Hopfield Nets and Self-Organizing Maps

Jeevesh Juneja

November 18, 2020

1 Hopfield Networks

Hopfield nets are a special type of Neural Nets which were designed to be used to store binary sequences. As they are stored, so they must be retrievable. The ingenuity of Hopfield nets lies in this mechanism of retrieval and storage.

1.1 Architecture

Hopfield network can be formally described as a complete undirected graph $G = (V, f)$, where V is a set of McCulloch–Pitts neurons and $f : V^2 \to R$ is a function that links pairs of units to a real value, the connectivity weight. The weights w_{ij} typically have the following restrictions :

- $w_{ii} = 0, \quad \forall i$
- $w_{ij} = w_{ji}, \quad \forall i, j$

In total, we used $V^2 - V$ weights to represent a Hopfield net with V nodes. But these $V^2 - V$ numbers will now allow us to store a exponential(in V) number of binary sequences, each of which is of length V.

1.2 How it works – The Update Rule

Each node is set to an initial state, a sequence of update operations is applied to the graph as a whole, repeatedly till the states of the nodes converge. The initial state acts as a key for retrieving the final state. The binary sequences are represented using $+1, -1$ rather than $0, 1$. The update rule is given as :

$$s_i^{new} \leftarrow \begin{cases} +1 & \text{if} \sum_j w_{ij} s_j^{old} \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases}$$

This update rule can be applied to all the states, in parallel(synchronous updates) or one by one(asynchronous updates) to each one of them. The θ_i's are the thresholds corresponding to the i^{th} node in the graph. To see what actually happens during the update, observe that if w_{ij} is positive, s_j with come with its sign in the

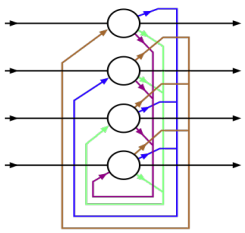

Figure 1: *A Hopfield Net with 4 neurons. The black lines represent input(initial state) and output(final state) of the neurons. Each neuron is connected to every other.*

sum, and thus will encourage s_i to have the same sign as itself. Next, when s_j is updated, since $w_{ij} = w_{ji}$, s_i will encourage s_j to have the same sign as itself. Hence, they both can be said to *attract* each other. We can also check if w_{ij} is negative, s_i will encourage s_j to have the sign opposite sign as itself, and vice versa. Hence, the *repel* each other.

1.3 Hebbian Learning

We saw in the previous section, how if w_{ij} is positive, both s_i and s_j will try to have same sign, and if it is negative, they will try to have opposite signs. Since, the learning problem involves figuring out the weights given the data, it makes sense, to set the weight w_{ij} as positive if in our data(i.e., the sequences we want our model to remember), the bits at the i^{th} and j^{th} positions have the same value often, and set it to negative otherwise.
Similar thing is observed in the brain, among the neurons, and is termed as the *Hebbian Principle*. Precisely, it states :
"Neurons that fire together, wire together. Neurons that fire out of sync, fail to link"
Mathematically, the rule is :

$$w_{ij} := \frac{1}{n} \sum_{\mu=1}^{n} \epsilon_i^\mu \epsilon_j^\mu$$

where ϵ_i^μ represents the i^{th} bit from pattern μ. Note that the bits of the patterns in our data are changed from $(0, 1)$ to $(-1, +1)$ respectively, before plugging into above formula.

1.4 Application

The above networks, can be used for storage/retrieval purposes, as shown above. Moreover, with the appropriate update rules the Hopfield networks can become equivalent to Transformer networks of [2], as shown in [1] .

2 Self Organizing Maps

Self Organising Maps(SOMs) is a method to map a grid in 2-D/3-D into the space of data, with a regularisation that constraints the mapping to preserve the distances to its neighbors in the grid. It can be seen as an unsupervised clustering problem, where each cluster corresponds to a node in a grid and nearby(or neighboring) nodes in the grid, correspond to nearby clusters. In more mathematical, it preserves the topological properties of a grid while clustering. For clarity see how the transformation of a random grid in Figure 2, looks like a homeomorphism of the initial space.

2.1 Architecture

Consider an n-dimensional space from which we sample our data. We know, the data comes from a distribution which has clusters, and our aim is to figure out those clusters. We construct an imaginary grid, of some size, say, mXm. Each of the m^2 nodes is assigned a "weight vector" which is n-dimensional. Hence, the weight vector lies in the same space as data. The weight vectors can be initialised randomly from the data-space or can be picked randomly from the subspace of our data-space whose bases are the first few principle components. Now we will try to tune our weight vectors to fit our data.

