Postscript on Supervised Neural Networks

One of my students, Robert, asked:

> Maybe I'm missing something fundamental, but supervised neural networks seem equivalent to fitting a pre-defined function to some given data, then extrapolating – what's the difference?

I agree with Robert. The supervised neural networks we have studied so far are simply parameterized nonlinear functions which can be fitted to data. Hopefully you will agree with another comment that Robert made:

> Unsupervised networks seem much more interesting than their supervised counterparts. I'm amazed that it works!

42

Hopfield Networks

We have now spent three chapters studying the single neuron. The time has come to connect multiple neurons together, making the output of one neuron be the input to another, so as to make neural networks.

Neural networks can be divided into two classes on the basis of their connectivity.

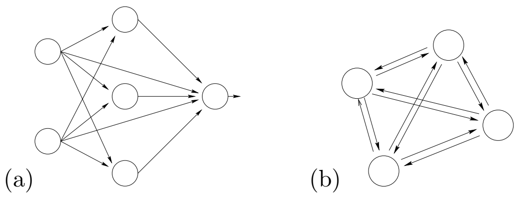


Figure 42.1. (a) A feedforward network. (b) A feedback network.

Feedforward networks. In a feedforward network, all the connections are directed such that the network forms a directed acyclic graph.

Feedback networks. Any network that is not a feedforward network will be called a feedback network.

In this chapter we will discuss a fully connected feedback network called the Hopfield network. The weights in the Hopfield network are constrained to be *symmetric*, i.e., the weight from neuron i to neuron j is equal to the weight from neuron j to neuron i.

Hopfield networks have two applications. First, they can act as *associative memories*. Second, they can be used to solve *optimization problems*. We will first discuss the idea of associative memory, also known as content-addressable memory.

▶ 42.1 Hebbian learning

In Chapter 38, we discussed the contrast between traditional digital memories and biological memories. Perhaps the most striking difference is the *associative* nature of biological memory.

A simple model due to Donald Hebb (1949) captures the idea of associative memory. Imagine that the weights between neurons whose activities are *positively correlated* are *increased*:

$$\frac{\mathrm{d}w_{ij}}{\mathrm{d}t} \sim \text{Correlation}(x_i, x_j). \tag{42.1}$$

Now imagine that when stimulus m is present (for example, the smell of a banana), the activity of neuron m increases; and that neuron n is associated with another stimulus, n (for example, the sight of a yellow object). If these two stimuli – a yellow sight and a banana smell – co-occur in the environment, then the Hebbian learning rule (42.1) will increase the weights w_{nm} and w_{mn}. This means that when, on a later occasion, stimulus n occurs in isolation, making the activity x_n large, the positive weight from n to m will cause neuron m also to be activated. Thus the response to the sight of a yellow object is an automatic association with the smell of a banana. We could call this 'pattern completion'. No teacher is required for this associative memory to work. No signal is needed to indicate that a correlation has been detected or that an association should be made. The unsupervised, local learning algorithm and the unsupervised, local activity rule spontaneously produce associative memory.

This idea seems so simple and so effective that it must be relevant to how memories work in the brain.

▶ 42.2 Definition of the binary Hopfield network

Convention for weights. Our convention in general will be that w_{ij} denotes the connection *from* neuron j *to* neuron i.

Architecture. A Hopfield network consists of I neurons. They are fully connected through *symmetric, bidirectional* connections with weights $w_{ij} = w_{ji}$. There are no self-connections, so $w_{ii} = 0$ for all i. Biases w_{i0} may be included (these may be viewed as weights from a neuron '0' whose activity is permanently $x_0 = 1$). We will denote the activity of neuron i (its output) by x_i.

Activity rule. Roughly, a Hopfield network's activity rule is for each neuron to update its state as if it were a single neuron with the threshold activation function

$$x(a) = \Theta(a) \equiv \begin{cases} 1 & a \geq 0 \\ -1 & a < 0. \end{cases} \qquad (42.2)$$

Since there is *feedback* in a Hopfield network (every neuron's output is an input to all the other neurons) we will have to specify an order for the updates to occur. The updates may be synchronous or asynchronous.

Synchronous updates – all neurons compute their activations

$$a_i = \sum_j w_{ij} x_j \qquad (42.3)$$

then update their states simultaneously to

$$x_i = \Theta(a_i). \qquad (42.4)$$

Asynchronous updates – one neuron at a time computes its activation and updates its state. The sequence of selected neurons may be a fixed sequence or a random sequence.

The properties of a Hopfield network may be sensitive to the above choices.

Learning rule. The learning rule is intended to make a set of desired *memories* $\{\mathbf{x}^{(n)}\}$ be *stable states* of the Hopfield network's activity rule. Each memory is a binary pattern, with $x_i \in \{-1, 1\}$.

(a)

```
moscow------russia
lima----------peru
london-----england
tokyo--------japan
edinburgh-scotland
ottawa------canada
oslo--------norway
stockholm---sweden
paris-------france
```

(b)

```
moscow---::::::::   ⟹   moscow------russia
::::::::::--canada  ⟹   ottawa------canada
```

(c)

```
otowa-------canada  ⟹   ottawa------canada
egindurrh-sxotland  ⟹   edinburgh-scotland
```

Figure 42.2. Associative memory (schematic). (a) A list of desired memories. (b) The first purpose of an associative memory is pattern completion, given a partial pattern. (c) The second purpose of a memory is error correction.

The weights are set using the *sum of outer products* or *Hebb rule*,

$$w_{ij} = \eta \sum_n x_i^{(n)} x_j^{(n)}, \tag{42.5}$$

where η is an unimportant constant. To prevent the largest possible weight from growing with N we might choose to set $\eta = 1/N$.

Exercise 42.1.[1] Explain why the value of η is not important for the Hopfield network defined above.

▶ 42.3 Definition of the continuous Hopfield network

Using the identical architecture and learning rule we can define a Hopfield network whose activities are real numbers between -1 and 1.

Activity rule. A Hopfield network's activity rule is for each neuron to update its state as if it were a single neuron with a sigmoid activation function. The updates may be synchronous or asynchronous, and involve the equations

$$a_i = \sum_j w_{ij} x_j \tag{42.6}$$

and

$$x_i = \tanh(a_i). \tag{42.7}$$

The learning rule is the same as in the binary Hopfield network, but the value of η becomes relevant. Alternatively, we may fix η and introduce a *gain* $\beta \in (0, \infty)$ into the activation function:

$$x_i = \tanh(\beta a_i). \tag{42.8}$$

Exercise 42.2.[1] Where have we encountered equations 42.6, 42.7, and 42.8 before?

