**Problem set 1: RBFs and classifier networks**

1. **Function approximation using radial basis functions**

Your job is to simulate the basic results of the Ghahramani et al paper. Well, at least to use radial basis functions (RBF) to approximate a linear mapping, then examine the consequences of a local shift in the mapping.

* 1. Create a data set for a one dimensional linear mapping corrupted by Gaussian noise (i.e. a mapping between visual and proprioceptive sensory input). In particular, create: **y** = 2\***x** + **e**; where **x** is drawn from a uniform random distribution (using unifrnd) between -10 and 10 with 1000 samples, and **e** is a normally distributed noise vector with mean zero and standard deviation of 1 (use normrnd). The code for this is supplied in ‘ps1 data sets.m’.
  2. Create a set of RBFs. Place their centers at -12 to 12 at every .5 along the x axis. Set the standard deviation of each RBF to 1. There should be 48 RBFs (give or take, depending on how exactly you place them).
  3. Find the (~48 dimensional) weight vector **w** giving the weight for each RBF to best approximate **y** given **x**. You can use either gradient descent or linear regression to find the vector **w**.
  4. Make a plot with the original **x** vs **y** data. Superimpose on this plot the values of **y** predicted by your RBF network.
  5. Add 50 data points to your x vector, all at x = 6. Add a corresponding 50 points to your y vector, according to: y = 2\*x + 10 + e. Your data set should now have 1050 data points in it (1050 pairs of x and y). These new data are meant to mimic the perturbation in the visual/proprioception map as introduced in the paper; i.e. at a specific location (x=6) shift the information the person gets about the output (y = 22 vs. the original y = 12).
  6. Find a new **w** vector for this new data set (using gradient descent or regression). Plot the original **x** vs **y** data, along with the superimposed new predicted **y** values.
  7. How do the results you see here recapitulate the experimental results of the Ghahramani et al paper? How does the generalization you see in your simulation reflect the semi-parametric network that you used?

1. **Linear classification**
   1. **Linear classification, 2 classes**

Use the code in ‘ps1 datasets’ (the code in the first cell block of code) to generate data according to two 2D Gaussians. Your job is to create a classifier which produces the correct class labels for each data point. You’ve been provided with the ‘correct’ class labels in the variable **y** in the mfile. Use a linear classifier/perceptron: **y\_hat** = sign(**w’x**). Note that the offset should be included in the **x** data vector (i.e. add another column of 1’s to your data, as for regression). Use gradient descent on the squared error to find the best fit weights for this classification. Plot the identified weight vector on the same plot as the raw data, incorporating the offset when you plot the vector (i.e. it shouldn’t go through zero). Interpret the result of your classifier according to the position of this weight vector.

* 1. **Linear classification, 2 classes, sigmoidal output**

Use the code in ‘ps1 datasets’ (2nd cell) to generate the data for this problem. Note that the class labels are slightly different here (0 and 1’s) than the last problem. Here your classifier will act according to: y\_hat = sigmoid(wx). Make the same plots as in (a). In addition, plot the output of the sigmoid (i.e. y\_hat) and interpret its values.

* 1. **Linear classification, 3 classes, sigmoidal outputs**

Use the code in ‘ps1 3classes sigmoid lda’ (3rd cell) to generate the data. Here there are three classes of data. There are three output units for the network – one for each class. When a data point is generated from the first class, the output is [1 0 0]; when the point is generated from the second class, the output is [0 1 0]; from the third class, the output is [0 0 1]. This is all in the variable y in the code. Your network should act according to **y\_hat** = sigmoid(**Wx**), where y is now a vector and W is a 3 by 3 matrix (3 dimensional output and 3 dimensional input). The predicted class for each data point is the dimension of y\_hat which is largest (look at the max function in Matlab). Use gradient descent to find W. Plot the classification decision of your network for each data point.