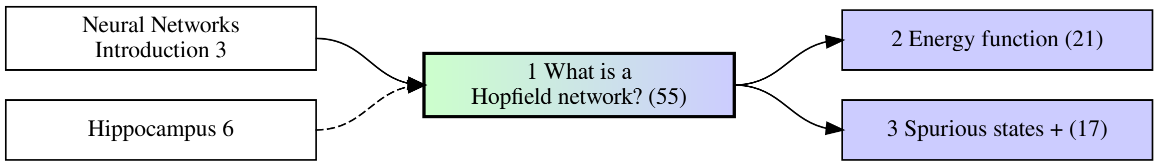
Hopfield Networks

— **Lecture Notes** —

Laurenz Wiskott
Institut für Neuroinformatik
Ruhr-Universität Bochum, Germany, EU

23 May 2022

— **Summary** —


Neural Networks Introduction 3
Hippocampus 6
1 What is a Hopfield network? (55)
2 Energy function (21)
3 Spurious states + (17)

1 What is a Hopfield network?

Contents

Sorry, these lecture notes lack references. This is all textbook knowledge.

1 What is a Hopfield network?

Learning material:[1]

- ☐
  - ● 4 min video 1.1 System Definition
  - ● 17 min video 1.2 System Dynamics
  - ● 7 min video 1.3.1 Storing One Pattern
  - ● 5 min video 1.3.2 Storing Multiple Patterns
  - ● 3 min video 1.4 Role of Self Connections
  - + 7 min video 1.5 Frustrated Systems
- ☐ Text below

[1]Generic instruction: Consider the (possibly nested) list of resources like a horizontal tree with an invisible root on the very left, and decide from left to right what you want to select to work through. The invisible root node has to be selected. For any selected parent node all children nodes marked with ■ or ● are mandatory and have to be selected. Children nodes marked with ☐ or ○ are optional and may be selected in addition to get a better understanding of the material. If a parent node has no mandatory child, then at least one optional child has to be selected. Children marked with + provide additional voluntary material that can be safely ignored, typically going beyond the scope of the section. Children of non-selected parents may be ignored. ■ and ☐ indicate children that cover (almost) the whole material of the section. Missing content might then be indicated by struck through references to the corresponding learning objectives. Items tend to be ordered by precedence and/or recommended temporal order from top to bottom, assuming that you prefer to first watch a video before reading through lecture notes. If a detailed table of content for videos or lecture notes is given, references to learning objectives might be provided in green, 1:30 should be read as 1 min and 30 seconds, and 1'30 should be read as page 1 at about 30% of the page. Video times may be linked directly to the indicated position in the video, but be aware that the video might be downloaded anew each time you click on a time. Resources without author name are usually authored by Laurenz Wiskott and his team.

Quizzes and Exercises: Deepen and test your understanding with the following

- Section 1 Exercises, Section 1 Solutions

1.1 System definition

See also 4 min video 1.1 System Definition

A Hopfield network is probably the most prominent example of a recurrent network, that means a network where the connections may form loops. It is an auto-associative network, meaning that is is able to complete stored patterns if initialized with corrupted versions, but we will get to that later. Usually recurrent networks are difficult to treat analytically. Hopfield networks are an exception, because they have symmetric weights, which means that one can define an energy or Lyapunov function for their dynamics. Hopfield networks are named after John Hopfield, whose great contribution to the field of neural networks is that he has introduced the energy function.

There are different variants of Hopfield networks. We consider here the discrete and noiseless case. Such **a Hopfield network consists of** a number, let's say N, of **units, see figure 1.1. The units are bipolar,** i.e. they can only assume the output values $y_i = \pm 1$. The units are **mutually connected with weights** w_{ij}**.** The **total input** z_i into a unit, which might also be referred to as its membrane potential, is given

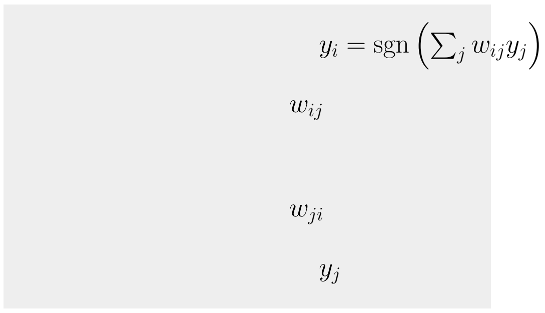


Figure 1.1: A Hopfield network with four units.

by the sum over the activity of all units times the respective weights, and the **output** y_i is then simply the sign of this, i.e.

$$♦ \quad z_i := \sum_j w_{ij} y_j \,, \tag{1.1}$$

$$♦ \quad y_i \rightarrow \text{sgn}(z_i) \,. \tag{1.2}$$

Note that **equation (1.2) does** not **have** an equal sign but **an arrow indicating an assignment** rather than equality. This is to avoid a recursive definition and reflects the fact that units have to be updated.

