Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 3

Hopfield Networks

Rakin Ali, Steinar Logi and Hassan Alzubeidi

February 21, 2024

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to build a Hopfield network from scratch only using Python and theory from the lab instructions.
- to explore associative memory capabilities of a Hopfield network.

The weights of the Hopfield network were adjusted to the data points, they were not manually created.

2 Methods

All code was created with Python v 3.9 and Jupyter notebook. This assignment was difficult to break into different parts so all instructions were completed three times thereafter we compared our results in order to check if our networks were behaving the way they were suppose to.

3 Results and discussion

This section presents the results provides a small discussion which analyzes the results.

3.1 Convergence and attractors

Given x1d, x2d and x3d where x1d has one bit error while x2d and x3d has two bits errors, we discovered that the network converged towards to the stored patterns. When we made the starting pattern even more dissimilar to the stored one where more than half the points were wrong, it did not converge correctly. A result image has been omitted in order to s

3.2 Sequential update

All three patterns were stable and tested with the network. The network was able to memorise them.

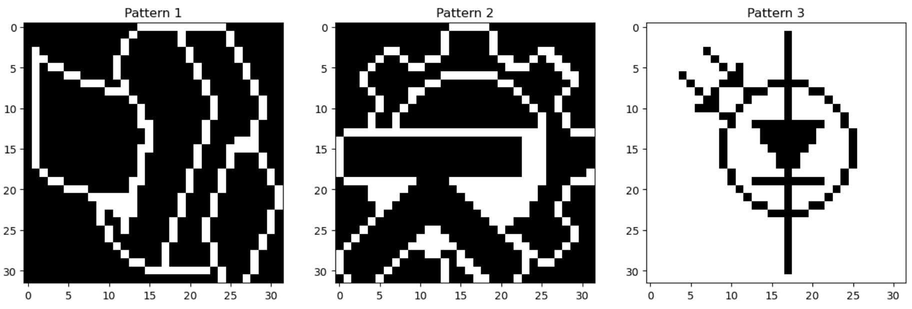


Figure 1: P1, P2 and P3 recreated with Hopfield network

Later when the network tried patterns p10 and p11, interesting results were noted. P10 was handled well while P11 did not. P11 seems to be spurious. See the figure below

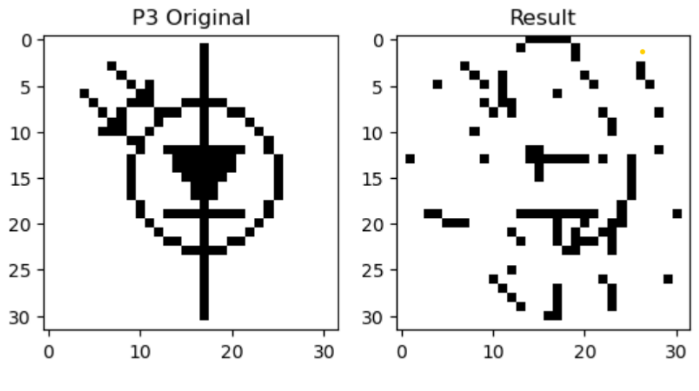


Figure 2: P3 on the left and P11 recall on the right

3.3 Energy

The energy of a Hopfield network at a given with relation to the given activations is defined as the following:

$$E = -\sum_i \sum_j w_{ij} x_i x_j$$

The energy of the network is guaranteed to decrease after each update if the weights between to nodes in the Hopfield network are symmetric. We initialized the weights using the Hebbian learning rule. We used the first three images in the image dataset and let the network memorize them. The energies at the different images are; -490287 for image 1, -465114 for image 2, and -498090.6 for image 3. We used image 11 from the dataset which is a mix of images 2 and 3. The energy of the network when presented with image 11 was around -58197 which is much higher than for the images the network has remembered. We performed recall on this image sequentially and plotted the energy after each iteration. The results from that can be seen in figure 3

