Lecturer: Henning Sprekeler

Decision making in low-rank recurrent neural networks

The report for this project as well as the source code should be handed in no later than **July 31 2023, 11:59pm** through the Moodle interface. You can discuss your advances and hurdles with the project instructor until **July 14** during the weekly supervision meetings, or by sending them an email.

1 Introduction

Much of the brain's computational arises from its massively recurrent connectivity. But which connectivity patterns are required for a particular computation? One approach is to construct small networks with hard-wired connectivity, several examples of which we have encountered during the lectures. Unfortunately, this approach does not scale to more challenging, high dimensional problems where our intuition breaks down. An alternative approach is therefore to train artificial neural networks to perform the task of interest. This way, we only tell the model what it should do, not how – which is then the remaining problem [7, 1].

In general, understanding how such a network functions is very challenging. Recently, Mastrogiuseppe and Ostojic [5] proposed to simplify this problem by studying recurrent neural network (RNN) models with low-rank connectivity. In this project, we will train and reverse-engineer such low-rank RNNs. The ultimate goal is to distill the high-dimensional RNN into an equivalent low-dimensional circuit.

This project is based on Dubreuil et al. [3]. Start by reading the paper (at least sections 1 and 2.1,2) to understand the basic framework. Comparing your results with those reported in the paper can help you debug your simulations throughout the project.

2 Perceptual decision making

We will start by modeling a perceptual decision making task, akin to the random dot kinematogram discussed in the lecture. See Figure 2 of Dubreuil et al. [3].

1. Create a function for generating the data. Both the input u (noisy stimulus) and the target output y (identity of the stimulus) are one-dimensional. The input is defined by:

$$u(t) = \begin{cases} \overline{u} + \xi(t), & \text{if } 5 \leq t \leq 45 \\ \xi(t) & \text{otherwise.} \end{cases}$$

Here $\overline{u}$ is the stimulus strength, drawn uniformly from $\pm\frac{3.2}{100}\{1, 2, 4, 8, 16\}$, and ξ is background noise, drawn from a normal distribution with mean 0 and standard-deviation 0.03. The stimulus strength is re-sampled for each trial, the noise is re-sampled for each time-step. The target y is defined as the sign of $\overline{u}$. Write your function such that it can generate multiple trials at the same time, this will be useful for training our network using stochastic gradient descent. Plot the data for several trials to make sure the labels match the inputs, and the noise levels are reasonable.

2. Implement the recurrent neural network in your favorite deep learning library. The network has one-dimensional inputs and outputs, and an arbitrary number N of recurrent units. Its trainable parameters are the left and right connectivity vectors m, n. It also has fixed input and output weights I and w, respectively. All (trainable and fixed) parameters are sampled from a standard normal distribution. The dynamics of the ith unit are given by

$$\tau \frac{dx_i}{dt} = -x_i + \sum_{j=1}^{N} J_{ij}\phi(x_j) + I_i u(t), \quad i = 1, \ldots, N$$

   Here $\phi = \tanh$ is the neuron's transfer function, and $\tau = 100\text{ms}$ the neuronal time constant. Simulate the network using forward Euler with a time step of $\Delta t = 20\text{ms}$ (the stimulus duration of 75 time steps corresponds to 1500ms real time). The rank one matrix J is defined as

$$J = \frac{1}{N} m n^T.$$

   The network's output is a linear readout of the rates:

$$z(t) = \frac{1}{N} \sum_{i=1}^{N} w_i \phi(x_i).$$

3. Train a network of size $N = 128$[1] by minimizing the mean squared error

$$\frac{1}{BT} \sum_{i,t} (z_i(t) - y_i(t)^2$$

   using stochastic gradient descent. Here, the sum runs over all B trials in a mini-batch and the last $T = 15$ time steps (the target is only defined during this last 'decision epoch'). Use Adam [4] with a learning rate of 5e-3 and a batch size of 32. The loss should converge to a value below 5e-2, which might require 1000 parameter updates. If the loss converges to higher values, debug your code by (1) overfitting the network on a single batch, and (2) training a full-rank network. Make sure to look at the the network's input, output, and hidden dynamics once it has successfully converged.

