Neural Networks

Hopfield Nets and Auto Associators

Fall 2020
Story so far

- Neural networks for computation
- All feedforward structures
- But what about..
Consider this loopy network

\[
\Theta(z) = \begin{cases} 
+1 & \text{if } z > 0 \\
-1 & \text{if } z \leq 0 
\end{cases} \quad y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right)
\]

- Each neuron is a perceptron with +1/-1 output
- Every neuron \textit{receives} input from every other neuron
- Every neuron \textit{outputs} signals to every other neuron

The output of a neuron affects the input to the neuron
Consider this loopy network

\[
\Theta(z) = \begin{cases} 
+1 & \text{if } z > 0 \\
-1 & \text{if } z \leq 0 
\end{cases}
\]

\[
y_i = \Theta\left(\sum_{j \neq i} w_{ji}y_j + b_i\right)
\]

- Each neuron is a perceptron with +1/-1 output
- Every neuron \textit{receives} input from every other neuron
- Every neuron \textit{outputs} signals to every other neuron
Hopfield Net

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

- Each neuron is a perceptron with +1/-1 output
- Every neuron receives input from every other neuron
- Every neuron outputs signals to every other neuron

A symmetric network:

\[ w_{ij} = w_{ji} \]
Loopy network

• At each time each neuron receives a “field” \( \sum_{j \neq i} w_{ji} y_j + b_i \)

• If the sign of the field matches its own sign, it does not respond

• If the sign of the field opposes its own sign, it “flips” to match the sign of the field

\[
y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right)
\]

\[
\Theta(z) = \begin{cases} 
+1 & \text{if } z > 0 \\
-1 & \text{if } z \leq 0
\end{cases}
\]
Loopy network

At each time each neuron receives a “field” $\sum_{j \neq i} w_{ji}y_j + b_i$

If the sign of the field matches its own sign, it does not respond

If the sign of the field opposes its own sign, it “flips” to match the sign of the field

$$y_i = \Theta \left( \sum_{j \neq i} w_{ji}y_j + b_i \right)$$

$$\Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases}$$
A neuron “flips” if weighted sum of other neurons’ outputs is of the opposite sign to its own current (output) value.

But this may cause other neurons to flip!

If the sign of the field opposes its own sign, it “flips” to match the sign of the field.

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji}y_j + b_i \right) \]

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]
Example

- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1
• Red edges are +1, blue edges are -1
• Yellow nodes are -1, black nodes are +1
• Red edges are +1, blue edges are -1
• Yellow nodes are -1, black nodes are +1
Example

- Red edges are +1, blue edges are -1
- Yellow nodes are -1, black nodes are +1
• If the sign of the field at any neuron opposes its own sign, it “flips” to match the field
  – Which will change the field at other nodes
    • Which may then flip
      – Which may cause other neurons including the first one to flip...
        » And so on...
20 evolutions of a loopy net

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

A neuron “flips” if weighted sum of other neuron’s outputs is of the opposite sign. But this may cause other neurons to flip!

• All neurons which do not “align” with the local field “flip”
120 evolutions of a loopy net

- All neurons which do not “align” with the local field “flip”
• If the sign of the field at any neuron opposes its own sign, it “flips” to match the field
  – Which will change the field at other nodes
    • Which may then flip
      – Which may cause other neurons including the first one to flip...

• Will this behavior continue for ever??
Loopy network

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

\[ \Theta(z) = \begin{cases} 
+1 & \text{if } z > 0 \\
-1 & \text{if } z \leq 0 
\end{cases} \]

- Let \( y_i^- \) be the output of the \( i \)-th neuron just *before* it responds to the current field.
- Let \( y_i^+ \) be the output of the \( i \)-th neuron just *after* it responds to the current field.
- If \( y_i^- = \text{sign} \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \), then \( y_i^+ = y_i^- \)
  - If the sign of the field matches its own sign, it does not flip.

\[ y_i^+ \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) - y_i^- \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) = 0 \]
Loopy network

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]

- If \( y_i^- \neq \text{sign} \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \), then \( y_i^+ = -y_i^- \)

\[ y_i^+ \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) - y_i^- \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) = 2y_i^+ \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

- This term is always positive!