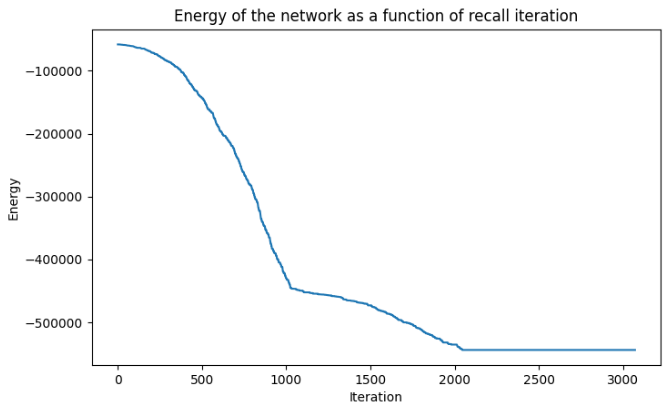
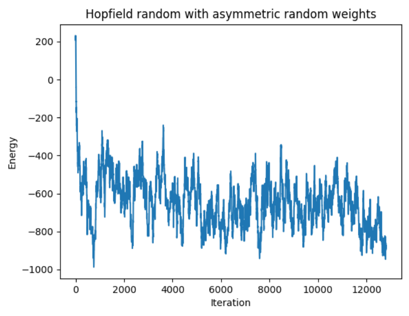


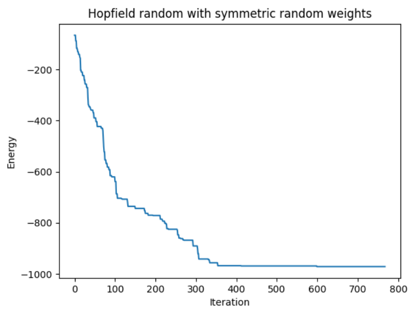
Figure 3: Energy of the network

As can be seen in figure 3 the energy either lowers or stays the same after each iteration. This was expected since as mentioned in the beginning of this section one should expect the energy of the network to get lower if the weights are symmetric. We also tried initializing the weights randomly in two ways. One way was initializing each weight using a normal distribution with mean 0 and variance 1 and the other way we used a normal distribution with mean 0 and variance 1 to initialize the weights but we ensured that the weight matrix was symmetrical. Figure 4 shows the evolution of the energy of the network for each iteration when recalling image 11 from the data set.

As the figure shows the energy of the network with asymmetric weights does not always lower with each iteration, however, the network with the symmetric random weights does lower the energy with each iteration in the recall. It did not recall one of the pictures in the data set, since the network has not memorized the patterns, but it did end in some spurious pattern.

