Fast Hopfield Neural Networks Using Subspace Projections[1]

Daniel Calabuig, Sonia Gimenez, Jose E. Roman, Jose F. Monserrat

dacaso@iteam.upv.es, sogico@teleco.upv.es,

jroman@itaca.upv.es, jomondel@iteam.upv.es,

Abstract – Hopfield Neural Networks are well-suited to the fast solution of complex optimization problems. Their application to real problems usually requires the satisfaction of a set of linear constraints that can be incorporated with an additional violation term. Another option proposed in the literature lies in confining the search space onto the subspace of constraints in such a way that the neuron outputs always satisfy the imposed restrictions. This paper proposes a computationally efficient subspace projection method that also includes variable updating step mechanisms. Some numerical experiments are used to verify the good performance and fast convergence of the new method.

Keywords – Hopfield Neural Networks, Linear Constraints, Projection.

I. Introduction

A Hopfield Neural Network (HNN) is a specific kind of recurrent neural network designed for the minimization of an energy function that contains several terms [9]. From the Hopfield neuron model, any problem that can be written in terms of a second order Lyapunov function can be solved with a quasi-optimal solution using HNNs. These neural networks have gained much relevance in the last decade as a good tool to solve complex optimization problems mainly thanks to their fast response time. Certainly, one of the main advantages of neural techniques is the high computational speed obtained through their hardware implementations, which is even more valuable when considering their usage for industrial applications. Actually, the use of HNNs has

[1] NOTICE: this is the author's version of a work that was accepted for publication in Neurocomputing. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Neurocomputing, vol. 73, no. 10, pp. 1794-1800, 2010, DOI:10.1016/j.neucom.2009.12.031.

been recently suggested for several time-constrained practical problems due to this characteristic – see for example [2], [4], [5].

However, HNNs have also acquired many detractors because of the poor quality of the obtained solutions, mostly when the energy function is non-convex [8]. This condition provokes some additional problems related to the convergence to non-desired local minima. These problems could be solved by means of a fine tuning of the energy function parameters that properly penalizes the non-desired states [4]. In general, a mathematical analysis of the borderline cases must be performed in order to derive the proper weighting values. However, this problem becomes even more complex in practical optimization problems, since reality imposes a set of strict constraints that must be taken into account. Consequently, in real problems a set of linear constraints is incorporated in the energy function, adding some additional constraint violation terms [9]. Despite being a common practice, some authors have demonstrated that the inclusion of the violation terms results in more likely invalid solutions [13] and even in a change in the network behaviour [3]. The underlying problem that causes these convergence problems lies in the fact that the cost terms run contrary to the constraint terms and, as a result, the networks converge to local minima far from the absolute minimum.

In order to tackle this problem, some authors proposed to confine the HNN to the feasible constraint subspace, hence ensuring the final solution validity [6], [11]. Chu [6] proposed to project the energy gradient, which indicates the direction of movement, modifying the neuron inputs in such a way that the neuron outputs are always in the constraint plane. Although this seminal work was the first one bringing forward the usage of Projection-based HNNs (P-HNNs), it assumed a continuous-time neural network and no reference was made to more realistic computer implementations which are inherently discrete in time. The main consequence is that discrete-time implementations continuously separate from the feasible subspace due to large computational errors when neurons are near the extremes. Moreover, the projection matrix was explicitly calculated from the constraints matrix, without considering issues of practical relevance such as computational efficiency and numerical robustness. On the other hand, Smith *et al.* [11] defined an iterative mechanism based on the integration of the projection of the neuron outputs – instead of projecting the energy gradient –

together with an annealing technique for escaping local minima by permitting, in a controlled way, increments of the energy function. Comparing this mechanism with [6], both use the same calculation method for the projection matrix. However, Smith *et al.* incorporated more effective means to guarantee stability and convergence to feasible solutions, but at the expense of extremely increasing the computational burden.

This paper presents a new, computationally efficient subspace projection method that includes the variable updating step mechanism proposed by Talaván and Yáñez [12]. Once the direction of movement is confined in the subspace of constraints, neuron states are modified so that the energy is reduced as much as possible. The proposed method performs projections by means of the orthogonalization with respect to an appropriate basis of vectors. This basis is dynamically augmented in order to guarantee that the modified neuron state vector does not exit the space of allowed solutions. This numerical procedure for the projection is computationally efficient.

Concerning the implementation of this Fast HNN (F-HNN), it is worth highlighting that, although some authors state that the HNN response could be attained in a few microseconds [2], this order of magnitude corresponds with the analogue implementation of continuous HNN. Nevertheless, so far the performance of HNN has been determined assuming a discrete-time implementation in computers and without implementing the analogue circuit. This fact is due to the enormous size of the hardware network when considering a high number of neurons and the difficulty of accurately implementing the resistor values, which can change network behaviour. In fact, the few hardware implementations found in the literature have been developed on digital devices, for example [1], in order to reduce the size of the network and solve the precision problems in the resistors' values. However, these implementations lose the benefits of the parallel interworking, since they make use of a central processing unit to update the neuron outputs sequentially in each iteration. An implementation on a computer can exploit the increasing potential of parallelism offered by current processor architectures. The proposed F-HNN method retains this inherent capability of parallelization so that fast response times can be expected. Even in a sequential implementation, the method presents advantages. Some numerical experiments are used to verify the performance and fast convergence of the proposed method as compared with other P-HNN techniques.

II. Mathematical Foundations of the Fast Hopfield Neural Network

The objective of this section is to describe the main mathematical principles on which the F-HNN proposed in this paper is based. With this aim, the classical HNN is firstly explained. Next, there is a description of the two main concepts that, jointly used, enable the fast convergence of the new neural network: the variable updating step and projection of the energy gradient. In Section II.E another possibility that consists in executing variable reduction methods before the optimum search is discussed, justifying why this option was dismissed. The last part of this section deals with some efficient computational methods to reduce the number of operations and numerical errors. Section III summarizes the set of steps to be taken in each update of the F-HNN state.

