Artificial Neural Networks

Prof. Dr. Sen Cheng

Jan 13, 2020

Problem Set 13: Hopfield Network

Tutors: Mohammadreza Mohagheghi Nejad (m.mohagheghi@ini.rub.de), Eloy Parra Barrero (eloy.parrabarrero@rub.de)

1. **Asynchrounous updates in Hopfield network.** We stored one input pattern in a Hopfield network, which yielded the following weight matrix:

$$w = \begin{bmatrix} 0 & -1 & -1 & +1 \\ -1 & 0 & +1 & -1 \\ -1 & +1 & 0 & -1 \\ +1 & -1 & -1 & 0 \end{bmatrix}$$

   Based on this, answer the questions below.

   (a) How many elements did the pattern have?

   (b) Starting with any valid pattern you like, use algorithm 13.1 from the lecture notes using pen and paper to find the pattern the network converges to. Is the pattern you have found necessarily the one the network was trained on?

   (c) Now write a Python script that automatizes this process and compare your results.

   (d) Modify your code so as to calculate and store the energy of the network for each step in (c). How does it vary across multiple repetitions of your script?

2. **Pattern storage and retrieval in Hopfield network.** There are seven images stored in `images.npy`. Follow the tasks below to complete this exercise.

   (a) Load the data. You will observe that the shape of the data is 1150×7. The first dimension corresponds to the number of pixels in each image (46×25 pixels stacked column-wise into vectors). The second dimension corresponds to the number of images. The content of each pixel is either -1 or 1, where -1 corresponds to black and 1 to white.

   (b) Reshape the image vectors according to the information provided in 2a and visualize the images.

   (c) Store these images in a Hopfield network using Eq. 2 from the lecture notes.

   (d) Add noise to the images by randomly choosing a subset of pixels and flipping their values from -1/1 to 1/-1. Generate noisy images by setting the subset size to 200 and provide them as input to the network. Run the algorithm for 100 iterations to retrieve the stored images. Visualize this process by showing the original, noisy and retrieved images side by side.

   (e) For each image compute the correlation coefficient (`numpy.corrcoef`) of the original against the retrieved images as a measure of retrieval quality. The correlation coefficient (cc) outputs a number between -1 and 1, $cc \in [-1, 1]$. Interpret what these values mean and which value(s) correspond(s) to lossless retrieval.

   (f) During retrieval, even with 0 noise, one image is confused with another one. Compute the correlation coefficient across all pairs of images. Do you find any relationship between the computed correlation coefficients and the incorrectly retrieved image?

(g) Vary systematically the number of flipped pixels from 0 to 1150 with a step size of 25 and record the correlations between original and retrieved images. Plot these correlations for all images and noise levels, e.g., using `pyplot.matshow`. Describe the effect of noise in the retrieval process.

3. **Capacity of Hopfield network.** To study the memory capacity of the Hopfield network, generate and store random patterns in a network of 100 units. Then, initialize the network with each of the stored patterns and let it evolve for 100 iterations. After this, count the number of pixels which have deviated from the original value (`numpy.count_nonzero` might be of help here). Repeat this process for different numbers of patterns and plot the error rate as a function of the network load (= number of stored patterns / network size). Focus on network loads between 0 and 0.5. At which point does performance start to degrade steeply?

Solutions

1. **Asynchrounous updates in Hopfield network.**

   (a) The stored pattern had 4 elements. This can be inferred by counting how many rows (or columns) the weight matrix has.

   (b) We should start with an arbitrary pattern x and iterate through the update algorithm until the pattern does not change. We initialize the pattern arbitrarily with $x = [+1,-1,+1,-1]^T$. We start updating from the first element:

$$\begin{bmatrix} \mathbf{0} & \mathbf{-1} & \mathbf{-1} & \mathbf{+1} \\ -1 & 0 & +1 & -1 \\ -1 & +1 & 0 & -1 \\ +1 & -1 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} +1 \\ -1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} \mathbf{-1} \\ -1 \\ +1 \\ -1 \end{bmatrix} \tag{1}$$

   Now we continue with the second one:

$$\begin{bmatrix} 0 & -1 & -1 & +1 \\ \mathbf{-1} & \mathbf{0} & \mathbf{+1} & \mathbf{-1} \\ -1 & +1 & 0 & -1 \\ +1 & -1 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} -1 \\ -1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ \mathbf{+1} \\ +1 \\ -1 \end{bmatrix} \tag{2}$$

