Learning objectives and goals:

In the first half of this problem set, we will build a perceptron neural network that performs binary classification. We will derive the weight vector for the perceptron in two ways: via analytical work, and by performing numerical simulations in MATLAB that implement the perceptron learning and weights update rule.

In the second half of this problem, we will use Principal Component Analysis to reduce dimensionality of a relatively large neuronal data set. We will use this dimensionality reduction to generate meaningful visualization of the data set, and also to build a crude and rudimentary decoder for neuronal activity

The topics of this PSET were covered in lectures 14-17 and recitations 10-11.

These two problems are a great way to put all the linear algebra we have learned so far to work. They are also helpful in emphasizing the two roles that matrices play: to represent and to transform data. By the end of this PSET, you should be familiar with:

- Dot products, unit vectors, matrix multiplication and change of basis.
- Performing linear transformations such as rotations and scaling.
- Implementing the perceptron algorithm in MATLAB
- Perform mean subtraction, construct covariance matrices and compute eigenvectors and eigenvalues in MATLAB
- Use Principal Component Analysis to find the directions of maximum variance of a given dataset.
- Use Principal Component Analysis to perform a change of basis and generate novel data visualizations.

MATLAB functions you will need:

The following functions will be helpful:

`repmat`	Replicate and tile array (repeat copies of an array)
`eig`	Compute the eigenvectors and eigenvalues of a matrix
`cumsum`	Cumulative sum of array elements.

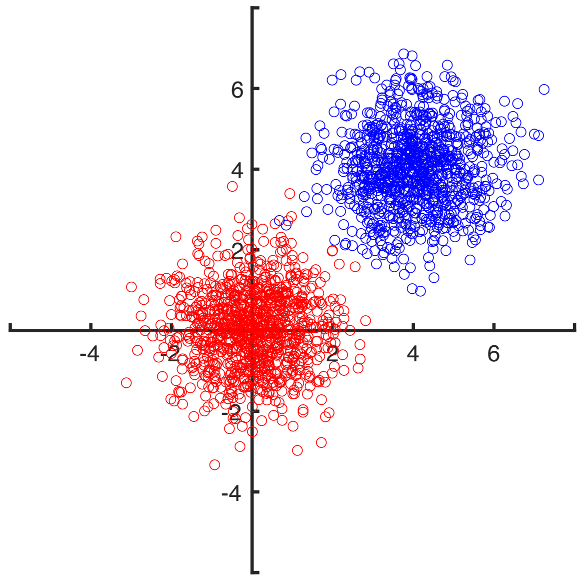
Problem 1: Neural Networks and Linear Algebra

We will use this exercise to start gaining insight on how feed-forward neural networks transform input data. In class, we discussed modeling a network of neurons where each input neuron has a firing rate and specified synaptic weights onto a set of output neurons. Such a network can perform matrix multiplications. To get us warmed up we will start with a simple question.

1. **Construct a feed-forward neural network with 2 input neurons and 2 output neurons that performs a 60 degrees clockwise rotation in 2D space.**

Problem 2: The Perceptron Learning Algorithm

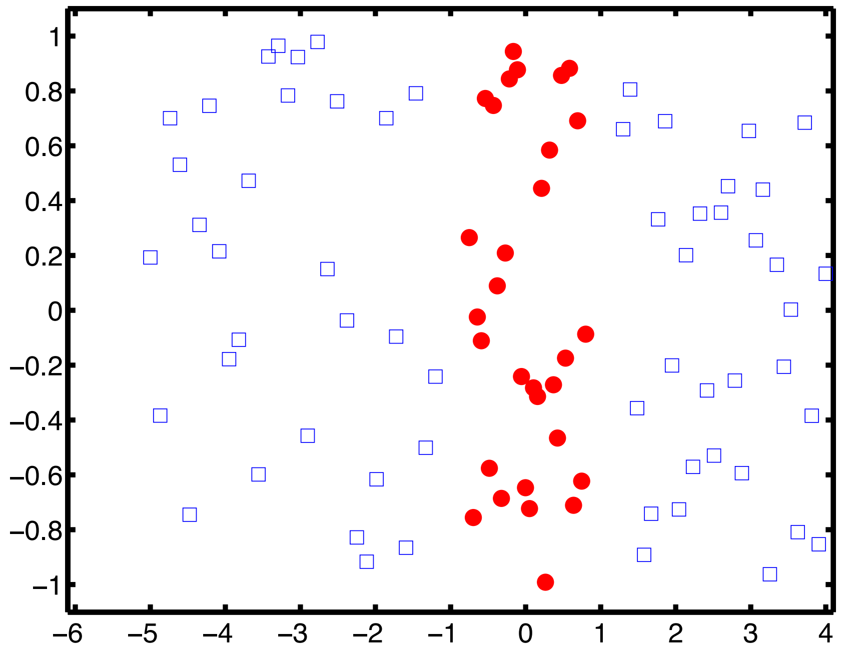
In this exercise, you will build neural networks that not transform their inputs, but that are capable of performing binary classification. To illustrate the principle behind the perceptron learning algorithm, you will construct a neural network that classifies stimuli into two categories. The perceptron receives two inputs, which represent two stimulus features. The stimuli are plotted below in this 2D feature-space, colored according to their category. Your perceptron should fire for stimuli in the blue category and not for stimuli in the red category.