A. Fundamentals of Hopfield Neural Networks

HNNs can be completely defined with an energy function of the form [9]:

$$E\left(\mathbf{v}(t)\right) = -\frac{1}{2}\mathbf{v}(t)'\mathbf{T}\mathbf{v}(t) - \mathbf{i}'\mathbf{v}(t), \tag{1}$$

where $\mathbf{v}(t)$ is the $N \text{x} 1$ vector of the neuron outputs at time t with elements $V_i(t) \in [0,1]$, $\mathbf{T}$ is an $N \text{x} N$ symmetric matrix, $\mathbf{i}$ is an $N \text{x} 1$ vector, with elements T_{ij} and I_i respectively, of constant parameters that define the neural network, and N is the number of neurons. These parameters should be selected so that the energy minima inside the unit hypercube, that is, $V_i(t) \in [0,1]$, are the desired solutions of the optimization problem.

Let $\mathbf{d}(t)$ be defined as the $N \text{x} 1$ vector of the updating direction at time t. Then, each neuron is updated following:

$$V_i(t+1) = V_i(t) + \Delta_i(t), \tag{2}$$

$$\Delta_i(t) = \begin{cases} 0, & d_i(t) > 0, V_i(t) = 1, \\ 0, & d_i(t) < 0, V_i(t) = 0, \\ \beta(t) d_i(t), & \text{otherwise}, \end{cases} \tag{3}$$

where $\beta(t) > 0$ is the updating step at time t, and $d_i(t)$ is the i-th element of $\mathbf{d}(t)$. The updating step and direction must be selected to make the network converge towards a local minimum of the energy. For that reason, the updating direction is typically minus the energy gradient in the bibliography, since it points to the maximum decrement of the energy. However, any direction with a negative directional derivative decreases the energy too and hence is also a good candidate. The updating step is typically constant but, if so, has to be sufficiently small to prevent oscillations. Therefore, a lot of iterations are needed until the convergence point is reached.

B. Variable Updating Step

The Variable Updating Step (VUS) technique was proposed by Talaván and Yáñez [12] to increase the convergence speed of the neural network. The idea is to move in the direction $\mathbf{d}(t)$ until the energy minimum – in that direction – is reached. The energy over direction $\mathbf{d}(t)$ is:

$$E\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right)=-\frac{1}{2}\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right)'\mathbf{T}\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right)-\mathbf{i}'\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right), \qquad (4)$$

$$E\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right)=E\left(\mathbf{v}(t)\right)-S_1\alpha+S_2\alpha^2, \qquad (5)$$

$$S_1=\mathbf{d}(t)'\left(\mathbf{Tv}(t)+\mathbf{i}\right), \qquad (6)$$

$$S_2=-\mathbf{d}(t)'\mathbf{Td}(t). \qquad (7)$$

Parameter α is any possible updating step. The energy of (5) is a quadratic function with respect to α. Thus, it has a critical point that can be either a maximum or a minimum. This critical point is [12]:

$$\left.\frac{dE\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right)}{d\alpha}\right|_{\alpha=\alpha_0}=0\Rightarrow\alpha_0=\frac{S_1}{S_2}. \qquad (8)$$

It is worth noting that S_1 is minus the directional derivative of the energy over the direction $\mathbf{d}(t)$. Therefore, let us assume that $S_1\geq 0$ and hence that direction $\mathbf{d}(t)$ points towards points with less energy, that is:

$$\exists\varepsilon>0:\forall\alpha,0<\alpha<\varepsilon\rightarrow E\left(\mathbf{v}(t)+\alpha\mathbf{d}(t)\right)\leq E\left(\mathbf{v}(t)\right). \qquad (9)$$

Moreover, S_2 is the second derivative of the energy of (5) with respect to α. Therefore, if $S_2>0$ then the critical point α_0 is a minimum of (5). Conversely, if $S_2<0$ then α_0 is a maximum. In the first case, the VUS should be α_0 at most, since greater steps will not further reduce the energy and could even increase it. In the second case, the VUS can be as high as possible, since the energy will continuously decrease in the direction $\mathbf{d}(t)$.

Additionally to the previous paragraph, the VUS must satisfy some other constraints. More specifically, the updated neuron outputs must remain in the unit hypercube, that is, $\mathbf{v}(t)+\beta(t)\mathbf{d}(t)\in[0,1]^N$. For this aim it is necessary to take into account the distances between the current neuron outputs and the extremes 0 and 1. Let us define $\mathbf{l}(t)$ as the Nx1 vector of limits whose elements are:

$$l_i(t)=\begin{cases}\dfrac{1-V_i(t)}{d_i(t)}, & d_i(t)>0,\\ -\dfrac{V_i(t)}{d_i(t)}, & d_i(t)<0,\\ \infty, & d_i(t)=0.\end{cases} \tag{10}$$

Then, the VUS is:

$$\beta(t)=\begin{cases}\min\left\{\min_i\{l_i(t)\},\alpha_0\right\}, & S_2>0,\\ \min_i\{l_i(t)\}, & S_2\le 0.\end{cases} \tag{11}$$

C. Projection of the energy gradient

Usually the solution of a specific problem must strictly satisfy certain conditions. Examples of these problems are the Travelling Salesman Problem (TSP) [9], the N-Queens Problem (NQP) [10], and some resources distribution algorithms [2], [5]. In all these problems, some neuron subsets can be identified in such a way that the desired solution has only one neuron active in each subset.

