Training Artificial Neural Networks: Backpropagation via Nonlinear Optimization

Jadranka Skorin-Kapov\(^1\) and K. Wendy Tang\(^2\)

\(^1\) W.A. Harriman School for Management and Policy, State University of New York at Stony Brook, Stony Brook, USA
\(^2\) Department of Electrical and Computer Engineering, State University of New York at Stony Brook, Stony Brook, USA

In this paper we explore different strategies to guide backpropagation algorithm used for training artificial neural networks. Two different variants of steepest descent-based backpropagation algorithm, and four different variants of conjugate gradient algorithm are tested. The variants differ whether or not the time component is used, and whether or not additional gradient information is utilized during one-dimensional optimization. Testing is performed on randomly generated data as well as on some benchmark data regarding energy prediction. Based on our test results, it appears that the most promising backpropagation strategy is to initially use steepest descent algorithm, and then continue with conjugate gradient algorithm. The backpropagation through time strategy combined with conjugate gradients appears to be promising as well.

**Keywords:** Artificial intelligence: backpropagation in neural networks, Nonlinear unconstrained programming: conjugate gradient method, steepest descent method

1. Introduction

One of the key issues when designing a particular neural network is to calculate proper weights for neuronal activities. These are obtained from the training process applied to the given neural network. To that end, a training sample is provided, i.e. a sample of observations consisting of inputs and their respective outputs. The observations are ‘fed’ to the network. In the training process the algorithm is used to calculate neuronal weights, so that the squared error between the calculated outputs and observed outputs from the training set is minimized. Such an approach gives rise to an unconstrained nonlinear minimization problem, which can be solved with a number of existing methods. First order methods (such as steepest descent) sometimes lack fast convergence, while second order methods (e.g. Newton’s method) are computationally expensive \([3, 4]\). Johansson et al. \([6]\) compare the use of conjugate gradient method in backpropagation with conventional backpropagation and steepest descent. (Conventional backpropagation refers to steepest descent with constant learning rate, i.e. step size, instead of using line search to get step sizes for each iteration.) For the data tested (three, four and five bit parity problems) the authors report that conjugate gradient backpropagation is much faster than conventional backpropagation. Mangasarian \([7]\) discusses the role of mathematical programming, particularly linear programming, in training neural networks, and demonstrates it on the system developed for breast cancer diagnosis.

Our study is motivated by neuro-control applications in complex systems. As neural networks are originally inspired by the human brain, one of the ultimate goals of neural network research is to demonstrate possible “brain-like” intelligence in the control area \([15]\). To this end, Werbos has summarized the pros and cons of five major neuro-control strategies \([16]\). Of these strategies, backpropagation of utility is capable of controlling a complex system to maximize a utility function. Tang and Pingle \([11]\) have demonstrated that the backpropagation of utility algorithm is capable of modeling the dynamics of a one-dimensional planar robot and eventually providing proper control signal to drive the manipulator to follow a prescribed trajectory.
However, the success of the algorithm hinges upon sufficient training of a neural network to emulate the dynamic system.

In this paper we show that combined steepest descent and conjugate gradient methods offer advantages regarding neural network training over both of those methods if taken separately. Our task is to decide on the most appropriate strategy utilizing a combination of steepest descent and conjugate gradients for solving some randomly generated, as well as some benchmark data sets. In addition, we stress the importance of the above neural network training over both of those methods if taken separately. Our task is to decide on the most appropriate strategy utilizing a combination of steepest descent and conjugate gradient methods. An efficient algorithm should be stable and should converge as close as possible to its desired (or true) set of outputs $Y_j(t)$, $j = 1, \ldots, n$, which is as close as possible to its desired (or true) set of outputs $Y_j(t)$, $j = 1, \ldots, n$. In the sequel, $W_t$ will denote the weight of network node $l$, and $s(z) = 1/(1 + e^{-z})$ will denote the sigmoidal transfer function.

In the basic backpropagation algorithm, the feedback equations used to calculate the error function are given as follows [14]:

$$
\text{hid1}_{h1}(t) = \sum_{i=1}^{m} W_{(h1-1)sm+i} \cdot X_i(t),
$$

$$
h1 = 1, \ldots, H1; \ t = 1, \ldots, T \tag{1}
$$

$$
z1_{h1}(t) = s(hid1_{h1}(t)),
$$

$$
h1 = 1, \ldots, H1; \ t = 1, \ldots, T \tag{2}
$$

$$
\text{hid2}_{h2}(t) = \sum_{h1=1}^{H1} W_{H1sm+(h2-1)sh1} \cdot z1_{h1}(t),
$$

$$
h2 = 1, \ldots, H2; \ t = 1, \ldots, T \tag{3}
$$

$$
z2_{h2}(t) = s(hid2_{h2}(t)),
$$

$$
h2 = 1, \ldots, H2; \ t = 1, \ldots, T \tag{4}
$$

$$
\text{net}_j(t) = \sum_{h2=1}^{H2} W_{H1sm+H1sh2+(j-1)sh2+h2} \cdot z2_{h2}(t),
$$

