Temperature Cycling on Simulated Annealing for Neural Network Learning

Sergio Ledesma, Miguel Torres, Donato Hernández, Gabriel Aviña, and Guadalupe García

Dept. of Electrical & Computer Engineering. University of Guanajuato. Salamanca, Gto. 36700, México
{selo, mtorres, donato, avina, garciag}@salamanca.ugto.mx

Abstract. Artificial neural networks are used to solve problems that are difficult for humans and computers. Unfortunately, artificial neural network training is time consuming and, because it is a random process, several cold starts are recommended. Neural network training is typically a two step process. First, the network's weights are initialized using a no greedy method to elude local minima. Second, an optimization method (i.e., conjugate gradient learning) is used to quickly find the nearest local minimum. In general, training must be performed to reduce the mean square error computed between the desired output and the actual network output. One common method for network initialization is simulated annealing; it is used to assign good starting values to the network's weights before performing the optimization. The performance of simulated annealing depends strongly on the cooling process. A cooling schedule based on temperature cycling is proposed to improve artificial neural network training. It is shown that temperature cycling reduces training time while decreasing the mean square error on auto-associative neural networks. Three auto-associative problems: The Trifolium, The Cardioid, and The Lemniscate of Bernoulli, are solved using exponential cooling, linear cooling and temperature cycling to verify our results.

1 Introduction

Artificial neural networks (ANNs) can be used to solve complex problems where noise immunity is important. There are two ways to train an ANN: supervised and un-supervised training. Supervised training requires a *training set* where the input and the desired output of the network are provided for several training cases. On the other hand, un-supervised training requires only the input of the network, and the ANN is supposed to classify (separate) the data appropriately.

When the desired output of the network is equal to the input of the network for each training case, the ANN is know as an auto-associative neural network. Alternatively, when the ANN is used to separate objects based on their properties, the ANN is known as a classifier. Lastly, when the ANN is used to map a specific input to a desired output, the network is known as a generic or a mapping neural network. This paper is focused on auto-associate neural networks under supervised learning. To verify our results, the classical curves of Figure 1 were used for training three different auto-associate neural networks. The mean squared error (mse) and the training time were measured to compare our method with a common ANN initialization method.

In order for an ANN to learn, a *training set* is required, that is, for each curve in Figure 1 an appropriate *training set* containing the curve at all possible rotation angles needs to be built. The design of the training is very important because an undesired ANN training effect known as overfitting may result, see [10]. Basically, if the number of training cases is not big enough compare with the number of network's weights, the training process may no have enough cases to adjust appropriately the network's weights, that is, the network's weights may not be good for other data but the *training set*. Thus, an ANN with more neurons (more weights) requires training sets with more training cases, and more training cases means a longer training time. Because of this, training methods that can train ANNs fast and with little error are required. We propose a method that can reduce the training time on auto-associate neural networks.

This paper is organized as follows. In Section 2, background information about ANN training is reviewed. In Section 3, a new cooling schedule for simulated annealing is presented. In Section 4, simulation experiments are performed to verify our results. Finally, Section 5 presents some conclusions and direction for future work.

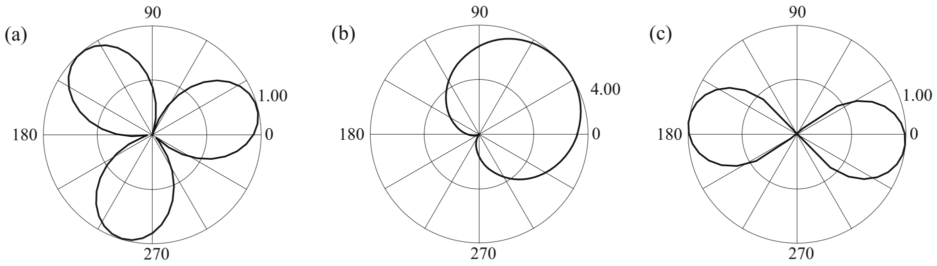

Fig. 1. (a) The Trifolium, (b) The Cardioid, and (c) The Lemniscate of Bernoulli

2 Background Information

2.1 Neural Network Learning

Neural networks are useful when modeling the data is difficult or impossible. For these cases, neural networks can be used to learn from a data set known as the training set; and these networks usually may not have any specific information about the data source.