In this paper, the NQP has been chosen as a case study. The scope of this problem is to distribute a set of Q queens on a $Q\times Q$ chessboard in such a way that no queen could attack the others. The HNN proposed to solve this problem has a total of Q^2 neurons. The neurons can be organized in a neuron matrix so that if a neuron located at coordinates (i,j) is active then there is a queen located at these coordinates. Since there can be only one queen in each row, any solution to this problem must satisfy the requirement that the sum of all neuron outputs of each row is exactly 1. Moreover, there can be only one queen in each column. Therefore, the sum of all neuron outputs of each column must be exactly 1 too. In general, in many optimization problems, valid solutions must satisfy some strict constraints that can be written as linear equations, that is:

$$\mathbf{Av}=\mathbf{b}, \tag{12}$$

where $\mathbf{A}$ is an $M \times N$ matrix and $\mathbf{b}$ is an $M \times 1$ vector that define M strict constraints. This system defines a subset F of network states inside the hypercube where the M constraints are satisfied, that is:

$$\mathsf{F}=\left\{\mathbf{v}:\mathbf{Av}=\mathbf{b},\mathbf{v}\in[0,1]^N\right\}. \tag{13}$$

Although the initial state $\mathbf{v}(0)$ belongs to F, the direction of movement – generally minus energy gradient – may move the network state away from this subset. Obviously, if the problem is well defined, the stable state belongs to F and thus the network must return to it. Figure 1 represents this idea. The typical path is an illustrative neuron evolution followed by an HNN. Nevertheless, if the stable state is inside the subset F, reaching it could be faster if the search space were reduced to this subset. Alternatively to the typical path, a new path could be followed which entirely belongs to F.

Let us define $\overline{\mathsf{F}}$ as the extension of F to all $\mathbb{R}^N$, that is:

$$\overline{\mathsf{F}}=\{\mathbf{v}:\mathbf{Av}=\mathbf{b}\}, \tag{14}$$

and let us define F_0 as the set parallel to $\overline{\mathsf{F}}$ that has the coordinate origin, that is:

$$\mathsf{F}_0=\{\mathbf{v}:\mathbf{Av}=0\}. \tag{15}$$

Then, if the neuron outputs at some time t, $\mathbf{v}(t)$, belong to F, and the updating direction belongs to F_0, then $\mathbf{v}(t+1)\in\overline{\mathsf{F}}$ independently of the updating step, since:

$$\mathbf{Av}(t+1)=\mathbf{A}\left(\mathbf{v}(t)+\beta(t)\mathbf{d}(t)\right)=\mathbf{Av}(t)+\beta(t)\mathbf{Ad}(t)=\mathbf{b}+\beta(t)0=\mathbf{b}. \tag{16}$$

Moreover, if the VUS is selected following the explanation of previous section, then $\mathbf{v}(t+1)\in\mathsf{F}$, since the neuron outputs will always belong to the unit hypercube. As previously mentioned, other works use the direction of minus energy gradient as the updating direction. Then, the path followed by neuron outputs is something similar to the typical path depicted in Figure 1. It is also possible to project the energy gradient into the set F_0 to define a new updating direction. In this case, the alternative path is similar to that of Figure 1. Let us define $\mathbf{P}$ as the projection matrix that projects any point onto the set F_0, that is:

$$\mathbf{APv}=\mathbf{0},\ \forall\mathbf{v}\in\mathbb{R}^N, \tag{17}$$

Then the updating direction of the alternative path is:

$$\mathbf{d}(t)=\mathbf{P}\left(\mathbf{T}\mathbf{v}(t)+\mathbf{i}\right). \tag{18}$$

This direction satisfies $S_1 \geq 0$ since the angle between $\mathbf{d}(t)$ and the direction of minus energy gradient is always less than or equal to 90º, and thus:

$$S_1=\mathbf{d}(t)'\left(\mathbf{T}\mathbf{v}(t)+\mathbf{i}\right)=\left\|\mathbf{d}(t)\right\|\left\|\mathbf{T}\mathbf{v}(t)+\mathbf{i}\right\|\cos\gamma \geq 0, \tag{19}$$

where $\|\mathbf{x}\|$ is the norm of $\mathbf{x}$, and γ is the angle between $\mathbf{d}(t)$ and the direction of minus energy gradient.

D. Projections in the hypercube facets

The main challenge of the projections approach is how to deal with projections in the hypercube facets. Inside the facets, at least one neuron is at one of the extremes 0 or 1. For instance, if neuron i is at the extreme $V_i(t)=0$ at some time t, that neuron should not be modified if the updating direction is $d_i(t)<0$; see (3). This fact is equivalent to changing the updating direction from $\mathbf{d}(t)$ to $\hat{\mathbf{d}}(t)$, where the components of $\hat{\mathbf{d}}(t)$ are:

$$\hat{d}_j(t)=\begin{cases} d_j(t), & j \neq i, \\ 0, & j=i. \end{cases} \tag{20}$$

Since $\mathbf{A}\mathbf{d}(t)=0$, then $\mathbf{A}\hat{\mathbf{d}}(t)\neq 0$ due to the change performed in (20). This fact means that the next neuron state will not belong to $\overline{\mathsf{F}}$. Obviously, neuron outputs must leave neither the unit hypercube nor the constraints subspace. Both requirements can be accomplished by adding new constraints to matrix $\mathbf{A}$. More specifically, (20) can be understood as a new linear constraint of the form $\hat{d}_j(t)=0$. Therefore, it is possible to build a matrix $\mathbf{B}$ with all the new constraints. Then, the projection matrix $\widehat{\mathbf{P}}$ is computed from the combination of matrices $\mathbf{A}$ and $\mathbf{B}$ so that:

$$\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\end{bmatrix}\widehat{\mathbf{P}}\mathbf{v}=\mathbf{0}, \forall \mathbf{v}\in\mathbb{R}^N. \tag{21}$$

Finally, the updating direction is now:

$$\mathbf{d}(t)=\widehat{\mathbf{P}}(\mathbf{T}\mathbf{v}(t)+\mathbf{i}). \tag{22}$$

Continuing with the previous example, this updating direction not only belongs to subset F_0 but, additionally, has its i-th component null.

