Increasing Boltzmann Machine Learning Speed and Accuracy with the High Order Decimation Method

Enric Farguell[1], **Eduardo Gómez-Ramírez**[2], **and Ferran Mazzanti**[3]

[1] Dept. d'Electrònica, EALS La Salle, Universitat Ramon Llull, Quatre Camins 2, E-08022 Barcelona, Spain
`efarguell@salle.url.edu`

[2] Postgrado e Investigación, Universidad La Salle México, Benjamín Franklin 47, Col. Hipódromo Condesa 06140 México D.F., México
`egr@ci.ulsa.mx`

[3] Departament de Física i Enginyeria Nuclear, Universitat Politècnica de Catalunya, Comte Urgell 187, E-08036 Barcelona, Spain
`ferran.mazzanti@upc.edu`

(Paper received on July 31, 2006, accepted on September 25, 2006)

Abstract. The Boltzmann Machine is a stochastic recurrent neural network with the ability to learn and extrapolate probability distributions. However, it has a serious drawback in the exhaustive computational cost involved at learning and simulation stages. Decimation [1, 2] was originally introduced as a way to overcome this problem. Despite its success, the method was too restrictive as it could only be used on sparsely connected Boltzmann Machines with stringent constraints on the connections between the units. The High Order Decimation technique proposed in this work is an extension of the previous method to any Boltzmann Machine with no restrictions on connections nor topology, and is based on the use of high order weights which incorporate additional degrees of freedom. An example on binary patterns is also presented, showing the improvement in efficiency against the performance of the original Boltzmann Machine and a suitably tuned perceptron.

1 Introduction

The Boltzmann Machine (BM) [3] is a stochastic and recurrent neural network based on the Hopfield model [4] with the ability to learn and extrapolate probability distributions. It is formally equivalent to a spin glass [5], thus providing a clear definition of its dynamic and static properties. On the other hand, the exhaustive computational cost implied by the model has prevented its use in most applications of practical interest.

Standard decimation [1, 2] was proposed as a method to reduce the total amount of computational power needed at the learning stage, reducing this process to the solution of a coupled set of nonlinear equations. However and as conceived originally, this set could only be easily solved for a very special and sparse kind of topologies known nowadays as decimatable [2] due to the lack of enough degrees of freedom when applied to regular BMs. The High Order Decimation method explained in the following sections is able to deal with any Boltzmann Machine regardless of its topology, as it adds high order connections which incorporate the extra variables into the equations without a dramatic increase in complexity.

2 The Boltzmann Machine

2.1 Boltzmann Machine Dynamics

The Boltzmann Machine is a Hopfield-like neural network with dynamics inherited from the Simulated Annealing [6] optimization algorithm. It is governed by a cost function that is written in terms of its unit states $S_i = [-1, +1]$

$$E = -\frac{1}{2}\sum_{i,j} w_{ij}^{(2)} S_i S_j - \sum_i w_i^{(1)} S_i \ , \tag{1}$$

where $w_{ij}^{(2)} = w_{ji}^{(2)}$ is a symmetric weight connecting units S_i and S_j, while $w_i^{(1)}$ is a bias term connected to unit S_i. This functional is evaluated with the Simulated Annealing algorithm until the Boltzmann Machine reaches thermal equilibrium at the lowest temperature of the annealing schedule. However, the goal of the process is not to optimize this function but rather to reach a stationary probability distribution which in the end follows a Boltzmann law

$$p(E_\alpha) = \frac{e^{-E_\alpha/T}}{Z} \ , \tag{2}$$

$$Z = \sum_\gamma e^{-E_\alpha/T} \ , \tag{3}$$

where the index α denotes the state of the network and could be written as a string of $+1$ and -1 indicating the state of every unit in the network. Finally, T is the current temperature from the annealing schedule and Z is a normalization term usually known as the *partition function*.

