Homework 6

CS 559 - Neural Networks - Fall 2019

Matteo Corain 650088272

November 4, 2019

1 Question 1

1.1 Generation of patterns and weights

The first step that has been taken for the solution of this problem has been to generate all the patterns represented by sequences of ± 1 with length m (in the considered case, $m = 4$ has been used). For this purpose, the recursive function `gen_patterns()` has been used, which generates the dispositions with replacement of the two values on a vector of length m. This function receives the vector x to be used, the position to be filled for the current recursive call, the length m of the vector and the array of patterns to be returned, and performs the following actions:

- It checks whether the x vector has already been filled or not (`pos == m`), in which case it appends the generated pattern to the list of patterns and returns it;
- Otherwise, it proceeds to set the `pos`-th value of the vector first to -1 and then to 1, each time recursively calling itself on the successive position.

Once that all pattern combinations have been computed, n independent samples (such that, if x is selected, then $-x$ is not selected) are chosen to be stored in the network (in our case, $n = 5$ was chosen). For this purpose, the array of pattern is randomly shuffled, then a loop is started (running until the length of the list of selected samples reaches the value of n), checking whether each pattern x (or its opposite $-x$) has already been chosen for the purpose and, if not, adding it to the list of selected samples.

Having set the random seed to a fixed value for reproducibility of the results, the following patterns have been generated for storage:

$$x_1 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}, x_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ +1 \end{bmatrix}, x_3 = \begin{bmatrix} -1 \\ -1 \\ +1 \\ -1 \end{bmatrix}, x_4 = \begin{bmatrix} -1 \\ -1 \\ +1 \\ +1 \end{bmatrix}, x_5 = \begin{bmatrix} -1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

Starting from those patterns, the weight matrix for the network has been generated using the standard approach, obtaining the following result:

$$W = \sum_{i=+1}^{n} x_i x_i^T = \begin{bmatrix} +5 & +3 & -1 & +1 \\ +3 & +5 & +1 & -1 \\ -1 & +1 & +5 & -1 \\ +1 & -1 & -1 & +5 \end{bmatrix}$$

1.2 Search of spurious patterns

For the search of spurious patterns, the `spurious_search()` has been introduced, which takes as arguments the list of all patterns, the list of selected samples and the weight matrix. For each pattern z, it checks whether z or $-z$ were chosen for storage and, in case they were not, if they are effectively spurious patterns, meaning that:

$$z = \phi(Wz), z \notin \{x_i\}$$

All spurious patterns are collected in a list that is finally returned. For the described case, the following spurious patterns have been identified:

$$z_1 = \begin{bmatrix} -1 \\ +1 \\ -1 \\ -1 \end{bmatrix}, z_2 = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}, z_3 = \begin{bmatrix} +1 \\ -1 \\ -1 \\ -1 \end{bmatrix}, z_4 = \begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$

1.3 Complete Python code

```
import numpy as np
import matplotlib.pyplot as plt

# Set the random seed for reproducibility
np.random.seed(2019)

def gen_patterns(x, pos, m, patterns):
    if pos == m:
        patterns.append(np.array(x))
        return patterns

    for i in [-1, 1]:
        x[pos] = i
        patterns = gen_patterns(x, pos + 1, m, patterns)

    return patterns

def spurious_search(patterns, samples, weights):
    spurious = []

    for z in patterns:
        # Check that pattern is spurious
        if (not any([np.array_equal(z, y) for y in samples]) and
                not any([np.array_equal(-z, y) for y in samples]) and
                np.array_equal(z, np.sign(weights @ z))):
            spurious.append(np.array(z))

    return spurious
```

```
n = 5
m = 4

# Generate patterns
patterns = gen_patterns(np.zeros((m, 1)), 0, m, [])

# Extract some independent samples
samples = []
shuffled_samples = np.random.permutation(patterns)

i = 0
while len(samples) < n:
    if (not any([np.array_equal(shuffled_samples[i], y) for y in samples]) and
           not any([np.array_equal(-shuffled_samples[i], y) for y in samples])):
        samples.append(patterns[i])
    i = i + 1

print("Selected patterns:")
for x in samples:
    print(x.transpose())

# Compute weights
weights = sum([x @ x.transpose() for x in samples])
print("Weight matrix:")
print(weights)

# Retrieve spurious patterns
spurious = spurious_search(patterns, samples, weights)
print("Spurious patterns:")
for x in spurious:
    print(x.transpose())
```

2 Question 2

2.1 Network representation

The proposed Hopfield network is characterized by the following parameters:

- Biases $\theta_0 = \theta_1 = \theta_2 = 0.5$;
- Weights $w_{12} = w_{21} = 2$;
- Weights $w_{13} = w_{31} = -1$;
- Weights $w_{23} = w_{32} = -1$;
- Weights $w_{11} = w_{22} = 0$;
- Weight $w_{33} = 1$.