E. Reduction of the dimensionality of the problem

For an HNN of N neurons and M strict constraints, the reader could think that a good option could be to reduce the number of variables, that is, to transform the N neurons problem into $N-M$ instead of projecting the energy gradient in every iteration. This type of transformation and the projections have a similar computational cost. The idea would be to perform the transformation only once at the beginning of the algorithm, hence the expected computational gain. Nevertheless, the problem with this approach is that the hypercube becomes a convex polytope in the subspace and it is not easy to know whether any of the $N-M$ new neurons has reached one of the facets. The easiest way to confirm this is to undo the transformation, check it, modify neuron outputs if necessary – something similar to section II.D – and perform the transformation again. This procedure should be done for all iterations and is, obviously, hugely more costly than gradient projections. For this reason, this procedure is not recommended.

F. Practical realization of the projection

Although it has been done so far in other works dealing with projected HNNs, explicitly computing the projector matrix, $\mathbf{P}$, for the projection in (18) is not practical because of the computational inefficiency and also due to potential difficulties related to numerical error. A better approach can be derived from a subspace analysis. The subspace F_0 is the null space of matrix $\mathbf{A}$. It is well known that the null space of a matrix is the orthogonal complement to the row space of that matrix – the subspace spanned by its rows. In this setting, $\mathbf{P}$ is the orthogonal projector onto the null space of $\mathbf{A}$. Orthogonal projectors admit a simple representation. Let $\mathbf{Q}$ be an $N \text{ x } M$ matrix whose columns constitute an orthonormal basis of the row space of $\mathbf{A}$, that is, $\mathbf{Q}'\mathbf{Q} = \mathbf{I}$ and $\text{span}(\mathbf{Q}) = \text{span}(\mathbf{A}')$. Then, the projector can be written as $\mathbf{P} = \mathbf{I} - \mathbf{Q}\mathbf{Q}'$. With this representation, the projection of vector $\mathbf{x}$ can be done with:

$$\mathbf{Px} = (\mathbf{I} - \mathbf{Q}\mathbf{Q}')\mathbf{x} = \mathbf{x} - \mathbf{Q}\mathbf{Q}'\mathbf{x}. \tag{23}$$

That is, first the vector is projected onto the subspace spanned by the columns of $\mathbf{Q}$ (the row space of $\mathbf{A}$), and then this projection is removed from the original vector. This procedure is extensively used in many areas of numerical linear algebra and is often referred to as the orthogonalization of a vector with respect to a set of orthonormal vectors.

An interesting property of orthogonal projectors is that, in cases of partitioning the columns of $\mathbf{Q}$ in two sets, $\mathbf{Q} = [\mathbf{Q}_1\ \mathbf{Q}_2]$, then the projection can be written as:

$$\mathbf{Px} = \left(\mathbf{I} - \mathbf{Q}_2\mathbf{Q}_2'\right)\left(\mathbf{I} - \mathbf{Q}_1\mathbf{Q}_1'\right)\mathbf{x}. \tag{24}$$

Thus, orthogonalization can be applied first with just a subset of vectors and then the final result is obtained after orthogonalizing the previous outcome with respect to the rest. Note that the order is irrelevant because $\left(\mathbf{I} - \mathbf{Q}_2\mathbf{Q}_2'\right)\left(\mathbf{I} - \mathbf{Q}_1\mathbf{Q}_1'\right)$ commute, since $\mathbf{Q}_2'\mathbf{Q}_1 = 0$. In the limit, the orthogonalization can be carried out one vector at a time. This procedure is known as the Modified Gram-Schmidt, as opposed to the Classical Gram-Schmidt procedure of (23). In floating point arithmetic, neither classical nor modified Gram-Schmidt procedures guarantee that the resulting vector is orthogonal to full machine precision. Therefore, in order to ensure complete numerical robustness, it is necessary to use iterative reorthogonalization [7].

In the context of F-HNN, the explicit computation of the projector $\mathbf{P}$ is replaced by the initial computation of $\mathbf{Q}$, that is, the orthogonalization of the rows of $\mathbf{A}$. This can be done, for instance, by means of Gram-Schmidt procedures for computing the QR matrix decomposition. This initial step also has the advantage that it will eventually detect redundant constraints, since when the result of an orthogonalization is the zero vector it means that the original vector already belonged to the subspace. On the other hand, the property of (24) allows additional constraints to be dynamically included as necessary. This is essential for adding new constraints as explained in section II.D.

III. Different Alternatives for Projection Hopfield Neural Networks

A. Fast HNN (F-HNN)

This is the P-HNN proposed in this paper, which can be summarized as follows:

- Step 1: Initialize matrix $\mathbf{A}$ and derive $\mathbf{Q}$. Define a random vector $\mathbf{v}(t)$ for $t = 0$, so that $\mathbf{Av}(0) = \mathbf{b}$.
- Step 2: Calculate the energy gradient as $\nabla E = -\mathbf{Tv}(t) - \mathbf{i}$.
- Step 3:

    1. Obtain the updating direction as $\mathbf{d}(t) = -\nabla E + \mathbf{Q}\mathbf{Q}'\nabla E$.
    2. Check that all neurons will be confined in the hypercube (see the procedure described in section II.D).
    3. If all neurons are confined, go to Step 5. Otherwise, go to Step 4.