2.2 Learning in Boltzmann Machines

The learning process in a BM is done by minimization of the Kullback-Leibler distance [7] between the actual probability distribution and the one to be learnt. For a BM with different input and output units this quantity reads

$$G = \sum_\alpha R_{\alpha|\gamma} \ln \frac{R_{\alpha|\gamma}}{P_{\alpha|\gamma}} \ , \tag{4}$$

where $P_{\alpha|\gamma}$ is the Boltzmann probability of finding an output state α when a state γ has been set in the input units and, consequently, depends on the weights and biases of Eq. (2). $R_{\alpha|\gamma}$ is the desired probability distribution to be learnt for the same inputs and outputs, and is shown to the network in the form of an input-output training set. Since $G > 0$ for any $P_{\alpha|\gamma} \neq R_{\alpha|\gamma}$ and has a global minimum $G = 0$ when $P_{\alpha|\gamma} = R_{\alpha|\gamma}$, gradient descent is usually employed on Eq. (4) to find the update rule for both weights and biases [5] during learning. Thus

$$\Delta w_\sigma^{(m)} = -\eta \frac{\partial G}{\partial w_\sigma^{(m)}} \ , \tag{5}$$

where $w_\sigma^{(m)}$ is a weight term connecting the m units denoted by the general index σ. Hence, for $m = 2$ and $\sigma = i, j$, $w_{ij}^{(2)}$ is the two-unit weight connecting S_i, S_j. When $m = 1$, $\sigma = i$ and becomes the bias term $w_i^{(1)}$ associated to S_i. Calculating derivative from Eq. 5 will lead to expression

$$w_\sigma^{(m)} = \frac{\eta}{T}\left(\left\langle \prod_{\rho\in\sigma} S_\rho \right\rangle^{*} - \left\langle \prod_{\rho\in\sigma} S_\rho \right\rangle\right) \ , \tag{6}$$

where $< \ldots >^*$ and $< \ldots >$ denote expectation values of its arguments on the current probability distribution, while the star on the average indicates that the output units are *clamped* to the desired output for a given input. These moments have to be calculated for each vector of the training set, and are estimated in the Monte Carlo simulations or annealing process. Finally, η is a convergence parameter and T the lowest equilibrium temperature from the cooling schedule.

2.3 The High Order Boltzmann Machine

The High Order Boltzmann Machine (HOBM) [8, 9] is an extension to the original model where connections through more than two units are allowed. In this way, weights on a M-th order HOBM can connect up to M units, though not all feasible combinations may necessarily be implemented. An example of a high order weight connecting three units is shown in Fig. 1.

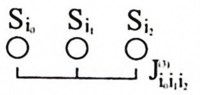

Fig. 1. Simplified notation for 3rd order connections

Of course the cost function or energy functional also changes as it has to allow for the inclusion of the new weights. The most natural extension is [9]

$$E = -\sum_{i_0} w_{i_0}^{(1)} S_{i_0} - \frac{1}{2!}\sum_{i_0,i_1} w_{i_0 i_1}^{(2)} S_{i_0} S_{i_1} - \frac{1}{3!}\sum_{i_0,i_1,i_2} w_{i_0 i_1 i_2}^{(3)} S_{i_0} S_{i_1} S_{i_2} - \ldots$$

$$= -\sum_{m=1}^{M}\left(\frac{1}{m!}\sum_{\sigma} w_{\sigma}^{(m)} \prod_{\rho\in\sigma} S_{\rho}\right) , \tag{7}$$

where m denotes the order of the weight and σ runs over the sets of m units actually connected in the network considered. Despite the change in E the dynamics remain the same, and a Boltzmann probability distribution corresponding to that energies is still reached at equilibrium.

Learning in a HOBM is carried out as in a standard BM, and so the Kullback-Leibler distance is still used to find weight updating expressions, which conform to Eq. (6) with the appropriate extension of the indexes displayed. Nevertheless, the evaluation of the expectation values in the Monte Carlo simulation becomes more and more involved with increasing m, as stated on Ref. [10]. The High Order Decimation method allows an analytical evaluation of these values, as it is explained in the following sections.

3 Standard Decimation of Boltzmann Machines

Decimation is a standard technique in statistical mechanics [11] that is used in this context as a procedure to reduce the number of weights and units on a Boltzmann Machine with stringer connectivity constraints [1, 2]. Given a standard 2nd order BM with N units and tree topology, it is possible to build a smaller network with $n < N$ units that still produces the same expectation values of Eq. (6). This is accomplished by decimating any given S unit connected to no more than three other units S_i, S_j and S_k, regardless of whether they are clamped or not.

