

FIRST UNIT OF Q520: HOPFIELD NETS

Larry Moss

Indiana University

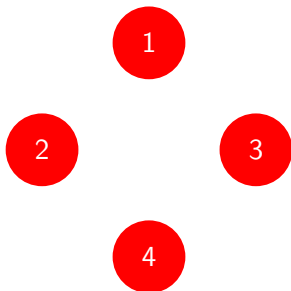
Lecture notes for Q520

HOPFIELD NETS: TOY EXAMPLE

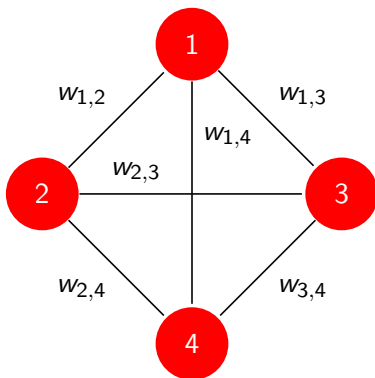
Suppose we want to store three patterns of length 4, say

$$x^1 = \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \quad x^2 = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \quad x^3 = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

We would use a network with 4 nodes.



HOPFIELD NETS: WEIGHTS ON THE EDGES



The weights will be *symmetric*: $w_{1,2} = w_{2,1}$, $w_{3,4} = w_{4,3}$, etc.

And we also will have $w_{1,1} = w_{2,2} = w_{3,3} = w_{4,4} = 0$.

The weights will be given in terms of a formula, from the vectors x^1 , x^2 , and x^3 which we wish to store.

HOPFIELD NETS: WEIGHT FORMULA

Recall our vectors

$$x^1 = \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \quad x^2 = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \quad x^3 = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

For example we have $x_1^1 = -1$, $x_2^1 = 1$, $x_3^1 = -1$, $x_4^1 = 1$.

For $i \neq j$, our formula is

$$w_{i,j} = \sum_k x_i^k x_j^k$$

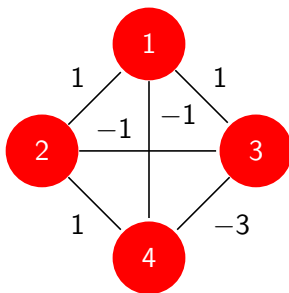
We would have

$$w_{1,2} = (-1)(1) + (1)(1) + (-1)(-1) = 1$$

$$w_{1,3} = (-1)(-1) + (1)(-1) + (-1)(-1) = 1$$

$$w_{1,4} = -1, w_{2,3} = -1, w_{2,4} = 1, w_{3,4} = -3.$$

HOPFIELD NETS: OUR EXAMPLE



Again, the weights are symmetric, and $w_{i,i} = 0$ for all i .

Again, this is the net that is **intended to store** x^1 , x^2 , and x^3 .

Note that for numbers 1 and -1 , the product is 1 if the numbers are the same, and -1 if they differ.

So the weight $w_{i,j}$ is a measure of how correlated the i th and j th entries of the stored vectors is.

RUNNING A NET

We start with a vector $\vec{o} = (o_1, \dots, o_n)$ of inputs,
The length of \vec{o} is the same length n as our stored patterns.
Then for each i from 1 to n we calculate

$$\sum_j w_{i,j} o_j$$

- ★ If this sum is positive, we (re)set o_i to 1.
- ★ If this sum is negative, we (re)set o_i to -1 .
- ★ If this sum is zero, we keep our old value.

We do these *small steps* for each i from 1 to n .
This makes one *big step*. We do as many big steps as it takes to *stabilize*.

MORE ON RUNNING HOPFIELD NETS

There are actually several different ways to run a net:

- 1 All small steps in parallel (**synchronously**).
- 2 The small steps done **sequentially** in order: node 1, node 2, node 3, ..., node n .
- 3 The small steps done sequentially in some random fashion.

The first alternative is most in the spirit of the applications.

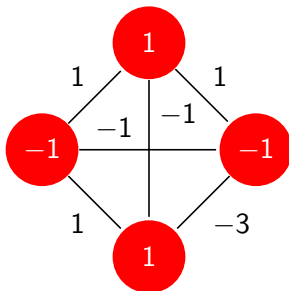
However, it has a weird behavior as we'll see.

The second is the easiest to analyze, and it has a nice behavior.

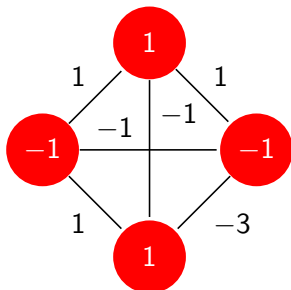
HOPFIELD NETS: RUNNING IT

Let's run our example net on $\begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$.

