Homework #3

CSE 599N: Machine Learning for Neuroscience
Professor Matt Golub
Due: **Wednesday** May 8, 2024 1:00pm

Please review all homework guidance posted on the website before submitting to Gradescope. Reminders:

- All code must be written in Python and all written work must be typeset (e.g. LaTeX).
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to 10% of the value of each question not properly linked. For instructions, see `https://www.gradescope.com/get_started#student-submission`.

Important: By turning in this assignment (and all that follow), you acknowledge that you have read and understood the collaboration policy with humans and AI assistants alike: `https://courses.cs.washington.edu/courses/cse599n/24sp/assignments/`. Any questions about the policy should be raised at least 24 hours before the assignment is due. There are no warnings or second chances. If we suspect you have violated the collaboration policy, we will report it to the college of engineering who will complete an investigation. Not adhering to these reminders may result in point deductions.

Instructions

For HW3, we have provided you with the following files:

- `HW3.pdf` (this file)
- `HW3-Q12.ipynb` (Jupyter notebook containing the starter code for questions 1 and 2)
- `HW3-Q3.ipynb` (Jupyter notebook containing the starter code for question 3)
- `HW3-data.zip` (file containing data for all questions)

Please fill out each of the cells in the notebooks that are labeled with `"YOUR CODE HERE"` with your answer for each sub-question. You can either make the notebook self-contained, with all of your code in it, or feel free to write additional Python code which you import and call from your notebook. Once completed, add all relevant plots and written answers to a PDF for your writeup submission.

Submission

Please follow the following instructions to submit your assignment.

1. Make sure your code files are named `HW3-Q12.ipynb` and `HW3-Q3.ipynb` and submit to the HW3 Code Gradescope assignment. If you have any additional Python code, please also submit these files to the same assignment.
2. Convert your `HW3-Q12.ipynb` and `HW3-Q3.ipynb` files into PDFs and combine into one file. Submit this file to the HW3 Writeup Gradescope assignment. You may find Preview (on macOS) or the `pdfpages` LaTeXpackage to be useful in merging PDFs, but there are many ways to accomplish this. For both the code and writeup, you may submit as many times as you would like before the deadline.
3. For the writeup, make sure to tag the pages which include your answers to the questions.

Questions

Data for Questions 1 and 2:

Data for Questions 1 and 2 can be found on the course Canvas page under the "HW 3" module.

`X_real`, from `hw3-q1_train.npy`: a 728×97 matrix of real data, where each row is a spike count vector across 97 simultaneously-recorded neurons [1]. This is exactly the same data analyzed in HW2 (problem 4), but now presented in a slightly different format. As in HW2, the spike counts are taken in a 200 ms bin during the plan period. There are 91 trials for each of 8 reaching angles, for a total of 728 trials. Trials 1 to 91 correspond to reaching angle 1, trials 92 to 182 correspond to reach angle 2, etc.

`X_sim`, from `hw3-q2_sim.npy`: a 8×2 matrix of simulated data, where each row is a data point.

1. **[20 points] Visualization of high-dimensional neural activity using PCA**

   In Homework 2, we classified neural activity from 97 neurons across 8 different reaching angles with greater than 90% accuracy. This indicates that there were 8 well-separated clusters of data points in 97-dimensional space. However, we did not attempt to visualize these clusters due to the high dimensionality of the data. Here, we will apply PCA to these same data (`X_real`) to gain some intuition about why we were able to classify so well in Homework 2. The data points are $\mathbf{x}_n \in \mathbb{R}^D (n = 1, \ldots, N)$, where $D = 97$ is the data dimensionality and $N = 728$ is the number of data points.

   (a) [**5 points**] Plot the square-rooted eigenvalue spectrum. If you had to identify an elbow in the eigenvalue spectrum, how many dominant eigenvalues would there be? What percentage of the overall variance is captured by the top 3 principal components?

   (b) [**10 points**] For the purposes of visualization, we'll consider the PC space defined by the top $M = 3$ eigenvectors. Project the data into the three-dimensional PC space. Plot the three-dimensional projected points, and color each dot appropriately according to reaching angle (there should be a total of 728 dots). Find a rotation of the three-dimensional plot in which the clusters are well-separated.

   (c) [**5 points**] Define a matrix $U_M \in \mathbb{R}^{D \times M}$ containing the top three eigenvectors (i.e., PC directions), where $U_M(\mathtt{d}, \mathtt{m})$ indicates the contribution of the $\mathtt{d}^{\text{th}}$ neuron to the $\mathtt{m}^{\text{th}}$ principal component. Show the values in U_M by calling `plt.imshow(U_M)`. (Note: Also call `plt.colorbar()` to show the scale.) Are there any obvious groupings among the neurons in each column of U_M, as hypothesized in class?