$$
j = 1, \ldots, n; \ t = 1, \ldots, T \tag{5}
$$

$$
\overline{Y}_j(t) = s(\text{net}_j(t)),
$$

$$
j = 1, \ldots, n; \ t = 1, \ldots, T \tag{6}
$$

The objective is to minimize the weighted squared sum of errors, i.e.
In order to enhance the basic backpropagation algorithm, Werbos [13, 14] proposed the backpropagation through time algorithm, which incorporates limited memory from past time periods. This is achieved via a second set of weights $W'$, added to each hidden and output node. (See, for example, Tang and Chen [10])

### 2.1. Steepest Descent Versus Conjugate Gradient Algorithm

When minimizing an objective function, we want to find directions of steepest decrease in functional values. One can pose the following problem: For all directions $y$ with some bounded length, find the direction of steepest descent in functional value of $f$ at a given point $x^0$, for which $\nabla f(x^0) \neq 0$. This is a nonlinear problem and its solution states that the steepest descent is the direction of the negative gradient. The steepest descent method is an iterative method moving from an initial point through a sequence of points in directions of negative gradients. At each iteration the step size is either obtained via one-dimensional optimization, or it is given in advance as a parameter (in the neural network vocabulary, this strategy corresponds to the basic backpropagation). For a quadratic function, a convergence of steepest descent method depends highly on the condition number of the quadratic matrix, which in turn depends on the difference between the smallest and largest eigenvalue of the matrix: if there is a big difference, the contours of the quadratic function are cigar shaped ellipsoids around the minimal point. This results in poorer convergence, especially closer to the minimal point. (For a non-quadratic function the idea is to use the Hessian of the objective at the solution point as if it were the quadratic matrix of a quadratic problem.)

Tang and Chen [10] studied backpropagation strategies where unconstrained minimization was performed via steepest descent algorithm. They concluded that over an extended number of iterations “the backpropagation through time algorithm is more robust and has a faster convergence rate”. In this paper we study backpropagation using conjugate gradient method for unconstrained minimization, as well as a hybrid
between steepest descent and conjugate gradient. To that end, we briefly summarize the main idea behind a conjugate gradient algorithm.

Suppose we have \( f : \mathbb{R}^n \rightarrow \mathbb{R} \), where \( f \) is a twice continuously differentiable function. When minimizing a function, we want to have a descent method (like the steepest descent), but also a method that converges fast, say in a finite number of steps (unlike the steepest descent), when applied to the quadratic function. (It is reasonable to assume that a nonlinear function can be reasonably well approximated by a quadratic function in the neighborhood of a minimal point.) The method of conjugate gradients combines descent property and a finite convergence for the quadratic case. The method proceeds iteratively as follows. Start with \( x^0 \). At iteration \( k \) let the new point \( x^k \) be obtained as 
\[
x^k = x^{k-1} + \alpha_k z^k,
\]
where \( z^k \) is the direction of the move, and \( \alpha_k \) is the step size. For a given direction \( z^k \), \( \alpha_k \) is obtained by minimizing \( f \) along \( z^k \). Namely, let \( F(\alpha_k) = f(x^{k-1} + \alpha_k z^k) \), and let \( \alpha_k^* \) denote the minimum of \( F \). Therefore
\[
\frac{\partial F(\alpha_k^*)}{\partial \alpha_k} = z^k^T \nabla f(x^{k-1} + \alpha_k^* z^k) = 0.
\]
In the case of a quadratic function, we can get an explicit representation of \( \alpha_k^* \). Let \( f(x) = a + b^T x + \frac{1}{2} x^T Q x \), where \( Q \) is a symmetric positive definite matrix. Then, \( \nabla f(x^k) = b^T + Q x^k \), so
\[
\alpha_k^* = \frac{-z^k^T \nabla f(x^{k-1})}{z^k^T Q z^k}.
\]

Conjugate gradients are extensions of conjugate directions for differentiable functions. The conjugate gradient algorithm due to Fletcher and Reeves [2] runs as follows:

1. Select \( x^0 \in \mathbb{R}^n \), the initial or starting solution.
2. Evaluate \( \nabla f(x^0) \), and let \( z^1 = -\nabla f(x^0) \).
3. Generate \( x^1, x^2, \ldots, x^n \) by minimizing \( f \) along the directions \( z^1, z^2, \ldots, z^n \), respectively, where \( x^{k+1} = x^k + \alpha_k^* z^{k+1} \), and \( \alpha_k^* \) minimizes \( f(x^k + \alpha_k z^{k+1}) \). (It is found, e.g., by line search). Define
\[
z^{k+1} = -\nabla f(x^k) + \frac{\nabla f(x^k)^T \nabla f(x^k)}{\nabla f(x^{k-1})^T \nabla f(x^{k-1})} z^k.
\]

The first part of \( z^{k+1} \) corresponds to steepest descent, and the second part is the "modification" introduced to speed up the convergence, based on conjugate directions.

