Problem set 3: 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.

   a. 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).
   b. 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.
   c. Use linear regression to find W for the weighting of each RBF and show the predicted values of y for each x value. Make a plot with the original x vs y data and the x vs predicted y superimposed.
   d. 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. (the x used here should correspond to the newly added data points) I.e. displace some of the y values at a particular value of x – as in the paper.
   e. Find a new W vector for this new data set and show the predicted values superimposed on the actual data. Where does the generalization take place? Feel free to repeat with different RBF centers and standard deviations to examine how the generalization changes.

2. **Linear classification**

   a. **Linear classification, 2 classes**

      Use the code in 'ps1 datasets' (1st cell) 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(wx). 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 LMS 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. Also plot the separation boundary for this classifier (i.e. the line orthogonal to the weight vector).

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

      Use the code in 'ps1 datasets' (2nd cell) to generate the data. Note that the class labels are slightly different here (0 and 1's). 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.

   c. **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]; 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.

3. Classifier – back propagation

I've posted code (in ps1 datasets.m, 4th cell) to create three data clusters which are vertically stacked and therefore not linearly separable, as we discussed in class. Your job is to create code implementing back propagation for a two layer neural network which can perform this classification. Use a network with 4 hidden units, as indicated in the shell code. Don't worry about cross validation and all that – feel free to just use all the data.