Where neuron 1 indicates the topmost in the figure, neuron 2 the leftmost and neuron 3 the rightmost. These features may be formalized in a matrix form as:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}, W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

2.2 State energy computation

Given these two matrices, for each state x of the network the energy associated with that state is defined by:

$$E(x) = -x^T W x - 2x^T \theta$$

Applying the presented formula to each of the possible $2^n = 2^3 = 8$ states in which the considered network can be, we obtain the results in table 1.

x_1	x_2	x_3	$E(x)$
-1	-1	-1	+2
-1	-1	+1	-8
-1	+1	-1	+4
-1	+1	+1	+2
+1	-1	-1	+4
+1	-1	+1	+2
+1	+1	-1	-10
+1	+1	+1	-4

Table 1: Energy of each network state

2.3 Synchronous update

In the context of Hopfield networks, synchronous update prescribes that all the components of the state x of the considered network are updated at once, according to the update rule:

$$x \leftarrow \phi(Wx + \theta)$$

Where W is the weight matrix of the network and θ the bias vector. Applying the aforementioned update rule to each possible state of the network, it is possible to compute the successor state for each possible state of the network. Derivations are shown below:

- $x = [-1 \quad -1 \quad -1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} -0.5 \\ -0.5 \\ +1.5 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}$$

- $x = [-1 \quad -1 \quad +1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} -2.5 \\ -2.5 \\ +3.5 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}$$

- $x = [-1 \quad +1 \quad -1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} +3.5 \\ -0.5 \\ -0.5 \end{bmatrix}\right) = \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}$$

- $x = [-1 \quad +1 \quad +1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} +1.5 \\ -2.5 \\ +1.5 \end{bmatrix}\right) = \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix}$$

- $x = [+1 \quad -1 \quad -1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} -0.5 \\ +3.5 \\ -0.5 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix}$$

- $x = [+1 \quad -1 \quad +1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} -2.5 \\ +1.5 \\ +1.5 \end{bmatrix}\right) = \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix}$$

- $x = [+1 \quad +1 \quad -1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} +3.5 \\ +3.5 \\ -2.5 \end{bmatrix}\right) = \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix}$$

- $x = [+1 \quad +1 \quad +1]^T$:

$$x' = \phi(Wx + \theta) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \\ 2 & 0 & -1 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}\right) = \phi\left(\begin{bmatrix} +1.5 \\ +1.5 \\ -0.5 \end{bmatrix}\right) = \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix}$$

The obtained state transition diagram is shown in figure 1. As it can be seen, this form of update may generate instability, since there are some states that are never able to converge to a stable configuration. In this case, the *urstates* of the network are $[-1 \quad -1 \quad -1]^T$ and $[+1 \quad +1 \quad +1]^T$ (they have no predecessor states, including themselves), while the *stable states* of the network are $[-1 \quad -1 \quad +1]^T$ and $[+1 \quad +1 \quad -1]^T$ (they have no successor states, excluding themselves).

