Assignment One: Hopfield Networks

COMP596: Brain-Inspired AI

Solim LeGris

February 22, 2021

1 Part I - Understanding Hopfield Networks

1.1

- Processing units: A Hopfield network is one in which there is a set of N fully connected processing units with symmetric weights whose task is to store binary patterns.
- State of activation: The state of activation at time t for any unit i is $a_i(t) \in \{0, 1\}$.
- Output function: The output function of a unit in a Hopfield network is a threshold function that depends on the activation with respect to some θ_i. If the difference between the activation and the threshold is positive, then the output is 1, else it is -1 as defined by $o_i(t) = \{_{-1 \quad otherwise}^{1 \quad if \quad net_i(t)-\theta_i>0}$.
- Pattern of connectivity: The weights are symmetric such that for all i, j $W_{ij} = W_{ji}$ and for all i, j $W_{ij} = 0$ if $i = j$.
- Propagation rule: The propagation rule is defined by a linear sum of the product of the weights and outputs (also referred to as the states) at time t-1 or $net_i(t) = \Sigma_j W_{ij} o_j(t-1)$.
- Activation rule: The activation of a unit at time t is $a_i(t) = o_i(t)$. In a Hopfield network, the activation of a unit is identical to its output and the network is updated asynchronously.
- Learning rule: The learning rule can be defined by a gradient descent where $\Delta W = \eta o_i(t) o_j(t)$ is the partial derivative of the energy function with respect to a synaptic weight and $\Delta\theta_i = -\eta o_i(t)$.
- The environment is a set of binary patterns that correspond to memories that need to be stored and retrieved.

1.2

The energy function of a traditional Hopfield network is defined by $E(o(t)) = -\sum_{i<j} W_{ij} o_i(t) o_j(t) + \sum_i \theta_i$. It is called an energy function because it was inspired by the Ising model from physics. The dynamics of a Hopfield Network are such that as we let it evolve through time, starting from any arbitrary state it will evolve to a local energy minimum. By decreasing the energy of a pattern, we are effectively transforming it into an attractor state which is what allows the network to store the memory. The net learns these attractor states by learning the appropriate weights. Upon recall, the network's initial state will converge to the attractor state for the stored pattern that is closest to the one presented thereby leading it to recall.

1.3

The partial derivative of the energy function with respect to a synaptic weight is defined by $\Delta W = \eta o_i(t) o_j(t)$.

1.4

The derived learning rule in the previous question is a form of Hebbian learning. Hebbian learning is the process by which neurons that fire together wire together. Learning occurs as a result of two neurons' paired activity. In the case of the Hopfield Network, the weight between two units increases in value if the state values i and j are positive, whereas it decreases for state values where i or j has a negative activity.

1.5

The equation to be used would be $w_{ij} = \eta \sum_\mu \epsilon_i^\mu * \epsilon_j^\mu$ η is the learning rate and ϵ^μ the μ^{th} pattern to be stored in the network.

1.6

The energy landscape can be thought of as changing through time as the Hopfield network learns patterns. The network learns weights that cause the landscape to evolve towards one of peaks and valleys, where the valleys are local minima or attractor states that allow recall of the patterns they attract. The analogy of peaks and valleys is illustrative in that we can think of the energy landscape as these peaks and valleys and any given state of the network as some position within that landscape. When the network has learnt a local minimum within that landscape and when presented with some pattern at a later time, it will recall the pattern by lowering the energy of its states from an initial state representing the pattern leading it to the closest local minimum or valley.

1.7

Presented with the complement of a stored pattern, the network will return the complement of the stored pattern. Although this is somewhat unexpected behaviour, one can conclude that network in fact also learns the local minimum of the complement of every pattern it stores when considering the equations in Questions 1.2 and 1.5. Consider some pattern o_0^μ and its complement o_1^μ. Every 1 in o_0^μ is a -1 in o_1^μ and every -1 a 1. The weights that minimize the energy function of o_0^μ are the same weights as the ones that minimize the energy function of o_1^μ as a consequence of our learning rule. This was tested in the code using randomly generated patterns and the expected result was obtained.

2 Part II: Designing your Hopfield Network

2.1

In the constructor,

- `n` is the number of nodes in the network used mostly in `for loop` or list comprehension.
- `state` is a vector of length n representing the current state of every unit in the network initialized to 0.
- `thresh` is a vector of length n representing the thresholds for every unit in the network to determine their activations as a function of their inputs. It is initialized to 0.
- `W` is an $n * n$ matrix initialized as a 0 matrix representing the weight connectivity of the units in the network.
- `eta` is the learning rate of the network which normalizes the weights. It is initialized to 1/n.

2.2

The one-pattern-at-at-time learning rule function updates the weights by computing the product of the i^{th} node's activity and all j nodes' activities where $j > i$. Each resulting vector represents weight updates W_{ij} for all j and the current i^{th} node in the for loop. This vector is summed with the i^{th} row and i^{th} column in the weight matrix `W` and stored at these respective positions. Naturally, the diagonal of the matrix will be a zero diagonal in this case since it was initialized as such and is never changed.Thresholds are computed by subtracting the current pattern to previously stored thresholds resulting in the threshold vector for every unit in the network.

2.3

The mass pattern storage storage learning function makes use of matrix operations to optimize computations. It simply computes the vector dot product of its input, the images, with the transpose of itself. This results in an $n * n$ matrix representing the weights summed across images. The diagonal is then zeroed in order to respect the constraints imposed on Hopfield networks. The image vector is summed and negated to be stored as the threshold vector for every unit in the network.

2.4

The python notebook as submitted is set to run on the assignment guidelines but other code cells were included to showcase additional work done in the aim of further understanding Hopfield networks. An `image_preprocessing` flag when set to `True` will allow the code to run some functions on the

images. This was done to try and find digit images within the MNIST dataset that were "structurally" far in an attempt to increase the accuracy of the network and force a certain distance between local minima in the energy landscape, thereby avoiding the problem of averaging across images. The improvements were rather small and the net's performance did not improve significantly as showcased by the convergence from noisy and combined patterns results.