During a process call training, the network's weights are adjusted until the actual network's output and a desired output are as close as possible, see [13]. During this process, it is valid to re-design the network structure adjusting the number of layers or neurons until a specified performance is obtained. Once the network has been trained, it must be evaluated using a different training set, called the *validation set.* Several heuristic approaches might be used until the *validation set* and the *training set* perform similarly, see [7].

The training process can be divided in two steps: initialization and optimization. The initialization process might be something as simple as assigning initial random values to the network's weights [12] or something much more sophisticate such as: genetic algorithms, simulated annealing, or regression. The optimization process is usually a gradient based algorithm, and it requires a good starting point to be successful. This means that the complete training process depends on both, the initialization process and the optimization process.

2.2 The Method of Simulated Annealing (SA)

At the heart of the method of simulated annealing is an analogy with thermodynamics, specifically with the way that liquids freeze and crystallize, or metals cool and anneal [11]. The method of simulated is an optimization method that tries to imitate the natural annealing process which occurs when a material is heated and then cooled down in a controlled manner. One classic problem solved by SA is the Traveling Salesman Problem, see [4]. Contrary to other optimization methods, SA is a no greedy optimization method and, hence it does not fall easily into local minima. One of the most important practical considerations when implementing the method of SA is to use a high quality random generator, see [6] and [11]. For ANN training, the method of SA perturbs randomly the network's weights following a cooling schedule. Once the network's weights have been perturbed, the performance of the neural network is evaluated using an appropriate *training set.* In general, the cooling schedule may be linear or exponential, and may iterate at each temperature or increase the number of iterations at a specific temperature when improvement occurs, see [1], [2] and [6].

Each time a new solution has been perturbed and evaluated, the oracle makes a decision about whether the new solution is accepted or rejected using the metropolis algorithm [3] and [11]. Some implementations of SA accept a new solution only if the new solution is better than the old one, i.e. it accepts the solution only when the mse decreases; see[6]. However, it is always more efficient to accept the solution even when the new solution has not less error than the previous solution. The probability to accept a new solution was first incorporated into numerical calculations by Metropolis [8] as shown

$$P_a(\Delta E, y) = \begin{cases} e^{-\frac{k\Delta E}{y}}, & \Delta E > 0 \\ 1, & \Delta E \leq 0, \end{cases} \tag{1}$$

where

$$\Delta E = Error_{new\ solution} - Error_{current\ solution}$$

$$y = \text{ Current temperature,}$$

and k is the acceptance constant that depends on the range of the network's weights and the network's input. Thus, at high temperatures, the algorithm may frequently accept a solution even if it is not better than the previous one [3]. During this phase, the algorithm explores in a very wide range looking for an optimal solution, and it is not concerned with the quality of the solution. As the temperature decreases, the algorithm is more selective, and it accepts a new solution only if its error is less than or very similar to the previous solution error following the decision made by the oracle.

In the next section, a cooling schedule will be proposed for implementing SA for ANN's initialization.

3 Proposed Method

One key factor when training multi-layer feed forward neural networks using SA for initialization is to choose an appropriate acceptance constant, k in Equation 1. This acceptance constant depends directly on the temperature range, the training set, and the network's weight allowed values. Failing to take into consideration these factors may affect adversely ANN's learning when using SA or other training methods.

During the preparation of the *training set* is important to scale the data appropriately. In general, it is convenient to restrict the network's input data in the range from -3 to 3 (or less), resulting in network's weights in the range from -10 to 10 or so. Increasing the network's input range more is not recommended for SA, because a wider input range means a wider weights' range and, therefore more combinations for SA, which results on long trainings. To simplify the implementation of our method and without losing generality, let the networks' weights be in the same range as the SA temperature. This is also pretty convenient for monitoring purposes as the current temperature (when cooling or heating) indicates by how much the network's weights are being perturbed.

Once the network's input has been limited and, hence, the network's weights, it is possible to choice an appropriate value for k. This constant affects the probability of acceptance for a given value of ΔE. We found that a value of $k = 1500$, for a cooling schedule using an initial temperature of 15 and a final temperature of 0.015, provides reasonable learning. Where reasonable learning means that the mse does not wander excessively from high and low values due to an excessive probability of acceptance (read excessive heating). Typical cooling schedules start at a high temperature and gradually cool down using different functions until they reach a specified final temperature, see [5]. On the other hand, the proposed method requires the temperature to increase and decrease periodically. Figure 2.a shows a typical cooling schedule (linear cooling), Figure 2.b shows the exponential cooling schedule which is also very popular, Figure 2.c shows the cooling schedule proposed.