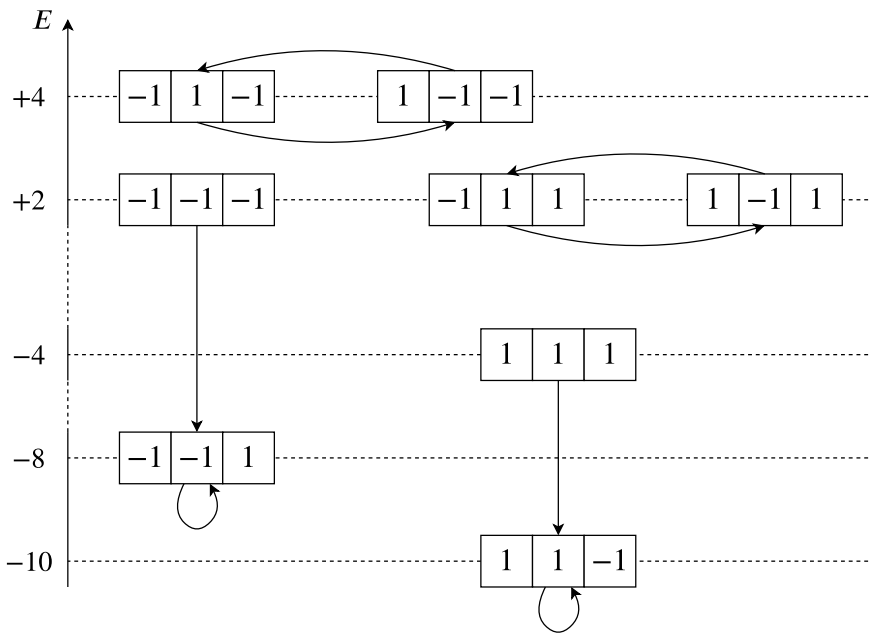

Figure 1: State transition diagram with synchronous update

2.4 Asynchronous update

In the context of Hopfield networks, asynchronous update prescribes that the components of the state x of the considered network are updated one at a time, fixing the values of the others. In formulas, we have that:

$$x_i \leftarrow W_i x + \theta_i, i = 1, \ldots, m$$

Where W_i denotes the i-th row of the weight matrix and θ_i the i-th component of the bias vector. In the case of asynchronous update, therefore, different updates are possible on different components, potentially making the network transition to different states. Derivations in the case of independent update of the single components are shown below:

- $x = [-1 \quad -1 \quad -1]^T$:

$$x_1' = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\end{bmatrix} + 0.5\right) = \phi(-0.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\-1\end{bmatrix}$$

$$x_2' = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\end{bmatrix} + 0.5\right) = \phi(-0.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\-1\end{bmatrix}$$

$$x_3' = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\end{bmatrix} + 0.5\right) = \phi(1.5) = +1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

- $x = [-1 \quad -1 \quad +1]^T$:

$$x'_1 = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}-1\\-1\\+1\end{bmatrix} + 0.5\right) = \phi(-2.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

$$x'_2 = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}-1\\-1\\+1\end{bmatrix} + 0.5\right) = \phi(-2.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

$$x'_3 = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}-1\\-1\\+1\end{bmatrix} + 0.5\right) = \phi(+3.5) = +1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

- $x = [-1 \quad +1 \quad -1]^T$:

$$x'_1 = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}-1\\+1\\-1\end{bmatrix} + 0.5\right) = \phi(+3.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\-1\end{bmatrix}$$

$$x'_2 = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}-1\\+1\\-1\end{bmatrix} + 0.5\right) = \phi(-0.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\-1\end{bmatrix}$$

$$x'_3 = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}-1\\+1\\-1\end{bmatrix} + 0.5\right) = \phi(-0.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\+1\\-1\end{bmatrix}$$

- $x = [-1 \quad +1 \quad +1]^T$:

$$x'_1 = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}-1\\+1\\+1\end{bmatrix} + 0.5\right) = \phi(+1.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\+1\end{bmatrix}$$

$$x'_2 = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}-1\\+1\\+1\end{bmatrix} + 0.5\right) = \phi(-2.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

$$x'_3 = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}-1\\+1\\+1\end{bmatrix} + 0.5\right) = \phi(+1.5) = +1 \Rightarrow x' = \begin{bmatrix}-1\\+1\\+1\end{bmatrix}$$

- $x = [+1 \quad -1 \quad -1]^T$:

$$x_1' = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}+1\\-1\\-1\end{bmatrix} + 0.5\right) = \phi(-0.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\-1\end{bmatrix}$$

$$x_2' = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}+1\\-1\\-1\end{bmatrix} + 0.5\right) = \phi(+3.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\-1\end{bmatrix}$$

$$x_3' = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}+1\\-1\\-1\end{bmatrix} + 0.5\right) = \phi(-0.5) = -1 \Rightarrow x' = \begin{bmatrix}+1\\-1\\-1\end{bmatrix}$$