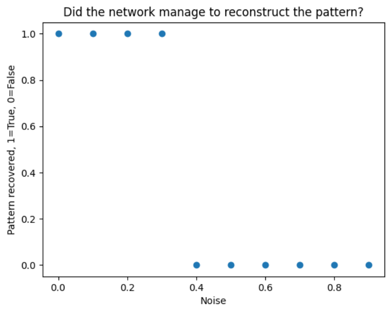


(a) Asymmetric weights



(b) Symmetric weights

Figure 4: The energy of networks with random weights



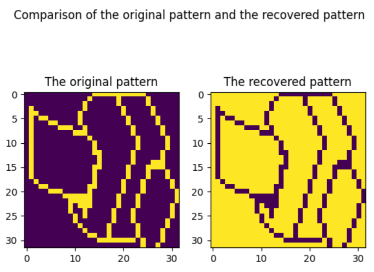


Figure 5: Recovering distorted images

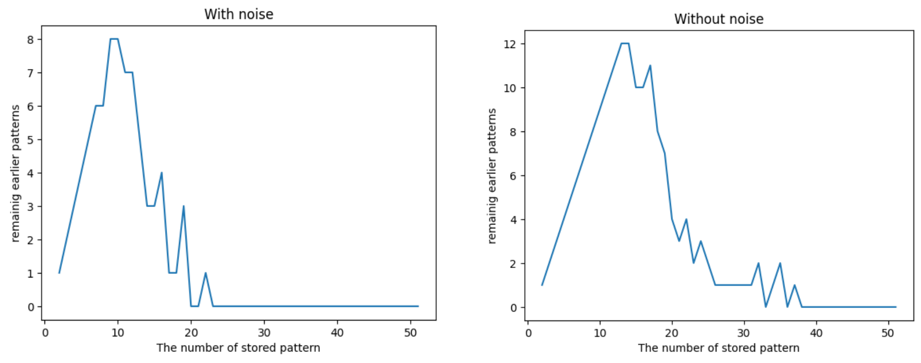
3.4 Distortion Resistance

We trained a Hopfield network using images 1, 2 and 3 from the data set. We then distorted the patterns by flipping a random part of the bits. We checked with different levels of distortion, the values we checked were on the interval 0% to 100% distortion with a step size of 10%. The plot in figure **??** shows whether we managed to recover the figure using the model as a function of the distortion. The model performed pretty well. It managed to recover the image with up to 40% distortion. After that it did not manage to recover the original pattern. However we noticed that when the percentage of distorted pixels was greater than 50% it sometimes converged to a pattern that was the original pattern with flipped bits. That is every 1 was flipped to -1 and vice versa. These results can be seen in figure 5. These results are expected since the Hopfield network is symmetric in this way.

3.5 Capacity

We compared how many patterns the network can store in two cases. The comparison between pictures and random patterns. Our results indicate that the network can only store three pictures but store more than three random patterns. We think it's due to the similarity between the random patterns.

Furthermore, we created a network that consists of 100 nodes to compare how many of the earlier patterns the network can remember once we add more patterns to it. Figure 6 shows the comparison with and without noise.



(a) Stored patterns with 10% noise (b) Stored patterns without noise

Figure 6: Number of patterns the network was able to store

3.6 Sparse Patterns

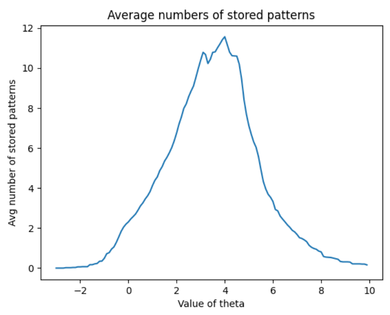
In this section of the assignment we investigated sparse patterns and how the learning rule could be adjusted using average activity $\rho = \frac{1}{NP}\sum_\mu \sum_i x_i^\mu$. The learning rule with this average activity is:

$$w_{ij} = \sum_{\mu=1}^{P} (x_i^\mu - \rho)(x_j^\mu - \rho)$$

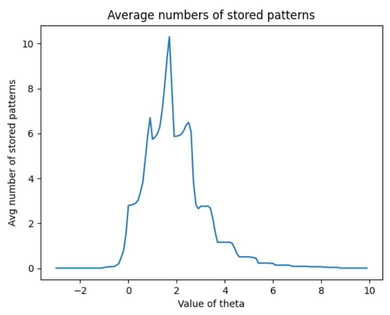
And when we updated the activations during recall we used the following equation that relies on a bias term, θ:

$$x_i \leftarrow 0.5 + 0.5 * sign(\sum_j w_{ij}x_j - \theta)$$

We generated patterns with 10% activity and investigated how many patterns could be stored for different values of the bias term, θ. We checked for values of theta between -3 and 10 and used a step size of 0.1, based on our empirical findings. We made a script that repeated this 100 times and used the average number of patterns stored to get better results. We then plotted the average number of patterns the network was able to store as a function of the theta value. This graph can be seen in figure 7. We also tried this for random patterns with 5% activity. The results for that can be seen in the graph on the right in figure 7



(a) Patterns with 10% activation



(b) Patterns with 5% activation

Figure 7: Number of patterns the network was able to store

The network was able to store up to around 12 patterns when the patterns contained 10% activity with a value of θ around 3.9. When the activations of the patterns were 5% the network was able to store up to around 10 patterns using a value of θ around 2.

4 Final remarks

In this lab we learned that

- Hopfield networks are good at solving optimization problem. By encoding a problem as an energy landscape where each state represents a possible solution and the energy corresponds to the objective function, the network can converge to low-energy states which correspond to optimal solutions. This was learned from section 3.3
- Hopfield networks are used to store patterns and retrieve them based on partial or noisy inputs. They are suitable for tasks such as auto-association and content-addressable memory. This was learned from 3.4 and 3.1
- We learned about Catastrophic forget phenomenon in Hopfield networks. The network, according to the theory covered in the lecture, can recall roughly 138 patterns from 1000 nodes and once that threshold is exceeded, the networks forgets almost everything entirely in an abrupt manner.