It is useful to visualize the states such a bipolar network can assume in state space, **see figure 1.2.** Since

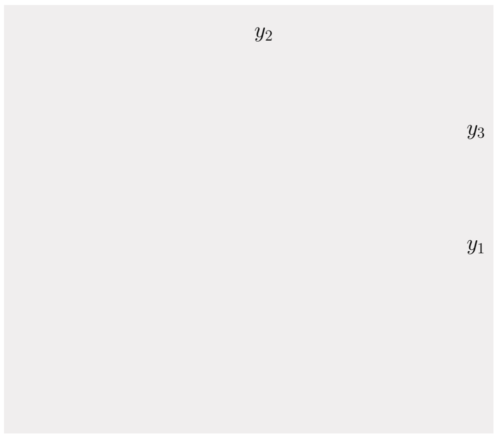


Figure 1.2: State space of a bipolar Hopfield network. Each unit has an axis. The network can only be at the corners of the hypercube. The red dotted arrow indicates the update of a single unit, which is a transition of one corner to a neighboring one.

each unit can assume only the values ± 1, **the states the network can assume as a whole are the corners of a hypercube.**

1.2 System dynamics

See also 17 min video 1.2 System Dynamics

A Hopfield network does not have input units in the sense of feedforward networks. Instead **all units are initialized to an initial value** before the dynamics is started. **Then**, to get the dynamics, **one has to update the units according to (1.2). Also the recall of a stored pattern is done by first initializing the units to some cue**, usually a corrupted version of the stored pattern, **and then running the recurrent dynamics until convergence.** The update of units can be done in different ways:

Synchronous update: All units are repeatedly updated simultaneously.

Asynchronous ordered update: All units are updated individually in a given order that repeats over and over again.

Asynchronous random update: All units are updated individually in a random order that differs for each sweep through all units.

This can also be visualized **in state space. Updating just one unit means jumping from one corner to a neighboring one, see figure 1.2.** Updating the whole network can lead to larger jumps.

Figure 1.3: Transitions in state space for two very simple networks with two units and three different update schedules.

As a very simple example consider a network with just two units and the weights $w_{12} = -1$ and $w_{21} = +1$, see **figure 1.3** top. For such a simple network we can figure out all transitions by hand. For instance, if the network is in state $--$ and we update unit 1, it switches to $+1$ because unit 2 has value $y_2 = -1$ and the weight from unit 2 to unit 1 has value $w_{12} = -1$ so that $y_1 \to w_{12} y_2 = (-1) \cdot (-1) = +1$.

For synchronous update we have to simultaneously update also unit 2, resulting in $y_2 \to w_{21} y_1 = (+1) \cdot (-1) = -1$. Note that for the update of unit 2 the value of unit 1 is still -1, because both units are updated simultaneously and the change of one unit does not yet affect the other one. Thus, in synchronous update, state $--$ goes over to state $-+$, which is indicated in the figure by a dotted arrow.

For asynchronous random update it is more appropriate to consider the update of single units instead of the whole network as a transition. If we start with state $--$ and update unit 1 it switches to $+1$, as seen above, and the system goes over to state $+-$. If we update unit 2 instead, it stays at -1, because $y_2 \to w_{21} y_1 = (+1) \cdot (-1) = -1$, and thus the transition goes from state $--$ onto itself. In the figure, the two different transitions, depending on whether we update unit 1 or 2, are marked with a little number indicating the unit updated.

Once we have worked out all transitions for asynchronous random update it is easy to work out the transitions for asynchronous ordered update for the whole network. We simply follow the transitions for random update in the given order. For instance, if we assume the order is always unit 1 followed by unit 2, we can read from the graph for random update that the system goes from state $--$ over state $+-$ to state $++$.

A similarly simple network with symmetric weights, $w_{12} = w_{21} = -1$, and its transitions are shown in figure 1.3 bottom. With asynchrounous update it has two stable states, namely $-+$ and $+-$. It is intuitively clear that two units connected with negative weights should tend to have opposite value. Likewise one can easily infer that two units connected with positive weights should tend to have the same value.