If \( \nabla f(x^k)^T \nabla f(x^k) \leq \epsilon \), where \( \epsilon \) is a small tolerance, the algorithm will stop, since it will assume \( \nabla f(x^k) = 0 \). Otherwise, for quadratic \( f \) it will stop after \( n \) iterations. If the function is not quadratic, the convergence in \( n \) steps is not guaranteed. Polak-Ribière-Polyak (see e.g. Avriel [1]) proposed the version of conjugate gradient algorithm where \( z^{k+1} = -\nabla f(x^k) + \beta_k z^k \), \( k = 1, 2, \ldots \) and
\[
\beta_k = \frac{[\nabla f(x^k) - \nabla f(x^{k-1})]^T \nabla f(x^k)}{\nabla f(x^{k-1})^T \nabla f(x^{k-1})}.
\]

Global convergence holds if the method is periodically restarted. For example, if every \( n \) steps we choose \( z^{k+1} = -\nabla f(z^n) \), \( k = n, 2n, 3n, \ldots \) (which corresponds to the steepest descent direction). The convergence of conjugate gradient algorithm strongly depends on the initial point. If relatively 'far away' from the minimum, the approximation with a quadratic function will not be accurate (which is a prerequisite for the success of conjugate gradient strategy), and the method will be incapable to get a good convergence. Therefore, it seems appropriate to use a combination of the two gradient approaches: first, in a limited number of iterations use steepest descent, and then switch to conjugate gradient to speed up the convergence towards the minimal point.

**Definition** Two vectors \( x \in \mathbb{R}^n \), \( y \in \mathbb{R}^n \) are said to be conjugate directions with respect to an \( n \times n \) symmetric positive definite matrix \( A \) if \( x^T A y = 0 \). (If \( A \equiv I \), conjugate directions are standard orthogonal directions.)
2.2. Conjugate Gradient Algorithm Applied to Backpropagation in Neural Networks

First, the feedforward equations are used to calculate the objective function (a differentiable function of weights) to be minimized. Then, the feedback equations lead to the gradient calculation. Those subroutines are used in the conjugate gradient algorithm employing Polak-Ribiére-Polyak (PRP) strategy. One-dimensional subproblems (to get the right step size in each iteration) employ line search minimization. One-dimensional minimization was performed using parabolic interpolation and Brent’s method [see, e.g. Vetterling et al. [12]]. Brent’s method can be implemented so that it optionally utilizes the derivative information available for one-dimensional problems. In the next section we compare different backpropagation strategies.

3. Computational Results

Our approach of combining steepest descent and conjugate gradients was first tested on randomly generated data for the $sinus$ function. The network has one input node ($m = 1$), one output node ($n = 1$), and 20 hidden nodes arranged in two hidden layers each having 10 nodes. We generated twenty pairs ($T = 20$) of $(X_i, Y_i = sin(X_i))$ points and ‘fed’ it to the network. By adjusting the weights, the task was to minimize the sum of squared errors between values $Y_i$ (which the network would output) and the true values $Y_i$. The program was written in C and conjugate gradient routines from Vetterling et al. [12] were used. The testing was performed on a Sun SPARC 2 workstation.

The following versions of backpropagation strategy were tested:

1. BBSD (Basic backpropagation with steepest descent);
2. BTSD (Backpropagation through time with steepest descent);
3. BBCG (Basic backpropagation with conjugate gradients);
4. BTCG (Backpropagation through time with conjugate gradients);
5. BBCGD (Basic backpropagation with conjugate gradients using derivatives for one-dimensional optimization);
6. BTCGD (Backpropagation through time with conjugate gradients using derivatives for one-dimensional optimization);

![Fig. 1. Sine Data with Random Initial Weights](image)
BBSD and BTSD strategies were implemented using the same parameters, and delta-bar-delta rule as in Tang and Chen [10], so we will not elaborate it further. In the BTSD algorithm, the time component was introduced after 100 iterations of BBSD, since the preliminary testing suggested that a delay in introducing the time component results in more robust algorithm. (One iteration is one run over the complete training set.)

We first ran 2000 iterations of each of the six different strategies with randomly generated initial weights (from the interval [-0.1,0.1]). The results are displayed in Figure 1. The versions BTCG and BTCGD did not show convergence, hence we omitted them from display. The tabular displays accompanying the figures and showing computational times are presented in the Appendix.

If the criterion $\nabla f(x^k)^T \nabla f(x^k) \leq \epsilon$ is achieved, the conjugate gradient algorithm stops. In our implementation $\epsilon = 10e - 8$ is the prescribed tolerance. From the results, it appears that for a small number of iterations BBSD works better than other variants. (For an extended number of iterations, BTSD outperforms the basic backpropagation, but the convergence overall is slow compared with the convergence obtained when conjugate gradients are introduced.) Therefore, we decided to save in a file the weights obtained after 2000 iterations of BBSD, and use them as initial weights for the next round of testing. The results are displayed in Figure 2.

