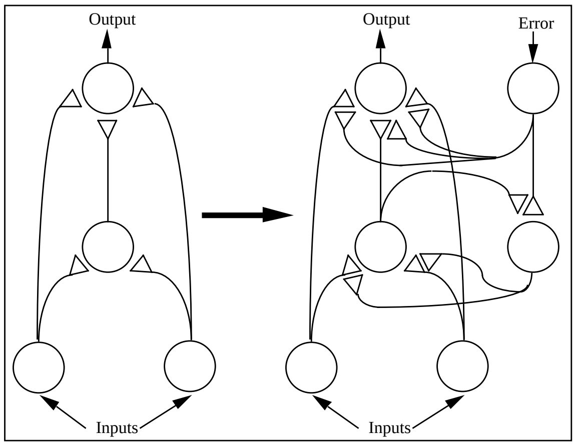

Figure 2: Architecture transformation to enable local evaluation of the network error. A backward path of modulator neurons is added, which influence each forward connections.

- For the standard optimization methods, we used a constrained form for the parametric learning rule. It had 8 parameters, each modulating a specific mechanism. For instance we can see in equation (2) that θ_4 modulates a Hebbian mechanism, while θ_7 modulates a local version of the backpropagation learning rule.

$$\begin{aligned}\Delta w(i,j) &= \theta_0 + \theta_1\ y(i) + \theta_2\ x(j) + \\ &\quad \theta_3\ y(mod(j)) + \\ &\quad \theta_4\ y(i)\ y(mod(j)) + \\ &\quad \theta_5\ y(i)\ x(j) + \theta_6\ y(i)\ w(i,j) + \\ &\quad \theta_7\ y(i)\ y(mod(j))\ f'(x(j)) \qquad (2)\end{aligned}$$

In this equation, $w(i,j)$ is the synaptic efficacy between neurons i and j, $x(j)$ is the activation potential of neuron j (postsynaptic potential), $y(i)$ is the output of neuron i (presynaptic activity), $y(mod(j))$ is the output of a modulatory neuron influencing neuron j, and $f(\cdot)$ is the activation function. This rule has produced good results in previous experiments ([2]). The last mechanism includes backpropagation in the space of learning rules reachable by optimization. This has been done in order to see if we could find it by optimization methods starting from a randomly chosen learning rule (i.e. a set of random parameters θ).

- For both genetic methods, the population size was 100 and the number of generations was limited to 100. Crossover and mutation rate were 60% and we used a rank selection operator [10].
- For the genetic programming method, we used standard arithmetic operators $(+,-,*,/)$ and local variables that could be chosen were the same as in equation (2). We have shown in [2] that to obtain good generalization results, the capacity of the learning rule should be controlled. For this reason, we put an upper bound on the number of parameters θ to 10 and an upper bound on the number of terms of the learning rules to 40.
- A typical experiment was conducted as follows:
  - First we chose an optimization method (genetic algorithms, simulated annealing, or genetic programming[1]) and a set of *training* tasks (from 1 to 9 different tasks, linearly as well as non-linearly separable).
  - Then we optimized the rule for a fixed number of iterations.
  - Finally, we tested the new rule over tasks different from the training tasks.

To learn a task, the weights of the neural network were first initialized to random values (to be sure that our learning rule is able to solve the task from many initial conditions) and then, we modify the weights of the network using the current learning rule applied to the 800 training examples of the task. The generalization error for the current task is then calculated over the 200 other examples.

Figure 3 summarizes these experiments. We can see that the rule found by genetic programming generalized better than those found by the other optimization methods. When optimized with more than one task, it was even better than the backpropagation learning rule. This new rule has the following simple form:

$$\Delta w(i,j) = y(i) \cdot y(mod(j)) \cdot f'(x(j))^3 \qquad (3)$$

For comparison, the backpropagation rule can be written in a local way by the following rule:

$$\Delta w(i,j) = y(i) \cdot y(mod(j)) \cdot f'(x(j)) \qquad (4)$$

We can see that the only difference with backpropagation is the exponent of the activation function derivative. As the overall sign of this term is conserved, and since the derivative is a real number between 0 and 1, the effect of the derivative over the weight change is thus stronger than in backpropagation, but in a 2-layer net, the resulting weight change direction is guarenteed to be downhill.

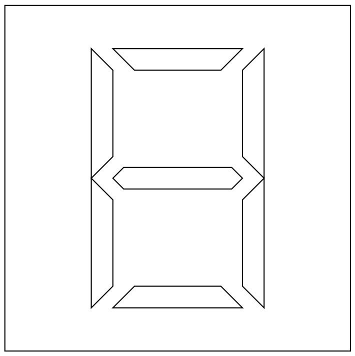
Figure 4: Representation of the input space of the LED task.