2.2 Learning the Weight Vectors

Algorithm 1, below shows the procedure of how the weight vectors of the grid are tuned. We pick a data point, and find the nearest weight vector to it(Lines 6-11). It is called the Best Matching Unit(BMU). The BMU can be thought of as the cluster to which the data point actually belongs.

Then comes the main step which updates the weight vector of BMU and its neighboring nodes(Line 14). We find the element wise difference between the data vector $D(t)$ and the weight vector. This is scaled by two factors, θ and α, and the weight vector is moved in the direction of this scaled difference. θ captures

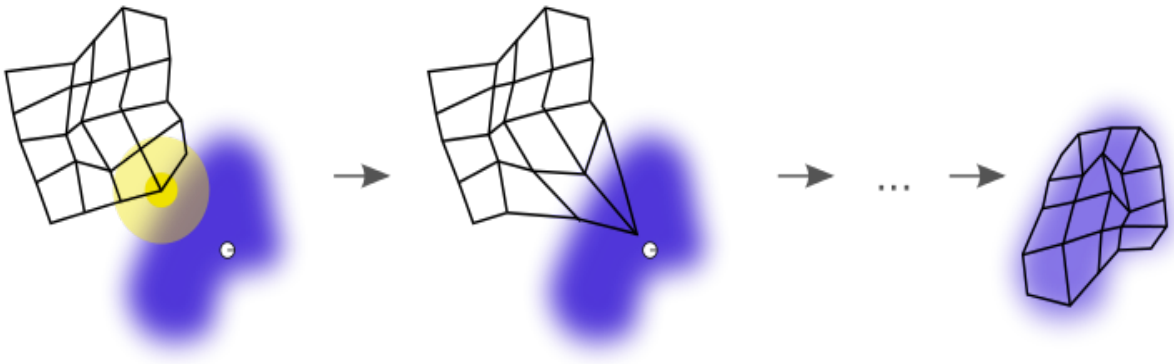

Figure 2: *An illustration of the training of a self-organizing map. The blue blob is the distribution of the training data, and the small white disc is the current training datum drawn from that distribution. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node (highlighted in yellow) which is nearest to the training datum is selected. It is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).*

the variation of the influence of the current data point on the weight vectors of the nodes, as the nodes becomes farther and farther from BMU. That is, it captures the fact that only weight vectors of clusters close to the BMU must be brought closer to the data point; all other clusters are already far away and capturing other aspects of data, so there is no use bringing their weight vectors close to the current data point. **This factor is the reason, why nearby clusters get assigned to nearby nodes, in this model**.

$\alpha(t)$ is a monotonically decreasing factor, which slows the learning down, as it progresses. It ensures that we don't overshoot and oscillate, and actually converge. the fact that the weight vectors of clusters far away from BMU need not be moved much since the point $D(t)$ belongs to the cluster of BMU.

Another point worth noting is that the dependency of θ on t, allows us to exhibit different behaviors at various times during training. For example, if θ can allow the range of influence to be large during the initial iterations, and slowly reduce the range of influence. This will allow for each data point to pull all weight vectors towards it initially, but later on in the training, each data point, will only be able to pull the weight vectors that are nearby.

Algorithm 1: The Learning Algorithm

W_v represents the weight vector of node v
The W's at the start of an iteration, and updated ones have superscript *old* and *new*, respectively.
Pick number of iterations, λ
for t from 1 to λ **do**
  Randomly pick a data vector $D(t)$
  Initialize : $BMU = 0$, $d = \infty$
  for each node N in the map **do**
    $x \leftarrow$ the distance between $D(t)$ and W_N^{old}
    if $x \leq d$ **then**
      $d \leftarrow x$ and $BMU \leftarrow N$
    end if
  end for
  for Node v in the neighborhood BMU **do**
    $W_v^{new} = W_v^{old} + \theta_t(BMU, v) \cdot \alpha(t) \cdot (D(t) - W_v^{old})$
  end for
end for

2.3 Applications

The main application of the above algorithm is in dimensionality reduction and unsupervised clustering. It is found to perform better than conventional feature extraction methods like PCA etc. We can even do step wise clustering, that is make a thousands nodes over a data space, and then cluster those thousand nodes.

References

[1] Hubert Ramsauer et al. "Hopfield networks is all you need". In: *arXiv preprint arXiv:2008.02217* (2020).

[2] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.