CSI 5325 Assignment 5

Greg Hamerly

Assigned: 4/03/2018; Due: 4/24/2018

Instructions

The instructions for this assignment are the same as for all assignments in this course; for details refer to Assignment 1. As a refresher: use LaTeX, make your document beautiful (proofread it!), use well-labeled figures to illustrate things, turn in a hardcopy and an email copy, and keep attachments small.

1. Experiments with k-means (20 points)

Implement the k-means algorithm, preferably in a high-level language like Matlab. Use the symmetry-and-intensity dataset `http://amlbook.com/support.html` Note that this is a labeled dataset (appropriate for use in supervised learning) but we are using it for unsupervised learning here.

You should initialize your algorithm using a random k-sample of the points, or (if you're feeling adventurous) using k-means++ initialization (look it up).

Make sure that you write a good visualization subroutine so you can see what happens as the algorithm proceeds. In Matlab, something like the code in Figure 1 may help.

Try the following experiments:

1. For k values of $\{2, 5, 10, 20\}$, describe with appropriate statistics (e.g. counts and / or averages) how many points within each cluster have the same or different (supervised) class labels. Is unsupervised learning able to discriminate supervised labels well?
2. For $k = 5$, try 100 different random restarts of the algorithm (with a different initialization each time). Describe the distribution over these restarts of:
   - the number of iterations,
   - the cluster size distribution (e.g. largest - smallest),
   - the k-means quality metric (sum of squared distances),
   - if you ever observe the algorithm getting 'stuck' (e.g. empty clusters, or a really poor solution).

2. Feedforward neural networks (20 points)

1. Derive the gradient for feedforward neural network training using backpropagation. In particular, assume tanh activation and squared error (on the output).

```
% let "cluster" be the vector of cluster assignments for the N points
% let "x" be the N-by-d matrix of examples

clf; % clear the figure
hold on; % don't erase with each plot command

colors = 'krgbmc'; % plotting colors

for c = 1:k
    assigned = find(cluster == c);

    % plot all the points for this cluster as dots in the same color
    color = colors(mod(c, length(colors)) + 1)
    plot(x(assigned,1), x(assigned,2), [color, '.']);

    % plot this centroid with a big circle
    centroid = mean(x(assigned,:), 1);
    plot(centroid(1), centroid(2), [color, 'o'], 'markersize', 10);
end
```

Figure 1: Skeleton code for plotting

What you want to find is

$$\frac{\partial e}{\partial w_{ij}^{(\ell)}}$$

for output error e (on a particular example, say) and a particular weight w_{ij}^{ℓ} (from node i in layer $\ell - 1$ to node j in layer ℓ). Use the notation in your book (e.g. s^{ℓ}, x^{ℓ}, etc.) as a standard to follow.

Recall the chain rule which is used in several places, i.e.

$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c}\frac{\partial c}{\partial b}.$$

which allows you to express the derivative of a with respect to b by using an intermediate element c.

2. Using the derivation you just did, implement a 2-layer network (one input layer, one hidden layer, one output layer) and train it on the same symmetry-and-intensity dataset you used previously. There will be 3 inputs: symmetry, intensity, and bias (constant 1). Try 2, 5, 10, and 20 hidden layer nodes. Don't forget the bias inputs for the hidden and output nodes. As this is classification, there will be one output node, and the activation function for all non-input nodes will be the tanh function.

   Use either SGD or batch gradient descent training, and use a hold-out set to measure validation error.

   Examine how well the network does (training and validation error) as a function of the (a) number of hidden layer nodes and (b) number of epochs. It would also be instructive to make a 2-d plot of the network's classification of the 2-d input space (into +1/-1 regions) for different networks.