1. **On the above plot, draw a decision boundary and weight vector that would distinguish the two categories.**
2. **Draw a diagram of your perceptron and its inputs. Find the weights and label your diagram accordingly. Use binary threshold units as described in class, with outputs 0 or 1 and a threshold θ=1.**
3. **In this case, you were able to compute the weights by hand. But in higher dimensions, with more complicated inputs, that can be impossible. We will now implement the perceptron-learning algorithm, which automatically determines the weights by starting from a random guess and updating that guess whenever it makes an error.**
    - **The file `perceptron.m` implements the perceptron-learning algorithm in MATLAB for the example above. The blue category has label 1 and the red category has label 0. The main for-loop is incomplete. Your job is to complete it, with help from the comments in the code.**
    - **Once completed, the file plots the original points and the final decision boundary.**
4. **Include the plot generated by `perceptron.m` in your write-up. How do the automatically determined weights compare to the weights you derived by hand?**

Problem 3: Combining Multiple Perceptrons

The following figure illustrates a binary classification problem that cannot be solved with a single perceptron.



1. **Why is this problem not solvable by a single perceptron?**
2. **Find, draw, and label properly a multilayer perceptron that correctly classifies the dots and the squares. Use binary threshold units as described in class, with outputs 0 or 1 and a threshold θ=1.**

Problem 4: Reducing Dimensionality of Motor Cortex Activity During a Reaching Task

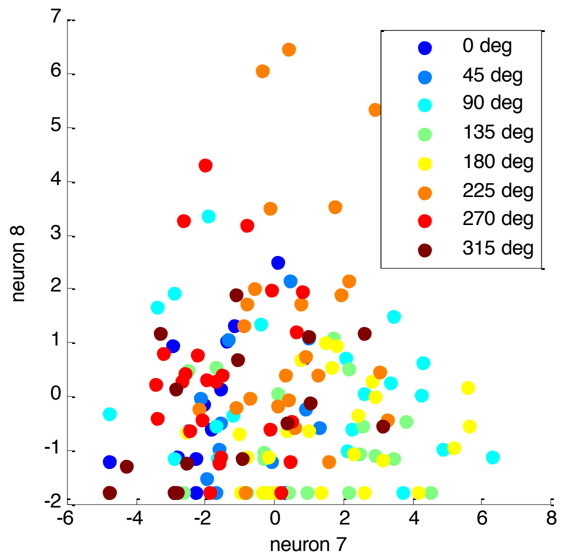
In this problem, we will use PCA to visualize the activity of a large population of neurons recorded in macaque motor cortex during a center-out reaching task. Based on the results of the PCA, we will design an algorithm that could control a simple prosthetic arm to move according to the monkey's motor cortex commands. The data were collected in Nicho Hatsopolous's lab at University of Chicago, and are contained in the file `ReachData.mat`.

The matrix `R` contains the firing rate of each of 143 neurons for a total of 158 trials. The vector `direction` specifies the reach direction for each of the 158 trials. Direction 1 corresponds to a rightward reach (0 degrees), direction 2 to 45 degrees, direction 3 to 90 degrees (straight up), etc, with direction 8 corresponding to 315 degrees. Think about the orientation of the matrix `R` before you proceed. If needed, transpose `R` to flip the rows and columns.