▶ 42.4 Convergence of the Hopfield network

The hope is that the Hopfield networks we have defined will perform associative memory recall, as shown schematically in figure 42.2. We hope that the activity rule of a Hopfield network will take a partial memory or a corrupted memory, and perform pattern completion or error correction to restore the original memory.

But why should we expect *any* pattern to be stable under the activity rule, let alone the desired memories?

We address the continuous Hopfield network, since the binary network is a special case of it. We have already encountered the activity rule (42.6, 42.8) when we discussed variational methods (section 33.2): when we approximated the spin system whose energy function was

$$E(\mathbf{x};\mathbf{J}) = -\frac{1}{2}\sum_{m,n} J_{mn}x_m x_n - \sum_n h_n x_n \tag{42.9}$$

with a separable distribution

$$Q(\mathbf{x};\mathbf{a}) = \frac{1}{Z_Q}\exp\left(\sum_n a_n x_n\right) \tag{42.10}$$

and optimized the latter so as to minimize the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})E(\mathbf{x};\mathbf{J}) - \sum_{\mathbf{x}} Q(\mathbf{x};\mathbf{a})\ln\frac{1}{Q(\mathbf{x};\mathbf{a})}, \tag{42.11}$$

we found that the pair of iterative equations

$$a_m = \beta\left(\sum_n J_{mn}\bar{x}_n + h_m\right) \tag{42.12}$$

and

$$\bar{x}_n = \tanh(a_n) \tag{42.13}$$

were guaranteed to decrease the variational free energy

$$\beta\tilde{F}(\mathbf{a}) = \beta\left(-\frac{1}{2}\sum_{m,n} J_{mn}\bar{x}_m\bar{x}_n - \sum_n h_n\bar{x}_n\right) - \sum_n H_2^{(e)}(q_n). \tag{42.14}$$

If we simply replace J by w, $\bar{x}$ by x, and h_n by w_{i0}, we see that the equations of the Hopfield network are identical to a set of mean-field equations that minimize

$$\beta\tilde{F}(\mathbf{x}) = -\beta\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{W}\mathbf{x} - \sum_i H_2^{(e)}[(1+x_i)/2]. \tag{42.15}$$

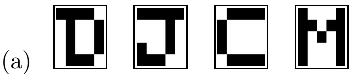
There is a general name for a function that decreases under the dynamical evolution of a system and that is bounded below: such a function is a *Lyapunov function* for the system. It is useful to be able to prove the existence of Lyapunov functions: if a system has a Lyapunov function then its dynamics are bound to settle down to a *fixed point*, which is a local minimum of the Lyapunov function, or a *limit cycle*, along which the Lyapunov function is a constant. Chaotic behaviour is not possible for a system with a Lyapunov function. If a system has a Lyapunov function then its state space can be divided into *basins of attraction*, one basin associated with each attractor.

So, the continuous Hopfield network's activity rules (if implemented asynchronously) have a Lyapunov function. This Lyapunov function is a convex function of each parameter a_i so a Hopfield network's dynamics will always converge to a stable fixed point.

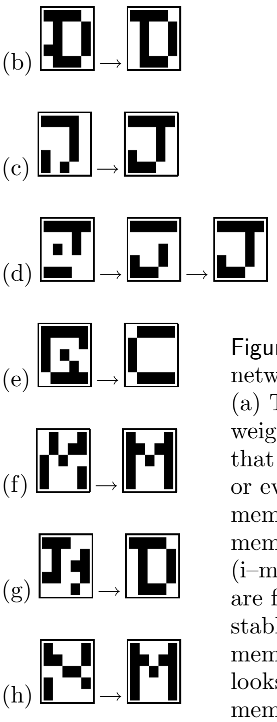
This convergence proof depends crucially on the fact that the Hopfield network's connections are *symmetric*. It also depends on the updates being made asynchronously.

Exercise 42.3.[*2*, p.520] Show by constructing an example that if a feedback network does not have symmetric connections then its dynamics may fail to converge to a fixed point.

Exercise 42.4.[*2*, p.521] Show by constructing an example that if a Hopfield network is updated synchronously that, from some initial conditions, it may fail to converge to a fixed point.



```
  .  0  0  0  0 -2  2 -2  2  2 -2  0  0  0  2  0  0 -2  0  2  2  0  0 -2 -2
  0  .  4  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  4  2 -2
  0  4  .  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  4  2 -2
  0  4  4  .  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  4  2 -2
  0  0  0  0  .  2 -2 -2  2 -2  2 -4  0  0 -2  4 -4 -2  0 -2  2  0  0 -2  2
 -2 -2 -2 -2  2  .  0  0  0  0  4 -2  2 -2  0  2 -2  0 -2  0  0 -2 -2  0  4
  2 -2 -2 -2 -2  0  .  0  0  4  0  2  2 -2  4 -2  2  0 -2  4  0 -2 -2  0  0
 -2 -2 -2 -2 -2  0  0  .  0  0  0  2  2  2  0 -2  2  4  2  0  0 -2 -2  0  0
  2 -2 -2 -2  2  0  0  0  .  0  0 -2  2  2  0  2 -2  0  2  0  4 -2 -2 -4  0
  2 -2 -2 -2 -2  0  4  0  0  .  0  2  2 -2  4 -2  2  0 -2  4  0 -2 -2  0  0
 -2 -2 -2 -2  2  4  0  0  0  0  . -2  2 -2  0  2 -2  0 -2  0  0 -2 -2  0  4
  0  0  0  0 -4 -2  2  2 -2  2 -2  .  0  0  2 -4  4  2  0  2 -2  0  0  2 -2
  0 -4 -4 -4  0  2  2  2  2  2  2  0  .  0  2  0  0  2  0  2  2 -4 -4 -2  2
  0  0  0  0  0 -2 -2  2  2 -2 -2  0  0  . -2  0  0  2  4 -2  2  0  0 -2 -2
  2 -2 -2 -2 -2  0  4  0  0  4  0  2  2 -2  . -2  2  0 -2  4  0 -2 -2  0  0
  0  0  0  0  4  2 -2 -2  2 -2  2 -4  0  0 -2  . -4 -2  0 -2  2  0  0 -2  2
  0  0  0  0 -4 -2  2  2 -2  2 -2  4  0  0  2 -4  .  2  0  2 -2  0  0  2 -2
 -2 -2 -2 -2 -2  0  0  4  0  0  0  2  2  2  0 -2  2  .  2  0  0 -2 -2  0  0
  0  0  0  0  0 -2 -2  2  2 -2 -2  0  0  4 -2  0  0  2  . -2  2  0  0 -2 -2
  2 -2 -2 -2 -2  0  4  0  0  4  0  2  2 -2  4 -2  2  0 -2  .  0 -2 -2  0  0
  2 -2 -2 -2  2  0  0  0  4  0  0 -2  2  2  0  2 -2  0  2  0  . -2 -2 -4  0
  0  4  4  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  .  4  2 -2
  0  4  4  4  0 -2 -2 -2 -2 -2 -2  0 -4  0 -2  0  0 -2  0 -2 -2  4  .  2 -2
 -2  2  2  2 -2  0  0  0 -4  0  0  2 -2 -2  0 -2  2  0 -2  0 -4  2  2  .  0
 -2 -2 -2 -2  2  4  0  0  0  0  4 -2  2 -2  0  2 -2  0 -2  0  0 -2 -2  0  .
```



