Hopfield Networks and their Applications

Miniproject for course : Machine Learning for Physicists by Prof. Florian Macquedart

Rajarsi Pal
rajarsi14p@gmail.com

Indian Institute of Technology, Madras
Institute-mail: ph20c032@smail.iitm.ac.in
`https://github.com/pafloxy/Hopfield_MlPh`

ABSTRACT

Here I have explored and implemented some elementary application of Hopfield networks, particularly in pattern retrieval and optimisation. For pattern retrieval, images from MNISt data were stored into the network and the efficiency of the network in retrieving those were experimented upon. For optimisation, a network to solve N-rook's problem was implemented and its possible extension to solve the Stable-matching problem was explored.

Keywords: Recurrent Networks, Pattern retrieval, Optimisation

INTRODUCTION

Hopfield networks came up originally as models for content addressable memory, referring to their ability to regenerate corrupted form of a stable pattern. However, their use in optimisation has also been .

Network architecture and working

For our purpose we will use a simple fully-connected recurrent network, where every neuron affects every other neuron on their next time step through their weights. Neurons can be updated either synchronously(all at once) or asynchronously(one at random).

Say the activation of a particular neuron is $X_i(\mathbf{X})$ and corresponding bias $b_i(\mathbf{b})$, then the energy of configuration is given by $E(X) = -\mathbf{X^T W X} + \mathbf{b^T X}$, where $\mathbf{W}$ is the weight matrix. Now any neuron can be updated as, $X_i(t+1) = g(-\frac{\partial E}{\partial X_i}) = g(\mathbf{W_i}.\mathbf{X(t)} - \mathbf{b})$, $\mathbf{g}$ being the activation function.

This particular update strategy guarantees that if $\mathbf{W}$ is symmetric and has diagonal elements zero then **Energy** decreases monotonically with iterations, and as there are only a finite number of configurations available the network configuration must converge to an energy minimum.

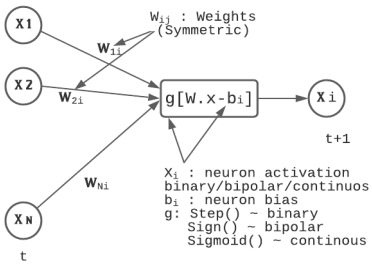


Figure 1. Network structure

PATTERN RETRIEVAL

The main idea of using Hopfield nets for pattern retrieval follows from fact that in presence of multiple energy minimums in configuration space, any arbitrary initial configuration must converge to its nearest energy minimum available. Basing on this, the weight matrix can be prepared such that the patterns to be stored in the network corresponds to this local Energy minimums.*see [JHo82]*

Implementation

For this project, I have trained the network to store images of handwritten digits available from MNISt data as stored patterns. Standard MNISt images $(28\times 28p, (0,255))$ were recast into bipolar array of form $(28\times 28p, \{+1,-1\})$, where each of 784 pixels corresponds to a distinct neuron.
To store configuration $\{C_1, C_2, C_3, ..\}$, weight matrix was set as $W = \mathbf{C_1}.\mathbf{C_1^T} + \mathbf{C_2}.\mathbf{C_2^T} + ...$
For training purpose the biases were all set to zero and the weight matrix was normalised by a constant factor, the **Sign** function was used as activation.
For an easy visualisation of the update process, the current configuration and corresponding energy were plotted iteratively. Hamming distance, D_h was used as the degree of similarity between the current configuration against either of the stored patterns.

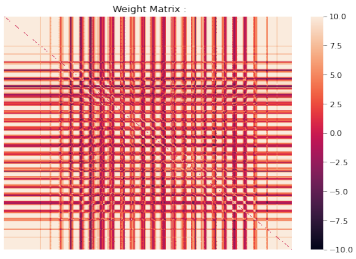


Figure 2. Weight matrix for *E.1*

Experiments

E.1 In my first go, I stored all 10 digits (picked randomly from MNISt data) into the network. From a theoretical perspective*(**rather ambitiously)*, I expected the network to retrieve the stored version of the digit when initiated with a variant of the same digit, for e.g; when initiated at a randomly chosen **7**, the network should return the **7** it was trained with.
However when I tried the same with my implementation, far from expectation the network ended up converging only to an distorted **3** or **1** *(**had least D_h)* in most cases or something unrecognisable in others. To remove this behaviour, I also tried strengthening the weights by storing each pattern twice, however no significant improvement in retrieval was observed. Neither using a probabilistic update rule (** *with low temp.*) yielded anything better.