Temperature cycling has been previously used by Möbius et al. to solve the Traveling Salesman Problem, see [9]. On the other hand, to describe how temperature cycling must be used for ANN learning consider the finite length series, $x[n]$, defined as

$$x[n+1] = \rho\, x[n], \quad n = 1, 2, 3, \cdots N, \tag{2}$$

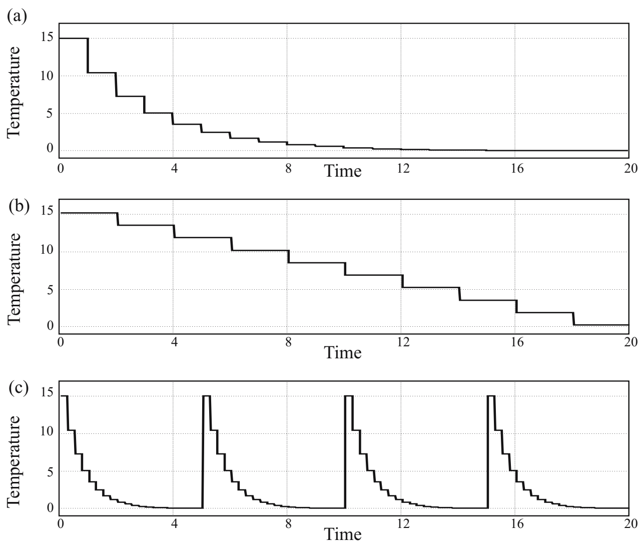

Fig. 2. (a) Exponential cooling, (b) Linear cooling, (b) Temperature cycling

where N is the number of temperatures, $x[1]$ is the initial temperature, $x[N]$ is the final temperature, and ρ is a cooling constant defined as

$$\rho = e^{\log(\frac{(N-1)x[N]}{x[1]})}. \tag{3}$$

The cooling schedule for temperature cycling is defined as

$$y[n] = \sum_{m=0}^{M-1} x[n - mN], \tag{4}$$

where $y[n]$ is the SA temperature, M is the number of cycles before starting the optimization process. Additionally, temperature cycling requires keeping the number of iterations at each temperature to a relatively low value, i.e., 10, 20 or 30 are good values. Note that if ANN's training is performed using 100 iterations per temperature or more, the benefit gained for using temperature cycling is lost. Iterating too much at each temperature can be bad for temperature cycling because the solution may fall too much and it will be difficult to escape from this minimum. Additionally, the network's weights must be updated using the recursive equation shown

$$w_{i,j}[n+1] = \gamma(1-\lambda)\ w_{i,j}[n] + \lambda\ u[n] - \frac{1}{2}\lambda, \tag{5}$$

where $w_{i,j}$ is the network's weight connecting the j neuron with the i neuron in the next layer, λ is the perturbation ratio defined as

$$\lambda = \frac{y[n]}{x[1]}, \tag{6}$$

$u \in [0, 1)$ is a uniformly distributed random variable, and γ depends on the weights' range. Typical values for this constant are 20 or 30 depending on the network's input. Observe that for a current temperature $y[n]$ close to the initial temperature x[1], λ takes values close to 1 and the network's weights are violently perturbed. As the temperature decreases the network's weights wandered randomly around a fixed value. Observe also that Equation 5 properly combines the perturbation and the current weight value. Other approaches for SA perturb the network's weights, and then clip the perturbed value to keep the network's weights within a specified range, this is incorrect because some information is lost during the clipping process.

4 Simulation Results

For simulation purposes three different multi-layer feed forward neural networks were designed and trained. The first neural network was designed to learn the Trifolium. The resulting auto-associative neural network required 64 inputs, 64 outputs and 4 neurons on the hidden layer. The second neural network was designed to learn the Cardioid, and it required 64 inputs, 64 outputs and 4 neurons on the hidden layer. The third neural network was designed to learn the Lemniscate of Bernoulli, and it required 64 inputs, 64 outputs and 5 neurons on the hidden layer. Because of the number of network's outputs, the network's training may be slow, if the method of Levenberg-Marquardt for optimization is used. Thus, the method of Conjugate-Gradient method was used. It is important to note that the training process was monitored during the initialization and optimization phases separately on an mse-time plot, and this will be explained next.

