Constrained Formulations for Neural Network Training and Their Applications to Solve the Two-Spiral Problem[*]

Benjamin W. Wah and Minglun Qian
Department of Electrical and Computer Engineering
and the Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801, USA
E-mail: {wah, m-qian}@manip.crhc.uiuc.edu

Abstract

In this paper, we formulate neural-network training as a constrained optimization problem instead of the traditional formulation based on unconstrained optimization. We show that constraints violated during a search provide additional force to help escape from local minima using our newly developed constrained simulated annealing (CSA) algorithm. We demonstrate the merits of our approach by training neural networks to solve the two-spiral problem. To enhance the search, we have developed a strategy to adjust the gain factor of the activation function. We show converged training results for networks with 4, 5, and 6 hidden units, respectively. Our work is the first successful attempt to solve the two-spiral problem with 19 weights.

1 Formulation of Supervised Neural-Network Training

Traditional supervised neural-network training is formulated as an unconstrained optimization problem of minimizing the sum of squared errors of the output over all training patterns:

$$\min_{w} \quad E(w) = \sum_{i=1}^{n} (o_i(w) - d_i)^2, \tag{1}$$

where o_i and d_i are, respectively, the actual and desired outputs of the network for the i^{th} pattern, n is the number of training patterns, and w is a vector of weights of the neural network trained.

In order for a neural network to generalize well to unseen patterns, we like it to have a small number of weights. Training is difficult in this case because the terrain modeled by (1) is often very rugged, and existing local-search algorithms may get stuck easily in deep local minima. Although global search can help escape from local minima, it has similar difficulties when the terrain is rugged.

Instead of using an unconstrained formulation (1), we propose to formulate neural-network training as a constrained optimization problem that includes constraints on each training pattern. An unsatisfied training pattern in a local minimum of the weight space may provide an additional force to guide a search out of the local minimum. The constrained formulation considered in this paper is:

$$\min_{w} E(w) = \sum_{i=1}^{n} max\{|o_i(w) - d_i| - \epsilon, 0\} \tag{2}$$
$$\text{subject to } h_i(w) = max\{|o_i(w) - d_i| - \epsilon, 0\},$$

where $i = 1, \cdots, n$, $h_i(w)$ is the constraint on the i^{th} pattern, and ϵ is a small positive number that decreases towards 0 as looser constraints are satisfied.

In constrained formulation (2), $E(w)$ should be zero and all constraints should be satisfied when training converges. If ϵ is small enough, then the solution found is a global solution to (1).

Note that the use of constraints does not incur additional computational overheads because they can be computed as $E(w)$ is computed. In the rest of this paper, we use the two-spiral problem to demonstrate the effectiveness of our proposed approach.

[*]Research supported by National Science Foundation Grant CCR 96-32316.

Proc. of the Fifth International Conference on Computer Science and Informatics, Feb. 2000.

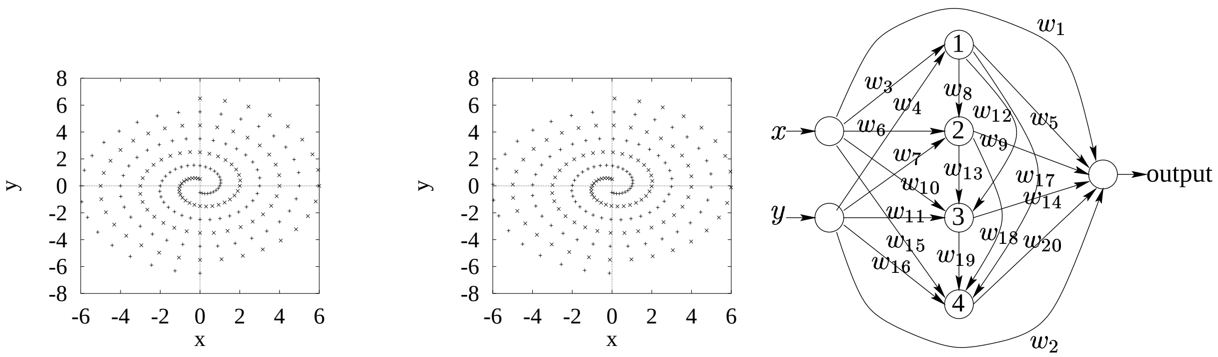

(a) Training set (b) Testing set (c) 4-hidden-unit network

Figure 1. Learning and testing data and a network topology for solving the two-spiral problem.