   Since the second element has changed, we should redo the procedure for the first element to see whether it changes:

$$\begin{bmatrix} \mathbf{0} & \mathbf{-1} & \mathbf{-1} & \mathbf{+1} \\ -1 & 0 & +1 & -1 \\ -1 & +1 & 0 & -1 \\ +1 & -1 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} -1 \\ +1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} \mathbf{-1} \\ +1 \\ +1 \\ -1 \end{bmatrix} \tag{3}$$

   Since it did not change, we can continue for the third element:

$$\begin{bmatrix} 0 & -1 & -1 & +1 \\ -1 & 0 & +1 & -1 \\ \mathbf{-1} & \mathbf{+1} & \mathbf{0} & \mathbf{-1} \\ +1 & -1 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} -1 \\ -1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ +1 \\ \mathbf{+1} \\ -1 \end{bmatrix} \tag{4}$$

   And for the last one:

$$\begin{bmatrix} 0 & -1 & -1 & +1 \\ -1 & 0 & +1 & -1 \\ -1 & +1 & 0 & -1 \\ \mathbf{+1} & \mathbf{-1} & \mathbf{-1} & \mathbf{0} \end{bmatrix} \times \begin{bmatrix} -1 \\ -1 \\ +1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ \mathbf{-1} \end{bmatrix} \tag{5}$$

   In this case we have found a stable solution at $[+1,-1,-1,+1]^T$. The pattern the network was trained on could be this or its inverse, $[-1,+1,+1,-1]^T$, since if x is an attractor of the network, so is $-x$.

   (c) Refer to `ex1.py` for the code.

   (d) The energy of the network always decreases or stays constant, and the stored pattern corresponds to the lowest energy level. Depending on the initial guess and the order in which units are updated, the solution is found more or less quickly.

2. **Pattern storage and retrieval in Hopfield network.** Refer to `ex2.py` for the code.

   (b) Visualization of the input images (Fig. 1).

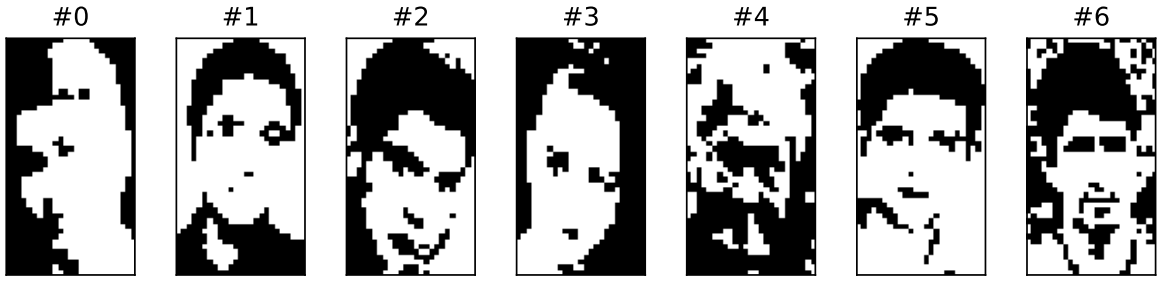

Figure 1: Input images

(d) Input images were stored in a Hopfield network. Noisy versions of the images were given as retrieval cues and the network was let to evolve for 100 iterations of the update algorithm (Fig. 2).

(e) We used correlation coefficients to measure the similarity between the stored and retrieved images (Fig. 2, *r* values in the lower row). If the correlation coefficient is 1, the retrieved image is identical to the stored one and we have a lossless retrieval. A correlation coefficient of 0 would imply that the retrieved image has no relation to the stored one. On the other hand, a correlation coefficient of -1 means that the retrieved image is the negative version of the stored image.

