Grace period: The project can be submitted until 11:59 PM of the day of the deadline with 20% penalty. Any change in the project after the deadline is considered late submission. One second late is late. The project is graded based on when it was submitted, not when it was finished. Homework late days cannot be used for the project. Note that this project is in lieu of a final exam and is completed over multiple days, hence OSAS accommodations do not apply to the time allowed for it.

The goal of this project is to measure the storage capacity and robustness to noise of Hopfield-type Associative Memories.[1] You can use Jupyter Notebooks and run your code and submit the notebook with the results.

1. **The Dynamics of The Hopfield Model**

   The Hopfield Neural Network is a Recurrent Neural Network that runs through time, and is governed by the following equations:

$$\mathbf{a}(0) = \mathbf{x}$$
$$\mathbf{n}(t+1) = \mathbf{W}\mathbf{a}(t) + \mathbf{b}$$
$$\mathbf{a}(t) = \mathbf{f}(\mathbf{n}(t))$$

   where t is the time step, $\mathbf{x}$ is an input pattern (here a bipolar vector of +1's and -1's that can be viewed as a black and white image) to be recognized, $\mathbf{n}(t)$ is the net signal received by the neurons and is a linear combination of the elements of the current output $\mathbf{a}(t)$. $f(\cdot)$ is the activation function of the network. Here, we use the bipolar step function:

$$\mathrm{sgn}(n) = \begin{cases} +1 & n \geq 0 \\ -1 & n < 0 \end{cases}$$

   The output of the network is initialized with the pattern to be recognized. The network is trained to act as an associative memory, i.e. it memorizes a set of (bipolar) patterns, and it is supposed to recall one of those patterns in its output, if that pattern (or a noisy version of it) is presented to the network as $\mathbf{a}(0)$, i.e. the initial pattern. This means that it is expected that there is a time T at which the output of the network stabilizes, i.e. $\mathbf{a}(t+1) = \mathbf{a}(t), \forall t \geq T$, and we hope that $\mathbf{a}(T)$ is one of the patterns that the network has stored. This unfortunately does not always happen, and sometimes the output of the network starts oscillating.

   Note that from the above equations, one can observe that the dimension of the input pattern is the same as the number of neurons in the network.

2. **Training the Hopfield Model** There multiple training methods for the Hopfield Neural Network, the most famous of which is Hebbian learning. Assume that the patterns to be stored in the Hopfield model are $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^L$. Hebbian training sets the weights and biases of the network as:

[1]John Hopfield won the 2024 Nobel Prize in Physics for inventing this model.

$$\mathbf{W} = \sum_{l=1}^{L} \mathbf{x}^l (\mathbf{x}^l)^T$$
$$\mathbf{b} = \mathbf{0}$$

Plugging in the equations of the Hopfield network:

$$\begin{aligned}\mathbf{n}(t+1) &= \sum_{l=1}^{L} \mathbf{x}^l (\mathbf{x}^l)^T \mathbf{a}(t)\\ &= \sum_{l=1}^{L} \langle \mathbf{x}^l, \mathbf{a}(t)\rangle \mathbf{x}^l\end{aligned}$$

which means the next net signal is a linear combination of the stored patterns, with coefficients of each stored pattern being the inner-product (similarity) of the output vector with that pattern. Because the output is initialized with a noisy version of a stored pattern, the hope is that it has a high correlation with the stored pattern, so the stored pattern receives a large coefficient, and its chance of appearance in the output increases, and eventually the output of the network becomes that stored pattern and stabilizes.

Another way of training the Hopfield model is using the so-called pseudo-inverse rule. Assume that we collect the patterns we want to store as columns of a matrix $\mathbf{X}$, i.e.:

$$\mathbf{X} = [\mathbf{x}^1\ \mathbf{x}^2 \cdots \mathbf{x}^L]$$

The pseudo-inverse rule uses:

$$\mathbf{W} = \mathbf{X}\mathbf{X}^+$$
$$\mathbf{b} = 0$$

where $\mathbf{X}^+$ is called the Moore-Penrose inverse or pseudo-inverse[2] of $\mathbf{X}$ and has the following properties:

$$\begin{aligned}\mathbf{X}\mathbf{X}^+\mathbf{X} &= \mathbf{X}\\ \mathbf{X}^+\mathbf{X}\mathbf{X}^+ &= \mathbf{X}^+\\ \left(\mathbf{X}\mathbf{X}^+\right)^T &= \mathbf{X}\mathbf{X}^+\\ \left(\mathbf{X}^+\mathbf{X}\right)^T &= \mathbf{X}^+\mathbf{X}\end{aligned}$$

You only need to know how to compute the pseudo-inverse to finish this project.[3] However, you can try to answer this question knowing the above facts about the Pseudo-inverse: what does this rule really do?