It is clear that, closer to the minimal point, the conjugate gradient strategy outperforms steepest descent. It is more computationally intensive and requires more computing time, but the quality of the solution is one order of magnitude better. (An iteration of steepest descent requires approximately 0.03 seconds, while an iteration of conjugate gradient method requires approximately 0.14 seconds.) The time component introduced in backpropagation (backpropagation through time) adds to solution quality, without adding extra computational time. Utilization of derivatives in one-dimensional optimizations (to obtain the best step size for moving along the given direction) adds to expensiveness of the algorithm, with marginal increase in the quality of the solution. (An iteration of conjugate gradient with derivatives requires approximately 0.22 seconds.) On the basis of tested problems, we therefore suggest to use the backpropagation through time conjugate gradient variant without derivatives, i.e. the version BTCG.

Additional testing was performed on a benchmark data set for the prediction of energy con-
Fig. 3. Energy Prediction Data with Random Initial Weights

This dataset was created based on a benchmark problem of "The Great Energy Predictor Shootout – the first building data analysis and prediction problem" contest, organized in 1993 for the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRSE) meeting in Denver, Colorado. The contest task was to predict the hourly consumption of electrical energy, hot water, and cold water based on the date, time of day, outside temperature, outside air humidity, solar radiation, and wind speed. Complete hourly data for 88 consecutive days were given for training.

The inputs to the neural networks including date, hour, weather-related parameters (temperature, humidity, solar radiation and wind speed) are encoded into $m = 14$ parameters. There are $n = 3$ outputs, the predicted hourly consumption of electrical energy, hot water and cold water. The dataset consists of 88 days of hourly data. With the exception of the first day (resp., the last day), which only has 22 hours (resp., 18 hours) of data, all 24-hours are available, hence a total of $T = 2104$ samples. We initially ran 200 iterations of each of the algorithms, and the results are displayed in Figure 3.

Backpropagation through time variants was introduced after 10 iterations. The iterations of steepest descent variants require less cpu time than the iterations of conjugate gradients versions, but the convergence is slower. Based on the experience with randomly generated sinus data, we saved the weights obtained after 200 iterations of steepest descent algorithm BBSD. In the next round of testing, we ran 100 iterations of the algorithms starting with weights generated from BBSD. The results are displayed in Figure 4, with better presentation of conjugate gradient variants repeated on a larger scale in Figure 5.

Similarly as in the case of randomly generated data, it seems that the best strategy is to initially run backpropagation based on steep-

---

1 The database is available through the internet via ftp to the Neural Bench archive at Carnegie Mellon University with internet address “ftp.cs.cmu.edu”, directory “/afs/cs/project/connect/bench/contrib/prechelt”
Fig. 4. Energy Prediction Data with Initial Weights after 200 Iterations of BBSD

Fig. 5. Energy Prediction Data with Initial Weights after 2000 Iterations of BBSD

est descent, followed by the conjugate gradient version of backpropagation. In this way, comparable results can be obtained much faster, due to ‘cheaper’ steepest descent initial iterations. When the search comes closer to a minimal point, the conjugate gradient strategy provides much better convergence. The conjugate gradient versions which employ derivatives for one-dimensional optimization (i.e. algorithms BBCGD and BTCGD) take much more time than the basic conjugate gradient versions, without improving the convergence. Regarding the convergence and speed, the backpropagation through time strategy combined with conjugate gradients provides similar results as the basic backpropagation. Hence, as in the case of randomly generated sinus data, it seems that the best backpropagation strategy is to initially run steepest descent, and then conjugate gradient algorithm.

In sum, from this study we confirmed our intuition that backpropagation with steepest decent
combined with conjugate gradient offers a powerful tool for training neural network.

4. Conclusions and Directions for Further Research

In this paper we studied the benefits of combining steepest descent and conjugate gradient algorithms as strategies to guide backpropagation when training artificial neural networks. A neural network architecture with 2 hidden layers (each with 10 hidden nodes) was trained on randomly generated data for the sinus function, as well as some benchmark data corresponding to energy prediction in applications from construction industry. The results suggest that with the combined strategy (start with steepest descent, then switch to conjugate gradient) one is able to achieve better accuracy in training than it would be achieved with separate strategies.

The next task is to utilize the gained knowledge in a more complex learning systems, and in neural networks with more complex structure (in terms of input, output, and hidden nodes). From this study, we believe that steepest descent combined with conjugate gradient will offer a fast alternative in training such neural network.