- Every flip of a neuron is guaranteed to locally increase

\[ y_i \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]
Globally

• Consider the following sum across all nodes

\[ D(y_1, y_2, \ldots, y_N) = \sum_i y_i \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

\[ = \sum_{i,j \neq i} w_{ij} y_i y_j + \sum_i b_i y_i \]

– Assume \( w_{ii} = 0 \)

• For any unit \( k \) that “flips” because of the local field

\[ \Delta D(y_k) = D(y_1, \ldots, y_k^+, \ldots, y_N) - D(y_1, \ldots, y_k, \ldots, y_N) \]
Upon flipping a single unit

\[ \Delta D(y_k) = D(y_1, \ldots, y_k^+, \ldots, y_N) - D(y_1, \ldots, y_k^-, \ldots, y_N) \]

• Expanding

\[ \Delta D(y_k) = (y_k^+ - y_k^-) \left( \sum_{j \neq k} w_{jk}y_j + b_k \right) \]

– All other terms that do not include \( y_k \) cancel out

• This is always positive!

• *Every flip of a unit results in an increase in* \( D \)
• Flipping a unit will result in an increase (non-decrease) of

\[ D = \sum_{i,j \neq i} w_{ij} y_i y_j + \sum_i b_i y_i \]

• \( D \) is bounded

\[ D_{max} = \sum_{i,j \neq i} |w_{ij}| + \sum_i |b_i| \]

• The minimum increment of \( D \) in a flip is

\[ \Delta D_{min} = \min_{i, \{y_i, i=1..N\}} 2 \left| \sum_{j \neq i} w_{ji} y_j + b_i \right| \]

• Any sequence of flips must converge in a finite number of steps
The Energy of a Hopfield Net

• Define the *Energy* of the network as

\[ E = - \sum_{i,j \neq i} w_{ij}y_i y_j - \sum_i b_i y_i \]

– Just the negative of *D*

• The evolution of a Hopfield network constantly decreases its energy
A Hopfield network is a loopy binary network with symmetric connections.

Every neuron in the network attempts to “align” itself with the sign of the weighted combination of outputs of other neurons.

- The local “field”

Given an initial configuration, neurons in the net will begin to “flip” to align themselves in this manner.

- Causing the field at other neurons to change, potentially making them flip

Each evolution of the network is guaranteed to decrease the “energy” of the network.

- The energy is lower bounded and the decrements are upper bounded, so the network is guaranteed to converge to a stable state in a finite number of steps
The Energy of a Hopfield Net

• Define the *Energy* of the network as

\[ E = - \sum_{i,j \neq i} w_{ij} y_i y_j - \sum_i b_i y_i \]

– Just the negative of *D*

• The evolution of a Hopfield network constantly decreases its energy

• Where did this “energy” concept suddenly sprout from?
Analogy: Spin Glass

• Magnetic dipoles in a disordered magnetic material
• Each dipole tries to *align* itself to the local field
  – In doing so it may flip
• This will change fields at *other* dipoles
  – Which may flip
• Which changes the field at the current dipole...
Analogy: Spin Glasses

- $p_i$ is vector position of $i$-th dipole
- The field at any dipole is the sum of the field contributions of all other dipoles
- The contribution of a dipole to the field at any point depends on interaction $J$
  - Derived from the “Ising” model for magnetic materials (Ising and Lenz, 1924)

Total field at current dipole:

$$f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i$$
• A Dipole flips if it is misaligned with the field in its location.

Total field at current dipole:

\[ f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i \]

Response of current dipole:

\[ x_i = \begin{cases} 
  x_i & \text{if } \text{sign}(x_i f(p_i)) = 1 \\
  -x_i & \text{otherwise}
\end{cases} \]
Analogy: Spin Glasses

Total field at current dipole:

\[ f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i \]

Response of current dipole

\[ x_i = \begin{cases} 
  x_i & \text{if } \text{sign}(x_i f(p_i)) = 1 \\
  -x_i & \text{otherwise} 
\end{cases} \]

- Dipoles will keep flipping
  - A flipped dipole changes the field at other dipoles
    - Some of which will flip
  - Which will change the field at the current dipole
    - Which may flip
  - Etc..
• When will it stop???