4.1 Learning

To illustrate how temperature cycling affects neural network learning. The networks' mse was recorded at equally spaced time intervals during the training process for each of the neural networks designed previously. In Sub-Section 4.2, it will be indicated the number of temperatures, iterations as well as the number of cycles used during the training of each of the three ANNs. Figure 3 shows the network's mse as a function of time for the Trifolium; from this figure, it can be seen that during the initialization phase the method of temperature cycling minimized drastically the mse. Observe that once the initialization phase was completed, the training process switched to the optimization phase. For the exponential and linear cooling schedules the initialization process did not offer a good initial solution, and the optimization process was not able to find the global minimum, and after 200 epochs, they were stuck without hope. On the other hand, the network trained using temperature cycling reached an acceptable mse value during the initialization phase, hence the optimization process attained an mse of 1×10^{-5} by the epoch 180.

Figure 4 and Figure 5 show the networks' mse during the training process for the Cardioid and the Lemniscate of Bernoulli, respectively. As it can be seen

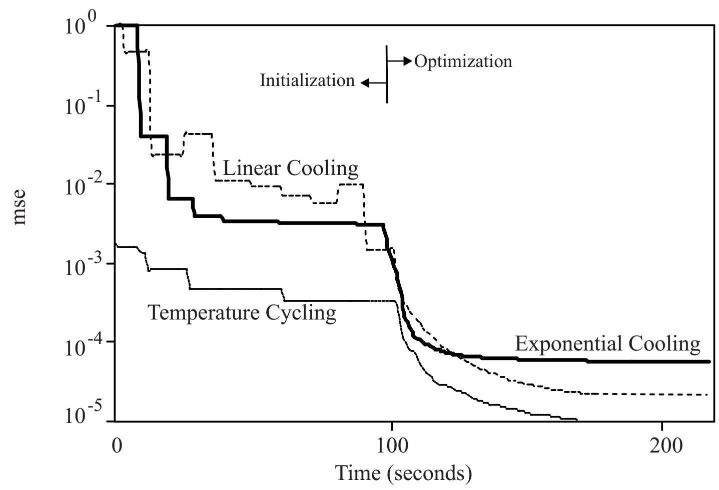

Fig. 3. Mean squared error for exponential cooling, linear cooling and temperature cycling while training an auto-associative neural network for learning of the Trifolium

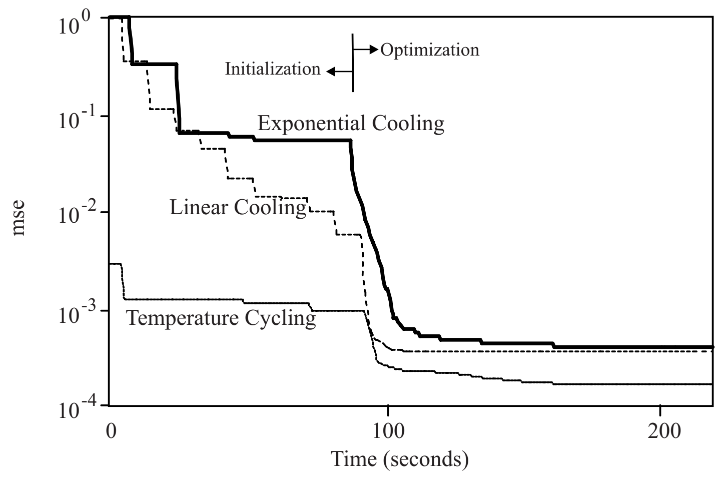

Fig. 4. Mean squared error for exponential cooling, linear cooling and temperature cycling while training an auto-associative neural network for learning of the Cardioid

from these figures, the networks trained using temperature cycling outperformed those trained by exponential or linear cooling. Before leaving this section, it is important to mention that the success of temperature cycling requires performing only a few iterations at each temperature. Next, the network's ability to reduce noise will be discussed and analyzed.

