Hopfield Model

Computer modeling of complex systems

OVERVIEW

1 | Hopfield network
first task

2 | Asynchronous updates
second task

3 | Increasing patterns
extra task

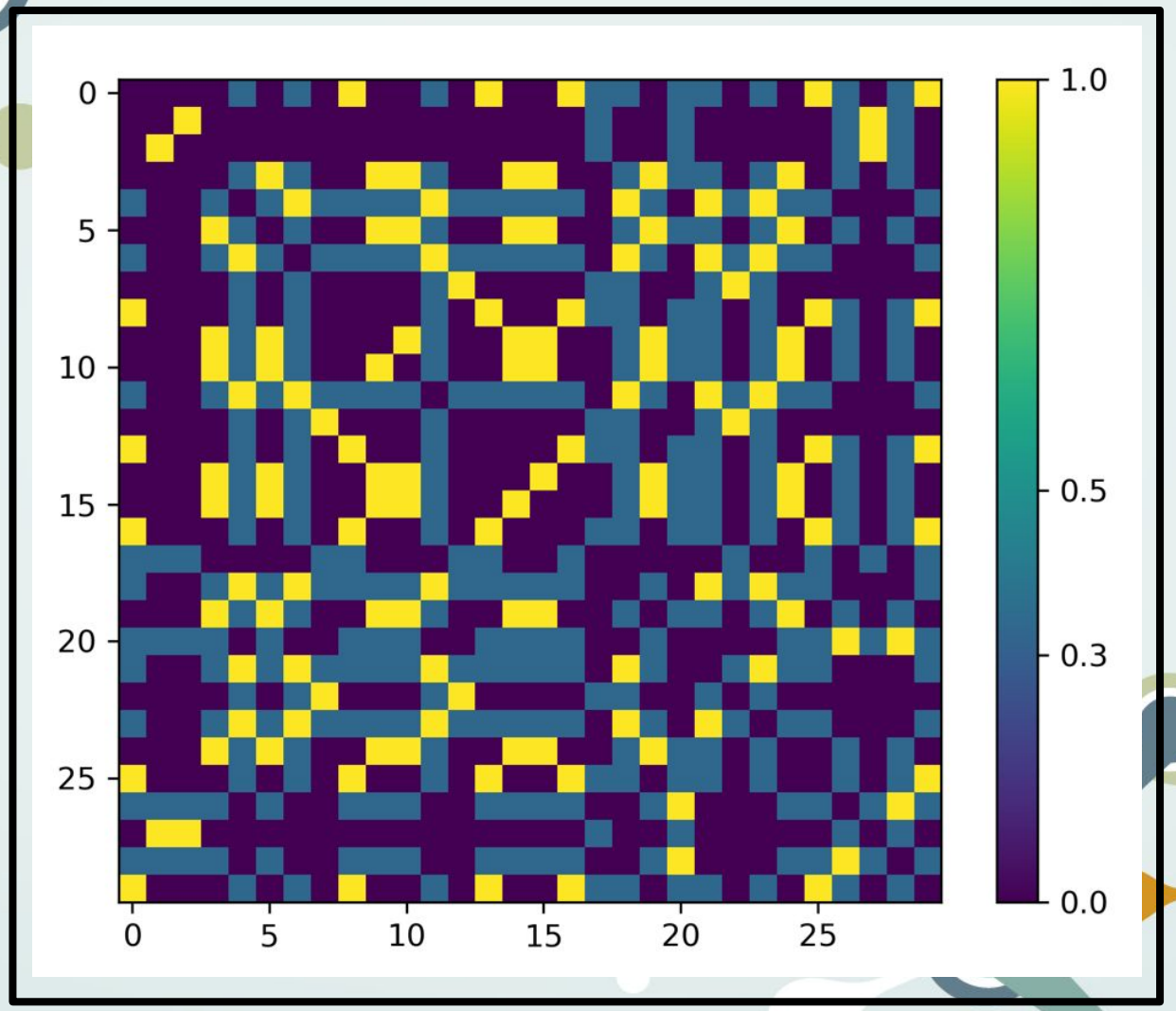
Receiving matrix: w

When learning n binary (spin) patterns x^{μ}:

$$\mathbf{w} = \frac{1}{n}\sum_{\mu=1}^{n} \mathbf{x}^{\mu} \otimes \mathbf{x}^{\mu} - \mathbf{1}$$

```python
def hopfield_network(training_matrices):
    """
    :param training_matrices: list of matrices, which we will use to training our network.
    :return: matrix, which describes network `after training`.
    """
    n_matrix = len(training_matrices)
    size_matrix = len(training_matrices[0].A[0])

    w_matrix = np.zeros((size_matrix, size_matrix))  # network after training
    for j in range(0, n_matrix):
        # create training pattern
        matrix = training_matrices[j]
        pattern = matrix.flatten()
        # update network
        w_matrix = w_matrix + np.outer(pattern, pattern) / n_matrix

    # Subtract identity matrix
    identity_matrix = np.identity(size_matrix)
    w_matrix = w_matrix - identity_matrix

    return np.matrix(w_matrix)
```

Martix w itself



Updating the network

The energy of ith spin can be written as:

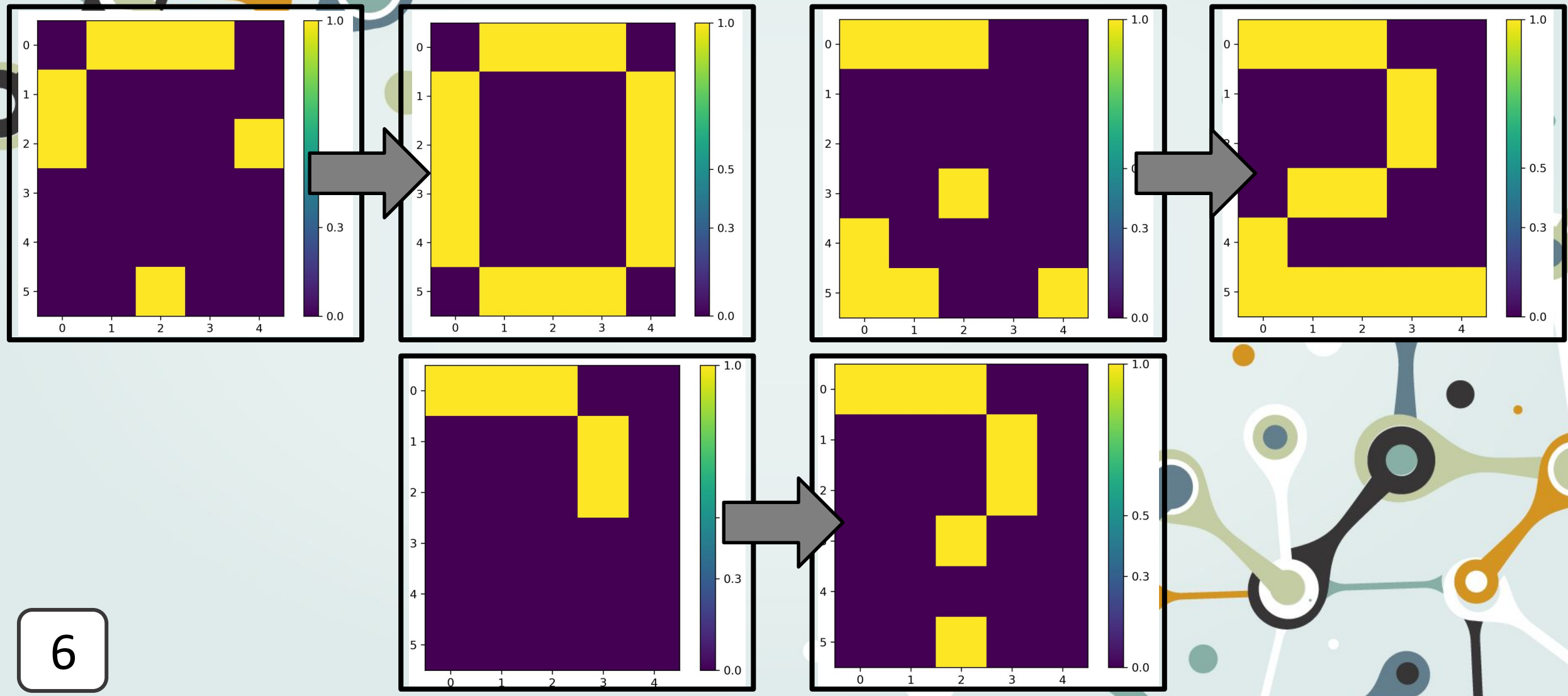
$$E_i = -\frac{1}{2} x_i \left(\sum_j w_{ij} x_j \right) = -\frac{1}{2} x_i h(i)$$

where h(i) is the field acting on the spin. If the product $x_i h(i)$ is negati
then the field would try to flip the spin:

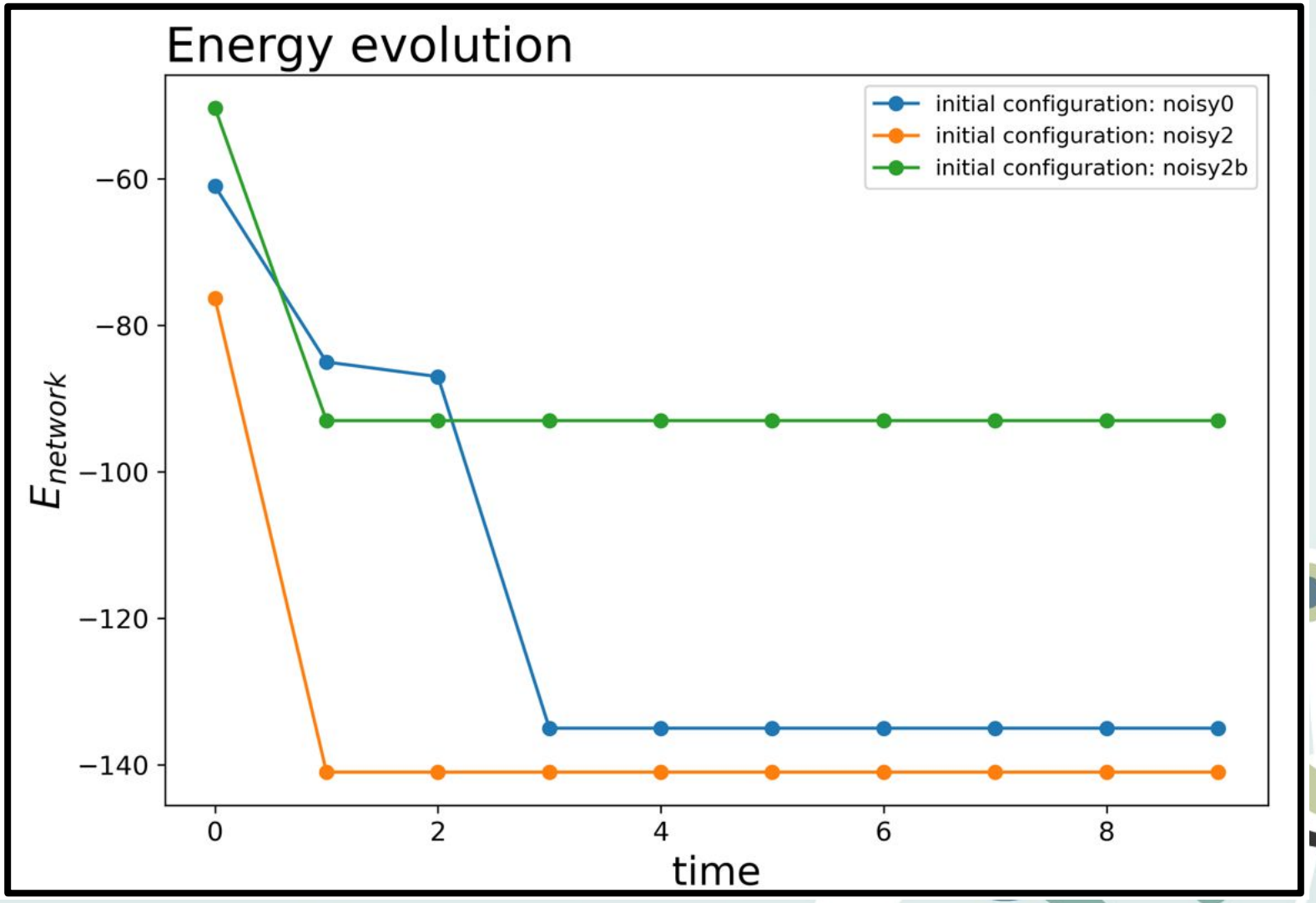
$$\mathbf{x}(t+1) = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x}(t))$$

```python
def test_network(pattern, w_matrix, steps=10):
    """
    :param pattern: pattern, which we will use as starting configuration
        of network.
    :param w_matrix: pattern, which we get after training.
    :param steps: number of updates, which we want make.
    :return: ending configuration and list contain energy of whole network after
        updating the network.
    """
    n_elements = pattern.size
    E_network_list = []
    for step in range(0, steps):
        # do one-step operation of hopfield network
        E_list = []
        for j in range(0, n_elements):
            x_i = pattern[0, j]
            # calculate: sum over j of `w_ij * x_j`
            h_i = w_matrix[j].dot(pattern.T)
            # compute energy
            E_i = float(- 0.5 * x_i * h_i)
            E_list.append(E_i)
        # update test matrix
        pattern = np.sign(w_matrix.dot(pattern.T).T)
        E_network = sum(E_list)
        E_network_list.append(E_network)

    return pattern, E_network_list
```

Results task 1



Energy Evolution



OVERVIEW

1 | Hopfield network

first task

2 | Asynchronous updates

second task

---

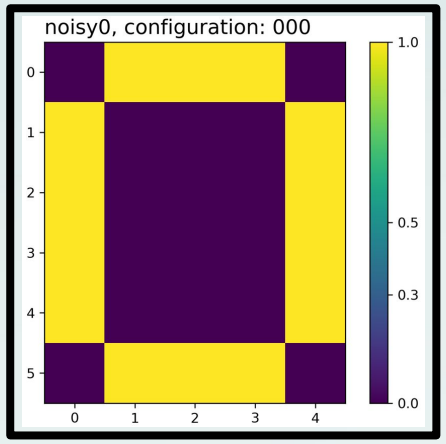
3 | Increasing patterns

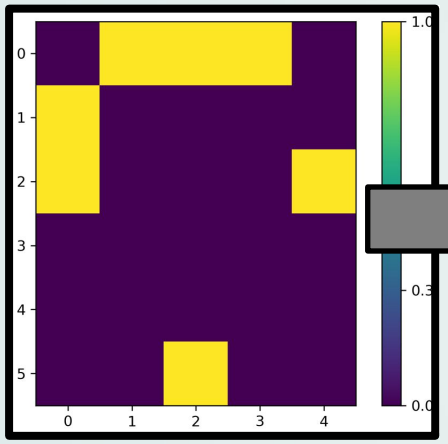
extra task

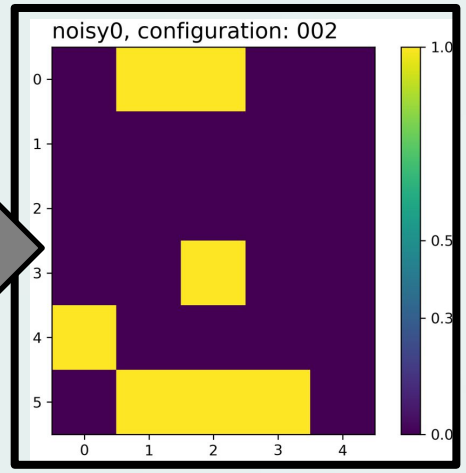
Updating the network

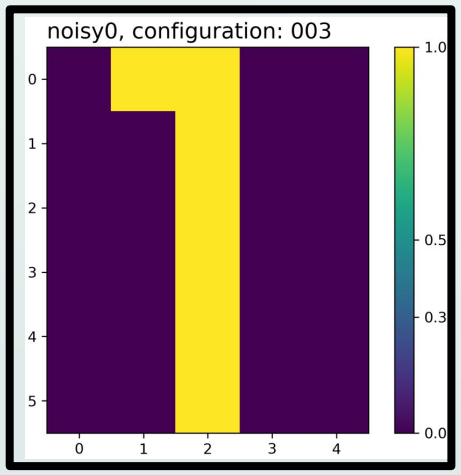
```
def test_network_asynchronously(pattern, w_matrix, steps=10):
    """
    :param pattern: pattern, which we will use as starting configuration
        of network.
    :param w_matrix: pattern, which we get after training.
    :param steps: number of updates, which we want make.
    :return: ending configuration and list contain energy of whole network after
        updating the network.
    """
    n_elements = pattern.size
    E_network_list = []
    for step in range(0, steps):
        # do one-step operation of hopfield network
        E_list = []
        # ordering of choosing spin, which are picked up randomly
        order_index = random.sample(range(n_elements), n_elements)

        for i in range(0, n_elements):
            index = order_index[i]
            x_i = pattern[0, index]
            # flipped spin
            if x_i == -1:
                x_i_flipped = 1
            else:
                x_i_flipped = -1
            pattern_flipped = pattern.copy()
            pattern_flipped[0, index] = x_i_flipped
            # calculate: sum over j of `w_ij * x_j`
            h_i = w_matrix[index].dot(pattern.T)
            h_i_flipped = w_matrix[index].dot(pattern_flipped.T)
            # compute energy
            E_i = float(- 0.5 * x_i * h_i)
            E_i_flipped = float(- 0.5 * x_i_flipped * h_i_flipped)
            if E_i_flipped < E_i:
                # we flip spin
                E_list.append(E_i_flipped)
                pattern = pattern_flipped
            else:
                E_list.append(E_i)
        # update test matrix
        pattern = np.sign(w_matrix.dot(pattern.T).T)
        E_network = sum(E_list)
        E_network_list.append(E_network)
```