4.1. Appendix: Tabulated Data

<table>
<thead>
<tr>
<th>iter</th>
<th>BBSD obj. value</th>
<th>time (sec)</th>
<th>BTSD obj. value</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>5.51918e-02</td>
<td>5.89</td>
<td>5.84926e-02</td>
<td>6.15</td>
</tr>
<tr>
<td>400</td>
<td>4.20617e-02</td>
<td>11.86</td>
<td>3.77279e-02</td>
<td>12.48</td>
</tr>
<tr>
<td>600</td>
<td>4.09564e-02</td>
<td>17.77</td>
<td>3.67574e-02</td>
<td>18.83</td>
</tr>
<tr>
<td>800</td>
<td>4.02316e-02</td>
<td>23.73</td>
<td>3.40680e-02</td>
<td>25.16</td>
</tr>
<tr>
<td>1000</td>
<td>3.84569e-02</td>
<td>29.70</td>
<td>1.47054e-02</td>
<td>31.48</td>
</tr>
<tr>
<td>1200</td>
<td>3.49867e-02</td>
<td>35.66</td>
<td>4.61972e-03</td>
<td>37.81</td>
</tr>
<tr>
<td>1400</td>
<td>3.31051e-02</td>
<td>41.63</td>
<td>2.43466e-03</td>
<td>44.11</td>
</tr>
<tr>
<td>1600</td>
<td>8.13298e-03</td>
<td>47.60</td>
<td>1.87984e-03</td>
<td>50.41</td>
</tr>
<tr>
<td>1800</td>
<td>5.96908e-04</td>
<td>53.57</td>
<td>1.67619e-03</td>
<td>56.68</td>
</tr>
<tr>
<td>2000</td>
<td>4.32921e-04</td>
<td>59.54</td>
<td>1.60801e-03</td>
<td>62.98</td>
</tr>
</tbody>
</table>

*Table 1.* Steepest descent strategy with random initial weights applied to randomly generated sinus data with $m = 1, n = 1, T = 20$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCG obj. value</th>
<th>time (sec)</th>
<th>BTCG obj. value</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1.66632e-01</td>
<td>30.81</td>
<td>(iter 195) 1.68171e-01</td>
<td>32.64</td>
</tr>
<tr>
<td>400</td>
<td>1.63184e-01</td>
<td>58.53</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>600</td>
<td>1.63490e-01</td>
<td>85.73</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>800</td>
<td>1.63240e-01</td>
<td>113.07</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1000</td>
<td>1.63143e-01</td>
<td>141.18</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1200</td>
<td>1.63007e-01</td>
<td>169.79</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1400</td>
<td>1.62951e-01</td>
<td>197.63</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1600</td>
<td>1.62914e-01</td>
<td>225.36</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1800</td>
<td>1.62874e-01</td>
<td>253.18</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2000</td>
<td>1.62846e-01</td>
<td>282.23</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 2.* Conjugate gradient strategy with random initial weights applied to randomly generated sinus data with $m = 1, n = 1, T = 20$
<table>
<thead>
<tr>
<th>iter</th>
<th>BBGD</th>
<th>obj. value</th>
<th>time (sec)</th>
<th>BTCGD</th>
<th>obj. value</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>2.99806e-01</td>
<td>45.91</td>
<td></td>
<td></td>
<td>2.80799e-01</td>
<td>41.37</td>
</tr>
<tr>
<td>400</td>
<td>1.67681e-01</td>
<td>90.59</td>
<td></td>
<td></td>
<td>1.66815e-01</td>
<td>129.10</td>
</tr>
<tr>
<td>600</td>
<td>1.64857e-01</td>
<td>169.27</td>
<td></td>
<td></td>
<td>3.79787e-02</td>
<td>210.03</td>
</tr>
<tr>
<td>800</td>
<td>3.79787e-02</td>
<td>247.07</td>
<td></td>
<td></td>
<td>3.78247e-02</td>
<td>247.07</td>
</tr>
<tr>
<td>1000</td>
<td>3.79787e-02</td>
<td>283.25</td>
<td></td>
<td></td>
<td>3.76944e-02</td>
<td>302.24</td>
</tr>
<tr>
<td>1200</td>
<td>3.79787e-02</td>
<td>356.84</td>
<td></td>
<td></td>
<td>3.74953e-02</td>
<td>356.84</td>
</tr>
<tr>
<td>1400</td>
<td>3.79787e-02</td>
<td>393.61</td>
<td></td>
<td></td>
<td>3.72957e-02</td>
<td>393.61</td>
</tr>
</tbody>
</table>

Table 3. Conjugate gradient using one-dimensional derivatives strategy with random initial weights applied to randomly generated sinus data with $m = 1$, $n = 1$, $T = 20$

