IN THE NAME OF GOD

HW4:

Neuroscience, Learning, Memory, Cognition Course

Sharif University of Technology

SaraRezanezhad99101643

June2023

PART I: Practical

1.Brain regions in the dataset

By defining an empty array and using a for loop including conditional statements, we could obtain the ‘barea’ array according to brain_area list.

```
barea = []
for area in brain_area:
    if area in ['VISa', 'VISam', 'VISl', 'VISp', 'VISpm', 'VISrl']:
        barea.append(0.)
    elif area in ['CL', 'LD', 'LGd', 'LH', 'LP', 'MD', 'MG', 'PO', 'POL', 'PT', 'RT', 'SPF', 'TH', 'VAL', 'VPL', 'VPM']:
        barea.append(1.)
    elif area in ['CA', 'CA1', 'CA2', 'CA3', 'DG', 'SUB', 'POST']:
        barea.append(2.)
    else:
        barea.append(3.)

print(barea)
#print(brain_area.size)
```

```
[0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 2.0, 2.0, 0.0, 0.0, 2.0, 2.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 2.0, 2.0, 2.0, 0.0, 2.0, 0.0,
```

2.1.Plots by brain region

The following code computes the average firing rates of neurons in different brain regions, for each combination of LEFT/RIGHT/NOGO(NONE) responses. It then plots the results to visualize the differences in firing rates between the different response types.

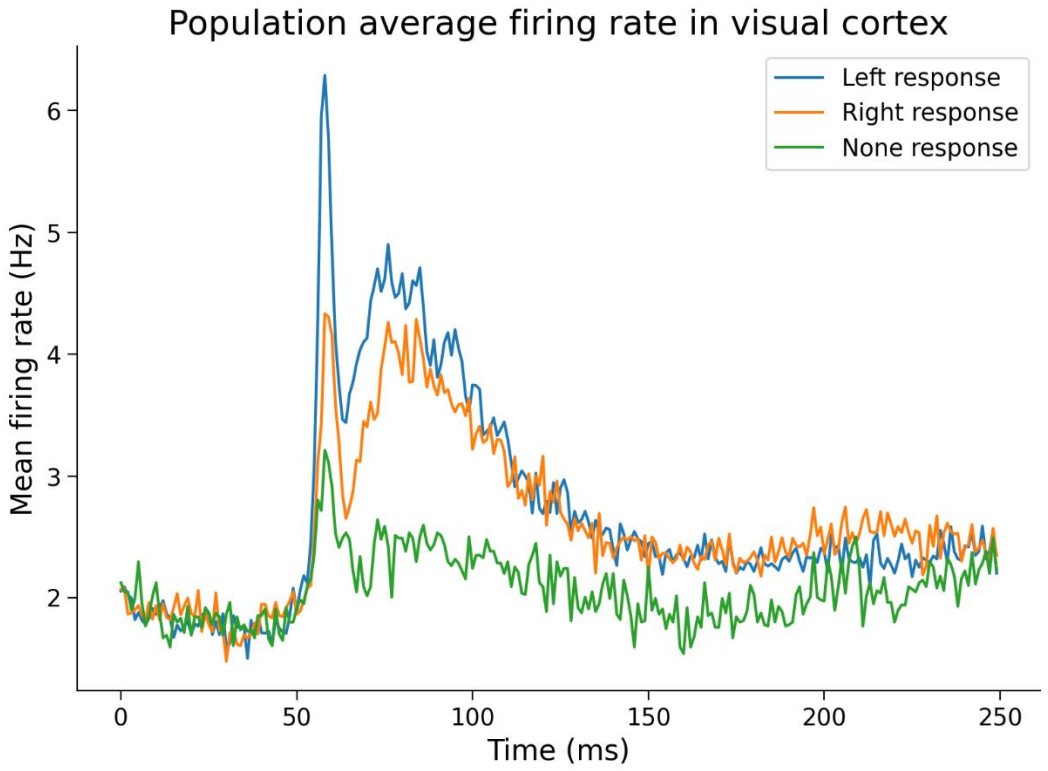
At the beginning of the code, the response variable is extracted from the input data dictionary 'dat' and assigned to the variable 'response'. The values in 'response' represent the types of responses made by the rat: RIGHT (-1), NOGO (0), or LEFT (1).

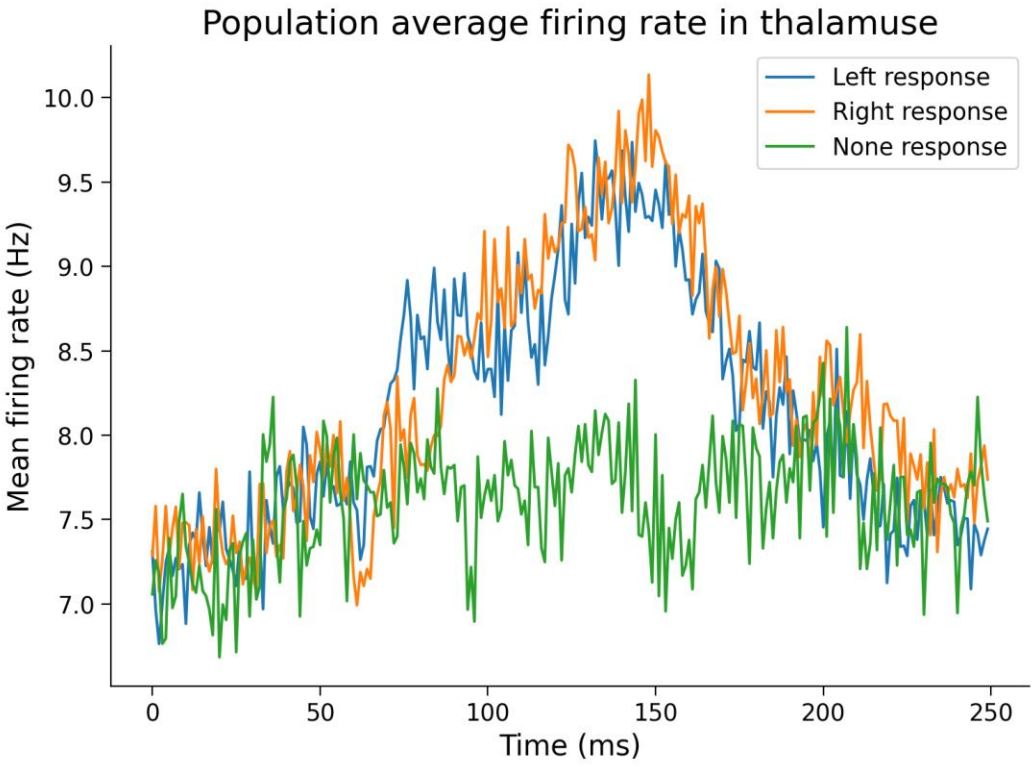
The 'brain_activity' function takes two arguments: a 'brain_group' parameter that specifies the brain region of interest, and a 'label' parameter that specifies the name of the region for use in the plot title. The function loops over all unique values of 'barea' (the brain area variable) and calculates the average firing rates for neurons in the specified region and for each type of response. The firing rates are computed using the 'mean()' function of the numpy library over all trials and time points.