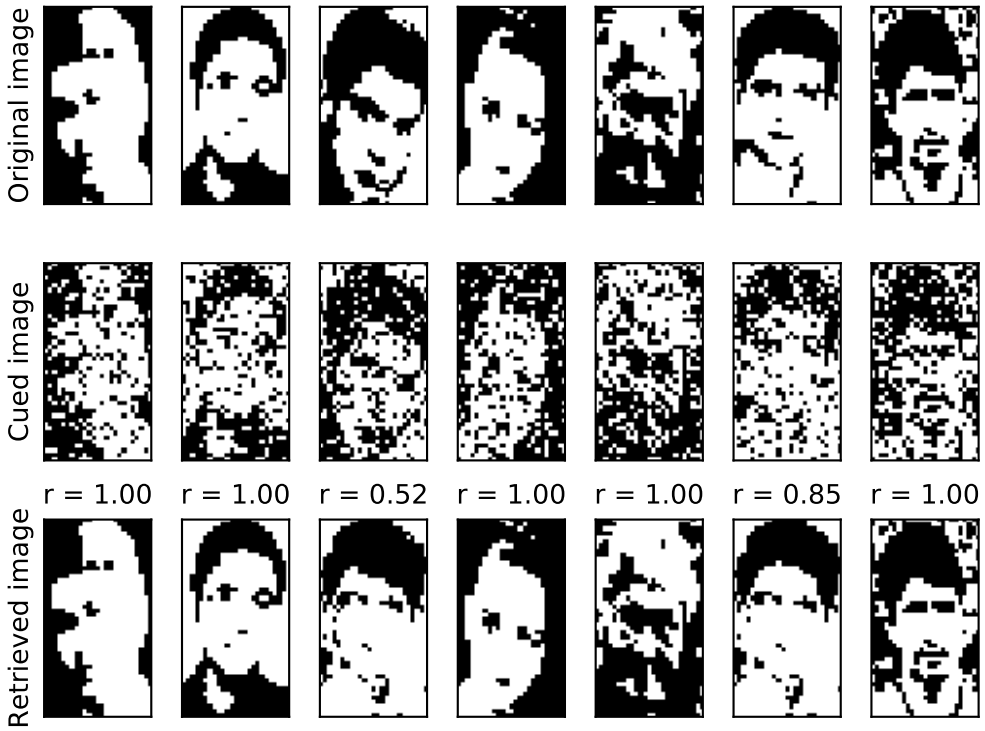

Figure 2: Retrieval process. Original images (upper row), cued noisy images (middle row) and the retrieved ones (lower row).

(f) The network confused image #2 with image #5 and by looking at the correlation coefficient computed for all pairs of images (Fig. 3) we see that these two images are more correlated than the other pairs.

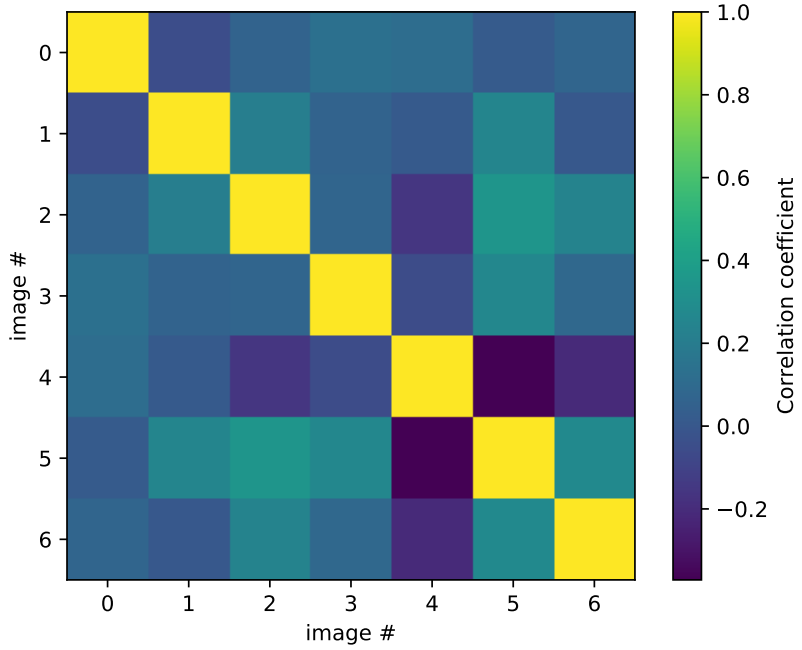

Figure 3: Correlation coefficient computed across all pairs of images.

(g) Our network proves very resistant to noise, being able to restore most noisy patterns, although the performance breaks down when approaching a noise level of 575 (1150/2, equivalent to flipping 50% of all pixels). For noise levels above this, the cued images become more similar to their negative versions, which are also spurious attractors of the network. The network then converges to those attractors leading to negative correlation values.