E.2 Next I tried, storing only two distinct digits(**each was stored twice*) along with some random patterns. This time significant improvement in the retrieval process was observed, though it varied somehow on the stored patterns. Similar retrieval efficiency persisted even when three/four distinct patterns were used.

Exploring retrieval issues

i. The main reason for this mis-convergence of the network configuration can be attributed to formation of spurious energy minimum, which are formed as a result of extensive overlap of basin of attraction of the various stored configurations. For eg.: the distorted **3** in **E.1**.

ii. Observing the behaviour of Hamming distance plots for the various trials, a dependence of spurious minimums to closeness in Hamming space can be inferred. D_h plots for **E.1** showed that the stored patterns were confined to a region of $D_h = 0.2$ while having maximum separation of $D_h = 0.03$ between some of them. Whereas for **E.2**, the patterns were separated by large margins of D_h (** *random patterns are more likely to have higher D_h separations.*), which resulted in a much better retrieval.

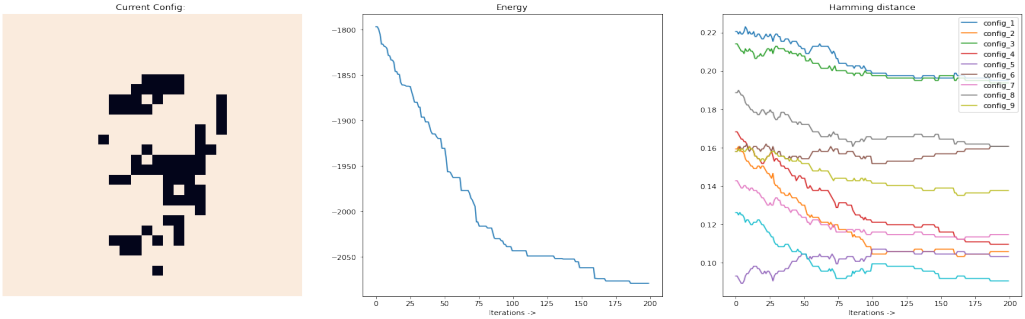


Figure 3. Retrieval for **E.1**

Why Hamming distance matters ?

Since the all neuron values are bipolar we can express the inner product of two distinct configurations**A,B** as $\mathbf{A^T B} = N - 2\mathbf{D_h}(\mathbf{A},\mathbf{B})$. Thus for pattern retrieval, $\frac{\partial E}{\partial X_i} \approx \mathbf{C_{1}}_i(\mathbf{C_1^T X_0}) + \mathbf{C_{2}}_i(\mathbf{C_2^T X_0}) + .. \approx \mathbf{C_{1}}_i(N - 2\mathbf{D_h}(\mathbf{C_1},\mathbf{X_0})) + \mathbf{C_{2}}_i(N - 2\mathbf{D_h}(\mathbf{C_2},\mathbf{X_0})) + ..$ where $\mathbf{X_0}$ is the current configuration. This signifies that attraction of $\mathbf{X_0}$ to a particular configuration is moderated by their separation in D_h.
From here it seems, that D_h could be a potential candidate to probe the energy landscape of configurations and the boundaries of attraction basins of the stored patterns. However, this claim would require more detailed investigation and was not followed by me any further. *see [RRo96]*

OPTIMIZATION

A generic optimisation problem involves finding optimal values for a set of parameters such that a given cost function is minimised(** or maximised) while respecting some constraints imposed on the parameters. Since the dynamics of hopfield network guarantees a monotonic decrease(** increase) in its energy function, the challenge lies in setting up the weight matrix and biases such that it resembles the cost function of the original optimisation problem.*see [JD85]*

N-rook's Problem

The N-rook's problem involves placing N rooks on a $N \times N$ chessboard such that no rook can take out another rook in a single move. Since the only valid move for a rook is to move straight along a row or a column, any configuration where no two rooks are placed in the same row or column qualifies as solution to the problem.

Implementation

For my implementation, a $N \times N$ grid of cells were considered, each cell representing an unique neuron. The activation of the neurons were binary $\{1,0\}$, with 1 and 0 representing occupied and unoccupied respectively. The **Step-function** was used as activation.

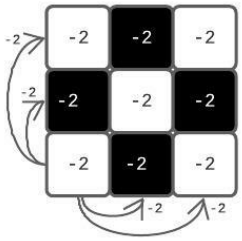


Figure 4. Biases & Weights for a 3x3 board

The choice of weights and biases can be set from the stability criterion. Since only one neuron in a row(*column) should be active at a time, an activated neuron must inhibit other neurons in the same row(*column). Setting the weight of inter-row(*column) connections to **-2** and all biases to **-2** ensures that a neuron doesn't activate when any other neuron in its row(*column) is active, as even a single active neuron would push the weight contribution below the bias. For my code, a random initial configuration was chosen and updated asynchronously, the configuration and energy were plotted iteratively for visualisation.