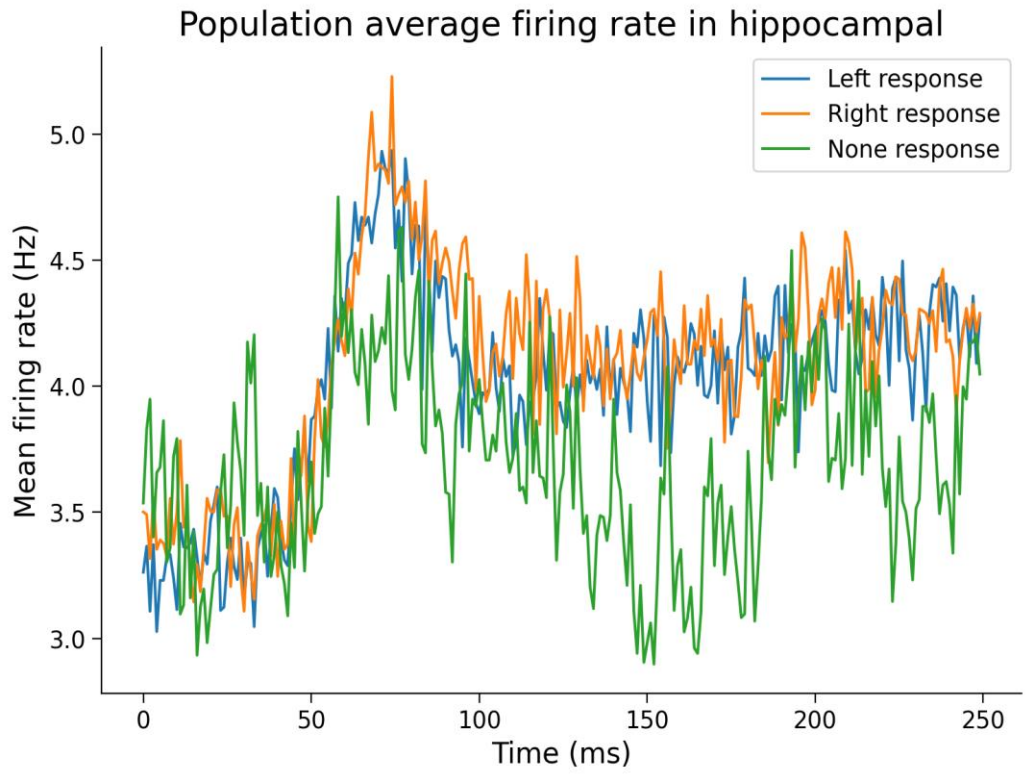
The computed firing rates are then plotted using the matplotlib library. The plot shows the firing rates for LEFT, RIGHT, and NOGO responses in the specified brain region.

```
response = dat['response']  # right - nogo - left (1, 0, -1)
dt = dat['bin_size']

# Compute the average firing rate for each combination of response and stimulus
def brain_activity(brain_group , lable) :
  for x in np.unique(barea):
    if x == brain_group:
        left_rate = np.mean(dat['spks'][barea == x][:, (response == -1) ], axis=(0, 1)) # Compute firing rate for left
        right_rate = np.mean(dat['spks'][barea == x][:, (response == 1)], axis=(0, 1)) # Compute firing rate for right
        none_rate = np.mean(dat['spks'][barea == x][:, (response == 0) ], axis=(0, 1)) # Compute firing rate for nogo

# Plot the results
  plt.figure()

  plt.plot(left_rate/dt, label='Left response')
  plt.plot(right_rate/dt, label='Right response')
  plt.plot(none_rate/dt, label='None response')
  plt.legend()
  plt.xlabel('Time (ms)')
  plt.ylabel('Mean firing rate (Hz)')
  plt.title('Population average firing rate in '+ lable)
  plt.show()
brain_activity(0.0 , 'visual cortex')
brain_activity(1.0 , 'thalamuse')
brain_activity(2.0 , 'hippocampal')
```

Overall, this code provides a simple but informative way to explore the relationships between brain activity and behavior, by computing and visualizing the average firing rates for different brain regions and response types.