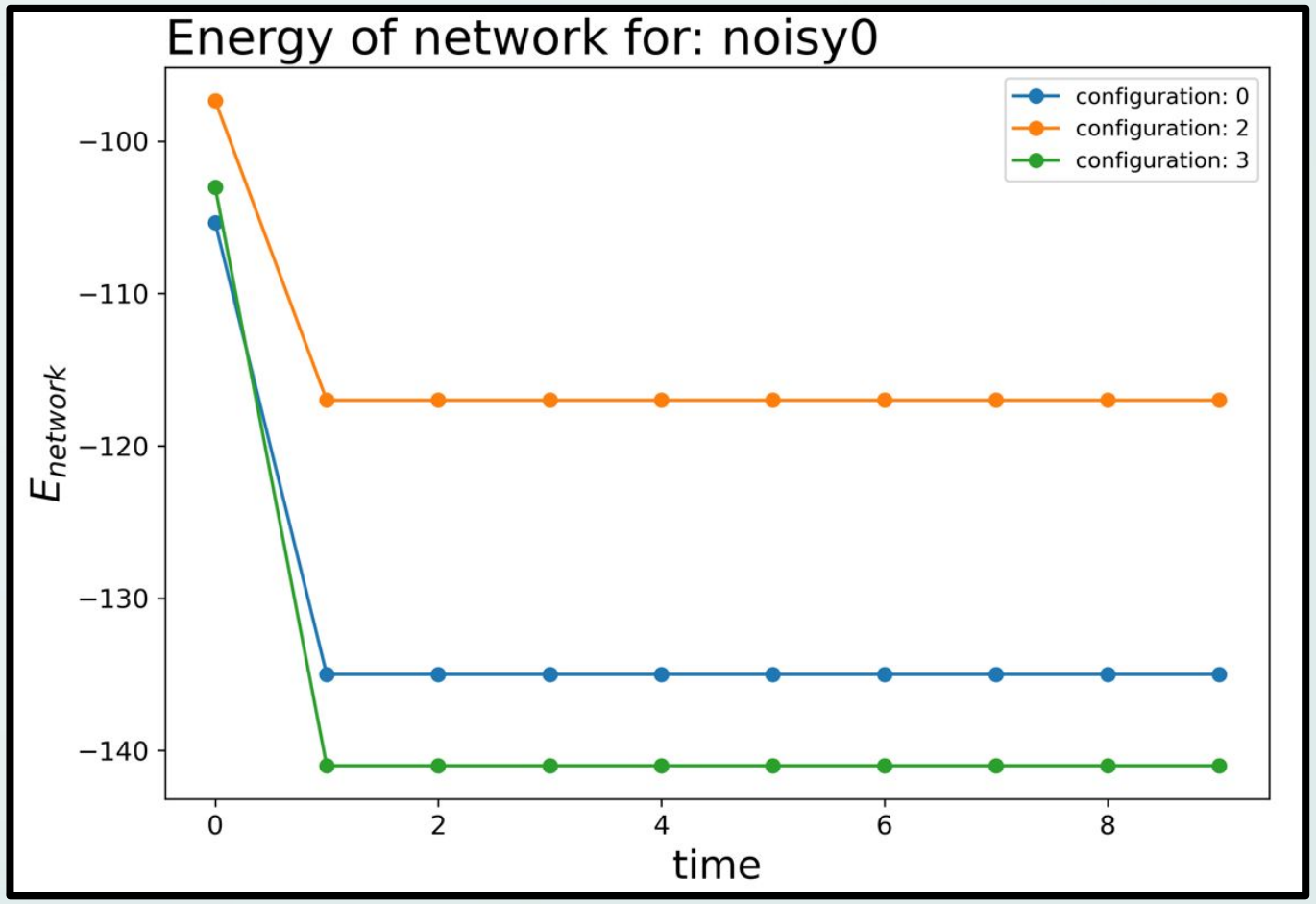
configuration:· noisy-0



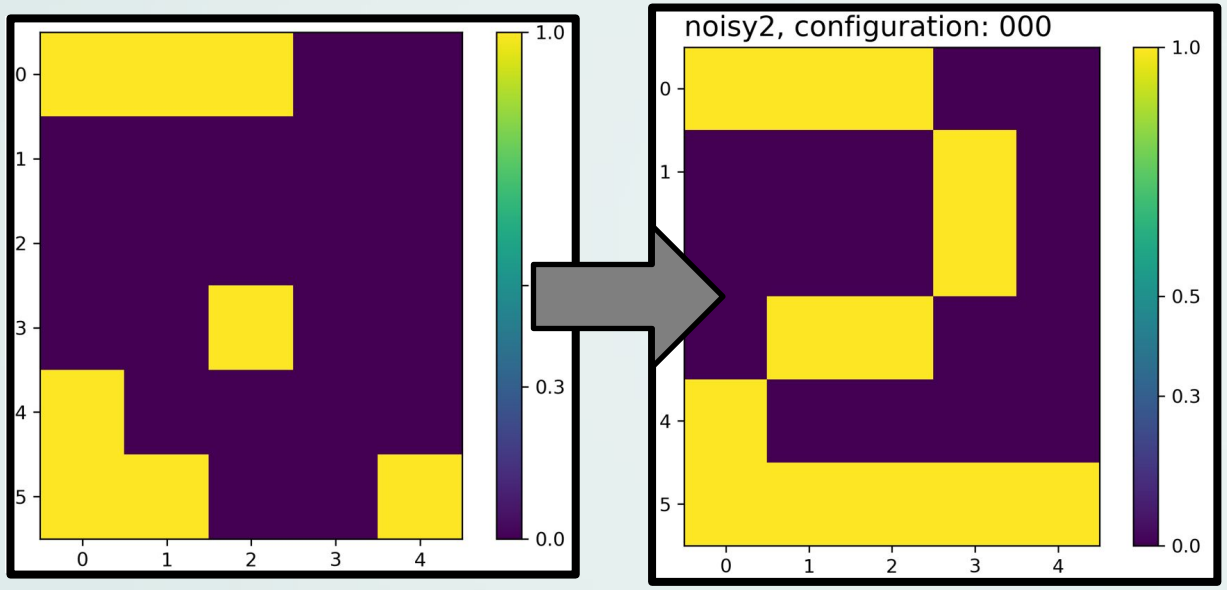


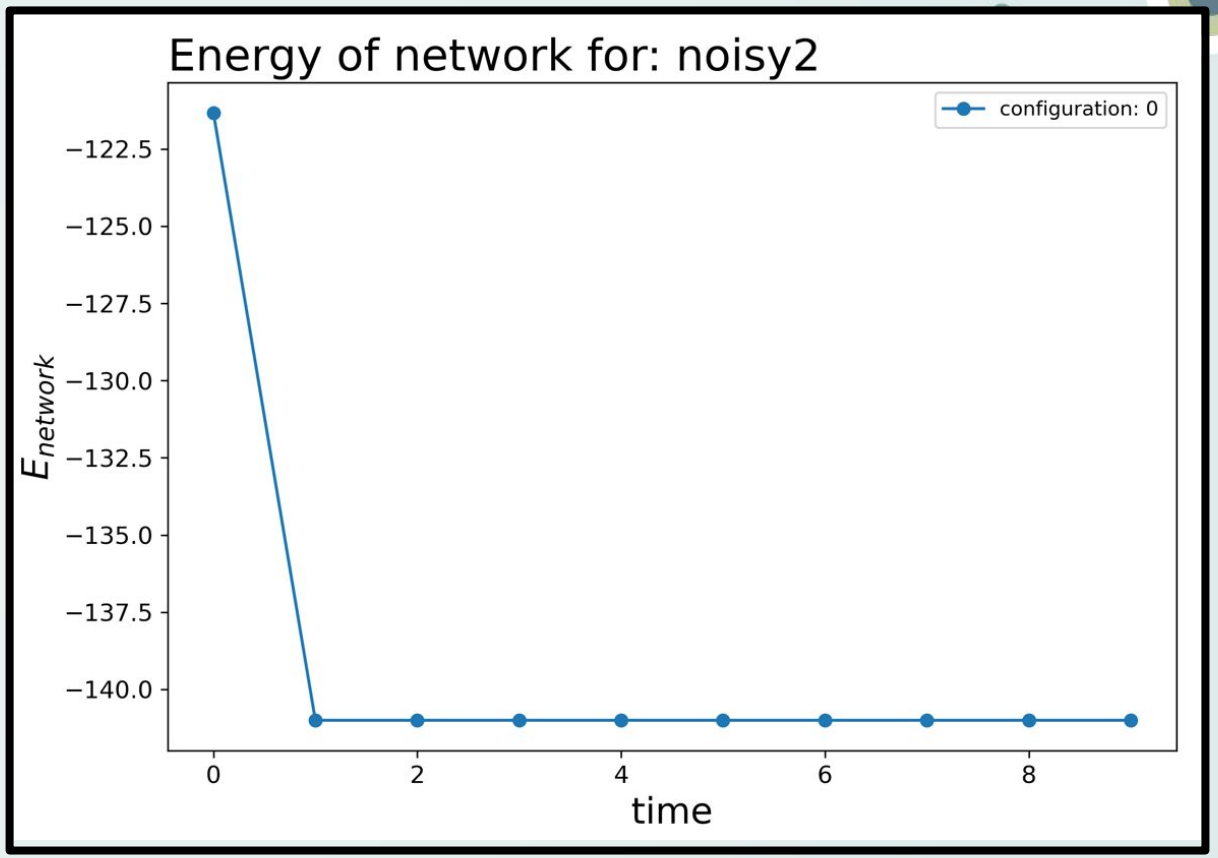




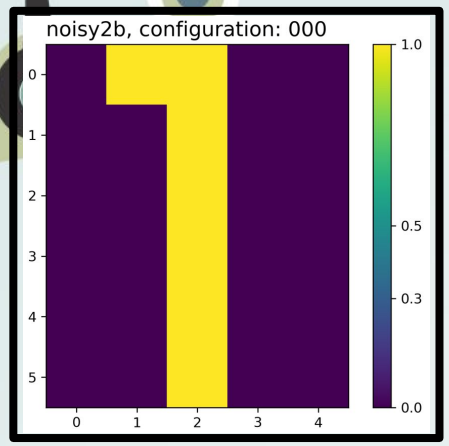


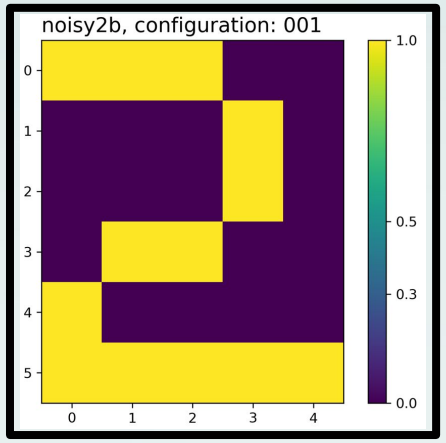