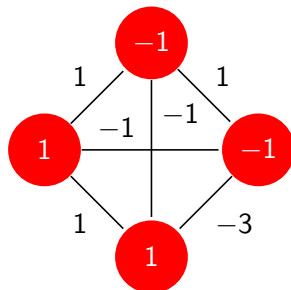
We stick these values on the nodes of our net:



HOPFIELD NETS: RUNNING IT, CONTINUED



before



after

HOPFIELD NETS: RUNNING IT, CONTINUED

Note that we did it in parallel; doing it sequentially would give different results on some nets.

In any case, a further parallel execution gives the same result.

So we say that running the net on

$$\begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \text{ gives } \begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$$

You try running our net on $\begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$.

THE ALGORITHM

Given a Hopfield net and an input vector \vec{o} :

1. Initialize the net to \vec{o} by putting the entries of \vec{o} on the nodes of the net.
2. For $i = 1, \dots, n$, we make the following small steps: Calculate the following number

$$\sum_j w_{i,j} o_j$$

- ★ If this number matches o_i in sign, don't do anything.
 - ★ If this number disagrees with o_i in sign, flip o_i to $-o_i$.
 - ★ If this number is zero, again we don't do anything.
3. If none of o_1, \dots, o_n were changed in step 2, then we're done. Otherwise, repeat step 2.

THE QUESTIONS

We are not really interested in running Hopfield nets.

We take it as a given that they do *something* interesting, and it's our goal to understand *what* that is.

To really gain a deep understanding would be very hard, and this is typical of many discrete dynamic mathematical models.

For us in this course, Hopfield nets are essentially an “excuse” for the resulting theory.

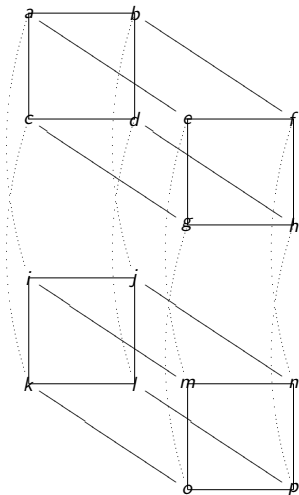
THE STATE SPACE

It will be convenient to have names of the 16 elements of our state space:

<i>a</i>	$(1, 1, 1, 1)$	<i>i</i>	$(-1, 1, 1, 1)$
<i>b</i>	$(1, 1, 1, -1)$	<i>j</i>	$(-1, 1, 1, -1)$
<i>c</i>	$(1, 1, -1, 1)$	<i>k</i>	$(-1, 1, -1, 1)$
<i>d</i>	$(1, 1, -1, -1)$	<i>l</i>	$(-1, 1, -1, -1)$
<i>e</i>	$(1, -1, 1, 1)$	<i>m</i>	$(-1, -1, 1, 1)$
<i>f</i>	$(1, -1, 1, -1)$	<i>n</i>	$(-1, -1, 1, -1)$
<i>g</i>	$(1, -1, -1, 1)$	<i>o</i>	$(-1, -1, -1, 1)$
<i>h</i>	$(1, -1, -1, -1)$	<i>p</i>	$(-1, -1, -1, -1)$

And then we want to draw them as a picture:

THE STATE SPACE PICTURED



NEIGHBORS

A *graph* in mathematics is a set of points together with edges that connect some of them.

“Points” are also called “nodes” or “vertices”.

There are actually two flavors of graphs:

- ★ undirected: as we have seen today.
- ★ directed: these have one-way edges, indicate by arrows.

As in every graph, *neighbors* are points connected by an edge.

For example, in the state space graph, the neighbors of a are b , c , e , and i .

Every point in that graph has exactly four neighbors. Why is this true??

MORE ON NEIGHHBORS

Given a vector

$$\vec{o} = (o_1, \dots, o_{i-1}, o_i, o_{i+1}, \dots, o_n),$$

it has n neighbors. These are

$$(o_1, \dots, o_{i-1}, -o_i, o_{i+1}, \dots, o_n).$$

where i ranges from 1 to n .

Remember that this whole topic is an excuse for the mathematics.
So we really want to brush up on math notation of various sorts.

WHAT DO THE SMALL STEPS OF THE ALGORITHM DO?

Each small step either keeps the current \vec{o} or moves to some neighbor \vec{o}' .