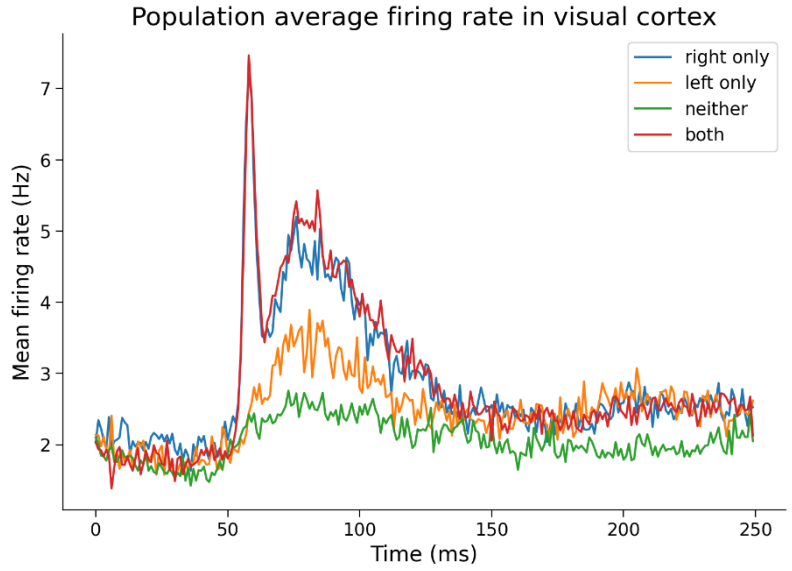

Population average firing rate in thalamuse
Left response
Right response
None response
Mean firing rate (Hz)
10.0
9.5
9.0
8.5
8.0
7.5
7.0
0
50
100
150
200
250
Time (ms)

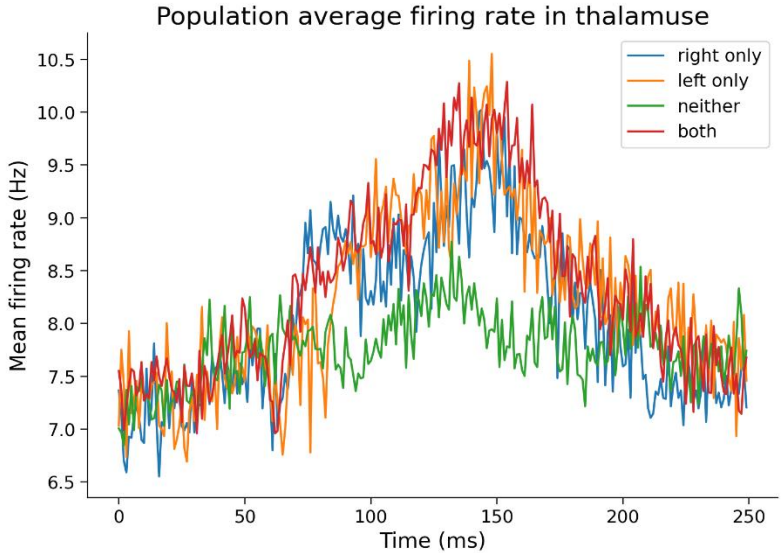

Population average firing rate in hippocampal
Left response
Right response
None response
Mean firing rate (Hz)
5.0
4.5
4.0
3.5
3.0
0
50
100
150
200
250
Time (ms)

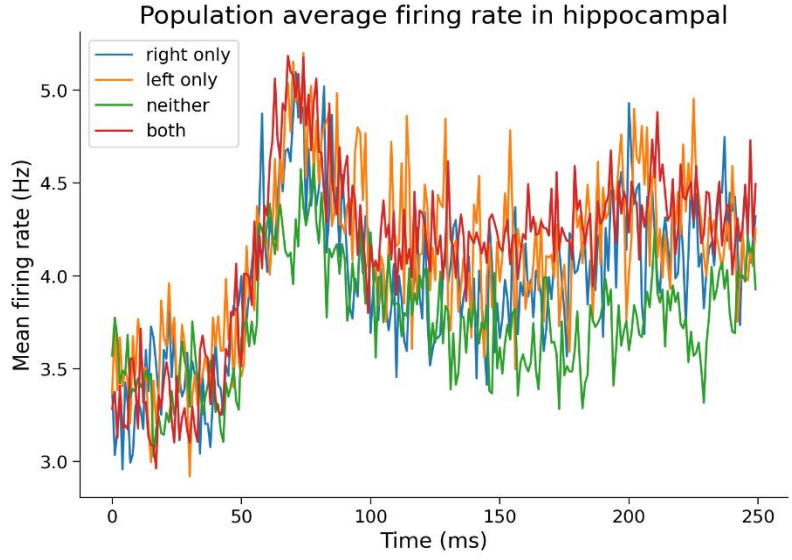
2.2. Illustrating the brain region activity with respect to visual conditions