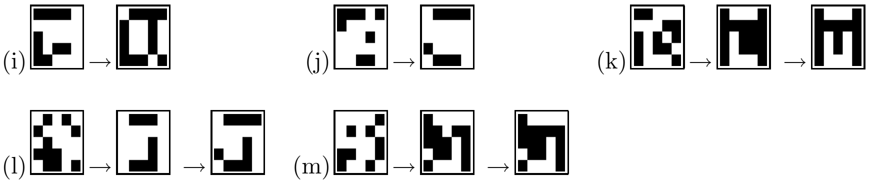


Figure 42.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to stable states other than the four memories; in (i), the stable state looks like a mixture of two memories, 'D' and 'J'; stable state (j) is like a mixture of 'J' and 'C'; in (k), we find a corrupted version of the 'M' memory (two bits distant); in (l) a corrupted version of 'J' (four bits distant) and in (m), a state which looks spurious until we recognize that it is the inverse of the stable state (l).

▶ 42.5 The associative memory in action

Figure 42.3 shows the dynamics of a 25-unit binary Hopfield network that has learnt four patterns by Hebbian learning. The four patterns are displayed as five by five binary images in figure 42.3a. For twelve initial conditions, panels (b–m) show the state of the network, iteration by iteration, all 25 units being updated asynchronously in each iteration. For an initial condition randomly perturbed from a memory, it often only takes one iteration for all the errors to be corrected. The network has more stable states in addition to the four desired memories: the inverse of any stable state is also a stable state; and there are several stable states that can be interpreted as mixtures of the memories.

Brain damage

The network can be severely damaged and still work fine as an associative memory. If we take the 300 weights of the network shown in figure 42.3 and randomly set 50 or 100 of them to zero, we still find that the desired memories are attracting stable states. Imagine a digital computer that still works fine even when 20% of its components are destroyed!

▷ Exercise 42.5.$^{[3C]}$ Implement a Hopfield network and confirm this amazing robust error-correcting capability.

More memories

We can squash more memories into the network too. Figure 42.4a shows a set of five memories. When we train the network with Hebbian learning, all five memories are stable states, even when 26 of the weights are randomly deleted (as shown by the ‘x’s in the weight matrix). However, the basins of attraction are smaller than before: figures 42.4(b–f) show the dynamics resulting from randomly chosen starting states close to each of the memories (3 bits flipped). Only three of the memories are recovered correctly.

If we try to store too many patterns, the associative memory fails catastrophically. When we add a sixth pattern, as shown in figure 42.5, only one of the patterns is stable; the others all flow into one of two spurious stable states.

▶ 42.6 The continuous-time continuous Hopfield network

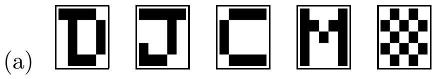
The fact that the Hopfield network's properties are not robust to the minor change from asynchronous to synchronous updates might be a cause for concern; can this model be a useful model of biological networks? It turns out that once we move to a continuous-time version of the Hopfield networks, this issue melts away.

We assume that each neuron's activity x_i is a continuous function of time $x_i(t)$ and that the activations $a_i(t)$ are computed instantaneously in accordance with

$$a_i(t) = \sum_j w_{ij} x_j(t). \tag{42.16}$$

The neuron's response to its activation is assumed to be mediated by the differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} x_i(t) = -\frac{1}{\tau}(x_i(t) - f(a_i)), \tag{42.17}$$



```
 .-1 1-1 1 x x-3 3 x x-1 1-1 x-1 1-3 x 1 3-1 1 x-1
-1 . 3 5-1-1-3-1-3-1-3 1 x 1-3 1-1-1-1-1-3 5 3 3-3
 1 3 . 3 1-3-1 x-1-3-1-1 x-1-1-1 1-3 1-3-1 3 5 1-1
-1 5 3 .-1-1-3-1-3-1-3 1-5 1-3 1-1-1-1-1-3 5 x 3-3
 1-1 1-1 . 1-1-3 x x 3-5 1-1-1 3 x-3 1-3 3-1 1-3 3
 x-1-3-1 1 .-1 1-1 1 3-1 1-1-1 3-3 1 x 1 x-1-3 1 3
 x-3-1-3-1-1 .-1 1 3 1 1 3-3 5-3 3-1-1 x 1-3-1-1 1
-3-1 x-1-3 1-1 .-1 1-1 3 1 x-1-1 1 5 1 1-1 x-3 1-1
 3-3-1-3 x-1 1-1 .-1 1-3 3 1 1 1-1-1 3-1 5-3-1 x 1
 x-1-3-1 x 1 3 1-1 .-1 3 1-1 3-1 x 1-3 5-1-1-3 1-1
 x-3-1-3 3 3 1-1 1-1 .-3 3-3 1 1-1-1-1-1 1-3-1-1 5
-1 1-1 1-5-1 1 3-3 3-3 .-1 1 1-3 3 x-1 3-3 1-1 3-3
 1 x x-5 1 1 3 1 3 1 3-1 .-1 3-1 1 1 1 1 3-5-3-3 3
-1 1-1 1-1-1-3 x 1-1-3 1-1 . x 1-1 3 3-1 1 1-1-1-3
 x-3-1-3-1-1 5-1 1 3 1 1 3 x . x 3-1-1 3 1-3-1-1 1
-1 1-1 1 3 3-3-1 1-1 1-3-1 1 x .-5-1-1-1 1 1-1-1 1
 1-1 1-1 x-3 3 1-1 x-1 3 1-1 3-5 . 1 1 1-1-1 1 1-1
-3-1-3-1-3 1-1 5-1 1-1 x 1 3-1-1 1 . 1 1-1-1-3 1-1
 x-1 1-1 1 x-1 1 3-3-1-1 1 3-1-1 1 1 .-3 3-1 1-3-1
 1-1-3-1-3 1 x 1-1 5-1 3 1-1 3-1 1 1-3 . x-1-3 1-1
 3-3-1-3 3 x 1-1 5-1 1-3 3 1 1 1-1-1 3 x .-3-1-5 1
-1 5 3 5-1-1-3 x-3-1-3 1-5 1-3 1-1-1-1-1-3 . 3 x-3
 1 3 5 x 1-3-1-3-1-3-1-1-3-1-1-1 1-3 1-3-1 3 . 1-1
 x 3 1 3-3 1-1 1 x 1-1 3-3-1-1-1 1 1-3 1-5 x 1 .-1
-1-3-1-3 3 3 1-1 1-1 5-3 3-3 1 1-1-1-1-1 1-3-1-1 .
```