In more detail, suppose that \vec{o} is the value before the i th small step of some big step. Then the i th small step either keeps \vec{o} as it is, or it replaces \vec{o} with

$$(o_1, \dots, o_{i-1}, -o_i, o_{i+1}, \dots, o_n).$$

We then think of the algorithm in a new way: instead of keeping or changing the i th bit, we think of keeping \vec{o} or moving to one of its neighbors.

ENERGY

The key to understanding the properties of Hopfield nets is to study the following function:

$$\Phi(\vec{o}) = -\frac{1}{2} \sum_{i,j} w_{i,j} o_i o_j.$$

Note that the sum here is over all the pairs of numbers i and j . Since $w_{i,j} = w_{j,i}$, we have terms like $w_{1,2} o_1 o_2$ and $w_{2,1} o_2 o_1$. Clearly these are equal, and clearly there's no need to repeat things.

So

$$\Phi(\vec{o}) = -\sum_{i < j} w_{i,j} o_i o_j.$$

ENERGY IN TERMS OF EDGES

As we just saw,

$$\Phi(\vec{o}) = - \sum_{i < j} w_{i,j} o_i o_j.$$

We can think of this in terms of the *edges* of the network instead of the points.

The sum tells us to look at each edge e , multiply the weight by the first and second endpoints, then add all these up.

$$\Phi(\vec{o}) = - \sum_e w_e o_{\text{first}(e)} o_{\text{second}(e)}.$$

Be sure you see why the 1/2 is gone.

TRY FOR YOURSELF

Draw a Hopfield net with 4 nodes, together with the edge weights and with node inputs, and arrange that the energy is 3.

Then find a set of target patterns that would give the edge weights in your net.

It's best if you patterns didn't have repeats.

$$w_{12} = 1, w_{13} = 1, w_{14} = -1, w_{23} = -1, w_{24} = 1, \\ w_{34} = -3,$$

$$\Phi(1, 1, 1, 1) = 2$$

$$\Phi(1, 1, 1, -1) = -4$$

$$\Phi(1, 1, -1, 1) = -4$$

$$\Phi(1, 1, -1, -1) = 2$$

$$\Phi(1, -1, 1, 1) = 4$$

$$\Phi(1, -1, 1, -1) = -6$$

$$\Phi(1, -1, -1, 1) = 2$$

$$\Phi(1, -1, -1, -1) = 4$$

$$\Phi(-1, 1, 1, 1) = 4$$

$$\Phi(-1, 1, 1, -1) = 2$$

$$\Phi(-1, 1, -1, 1) = -6$$

$$\Phi(-1, 1, -1, -1) = 4$$

$$\Phi(-1, -1, 1, 1) = 2$$

$$\Phi(-1, -1, 1, -1) = -4$$

$$\Phi(-1, -1, -1, 1) = -4$$

ENERGY IN TERMS OF EDGES, CONTINUED

The whole idea of a positive edge weight is that it “wants” the endpoints to be equal. So if they are, we get a positive contribution to the weight; if they are unequal, a negative contribution. Similarly, a negative edge weight “wants” unequal endpoint values. If it gets this, the overall product

$$w_e o_{\text{first}(e)} o_{\text{second}(e)}$$

will be positive.

So the closer \vec{o} is to “what the edges want”, the *higher* the sum $\sum_{i < j} w_{ij} o_i o_j$, hence the *lower* the energy.

THE BASIC LEMMA ON ENERGY

Let \vec{o} and \vec{o}' be neighbors, agreeing at all entries except the i th.
That is

$$o'_i = -o_i$$

Then

$$\Phi(\vec{o}') - \Phi(\vec{o}) = 2o_i \sum_j o_j w_{i,j}.$$

Here is the reasoning:

$\Phi(\vec{o}')$ and $\Phi(\vec{o})$ are both big sums over edges e . Most of the terms in the two sums are the same, so most will cancel when we subtract.

Which terms remain? The ones where i is an endpoint of e .

REASONING, CONTINUED

$$\begin{aligned}
& \Phi(\vec{o}') - \Phi(\vec{o}) \\
= & -\sum_e o'_{\text{first}(e)} o'_{\text{sec}(e)} w_e + \sum_e o_{\text{first}(e)} o_{\text{sec}(e)} w_e \\
= & \sum_e (-o'_{\text{first}(e)} o'_{\text{sec}(e)} + o_{\text{first}(e)} o_{\text{sec}(e)}) w_e \\
= & \sum_{j \neq i} (-o'_i o'_j + o_i o_j) w_{i,j} \\
= & \sum_{j \neq i} (-o'_i + o_i) o_j w_{i,j} \\
= & \sum_{j \neq i} (o_i + o_i) o_j w_{i,j} \\
= & \sum_{j \neq i} 2o_i o_j w_{i,j} \\
= & 2o_i \sum_{j \neq i} o_j w_{i,j}
\end{aligned}$$

Note that we used the important facts that $o'_i = -o_i$ and also that for $j \neq i$, $o'_j = o_j$.