2. **[80 points] From PCA to PPCA and FA**

   We will compare PCA, PPCA, and FA on a toy problem using the data in `X_sim`. The data points are $\mathbf{x}_n \in \mathbb{R}^D (n = 1, \ldots, N)$, where $D = 2$ is the data dimensionality and $N = 8$ is the number of data points. We will project the data into a $M = 1$ dimensional space.

   (a) [**10 points**] Create one plot containing all of the following for PCA:

   - Plot each data point $\mathbf{x}_n$ as a black dot in a two-dimensional space.
   - Plot the mean of the data $\boldsymbol{\mu}$ as a big green point.
   - Plot the PC space defined by $\mathbf{u}_1$ as a black line. (Hint: This line should pass through $\boldsymbol{\mu}$.)
   - Project each data point into the PC space, and plot each projected data point $\hat{\mathbf{x}}_n$ as a red dot. (Hint: The projected points should lie on the $\mathbf{u}_1$ line.)

[1] The neural data have been generously provided by the laboratory of Prof. Krishna Shenoy at Stanford University. The data are to be used exclusively for educational purposes in this course.

- Connect each data point $\mathbf{x}_n$ with its projection $\hat{\mathbf{x}}_n$ using a red line. (Hint: The red lines should be orthogonal to the PC space. To see this, you will need to call `plt.axis('equal')`.

(b) **[30 points]** Implement the EM algorithm for PPCA in Python. You may use common computing packages (such as `numpy` or `scipy`), but do not use an existing PPCA package (e.g., of `scikit-learn`). Run your algorithm on the data in `Xsim`. Plot the log data likelihood $\left(\sum_{n=1}^{N} \log P\left(\mathbf{x}_n\right)\right)$ versus EM iteration. (Hint: The log data likelihood should increase monotonically with EM iteration. You should run enough EM iterations to see a long, flat plateau.)

(c) **[5 points]** Using the parameters found in part (b), what is the PPCA covariance $\left(WW^{\mathrm{T}}+\sigma^2 I\right)$? If you did part (b) correctly, the PPCA covariance should be very similar to the sample covariance.

(d) **[10 points]** Create one plot containing all of the following for PPCA:

- Plot each data point $\mathbf{x}_n$ as a black dot in a two-dimensional space.
- Plot the mean of the data $\boldsymbol{\mu}$ as a big green point.
- The PC space found by PPCA is defined by W, which in this case is a twodimensional vector. Check that PC space defined by W is identical to that found by PCA. Plot the PC space as a black line. (Hint: If there is some small discrepancy between the PC spaces found by PCA and PPCA, run more EM iterations for PPCA.)
- Project each data point into the PC space using PPCA, and plot each projected data point $\hat{\mathbf{x}}_n = WE\left[\mathbf{z}_n \mid \mathbf{x}_n\right] + \boldsymbol{\mu}$ as a red dot. (Hint: The projected points should lie on the line defined by W.)
- Connect each data point $\mathbf{x}_n$ with its projection $\hat{\mathbf{x}}_n$ using a red line. Why are the red lines no longer orthogonal to the PC space?

(e) **[10 points]** Implement EM algorithm for FA. You should be able to do this with only a small modification to your PPCA code. Run the algorithm on the data in `Xsim`. Plot the log data likelihood $\left(\sum_{n=1}^{N} \log P\left(\mathbf{x}_n\right)\right)$ versus EM iteration. (Hint: The log data likelihood should increase monotonically with EM iteration. You should run enough EM iterations to see a long, flat plateau.)

(f) **[5 points]** Using the parameters found in part (e), what is the FA covariance $\left(WW^{\mathrm{T}}+\Psi\right)$? If you did part (e) correctly, the FA covariance should be very similar to the sample covariance.

(g) **[10 points]** Create one plot containing all of the following for FA:

- Plot each data point $\mathbf{x}_n$ as a black dot in a two-dimensional space.
- Plot the mean of the data $\boldsymbol{\mu}$ as a big green point.
- The low-dimensional space found by FA is defined by W, which in this case is a two-dimensional vector. Plot the low-dimensional space as a black line. (Hint: This line should pass through $\boldsymbol{\mu}$.) Why is the low-dimensional space found by FA different from that found by PCA and PPCA?
- Project each data point into the low-dimensional space using FA, and plot each projected data point $\hat{\mathbf{x}}_n = WE\left[\mathbf{z}_n \mid \mathbf{x}_n\right] + \boldsymbol{\mu}$ as a red dot. (Hint: The projected points should lie on the line defined by W.)
- Connect each data point $\mathbf{x}_n$ with its projection $\hat{\mathbf{x}}_n$ using a red line.