Finally we were curious to see if our new learning rule was able to solve tasks using a different network architecture. We tested it on a character recognition task with 7 input dimensions (the LED representation of figure 4), 10 hidden units and 10 output dimensions (one for each digit). The new rule was able to learn the task with no generalization error. For comparison, backpropagation was also able to solve perfectly the problem, but with more iterations[2].

[1]Gradient descent was also tested but there seems to be a lot of local minima in the space of learning rules which explains why we were not able to get any good solution with this local method.

[2]Since the number of iterations is not necessary a good indicator of the quality of a learning rule we do not suggest that our

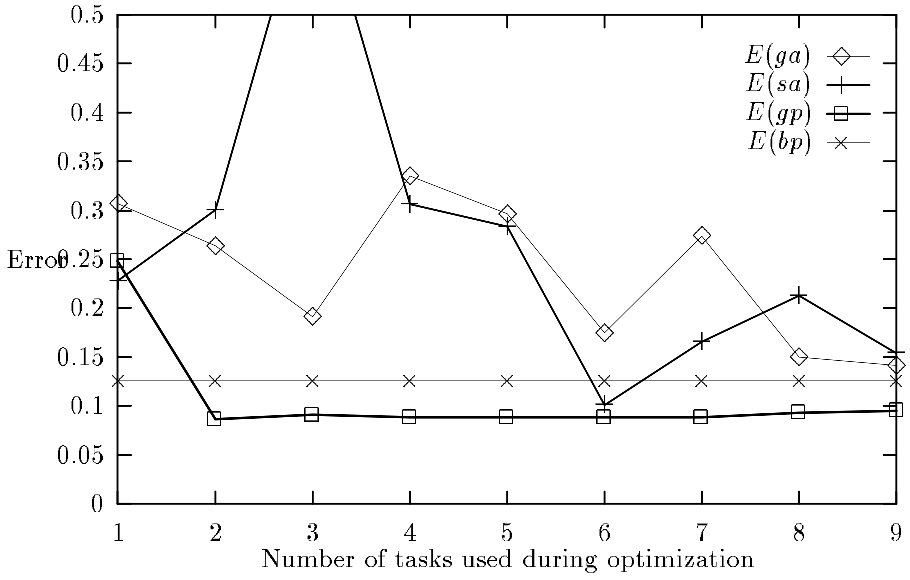

Figure 3: Comparison between generalization performance of the best rules found by different optimization methods and backpropagation. *ga* means genetic algorithms, *gp* is genetic programming, and *sa* stands for simulated annealing. Finally, *bp* means we used backpropagation instead of an optimized learning rule (of course in this case, the error does not depend on the number of tasks, which explains why the *bp* curve is a straight line). For each optimization method, we show the generalization error $E(\cdot)$ with respect to the number of tasks used to optimize the rule, where each task is trained using 800 examples and tested using 200 examples.

V. Conclusion

In this article we proposed the use of genetic programming for the search of new learning rules for neural networks. This method greatly simplify the design of parametric learning rules. Experiments on classification tasks showed that it can find better learning rules than any other optimization technique. In one case, it even found a rule that is better than backpropagation. Of course it has yet to be shown that the procedure is applicable to more complex tasks.

References

[1] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei, *On the optimization of a synaptic learning rule*, in Conference on Optimality in Biological and Artificial Networks, Dallas, USA, 1992.

[2] ——, *Generalization of a parametric learning rule*, in ICANN '93: Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, Nederlands, 1993.

[3] Y. Bengio and S. Bengio, *Learning a synaptic learning rule*, Tech. Rep. 751, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Montréal (QC) Canada, 1990.

[4] D. Chalmers, *The evolution of learning: An experiment in genetic connectionism*, in Proceedings of the 1990 Connectionist Models Summer School, D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, eds., San Mateo, CA, USA, 1990, Morgan Kaufmann.

[5] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA, USA, 1989.

[6] D. O. Hebb, *The Organization of Behavior*, Willey, New York, NY, USA, 1949.

[7] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Bradford Book, MIT Press, Cambridge, MA, USA, 1992.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, in Parallel Distributed Processing: Explorations in the Microstructure of Cognition - Volume 1: Foundations, D. E. Rumelhart and J. L. McClelland, eds., Bradford Book, MIT Press, 1986.

[9] V. N. Vapnik, *Estimation of Dependencies Based on Empirical Data*, Springer-Verlag, New-York, NY, USA, 1982.

rule is better than backpropagation.

[10] D. WHITLEY, *The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best*, in Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, USA, 1989, Morgan Kaufmann, pp. 116–121.