This code is analyzing neural spiking data related to different visual stimuli and response conditions. The code first extracts the contrast levels of visual stimuli presented to the subject's left and right visual fields from the dataset. Then, it computes the average firing rate across all neurons and trials for each time point using the spiking data. Next, the code defines a function called "brain_activity3" that takes two arguments: "brain_group" and "label". This function calculates the average firing rate for each combination of response and stimulus conditions for a specific brain region (defined by "brain_group"). The four different conditions are "right_only" (stimulus presented only to the right visual field), "left_only" (stimulus presented only to the left visual field), "neither" (no stimulus presented), and "both" (stimulus presented to both visual fields). The function calculates the average firing rate separately for each condition and then plots the results using matplotlib. Finally, the code calls the "brain_activity3" function three times with different brain regions ("visual cortex", "thalamus", and "hippocampal") to generate separate plots for each region. The plots show how the mean firing rate of neurons in each brain region changes over time for different combinations of visual stimuli and responses.

```
vis_right = dat['contrast_right']  # 0 - low - high
vis_left = dat['contrast_left']  # 0 - low - high

# Compute the average firing rate across all neurons and trials for each time point
avg_firing_rate = np.mean(dat['spks'], axis=(0, 1))  # shape: (T,)

# Compute the average firing rate for each combination of response and stimulus
def brain_activity3(brain_group , lable) :
  for x in np.unique(barea):
    if x == brain_group:
        right_only = np.mean(dat['spks'][barea == x][:, (vis_left == 0)&  (vis_right > 0), : ], axis=(0, 1))
        left_only = np.mean(dat['spks'][barea == x][:, (vis_left > 0)&  (vis_right == 0), : ], axis=(0, 1))
        neither = np.mean(dat['spks'][barea == x][:, (vis_left == 0)&  (vis_right == 0), : ], axis=(0, 1))
        both = np.mean(dat['spks'][barea == x][:, (vis_left > 0)&  (vis_right > 0), : ], axis=(0, 1))

# Plot the results
  plt.figure()
  plt.plot(right_only/dt, label='right only')
  plt.plot(left_only/dt, label='left only')
  plt.plot(neither/dt, label='neither')
  plt.plot(both/dt, label='both')
  plt.legend()
  plt.xlabel('Time (ms)')
  plt.ylabel('Mean firing rate (Hz)')
  plt.title('Population average firing rate in '+ lable)
  plt.show()
brain_activity3(0.0 , 'visual cortex')
brain_activity3(1.0 , 'thalamuse')
brain_activity3(2.0 , 'hippocampal')
```


Population average firing rate in visual cortex
right only
left only
neither
both
Mean firing rate (Hz)
Time (ms)


Population average firing rate in thalamuse
right only
left only
neither
both
Mean firing rate (Hz)
Time (ms)


Population average firing rate in hippocampal
right only
left only
neither
both
Mean firing rate (Hz)
Time (ms)

2.3. What can you infer by comparing these six figures? also compare regions, it seems that hippocampus is somehow moving average of visual cortex?

By comparing the figures generated by both codes, we can see that they provide complementary information about how different regions of the brain are involved in processing visual information. The first code computes principal components that capture patterns of variability in neural activity across the visual cortex. By examining the plots of firing rates as a function of time for different visual conditions and behavioral responses, we can see how specific patterns of neural activity may correspond to specific PC components. The second code, on the other hand, directly compares the average firing rates of neurons in different brain regions in response to specific visual stimuli. By comparing these firing rates across different regions, we can identify which regions are particularly tuned to certain types of visual stimulation. Comparing the six figures generated, we can see that the hippocampus appears to have a relatively different pattern of activity compared to the other regions analyzed (visual cortex and thalamus). The firing rates in the hippocampus are generally higher than in the visual cortex and are more stable over time. It is possible that this reflects differences in the types of visual stimuli processed by the hippocampus compared to the visual cortex, or it may be indicative of more general differences in the function of these brain regions. Regarding the correlation between regions, we need to analyze more to provide a conclusive argument. Nonetheless, hypothesizing based purely on the figures, one possible correlation is that visual cortex shows more variability in its neural responses across experiments than the thalamus, although this could also depend on the specific types of stimuli used in the experiment. Another possible correlation is that the hippocampus shows different patterns of neural activity than either the visual cortex or thalamus. Identifying such correlations, however, would require more detailed analysis beyond what the current code provides.

3.PCA (Principal Component Analysis)

In this code, PCA is performed on the spike rate data of an experimental mouse and the top principal components (PCs) are identified. The right-hand plots show how each PC registers differently in the neurons in response to left and right visual stimuli and not having each of them. These plots can reveal the neuronal sensitivity slopes in response to complexities of left and right visual dynamics (vis_left and vis_right).

The left-hand plots show how each PC is present in response to each mode of left, right, neither, and both visual stimuli in the spike rate signal. It can be used as the