- Step 4: While there is any neuron not confined in the hypercube:
    1. Add new constraints to $\mathbf{A}$ and derive the new columns of $\mathbf{Q}$, $\mathbf{Q}_2$.
    2. Update $\mathbf{d}(t) \leftarrow \mathbf{d}(t) - \mathbf{Q}_2\mathbf{Q}_2'\mathbf{d}(t)$.
    3. $\mathbf{Q} = [\mathbf{Q}\,\mathbf{Q}_2]$.
- Step 5: Calculate S_1, S_2, $\mathbf{l}(t)$, and $\beta(t)$ following the reasoning described in Section II.B – Equations (6), (7), (10), and (11), respectively.
- Step 6: Update neuron states as $\mathbf{v}(t+1) = \mathbf{v}(t) + \beta(t)\mathbf{d}(t)$.
- Step 7: $t \leftarrow t+1$. Go to Step 2 until the termination criterion is met.

All of this procedure is shown in Figure 2.

B. HNN with Gradient Projections (GP-HNN)

As described in the introduction, Chu [6] proposed the projection of the energy gradient considering a continuous HNN. However, Chu did not take into account all the implications brought by the discrete-time implementation of his proposal.

The GP-HNN method is the discrete counterpart of [6] and can be understood as a simplification of the F-HNN method, since no VUS technique is used. Rather, a fixed time step has been assumed, Δt. Therefore, the GP-HNN procedure is almost the same as F-HNN except that Step 5 is removed and $\beta(t) = \Delta t$. Obviously, Δt must be carefully selected to guarantee the fast convergence to the minimum of the energy.

C. Smith HNN (S-HNN)

The proposal of Smith *et al.* [11] can be summarized in the following steps:

- Step 1: Initialize matrix $\mathbf{A}$ and derive $\mathbf{Q}$. Define a random vector $\mathbf{v}(t)$ for $t=0$, so that $\mathbf{v}(0) \in [0,1]^N$. Obtain $\mathbf{s} = \mathbf{A}'(\mathbf{A}\mathbf{A}')^{-1}\mathbf{b}$ and initialize $U=1$ and $L=1$.
- Step 2: Calculate the energy gradient as $\nabla E = -\mathbf{T}\mathbf{v}(t) - \mathbf{i}$.

- Step 3: Update $k(t) = 1 - 2e^{-t/\tau}$ and generate a random $\alpha(t) \in [k(t), 1]$.
- Step 4: Calculate $\mathbf{x} = \mathbf{v}(t) - \Delta t \cdot \alpha(t) \nabla E$.
- Step 5: Perform the projection and clipping procedure according to the following steps:
    1. Find the projection of $\mathbf{x}$ onto the constraint subspace: $\mathbf{x}^p = \mathbf{x} - \mathbf{QQ'x} + \mathbf{s}$.
    2. Introduce $\mathbf{x}^p$ inside the unit hypercube, modifying all its elements as follows:

$$x_i^p = \begin{cases} U, & x_i^p \geq U \\ L, & x_i^p \leq L \\ \dfrac{x_i^p - L}{U - L}, & \textit{otherwise} \end{cases}$$

    3. $\mathbf{x} = \mathbf{x}^p$ and $\mathbf{e} = |\mathbf{Ax} - \mathbf{b}|$. If $e_i < tol, \forall i = 1 \cdots N,$ go to Step 6. Otherwise, go to Step 5.1.
- Step 6: Update neuron states as $\mathbf{v}(t+1) = \mathbf{x}$ and also update $U \leftarrow U - \varepsilon_0$ and $L \leftarrow L + \varepsilon_0$ according to the periodicity described in [11].
- Step 7: $t \leftarrow t+1$. Repeat from Step 2 until $k(t) = 1$ and $dv_i/dt = 0$ for all i.

Comparing this procedure and the one proposed in [11], it can be noticed that in Step 5 the Gram-Schmidt procedure has been used instead of the direct projection, according to the explanation given in Section II.F. This has been made in order to increase the efficiency of the procedure and allow a fair comparison between the three alternatives. It is also important to highlight that, by means of Step 3 and Step 4, an annealing-like procedure is implemented allowing punctual increments of the energy function. This mechanism was devised by Smith *et al.* to avoid local minima and increase the convergence probability. Step 5 is a projection and clipping procedure that converges to a point inside the unit hypercube and the constraints subspace. Updating U and L as shown in Step 6 makes the clipping more severe with each new iteration, hence forcing the neural network to converge.

In this paper, the same constant parameters defined in [11] are employed, namely, $\Delta t = 10^{-4}$, $\varepsilon_0 = 10^{-5}$, and $\tau = 40$. The value of Δt is the same as that used in GP-HNN.

IV. Convergence of F-HNNs

The main characteristic of HNNs is that they always converge. Nevertheless, projections and VUSs may change the good behaviour of HNNs. In order to prove the convergence of F-HNNs, it is worth highlighting two characteristics that F-HNNs share with HNNs:

- The energy function is a quadratic function defined inside the unit hypercube. Thus, it is upper and lower bounded inside the hypercube.
- From (19), the direction of minus the projection of the energy gradient points to neuron states with less energy. Moreover, the computation of the VUS ensures that the minimum in that direction is never exceeded. Consequently, the energy will be always reduced from one iteration to the next.

Therefore, F-HNNs must always converge to a point since the energy cannot be reduced unlimitedly. Finite precision calculus in computers may produce round-off errors preventing convergence when the network is very close to a minimum and neuron states change less than the round-off error. This can be solved with a tolerance value greater than the round-off error. This way, if neurons vary less than the tolerance, the network is assumed to converge. In this paper the tolerance, tol , is set to 10^{-4} for all algorithms.