Decimation is better carried out when a new, temperature normalized weights are employed

$$J_{\sigma}^{(m)} = \frac{w_{\sigma}^{(m)}}{T} . \tag{8}$$

In its practical form, decimation of a unit S connected to other neurons by two-unit weights is accomplished when the corresponding terms in the partition function fulfill the relation

$$\sum_{S=-1}^{+1} e^{S\sum_i S_i J_i^{(2)}} = \sqrt{C} e^{\sum_{j<i} S_i S_j J_{ij}^{(2)}} . \tag{9}$$

This leads to the master equation

$$\ln\cosh\left(\sum_i S_i J_i^{(2)}\right) = J^{(0)} + \sum_{j<i} S_i S_j J_{ij}^{(2)}, \tag{10}$$

where $J^{(0)} = \ln(\sqrt{C}/2)$ is a normalization constant that is required to solve the system of equations. Notice that the goal of this procedure is to *remove* unit S from the network by changing the weights among the rest of the neurons in the system. In this way the $J_i^{(2)}$ weights appearing on the lhs of the equations above stand for their values prior to the decimation, while the $J_{ij}^{(2)}$ weights on the rhs correspond to the new set of weights once the decimation has been carried out.

3.1 Decimation Equations

A simple example of high order decimation is the star–triangle transformation of Fig. 2. The new weights obtained from the decimation of the unit in the center of the star become

$$J_{ij}^{(2)} = \frac{1}{4} \ln \cosh \left(\frac{\left(J_i^{(2)} + J_j^{(2)} + J_k^{(2)}\right)\left(J_i^{(2)} + J_j^{(2)} - J_k^{(2)}\right)}{\left(J_i^{(2)} - J_j^{(2)} - J_k^{(2)}\right)\left(J_i^{(2)} - J_j^{(2)} + J_k^{(2)}\right)} \right),$$

$$J_{ik}^{(2)} = \frac{1}{4} \ln \cosh \left(\frac{\left(J_i^{(2)} + J_j^{(2)} + J_k^{(2)}\right)\left(J_i^{(2)} - J_j^{(2)} + J_k^{(2)}\right)}{\left(J_i^{(2)} - J_j^{(2)} - J_k^{(2)}\right)\left(J_i^{(2)} + J_j^{(2)} - J_k^{(2)}\right)} \right),$$

$$J_{jk}^{(2)} = \frac{1}{4} \ln \cosh \left(\frac{\left(J_i^{(2)} + J_j^{(2)} + J_k^{(2)}\right)\left(J_i^{(2)} - J_j^{(2)} - J_k^{(2)}\right)}{\left(J_i^{(2)} + J_j^{(2)} - J_k^{(2)}\right)\left(J_i^{(2)} - J_j^{(2)} + J_k^{(2)}\right)} \right). \quad (11)$$

The serial association of Fig. 2b is a particular case of the previous result when $J_k^{(2)}$ is set to zero

$$J_{ij}^{(2)} = \frac{1}{2} \ln \left(\frac{\cosh\left(J_i^{(2)} + J_j^{(2)}\right)}{\cosh\left(J_i^{(2)} - J_j^{(2)}\right)} \right). \quad (12)$$

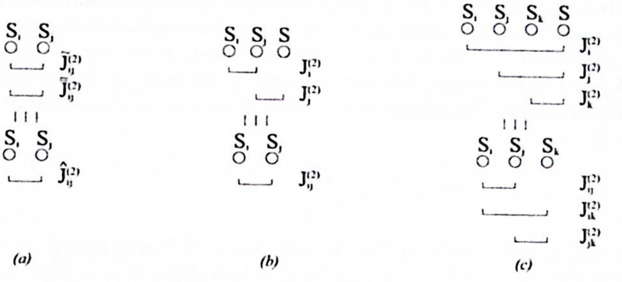

Fig. 2. Serial (a), parallel (b) and star-triangle association (c)

Finally, the parallel association of Fig. 2a is trivially seen to be the sum of the weights and is a simplification rule rather than a decimation one that is included for consistency reasons.