[2]`https://en.wikipedia.org/wiki/MoorePenrose_inverse`
[3]`https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html`

Unfortunately, when we attempt to store too many patterns in the network, or when there is too much noise in the input patterns, the network cannot perfectly recall one of the stored patterns and it only recalls a combination of the stored patterns (a phenomenon sometimes called cross-talk) or some pattern we did not intend the network to recall (those patterns are sometimes called spurious patterns).

It is well known that the number of patterns L that can be stored in a Hopfield network and be perfectly recalled by the network with S neurons is a function of the number of the neurons, i.e. $L = \alpha S$. The goal of this project is to measure α, the so-called storage capacity of the network, via simulation.

3. **Measuring the Storage Capacity of The Network**

   In this project, we want to measure how many patterns can be stored in a Hopfield Neural Network with $S = 100$ neurons, so that they can be perfectly recalled.

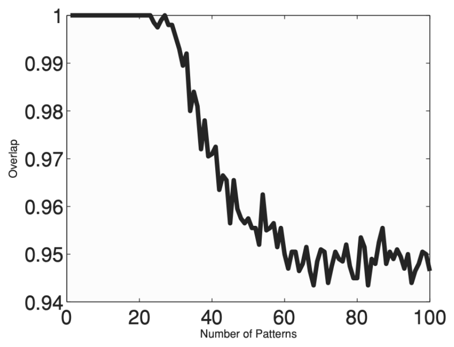
   (a) Do the following K times. You can choose K to be bigger than 20: (30 points)
   For $L \in \{1, 2, \ldots, S\}$:

   i. Create the matrix $\mathbf{X}_{S\times L}$ randomly. The elements must be $+1$ or -1 with equal probability. Remember the columns of this matrix are the patterns to be stored by your Hopfield model.

   ii. Use Hebbian learning to calculate $\mathbf{W}_{S\times S} = \sum_{l=1}^{L} \mathbf{x}^l(\mathbf{x}^l)^T$.

   iii. Choose one of the stored patterns randomly, initialize the output of the Hopfield network $\mathbf{a}(0)$ with that pattern and run it until its output stabilizes $\mathbf{a}(T+1) = \mathbf{a}(T)$. To exercise caution against possible oscillatory outputs, the maximum time steps can be set as 100. We wish that the stabilized output $\mathbf{a}(T)$ is the same as, or highly similar to the initial pattern $\mathbf{a}(0)$. Let's call the percentage of correctly recalled elements of $\mathbf{a}(0)$ in $\mathbf{a}(T)$, the overap, $O(L)$. When the number of patterns stored in the network L increases, it is more likely for this overlap to be less than 100%.

   (b) Average the K overlaps you got for each L and plot the result. You must see a plot like this:[4] (10 points)



[4]This is just an example and it does not mean that you will exactly get this plot.

(c) Repeat (3a) and (3b) for the Pseudo-inverse learning rule: $\mathbf{W} = \mathbf{X}\mathbf{X}^{+}$. (40 points)

(d) Determine the maximum number of patterns that can be stored in a Hopfield Neural Network and $\alpha = L/S$ with both Hebbian and Pseudo-inverse training. (20 points)

4. **Extra Credit: Robustness to Noise and Bias**

   (a) Repeat (3a) - (3d), but when creating the matrix $\mathbf{X}$, increase the probability of having +1 to 75%. What do biased patterns do to the storage capacity of the network? (15 points)

   (b) Repeat (3a)- (3d), but when doing (3(a)iii), choose one of the stored patterns randomly, randomly flip 20% of its elements (change +1 to -1 and vice versa), and initialize the output of the network $\mathbf{a}(0)$ with it. What does such noise do to the storage capacity of the network? (15 points)

Note: You must submit Extra Credit along with your main project. If you submit Extra Credit Late, your whole project will be late.