4. We now have a neural network model that performs perceptual decision making, but this is only a first step. Next, we want to understand how this model works, to generate hypotheses how the brain might perform analogous functions. Dubreuil et al.[3] found that their network relied on a specific pattern of correlations between the connectivity vectors (their figure 2b). Explain why this pattern works for the task, and compare it with the pattern in your network.

5. The theory of low-rank networks relies on Gaussian approximations (see methods and supplements of [3]), which characterizes connectivity vectors by their second order statistics. Test if this assumption holds in your trained network by (1) fitting a 4-dimensional

[1] We will train smaller networks than done in the paper. This makes the Gaussian approximation worse, but means that training is fast.

Gaussian distribution to the connectivity vectors, (2) Resampling from the fitted distribution, (3) Comparing the empirical with the resampled distribution, and (4) Testing networks with the resampled connectivity by comparing the sign of the output with the sign of the target.

6. The dynamics of our model are low-dimensional, making dimensionality reduction [2] a promising approach. In this case, we already know the 2-dimensional space in which the dynamics live: it is spanned by the input vector I and the left-connectivity vector m (why?). Confirm this by projecting the N dimensional, time-varying dynamics $x(t)$ onto the $m - I$ plane. Do the dynamics of different trials evolve in an interpretable manner? How could you reduce the dimensionality if you did not know m and I, as would be the case with an arbitrary RNN, or with experimental data?

7. Using the Gaussian assumption, we are ready for the final step: distilling the trained network into an equivalent one-dimensional dynamical system of the form

$$\frac{d\kappa}{dt} = -\kappa(t) + \tilde{\sigma}_{mn}\kappa(t) + \tilde{\sigma}_{nI}v(t),$$

with v the external inputs filtered by the neuronal time constant:

$$\tau\frac{dv}{dt} = -v(t) + u(t).$$

Each coupling term $\tilde{\sigma}_{ab}$ is the product $\sigma_{ab}\langle\Phi'\rangle(\Delta)$ of the corresponding covariance and the neuronal gain, averaged over the Gaussian statistics of the population:

$$\langle\Phi'\rangle(\Delta) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\phi'(\Delta z)\, e^{-z^2/2}dz.$$

Here, Δ captures the system's non-linearity because it depends on the state κ and inputs v:

$$\Delta = \sqrt{\sigma_m^2\kappa^2 + \sigma_I^2 u^2}.$$

First, compute the necessary covariances σ_{mn}, σ_{nI} etc. from your trained network. Then, use these to simulate the low-dimensional dynamics $d\kappa/dt$. You can use scipy's `quadrature` function to integrate the Gaussian integral. Does the equivalent circuit indeed perform the task? If it does not, try using the covariances given in the paper: $\sigma_{mn} = 1.4, \sigma_{nI} = 2.6, \sigma_{mw} = 2.1$, and $\sigma_w = \sigma_n = \sigma_I = 1$.

8. A complementary way of reverse-engineering a trained RNN, is to analyze its locally linear dynamics around fixed points [8]. This is a standard approach in low-dimensional systems, done numerically when applied to RNNs. Find the fixed points of the low-dimensional circuit by minimizing $(d\kappa/dt)^2$ over κ. You can use scipy's `minimize` for this. How would the result change in a circuit that integrates the exact value of its input, rather than the sign?

3 Parametric working memory

Most computations happen in more than one dimension. Here, we will look at such a computation by modeling parametric working memory (see Figure 3 from Dubreuil et al. [3]). The different steps in this exercise are analogous to those in the first exercise. The goal is to (1) see how the low rank framework applies to networks with rank two, and (2) get more practice with training and reverse-engineering RNNs without relying on the strict instructions from exercise 1.

1. Create a function for generating the data. Both the input u and the target output y are again one-dimensional. The input now consist of a first stimulus, followed by a delay and the second stimulus [2]:

$$u(t) = \begin{cases} u_1 & \text{if } 5 \le t \le 10 \\ u_2 & \text{if } 60 \le t \le 70 \\ 0 & \text{otherwise.} \end{cases}$$