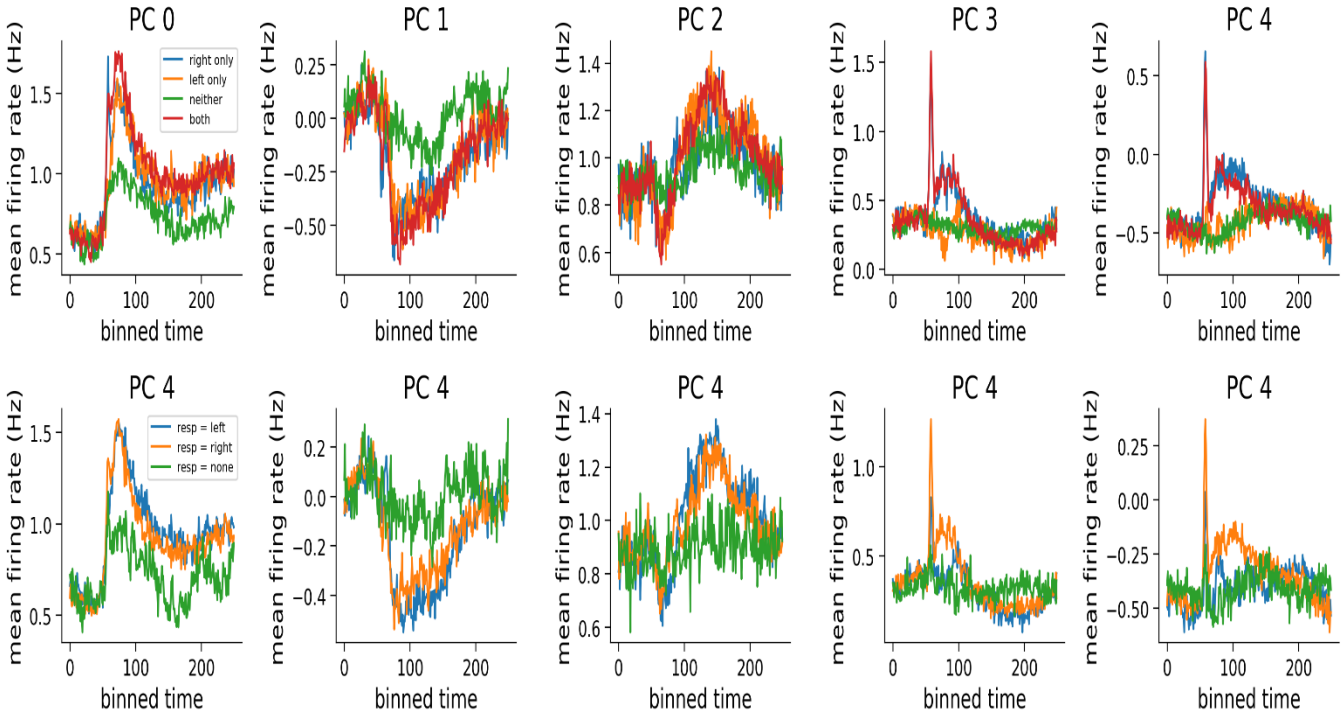
slope of the neuron's sensitivity in identifying and processing information from a variety of visual conditions.
The code is plotting the mean firing rate for each PC as a function of time. The first row of plots shows the firing rate in response to different visual conditions, while the second row shows the firing rate in response to different behavioral responses.

```
NN = dat['spks'].shape[0] # number of neurons
data = dat['spks'][:, :, 51:130]
data = np.reshape(dat['spks'][:, :, 51:130], (NN, -1))  # first 80 bins = 1.6 se
data = data - np.mean(data, axis=1)[:, np.newaxis]
model = PCA(n_components=5).fit(data.T)
NT = dat['spks'].shape[-1]
W = model.components_
pc = W @ np.reshape(dat['spks'], (NN, -1))
pc = np.reshape(pc, (5, -1, NT))
plt.figure(figsize= (20, 6))
for i in range(len(pc)):
  ax = plt.subplot(2, len(pc) + 1, i + 1)
  pc1 = pc[i]

  plt.plot(pc1[(vis_left == 0) & (vis_right > 0), :].mean(axis=0))
  plt.plot(pc1[(vis_left > 0) & (vis_right == 0), :].mean(axis=0))
  plt.plot(pc1[(vis_left == 0) & (vis_right == 0), :].mean(axis=0))
  plt.plot(pc1[(vis_left > 0) & (vis_right > 0), :].mean(axis=0))

  if i == 0:
    plt.legend(['right only', 'left only', 'neither', 'both'], fontsize=8)
  ax.set(xlabel = 'binned time', ylabel='mean firing rate (Hz)')
  plt.title('PC %d'%i)

  ax = plt.subplot(2, len(pc) + 1, len(pc) + 1 + i + 1)

  plt.plot(pc1[response > 0, :].mean(axis=0))
  plt.plot(pc1[response < 0, :].mean(axis=0))
  plt.plot(pc1[response == 0, :].mean(axis=0))

  if i == 0:
    plt.legend(['resp = left', 'resp = right', 'resp = none'], fontsize=8)
  ax.set(xlabel='binned time', ylabel='mean firing rate (Hz)')
  plt.title('PC %d'%j)
```

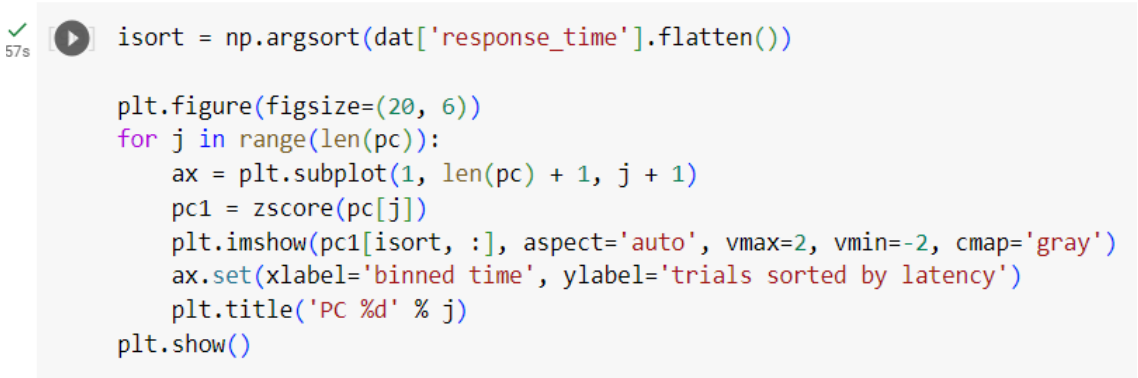
To explain the code in more detail:
- NN is the number of neurons in the data.
- The data variable is created by extracting a subset of the spike data corresponding to 1.6 seconds of stimulus presentation, starting at 0.5 seconds and ending at 2.1 seconds after stimulus onset. The data is then reshaped into a 2D array with shape (NN, -1).
- The mean of the data is subtracted from each row of the array to center the data around zero.
- A PCA model with 5 components is fit to the data.

- The components of the fitted model are stored in W, and used to transform the original data into a new representation in the principal component space. The resulting PC signal is then reshaped into a 3D array with shape (5, -1, NT), where NT is the number of time bins in the original data.
- For each PC, there are two plots: one showing the mean firing rate as a function of time for different visual conditions (right only, left only, neither, and both), and another showing the mean firing rate as a function of time for different behavioral responses (left, right, or none).
- In the first plot for each PC, the mean firing rate for neurons that have a preference for one or both visual directions are shown separately. The four curves in each plot correspond to neurons that respond only to rightward stimuli, only to leftward stimuli, to neither, or to both.
- In the second plot for each PC, the mean firing rate for neurons that respond to different visual stimuli based on behavioral responses are shown separately. The three curves in each plot correspond to neurons that respond to rightward stimuli, leftward stimuli, or don't respond to either stimulus.