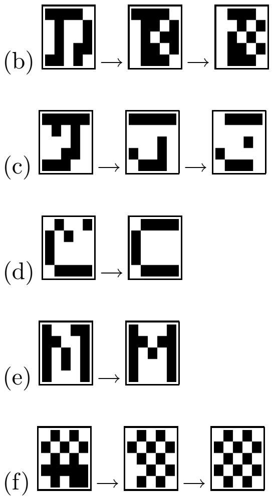


Figure 42.4. Hopfield network storing five memories, and suffering deletion of 26 of its 300 weights. (a) The five memories, and the weights of the network, with deleted weights shown by ‘x’. (b–f) Initial states that differ by three random bits from a memory: some are restored, but some converge to other states.

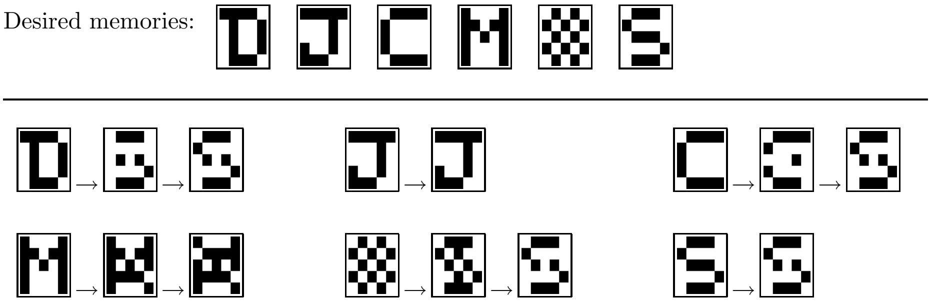


Figure 42.5. An overloaded Hopfield network trained on six memories, most of which are not stable.

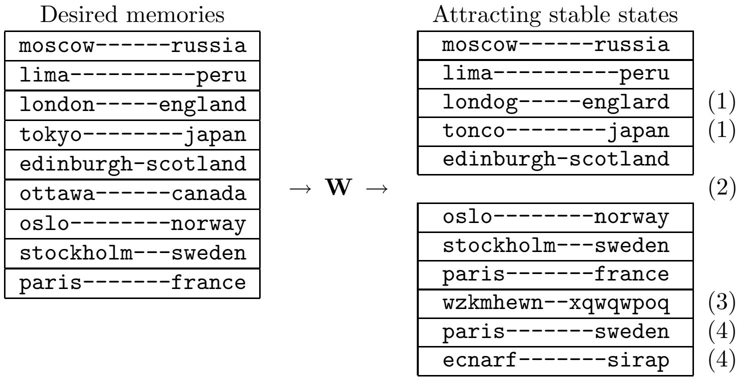


Figure 42.6. Failure modes of a Hopfield network (highly schematic). A list of desired memories, and the resulting list of attracting stable states. Notice (1) some memories that are retained with a small number of errors; (2) desired memories that are completely lost (there is no attracting stable state at the desired memory or near it); (3) spurious stable states unrelated to the original list; (4) spurious stable states that are confabulations of desired memories.

where $f(a)$ is the activation function, for example $f(a) = \tanh(a)$. For a steady activation a_i, the activity $x_i(t)$ relaxes exponentially to $f(a_i)$ with time-constant τ.

Now, here is the nice result: as long as the weight matrix is symmetric, this system has the variational free energy (42.15) as its Lyapunov function.

▷ Exercise 42.6.[1] By computing $\frac{\mathrm{d}}{\mathrm{d}t}\tilde{F}$, prove that the variational free energy $\tilde{F}(\mathbf{x})$ is a Lyapunov function for the continuous-time Hopfield network.

It is particularly easy to prove that a function L is a Lyapunov function if the system's dynamics perform steepest descent on L, with $\frac{\mathrm{d}}{\mathrm{d}t}x_i(t) \propto \frac{\partial}{\partial x_i}L$. In the case of the continuous-time continuous Hopfield network, it is not quite so simple, but every component of $\frac{\mathrm{d}}{\mathrm{d}t}x_i(t)$ does have the same sign as $\frac{\partial}{\partial x_i}\tilde{F}$, which means that with an appropriately defined metric, the Hopfield network dynamics do perform steepest descents on $\tilde{F}(\mathbf{x})$.

▶ 42.7 The capacity of the Hopfield network

One way in which we viewed learning in the single neuron was as communication – communication of the labels of the training data set from one point in time to a later point in time. We found that the capacity of a linear threshold neuron was 2 bits per weight.

Similarly, we might view the Hopfield associative memory as a communication channel (figure 42.6). A list of desired memories is encoded into a set of weights $\mathbf{W}$ using the Hebb rule of equation (42.5), or perhaps some other learning rule. The receiver, receiving the weights $\mathbf{W}$ only, finds the stable states of the Hopfield network, which he interprets as the original memories. This communication system can fail in various ways, as illustrated in the figure.

1. Individual bits in some memories might be corrupted, that is, a stable state of the Hopfield network is displaced a little from the desired memory.

2. Entire memories might be absent from the list of attractors of the network; or a stable state might be present but have such a small basin of attraction that it is of no use for pattern completion and error correction.