These two simple examples illustrate a number of properties of the dynamics that also apply to larger networks:

- **A Hopfield network can show** different types of dynamics: **stationary states** (states $+-$ and $-+$ for network 2), **limit cycles** (e.g. a four-cycle for network 1 and a two-cycle for network 2 with synchronous update), **and irregular dynamics** (network 1 with asynchronous random update).
- **A Hopfield network with symmetric weights and asynchronous update is guaranteed to converge into a stationary state.** With non-symmetric weights this is not the case.
- **A Hopfield network with symmetric weights and synchronous update is guaranteed to converge into a stationary state or into a limit cycle of length two.**
- **A Hopfield network is symmetric with respect to a flip of all units.** In the examples shown this would exchange states $--$ and $++$ and states $+-$ and $-+$ but would leave the pattern of transitions unchanged.
- **Two units connected with negative weights tend to have opposite value; two units connected with positive weights tend to have the same value.**

We see that a key property that stabilizes the dynamics of a Hopfield network and makes it useful is the symmetry of the weights. From now on we consider only networks with symmetric weights.

1.3 Storing patterns

After having considered the dynamics of a Hopfield network a bit, the question arises how to choose the weights. We have learned already that it is advantageous to have symmetric weights. But is there a way to choose the weights such that a particular pattern becomes a stationary point of the dynamics, a so-called fixed point?

1.3.1 Storing one pattern

See also 7 min video 1.3.1 Storing One Pattern

We have stated that two negatively coupled units tend to have different values and two positively coupled units tend to have the same value. We can use this property to initialize the weights such that a particular pattern $\boldsymbol{s}$ (and its inverse) is stable. A simple rule that would do that is

$$\Diamond \quad w_{ij} = \frac{1}{N} s_i s_j \quad \forall\, i, j\,. \tag{1.3}$$

Such a rule generates positive weights among all positive units and among all negative units and generates negative weights between positive and negative units. The $1/N$ is a constant introduced for mathematical convenience but irrelevant for the dynamics. It cancels when summing over all units, see (1.6).

It is quite intuitive that such weights would stabilize the stored pattern. We can verify this mathematically by considering a network initialized with the stored pattern.

$$\begin{aligned}
y_i &\overset{(1.1,1.2)}{\rightarrow} \operatorname{sgn}\left(\sum_j w_{ij} s_j\right) && (1.4)\\
&\overset{(1.3)}{=} \operatorname{sgn}\left(\sum_j \frac{1}{N} s_i s_j s_j\right) && (1.5)\\
&= \operatorname{sgn}\left(s_i \underbrace{\sum_j \frac{1}{N} s_j s_j}_{=1}\right) && (1.6)\\
&= \operatorname{sgn}(s_i) && (1.7)\\
&= s_i\,. && (1.8)
\end{aligned}$$

Thus, an update of any unit would just reproduce the correct value.

If we initialize the network not with the original pattern but with a corrupted version $\boldsymbol{y}$, we still get the original pattern back as long as more than half of the units are correct, i.e. as long as $\sum_j s_j y_j > 0$, because

$$\begin{aligned}
\Diamond \quad y_i &\overset{(1.1,1.2)}{\rightarrow} \operatorname{sgn}\left(\sum_j w_{ij} y_j\right) && (1.9)\\
\Diamond \quad &\overset{(1.3)}{=} \operatorname{sgn}\left(\sum_j \frac{1}{N} s_i s_j y_j\right) && (1.10)\\
\Diamond \quad &= \operatorname{sgn}\left(s_i \underbrace{\sum_j \frac{1}{N} s_j y_j}_{>0}\right) && (1.11)\\
\Diamond \quad &= \operatorname{sgn}(s_i) && (1.12)\\
\Diamond \quad &= s_i\,. && (1.13)
\end{aligned}$$

If more than half of the units are initialized incorrectly, the inverse pattern gets reconstructed, since $\sum_j s_j y_j < 0$. Thus, if only one pattern is stored the system has exactly two fixed points, the stored pattern and its inverse.

1.3.2 Storing multiple patterns

See also 5 min video 1.3.2 Storing Multiple Patterns

If one wants to store multiple (M) patterns $\boldsymbol{s}^\mu$, indexed with μ, one simply adds the effect of all patterns

$$\blacklozenge \quad w_{ij} = \frac{1}{N}\sum_{\mu=1}^{M} s_i^\mu s_j^\mu \quad \forall\, i,j\,. \tag{1.14}$$