Analogy: Spin Glasses

Total field at current dipole:

\[ f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i \]

Response of current dipole

\[
 x_i = \begin{cases} 
 x_i & \text{if } \text{sign}(x_i f(p_i)) = 1 \\
 -x_i & \text{otherwise}
\end{cases}
\]
The “Hamiltonian” (total energy) of the system

\[ E = -\frac{1}{2} \sum_i x_i f(p_i) = -\sum_i \sum_{j>i} J_{ji} x_i x_j - \sum_i b_i x_i \]

The system *evolves* to minimize the energy

- Dipoles stop flipping if any flips result in increase of energy

Total field at current dipole:

\[ f(p_i) = \sum_{j \neq i} J_{ji} x_j + b_i \]

Response of current dipole

\[ x_i = \begin{cases} x_i & \text{if } \text{sign}(x_i f(p_i)) = 1 \\ -x_i & \text{otherwise} \end{cases} \]

Analogy: Spin Glasses
Spin Glasses

- The system stops at one of its stable configurations
  - Where energy is a local minimum
- Any small jitter from this stable configuration returns it to the stable configuration
  - I.e. the system remembers its stable state and returns to it
Hopfield Network

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j + b_i \right) \]

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]

\[ E = - \sum_{i,j \neq i} w_{ij} y_i y_j - \sum_i b_i y_i \]

- This is analogous to the potential energy of a spin glass
  - The system will evolve until the energy hits a local minimum
Hopfield Network

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji}y_j + b_i \right) \]

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]

Typically will not utilize bias: The bias is similar to having a single extra neuron that is pegged to 1.0

Removing the bias term does not affect the rest of the discussion in any manner

But not RIP, we will bring it back later in the discussion
Hopfield Network

\[ y_i = \Theta \left( \sum_{j \neq i} w_{ji} y_j \right) \]

\[ \Theta(z) = \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z \leq 0 \end{cases} \]

\[ E = - \sum_{i,j<i} w_{ij} y_i y_j \]

- This is analogous to the potential energy of a spin glass
  - The system will evolve until the energy hits a local minimum
    - Above equation is a factor of 0.5 off from earlier definition for conformity with thermodynamic system
Evolution

- The network will evolve until it arrives at a local minimum in the energy contour

\[ E = - \sum_{i,j<i} w_{ij} y_i y_j \]
Each of the minima is a “stored” pattern
- If the network is initialized close to a stored pattern, it will inevitably evolve to the pattern

This is a content addressable memory
- Recall memory content from partial or corrupt values

Also called associative memory
Evolution

The network will evolve until it arrives at a local minimum in the energy contour.

\[ E = - \sum_{i,j < i} w_{ij} y_i y_j \]
Evolution

The network will evolve until it arrives at a local minimum in the energy contour.

We proved that every change in the network will result in *decrease* in energy.

- So path to energy minimum is monotonic

\[ E = - \sum_{i,j<i} w_{ij} y_i y_j \]
• For threshold activations the energy contour is only defined on a lattice
  – Corners of a unit cube on \([-1,1]^N\)
• For threshold activations the energy contour is only defined on a lattice
  – Corners of a unit cube on \([-1,1]^N\)
• For tanh activations it will be a continuous function
Evolution

For threshold activations the energy contour is only defined on a lattice
- Corners of a unit cube

For tanh activations it will be a continuous function
- With output in [-1 1]

\[ E = -\frac{1}{2} y^T W y \]

In matrix form
Note the 1/2
“Energy” contour for a 2-neuron net

- Two stable states (tanh activation)
  - Symmetric, not at corners
  - Blue arc shows a typical trajectory for tanh activation
“Energy” contour for a 2-neuron net

Why symmetric?

Because \(-\frac{1}{2} y^T W y = -\frac{1}{2} (-y)^T W (-y)\)

If \(\hat{y}\) is a local minimum, so is \(-\hat{y}\)

- Blue arc shows a typical trajectory for sigmoid activation
• 8 possible states
• 2 stable states (hard thresholded network)
Examples: Content addressable memory

Hopfield network reconstructing degraded images from noisy (top) or partial (bottom) cues.

Hopfield net examples
**Computational algorithm**

1. Initialize network with initial pattern

   \[ y_i(0) = x_i, \quad 0 \leq i \leq N - 1 \]

2. Iterate until convergence

   \[ y_i(t + 1) = \Theta \left( \sum_{j \neq i} w_{ji} y_j \right), \quad 0 \leq i \leq N - 1 \]

- Very simple
- Updates can be done sequentially, or all at once
- Convergence

\[ E = - \sum_{i} \sum_{j>i} w_{ji} y_j y_i \]

does not change significantly any more
A Hopfield network is a loopy binary network with symmetric connections. Neurons try to align themselves to the local field caused by other neurons.