V. Numerical Results and Discussion

This section compares different approaches using the NQP. As previously mentioned, for the NQP, neurons are usually organized in a matrix form. Nevertheless this does not invalidate the vector notation of previous sections. In fact, the matrix form aims only at making writing and understanding the different terms of the energy function easy. Therefore, it is important to remember that, although some equations of this section use two indices to refer to a specific neuron output, the set of all neurons will still be grouped into the column vector $\mathbf{v}(t)$ just as in previous sections.

The objective of the NQP is to distribute Q queens into a $Q\times Q$ chessboard in such a way that they could not attack each other. The energy function used to solve this problem is [10]:

$$E\left(\mathbf{v}(t)\right)=\frac{A}{2}\sum_{i=1}^{Q}\left(\sum_{j=1}^{Q}V_{ij}(t)-1\right)^2+\frac{A}{2}\sum_{j=1}^{Q}\left(\sum_{i=1}^{Q}V_{ij}(t)-1\right)^2+ \\ +\frac{B}{2}\sum_{j=1}^{Q}\sum_{i=1}^{Q}V_{ij}(t)\left(\sum_{\substack{1\le i-k\le N\\ 1\le j-k\le N\\ k\ne 0}}^{Q}V_{i-k,j-k}(t)\right)+\frac{B}{2}\sum_{j=1}^{Q}\sum_{i=1}^{Q}V_{ij}(t)\left(\sum_{\substack{1\le i-k\le N\\ 1\le j+k\le N\\ k\ne 0}}^{Q}V_{i-k,j+k}(t)\right). \tag{25}$$

The first term aims at allowing only one active neuron per column and, hence, only one queen in each column of the chessboard. Similarly, the second term tries to force only one neuron to be active in each row. The last two terms are focused on the diagonals of the chessboard. Whereas rows and columns must have exactly one queen each, diagonals can have one or zero queens. These terms are minimized in those situations.

The three approaches described in section III are studied in this section. The performance of all approaches was tested in terms of the number of iterations needed to reach equilibrium, the probability of reaching a good solution, and computational cost. These performance indicators were obtained using computer simulations for different numbers of queens. More specifically, 5000 different initial states were used for each quantity of queens ranging from $Q=4$ to $Q=16$.

Figure 3 shows the average number of iterations until an equilibrium state is reached. The differences between the algorithms are noteworthy. GP-HNN needs 10 times fewer iterations than S-HNN, whereas F-HNN needs 10 times fewer than GP-HNN and 100 times fewer than S-HNN. The good performance of F-HNN highlights the benefit of using a VUS. The high number of iterations of S-HNN is mainly due to the Simulated Annealing (SA) procedure since the system must be slowly “cooled”.

The probability of reaching a good solution of the NQP is very similar for all the approaches. Specifically, S-HNN reached a good solution 52.7% of the time, GP-HNN reached one 56.0% of the time, and F-HNN did so 54.0% of the time. These values are also the probabilities of reaching the global optimum. The energy function of (25) has been defined to be minimum for the valid/good solutions. Other stable states always have more energy. Therefore, the three techniques have very similar behaviours in terms of reaching the global optimum. It is worth noting that S-HNN has the worst behaviour in spite of using an SA procedure. This is due to two main causes. First, the SA procedure is not as good as it may seem initially. The “cooling” procedure needs too many iterations to converge with respect to the improvement in the probability of good solutions. Second, the mechanism of projection and clipping used by S-HNN for confining the neuron states into the constraints subspace produces severe instabilities. Although this procedure converges to a point, this point may be very different in two contiguous HNN iterations. For that reason the clipping is more and more severe with every new iteration, which forces the neural network to converge. The main problem of this procedure is that the convergence may be forced even if the neuron states are far from a good solution.

Finally, Figure 4 depicts some illustrative results of the computational cost of all the approaches. The three techniques were simulated on the same computer, an Intel Core 2 Duo processor T7500 working at 2.2 GHz and with 4 GB of physical RAM, using a prototype in MATLAB. As can be observed, S-HNN improves its performance with respect to Figure 3 and gets very close to GP-HNN. Therefore, although S-HNN needs many more iterations to converge, each iteration can be solved faster. Nevertheless, this fact does not suffice for S-HNN to be the best approach. F-HNN still has the best behaviour, reducing the time needed more than tenfold in comparison to the other techniques. Although this study was performed for the specific case of the NQP, it is not difficult to understand that F-HNNs maintain their good performance in other applications. Comparing GP-HNNs and F-HNNs, the main difference is the use of VUS. The faster response time of HNNs using VUS was proved in [12] and hence it is clear that F-HNNs will always outperform GP-HNNs. Regarding S-HNNs, the difference is how they project over the subspace and ensure that all neurons remain inside the unit hypercube. S-HNNs perform the projection and clipping procedure of Step 5 (described in Section III.C) iteratively, whereas F-HNNs use the orthogonalization explained in Section II.F. It is not easy to know which technique is faster, although for the NQP the fact that F-HNNs outperform S-HNNs is completely clear. Moreover, the projection and clipping procedure needs the selection of a tolerance which is not trivial and makes the method of orthogonalization more robust since it does not need any additional parameters.

VI. Conclusions

This paper has analysed two well-known alternatives for incorporating the strict satisfaction of linear constraints into Hopfield Neural Networks (HNNs) through the usage of subspace projections: the HNN with Gradient Projections (GP-HNN) and the Smith HNN (S-HNN). Both models have been optimized in their execution time by using more efficient projection methods. Moreover, a new P-HNN model has been proposed, called Fast HNN (F-HNN), that combines the efficient projection of the energy gradient with a variable updating step technique. Once the direction of movement is confined in the subspace of constraints, neuron states are modified so that the energy is reduced as much as possible. The joint usage of both concepts allows a faster convergence of the new neural network, reducing the required number of iterations significantly.