If the network is initialized with one of the patterns $\boldsymbol{s}^\nu$, its stability can again be estimated as above. We find

$$\lozenge \quad y_i \overset{(1.1,1.2)}{\rightarrow} \operatorname{sgn}\left(\sum_j w_{ij} s_j^\nu\right) \tag{1.15}$$

$$\lozenge \quad \overset{(1.14)}{=} \operatorname{sgn}\left(\sum_j \frac{1}{N}\sum_\mu s_i^\mu s_j^\mu s_j^\nu\right) \tag{1.16}$$

$$\lozenge \quad = \operatorname{sgn}\left(\sum_\mu s_i^\mu \frac{1}{N}\sum_j s_j^\mu s_j^\nu\right) \tag{1.17}$$

$$\lozenge \quad = \operatorname{sgn}\left(s_i^\nu \underbrace{\frac{1}{N}\sum_j s_j^\nu s_j^\nu}_{=1} + \sum_{\mu\neq\nu} s_i^\mu \frac{1}{N}\sum_j s_j^\mu s_j^\nu\right) \tag{1.18}$$

$$\lozenge \quad = \operatorname{sgn}\left(s_i^\nu + \underbrace{\sum_{\mu\neq\nu} s_i^\mu \frac{1}{N}\sum_j s_j^\mu s_j^\nu}_{\text{cross-talk term}}\right) \tag{1.19}$$

$$\lozenge \quad = s_i^\nu \qquad \text{if the cross-talk term is small}\,. \tag{1.20}$$

The *cross-talk term* (D: Übersprechterm) has to be less than one in magnitude to guarantee stability of unit i. If it has the same sign as s_i^ν, it may also be greater in magnitude.

If the patterns to be stored are orthogonal, i.e. $\sum_j s_j^\mu s_j^\nu = 0$, the cross-talk term is even zero. This means that multiple orthogonal patterns are all stable in a Hopfield network. They are usually even attractors, i.e. currupted versions get corrected to the originally stored version. Thus, a Hopfield network is an *auto-associative* (D: autoassoziativ) memory.

1.4 Role of self-connections

See also 3 min video 1.4 Role of Self Connections

The learning rule introduced above also generates positive self-connections

$$\diamond \qquad w_{ii} \overset{(1.14)}{=} \frac{1}{N} \sum_{\mu=1}^{M} s_i^\mu s_i^\mu = \frac{M}{N} \quad \forall\, i\,. \tag{1.21}$$

What is the effect these connections have on the dynamics of the system? If a unit has a positive value, the self-connection yields a positive term to the total input of the unit; if a unit has a negative value, the self-connection yields a negative term to the total input of the unit. Thus, the self-connection stabilizes the current value of a unit, no matter what it is. This is not necessarily a desired behavior, because a unit should actually switch if the total input coming from the other units suggests so. A unit should not keep its value just because of the self-connection. In other words, positive self-connections make a system more dependent on its initialization, which is not desired. Thus, it is generally better to set the self-connections to zero. However, in theoretical considerations it is often easier not to neglect the self-connections.

Depending on the number of units and the weights it can happen that the total input into a unit (without self-connection) is exactly zero. In such cases there is no evidence either way for the unit to switch. If that happens, we assume that the unit keeps its value, as would be the case with self-connection. This prevents such a unit from switching back and forth forever.

1.5 Frustrated systems

See also 7 min video 1.5 Frustrated Systems

Figure 1.4 illustrates two examples with three units. Since each unit is connected to two other units, the total input into a unit can be zero, so that it is not clear to which value it should switch. As argued above, in such cases we assume that the unit simply keeps its value.

What does the examples in figure 1.4 teach us? We notice that **network 2 is the inverse of network 1** in that **all weights have the opposite sign. This implies** that for a given network state the total input into a unit is inverted, which in turn leads to **an inverted switching behavior.** If one compares the transitions of the two networks, one notices that all directed arrows from one state to another one are inverted. The transistions from one state to itself do not follow this logic but just complement the updates that are left over.

We have argued above **two units connected with negative weights want to have opposite values; two units connected with positive weights want to have the same values.** In a larger network it is not always possible to satisfy all weights, there will usually be negative weights connecting units with the same value and positive weights connecting units with different value. **A system with** such **unsatisfied weights is called *frustrated*** (D: frustriert). We notice that network 1 is frustrated, because it is impossible to satisfy all weights. For instance, if unit 1 has value +1 then unit 2 should have value +1 as well, because it is positively linked to unit 1, unit 3 should have value +1 as well, because it is positively linked to unit 2, finally unit 1 should have value −1, because it is negatively linked to unit 3, which is not possible, because unit 1 has value +1 aready. Network 2 on the other hand can be satisfied completely, e.g. with the values +1, −1, and +1 for units 1, 2, and 3, respectively.

As a consequence of being frustrated, network 1 has more fixed points than network 2. We see from figure 1.4 that network 1 has 6 fixed points while network 2 has only two, where one is the inverse of the other one.

Figure 1.4: Transitions in state space for two very simple networks with three units and asynchronous random update.