configuration: noisy-2

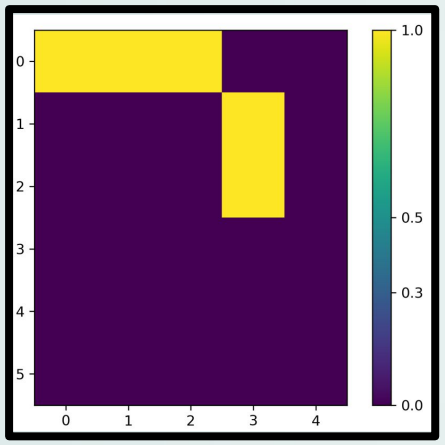


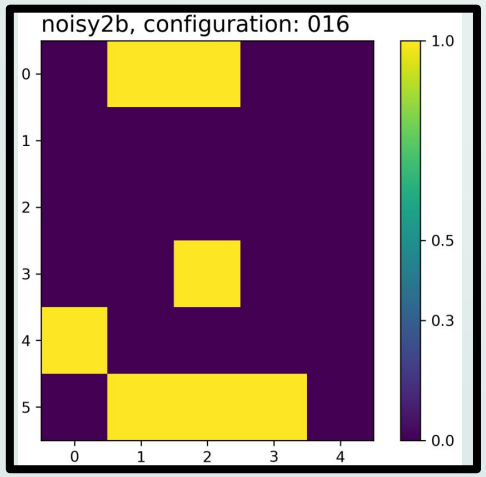


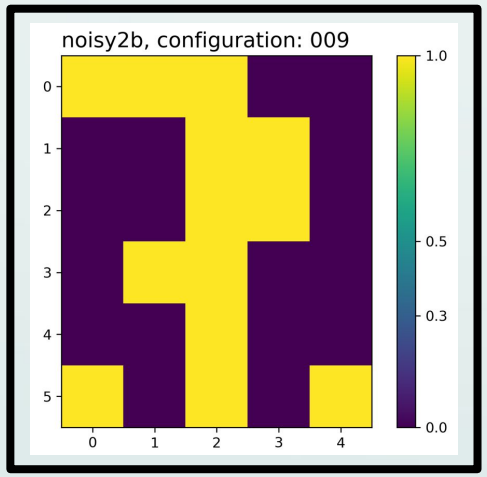
configuration: noisy-2b

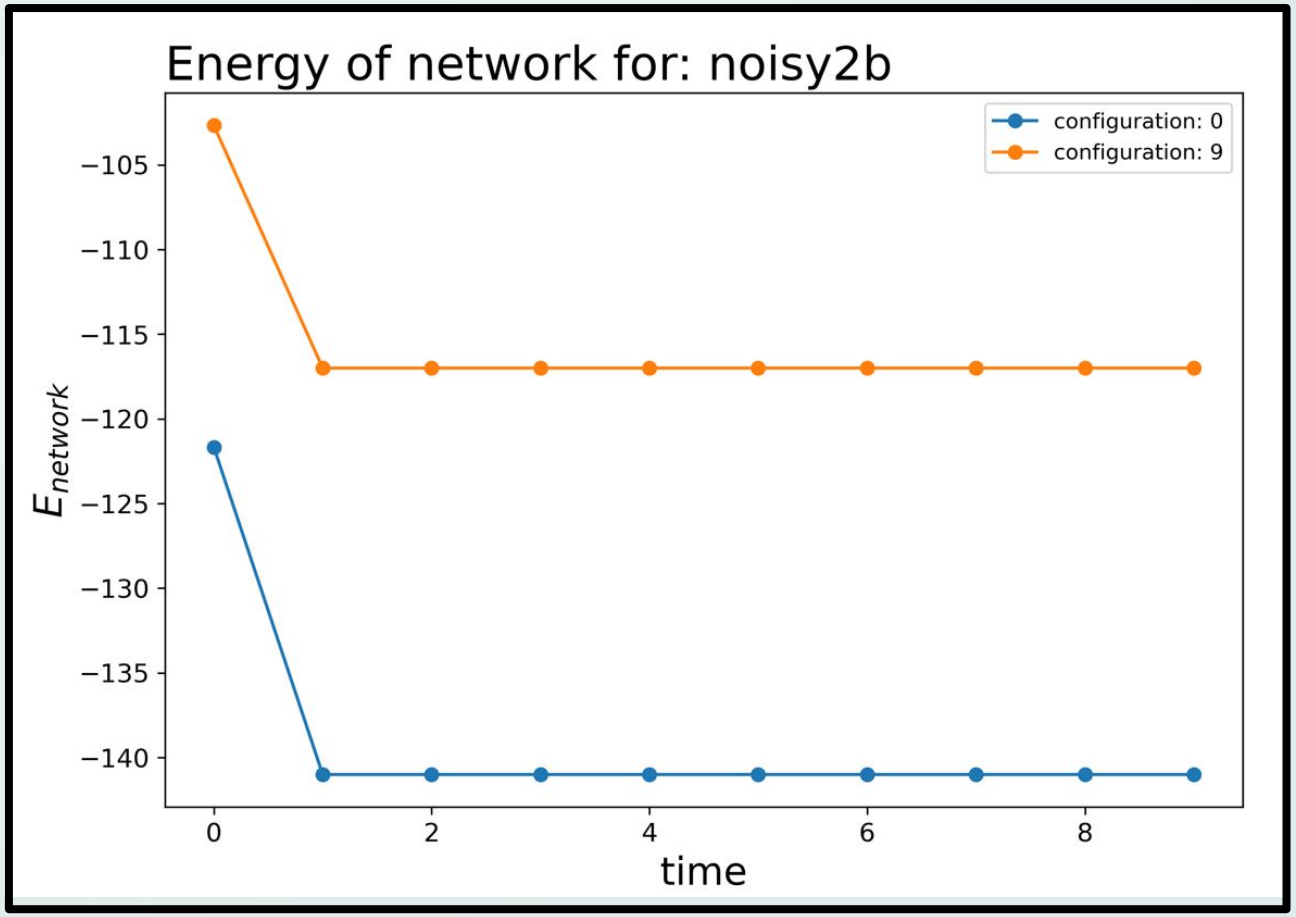