3. Spurious additional memories unrelated to the desired memories might be present.

4. Spurious additional memories derived from the desired memories by operations such as mixing and inversion may also be present.

Of these failure modes, modes 1 and 2 are clearly undesirable, mode 2 especially so. Mode 3 might not matter so much as long as each of the desired memories has a large basin of attraction. The fourth failure mode might in some contexts actually be viewed as beneficial. For example, if a network is required to memorize examples of valid sentences such as 'John loves Mary' and 'John gets cake', we might be happy to find that 'John loves cake' was also a stable state of the network. We might call this behaviour 'generalization'.

The capacity of a Hopfield network with I neurons might be defined to be the number of random patterns N that can be stored without failure-mode 2 having substantial probability. If we also require failure-mode 1 to have tiny probability then the resulting capacity is much smaller. We now study these alternative definitions of the capacity.

The capacity of the Hopfield network – stringent definition

We will first explore the information storage capabilities of a binary Hopfield network that learns using the Hebb rule by considering the stability of just one bit of one of the desired patterns, assuming that the state of the network is set to that desired pattern $\mathbf{x}^{(n)}$. We will assume that the patterns to be stored are randomly selected binary patterns.

The activation of a particular neuron i is

$$a_i = \sum_j w_{ij} x_j^{(n)}, \tag{42.18}$$

where the weights are, for $i \neq j$,

$$w_{ij} = x_i^{(n)} x_j^{(n)} + \sum_{m \neq n} x_i^{(m)} x_j^{(m)}. \tag{42.19}$$

Here we have split $\mathbf{W}$ into two terms, the first of which will contribute 'signal', reinforcing the desired memory, and the second 'noise'. Substituting for w_{ij}, the activation is

$$\begin{aligned} a_i &= \sum_{j \neq i} x_i^{(n)} x_j^{(n)} x_j^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)} & (42.20) \\ &= (I-1) x_i^{(n)} + \sum_{j \neq i} \sum_{m \neq n} x_i^{(m)} x_j^{(m)} x_j^{(n)}. & (42.21) \end{aligned}$$

The first term is $(I-1)$ times the desired state $x_i^{(n)}$. If this were the only term, it would keep the neuron firmly clamped in the desired state. The second term is a sum of $(I-1)(N-1)$ random quantities $x_i^{(m)} x_j^{(m)} x_j^{(n)}$. A moment's reflection confirms that these quantities are independent random binary variables with mean 0 and variance 1.

Thus, considering the statistics of a_i under the ensemble of random patterns, we conclude that a_i has mean $(I-1)x_i^{(n)}$ and variance $(I-1)(N-1)$.

For brevity, we will now assume I and N are large enough that we can neglect the distinction between I and $I-1$, and between N and $N-1$. Then we can restate our conclusion: a_i is Gaussian-distributed with mean $Ix_i^{(n)}$ and variance IN.

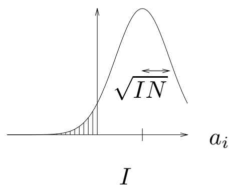


Figure 42.7. The probability density of the activation a_i in the case $x_i^{(n)} = 1$; the probability that bit i becomes flipped is the area of the tail.

What then is the probability that the selected bit is stable, if we put the network into the state $\mathbf{x}^{(n)}$? The probability that bit i will flip on the first iteration of the Hopfield network's dynamics is

$$P(i \text{ unstable}) = \Phi\left(-\frac{I}{\sqrt{IN}}\right) = \Phi\left(-\frac{1}{\sqrt{N/I}}\right), \tag{42.22}$$

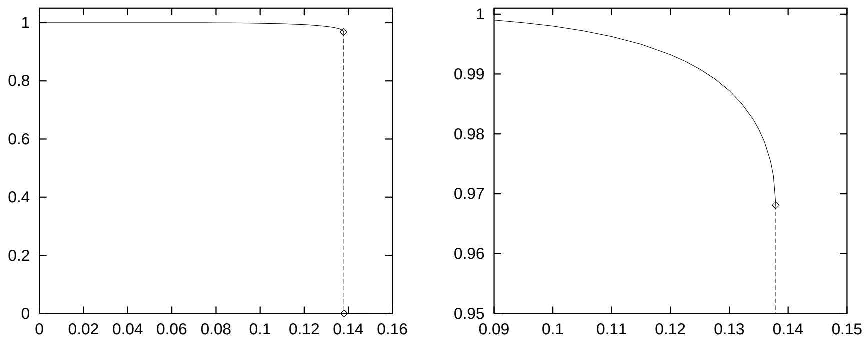


Figure 42.8. Overlap between a desired memory and the stable state nearest to it as a function of the loading fraction N/I. The overlap is defined to be the scaled inner product $\sum_i x_i x_i^{(n)}/I$, which is 1 when recall is perfect and zero when the stable state has 50% of the bits flipped. There is an abrupt transition at $N/I = 0.138$, where the overlap drops from 0.97 to zero.

where

$$\Phi(z) \equiv \int_{-\infty}^{z} \mathrm{d}z \, \frac{1}{\sqrt{2\pi}} \, e^{-z^2/2}. \tag{42.23}$$

The important quantity N/I is the ratio of the number of patterns stored to the number of neurons. If, for example, we try to store $N \simeq 0.18I$ patterns in the Hopfield network then there is a chance of 1% that a specified bit in a specified pattern will be unstable on the first iteration.

We are now in a position to derive our first capacity result, for the case where no corruption of the desired memories is permitted.

▷ Exercise 42.7.[2] Assume that we wish all the desired patterns to be completely stable – we don't want any of the bits to flip when the network is put into any desired pattern state – and the total probability of any error at all is required to be less than a small number ϵ. Using the approximation to the error function for large z,

$$\Phi(-z) \simeq \frac{1}{\sqrt{2\pi}} \frac{e^{-z^2/2}}{z}, \tag{42.24}$$

show that the maximum number of patterns that can be stored, $N_{\max}$, is

$$N_{\max} \simeq \frac{I}{4 \ln I + 2 \ln(1/\epsilon)}. \tag{42.25}$$

If, however, we allow a small amount of corruption of memories to occur, the number of patterns that can be stored increases.

The statistical physicists' capacity

The analysis that led to equation (42.22) tells us that if we try to store $N \simeq 0.18I$ patterns in the Hopfield network then, starting from a desired memory, about 1% of the bits will be unstable on the first iteration. Our analysis does not shed light on what is expected to happen on subsequent iterations. The flipping of these bits might make some of the other bits unstable too, causing an increasing number of bits to be flipped. This process might lead to an avalanche in which the network's state ends up a long way from the desired memory.