4 High Order Decimation

Decimation, as stated in the previous section, can not be used to simplify a unit connected to more than three other units. Hence, a structure like the one showed in Fig. 3 can not be decimated with the techniques explained so far.

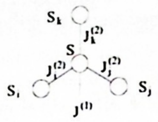

Fig. 3. Simplest non decimatable structure

Decimation fails to simplify these structures because the equations obtained are not compatible with one another. The High Order Decimation method adds enough high order terms for the system to become solvable. In order to explain why this happens, it is useful to take a look at the equations that describe the easiest non-solvable example, which consists of a unit S connected to three other units and a bias term as shown in Fig. 3

$$\ln\cosh\left(J^{(1)} + J_i^{(2)} S_i + J_j^{(2)} S_j + J_k^{(2)} S_k\right) =$$
$$J^{(0)} + J_{ij}^{(2)} S_i S_j + J_{ik}^{(2)} S_i S_k + J_{jk}^{(2)} S_j S_k J_{jk}^{(2)} \ . \tag{13}$$

The complete set of equations born from this expression can not be solved for arbitrary values of $J^{(1)}, J_i^{(2)}, J_j^{(2)}$ and $J_k^{(2)}$. The resulting system of equations can only be solved if there are as many equations as unknown variables, provided that the equations are linearly separable. Hence, four variables are needed. If bias terms associated to units S_i, S_j and S_k are included, there is only one unknown left. A third order weight can then be used to account for this variable. This term will link these three units in the set of equations, and appears as $J_{ijk}^{(3)}$

$$\ln\cosh\left(J^{(1)} + \sum_i S_i J_i^{(2)}\right) \tag{14}$$
$$= J^{(0)} + \sum_i S_i J_i^{(1)} + \sum_{j<i} S_i S_j J_{ij}^{(2)} + \sum_{k<j<i} S_i S_j S_k J_{ijk}^{(3)} \ .$$

This set of equations is characterized by a Hadamard matrix [12]. Hadamard matrices have orthogonal rows and columns so the system has a solution that is unique. The same thing happens when an $N+1$-star groups of units is considered, where a central biased S unit is connected to units S_0 to S_{N-1}. In this case, weights up to N-th order have to be added. As a result, a fully connected N-th order neural network is obtained. The total number of equations required to perform decimation is 2^N, as there are $\binom{N}{0} = 1$ normalization constants $J^{(0)}$, $\binom{N}{1}$ biases $J^{(1)}$, $\binom{N}{2}$ second order weights $J^{(2)}$, $\binom{N}{3}$ third order terms $J^{(3)}$ and so on, until the last $\binom{N}{N} = 1$ N-th order $J^{(N)}$ weight. This makes a total of 2^N variables. The process is schematically shown in Fig. 4, where the originally second order neural network on Fig. 4a is decimated to produce the result of Fig. 4b. It can be proved that this system has always an associated Hadamard matrix, meaning that there will always be a unique solution for a given structure.

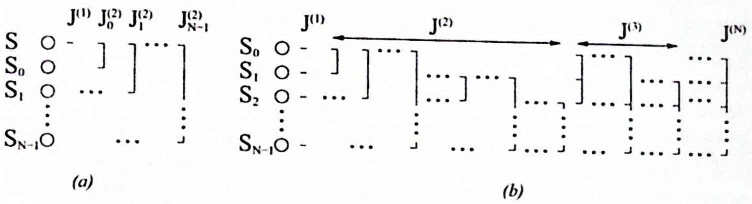

Fig. 4. Original (a) and decimated (b) structure. New weights $J^{(m)}$ connect as much as m different S_i units

When the decimation is to be carried out, all weights connected to the central unit S have to be included, and the corresponding equations have the following form

$$\ln\cosh\left(J^{(1)} + \sum_i S_i J_i^{(2)} + \sum_{j<i} S_i S_j J_{ij}^{(3)} + \ldots + \sum_\sigma J_\sigma^{(m)} \prod_{\rho\in\sigma, S\notin\rho} S_\rho\right) =$$

$$= J^{(0)} + \sum_i S_i J_i^{(1)} + \sum_{j<i} S_i S_j J_{ij}^{(2)} + \sum_{k<j<i} S_i S_j S_k J_{ijk}^{(3)} + \ldots \quad (15)$$