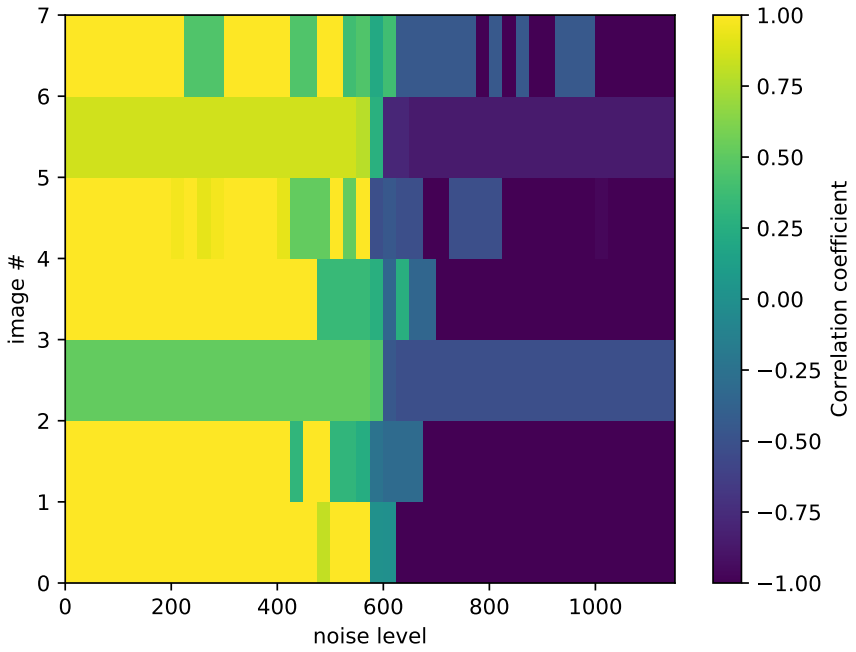

Figure 4: Correlation coefficient in the retrieval process was computed for all individual images with varying noise level.

3. **Capacity of Hopfield network.** Refer to `ex3.py` for the code.

   Under low network load conditions, the stored patterns are indeed stable attractors of the network, and thus the error rate remains close to 0. However, at a critical value of ≈ 0.14, catastrophic interference occurs and the network is unable to retrieve any patterns correctly. Therefore the maximum number of patterns that can be safely stored in a Hopfield network is approximately 0.14 times the network size.

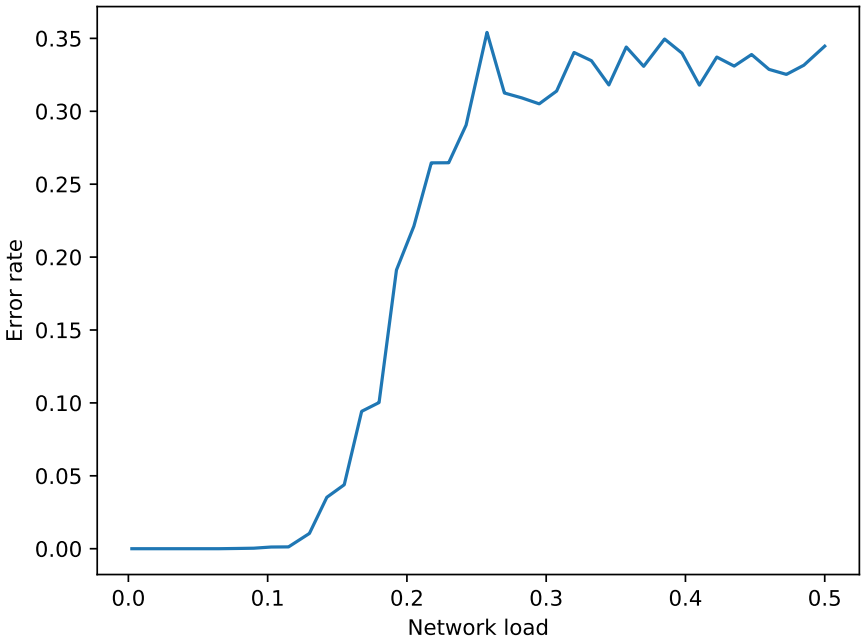

Figure 5: Probability of error in a Hopfield network as a function of the load.