2 The Two-Spiral Problem

Figures 1a and 1b show the two-spiral problem, a difficult nonlinear classification problem that discriminates between two sets of training points on two distinct spirals in the x-y plane.

The best converged network requires 4 hidden units and 25 weights and was trained by a gradient-based global-search procedure called *Novel* [3]. The disadvantage of *Novel* is that it took over 150 hours on a Sun SS10 to find a converged network, and the result could not be repeated from random starting points. Simulated annealing (SA) [2] is the next best algorithm to solve the two-spiral problem, although it only found converged networks with 6 hidden units [3]. Last, cascade correlation was able to find converged networks with 9 hidden units and 75 weights [1]. Figure 1c shows the topology of a 4-hidden-unit network with shortcuts, where training was done according to the *40-20-40* criterion. (An output is considered to be -1 if it is in the lower 40% of the output range, 1 if it is in the upper 40%, and incorrect otherwise.)

Based on (2), we reformulate neural-network training into a Lagrangian function by adding Lagrange multipliers [6]:

$$\mathcal{L}(w, \lambda) = E(w) + \sum_{i=1}^{n} \lambda_i \max(|o_i(w) - d_i| - \epsilon, 0), \quad (3)$$

where λ_i is the Lagrange multiplier for the constraint on the i^{th} training pattern. We then apply our newly developed constrained simulated annealing algorithm (CSA) [5] to find converged networks. CSA extends SA and has asymptotic convergence to a constrained global minimum with probability one.

One problem-specific characteristic of the two-spiral problem is that the output of the training set is antisymmetric; *i.e.*, if the desired output for (x, y) is 1 (resp. -1), then the desired output for $(-x, -y)$ is -1 (resp. 1). This leads us to use an antisymmetric network and train it using only half of the training set, leading to possible saving of 50% training time. Figure 1c shows an antisymmetric network that eliminates all bias weights and that uses an antisymmetric activation function, such as $tanh$, in both the hidden and output units.

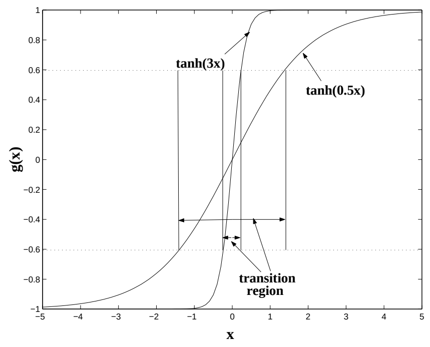


Figure 2. Transition regions of the $tanh(\alpha x)$ activation function with different α where $\epsilon = 0.4$.

3 Tuning the Activation Function

We use the $tanh$ function as our activation function in both the hidden and output units (Figure 2):

$$g(x) = tanh(\alpha x), \quad (4)$$

where α is the gain factor.

Our first implementation using a fixed α considerably larger than one led to a ping-pong effect in constraint satisfaction. Oftentimes, constraint h_i corresponding to pattern i was not satisfied for a long time, leading to large λ_i. At one point, h_i was satisfied and jumped from the wrong state (near -1 or 1) to the correct state (near 1 or -1), while at the same time, quite a few other constraints with relatively small Lagrange multipliers jumped from their correct states to their wrong states.

The above scenario happened because the $tanh$ function has a very sharp transition region when α

Table 1. Training results on 4, 5, and 6-hidden-unit networks. Success ratio m/n indicates m out of n runs that achieve 100% training correctness. The average training time shows the average CPU time per run that achieves 100% training correctness. All runs were done on Pentium 3/450 computers under Solaris.

