A Hopfield Network for Multi-Target Optimization

Kevin Swingler

Abstract— **This paper presents a method for training a binary Hopfield neural network so that its energy function represents the fitness surface of an optimization problem with one or more target solutions. The main advantage of this method is that once the network has been trained, new solutions to a problem can be generated without reference to the original fitness function (which may take time to run). This allows the network to move from poor solutions to locally optimal solutions at speed.**

I. INTRODUCTION

Hopfield networks [1] are a type of neural network capable of storing a number of patterns distributed across the same weight matrix. Patterns are stored using one-shot learning and recalled by the presentation of a partial pattern, which causes the network to produce the pattern from its memory that is closest to the input stimulus. Such networks have also been used on optimization tasks such as the travelling salesman problem, but the network needs to be configured by hand to reflect the structure of the problem to be solved. In this paper, we present a method for training a Hopfield network to represent local maxima in search space so that multiple searches from arbitrary starting points may be quickly executed. This work is restricted to binary patterns, where units are either on or off.

In section II we introduce the Hopfield network and its role as an auto-associative memory and optimization method. In section III we present a method for training a Hopfield network so that its energy function reflects the fitness of a given pattern as a potential solution to an optimization problem.

Section IV presents some experimental results from three different types of task: discovering a single target pattern, discovering a set of target patterns, and discovering a target concept.

II. HOPFIELD NETWORKS

A Hopfield network [1] is a neural network consisting of simple connected processing units. In this work, the network has the following properties. Networks have a fixed size of n binary processing units. The values the units take are represented by a vector, U, of n binary values.

$$U = u_1 \dots u_n, \quad u_i = \pm 1 \tag{1}$$

Kevin Swingler is with the Institute of Computing and Mathematics, University of Stirling, Stirling, FK9 4LA. kms@cs.stir.ac.uk

In the following, we use $\leftarrow$ to indicate 'becomes' to avoid nonsensical equalities such as $i=i+1$, preferring $i \leftarrow i+1$.

The processing units are connected by directed weighted connections, with subscripts denoting direction from pre-synaptic unit to post-synaptic unit:

$$W = [w_{ij}] \tag{2}$$

where w_{ij} is the strength of the connection from unit i to unit j. Units are not connected to themselves, i.e.

$$w_{ii} = 0 \tag{3}$$

and connections are symmetrical, i.e.

$$w_{ij} = w_{ji} \tag{4}$$

A single pattern, P is a point in n-dimensional binary space defined by a vector of binary values:

$$P = p_1 \cdots p_n, \quad p_i = \pm 1 \tag{5}$$

A pattern is entered into the network by setting:

$$u_i \leftarrow p_i \forall i \tag{6}$$

Once the input pattern has been entered, the network is allowed to settle to an attractor state determined by the values of its weights. The unit update rule is

$$u_i \leftarrow \sum_{i \neq j} w_{ji} u_j, \tag{7}$$

following which the unit's value is capped by a threshold, θ, such that:

$$u_i \leftarrow 1 \quad \text{if } u_i > \theta$$
$$u_i \leftarrow -1 \quad \text{otherwise} \tag{8}$$

In this paper, we will always use $\theta = 0$.

The process of settling repeatedly uses the unit update rule of Equation (7) for each unit in the network until no update produces a change. At that point, the network is said to have settled.

Learning in a standard Hopfield network takes place by

the process of setting the pattern to be learned using Equation (6) and applying the weight update rule:

$$w_{ij} \leftarrow w_{ij} + u_i u_j \tag{9}$$

This is known as the Hebbian update rule.

With the above restrictions in place, the network has an energy function, which is a Lyapunov function, which determines the set of possible stable states into which the network will settle.

The energy function is defined as:

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} u_i u_j \tag{10}$$

Settling the network, by Equation (7) produces a pattern corresponding to a local minimum of E in Equation (10).