In fact, when N/I is large, such avalanches do happen. When N/I is small, they tend not to – there is a stable state near to each desired memory. For the limit of large I, Amit *et al.* (1985) have used methods from statistical physics to find numerically the transition between these two behaviours. There is a sharp discontinuity at

$$N_{\text{crit}} = 0.138I. \tag{42.26}$$

Below this critical value, there is likely to be a stable state near every desired memory, in which a small fraction of the bits are flipped. When N/I exceeds 0.138, the system has only spurious stable states, known as *spin glass states*, none of which is correlated with any of the desired memories. Just below the critical value, the fraction of bits that are flipped when a desired memory has evolved to its associated stable state is 1.6%. Figure 42.8 shows the overlap between the desired memory and the nearest stable state as a function of N/I.

Some other transitions in properties of the model occur at some additional values of N/I, as summarized below.

For all N/I, stable spin glass states exist, uncorrelated with the desired memories.

For $N/I > 0.138$, these spin glass states are the only stable states.

For $N/I \in (0, 0.138)$, there are stable states close to the desired memories.

For $N/I \in (0, 0.05)$, the stable states associated with the desired memories have lower energy than the spurious spin glass states.

For $N/I \in (0.05, 0.138)$, the spin glass states dominate – there are spin glass states that have lower energy than the stable states associated with the desired memories.

For $N/I \in (0, 0.03)$, there are additional *mixture* states, which are combinations of several desired memories. These stable states do not have as low energy as the stable states associated with the desired memories.

In conclusion, the capacity of the Hopfield network with I neurons, if we define the capacity in terms of the abrupt discontinuity discussed above, is $0.138I$ random binary patterns, each of length I, each of which is received with 1.6% of its bits flipped. In bits, this capacity is

$$0.138I^2 \times (1 - H_2(0.016)) = 0.122\, I^2 \text{ bits.} \tag{42.27}$$

This expression for the capacity omits a smaller negative term of order $N \log_2 N$ bits, associated with the arbitrary order of the memories.

Since there are $I^2/2$ weights in the network, we can also express the capacity as *0.24 bits per weight.*

▶ 42.8 Improving on the capacity of the Hebb rule

The capacities discussed in the previous section are the capacities of the Hopfield network whose weights are set using the Hebbian learning rule. We can do better than the Hebb rule by defining an objective function that measures how well the network stores all the memories, and minimizing it.

For an associative memory to be useful, it must be able to correct at least one flipped bit. Let's make an objective function that measures whether flipped bits tend to be restored correctly. Our intention is that, for every neuron i in the network, the weights to that neuron should satisfy this rule:

> for every pattern $\mathbf{x}^{(n)}$, if the neurons other than i are set correctly to $x_j = x_j^{(n)}$, then the activation of neuron i should be such that its preferred output is $x_i = x_i^{(n)}$.

Is this rule a familiar idea? Yes, it is precisely what we wanted the single neuron of Chapter 39 to do. Each pattern $\mathbf{x}^{(n)}$ defines an input, target pair for the single neuron i. And it defines an input, target pair for all the other neurons too.

```
w = x' * x ;         # initialize the weights using Hebb rule

for l = 1:L          # loop L times

        for i=1:I             #
          w(i,i) = 0 ;        #  ensure the self-weights are zero.
        end                   #

        a  = x * w      ;    # compute all activations
        y  = sigmoid(a) ;    # compute all outputs
        e  = t - y      ;    # compute all errors
        gw = x' * e     ;    # compute the gradients
        gw = gw + gw'   ;    # symmetrize gradients

        w  = w + eta * ( gw - alpha * w ) ;  # make step

endfor
```

Algorithm 42.9. `Octave` source code for optimizing the weights of a Hopfield network, so that it works as an associative memory. cf. algorithm 39.5. The data matrix `x` has I columns and N rows. The matrix `t` is identical to `x` except that -1s are replaced by 0s.

So, just as we defined an objective function (39.11) for the training of a single neuron as a classifier, we can define

$$G(\mathbf{W}) = -\sum_i \sum_n t_i^{(n)} \ln y_i^{(n)} + (1 - t_i^{(n)}) \ln(1 - y_i^{(n)}) \tag{42.28}$$

where

$$t_i^{(n)} = \begin{cases} 1 & x_i^{(n)} = 1 \\ 0 & x_i^{(n)} = -1 \end{cases} \tag{42.29}$$

and

$$y_i^{(n)} = \frac{1}{1 + \exp(-a_i^{(n)})}, \text{ where } a_i^{(n)} = \sum w_{ij} x_j^{(n)} . \tag{42.30}$$

We can then steal the algorithm (algorithm 39.5, p.478) which we wrote for the single neuron, to write an algorithm for optimizing a Hopfield network, algorithm 42.9. The convenient syntax of `Octave` requires very few changes; the extra lines enforce the constraints that the self-weights w_{ii} should all be zero and that the weight matrix should be symmetrical ($w_{ij} = w_{ji}$).

As expected, this learning algorithm does a better job than the one-shot Hebbian learning rule. When the six patterns of figure 42.5, which cannot be memorized by the Hebb rule, are learned using algorithm 42.9, all six patterns become stable states.

Exercise 42.8.$^{[4C]}$ Implement this learning rule and investigate empirically its capacity for memorizing random patterns; also compare its avalanche properties with those of the Hebb rule.

▶ 42.9 Hopfield networks for optimization problems

Since a Hopfield network's dynamics minimize an energy function, it is natural to ask whether we can map interesting optimization problems onto Hopfield networks. Biological data processing problems often involve an element of *constraint satisfaction* – in scene interpretation, for example, one might wish to infer the spatial location, orientation, brightness and texture of each visible element, and which visible elements are connected together in objects. These inferences are constrained by the given data and by prior knowledge about continuity of objects.