The advantages of F-HNN have been proven through a direct comparison with the other two proposals. The N Queens Problem has been chosen as a case study because this is a classic optimization problem and, besides, it has been widely used in the field of HNN. First, the numerical experiments have revealed that F-HNN needs far fewer iterations to reach the equilibrium state than the other two alternatives. Obviously this fact is motivated by the usage of the variable updating step, which is especially efficient once the energy gradient is projected to the constraint subspace. Aside from converging in fewer iterations, the probability of reaching a valid solution is kept almost identical compared with S-HNN and GP-HNN. In addition, it has been shown that F-HNN requires less computational time. With a conventional personal computer, the F-HNN model was able to solve the 16 Queens problem in less than four seconds, being up to twenty times quicker than the other two proposals.

Thanks to the mathematical foundations derived in this paper, and as a future work, the design of an optimized implementation of F-HNNs on a parallel platform, for example using modern multi-core processors, is immediately feasible. In order to calculate the next state of the neuron outputs, the most costly operation in the F-HNN algorithm is the matrix-vector product required to compute the gradient, which is easily parallelizable. The rest of the computations, including orthogonalization, also fit well in a parallelization context. As a conclusion, in each iteration all neurons can be updated simultaneously, calculating the projection matrix and the updating direction vector at once. It is worth noting that all neurons must be obtained before moving to the next iteration, and hence the lower the number of iterations, the faster the system response. With regards to this metric, again, F-HNNs exhibit the best results, justifying their interest with regards to HNN parallel implementation.

References

[1] D. Abramson, K Smith, P. Logothetis, and D. Duke, FPGA based implementation of a Hopfield Neural Network for solving constraint satisfaction problems, in: Proc. of Euromicro Conference, Vol. 2, (IEEE, Västerås, 1998) 688–693.

[2] C. W. Ahn and R. S. Ramakrishna, QoS provisioning dynamic connection-admission control for multimedia wireless networks using Hopfield Neural Networks, IEEE Transactions on Vehicular Technology 53 (2004) 106–117.

[3] S. Aiyer, M. Niranjan and F. Fallside, A theoretical investigation into the performance of the Hopfield Model, IEEE Transactions on Neural Networks 1 (1990) 204–215.

[4] D. Calabuig, J.F. Monserrat, D. Gomez-Barquero, and O. Lazaro, An efficient dynamic resource allocation algorithm for packet-switched communication networks based on Hopfield neural excitation method, Neurocomputing 71 (2008) 3439–3446.

[5] D. Calabuig, J.F. Monserrat, D. Gomez-Barquero, and N. Cardona, A delay-centric dynamic resource allocation algorithm for wireless communication systems based on HNN, IEEE Transactions on Vehicular Technology 57 (2008) 3653–3665.

[6] P. Chu, A neural network for solving optimization problems with linear equality constraints, in: Proc. International Joint Conference on Neural Networks, Vol. II (IEEE, Baltimore, 1992) 272–277.

[7] J. W. Daniel, W. B. Gragg, L. Kaufman, and G. W. Stewart, Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization, Mathematics of Computation 30 (1976) 772–795.

[8] M. Forti, S. Manetti, and M. Marini, A condition for global convergence of a class of symmetric neural circuits, IEEE Transactions on Circuits and Systems I 39 (1992) 480–483.

[9] J.J. Hopfield and D. Tank, Neural computation of decisions in optimization problems, Biological Cybernetics 52 (1985) 141–152.

[10] T.N. Le and C.K. Pham, A new N-parallel updating method of the Hopfield type neural network for N-queens problem, in: Proc. International Joint Conference on Neural Networks (IEEE, Montreal, 2005).

[11] K. Smith, M. Palaniswami, and M. Krishnamoorthy, Neural techniques for combinatorial optimization with applications, IEEE Transactions on Neural Networks 9 (1998) 1301–1318.

[12] P. M. Talaván and J. Yáñez, A continuous Hopfield network equilibrium points algorithm, Computers and Operations Research 32 (2005) 2179–2196.

[13] G. V. Wilson and G. S. Pawley, On the stability of the travelling salesman problem algorithm of Hopfield and Tank, Biological Cybernetics 58 (1988) 63–70.

Figures

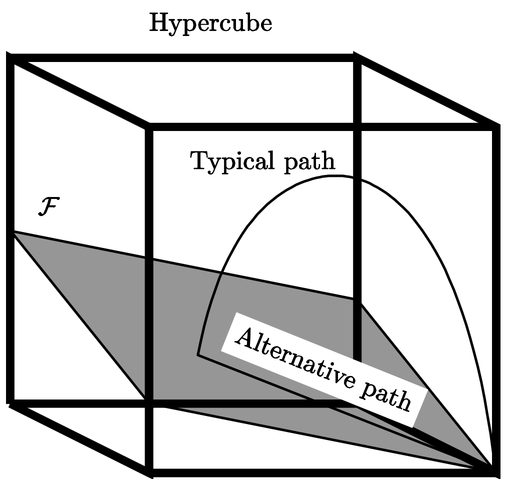

Figure 1. Representation of the subset $\mathcal{F}$ inside a hypercube, the typical path followed by an HNN, and the alternative path that belongs to $\mathcal{F}$.