Hopfield networks have been used to solve optimization tasks such as the travelling salesman problem [2] but weights are set by an analysis of the problem rather than by learning. In the next section, we show how random patterns and a fitness function can be used to train a Hopfield network as a search technique.

III. Training Optimization Networks

In this section we describe a new method for training a Hopfield network for use in optimization problems. The procedure is summarized below and specified formally in the following section.

Candidate solutions are generated randomly one at a time. Each solution is given a score that reflects its quality as a solution and this score is used as a learning rate in the Hopfield network. Consequently, each pattern is learned with a different strength, which reflects its quality as a solution. The scoring mechanism is specific to the problem at hand and some simple examples are presented in section IV of this paper. After a number of patterns have been scored and learned, the network is settled by repeated application of Equation (7) and the resulting pattern is scored. The score of the settled pattern will increase as learning proceeds. Learning terminates when a pattern of suitable quality has been found. It may, however, be desirable to continue the learning process until a number of good solutions have been found. This ensures that more of the search space is learned.

We now define the procedure formally.

A. The Search Space

A single pattern, P is a point in n-dimensional search space defined by a vector of binary values as specified in Equation (5).

For the purpose of human readability in the examples that follow, P is arranged into an m x m grid where $m = \sqrt{n}$ and elements are displayed as black when $p_i = 1$ and white when $p_i = -1$.

Target patterns are one of two types:

- A specific set of patterns $T = \{P^1 \ldots P^t\}$, where t is the number of target patterns
- A concept such as symmetry, producing a number of equally perfect targets

In both cases, a candidate pattern is scored by comparison to the target pattern or concept.

B. The Fitness Function

We define a fitness function, $f(P)$, which returns a real valued score corresponding to the quality of the pattern P as a solution to the problem at hand. $f(P)$ has the quality that better patterns have higher scores. We implement three example fitness functions in this paper, all with a score between zero and one.

For each pattern stored in a Hopfield network, there are two local minima in the energy function. One is at the point representing the pattern itself and the other is at the point representing the inverse of the pattern (to invert a pattern, toggle all the values so that -1 becomes 1 and 1 becomes -1). If the target pattern is one of a set (as in examples 1 and 2 below) then the fitness function is simply the hamming distance between the candidate pattern and the target. This simple measure does not work, however, due to the symmetrical nature of the network attractors. The fitness function is altered so that the inverse of any pattern and the pattern itself both score the same, that score being the maximum of the two. The modified hamming distance is used to calculate the fitness of a pattern P given a target T

$$f(P \mid T) = \left| 2\left(\sum_i \frac{\delta_{ti,pi}}{n}\right) - 1 \right| \tag{11}$$

where $\delta_{ti,pi}$ is the Kronecker delta function between pattern element i in T and its equivalent in P.

C. The New Weight Update Rule

This work utilizes a modification to the Hebbian rule so that

$$w_{ij} \leftarrow w_{ij} + f(P) u_i u_j \tag{12}$$

where $f(P)$ is the score function for the given pattern, as described above. This has the effect of learning high scoring patterns more than low scoring patterns.

D. Search Algorithm

The search algorithm generates random patterns in P and calculates $f(P)$ for a chosen scoring function. Each element p_i is set to either 1 or -1 with equal probability. Early results suggest that changing this distribution is detrimental to the learning process.

The pattern is loaded into the network units, U, and the

learning rule in Equation (12) is applied to the weights, W. The network is then settled by repeated application of the unit update rule in Equation (7) and $f(P)$ is re-calculated. This process is repeated until $f(P)$ produces a score of 1 (or some pre-determined limit is reached).

Note that the patterns being generated are always random, their probability distribution is not affected by W. That is to say that it is the value of $f(P)$ that is produced before the network settles that is used for learning. The settled pattern is not learned, and early experiments have shown that doing so is detrimental to the performance of the algorithm.