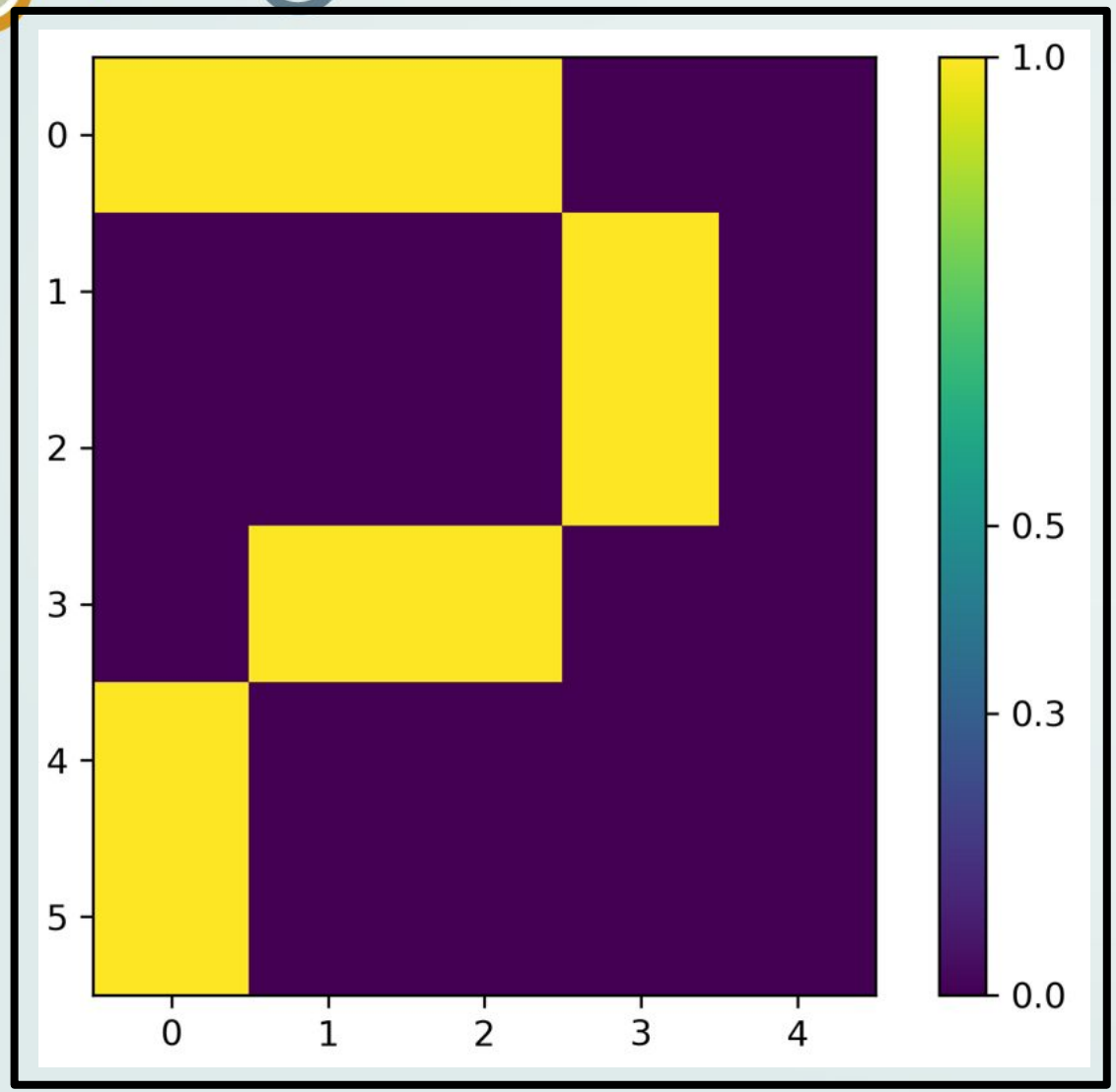
OVERVIEW

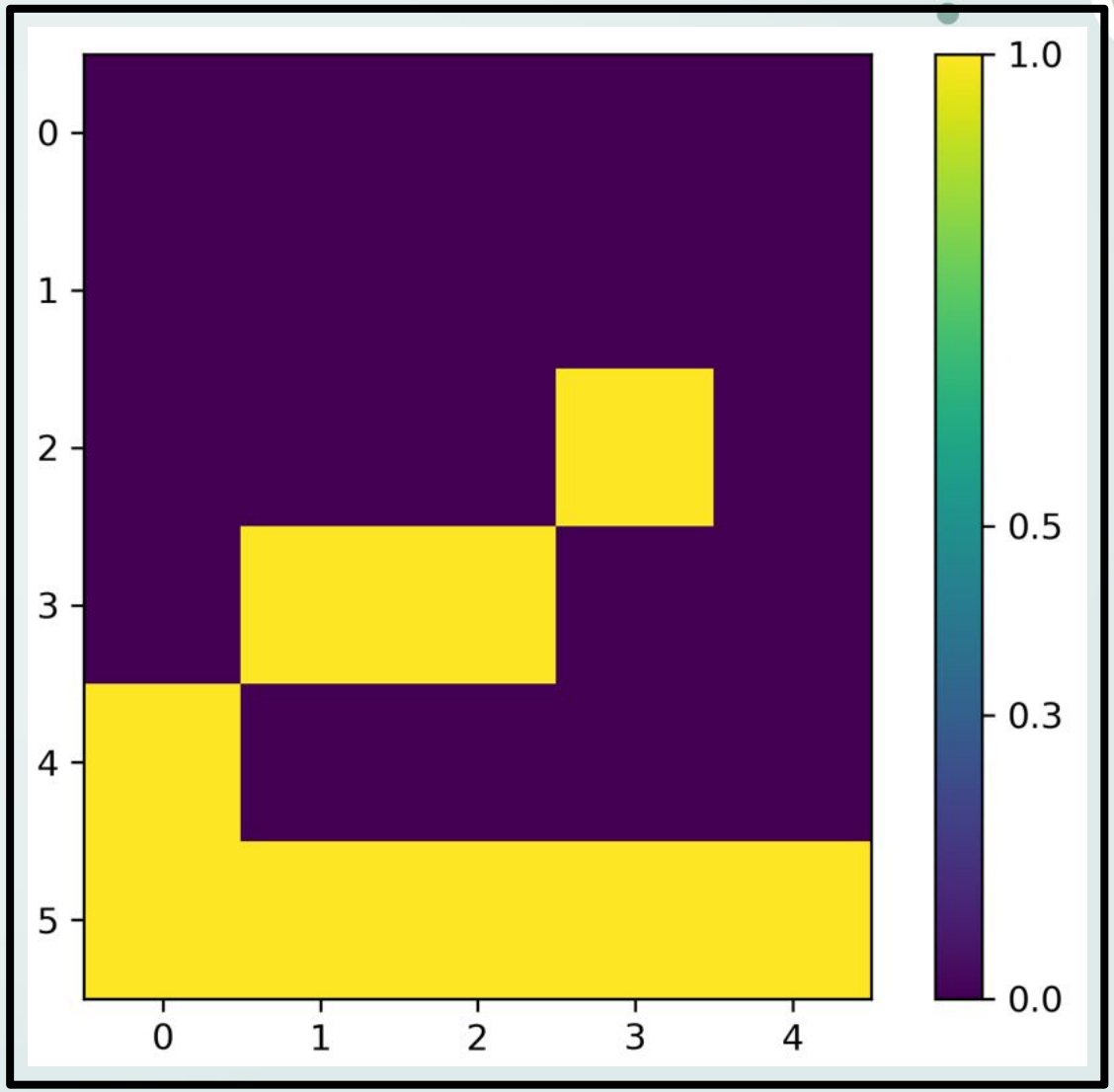
1 | Hopfield network
first task

2 | Asynchronous updates
second task

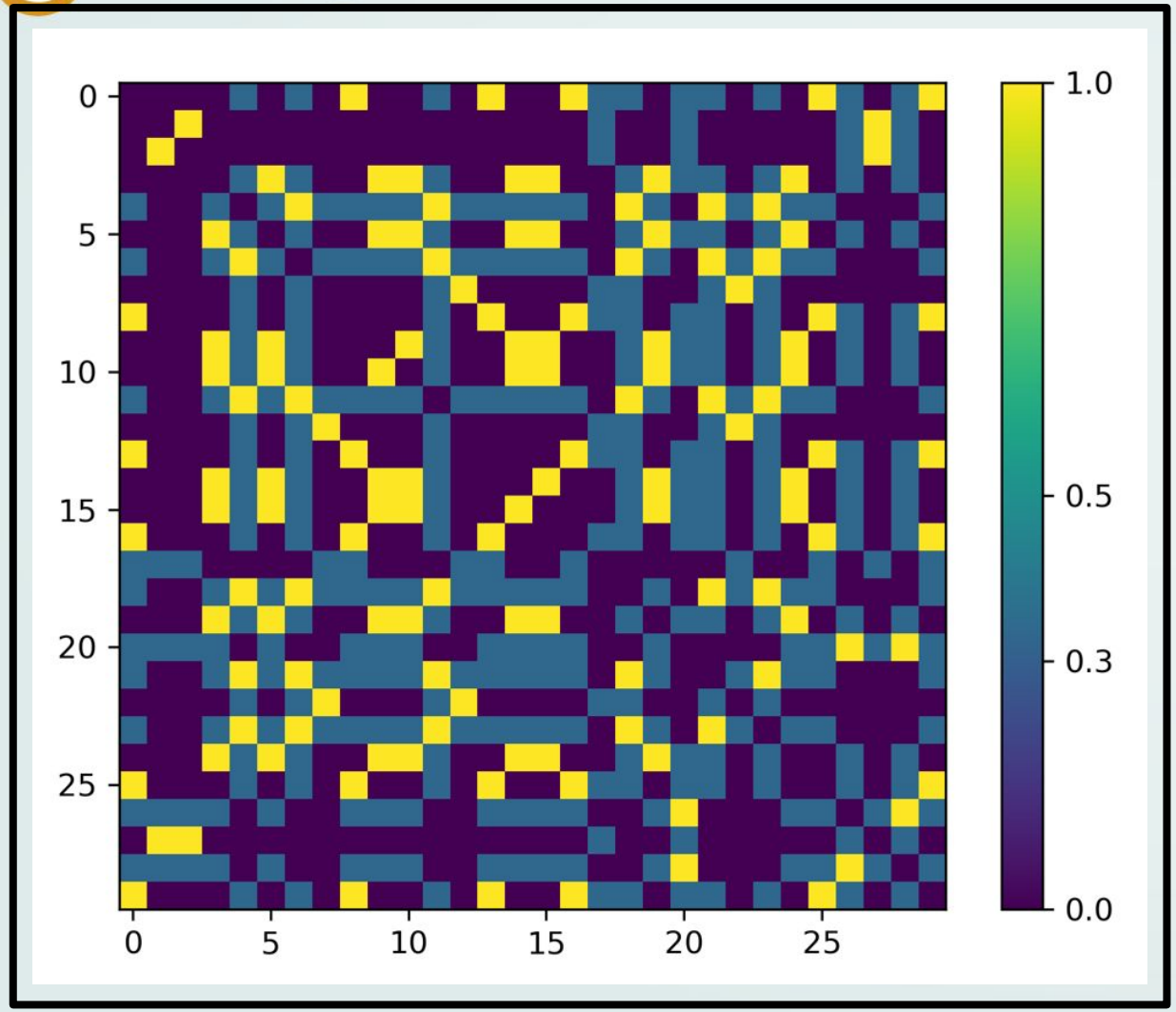
3 | Increasing patterns
extra task

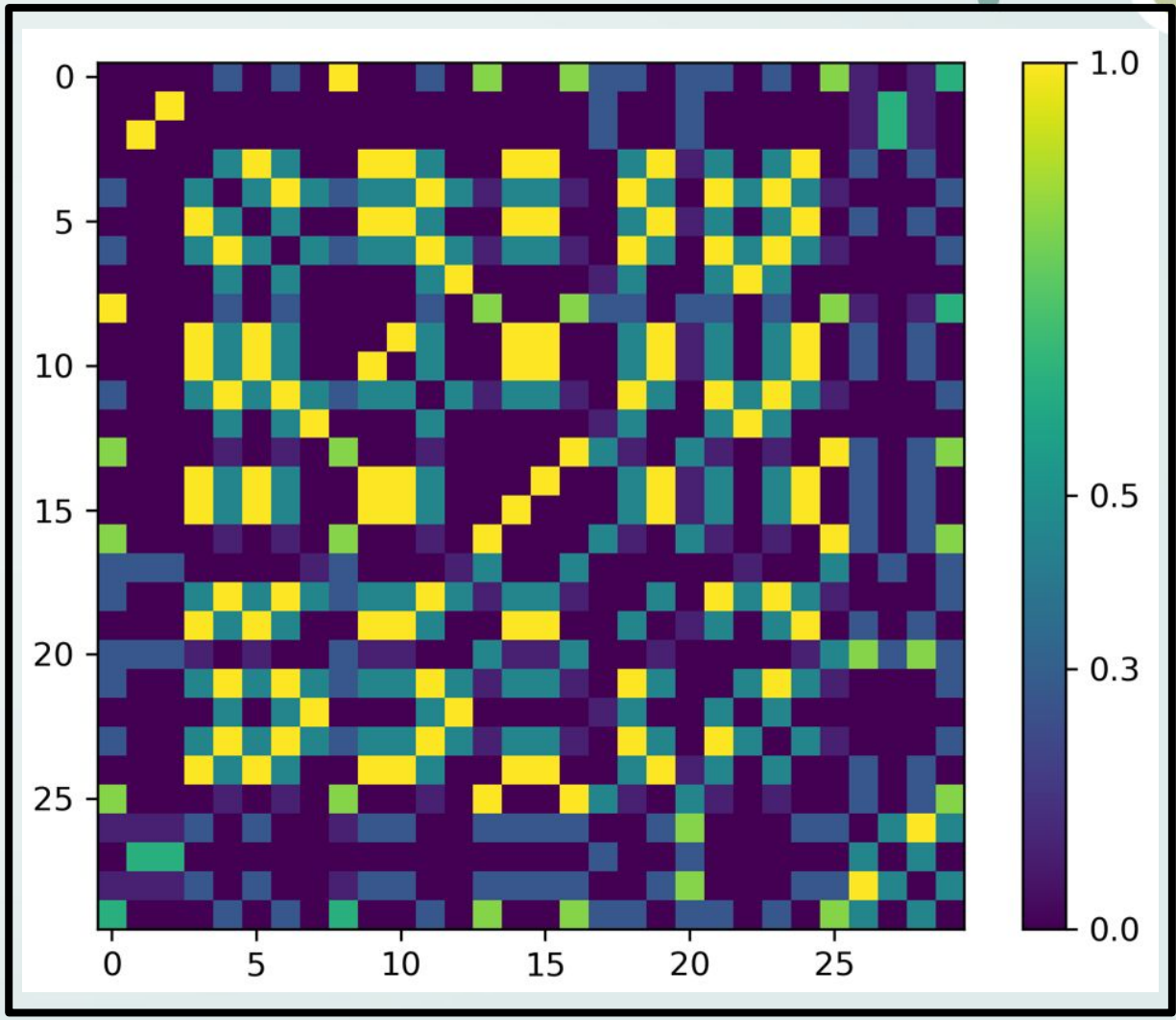