<table>
<thead>
<tr>
<th>iter</th>
<th>BSSD</th>
<th>obj. value</th>
<th>time (sec)</th>
<th>BTSD</th>
<th>obj. value</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>4.08008e-04</td>
<td>5.91</td>
<td></td>
<td>4.00017e-02</td>
<td>6.23</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>3.83492e-04</td>
<td>11.88</td>
<td></td>
<td>3.44060e-02</td>
<td>12.52</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>3.59130e-04</td>
<td>17.90</td>
<td></td>
<td>2.85534e-02</td>
<td>18.82</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>3.38109e-04</td>
<td>23.87</td>
<td></td>
<td>1.04022e-02</td>
<td>25.13</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>3.21317e-04</td>
<td>29.84</td>
<td></td>
<td>2.84873e-03</td>
<td>31.67</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>3.07937e-04</td>
<td>35.81</td>
<td></td>
<td>1.49020e-03</td>
<td>38.42</td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>2.97030e-04</td>
<td>41.78</td>
<td></td>
<td>1.16497e-03</td>
<td>45.25</td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>2.88016e-04</td>
<td>47.76</td>
<td></td>
<td>9.94031e-04</td>
<td>51.61</td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>2.80147e-04</td>
<td>53.74</td>
<td></td>
<td>8.99585e-04</td>
<td>57.93</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>2.73437e-04</td>
<td>59.71</td>
<td></td>
<td>8.31274e-04</td>
<td>64.38</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Steepest descent strategy with initial weights obtained after 2000 iterations of BBSD algorithm, applied to randomly generated sinus data with $m = 1$, $n = 1$, $T = 20$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCG</th>
<th>obj. value</th>
<th>time (sec)</th>
<th>BTCG</th>
<th>obj. value</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>2.28338e-04</td>
<td>27.77</td>
<td></td>
<td>1.77901e-04</td>
<td>30.38</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>1.50411e-04</td>
<td>54.82</td>
<td></td>
<td>1.16926e-04</td>
<td>58.28</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>1.27348e-04</td>
<td>81.87</td>
<td></td>
<td>6.44193e-05</td>
<td>85.64</td>
<td></td>
</tr>
<tr>
<td>800</td>
<td>1.02013e-04</td>
<td>109.61</td>
<td></td>
<td>5.26974e-05</td>
<td>112.97</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>7.49826e-05</td>
<td>137.67</td>
<td></td>
<td>3.15274e-05</td>
<td>140.00</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>5.86813e-05</td>
<td>165.36</td>
<td>(iter 1184)</td>
<td>2.28776e-05</td>
<td>165.49</td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>5.37407e-05</td>
<td>191.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>5.04510e-05</td>
<td>218.55</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>4.16955e-05</td>
<td>246.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>3.87314e-05</td>
<td>273.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5. Conjugate gradient strategy with initial weights obtained after 2000 iterations of BBSD algorithm, applied to randomly generated sinus data with $m = 1$, $n = 1$, $T = 20$
### Table 6.
Conjugate gradient using one-dimensional derivatives strategy with initial weights obtained after 2000 iterations of BBSD algorithm, applied to randomly generated sinus data with $m = 1$, $n = 1$, $T = 20$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCGD</th>
<th>BTCGD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj. value</td>
<td>time (sec)</td>
</tr>
<tr>
<td>200</td>
<td>2.20602e-04</td>
<td>38.33</td>
</tr>
<tr>
<td>400</td>
<td>1.37276e-04</td>
<td>78.95</td>
</tr>
<tr>
<td>600</td>
<td>1.29941e-04</td>
<td>119.54</td>
</tr>
<tr>
<td>800</td>
<td>1.12976e-04</td>
<td>157.41</td>
</tr>
<tr>
<td>1000</td>
<td>9.62196e-05</td>
<td>193.58</td>
</tr>
<tr>
<td>1200</td>
<td>8.86204e-05</td>
<td>235.02</td>
</tr>
<tr>
<td>1400</td>
<td>8.64339e-05</td>
<td>276.85</td>
</tr>
<tr>
<td>1600</td>
<td>8.13103e-05</td>
<td>318.90</td>
</tr>
<tr>
<td>1800</td>
<td>8.00507e-05</td>
<td>360.81</td>
</tr>
<tr>
<td>2000</td>
<td>7.27481e-05</td>
<td>401.32</td>
</tr>
</tbody>
</table>

### Table 7.
Steepest descent strategy with random initial weights applied to Energy prediction data with $m = 14$, $n = 3$, $T = 2014$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBSD</th>
<th>BTSD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj. value</td>
<td>time (sec)</td>
</tr>
<tr>
<td>20</td>
<td>24.519</td>
<td>93.39</td>
</tr>
<tr>
<td>40</td>
<td>27.016</td>
<td>186.93</td>
</tr>
<tr>
<td>60</td>
<td>20.885</td>
<td>280.61</td>
</tr>
<tr>
<td>80</td>
<td>18.322</td>
<td>374.43</td>
</tr>
<tr>
<td>100</td>
<td>14.581</td>
<td>468.30</td>
</tr>
<tr>
<td>120</td>
<td>77.067</td>
<td>562.20</td>
</tr>
<tr>
<td>140</td>
<td>8.628</td>
<td>656.1</td>
</tr>
<tr>
<td>160</td>
<td>8.120</td>
<td>750.03</td>
</tr>
<tr>
<td>180</td>
<td>8.536</td>
<td>844.12</td>
</tr>
<tr>
<td>200</td>
<td>7.667</td>
<td>939.17</td>
</tr>
</tbody>
</table>