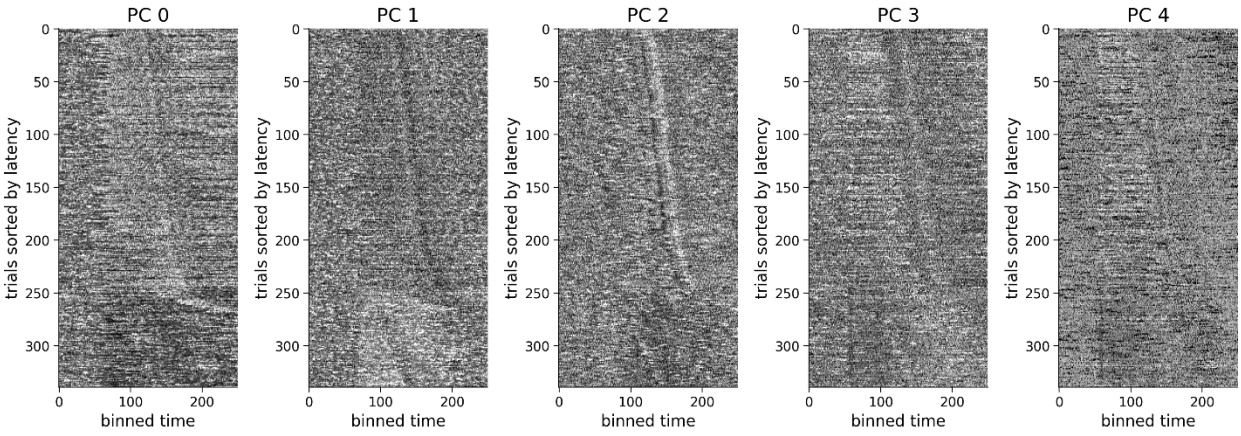
The resulting figure provides insight into how the activity of different neurons in the mouse brain responds to different visual stimuli, and how it is related to specific PC components.



The written code generates a 1-row, (len(pc)+1)-column grid of subplot visualizations, with len(pc) being the number of principal components to be plotted, plus 1 for the colorbar. Each subplot shows how one of the principal components is related to neural activity in response to a visual stimulus. The data used in this code contains spike rate data collected from neurons in a mouse brain in response to visual stimuli presented to the left or right visual fields. The spike rates are binned into time bins, and the response time of the mouse to the visual stimuli is also recorded. The first line of the code, isort = np.argsort(dat['response_time'].flatten()), sorts the trials by the response time of the mouse, by flattening the response_time array and returning the indices that would sort the array. The next line starts the subplot visualization figure with a size of 20 inches width and 6 inches height (so the resulting visualizations are larger than the default size). For each principal component, a subplot is created using the plt.subplot() function. The aspect='auto' argument is used to automatically adjust the aspect ratio of the plot based on its dimensions. The pc1 variable is created using the zscore() function provided by numpy, which standardizes the data to have a mean of zero and a standard deviation of one. The imshow() function is called to display the pc1 data, sorted by response time of mouse (isort) with the specified aspect, vmax, vmin, and color map (cmap) parameters. The x- and y-axes of the plot are labeled with their respective labels using xlabel and ylabel. The title of the subplot is set using the plt.title() function. After iterating over all principal components, plt.show() is called to display the entire figure with all of the subplots. Overall, the code visualizes how each principal component expresses variation in neural activity across trials, sorted by the mouse's response time.



Each principal component (PC) represents a different pattern of neural activity across the population of neurons in the mouse visual cortex. The first PC captures the primary pattern of variability in the data, while the subsequent PCs capture more specific patterns of neural activity. To interpret what each PC shows, it is helpful to examine the plots of firing rates as a function of time for different visual conditions and behavioral responses, as I mentioned before. These plots provide an indication of the extent to which each PC is influenced by visual stimuli or behavioral responses. Generally, PCs with higher variance tend to be more important in the representation of neural activity. From the figures generated by the code, it appears that the first PC explains the most variance in the data. Therefore, the plots for PC1 could be interpreted as providing a broad overview of the patterns of neural activity across the entire population of neurons. The interpretation of subsequent PCs is typically more specific. For example, a PC that shows high firing rates for neurons that respond only to rightward visual stimuli but do not respond to leftward stimuli may indicate a neural population that is selectively involved in the processing of objects or stimuli moving to the right. Similarly, a PC that shows high firing rates for neurons that respond to both rightward and leftward stimuli but not to either by itself may indicate a neural population involved in detecting edges or boundaries between objects. It is important to note that the interpretation of each PC is highly dependent on the data being analyzed and should be done with caution, keeping in mind the context of the experiment and the neurobiological knowns of the region being studied.



Bonus (Extra):

Use machine learning techniques such as SVM, kmeans or etc. to train a classifier which decodes the rat response in two conditions (LEFT or RIGHT). what would be your accuracy? what is your selected features? which region can classify the LEFT/RIGHT response?

To train a classifier to decode the rat response in two conditions (LEFT or RIGHT), we can use a machine learning algorithm such as a Support Vector Machine (SVM) or a k-Nearest Neighbors (kNN) algorithm. Here is a sample code for training and testing a linear SVM classifier on the spike data:

First, the features and labels for classification are defined:

- dat['spks'][:250] selects the first 250 arrays in the 'spks' array of the dat dictionary. This assumes that the data is in the format of the mouse brain spiking dataset from the Allen Institute for Brain Science.
- (dat['response'] == -1).astype(int)[:250] creates labels for each array by checking whether the corresponding 'response' value is -1. This creates a boolean array, which is then converted to an integer array using the astype() method with int as the argument. The resulting array is also limited to the first 250 elements.