Added patterns



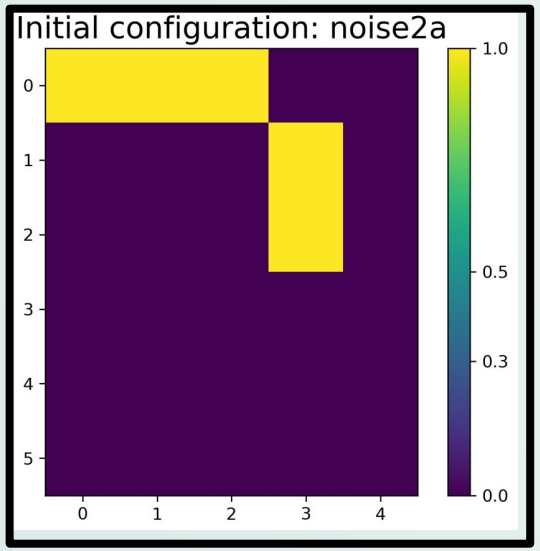


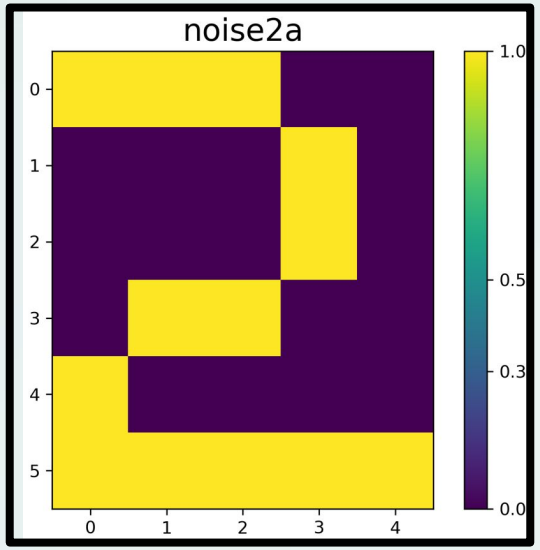
Comparison w matrix

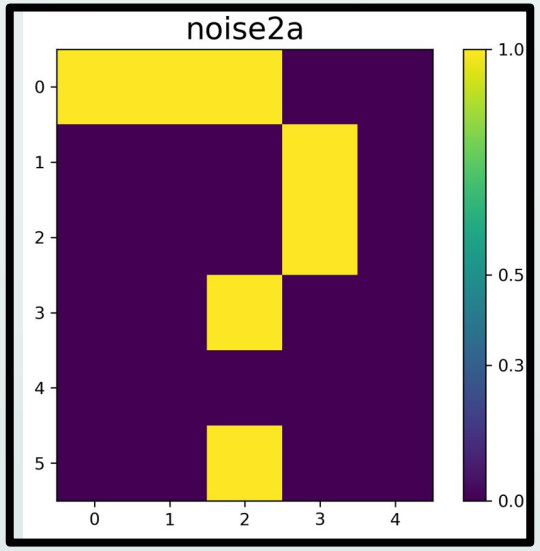


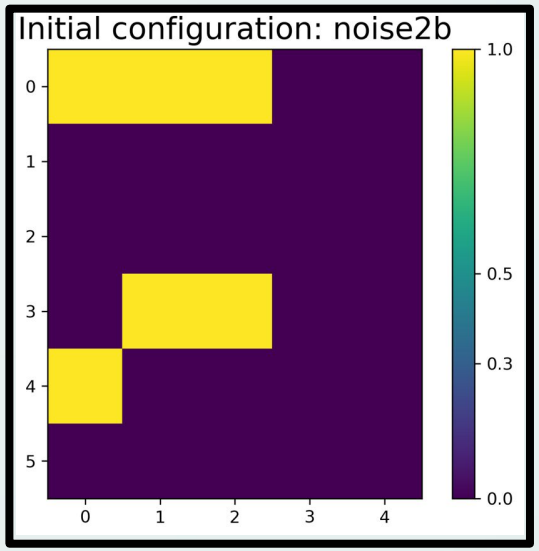