### Table 8.
Conjugate gradient strategy with random initial weights applied to Energy prediction data with $m = 14$, $n = 3$, $T = 2014$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCG</th>
<th>BTCG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj. value</td>
<td>time (sec)</td>
</tr>
<tr>
<td>20</td>
<td>14.323</td>
<td>713.01</td>
</tr>
<tr>
<td>40</td>
<td>13.267</td>
<td>1371.64</td>
</tr>
<tr>
<td>60</td>
<td>11.658</td>
<td>2045.48</td>
</tr>
<tr>
<td>80</td>
<td>9.125</td>
<td>2699.71</td>
</tr>
<tr>
<td>100</td>
<td>6.446</td>
<td>3340.48</td>
</tr>
<tr>
<td>120</td>
<td>4.666</td>
<td>3961.86</td>
</tr>
<tr>
<td>140</td>
<td>4.209</td>
<td>4577.71</td>
</tr>
<tr>
<td>160</td>
<td>3.965</td>
<td>5165.32</td>
</tr>
<tr>
<td>180</td>
<td>3.802</td>
<td>5729.07</td>
</tr>
<tr>
<td>200</td>
<td>3.637</td>
<td>6316.07</td>
</tr>
</tbody>
</table>
### Table 9. Conjugate gradient using one-dimensional derivatives strategy with random initial weights applied to Energy prediction data with $m = 14$, $n = 3$, $T = 2104$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCGD</th>
<th></th>
<th>BTCGD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj. value</td>
<td>time(sec)</td>
<td>obj. value</td>
<td>time(sec)</td>
</tr>
<tr>
<td>20</td>
<td>14.340</td>
<td>1180.28</td>
<td>14.232</td>
<td>1261.44</td>
</tr>
<tr>
<td>40</td>
<td>12.872</td>
<td>2061.87</td>
<td>12.539</td>
<td>2446.78</td>
</tr>
<tr>
<td>60</td>
<td>9.921</td>
<td>2944.66</td>
<td>9.475</td>
<td>3683.84</td>
</tr>
<tr>
<td>80</td>
<td>6.985</td>
<td>3795.30</td>
<td>6.725</td>
<td>4906.39</td>
</tr>
<tr>
<td>100</td>
<td>5.193</td>
<td>4624.81</td>
<td>5.230</td>
<td>6024.89</td>
</tr>
<tr>
<td>120</td>
<td>4.563</td>
<td>5462.30</td>
<td>4.647</td>
<td>7265.48</td>
</tr>
<tr>
<td>140</td>
<td>4.305</td>
<td>6238.41</td>
<td>4.421</td>
<td>8280.41</td>
</tr>
<tr>
<td>160</td>
<td>3.968</td>
<td>7769.87</td>
<td>3.957</td>
<td>10362.34</td>
</tr>
<tr>
<td>180</td>
<td>3.822</td>
<td>8495.42</td>
<td>3.870</td>
<td>11409.53</td>
</tr>
<tr>
<td>200</td>
<td>3.822</td>
<td>8495.42</td>
<td>3.870</td>
<td>11409.53</td>
</tr>
</tbody>
</table>

### Table 10. Steepest descent strategy with initial weights obtained after 200 iterations of BBSD algorithm, applied to Energy Prediction data with $m = 14$, $n = 3$, $T = 2104$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBSD</th>
<th></th>
<th>BTSD</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj. value</td>
<td>time(sec)</td>
<td>obj. value</td>
<td>time(sec)</td>
</tr>
<tr>
<td>10</td>
<td>18.384</td>
<td>47.15</td>
<td>102.315</td>
<td>48.05</td>
</tr>
<tr>
<td>20</td>
<td>46.407</td>
<td>94.19</td>
<td>104.516</td>
<td>96.93</td>
</tr>
<tr>
<td>30</td>
<td>567.611</td>
<td>141.36</td>
<td>23.288</td>
<td>146.14</td>
</tr>
<tr>
<td>40</td>
<td>567.571</td>
<td>188.54</td>
<td>22.785</td>
<td>197.94</td>
</tr>
<tr>
<td>50</td>
<td>566.193</td>
<td>235.71</td>
<td>20.575</td>
<td>248.69</td>
</tr>
<tr>
<td>60</td>
<td>221.390</td>
<td>282.94</td>
<td>20.298</td>
<td>300.30</td>
</tr>
<tr>
<td>70</td>
<td>324.167</td>
<td>330.18</td>
<td>21.458</td>
<td>350.17</td>
</tr>
<tr>
<td>80</td>
<td>315.227</td>
<td>378.23</td>
<td>21.617</td>
<td>399.52</td>
</tr>
<tr>
<td>90</td>
<td>59.630</td>
<td>425.51</td>
<td>44.465</td>
<td>450.53</td>
</tr>
<tr>
<td>100</td>
<td>202.933</td>
<td>472.65</td>
<td>21.100</td>
<td>501.05</td>
</tr>
</tbody>
</table>