Results

The network was able to reach a stable configuration in most trials, and this behaviour persisted even for chessboards of large sizes(** tested up to 70×70 grids). However, the time taken for convergence increased considerable with size (N) the scaling was almost quadratic(N^2). But when the network was initiated with a all-zero configuration(**none occupied) much faster convergence was observed, whereas initiating the network with all-one configuration(**all occupied) resulted in similar convergence behaviour as in the case of random initial configuration.

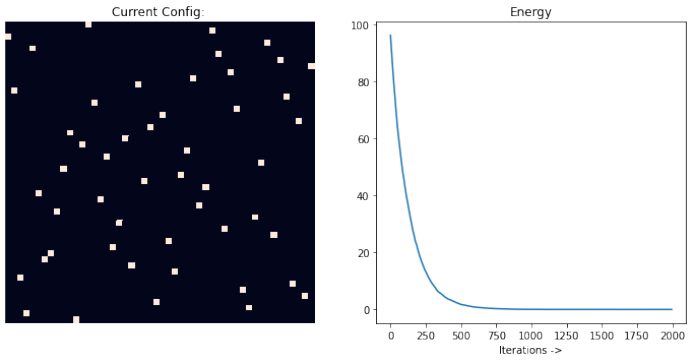


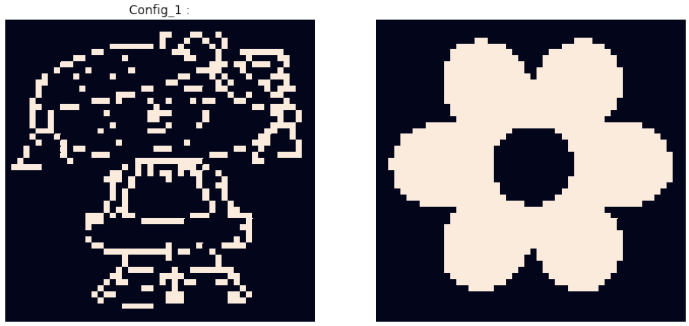
Figure 5. Convergence for N rook's implementation

Further Explorations

Basing on the efficient implementation of the N rook's problem I tried extending it to solve the Stable-Matching Problem, which aims at finding an optimal one-to-one matching between two sets of distinct individuals(** say boys & girls) such that there are no **rogue couples** based on their preference ,*see [TLe]*.

I tried implementing the same in a manner similar to the N-rook's problem, for matching of size N(** N boys and girls) we can consider an $(N \times N)$ grid *indexed i,j* where each cell represents a neuron, which if on would denote a matching between boy_i and $girl_j$. Now setting up the weights as in N-rook's automatically prohibits any matching that is not one-to-one, next I tried encoding their respective preferences into the bias of corresponding neuron. However problem arises as both parties have their preference order for every other member of the other party whereas in the network I could only use one scalar value to modulate the bias. Though schemes like adding/multiplying the two preferences were tried, but none of them yielded a stable configuration.

FUNNNY INSTANCES



(a) *Stored patterns*

(b) *Stable pattern formed*

Figure 6. Girl & flower

ACKNOWLEDGMENTS

I would like to thank Prof. Florian Marquardt for providing me a chance to do this project and his guidance throughout the length of the course. I would also like to thank my colleague Debanjan Dutta for his help in resolving programming related issues and giving suggestions on the overall structure of the project.

REFERENCES

[JHo82] J.Hopfield. “Neural Networks and Physical Systems with Emergent Collective Computational Abilities”. In: *Proceedings of the National Academy of Sciences 79(8):2554-8* (1982). DOI: 10.1073/pnas.79.8.2554.

[JD85] J.Hopfield and DW.Tank. “Neural Computation of Decisions in Optimization Problems”. In: *February 1985,Biological Cybernetics 52(3):141-52* (1985). DOI: 10.1007/BF00339943.

[RRo96] R.Rojas. *Neural networks: A Systematic Introduction*. Springer-VerlagBerlin, Heidelberg, https://www.springer.com/gp/book/9783540605058, 1996.

[TLe] T.Leighton. “Stable Matching Problem”. In: *Mathematics for Computer Science MIT OpenCourseWare https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010* ().