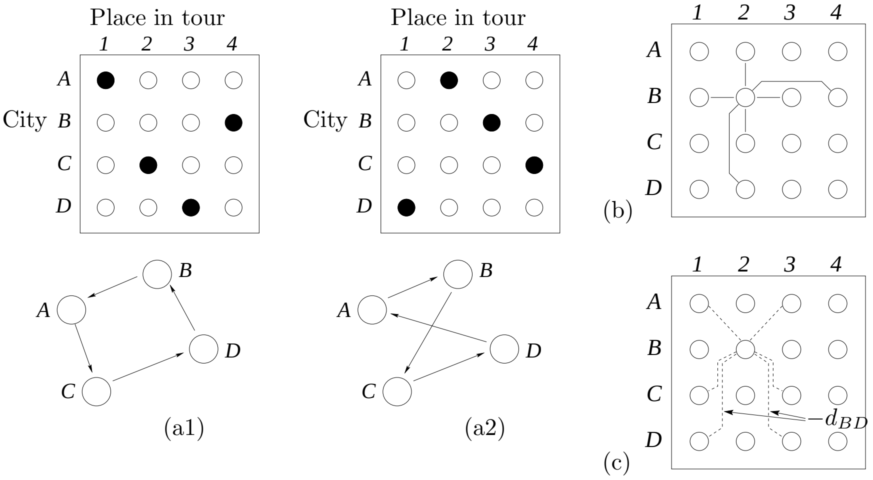


Figure 42.10. Hopfield network for solving a travelling salesman problem with $K = 4$ cities. (a1,2) Two solution states of the 16-neuron network, with activites represented by black = 1, white = 0; and the tours corresponding to these network states. (b) The negative weights between node $B2$ and other nodes; these weights enforce validity of a tour. (c) The negative weights that embody the distance objective function.

Hopfield and Tank (1985) suggested that one might take an interesting constraint satisfaction problem and design the weights of a binary or continuous Hopfield network such that the settling process of the network would minimize the objective function of the problem.

The travelling salesman problem

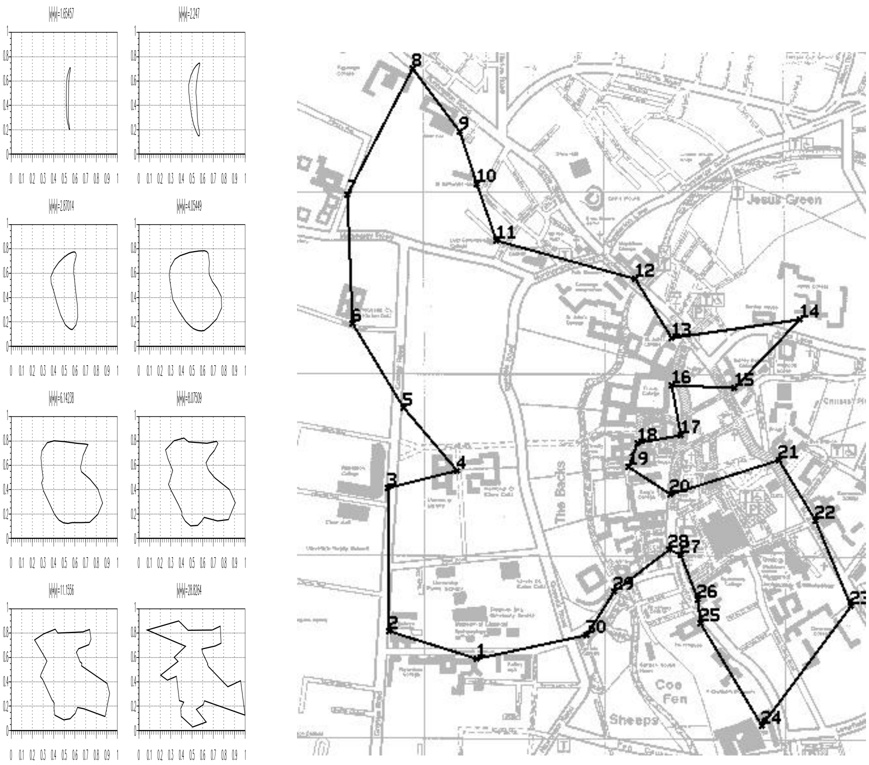
A classic constraint satisfaction problem to which Hopfield networks have been applied is the travelling salesman problem.

A set of K cities is given, and a matrix of the $K(K-1)/2$ distances between those cities. The task is to find a closed tour of the cities, visiting each city once, that has the smallest total distance. The travelling salesman problem is equivalent in difficulty to an NP-complete problem.

The method suggested by Hopfield and Tank is to represent a tentative solution to the problem by the state of a network with $I = K^2$ neurons arranged in a square, with each neuron representing the hypothesis that a particular city comes at a particular point in the tour. It will be convenient to consider the states of the neurons as being between 0 and 1 rather than -1 and 1. Two solution states for a four-city travelling salesman problem are shown in figure 42.10a.

The weights in the Hopfield network play two roles. First, they must define an energy function which is minimized only when the state of the network represents a *valid* tour. A valid state is one that looks like a permutation matrix, having exactly one '1' in every row and one '1' in every column. This rule can be enforced by putting large negative weights between any pair of neurons that are in the same row or the same column, and setting a positive bias for all neurons to ensure that K neurons do turn on. Figure 42.10b shows the negative weights that are connected to one neuron, '$B2$', which represents the statement 'city B comes second in the tour'.

Second, the weights must encode the objective function that we want to minimize – the total distance. This can be done by putting negative weights proportional to the appropriate distances between the nodes in adjacent columns. For example, between the B and D nodes in adjacent columns, the weight would be $-d_{BD}$. The negative weights that are connected to neuron $B2$ are shown in figure 42.10c. The result is that when the network is in a valid state, its total energy will be the total distance of the corresponding



(a) (b)

Figure 42.11. (a) Evolution of the state of a continuous Hopfield network solving a travelling salesman problem using Aiyer's (1991) graduated non-convexity method; the state of the network is projected into the two-dimensional space in which the cities are located by finding the centre of mass for each point in the tour, using the neuron activities as the mass function. (b) The travelling scholar problem. The shortest tour linking the 27 Cambridge Colleges, the Engineering Department, the University Library, and Sree Aiyer's house. From Aiyer (1991).

tour, plus a constant given by the energy associated with the biases.