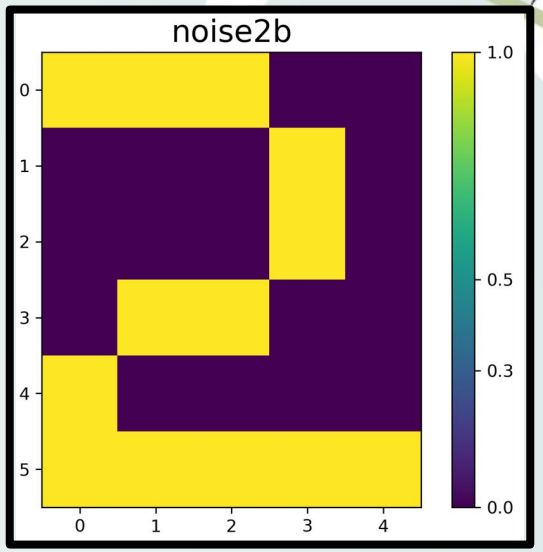
Comparison part one

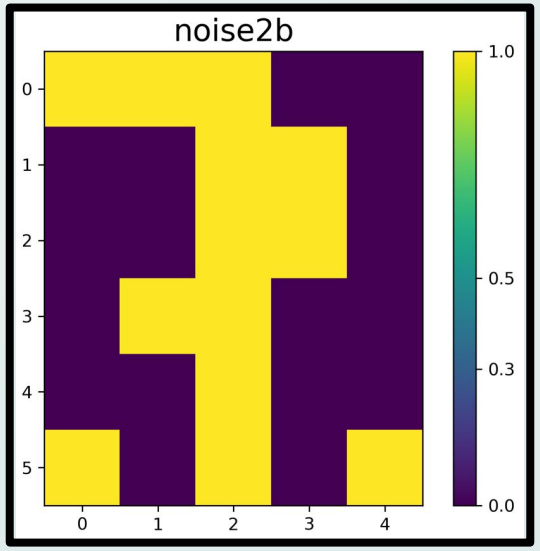




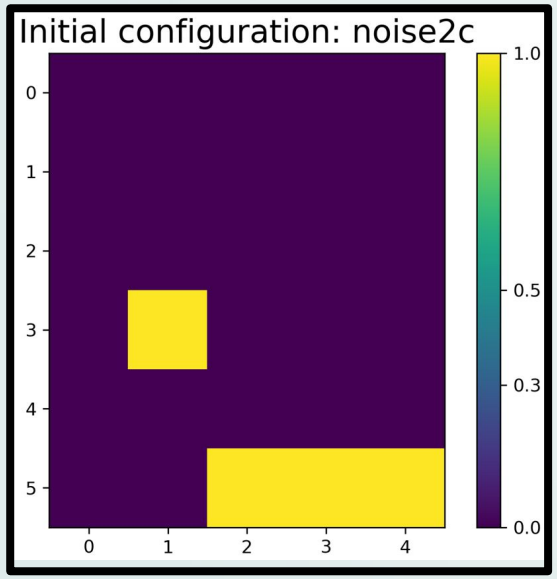


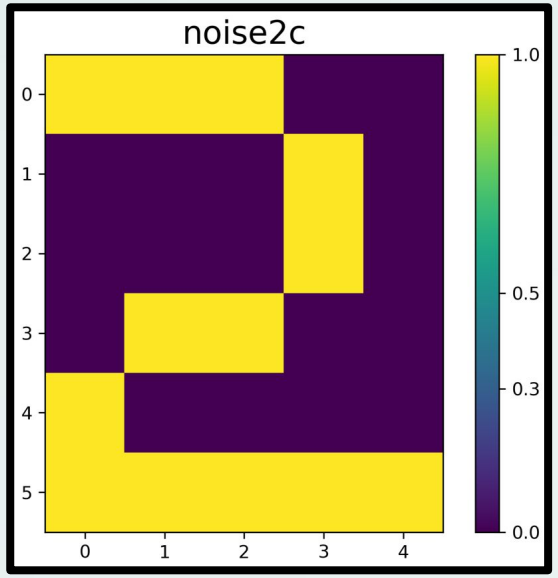


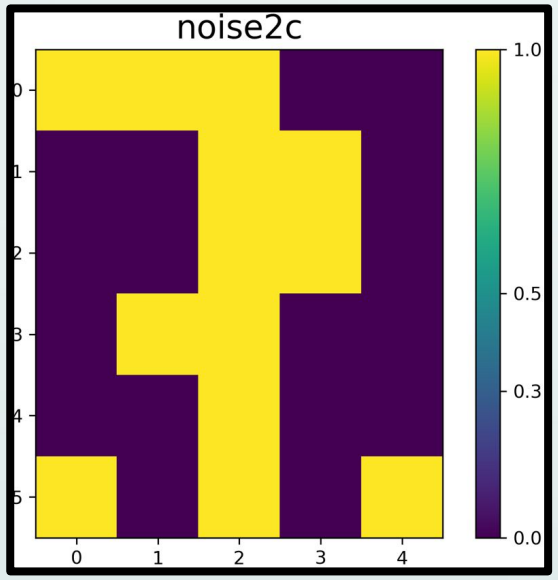


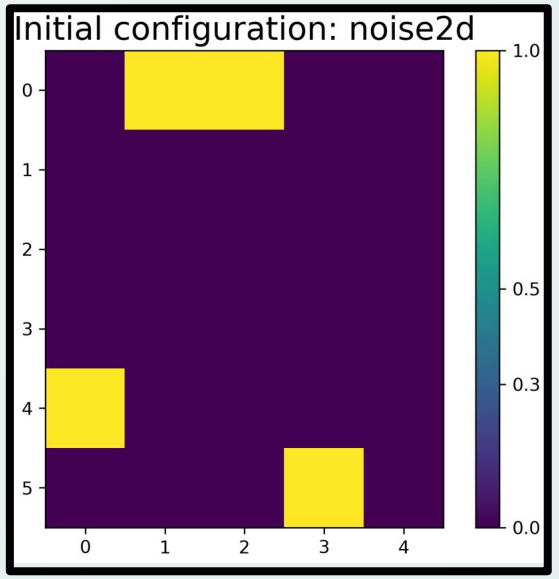