   The two stimuli are defined as

$$u_i = \frac{1}{f_{\max} - f_{\min}} \left(f_i - \frac{f_{\max} + f_{\min}}{2} \right), \quad i = 1, 2$$

   with f_i sampled uniformly from the set $\{10, 14, 18, 22, 26, 30, 34\}$, and $f_{\min} = 10, f_{\max} = 34$ the smallest and largest elements of that set. The target output y is the normalized difference between the stimuli:

$$y = \frac{f_1 - f_2}{f_{\max} - f_{\min}}.$$

2. Extend your network implementation to allow for recurrent connectivity of arbitrary rank.

3. Train a network of size $N = 128$ and rank $R = 2$ by minimizing the mean squared error between the target and the network output during the last 5 time steps of each trial. The loss should drop below 5e-3. If it does not, make sure to visualize your data and the networks output, and try debugging the network using the tips from exercise 1. In addition, you can try a curriculum learning approach by gradually increasing the delay period between the two stimuli, starting from 25 time steps.

4. Visualize the connectivity patterns from the trained network, and compare with those reported by Dubreuil et al. (their figure 3b). Why do these patterns work? Why could a rank 1 network not solve this task.

5. Project the N-dimensional activity onto the m_1, m_2, to obtain latent variables κ_1, κ_2. Is their activity show the two time scales you might expect from the connectivity patterns?

6. Test if the covariance patterns characterize the network connectivity by fitting and resampling a 6-dimensional Gaussian to the connectivity patterns.

7. Under the appropriate assumptions, the trained network can now be described by a two-dimensional equivalent circuit. Its dynamics are given by

$$\begin{aligned} \frac{d\kappa_1}{dt} &= -\kappa_1 + \tilde{\sigma}_{m_1 n_1} \kappa_1 + \tilde{\sigma}_{n_1 I} v(t), \\ \frac{d\kappa_2}{dt} &= -\kappa_2 + \tilde{\sigma}_{m_2 n_2} \kappa_2 + \tilde{\sigma}_{n_2 I} v(t). \end{aligned}$$

   The coupling terms again encapsulate the system's nonlinearity, here via the scaling

$$\Delta = \sqrt{\sigma_{m_1}^2 \kappa_1^2 + \sigma_{m_2}^2 \kappa_2^2 + \sigma_I^2 v^2}.$$

   Simulate the effective circuit for different inputs v. The circuit should perform qualitatively, but not necessarily quantitatively, the same as the full network. If it does not, try using the covariances reported in the paper: $\sigma_{n_1 m_1} = 1, \sigma_{n_2 m_2} = 1.0, \sigma_{n_1 I} = 0.5, \sigma_{n_2 I} = 1.9, \sigma_{m_1 w} = 2.8, \sigma_{m_2 w} = -2.2$.

[2] The (somewhat circuitous) definition of the stimuli is meant to evoke the working memory task developed by Romo et al. [6] in which subjects have to compare vibrotactile stimuli of different frequencies.

4 Report

Write a report (no more than 4 pages of text with discussion and interpretation as a PDF file) that presents the results of your simulations and analysis.

- Give a brief introduction to the low-rank framework.
- Explain how the connectivity patterns used in this project solve the tasks.
- Present your simulations and their results.
- Discuss potential future work. For example, what are current limitations of the framework you would like to address? What neural computations could be investigated using the framework?

Marks will be based on the clarity of presentation and on the thoroughness of the analysis.

References

[1] Omri Barak. Recurrent neural networks as versatile tools of neuroscience research. *Current opinion in neurobiology*, 46:1–6, 2017.

[2] John P Cunningham and M Yu Byron. Dimensionality reduction for large-scale neural recordings. *Nature neuroscience*, 17(11):1500, 2014.

[3] Alexis Dubreuil, Adrian Valente, Manuel Beiran, Francesca Mastrogiuseppe, and Srdjan Ostojic. Complementary roles of dimensionality and population structure in neural computations. *bioRxiv*, 2020.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Francesca Mastrogiuseppe and Srdjan Ostojic. Linking connectivity, dynamics, and computations in low-rank recurrent neural networks. *Neuron*, 99(3):609–623, 2018.

[6] Ranulfo Romo, Carlos D Brody, Adrián Hernández, and Luis Lemus. Neuronal correlates of parametric working memory in the prefrontal cortex. *Nature*, 399(6735):470–473, 1999.

[7] David Sussillo. Neural circuits as computational dynamical systems. *Current opinion in neurobiology*, 25:156–163, 2014.

[8] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.