Imperial College London

Using Hopfield Networks to restore corrupted greyscale images

An introduction to Hopfield Networks

Hopfield Networks are a form of Ising model that have many uses in various sectors: computational neuroscience, optimisation, and more. Similar to an Ising model, they are comprised of an array of nodes that are given a *magnetic spin*, either $+1$ or -1, and connections, *'weights'*, linking them. We can use this archetecture to *'train'* the network on given sets of data, effectively *'storing the memory'* of the data on the network. Then, we can recover the stored data by inputting a new set of data and *'updating'* the network using the weights we obtained from training it on our sets of data.

In this project, we will look specifically at its application in image processing: how it can be used to restore and discern between corrupted pixellated images. By training the network on a set of images, we can then input corrupted versions of the images into the network, and updating the network will yield an image that should hopefully resemble the orignal image to a good degree of accuracy.

Basic definitions

We start by defining our n node spins V_i, where $i \in \{0, 1, ..., n\}$. These nodes all have connections to other nodes, and this is denoted by T_{ij}, which represents the connection between nodes i and j. We note that, $\forall i, j, T_{ij} = T_{ji}$. Further, $T_{ij} = T_{ji} = 0$ whenever nodes i and j are not connected, or when $i = j$.

Like an Ising model, we *'update'* the network given a set of spins by considering a threshold for each node, denoted by U_i. Then, we say:

$$V_i = \begin{cases} +1, & \sum_j T_{ij} V_j > U_i \\ -1, & \sum_j T_{ij} V_j \leq U_i \end{cases} \tag{1}$$

This threshold vector could be defined specifically to influence how specific nodes tend to behave, but for the purposes of this project we will define the threshold vector to be the zero vector, that is, $U_i = 0, \forall i$.

This update can either be done *asynchronously* or *synchronously*, for the former, each V_i is updated individually and the new V_i is used for the updates of other nodes, and for the latter, all updates use the initial set of node spins before any updates took place. If updating synchronously, some kind of internal clock or way to save previous states of the network must exist, and so most biological systems, such as the brain, will update asynchronously.

The Hebbian learning rule

The core of the Hopfield network lies in its use in *'storing'* states of the network to be recovered later. This is done using the *Hebbian learning rule*. [1]

If we have a set of m states V^s, $s \in \{1, ..., m\}$, we can use the Hebbian learning rule:

$$T_{ij} = \sum_s V_i^s V_j^s \tag{2}$$

Since this equation yields nonzero values for the case where $i = j$, we must then adjust for this by resetting T_{ij} to 0 whenever $i = j$. This will update the weights of the network and effectively *'imprint'* the states into the network.

Now we have our basic tools for applying a Hopfield network to our problem, we can start looking at basic uses of the network.

Initialising and training the network

Suppose we have a set of k n-by-m uncorrupted images to train the model on, where each pixel can either be black or white. We can, without loss of generality, assign the colour black to the spin $+1$ and white to -1. Now, each image can be represented as a set of nm spins, using each pixel as a node in the network, so we can use a Hopfield network to store them.

To set up our problem, we can store each image as a vector of its spins, so for the sth image:

$$\boldsymbol{V^s} = \begin{bmatrix} V_1^s \\ V_2^s \\ \vdots \\ V_{nm}^s \end{bmatrix}$$

Similarly, we also store the weights in an nm-by-nm matrix:

$$\boldsymbol{T} = \begin{bmatrix} T_{1,1} & \dots & T_{1,nm} \\ \vdots & \ddots & \\ T_{nm,1} & & T_{nm,nm} \end{bmatrix}$$

Then, we can apply the Hebbian learning rule to train the network on our images, using (2):

$$\boldsymbol{T} = \frac{1}{k} \sum_s (\boldsymbol{V^s} \otimes \boldsymbol{V^s} - diag(V_1^s V_1^s, \dots, V_{nm}^s V_{nm}^s)) \tag{3}$$

where $\otimes$ indicates the vector outer product.

This then gives us:

$$\boldsymbol{T} = \frac{1}{k} \sum_s \begin{bmatrix} 0 & V_1^s V_2^s & \dots & V_1^s V_{nm}^s \\ V_2^s V_1^s & \ddots & & \vdots \\ \vdots & & & \vdots \\ V_{nm}^s V_1^s & \dots & \dots & 0 \end{bmatrix}$$

as desired.

Restoring images

Et rutrum ex euismod vel. Pellentesque ultricies, velit in fermentum vestibulum, lectus nisi pretium nibh, sit amet aliquam lectus augue vel velit. Suspendisse rhoncus massa porttitor augue feugiat molestie. Sed molestie ut orci nec malesuada. Sed ultricies feugiat est fringilla posuere.

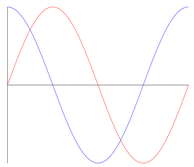

Figure 1. Another figure caption.

t

Etiam sit amet tempus lorem, aliquet condimentum velit. Donec et nibh consequat, sagittis ex eget, dictum orci. Etiam quis semper ante. Ut eu mauris purus. Proin nec consectetur ligula. Mauris pretium molestie ullamcorper. Integer nisi neque, aliquet et odio non, sagittis porta justo.

A highlighted block containing some math

A different kind of highlighted block.

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$$

Interdum et malesuada fames $\{1, 4, 9, \dots\}$ ac ante ipsum primis in faucibus. Cras eleifend dolor eu nulla suscipit suscipit. Sed lobortis non felis id vulputate.

A heading inside a block

Praesent consectetur mi $x^2 + y^2$ metus, nec vestibulum justo viverra nec. Proin eget nulla pretium, egestas magna aliquam, mollis neque. Vivamus dictum $\mathbf{u}^\intercal \mathbf{v}$ sagittis odio, vel porta erat congue sed. Maecenas ut dolor quis arcu auctor porttitor.

Another heading inside a block

Sed augue erat, scelerisque a purus ultricies, placerat porttitor neque. Donec $P(y \mid x)$ fermentum consectetur $\nabla_x P(y \mid x)$ sapien sagittis egestas. Duis eget leo euismod nunc viverra imperdiet nec id justo.

Nullam vel erat at velit convallis laoreet

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Phasellus libero enim, gravida sed erat sit amet, scelerisque congue diam. Fusce dapibus dui ut augue pulvinar iaculis.

First column	Second column	Third column	Fourth
Foo	13.37	384,394	α
Bar	2.17	1,392	β
Baz	3.14	83,742	δ
Qux	7.59	974	γ

Table 1. A table caption.

Donec quis posuere ligula. Nunc feugiat elit a mi malesuada consequat. Sed imperdiet augue ac nibh aliquet tristique. Aenean eu tortor vulputate, eleifend lorem in, dictum urna. Proin auctor ante in augue tincidunt tempor. Proin pellentesque vulputate odio, ac gravida nulla posuere efficitur. Aenean at velit vel dolor blandit molestie. Mauris laoreet commodo quam, non luctus nibh ullamcorper in. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Nulla varius finibus volutpat. Mauris molestie lorem tincidunt, iaculis libero at, gravida ante. Phasellus at felis eu neque suscipit suscipit. Integer ullamcorper, dui nec pretium ornare, urna dolor consequat libero, in feugiat elit lorem euismod lacus. Pellentesque sit amet dolor mollis, auctor urna non, tempus sem.

References

[1] J. J. Hopfield.
Neural networks and physical systems with emergent collective computational abilities.
Proceedings of the National Academy of Sciences of the United States of America, 79:2554–2558, April 1982.

Daniel Haywood