### Table 11. Conjugate gradient strategy with initial weights obtained after 200 iterations of BBSD algorithm, applied to Energy Prediction data with $m = 14$, $n = 3$, $T = 2104$

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCG</th>
<th></th>
<th>BTCG</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>obj. value</td>
<td>time(sec)</td>
<td>obj. value</td>
<td>time(sec)</td>
</tr>
<tr>
<td>10</td>
<td>14.086</td>
<td>330.07</td>
<td>14.086</td>
<td>347.16</td>
</tr>
<tr>
<td>20</td>
<td>8.925</td>
<td>631.01</td>
<td>8.632</td>
<td>682.67</td>
</tr>
<tr>
<td>30</td>
<td>7.487</td>
<td>906.01</td>
<td>6.316</td>
<td>987.66</td>
</tr>
<tr>
<td>40</td>
<td>6.515</td>
<td>1184.68</td>
<td>5.073</td>
<td>1291.89</td>
</tr>
<tr>
<td>50</td>
<td>5.225</td>
<td>1474.37</td>
<td>4.739</td>
<td>1581.21</td>
</tr>
<tr>
<td>60</td>
<td>4.525</td>
<td>1748.22</td>
<td>4.321</td>
<td>1877.69</td>
</tr>
<tr>
<td>70</td>
<td>4.124</td>
<td>2017.45</td>
<td>4.214</td>
<td>2159.06</td>
</tr>
<tr>
<td>80</td>
<td>3.880</td>
<td>2301.37</td>
<td>4.139</td>
<td>2438.4</td>
</tr>
<tr>
<td>90</td>
<td>3.736</td>
<td>2576.11</td>
<td>3.967</td>
<td>2727.34</td>
</tr>
<tr>
<td>100</td>
<td>3.628</td>
<td>2852.75</td>
<td>3.804</td>
<td>3033.29</td>
</tr>
</tbody>
</table>
Table 12. Conjugate gradient using one-dimensional derivatives strategy with initial weights obtained after 200 iterations of BBSD algorithm, applied to Energy Prediction data with $m = 14$, $n = 3$, $T = 2104$.

<table>
<thead>
<tr>
<th>iter</th>
<th>BBCGD obj. value</th>
<th>BBCGD time(sec)</th>
<th>BTCGD obj. value</th>
<th>BTCGD time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>14.085</td>
<td>508.72</td>
<td>14.085</td>
<td>526.19</td>
</tr>
<tr>
<td>20</td>
<td>8.926</td>
<td>938.08</td>
<td>8.578</td>
<td>1021.41</td>
</tr>
<tr>
<td>30</td>
<td>7.490</td>
<td>1329.16</td>
<td>6.004</td>
<td>1618.57</td>
</tr>
<tr>
<td>40</td>
<td>6.518</td>
<td>1713.83</td>
<td>4.935</td>
<td>2165.16</td>
</tr>
<tr>
<td>50</td>
<td>5.240</td>
<td>2094.81</td>
<td>4.355</td>
<td>2657.51</td>
</tr>
<tr>
<td>60</td>
<td>4.538</td>
<td>2485.24</td>
<td>4.136</td>
<td>3209.48</td>
</tr>
<tr>
<td>70</td>
<td>4.135</td>
<td>2881.54</td>
<td>4.015</td>
<td>3664.97</td>
</tr>
<tr>
<td>80</td>
<td>3.887</td>
<td>3285.44</td>
<td>3.931</td>
<td>4253.95</td>
</tr>
<tr>
<td>90</td>
<td>3.746</td>
<td>3691.18</td>
<td>3.831</td>
<td>4828.65</td>
</tr>
<tr>
<td>100</td>
<td>3.632</td>
<td>4073.43</td>
<td>3.715</td>
<td>5349.05</td>
</tr>
</tbody>
</table>

Acknowledgement

The research of Jadranka Skorin-Kapov was partially supported by the NSF grants DDM-937417, SBR-9602021 and ANIR-9814014. The research of Wendy Tang was partially supported by the NSF grants ECS-9407363, ECS-9626655, ECS-9700313.

References


K. WENDY TANG received her B.Sc. (1986), M.Sc. (1988), and Ph.D. (1991) degrees in Electrical Engineering from the University of Rochester in Rochester, New York. She is currently an Associate Professor at the Department of Electrical and Computer Engineering at the State University of New York at Stony Brook. Her research interests include artificial neural networks and communication networks. She is a recipient of the IEEE Third Millennium Award (2000), the IEEE Region 1 Award (1998), and the IEEE/RAB Achievement Award (1998). She has also received several National Science Foundation grants and published in various journals in computers and communications.