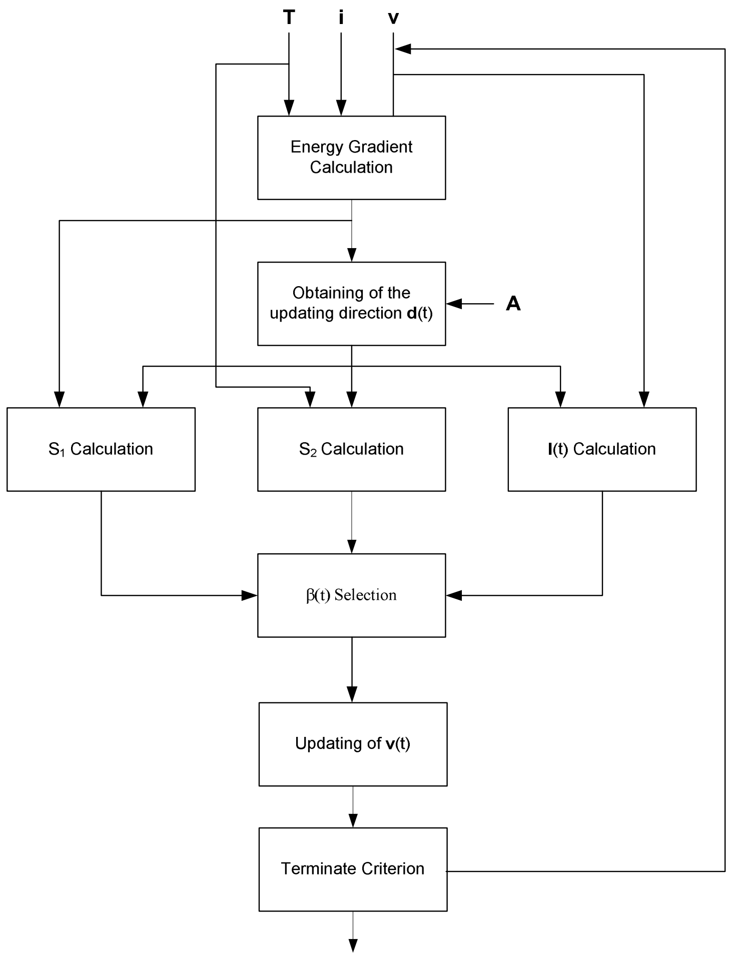

Figure 2. Flowchart representation of F-HNN

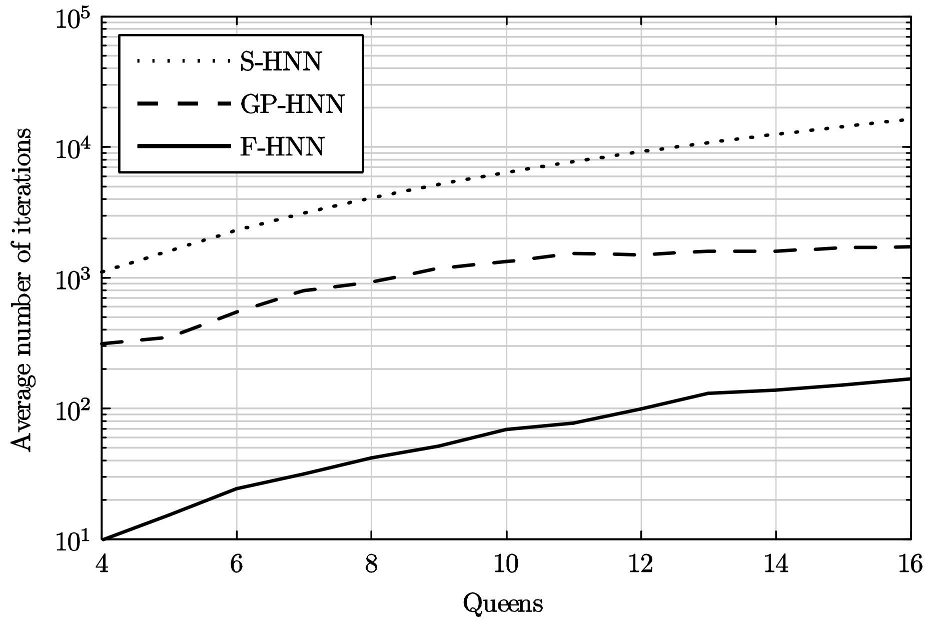

Figure 3. Average number of iterations of the three P-HNN alternatives

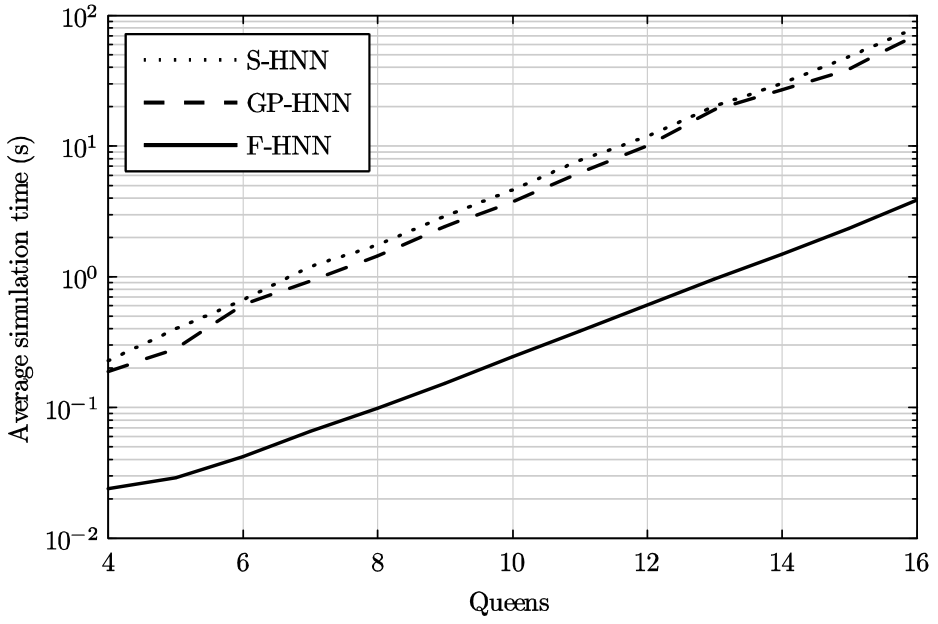

Figure 4. Average simulation time of the three P-HNN alternatives