Problem Set 3

Quan Wen

Due date: Nov 10, 2026

Reading

Reading. Sussillo, D., Churchland, M.M., Kaufman, M.T., & Shenoy, K.V. (2015). *A neural network that finds a naturalistic solution for the production of muscle activity. Nature Neuroscience*, 18, 1025–1033.

Clark, D.G., Bordelon, B., Zavatone-Veth, J.A., & Pehlevan, C. (2026). *Structure, disorder, and dynamics in task-trained recurrent neural circuits. bioRxiv*, 2026.03.02.708943. (Provides the publicly available Sussillo data and reference RNN training code.)

The question

Train a recurrent neural network (RNN) to reproduce muscle activity (EMG) following Sussillo et al. (2015), and evaluate how the learned network dynamics relate to recorded neural population activity.

Data

You must use the Sussillo et al. (2015) dataset, which includes both recorded neural activity (M1/PMd) and EMG targets for training the RNN. The data is packaged as a single `sussillo_data.npz` file.

- **GitHub:** `https://github.com/davidclark1/RNN-Learning-Theory`
- **Zenodo (direct download):** Zenodo record (download `sussillo_data.tar`)
- **Data loader:** `sussillo_data/load_sussillo_data.py` converts raw MATLAB files to `.npz`
- **Data documentation:** `sussillo_data/README.md` for variable descriptions and provenance

The repository also includes a reference RNN training script (`monkey/train.py`) and analysis notebook (`monkey/analysis.ipynb`). You may consult these for guidance, but you should write your own implementation for the homework.

Background: Sussillo et al. (2015)

Sussillo, Churchland, Kaufman, and Shenoy (2015) trained recurrent neural networks to reproduce the electromyographic (EMG) signals recorded from multiple arm muscles during the same reaching task. The key design choices were:

- **Architecture:** A continuous-time RNN with $N = 200$ recurrent units. The dynamics follow:

  $$\tau \, \dot{\mathbf{r}} = -\mathbf{r} + f(W_{\text{rec}} \, \mathbf{r} + W_{\text{in}} \, \mathbf{u} + \mathbf{b})$$

  where $\mathbf{r}$ is the vector of firing rates, f is a pointwise nonlinearity (tanh), W_{rec} is the recurrent weight matrix, W_{in} maps inputs to the network, and $\mathbf{u}$ is the external input.

- **Inputs:** A condition-specific input signal (encoding reach direction) provided during the preparatory period and removed at movement onset, plus a go cue signal at movement onset.

- **Output:** A linear readout $W_{\text{out}} \, \mathbf{r}$ that maps firing rates to EMG signals for multiple muscles:

  $$\mathbf{y} = W_{\text{out}} \, \mathbf{r} + \mathbf{b}_{\text{out}}$$

- **Training objective:** Minimize the mean squared error between the RNN output and the recorded EMG during the movement epoch. The RNN was *not* trained to match neural activity—only to produce the correct muscle outputs.

- **Regularization:** Critically, the RNN was regularized to encourage simple dynamics. Regularization terms penalized the magnitude of firing rates ($\|\mathbf{r}\|^2$), the input drive ($\|W_{\text{rec}} \, \mathbf{r}\|^2$), and the squared norm of the weight matrices. The strength of regularization was a key parameter: heavily regularized networks produced dynamics that more closely resembled recorded neural activity than unregularized networks.

- **Training method:** Hessian-free optimization (Martens & Sutskever, 2011), though modern implementations can use BPTT with Adam.

Building the RNN

Implement a continuous-time RNN in PyTorch (or a framework of your choice) with the following specifications:

- $N = 200$ recurrent units with tanh nonlinearity
- Time constant $\tau = 50\,\text{ms}$
- Condition-specific input (2D vector encoding reach direction) provided during the preparatory period

- A binary go cue signal at movement onset
- Linear readout to K output channels, where K is the number of EMG channels in the dataset

EMG data. Use the EMG targets from the Sussillo et al. (2015) dataset (`sussillo_data.npz`), which contains trial-averaged, rectified, and smoothed EMG from multiple arm muscles recorded simultaneously with the neural data. These are the *same* EMG signals used as RNN training targets in the original paper. Load the data using the provided `load_sussillo_data.py` script or directly from the `.npz` file (see `sussillo_data/README.md` for field names and shapes). The EMG targets, the condition-specific inputs, and the time axis are all included.

(a) Describe your network architecture, including the number and form of inputs, the recurrent dynamics equation, the readout, and how you handle the preparatory vs. movement epochs.

(b) Define your loss function. Include both the task loss (MSE on outputs during movement epoch) and regularization terms. Following Sussillo et al., add:

(i) L2 on firing rates: $\lambda_r \cdot \|\mathbf{r}\|^2$

(ii) L2 on recurrent input: $\lambda_u \cdot \|W_{\text{rec}}\,\mathbf{r}\|^2$

(iii) L2 on weights: $\lambda_w \cdot \left(\|W_{\text{rec}}\|^2 + \|W_{\text{in}}\|^2 + \|W_{\text{out}}\|^2\right)$

Report the values of λ_r, λ_u, λ_w you use.

Training and Evaluating the RNN

Train the RNN using backpropagation through time (BPTT) with Adam optimizer (or your optimizer of choice). Train until the network can accurately reproduce the target outputs for all reach conditions.

(a) Plot the training loss over epochs. Demonstrate that the RNN has learned to produce reasonable outputs by plotting the target vs. predicted outputs for 4 representative conditions.

(b) Train two versions of the RNN: one with strong regularization and one with weak or no regularization. Report the final task loss and the regularization loss for both.

(c) Visualize single-unit activity in both networks for 6 example units across all conditions. In the strongly regularized network, do you observe multiphasic, temporally complex firing patterns similar to those in the recorded neural data provided with the dataset? How do the weakly regularized units differ?

Comparing RNN Dynamics to Recorded Neural Activity

Use dimensionality reduction and subspace comparisons to quantify how similar the RNN hidden-unit trajectories are to the recorded neural population trajectories (M1/PMd firing rates) in the same dataset.

(a) Compute PCA on (i) recorded neural trajectories and (ii) RNN hidden-unit trajectories (for both strongly and weakly regularized networks). Plot cumulative variance explained and example 2D/3D trajectory projections for several conditions.

(b) Compute canonical correlations (CCA) between the top-6 PC trajectories of the recorded neural data and the top-6 PC trajectories of each RNN. Report the top 3 canonical correlations for each network.

(c) In 3–4 paragraphs, discuss: How does regularization change the geometry/complexity of the learned dynamics? Why might a simplicity bias push networks toward low-dimensional solutions that better match recordings? Relate your findings to Occam's razor in neural computation (metabolic costs, wiring economy, and robustness).

(d) explore other summary statistics of neural dynamics you can think about that may capture the similarity and fundamental differences between the trained model and the experimental data.

Note. Submit a Jupyter notebook (or equivalent) containing your code, figures, and written responses. Figures should be clearly labeled (axes, titles, legends) and your written responses should be concise but complete. Start by debugging with a small network (e.g., $N = 50$) before scaling to $N = 200$; regularization strength is typically the most important hyperparameter; and for CCA you may use `sklearn.cross_decomposition.CCA` (or implement CCA via SVD).