Once the system is solved, a new Boltzmann Machine with one less unit is left, at the expense of it being highly connected due to the many different high order weights produced by the decimation reduction. In order to compute the needed m-th order correlations appearing in the weight update learning rule, the process is iterated until all but the required units are decimated, in such a way that the expectation values entering in the updates can be computed analytically as noted in Ref. [1, 2]. The whole process has to be repeated for every weight that has to be updated.

5 Results and Discussion

The high order decimation method has been tested against a perceptron and a traditional BM in a letter recognition dataset context, where the network has to recognize characters from a noisy source. A system of 24 letters written with a *Times New Roman* font is used, and each letter is represented by a 50x50 pixels binary image. While these neat characters are the ones to be learned, a set of 100 different images for each letter is generated by adding random noise which is implemented via bit negation. The amount of noise present is characterized by a parameter γ, which is proportional to the percentage of negated bits. Thus for example, 10% of bits are reversed when $\gamma = 10$. Once the learning using the previous patterns has been carried out, a new set of noisy characters is generated using the same procedure to test the network.

In practical terms, execution time is sped up when a momentum term characterized by an extra parameter α is added to the original gradient descent expression [13]

$$w^{(k+1)} = w^{(k)} - \eta\,(1-\alpha)\,\frac{\partial \mathcal{E}}{\partial w} + \alpha w^{(k-1)} \ , \tag{16}$$

$\partial\mathcal{E}/\partial w$ denoting the partial derivative of the error. Since each neural network uses a different measure for the error, this expression may be thought as a generalization of the learning algorithm which has been adapted for the three networks used in this work.

5.1 Monte Carlo Implementation

Table 1 shows the amount of time and epochs (equal to the number of times weights are updated in a complete run of the learning algorithm) required by a Boltzmann Machine trained with the High Order Decimation method compared with results for the same network trained with the standard Monte Carlo algorithm using the above stated patterns for $\gamma = 20$. All calculations have been performed on a DELL workstation mounting a Pentium Xeon EMT64 with 2Mb of cache processor working at 3.0 GHz and equipped with 1.0 Gb DDR2 ECC RAM memory. As it can be seen, not only the High Order decimation performs *faster* but also requires less epochs to reach the desired result. This is due to the fact that every Monte Carlo simulation has an associated statistical error, and bringing that below a certain limit (imposed by the accuracy to be achieved) can be very expensive in computational terms. The network used in this calculation has 2500 input units (corresponding to the 50x50 pixel images used as input), 1 hidden and 5 output units. The training parameters were $\eta = 0.2$, $\alpha = 0.1$, maximum absolute error $\mid \partial\mathcal{E}/\partial w \mid = 0.05$ and maximum absolute initial random value for the weights $|w_0| = 1.0$.

The relation *time/epoch* describes how long does it take to run a complete epoch in the simulation. As it can be seen from the table, Decimation performs considerably better in both aspects. A Decimation epoch is faster because there is no need to run a Simulated Annealing but only to solve a system of equations.

Table 1. Decimation method against Monte Carlo implementation

Algorithm	Mean epochs	Mean time/epoch (seconds)
Monte Carlo	45	586.68
High Order Decimation	11	4.39

On the other hand, it needs less epochs to end because it does not suffer from statistical errors as does a Monte Carlo simulation.

5.2 Perceptron

Finally, a comparison between the performance of the BM trained with the High Order Decimation method and a dual layer perceptron is presented. The comparison is made on the basis that both networks can provide a *full* solution to the problem at hand if enough learning instances are allowed. In fact both networks have been trained many times and its efficiency tested at the end of each learning process, finding that both systems can be 100% efficient in many cases. Taking into account this fact, the mean efficiency over a batch of instances of the same problem has been measured, and this parameter used t decide which network performs better in a statistical sense.