Due to the fact that patterns and their inverse are equally likely to be found – as described above – it may be necessary to score both the solution and its inverse to find the true solution.

Multiple solutions may be found by allowing the search algorithm to continue to run after the first solution is found, hence learning other high scoring patterns.

E. New Searches with an Existing Network

Once a number of solutions have been found, the energy function of the network will have a number of local minima, all of which are attractors to which the settling process will move. By presenting a new pattern and allowing the network to settle, the closest solution (or locally optimal solution) will be found. This is done by settling the network and does not require the fitness function $f(P)$ to be evaluated. This offers a useful fast search method for problems with the following qualities:

1. More than one pattern has an optimal (or near optimal) score;
2. High scoring patterns need to be found repeatedly from different starting points;
3. The preferred pattern is that which is closest to the starting point.

IV. Experimental Results

A. Example 1 – Finding a Single Pattern

In this example, a 36 pixel vector was used to represent a 6 x 6 binary image. The single target pattern P is shown below in Fig. 1.

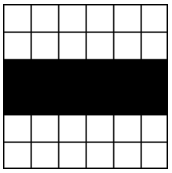

Fig. 1. The first example target pattern.

Finding this one pattern from the 2^{36} possible binary patterns is a relatively simple task for a genetic algorithm [3] or estimation of distribution algorithm [4]. In this section, we compare simple implementations of these two approaches with the Hopfield approach. Our aim is to discover whether or not the new approach is comparable to basic implementations of these popular search methods. In all cases, $f(P)$ was calculated as the normalized hamming distance between a candidate pattern and the target.

1) A Simple Genetic Algorithm

Genetic algorithms (GAs) are optimization techniques that maintain a population of candidate solutions which are scored against a fitness function. Higher scoring candidates are combined and randomly altered (mutated) to produce a new generation that is closer to the target than previous solutions. The GA used for comparison in this study maintained a population of 100 candidate solutions, each of which were scored using $f(P)$. Fitness proportional selection was used to select 10 patterns from each generation. Uniform crossover with a rate of 100% was used with a mutation rate of 5%. The offspring replaced the lowest scoring 10 candidates in the population.

2) A Simple Estimation of Distribution

Optimization using estimation of distribution algorithms (EDAs) looks for solutions by maintaining a probability distribution over the pattern elements from high scoring members of a population of candidate solutions. New solutions are generated by sampling from these distributions. In this case, a vector of probabilities, $R = \{r_1 \ldots r_{36}\}$, was maintained and patterns were generated with pixel values set to 1 with a probability r_i and -1 with probability $(1- r_i)$. That is to say that new generations of solutions are sampled from the evolving distributions in R:

$$P(p_i = 1) = r_i \tag{13}$$

A population of 100 solutions was generated based on these probabilities and each pattern was scored using $f(P)$. The probabilities in R were then updated based on $f(P)$ for the best 10 scoring patterns. The update rule is:

$$r_i \leftarrow r_i + f(P) \text{ if } p_i = 1$$
$$r_i \leftarrow r_i \text{ otherwise} \tag{14}$$

The probabilities, r_i are then normalized so that they cover the range 0…1. Sampling was done with a simple rejection algorithm.

3) Hopfield Network

The Hopfield network used to build the energy function for the search had 36 units connected by 1260 weights. Random patterns were presented to the network and learning was performed as described above. Note that in this approach, there is no population of candidate solutions and no method of generating new, improved candidates. Stimuli are always random and are learned one at a time.

4) Results

Each of the search techniques was run 150 times with the

pattern from Fig. 1 as a target. The number of calls to $f(P)$ made before the target was found were counted for each trial. Table 1 below shows the mean and standard deviation search length for each method over the 150 trials. The Hopfield network was quicker than the other two approaches.

TABLE I
SEARCH LENGTHS FOR EXAMPLE 1

Method	**GA**	**EDA**	**Hopfield Network**
Mean Search Length	2217	1410	1190
Standard Deviation	416	404	596

Average number of score comparisons made by different search techniques over 150 trials searching for the pattern in Fig. 1 above.