1. **Center the data from each neuron by subtracting off the average firing rate of that neuron, using `repmat` as described in class. Why do we do this before performing**

PCA? Hint: think about, in 2D, what goes wrong if you perform PCA on a cloud of data that is not centered at 0.

2. **Compute the covariance matrix that we will use to determine the principal neuronal components. What size is this matrix? What does the ij-th entry of this matrix represent?**
3. **Use the MATLAB `eig` command to compute the eigenvectors and eigenvalues of the covariance matrix. What do the eigenvectors represent?**
4. **In two subplots, plot the eigenvalues in descending order, and the percentage of variance explained (`cumsum` may be helpful). How much variance is explained by the first two principal components?**
5. **We will now visualize the neuronal data in two dimensions to observe how the population of neurons in motor cortex fires differently when the monkey reaches in different directions.**
   - **Write MATLAB code to plot two dimensions of the mean-subtracted data. Each point on your plot should represent a trial, with a different color for each reach direction. This will help you visualize how well the neuronal activity clusters according to reach direction. Later, you will use this code to plot the first two principal components of your data. To debug, check that your plot of responses from neurons 7 and 8 matches the figure below (note a lack of clear clustering). The MATLAB commands `jet` and `legend` may be helpful. Note that you can set the color in a plot as follows: `plot(x,y,'.', 'Color', [1 0 0]),` where [1 0 0] refers to the RGB channels. Also you can see a list of possible properties to set by (for example): `set(plot(x,y,'.')`**

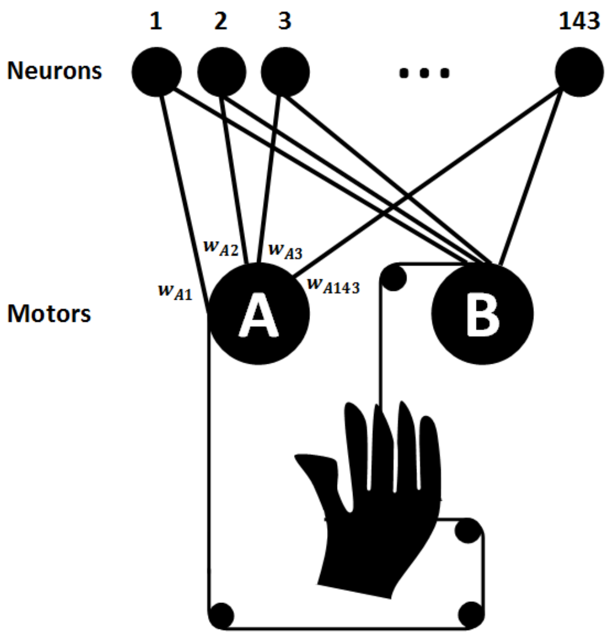


   - **Rotate the mean-subtracted data into the eigenvector basis. (Hint: this is just matrix multiplication). The first two elements of each vector in this rotated basis correspond to the projection of the data onto the first two eigenvectors. This is called the first two principal components.**

6. **Plot the first two principal components of your data. Each point should correspond to a different trial and should be colored according to the reach direction for that trial. What do you notice?**

Problem 5: Principal Components to Decode Neuronal Activity

Now we will consider a simple way of decoding the neuronal activity in motor cortex to move an artificial hand. Suppose we have a simple robotic hand that we want the monkey to control. The simple hand has two motors: positive voltage to motor A makes the hand move right, and positive voltage to motor B makes the hand move up (negative voltage to either motor makes the hand move in the opposite direction as positive voltage does). For each motor, we assign each neuron a weight, which determines how much the voltage to that motor is influenced by the activity of that neuron. Specifically, $V_A = \sum_i w_{Ai} \cdot r_i$, where V_A is the voltage to motor A, w_{Ai} is the weight from neuron i to motor A, and r_i is the mean-subtracted firing rate of neuron (See the diagram below).



1. **Suppose we connected each neuron to motor A with the weights specified by the first principal component. What would happen when the monkey tried to reach up (90 degrees reach direction)?**
2. **Suppose we also connected each neuron to motor B with the weights specified by the second component. What would happen now when the monkey tried to reach up?**
3. **What linear algebra technique could we use to correct the weights so that the artificial hand moves in the correct direction?**