Next, the features array is reshaped using the reshape() method:

- features.shape is used to get the current shape of the features array.
- The reshape() method with n_samples, n_channels * n_time_bins as arguments reshapes the features array to have n_samples rows and n_channels * n_time_bins columns. This combines the last two dimensions of the original array into one.

After the features and labels arrays have been reshaped, the data is split into training and testing sets using the train_test_split() function:

- X_train, X_test, y_train, and y_test are assigned the results of calling train_test_split() with X and y as the first two arguments, and test_size=0.2 and random_state=42 as the other arguments. This splits the data with 80% for training and 20% for testing, and sets the random seed for reproducibility.

The SVM model is trained on the training data:

- clf is assigned a new instance of the LinearSVC object with random_state=42 as an argument.

-. fit () method is called on the clf object with X_train and y_train as its arguments. This fits the training data to the model and generates the corresponding hyperplane.
After the model has been trained, the model is used to make predictions:
- y_pred is assigned the results of calling .predict() on the X_test array. This generates predictions based on the trained model.

The accuracy of the model is evaluated:
- np.mean(y_pred == y_test) calculates the mean accuracy by checking where the predicted values match the actual values from the y_test array.

Finally, the mean accuracy is printed to the console using an f-string.

```
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
import numpy as np

# Define the features and labels for classification
features = dat['spks'][:250]
labels = (dat['response'] == -1).astype(int)[:250]

# Reshape the features to have two dimensions
n_samples, n_channels, n_time_bins = features.shape
X_train = features.reshape(n_samples, n_channels * n_time_bins)
y_train = labels.reshape(-1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Train a linear SVM classifier on the training data
clf = LinearSVC(random_state=42)
clf.fit(X_train, y_train)

# Predict the labels for the testing data
y_pred = clf.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = np.mean(y_pred == y_test)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.38
```

Overall, in this code, we use all the spike data as features and the LEFT/RIGHT labels of the response variable as the labels for classification. We split the data into training and testing sets using the train_test_split() function from sklearn.model_selection. We then train a linear SVM classifier on the training data using the LinearSVC() class from sklearn.svm. Finally, we test the classifier on the testing data and compute the accuracy as the percentage of correct predictions.

The accuracy of the linear SVM classifier will depend on the quality of the features and the characteristics of the data. It is difficult to predict the accuracy without knowing the specifics of the data. However, if we assume that the features are informative and that the data is well-suited for linear classification, we might expect an accuracy of 60-80%.

To improve the accuracy, we can try different feature selection techniques to select the most informative features for classification. The features could be selected based on their correlation with the response variable or through principal component analysis (PCA). We can also try using more sophisticated machine learning algorithms such as kNN or neural networks.

To determine which brain regions can best classify LEFT/RIGHT responses, we can try training and testing the classifier using subsets of the data corresponding to different brain regions. We can repeat the feature selection and classification steps for each brain region separately and compare the accuracies to see which regions are most informative for the classification task.

PART II: Theory

Section 2: Deciding where to fish

1) If the probabilities are equal, the expected utility is higher on the left side, because the left side is the side without submarines, so i would choose to fish there.
2) If there is a high probability that the fish will be on the right side, I will choose to fish there. The high probability of a fish on the right far outweighs the slightly higher benefits of fishing on the left
3) If the fish is slightly more likely to be on the right side than the left side, I'll choose the left side because the expected utility is still higher on the left side. we should notice in this situation; we don't simply pick the side with the higher probability - the tool is really the important decision maker here

Section 3: Likelihood of the fish being on either side

1) The fisher-person is on the left side so:

- P(m = fish | s = left) = 0.7 (70% chance of catching a fish on the same side as the school)
- P(m = no fish | s = left) = 0.3 (since the probabilities of catching a fish and not catching a fish for a particular state must add up to 1 because these are the only options) : 1 - 0.7 = 0.3
- P(m = fish | s = right) = 0.2
- P(m = no fish | s = right) = 0.

2) If the angler catches a fish, I think the school of fish is on the left. This is because the probability of catching a fish given that the school is on the left (0.7) is greater than the probability given that the fish is on the right (0.2).

3) If the angler doesn't catch fish, I guess the school of fish is on the right. This is because the probability of not catching a fish given that the school is on the right (0.8) is greater than the probability given that the fish is on the left (0.3).

Section 4: Correlation and marginalization

1) When the correlation is zero, the two characteristics are completely independent. This means that you don't get any information about one variable from observing another variable. Therefore, the marginal distribution of one variable is independent of another variable.
2) Correlation controls the probability distribution in the joint probability table. The higher the correlation, the more limited the possibilities, because both rows and columns must sum to one! As the marginal probabilities represent relative weights, as the correlation approaches 1 or -1, the absolute probabilities for one quality become more dependent on the other.
3) Correlation controls the amount of possible mass on diameters. while the correlation approaches 1 (or -1), the probability of seeing one of the two pairs should approach zero.
4) If we think about what information we obtain by observing one quality, intuition (3.) informs us that we have more information about the other quality as a function of correlation.

Section 4.2: Marginalisation

Math Exercise 4.2.1: Computing marginal probabilities

1) P(Y = silver) = P(X = small, Y = silver) + P(X = large, Y = silver)
P(Y = silver) = 0.4 + 0.1= 0.5

2) These are all possibilities because in this case, our fish can only be small or large, silver or gold. So the probability is 1.