Undoubtedly, an expert in the use of the other two methods would be able to reduce the search time for each. We hope to demonstrate only that our new technique is sufficiently quick on a simple search that it is worthy of further investigation. We aim to say nothing about whether it is generally better than the other techniques.

B. Example 2 – Multiple Targets

This example investigates the behavior of the Hopfield network when the search target consists of a set of patterns, rather than a single pattern. In this case, we use binary pixel representations of the digits 0 to 3, as shown in Fig. 2.

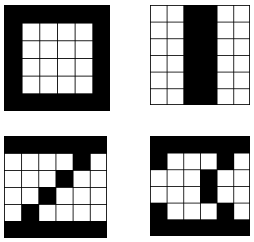

Fig. 2. Binary pixel images of the digits 0, 1, 2, and 3.

The learning algorithm runs as follows:

1. Generate a random pattern across the network units;
2. For each target pattern:
   2.1. Score the random pattern using Equation (11), the modified hamming distance between the pattern and the target;
   2.2. Update the weights in the network using Equation (12);
3. Allow the network to settle by repeated application of Equation (7);
4. Score the settled network pattern against each target as in step 2.1, but do not adjust the weights;
5. Stop when all targets have been found at least once.

To test the resultant network, distorted patterns were presented and the network allowed to settle. Fig. 3 shows some results. Note that the network has only ever seen random patterns, it has never been presented with the stimuli in Fig. 2.

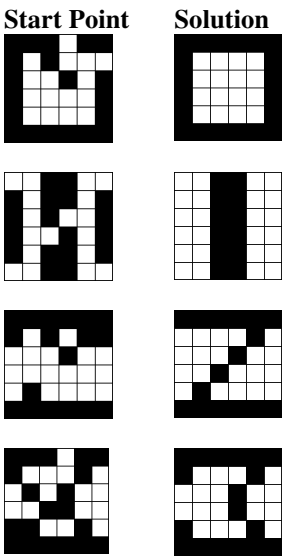

Fig. 3. Binary pixel images of distorted digits 0, 1, 2, and 3 on the left and their reconstructed solutions on the right. The network was not presented with the patterns in this table during learning.

Each of the outputs on the right hand side of Fig. 3 score 1 when compared to one of the target patterns. During training, the network is presented with random patterns where each pixel has an equal chance of being on or off, so the average score of the training patterns is 0.5. The training data almost never contains a pattern with a score of 1. However, the network settles on a pattern that maximizes the score against one of the target patterns – the one closest to its starting point.

C. Example 3 – Finding Symmetrical Patterns

In these trials, the fitness function $f(P)$ is a measure of the degree of vertical symmetry of an image. Images where the left hand side is a mirror image of the right hand side score 1. The score becomes closer to zero the larger the number of non-symmetrical pixels there are across this vertical split.

This trial is designed to demonstrate some of the qualities of the Hopfield network search algorithm and is not a comparison with other techniques. The purpose is to show the network learning a concept (symmetry) based on scores given to randomly generated patterns. Unlike the traditional use of a Hopfield network, no known examples are shown to the network. Indeed, it is common that the network will arrive at the correct function without having been presented with a perfect scoring example.

Patterns were generated in a 6 x 6 image of binary pixels as described above. There are 2^{36} (68,719,476,736) possible patterns in such a matrix. Of those, there will be one vertically symmetrical pattern for every possible pattern in one half of the image. There are 2^{18} (262,144) such half patterns, representing 0.00038% of the total number of possible patterns.

In this trial the network was allowed to learn until ten different symmetrical patterns had been found. At that point, the learning process was terminated and the network was

tested with a set of local searches. A local search starts from a random pattern and moves towards a higher scoring pattern by allowing the network to settle to a local minimum of the energy function described by the learned weights.

