Model analysis and summary

This problem set addresses the final part of your project!

Over the course of the semester you have developed a project with tests and optimized the code via profiling. **Everything should be in place, so it is time to study the models!**

Continue with the good practice of writing modular code (classes, functions) when you write your code for the experiments!

Release the code as 'v7' (see release schedule). **Document your results including the relevant figures/tables in a markup file in your GitHub repository called 'summary.md'. Save results as pandas dataframes in '.hdf5' format**. Link to the summary from your 'Readme.md'. Some practical tips for organizing your report are in Section 3. The deadline for the release is January 14th!

1 Hopfield network

1.1 Capacity of the Hopfield network

An important feature of a Hopfield network is the storage capacity, i.e. how many patterns can the model store. The network is said to have stored a pattern if, when presented with a perturbed version of such pattern, the dynamical system converges to the original one. McEliece et al. [McE+87] derived an asymptotic convergence estimate for the number of patterns that a Hopfield network with the Hebbian rule can store as a function of the number of neurons. The asymptotic bound is $\frac{n}{2\log n}$ for the number of neurons n. In comparison, the asymptotic bound for a network trained with the Storkey rule is $\frac{n}{\sqrt{2\log n}}$ and hence the capacity is, higher than for networks trained with the Hebbian rule [Sto97].

In this final exercise you will empirically estimate the capacity of the Hopfield network, trained with the Hebbian and Storkey rules and simulated with the synchronous update rule.

Specific settings:

- Consider 10 networks of size ranging from 10 to 2500 with logarithmically arranged neuron numbers (`sizes=[10, 18, 34, 63, 116, 215, 397, 733, 1354, 2500]`).
- For each network size and learning rule, define t random base patterns, sampled around the theoretical estimate of the network capacity $C(N)$ (depending on the rule) use the following ranges:

  ```
  np.linspace(0.5 * C(N), 2 * C(N), 10).astype(int)
  ```

  Compute the network weights, sample patterns from the memorized ones, and apply a perturbation by changing 20% of the values of each base pattern.
- Run 10 trials for each network size by running the dynamical system varying the initial pattern and the perturbation. Log how many patterns converge to the original one, in order to compute the fraction of retrieved patterns for every t (and n). For doing so, we recommend to organize an experiment in the following way:

```
def experiment(size, num_patterns, weight_rule, num_perturb,
               num_trials=10, max_iter=100):
    # YOUR CODE HERE
    return results_dict
```

The `results_dict` should have the following keys: `"network_size"`, `"weight_rule"`, `"num_patterns"`, `"num_perturb"` and `"match_frac"`. For each n, these dictionaries can be saved in a list `results` that then is automatically converted to a pandas dataframe (see Section 3.3).

- For each network size, run the previously defined experiment (10 runs with different initial patterns) for different values of the parameter t. Include in your experiments values of t for which the system successfully converges to the initial pattern and others for which it does not. Estimate in this way the network capacity as a function of the network size and compare it with the theoretical asymptotic estimate.
- In addition to the result tables, save capacity curves for your experiments (plotting fraction of retrieved patterns vs. t for each n). Make sure the figure is well formatted.
- Save one plot with your empirical capacity curves including number of neurons vs. capacity (defined as #patterns that can be retrieved with $>= 90\%$ probability) for both learning rules and compare it to the theoretical estimate!

1.2 Robustness to perturbations

You just computed the capacity, assuming that a pattern can be stored if it converges to the original pattern from a 20% perturbed pattern in 90% of the cases. You might wonder how much you can perturb the original pattern, to still allow retrieval. You are soon going to find an answer to this question.
Your task:

- Consider the same network size range as in the previous tasks, but limit yourself to a value of t for which you found that the pattern retrieval (with 20% perturbation) works correctly for all your experiments (we recommend picking $t = 2$). Compute the network weights with the Hebbian rule. Progressively increase the initial perturbation by steps of 5%. At which point does the system stop converging to the initial pattern (in at least 9 out of 10 of your runs)?
- Repeat the experiment with the Storkey weights. Are the results the same as before?
- In addition to the result tables, save robustness curves for your experiments (plotting fraction of retrieved patterns vs. number of perturbations). Make sure the figure is well formatted!

1.3 Recalling image content from corrupted images

Select a few 100 x 100 pixel images (from the internet or other sources) and binarize them.[1] Store them in a Hopfield network and demonstrate that the model can recall the complete image from an incomplete subsets of an image (i.e. cover all pixels but the upper left corner). Show a few frames for the dynamics of your favorite retrieval example in your summary.

Remark: You can also experiment with image corruptions of a stored image and see if the Hopfield network can fix them.[2]

[1] Be sure to have the copyright for images that you will put online; e.g. utilize images from https://unsplash.com/s/photos/kostenlose-bilder. Also you can use https://scikit-image.org for downsampling and https://scikit-image.org/docs/dev/auto$_{e}xamples/developers/plot_{t}hreshold_{l}i.htmlsphx - glr - auto - examples - developers - plot - threshold - li - py$.

[2] These experiments with additional image corruptions will not be graded.

2 Turing pattern generation

During the last weeks you have noticed that simulating the evolution of even a simple reaction-diffusion system is a computationally intensive task. However, we have optimized the runtime and now can explore the emerging patterns!

2.1 Stability of the reaction-diffusion equation

Running the simulation of the dynamics for 1000 iterations with $\Delta t = 0.0001$ correspond to just 0.1 s of evolution of the dynamical system in real-world time! You are probably tempted to increase the value of Δt to simulate a longer duration with fewer iterations, but this might not be a good idea. In fact, a small Δt is a critical condition for the stability of the numerical method. The other parameters which contribute to the numerical instability of the system are the size of the grid cells (Δx and Δy) and the intensity of the reaction term (here proportional to γ). First, you will run some experiments to find for which combinations of these parameters the numerical method becomes unstable. To check whether the system is stable or not, just check the maximum of the reaction or diffusion term: if any of the two reaches a value of 1000 can be considered unstable. For example, this would mean that Δt is too big for the given set of parameters.