THE POINT

Running Hopfield's algorithm asks us to consider

$$\sum_{j \neq i} o_j w_{i,j}$$

We then change o_i if it differs from the sign of the number above. We see that this number has *something* to do with energy! Incidentally, there's no *mathematical* significance to the word "energy".

But there's a connection to ideas from physics.

THE CRUX OF THE MATTER

Let \vec{o} be the vector at the start of some big step of the algorithm. If some small step of the algorithm changes \vec{o} to \vec{o}' , then we must have $\Phi(\vec{o}') < \Phi(\vec{o})$.

And if some neighbor \vec{o}' of \vec{o} has smaller energy than \vec{o} , then some small step of the algorithm will change \vec{o} .

This tells us what the algorithm does, in terms of energy.

Beginning with \vec{o} , we look at the neighbors one at a time, trying to find one of strictly smaller energy.

If we never find one, the overall big step stabilizes.

The algorithm will always stabilize at some **local minima of energy**.

FIRST HALF

We prove that if a small step of the algorithm changes \vec{o} to its neighbor \vec{o}' , then we must have $\Phi(\vec{o}') < \Phi(\vec{o})$.

Let i be the place where \vec{o} and \vec{o}' differ, so $o'_i = -o_i$.

The way the algorithm works, $\sum_j w_{i,j} o_j$ is not zero. Further, the sum and o_i have opposite signs. So $o_i \sum_j o_j w_{i,j}$ is < 0 .

By the basic lemma,

$$\Phi(\vec{o}') - \Phi(\vec{o}) = 2o_i \sum_j o_j w_{i,j}.$$

And we now know that this is negative.

So $\Phi(\vec{o}') < \Phi(\vec{o})$.

SECOND HALF

Start at the beginning of one of the big steps.

We show that if some neighbor \vec{o}' of \vec{o} has smaller energy than \vec{o} , then some small step of the algorithm will change \vec{o} .

Let's consider the first neighbor \vec{o}' with strictly smaller energy than \vec{o} .

Suppose that this is the i th neighbor.

For $j < i$, \vec{o} does not change at the j th small step.

We'll show that at the i th small step, \vec{o} will change to \vec{o}' .

Since $\Phi(\vec{o}') < \Phi(\vec{o})$, we have $\Phi(\vec{o}') - \Phi(\vec{o}) < 0$.

As we know, $2o_i \sum_j o_j w_{i,j} < 0$.

So the sum $\sum_j o_j w_{i,j}$ is not zero, and it has opposite sign from o_i .

Thus at the i th small step, we change o_i to $-o_i$.

That is, we move from \vec{o} to \vec{o}' .

SUMMARY

The set of possible states \vec{o} is the set of n -tuples of 1's and -1 's. This set has a *graph structure*: we have a natural definition of neighbors on it.

The sequential algorithm takes an input vector \vec{o} and wanders around, always going to a neighbor, and always moving to a point of lower energy.

The whole set of possible states is finite, so there are always absolute minima of energy – hence local minima as well.

The algorithm always stabilizes at a local minimum of energy.

NB: The *parallel* algorithm does not have this property.

IMPLICATIONS

Recall that the whole purpose of the Hopfield net is to store some target vectors x^1, x^2, \dots, x^r .

As we now know, what the net in fact stores are the local minima of Φ .

So the best possible situation is when our targets are *exactly* the local minima of energy. This is explored a bit in the exercises.

In practice, if the targets are “spread out” and if there are “not too many” of them, then each will be a local minimum of energy. But there also usually are lots of **unintended** local minima.

This is a drawback for Hopfield nets.

IMPLICATIONS

Another drawback is that the stored patterns are not stored in a robust way.

Finally, recall that every node in the graph is connected to every other node. This is unattractive as a biological model, and of course it makes for lots of edges.

It's possible to modify the overall architecture of Hopfield nets. Can you think of some ways?

And what do you think the effect of them would be for our analysis?

OUR LEARNING ON HOPFIELD NETS

Here's what I want people to be ok with at this point:

- ★ Notation for sums
 - ★★ changing variables in sums
 - ★★ changing order of sums
- ★ The idea that when you see a mathematical definition you try it for yourself on small examples.
- ★ Informal understanding of algorithms.
- ★ The reasoning involved in the main work on the algorithm.
- ★ the words “if” and “if and only if” in mathematics and logic