Given an initial configuration, the patterns of neurons in the net will evolve until the “energy” of the network achieves a local minimum. The evolution will be monotonic in total energy. The dynamics of a Hopfield network mimic those of a spin glass. The network is symmetric: if a pattern $Y$ is a local minimum, so is $-Y$.

The network acts as a content-addressable memory. If you initialize the network with a somewhat damaged version of a local-minimum pattern, it will evolve into that pattern. Effectively “recalling” the correct pattern, from a damaged/incomplete version.
Issues

• How do we make the network store a specific pattern or set of patterns?

• How many patterns can we store?

• How to “retrieve” patterns better..
Issues

• How do we make the network store a specific pattern or set of patterns?

• How many patterns can we store?

• How to “retrieve” patterns better.
How do we remember a specific pattern?

• How do we teach a network to “remember” this image

• For an image with $N$ pixels we need a network with $N$ neurons
• Every neuron connects to every other neuron
• Weights are symmetric (not mandatory)

$$\frac{N(N-1)}{2}$$ weights in all
A network that stores pattern $P$ also naturally stores $-P$.

- Symmetry $E(P) = E(-P)$ since $E$ is a function of $y_i y_j$

$$E = - \sum_i \sum_{j<i} w_{ji} y_j y_i$$
A network can store *multiple* patterns

- Every stable point is a stored pattern
- So we could design the net to store multiple patterns
  - Remember that every stored pattern $P$ is actually *two* stored patterns, $P$ and $-P$
Storing a pattern

\[ E = - \sum_i \sum_{j<i} w_{ji} y_j y_i \]

- Design \( \{w_{ij}\} \) such that the energy is a local minimum at the desired \( P = \{y_i\} \)
Storing specific patterns

- Storing 1 pattern: We want
  \[ \text{sign} \left( \sum_{j \neq i} w_{ji} y_j \right) = y_i \quad \forall \ i \]

- This is a stationary pattern
Storing specific patterns

HEBBIAN LEARNING:

\[ w_{ji} = y_j y_i \]

• Storing 1 pattern: We want

\[ \text{sign} \left( \sum_{j \neq i} w_{ji} y_j \right) = y_i \quad \forall \; i \]

• This is a stationary pattern
Storing specific patterns

HEBBIAN LEARNING:

\[ w_{ji} = y_j y_i \]

- \[ \text{sign} \left( \sum_{j \neq i} w_{ji} y_j \right) = \text{sign} \left( \sum_{j \neq i} y_j y_i y_j \right) \]

\[ = \text{sign} \left( \sum_{j \neq i} y_j^2 y_i \right) = \text{sign}(y_i) = y_i \]
Storing specific patterns

HEBBIAN LEARNING:

\[ w_{ji} = y_j y_i \]

The pattern is stationary

\[ \text{sign} \left( \sum_{j \neq i} w_{ji} y_j \right) = \text{sign} \left( \sum_{j \neq i} y_j y_i y_j \right) \]

\[ = \text{sign} \left( \sum_{j \neq i} y_j^2 y_i \right) = \text{sign}(y_i) = y_i \]
Storing specific patterns

HEBBIAN LEARNING:

\[ w_{ji} = y_j y_i \]

\[
E = - \sum_{i} \sum_{j<i} w_{ji} y_j y_i = - \sum_{i} \sum_{j<i} y_i^2 y_j^2
\]

\[
= - \sum_{i} \sum_{j<i} 1 = -0.5N(N - 1)
\]

- This is the lowest possible energy value for the network
Storing specific patterns

HEBBIAN LEARNING:

\[ w_{ji} = y_j y_i \]

The pattern is **STABLE**

\[
E = - \sum_i \sum_{j<i} w_{ji} y_j y_i = - \sum_i \sum_{j<i} y_i^2 y_j^2 \\
= - \sum_i \sum_{j<i} 1 = -0.5N(N - 1)
\]

- This is the lowest possible energy value for the network
Hebbian learning: Storing a 4-bit pattern

- Left: Pattern stored. Right: Energy map
- Stored pattern has lowest energy
- Gradation of energy ensures stored pattern (or its ghost) is recalled from everywhere
Storing multiple patterns

- To store *more* than one pattern

\[ w_{ji} = \sum_{y_p \in \{y_p\}} y_i^p y_j^p \]

- \( \{y_p\} \) is the set of patterns to store
- Super/subscript \( p \) represents the specific pattern
How many patterns can we store?