Your task:

- consider the following parameters: $M = 100$, $N = 100$, $\Delta x = 0.1$, $\Delta t = 0.0001$, $\alpha = 1.5$, $K = 0.125$, $\rho = 13$, $a = 103$, $b = 77$, $d = 7$, $\gamma = 0.5$ and verify that the system is numerically stable. Start from a perturbation of the stable state, which is 23 for v and 24 for u. For each experiment, create a pandas dataframe which contains the parameter used, if the system is stable or not and the time needed to run the experiment. Save the resulting table in a *hdf5* format.
- run the dynamics using 10 different values of the parameter γ, linearly spaced from from 1000 to 10000 using *np.linspace*. At which point does the system become unstable?
- go back to $\gamma = 0.5$. Now run the dynamics using 10 different values for $\Delta x = 0.1$ starting from 0.1 and decreasing to 0.0001. At which point does the system become unstable?
- Now use γ = 5000 for which the system is unstable. Can you make it stable again? Run the dynamics by reducing the value of Δt using 10 different values from 10^{-4} to 10^{-10} (use *np.linspace*).

2.2 Different patterns

An appealing feature of this pattern formation model is that it can explain the formation of a wide range of fur patterns (see Lecture 11). Indeed, it is possible to generate different patterns simply by modifying γ and the aspect ratio of the rectangular domain. Remember: a larger γ means increasing the size of the domain, as if we were modeling a larger rectangle patch of the animal's fur.

Your task:

- Now, generate a figure (using subplots containing 5 rows and 2 columns) for each of the following experiment (both for the reaction and diffusion term) of the last generated pattern with respect to the different parameters. In addition, as examples, save the resulting video only for some parameters and refer to it in your markdown summary. Do not store all the states as images, but only every 100th frame, so that the resulting video is smaller. Do you observe qualitative differences in the patterns which are created? In addition

- Consider 10 values of γ logarithmically arranged between 0.01 and 100 and keep all the other parameters as initially in section 2.1. For each value of γ, run a simulation of at least 20,000 steps.
- Consider 5 aspect ratios for the domain N (height) between 5 and 100 (use *np.linspace*) and keep the width $M = 100$. Like before, run a simulation of at least 20000 steps for each aspect ratio and observe whether you can appreciate some qualitative change in the pattern formation. In the summary figure, you will have now only 5 rows and 1 column.

3 Practical tips for your report

3.1 What should be in your summary?

In general, explain the experiment that you have performed and the results obtained.

Hopfield

Include two main sections:

- 1.1 - Store a pandas file with all your results and show capacity plots for all simulated network sizes (fraction of retrieved patterns vs. number of patterns) & a summary capacity plot (capacity (# patterns that can be retrieved with $>= 0.9$ probability) vs. number of neurons)
- 1.2 - Store a pandas file with all your results and show robustness plots for all simulated network sizes (fraction of retrieved patterns vs. number of perturbations)
- 1.3 - Example image sequence during image retrieval for your favorite example.

Turing

There should be two main sections:

- 2.1 - Stability of the reaction-diffusion equation. Explain the experiment that you have performed and the results obtained showing a table. There should be one table for each of the following experiments:
  - Stability of the system with respect to gamma
  - Stability of the system with respect to Δx
  - Stability of the system with respect to Δt
- 2.2 - Different patterns. Explain the experiment that you have performed and the results referring to the example videos. There should be one summary figure for each of the following experiments:
  - Different patterns for the diffusion term with respect to gamma
  - Different patterns for the reaction term with respect to gamma
  - Different patterns for the diffusion term with respect to the aspect ratio
  - Different patterns for the reaction term with respect to aspect ratio

3.2 Storing intermediate results

The simulations for the experiments will take around 20 minutes for Turing and 25 minutes for Hopfield. So it is highly recommended to store the results for each experiment (for particular parameters). We recommend pandas for this. Once you have generated the pandas dataframe for a particular experiment, you can save it using the .to_hdf(your_path, key ='df') function.

3.3 Formatting your report

Your report will be released as ‘summary.md‘. These files are written in markup, you can learn about the syntax in the Markup Guide.

The following code snippet shows you an example of how to save the results of your experiments:

```python
import pandas as pd

# Create a pandas DataFrame from your results dictionary
df = pd.DataFrame(results)

# Save dataframe as an hdf5 file
df.to_hdf(out_path, key='df')

# Additionally you can let pandas print the table in markdown format for easy pasting!
print(df.to_markdown())
```

You can embed figures in your markdown using the following command: ![alt text](url_link). In the *url_link* you can either specify the relative path in your github folder or the url that link to that image on the GitHub website.

Videos cannot be directly embedded in the markdown but you can upload your videos in the repository and then create an hyperlink to refer to the videos. For example, you can use the following command [gamma parameter](url_link). As before, you can either specify the relative path in your github folder or the url that link to that video on the GitHub website. In this case, ”gamma parameter” will be the text that is shown in the markdown that links to the reference.

References

[McE+87] ROBERTJ McEliece et al. “The capacity of the Hopfield associative memory”. In: *IEEE transactions on Information Theory* 33.4 (1987), pp. 461–482.

[Sto97] Amos Storkey. “Increasing the capacity of a Hopfield network without sacrificing functionality”. In: *International Conference on Artificial Neural Networks*. Springer. 1997, pp. 451–456.