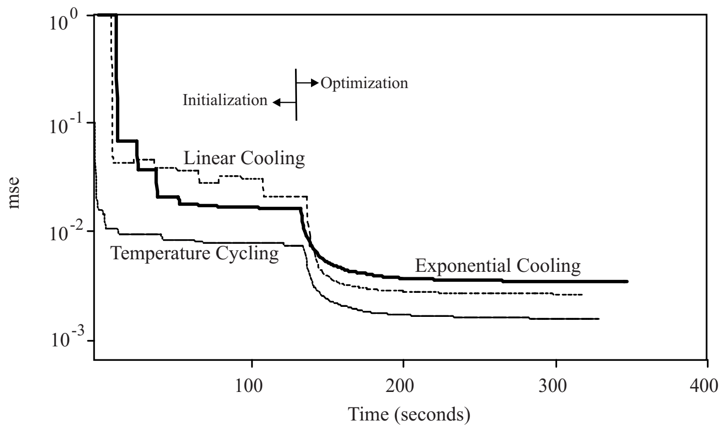

Fig. 5. Mean squared error for exponential cooling, linear cooling and temperature cycling while training an auto-associative neural network for learning of the Lemniscate of Bernoulli

4.2 Noise Reduction

Once an appropriate network to learn the Trifolium was designed, it was trained using a set of 780 different training cases; they included the Trifolium shape at several rotation angles using a resolution of 64 points. The number of training cases for this *training set* was computed based on the numbers of network's weights to be adjusted and avoid overfitting, see [10]. Additionally, a Trifolium, with a random rotation angle, was contaminated with noise to test the network and its training. Figure 6.a shows the noisy Trifolium sample. First, an auto-associate neural network was trained using exponential cooling: 10 temperatures and 1000 iterations per temperature. Once the training was completed, the network was excited using the noisy sample of Figure 6.a. The output of this network is shown in Figure 6.b. The same experiment was repeated using linear cooling; Figure 6.c shows the results obtained on this case. Last, another neural network, with the same structure as the used for exponential cooling, was training using temperature cycling: 10 temperatures, 10 iterations per temperature and 100 cooling cycles; note that the same number of iterations was used for all experiments, $10,000$ iterations. Figure 6.d shows the output of the neural network trained using temperature cycling; as it can be seen from this figure, the performance of the network trained using temperature cycling is much better than the performance of the neural networks trained using exponential or linear cooling.

Figure 7.a shows a noisy cardioid. Figure 7.b shows the output of a neural network trained using exponential cooling, Figure 7.c shows the output of a neural network trained using linear cooling and Figure 7.d shows the the output of neural network trained using temperature cycling. Figure 8 shows the results for the Lemniscate of Bernoulli. For all cases, it can be seen that those networks trained using temperature cycling outperform those networks trained using regular cooling for noise reduction.

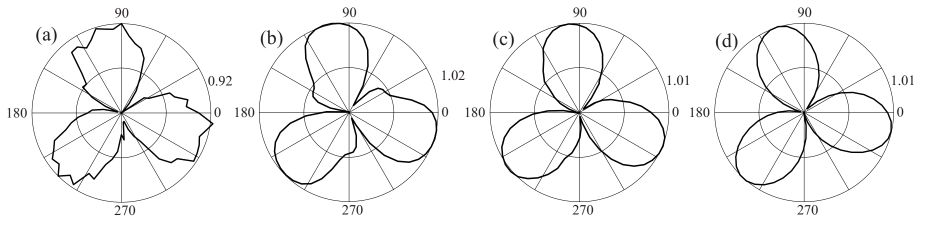

Fig. 6. (a) Noisy Trifolium sample, (b) Noise reduction of a neural network trained using exponential cooling, (c) Noise reduction of a neural network trained using linear cooling, (d) Noise reduction of a neural network trained using temperature cycling

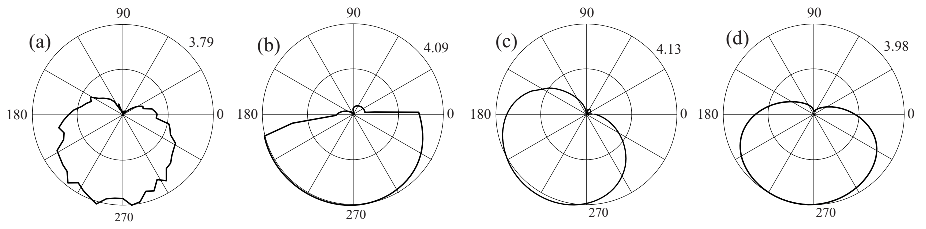

Fig. 7. (a) Noisy Cardioid sample, (b) Noise reduction of a neural network trained using exponential cooling, (c) Noise reduction of a neural network trained using linear cooling, (d) Noise reduction of a neural network trained using temperature cycling