Number of hidden units	Number of Weights	Best Solution		Training Time (Seconds)			Succ. Ratio
		Training (%)	Testing (%)	Shortest	Longest	Average	
6	35	100	95.9	1532.3	2824.3	2210.0	5/5
5	27	100	94.8	2334.4	6987.3	4040.7	5/5
4	20	100	97.9	19473.6	54449.7	35866.2	4/5

is considerably larger than 1 (Figure 2). When constraint h_i is in the wrong state and far away from the transition region, a move of x in the right direction translates into very little changes in $g(x)$. In fact, this change is so small that it may easily be dominated by other patterns' movements in the Lagrangian function. Hence, h_i may be unsatisfied for a long time, leading to large λ_i. When finally λ_i is so large that causes h_i to be satisfied, a few other patterns may change from their correct states to incorrect ones due to the sharp transition region in $g(x)$. Consequently, we observed oscillations during training: some constraints that were unsatisfied for a long time were finally satisfied, but led to the violation of some other constraints; after a while, these unsatisfied constraints got satisfied because their Lagrange multipliers grew to be very large, but some other satisfied constraints might become unsatisfied. The oscillations continued, leading to very large Lagrange multipliers without convergence.

To avoid oscillations in constraint satisfaction and large changes in output values, we need the activation function to have a smooth transition region. This makes small α desirable. Figure 2 shows that a *tanh* function with a smaller α has a smoother transition region. However, when α is too small, the output when training converges can be considerably far away from the desired output of -1 or 1.

To allow the output to be close to 1 or -1 when training converges, consider the output of a neural network for solving the two-spiral problem as a function of input values x and y and hidden-layer outputs $N_1, \cdots, N_m$, where m is the number of hidden units. The output o for a pattern can be expressed as follows:

$$o = \tanh(\alpha(x \times w_x + y \times w_y + \sum_{j=1}^{m} N_j \times w_{N_j})), \quad (5)$$

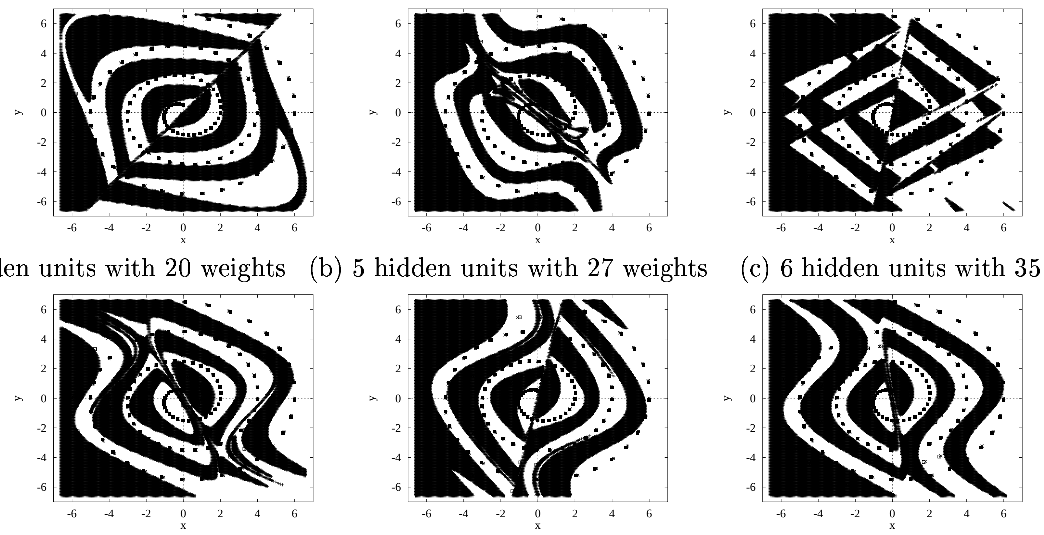
Hence, after a converged network has been found, we can obtain a new converged network by multiplying either the weights or α in (5) by a constant factor. This property gives us the freedom to dynamically adjust the shape of the activation function of the output unit by changing its α in order to control its convergence behavior. Specifically, we decrease (resp. increase) α of the output unit when our run-time statistics shows that the output changes too frequently by more (resp. less) than 1.4 (resp. 0.2). Finally, after training converges, we increase α of the output unit in order to stabilize its value at the extreme state of -1 or 1.