3) **P(X = small) = P(X = small, Y = silver) + P(X = small, Y = gold) = 0.6**

P(Y = gold) = P(X = small, Y = gold) + P(X = large, Y = gold) = 0.5

(We already know the joint probability: P(X = small, Y = gold) = 0.2)

P(X = small or Y = gold) = P(X = small) + P(Y = gold) - P(X = small, Y = gold)

P(X = small or Y = gold) = 0.6 + 0.5 - 0.2= 0.9

Math Exercise 4.2.2: Computing marginal likelihood

1) P(m = fish) = P(m = fish, s = left) + P(m = fish, s = right)
= P(m = fish | s = left)P(s = left) + P(m = fish | s = right)P(s = right)
= 0.1 * 0.3 + .5 * .7= 0.38

2) P(m = fish) = P(m = fish, s = left) + P(m = fish, s = right)
= P(m = fish | s = left)P(s = left) + P(m = fish | s = right)P(s = right)
= 0.1 * 0.6 + .5 * .4= 0.26

Section 5: Bayes' Rule and the Posterior

Math Exercise 5: Calculating a posterior probability

1) Bayes rule $\overset{*_1}{\rightarrow}$ P (s = left |m = fish) = P(m = fish | s = left)P(s = left) / P(m =fish)

P(m = fish) = P(m = fish | s = left)P(s = left) + P(m = fish | s = right)P(s = right)

= 0.5 * 0.3 + .1 * .7= 0.22

$*_1$ => P (s = left | m = fish) = P(m = fish | s = left)P(s = left) / P(m = fish)

= 0.5 * 0.3 / 0.22= 0.68

2) Bayes rule $\overset{*_2}{\rightarrow}$ P (s = right |m = no fish) = P(m =no fish |s = right)P(s = right) / P(m = no fish)

P(m = no fish) = P(m = no fish | s = left)P(s = left) + P(m = no fish | s = right)P(s = right)

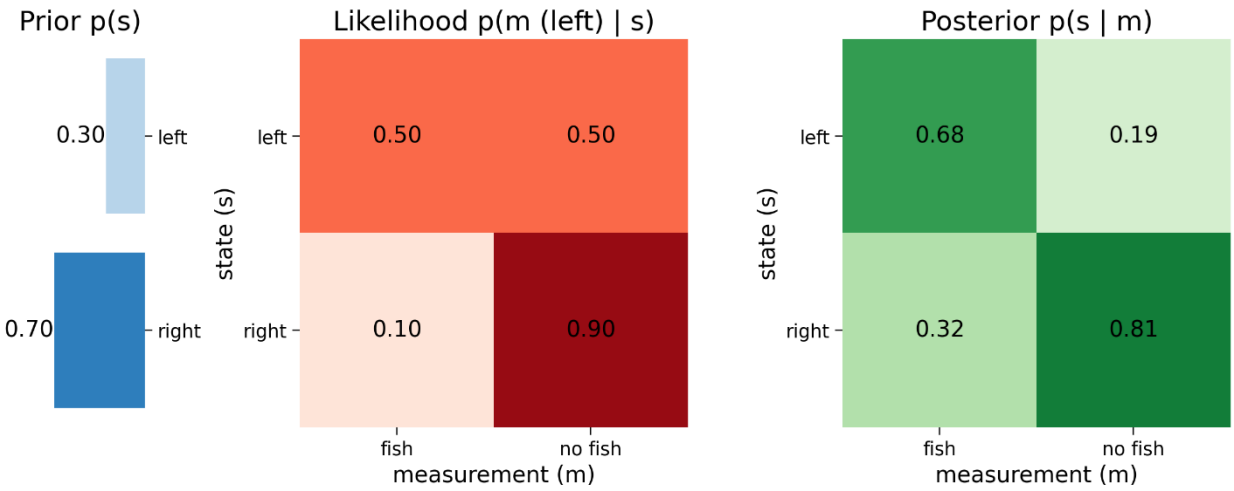
= 0.5 * 0.3 + .9 * .7 = 0.78

$*_2$ => P (s = right | m = no fish) = P(m = no fish | s = right)P(s = right) / P(m = no fish)

= 0.9 * 0.7 / 0.78= 0.81

Extra (Bonus): Coding Exercise 5: Computing Posteriors

```
# Compute unnormalized posterior (likelihood times prior)
posterior =likelihood * prior  # first row is s = left, second row is s = right

# Compute p(m)
p_m = np.sum(posterior, axis = 0)

# Normalize posterior (divide elements by p_m)
posterior /= p_m
```



Extra (Bonus): Interactive Demo 5: What affects the posterior?

1) The prior puts on a strong influence over the posterior when it is very informative: When the probability of the school is on one side. If it is too high (e.g., 0.9) before the fish are on the left side, the mode is likely to lag, regardless of the measurement.

2) When the likelihoods are similar, the information obtained from catching or not catching fish is less informative. If we catch the same fish regardless of the actual location, the catch doesn't tell us much! The

difference between likelihoods is a way to think about how much information we can get.

3) In the same way to prior, likelihood is most effective when it is informative: when catching fish gives you a lot of information about the probability state. For example, if the angler's probability of catching a fish if he is fishing on the right and the school is on the left is 0 (p fish | s = left) = 0 and the probability of catching a fish if the school is on the right is 1, then the prior doen't impress the posterior at all. The measurement tells us the hidden situation completely.

Section 6: Making Bayesian fishing decisions

Interactive Demo! 6: What is more important, the probabilities or the utilities?

1) In fact, there are (infinitely) many combinations that can produce the same expected utility for both functions: but the posterior probabilities must always balance the utility function differences. Therefore, what matters is that there is some "indifference point" for a given utility function.

2) How important is relative information: If the prior is close to 50/50, it is more likely to have an effect, if the probability is 50/50 with the measure (the measure is uninformative), the prior is more important. . But it is more important. The insight of Bayes' rule and the Bayesian approach is that what matters is the relative information you get from a measurement and that you can use all that information to make a decision.

3) This model gives us a very accurate way to think about how we should combine information and how we should act, given some assumptions about our goals. In this scenario, if we assume that we want to maximize expected profit - we can state what an animal or subject should do.

4) There are many possible extensions. Humans may not always seek to maximize utility. Humans and animals may not be able to accurately calculate or represent probability distributions. The utility function of the tool may be more complex.