Fig. 4 shows the result of testing the learned network from five different starting patterns. The score of the starting pattern and the settled pattern are given. The network settling process is described above, but it should be noted that during this search, no new learning takes place and the fitness function is not evaluated.

The inverse pattern problem does not affect the fitness function in this example as any pattern has the same degree of symmetry as its inverse.

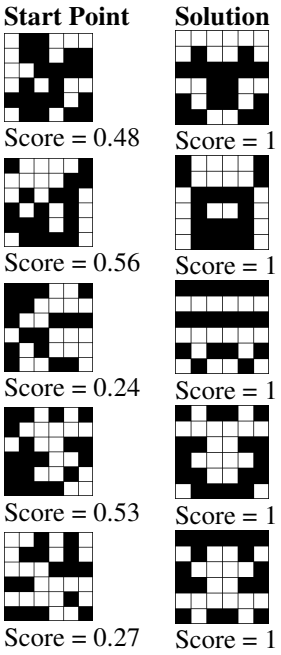

Fig. 4. Binary pixel images of random stimuli on the left and resultant generated images on the right.

V. Analysis

This section presents some observations on the method and its results. The energy function of the network in Equation (10) is globally minimized when the units' activation values are consistent with the weight matrix that connects them. The process of settling the network finds the pattern of activation that is most consistent with the weights of the network. The weights can be viewed as constraints between units, and the act of settling the network is the process of satisfying those constraints. Weights can be seen as an extension to the idea of an EDA. Rather than modeling the probability (or importance) of single element values in a pattern, we are modeling the conditional importance of the value of each element given the value of all other elements. The learning process derives the constraints from random patterns, so does not need to solve the problem of generating new candidate patterns from an evolving population. There is no combination nor mutation and no selection nor sampling.

The patterns used during training must consist of elements drawn from an even binary distribution. Each pattern can be expected to have around half of its elements set to 1 and half to -1. This does not restrict the distribution of element values in the target population but further work is required to produce an algorithm that works when the training examples do not have an even distribution.

It is clear that the capacity of the network will limit the number of concurrent local minima possible and consequently the number of distinct solutions the network can store. Using Hebbian learning, the capacity of a network is shown by [5] to be $n / (2 \ln n)$ for random patterns, but that number can be higher for certain sets of patterns. The symmetrical patterns illustrated here are a good example, the 36 unit network being able to store over 130 such patterns.

Conclusions

We have shown that a Hopfield network presented with nothing but random patterns and a fitness score for each pattern is able to learn target patterns or concepts that it has never directly seen. Such a network allows new good solutions to be generated from existing poor solutions without the need for reference to the fitness function.

This early work has many limitations, which will be addressed in future work. Patterns are binary in nature and need to be extended to a continuous space. We have noted that training pattern elements must be drawn from an equal distribution of ±1, which means that any attempt at evolving the training distribution towards a target distribution is detrimental to the process. This requires further investigation. We also described the problem of patterns and their inverse both being attractor states in the network and presented a simple adjustment to the hamming distance fitness function. The impact of this problem on other evaluation criteria needs further investigation.

Acknowledgments

Thanks to Bruce Graham and David Cairns for their comments on this paper.

References

[1] Hopfield J.J. Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 79, No. 8, 1982. pp. 2554-2558.

[2] J. J. Hopfield and D. W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics*, vol.52, pp.141-152, Springer. 1985.

[3] Goldberg, David E. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley. pp. 41. 1989.

[4] Larrañaga, Pedro; & Lozano, Jose A. (Eds.). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Kluwer Academic Publishers, Boston, 2002.

[5] R. J. McEliece, E. C. Posner, E. R. Rodemich, and S. S. Venkatesh. 1987. The capacity of the Hopfield associative memory. *IEEE Trans. Inf. Theor.* 33, 4, 1987. pp 461-482.