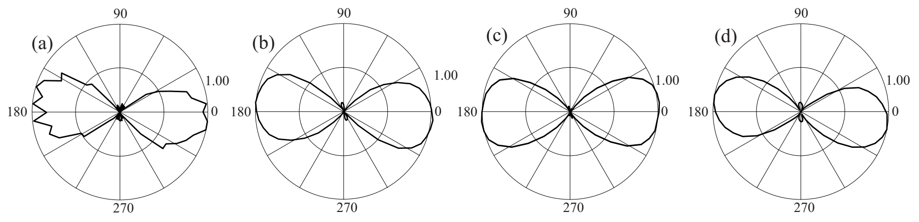

Fig. 8. (a) Noisy Lemniscate of Bernoulli sample, (b) Noise reduction of a neural network trained using exponential cooling, (c) Noise reduction of a neural network trained using linear cooling, (d) Noise reduction of a neural network trained using temperature cycling

5 Summary

There is not yet enough experience with the method of simulated annealing to say definitively what its future place among optimization methods will be. It is not greedy, in the sense that it is not easily fooled by the quick payoff achieved by falling into unfavorable local minima [11].

For auto-associate multi-layer feed forward neural networks (pattern learning) the method of simulated annealing offers great performance when temperature cycling is used as long as few iterations are use at each temperature. Too many iterations at each temperature prevents the method from continually reducing the mse when training auto-associate neural networks because at each temperature cycle the method might have fallen too much, and it is unable to escape from a false minimum.

Experimental results were obtained using some classical closed curves. The Trifolium, The Cardioid, and The Lemniscate of Bernoulli were used for training three different ANNs. It was shown that temperature cycling reduces both the mse and the training time when compared with exponential cooling. Additionally, these curves were contaminated with noise, then ANN's were used for noise reduction; in general, those networks trained using temperature cycling provided better noise reduction capabilities than those networks trained using exponential or linear cooling.

Classification or generic neural networks do not seem to benefit from temperature cycling when SA is used for initialization. Future work includes the extension of this method (temperature cycling or other cooling schedules) for classification or generic neural network training.

References

1. Abramson, D., Dang, H., Krishnamoorthy, M.: Simulated Annealing Cooling Schedules for the School Timetabling Problem. Asia-Pacific Journal of Operational Research , 1–22 (1999)
2. Huang, M., Romeo, F., Sangiovanni-Vincentelli, A.: An Efficient General Cooling Schedule for Simulated Annealing. In: ICCAD. Proc. of the IEEE International Conf. on Computer Aided Design, pp. 381–384 (1986)
3. Jones, M.T.: AI Application Programming, Charles River Media, 2nd edn. pp. 49–67 (2005)
4. Johnson, D., McGeoch, L.: The Traveling Salesman Problem: A Case Study in Local Optimization. In: Aarts, E.H., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, Wiley and Sons, Chichester
5. Luke, B.T.: Simulated Annealing Cooling Schedules, (June 1, 2007) available online at `http://members.aol.com/btluke/simanf1.htm,accessed`
6. Masters, T.: Practical Neural Network Recipes in C++, pp. 118–134. Academic Press, London (1993)
7. Masters, T.: Advanced Algorithms for Neural Networks, pp. 135–156. John Wiley & Sons Inc, Chichester (1995)
8. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, E.: Journal of Chemical Physics 21, 1087–1092 (1953)
9. Möbius, A., Neklioudov, A., Díaz-Sánchez, A., Hoffmann, K.H., Fachat, A., Schreiber, M.: Optimization by Thermal Cycling. Physical Review 79(22) (1997)
10. Nilsson, N.J.: Artificial Intelligence: A New Synthesis, pp. 37–58. Morgan Kaufmann Publishers, San Francisco (1998)
11. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C++: The Art of Scientific Computing, 2nd edn. pp. 448–459. Cambridge University Press, Cambridge (2002)

12. Reed, R.D., Marks II, R.J.: Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, pp. 97–112. The MIT Press, Cambridge (1999)
13. Russel, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Englewood Cliffs (2002)