- **Hopfield**: For a network of $N$ neurons can store up to $\sim 0.15N$ patterns through Hebbian learning
  - Provided they are “far” enough

- Where did this number come from?
The limits of Hebbian Learning

• Consider the following: We must store $K$ $N$-bit patterns of the form

$$y_k = [y_k^1, y_k^2, ..., y_k^N], k = 1 ... K$$

• Hebbian learning (scaling by $\frac{1}{N}$ for normalization, this does not affect actual pattern storage):

$$w_{ij} = \frac{1}{N} \sum_{k} y_i^k y_j^k$$

• For any pattern $y_p$ to be stable:

$$y_i^p \sum_j w_{ij} y_j^p > 0 \ \forall i$$

$$y_i^p \frac{1}{N} \sum_j \sum_k y_i^k y_j^k y_j^p > 0 \ \forall i$$
The limits of Hebbian Learning

• For any pattern \( y_p \) to be stable:

\[
\sum \sum_{j} y_i^k y_j^k y_j^p > 0 \quad \forall i
\]

\[
y_i^p \frac{1}{N} \sum y_j^p + y_i^p \frac{1}{N} \sum_{k \neq p} y_i^k y_j^k y_j^p > 0 \quad \forall i
\]

• Note that the first term equals 1 (because \( y_j^p y_j^p = y_i^p y_i^p = 1 \))
  – i.e. for \( y_p \) to be stable the requirement is that the second *crosstalk term*:

\[
y_i^p \frac{1}{N} \sum_{j} \sum_{k \neq p} y_i^k y_j^k y_j^p > -1 \quad \forall i
\]

• The pattern will *fail* to be stored if the *crosstalk*

\[
y_i^p \frac{1}{N} \sum \sum_{j} y_i^k y_j^k y_j^p < -1 \quad \text{for any } i
\]
The limits of Hebbian Learning

• For any random set of K patterns to be stored, the probability of the following must be low

\[
C^p_i = \frac{1}{N} \sum_j \sum_{k \neq p} y^p_i y^k_i y^k_j y^p_j < -1
\]

• For large N and K the probability distribution of \( C^p_i \) approaches a Gaussian with 0 mean, and variance \( K/N \)
  – Considering that individual bits \( y^l_i \in \{-1, +1\} \) and have variance 1
• For a Gaussian, \( C \sim N(0, K/N) \)
  – \( P(C < -1 \mid \mu = 0, \sigma^2 = K/N) < 0.004 \) for \( K/N < 0.14 \)

• I.e. To have less than 0.4% probability that stored patterns will not be stable, \( K < 0.14N \)
How many patterns can we store?

A network of \( N \) neurons trained by Hebbian learning can store up to \(~0.14N\) patterns with low probability of error

- Computed assuming \( \text{prob}(bit = 1) = 0.5 \)
  - On average no. of matched bits in any pair = no. of mismatched bits
  - Patterns are “orthogonal” – maximally distant – from one another
- Expected behavior for non-orthogonal patterns?

- To get some insight into what is stored, lets see some examples
Hebbian learning: One 4-bit pattern

- Left: Pattern stored. Right: Energy map
- Note: Pattern is an energy well, but there are other local minima
  - Where?
  - Also note “shadow” pattern
Storing multiple patterns: Orthogonality

• The maximum Hamming distance between two \( N \)-bit patterns is \( N/2 \)
  – Because any pattern \( Y = -Y \) for our purpose

• Two patterns \( y_1 \) and \( y_2 \) that differ in \( N/2 \) bits are **orthogonal**
  – Because \( y_1^T y_2 = 0 \)

• For \( N = 2^M L \), where \( L \) is an odd number, there are at most \( 2^M \) orthogonal binary patterns
  – Others may be *almost* orthogonal
Two orthogonal 4-bit patterns

- Patterns are local minima (stationary and stable)
  - No other local minima exist
  - But patterns perfectly confusible for recall
Two non-orthogonal 4-bit patterns

- Patterns are local minima (stationary and stable)
  - No other local minima exist
  - Actual *wells* for patterns
    - Patterns may be perfectly recalled!
  - Note $K > 0.14 N$
Three orthogonal 4-bit patterns

- All patterns are local minima (stationary)
  - But recall from perturbed patterns is random
Three non-orthogonal 4-bit patterns

- Patterns in the corner are not recalled
  - They end up being attracted to the -1,-1 pattern
  - Note some “ghosts” ended up in the “well” of other patterns
- So one of the patterns has stronger recall than the other two
Four orthogonal 4-bit patterns

- All patterns are stationary, but none are stable
  - Total wipe out
Four nonorthogonal 4-bit patterns

- One stable pattern
  - “Collisions” when the ghost of one pattern occurs next to another
How many patterns can we store?