The BM used in the comparison is fully connected, with four output units and a variable number of hidden neurons ranging from zero to two. Learning parameters are once again $\eta = 0.2$, $\alpha = 0.1$, maximum absolute error $|\partial\mathcal{E}/\partial w|=$ 0.05 and maximum absolute initial random value for weights $|w_0|= 1.0$. On the other hand, the topology of the perceptron employed has been optimized to get best results. The experiment has been repeated using a number of hidden units spanning the range from 5 to 2500, using both lineal and hyperbolic tangent transfer functions, and a momentum α between 0.0 and 0.2 with an adaptive η learning rate. Results on the performance are presented in Table 2.

Table 2. Decimation method against perceptron

γ	Mean eff. for BM	Mean eff. for perceptron
0.10	97.25	60.38
0.15	96.87	68.77
0.20	94.58	83.49
0.25	94.05	88.55
0.30	86.90	84.87

It can be seen from the table that the Boltzmann Machine performs slightly *better* than the perceptron. This can be understood when the problem is carefully analyzed, as it has an original discrete nature. Since the Boltzmann Machine is a binary neural network and the perceptron is a continuous one, the BM is better suited to solve the problem. However, the perceptron is a widely used multi purpose network, and it can perform very well on problems where other continuous models fail. In any case and although the Boltzmann Machine does a better job, the perceptron still provides solutions that are more than satisfactory. Still when High Order Decimation is employed, the Boltzmann Machine not only outperforms the perceptron but also gets the solution in a similar period of time.

6 Summary and Conclusions

In this work a generalization of the standard decimation method in the learning of Boltzmann Machines is presented and discussed, showing that the inclusion of high order weights can simplify the learning process. Comparisons with the standard BM and an optimized perceptron show that the procedure is not only easy to implement but also that its performance in terms of efficiency and computational cost is better. In this way, high Order Decimation may be the key to bring Boltzmann Machines to a level where they become effective in the resolution of practical problems.

References

1. Saul, L., Jordan, M.I.: Learning in Boltzmann Trees. Neural Computation, Vol. 6., Num. 6 (1994) 1174–1184
2. Rüger, S.M.: Decimatable Boltzmann Machines for Diagnosis: Efficient Learning and Inference. World Congress on Scientific Computation, Modeling and Applied Mathematics, Vol. 4: Artificial Intelligence and Computer Science (1997) 319–324
3. Ackley, D.H., Hinton, G.E., and Sejnowski, T.J.: A learning algorithm for Boltzmann machines. Cognitive Science, Vol. 9. (1985) 147–169
4. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA 79 (1982) 2554–2558
5. Hertz, J., Krogh, A., Palmer, R.G.: Introduction to the theory of neural computation, Addison-Wesley Publishing Company, Santa Fe Institute (1991)
6. Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science, Vol. 220 (1983) 671–680
7. Kullback, S.: Information theory and statistics. New York: Willey, 2nd ed. (1959)
8. Sejnowski, T.J.: High-Order Boltzmann Machines. AIP Conference Proceedings 151 on Neural Networks for Computing, Snowbird, Utah, E.U.A. (1987) 398–403
9. Albizuri, F.X., d'Anjou, A., Graña, M., Torrealdea, F.J., Hernández, M.C.: The High-Order Boltzmann Machine: Learned distribution and Topology, IEEE Transactions on Neural Networks, Vol. 6, Num.: 3 (1995) 767–770
10. Graña, M., d'Anjou, A., Albizuri, F.X., Hernández, M, Torrealdea, F.J., de la Hera, A., González, A.I.: Experiments of Fast Learning with High Order Boltzmann Machines. Applied Intelligence, Vol. 7, Num. 4 (1997) 287–303
11. Itzykson, C., Drouffe, J.: Statistical field theory. Cambridge University Press (1991)
12. Álvarez, V., Armario, J.A., Frau, M.D., Real, P.: Matrices cocíclicas de Hadamard sobre productos semidirectos. III Jornadas de Matemàtica Discreta y Algorítmica 3JMDA, Sevilla (2002) 155–158
13. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Second Ed., Wiley-Interscience Publication, John Wiley & Sons, INC. (2001)