- $x = [+1 \quad -1 \quad +1]^T$:

$$x_1' = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}+1\\-1\\+1\end{bmatrix} + 0.5\right) = \phi(-2.5) = -1 \Rightarrow x' = \begin{bmatrix}-1\\-1\\+1\end{bmatrix}$$

$$x_2' = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}+1\\-1\\+1\end{bmatrix} + 0.5\right) = \phi(+1.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\+1\end{bmatrix}$$

$$x_3' = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}+1\\-1\\+1\end{bmatrix} + 0.5\right) = \phi(+1.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\-1\\+1\end{bmatrix}$$

- $x = [+1 \quad +1 \quad -1]^T$:

$$x_1' = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix}0 & 2 & -1\end{bmatrix}\begin{bmatrix}+1\\+1\\-1\end{bmatrix} + 0.5\right) = \phi(+3.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\-1\end{bmatrix}$$

$$x_2' = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix}2 & 0 & -1\end{bmatrix}\begin{bmatrix}+1\\+1\\-1\end{bmatrix} + 0.5\right) = \phi(+3.5) = +1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\-1\end{bmatrix}$$

$$x_3' = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix}-1 & -1 & 1\end{bmatrix}\begin{bmatrix}+1\\+1\\-1\end{bmatrix} + 0.5\right) = \phi(-2.5) = -1 \Rightarrow x' = \begin{bmatrix}+1\\+1\\-1\end{bmatrix}$$

- $x = [+1 \quad +1 \quad +1]^T$:

$$x'_1 = \phi(W_1 x + \theta_1) = \phi\left(\begin{bmatrix} 0 & 2 & -1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} + 0.5\right) = \phi\left(+1.5\right) = +1 \Rightarrow x' = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix}$$

$$x'_2 = \phi(W_2 x + \theta_2) = \phi\left(\begin{bmatrix} 2 & 0 & -1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} + 0.5\right) = \phi\left(+1.5\right) = +1 \Rightarrow x' = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix}$$

$$x'_3 = \phi(W_3 x + \theta_3) = \phi\left(\begin{bmatrix} -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} + 0.5\right) = \phi\left(-0.5\right) = -1 \Rightarrow x' = \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix}$$

The obtained state transition diagram is shown in figure 2. With respect to the synchronous case, it is possible to notice that this state transition model does not present any form of instability. In this case, there are no *urstates* (each state has at least itself as a predecessor, i.e. presents at least a self-loop), while the *stable states* of the network are unchanged, $[-1 \quad -1 \quad +1]^T$ and $[+1 \quad +1 \quad -1]^T$ (they have no successor states, excluding themselves).

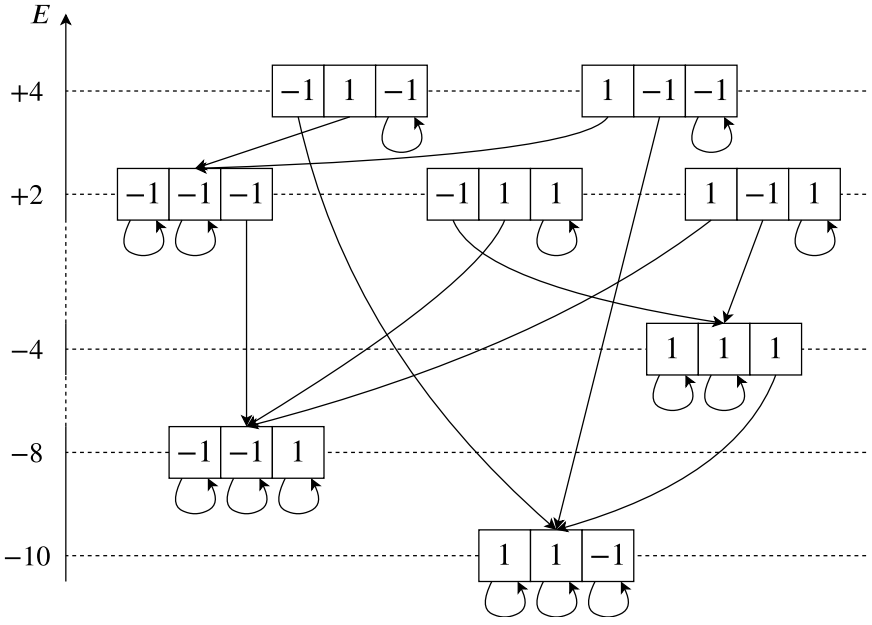

Figure 2: State transition diagram with asynchronous update