- Hopfield: For a network of \( N \) neurons can store up to \( 0.14N \) patterns

- Apparently a fuzzy statement
  - What does it really mean to say “stores” \( 0.14N \) patterns?
    - Stationary? Stable? No other local minima?

- \( N=4 \) may not be a good case (\( N \) too small)
A 6-bit pattern

- Perfectly stationary and stable
- But many spurious local minima...
  - Which are “fake” memories
Two orthogonal 6-bit patterns

- Perfectly stationary and stable
- Several spurious “fake-memory” local minima...
  - Figure overstates the problem: actually a 3-D Kmap
Two non-orthogonal 6-bit patterns

- Perfectly stationary and stable
- Some spurious “fake-memory” local minima..
  - But every stored pattern has “bowl”
  - *Fewer* spurious minima than for the orthogonal case
Three non-orthogonal 6-bit patterns

- Note: Cannot have 3 or more orthogonal 6-bit patterns..
- Patterns are perfectly stationary and stable ($K > 0.14N$)
- Some spurious “fake-memory” local minima..
  - But every stored pattern has “bowl”
  - Fewer spurious minima than for the orthogonal 2-pattern case
Four *non*-orthogonal 6-bit patterns

- Patterns are perfectly stationary for $K > 0.14N$
- *Fewer* spurious minima than for the orthogonal 2-pattern case
  - Most fake-looking memories are in fact ghosts.
Six non-orthogonal 6-bit patterns

- Breakdown largely due to interference from “ghosts”
- But multiple patterns are stationary, and often stable
  - For $K >> 0.14N$
More visualization..

• Lets inspect a few 8-bit patterns
  – Keeping in mind that the Karnaugh map is now a 4-dimensional tesseract
One 8-bit pattern

- It's actually cleanly stored, but there are a few spurious minima
Two orthogonal 8-bit patterns

- Both have regions of attraction
- Some spurious minima
Two non-orthogonal 8-bit patterns

- Actually have fewer spurious minima
  - Not obvious from visualization..
Four orthogonal 8-bit patterns

- Successfully stored
Four non-orthogonal 8-bit patterns

- Stored with interference from ghosts..
Eight orthogonal 8-bit patterns

- Wipeout
Eight non-orthogonal 8-bit patterns

- Nothing stored
  - Neither stationary nor stable
Observations

• Many “parasitic” patterns
  – Undesired patterns that also become stable or attractors

• Apparently a capacity to store more than 0.14N patterns
Parasitic patterns can occur because sums of odd numbers of stored patterns are also stable for Hebbian learning:

\[ y_{\text{parasite}} = \text{sign}(y_a + y_b + y_c) \]

They are also from other random local energy minima from the weights matrices themselves.
Capacity

• Seems possible to store $K > 0.14N$ patterns
  – i.e. obtain a weight matrix $W$ such that $K > 0.14N$ patterns are stationary
  – Possible to make more than $0.14N$ patterns at-least 1-bit stable

• Patterns that are non-orthogonal easier to remember
  – I.e. patterns that are closer are easier to remember than patterns that are farther!!

• Can we attempt to get greater control on the process than Hebbian learning gives us?
  – Can we do better than Hebbian learning?
    • Better capacity and fewer spurious memories?
Story so far

• A Hopfield network is a loopy binary net with symmetric connections
  – Neurons try to align themselves to the local field caused by other neurons

• Given an initial configuration, the patterns of neurons in the net will evolve until the “energy” of the network achieves a local minimum
  – The network acts as a content-addressable memory
    • Given a damaged memory, it can evolve to recall the memory fully

• The network must be designed to store the desired memories
  – Memory patterns must be stationary and stable on the energy contour

• Network memory can be trained by Hebbian learning
  – Guarantees that a network of N bits trained via Hebbian learning can store 0.14N random patterns with less than 0.4% probability that they will be unstable

• However, empirically it appears that we may sometimes be able to store more than 0.14N patterns
Bold Claim

• I can *always* store (upto) N orthogonal patterns such that they are stationary!

  – Why?

• I can avoid spurious memories by adding some noise during recall!