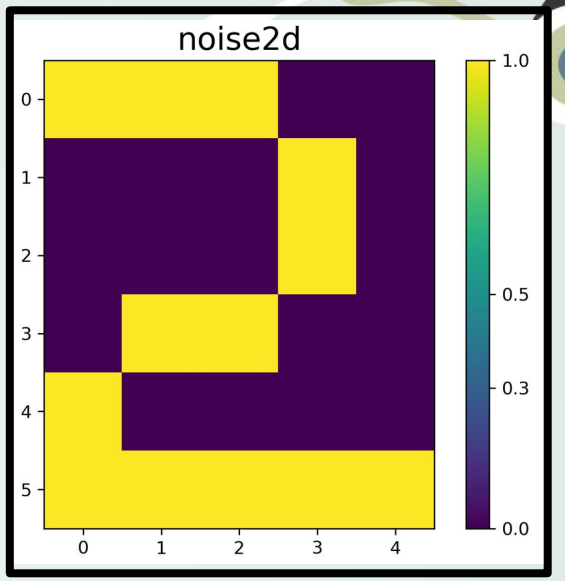
Comparison part two

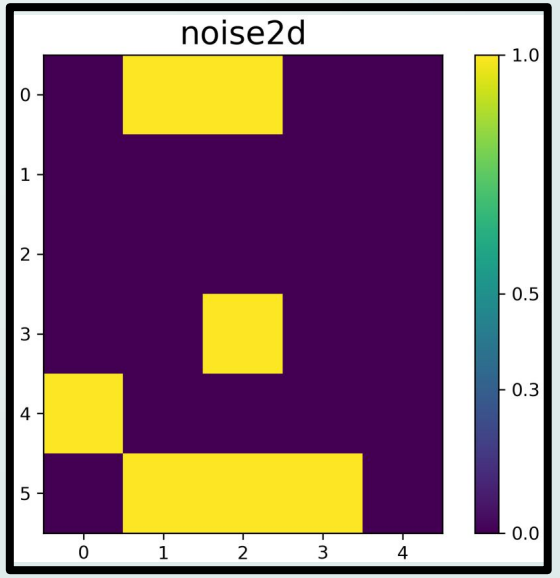




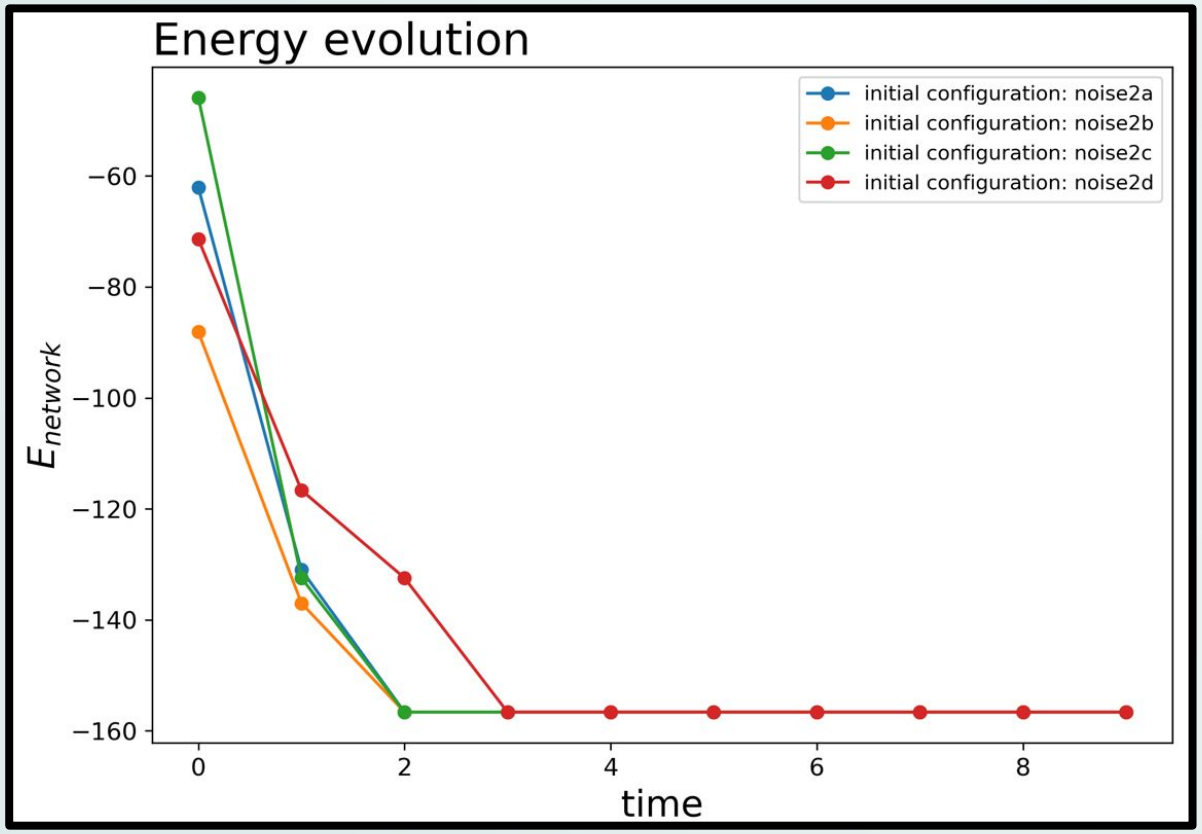


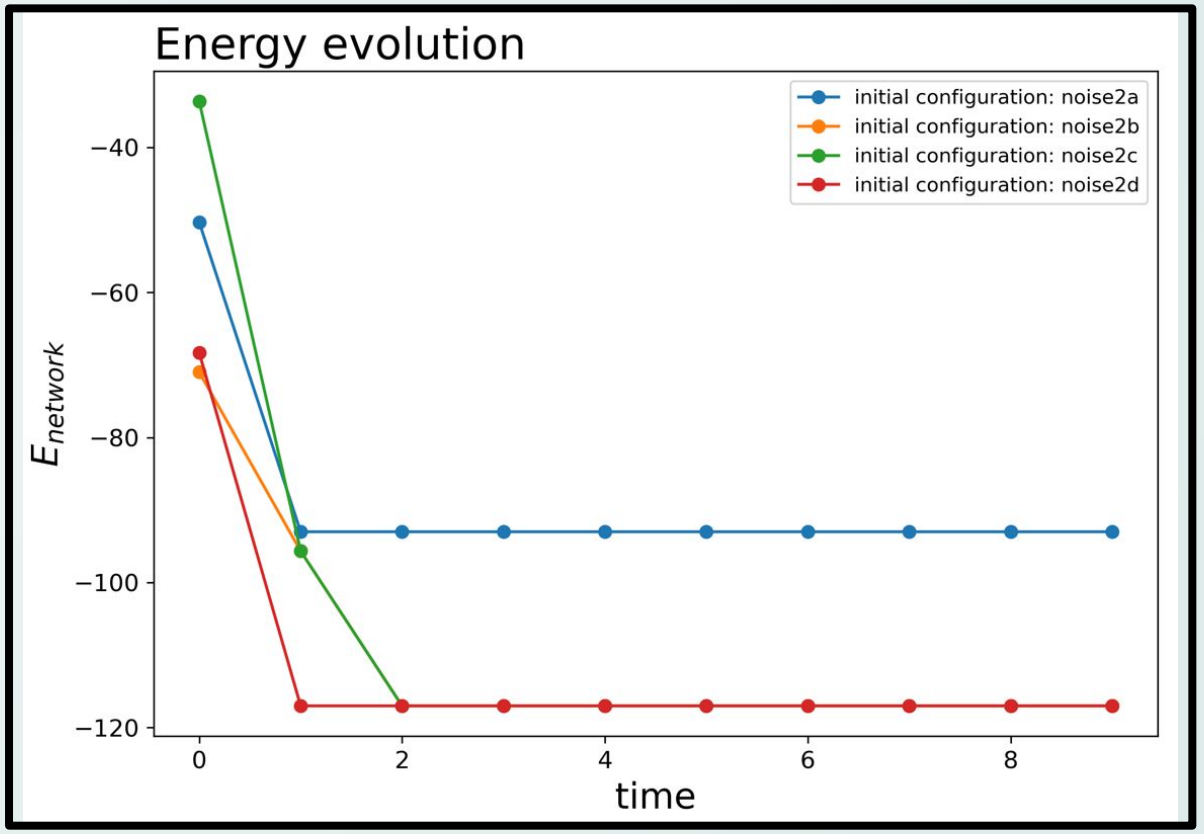






Energy evolution





Thank you!