4 Experimental Results

Due to space limitation, we do not present the CSA algorithm we have implemented [5]. Rather, we report only partial experimental results.

Table 1 shows the training results on networks with 4, 5, and 6 hidden units, respectively, with the corresponding classification graphs in Figures 3a-c. We have obtained converged training with 100% classification for all three cases. Our training outperforms *Novel* in term of training speed, success ratio, and number of weights [3].

As a comparison, we have solved the same training problems by SA using unconstrained formulations. Although SA guarantees asymptotic convergence for unconstrained formulations, it has difficulty in solving (1) and often terminated prematurely in our experiments when time was limited. It completed successfully in training a 6-hidden-unit network with 42 weights (including 7 biases), but took much longer time (8.61 hours on average). It did not converge in 12.7 (resp. 86.8) hours of CPU time when applied to train 5 (resp. 4) hidden-unit networks with 33 (resp. 25) weights; the best training results we have obtained is 97.9% (resp. 91.2%) correctness on all training patterns. Finally, we applied sequential quadratic programming (SQP) [4], a fast local-search method for constrained optimization, to solve both the constrained and unconstrained versions. SQP did not find any good solutions because the terrains searched were very rugged and good local minima were all in very deep and narrow regions. It often skipped these regions and failed to find global solutions in unconstrained formulations and feasible solutions in constrained formulations even when its starting point was very close to but not within the local region containing the global minimum.



(a) 4 hidden units with 20 weights (b) 5 hidden units with 27 weights (c) 6 hidden units with 35 weights

(d) 4 hidden units with 19 weights (e) 4 hidden units with 18 weights (f) 4 hidden units with 17 weights

Figure 3. Upper row shows 2-D classification graphs of the three neural networks in Table 1 trained using constrained formulations. Lower row shows classification graphs of 3 4-hidden-unit networks with different number of weights.

After successfully training a 4-hidden-unit network with 20 weights, we tried to train a network with a smaller number of weights. We cannot decrease weights on links connected to the output node because such decreases will be offset by an increase in the gain factor α in the output node. After experimentation, we tried eliminating some weights on links connecting hidden units. This is done by modifying the objective function as follows:

$$E'(w) = E(w) + \sum_{i \in \mathcal{I}} |w_i|^{0.25} \left(\frac{T_0}{T} \right)^2, \qquad (6)$$

where $\mathcal{I}$ is the set of indexes of weights to be eliminated, T_0 is the initial temperature in CSA, and T is current temperature.

By applying this technique to weight w_{17} (connecting the first and fourth hidden units), we were able find a converged network with 19 weights and 100% correctness in training. The weights of the converged network (with $\alpha = 100$ in the hidden layer and 500 in the output node) are as follows:

$$\begin{pmatrix} 0.4524 & 0.8012 & -0.0538 & -0.0202 & 0.5201 \\ 0.0027 & -0.0085 & 0.0176 & 2.9777 & 0.0977 \\ 0.2159 & 0.1584 & 0.9070 & -0.8930 & 0.1084 \\ 0.2970 & 0 & 1.1129 & -0.2234 & -0.6013 \end{pmatrix}.$$

Figure 3d shows the output of the corresponding neural network.

Applying weight elimination to w_{12} and w_{17} (resp. w_{12}, w_{17} and w_{18}), we were able to achieve 99% (resp. 97%) training correctness with 18 (resp. 17) weights. Figure 3e-f show the corresponding outputs.

Our results represent the first successful attempt to train feedforward neural networks to solve the two-spiral problem with less than 20 weights.

References

[1] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

[2] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[3] Y. Shang and B. W. Wah. Global optimization for neural network training. *IEEE Computer*, 29:45–54, March 1996.

[4] P. Spellucci. An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448, 1998.

[5] B. W. Wah and T. Wang. Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming*, pages 461–475, October 1999.

[6] B. W. Wah and Z. Wu. The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming*, pages 28–42, October 1999.