Now, since a Hopfield network minimizes its energy, it is hoped that the binary or continuous Hopfield network's dynamics will take the state to a minimum that is a valid tour and which might be an optimal tour. This hope is not fulfilled for large travelling salesman problems, however, without some careful modifications. We have not specified the size of the weights that enforce the tour's validity, relative to the size of the distance weights, and setting this scale factor poses difficulties. If 'large' validity-enforcing weights are used, the network's dynamics will rattle into a valid state with little regard for the distances. If 'small' validity-enforcing weights are used, it is possible that the distance weights will cause the network to adopt an *invalid* state that has lower energy than any valid state. Our original formulation of the energy function puts the objective function and the solution's validity in potential conflict with each other. This difficulty has been resolved by the work of Sree Aiyer (1991), who showed how to modify the distance weights so that they would not interfere with the solution's validity, and how to define a continuous Hopfield network whose dynamics are at all times confined to a 'valid subspace'. Aiyer used a *graduated non-convexity* or *deterministic annealing* approach to find good solutions using these Hopfield networks. The deterministic annealing approach involves gradually increasing the gain β of the neurons in the network from 0 to ∞, at which point the state of the network corresponds to a valid tour. A sequence of trajectories generated by applying this method to a thirty-city travelling salesman problem is shown in figure 42.11a.

A solution to the 'travelling scholar problem' found by Aiyer using a continuous Hopfield network is shown in figure 42.11b.

▶ 42.10 Further exercises

▷ Exercise 42.9.[3] Storing two memories.

Two binary memories $\mathbf{m}$ and $\mathbf{n}$ ($m_i, n_i \in \{-1, +1\}$) are stored by Hebbian learning in a Hopfield network using

$$w_{ij} = \begin{cases} m_i m_j + n_i n_j & \text{for } i \neq j \\ 0 & \text{for } i = j. \end{cases} \tag{42.31}$$

The biases b_i are set to zero.

The network is put in the state $\mathbf{x} = \mathbf{m}$. Evaluate the activation a_i of neuron i and show that in can be written in the form

$$a_i = \mu m_i + \nu n_i. \tag{42.32}$$

By comparing the signal strength, μ, with the magnitude of the noise strength, $|\nu|$, show that $\mathbf{x} = \mathbf{m}$ is a stable state of the dynamics of the network.

The network is put in a state $\mathbf{x}$ differing in D places from $\mathbf{m}$,

$$\mathbf{x} = \mathbf{m} + 2\mathbf{d}, \tag{42.33}$$

where the perturbation $\mathbf{d}$ satisfies $d_i \in \{-1, 0, +1\}$. D is the number of components of $\mathbf{d}$ that are non-zero, and for each d_i that is non-zero, $d_i = -m_i$. Defining the overlap between $\mathbf{m}$ and $\mathbf{n}$ to be

$$o_{\mathbf{mn}} = \sum_{i=1}^{I} m_i n_i, \tag{42.34}$$

evaluate the activation a_i of neuron i again and show that the dynamics of the network will restore $\mathbf{x}$ to $\mathbf{m}$ if the number of flipped bits satisfies

$$D < \frac{1}{4}(I - |o_{\mathbf{mn}}| - 2). \tag{42.35}$$

How does this number compare with the maximum number of flipped bits that can be corrected by the optimal decoder, assuming the vector $\mathbf{x}$ is either a noisy version of $\mathbf{m}$ or of $\mathbf{n}$?

Exercise 42.10.[3] Hopfield network as a collection of binary classifiers. This exercise explores the link between unsupervised networks and supervised networks. If a Hopfield network's desired memories are all attracting stable states, then every neuron in the network has weights going to it that solve a classification problem personal to that neuron. Take the set of memories and write them in the form $\mathbf{x}'^{(n)}, x_i^{(n)}$, where $\mathbf{x}'$ denotes all the components $x_{i'}$ for all $i' \neq i$, and let $\mathbf{w}'$ denote the vector of weights $w_{ii'}$, for $i' \neq i$.

Using what we know about the capacity of the single neuron, show that it is almost certainly impossible to store more than $2I$ random memories in a Hopfield network of I neurons.

Lyapunov functions

Exercise 42.11.[3] Erik's puzzle. In a stripped-down version of Conway's game of life, cells are arranged on a square grid. Each cell is either alive or dead. Live cells do not die. Dead cells become alive if two or more of their immediate neighbours are alive. (Neighbours to north, south, east and west.) What is the smallest number of live cells needed in order that these rules lead to an entire $N \times N$ square being alive?

In a d-dimensional version of the same game, the rule is that if d neighbours are alive then you come to life. What is the smallest number of live cells needed in order that an entire $N \times N \times \cdots \times N$ hypercube becomes alive? (And how should those live cells be arranged?)

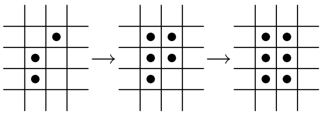


Figure 42.12. Erik's dynamics.

The southeast puzzle

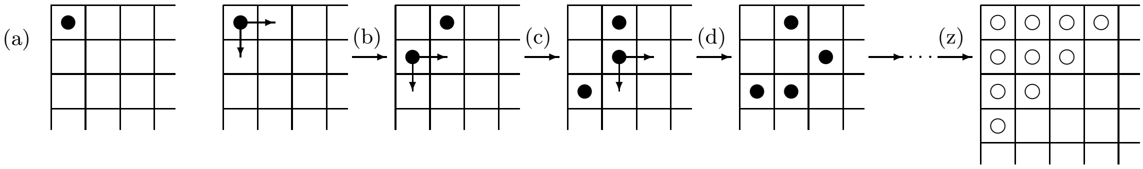


Figure 42.13. The southeast puzzle.

The southeast puzzle is played on a semi-infinite chess board, starting at its northwest (top left) corner. There are three rules:

1. In the starting position, one piece is placed in the northwest-most square (figure 42.13a).
2. It is not permitted for more than one piece to be on any given square.
3. At each step, you remove one piece from the board, and replace it with two pieces, one in the square immediately to the east, and one in the the square immediately to the south, as illustrated in figure 42.13b. Every such step increases the number of pieces on the board by one.

After move (b) has been made, either piece may be selected for the next move. Figure 42.13c shows the outcome of moving the lower piece. At the next move, either the lowest piece or the middle piece of the three may be selected; the uppermost piece may not be selected, since that would violate rule 2. At move (d) we have selected the middle piece. Now any of the pieces may be moved, except for the leftmost piece.

Now, here is the puzzle:

▷ Exercise 42.12.[4, p.521] Is it possible to obtain a position in which all the ten squares closest to the northwest corner, marked in figure 42.13z, are empty?

[Hint: this puzzle has a connection to data compression.]

▶ 42.11 Solutions

Solution to exercise 42.3 (p.508). Take a binary feedback network with 2 neurons and let $w_{12} = 1$ and $w_{21} = -1$. Then whenever neuron 1 is updated, it will match neuron 2, and whenever neuron 2 is updated, it will flip to the opposite state from neuron 1. There is no stable state.