3. **[100 points] Kalman Filter**

In Homework 2, we classified neural activity recorded during movement planning to one of eight possible classes (where each class corresponds to a reaching angle). This is referred to as discrete neural decoding and is analogous to typing on a keyboard. Here, we will decode the moment-by-moment details of arm movements using the neural activity recorded around the time of movement execution, as illustrated in class. This is referred to as continuous neural decoding. We will analyze real neural data recorded using a 100 electrode array in premotor cortex of a macaque monkey[2]. The dataset can again be found on Canvas, under the HW 3 module.

The following describes the data format. The `hw3-q3_train_trial.npy` and `hw3-q3_test_trial.npy` files have two variables: `train_trial` contains the training data and `test_trial` contains the test data. When `hw3-q3_train_trial.npy` is loaded, the resulting variable is a structured NumPy array of dimensions (91 trials) $\times$ (8 reaching angles). Each entry contains spike trains recorded simultaneously from 97 neurons while the monkey reached 91 times along each of 8 different reaching angles. The spike train recorded from the i-th neuron on the nth trial of the kth reaching angle is contained in `train_trial[n,k]['spikes'][i,:]`, where `n` = 1, ..., 91, `k` = 1, ..., 8, and `i` = 1, ..., 97. A spike train is represented as a sequence of zeros and ones, where time is discretized in 1 ms bins. A zero indicates that the neuron did not spike in the 1 ms bin, whereas a one indicates that the neuron spiked once in the 1 ms bin. Each spike train is taken from movement onset to movement end. The length of the spike trains differ from trial to trial because each arm reach takes a different amount of time, even for repeated reaches to the same target.

The arm trajectory recorded on the n-th trial of the k-th reaching angle is contained in `train_trial[n,k]['handPos']`. The hand position (in mm) in the horizontal (`handPos[0,:]`) and vertical (`handPos[1,:]`) directions are taken from movement onset to movement end, at 1 ms time steps. On each trial, the data in `spikes` and `handPos` are aligned in time. The structure of the loaded variable from `test_trial.npy` is the same as of `train_trial`.

- **Data Preprocessing**

  The first step is to take binned spike counts and prepare the arm state vectors.

  - [**10 points**] Take spike counts in non-overlapping 20 ms bins. For each trial, the result should be a $97 \times T$ matrix, where T is the number of 20 ms time bins on that trial. (Note: because the length of spike trains is generally not an integer multiple of 20 ms, you can discard any partial time bins at the end of each trial. Also, T may be different on each trial.)
  - [**10 points**] We will use a 4-dimensional arm state $\mathbf{z}_t$ in this problem, where

  $$\mathbf{z}_t = \begin{bmatrix} \text{horz position} \\ \text{vert position} \\ \text{horz velocity} \\ \text{vert velocity} \end{bmatrix}.$$

    Sample the arm position at 20 ms intervals at the end of each spike count bin. Then, take the first differences of the sampled arm position to obtain velocity (expressed in m/s). For each trial, the result should be a $4 \times T$ matrix, where T is the number of 20 ms timesteps (or time bins) in that trial.

- **[20 points] Training Phase**

  Fit the model parameters $\theta = \{A, Q, \pi, V, C, R\}$ to the training data. Show the values in the 4×4 matrices A and Q. (Hint: The relationship between position and velocity defined by A should be consistent with the laws of physics.)

- **Test Phase**

[2]The neural data have been generously provided by the laboratory of Prof. Krishna Shenoy at Stanford University. The data are to be used exclusively for educational purposes in this course.

Using the model parameters you found, we will create two figures; one for each of the following trials: `test_trial[0,0]` and `test_trial[0,3]`. Each figure should have three subplots. For the first subplot in each figure, apply the Kalman filter to decode an arm trajectory for each test trial. In the subplots, show:

  - [**10 points**] The mean position estimate $\mu_t^t (t = 1, \ldots, T)$ as red points connected by a red line (note: μ_t^t is a 4-dimensional vector that includes both position and velocity estimates, but you can simply ignore the velocity estimate here);
  - [**10 points**] The one-standard-deviation confidence ellipse corresponding to the uncertainty Σ_t^t of each position estimate in red; and
  - [**5 points**] The actual arm trajectory as a black line.

- Now, we will plot the same data but versus time. For *each* of the two trials, in the second subplot, show horizontal position versus time. In the third subplot, show vertical position versus time. In each of these subplots, show:
  - [**10 points**] The decoded mean trajectory as a red solid line;
  - [**10 points**] The one-standard deviation confidence intervals using red dotted lines (note: there should be one dotted line above and another below the decoded mean trajectory); and
  - [**5 points**] The actual arm trajectory as a black line.

- [**10 points**] Performance Evaluation

  At each time point, find the distance (in mm) between the decoded position and actual position. Average these distance errors across the timepoints of all test trials, and report this average error.