9.641 Neural Networks
Problem Set 10

(Due before class on Thursday, May 5th)

1. Backpropagation Algorithm

   In this problem, we will re-investigate the two pattern matching problems from the previous homework, *i.e.* face categorization and digit recognition.

   The sample code for the last problem set (perceptron_delta.m) pulled t_{max} images at random from the training set, and applied an online update after each image. For this problem set, we will measure the error as a function of *epoch* number, where an epoch is defined as a single pass through the entire data set (each image is evaluated once).

   Write your code so that in each epoch you cycle through every image in the training set (making an online update after each) *and* through every image in the test set (no updates, just record the error).

   We will look at two different error metrics for this problem: the epoch squared error and the epoch classification error. The squared error for an epoch is simply the sum of the squared errors for each image in the data set. This is precisely the function that is minimized in the gradient descent calculation. We can also think of the network as solving a classification problem. For each input, the image is classified according to the output neuron that is maximally active. The classification error is the percentage of images which the network classifies correctly.

   (a) For the multi-layer perceptron, experiment with different sizes n of the hidden layer, initial conditions for the weights and biases, and learning rate. Settle on particular choices of these parameters, and run the backprop algorithm until convergence. As your final results, submit the following (*for both datasets*):

      i. MATLAB codes

      ii. first layer of synaptic weights of converged network, shown as images

      iii. "confusion matrix" of converged network (C_{ij} is defined as the number of times class j is classified as class i). The confusion matrix should be 2×2 for the faces (binary classification problem) and 10×10 for the digits.

      iv. learning curves of training squared error, test squared error, training classification error, and test classification error as function of epoch number.

For your best parameters how does the squared error evolve over time for the training set? What about bad parameters? What about the test set? How do the two curves compare?

What about the classification error? Is the squared error a good predictor of the classification error?

(b) Run a single-layer perceptron (modify the perceptron_delta.m code from last time) on the two datasets and submit the items (i) through (iiii). How does the multi-layer perceptron compares with the single layer?

2. Limit Cycle Learning.

The classical test problem for trajectory learning in neural networks is the circle problem. For this problem we will use a circular desired trajectory $\mathbf{d}(t)$ with a center at $[0.5, 0.5]$ and radius 0.25, and neurons of the form:

$$\dot{\mathbf{x}} + \mathbf{x} = f(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where $f = \frac{1}{1+e^{-x}}$. The desired trajectory should make at least two full rotations around the origin.

- Consider the Euler discretization of $\dot{x} + x = f(Wx + b)$,

$$\frac{x(t) - x(t-1)}{dt} + x(t-1) = f(Wx(t-1) + b) \tag{1}$$

starting at the initial condition $x(0)$ and continuing until $x(T)$. Show that

$$\Delta W = \eta \sum_{t=1}^{T} s(t)x(t-1)^T \tag{2}$$

is a gradient update, where s is defined by the final condition $s(T+1) = 0$ and running the dynamics

$$\frac{s(t) - s(t+1)}{dt} + s(t+1) = D(t)W^T s(t+1) + D(t)\frac{\partial R}{\partial x(t)} \tag{3}$$

backwards in time until $t = 1$. The matrix $D(t) = diag\{f'(Wx(t-1) + b)\}$ is diagonal. The continuous time limit of this equation is

$$-\dot{s} + s = D(t)W^T s + D(t)\frac{\partial R}{\partial x(t)} \tag{4}$$

We will do that in 2 steps:

(a) To derive ΔW and Δb we first need to compute $\frac{\partial R}{\partial W_{ij}}$ and $\frac{\partial R}{\partial b_j}$. This is difficult because R is an implicit function of W_{ij} and b.

The good news is that you do not need to compute both, because the sensitivity lemma tells you than

$$\frac{\partial R}{\partial W_{ij}} = \sum_t \frac{\partial R}{\partial b_i(t)} x_j(t-1)$$

So we just have to compute $\frac{\partial R}{\partial b_i(t)}$. The second good news is that there exists a simple coordinate transform that exists between $\frac{\partial R}{\partial b_i}$ and $\frac{\partial R}{\partial x_j(t)}$, which is very easy to compute (R is an explicit function of $x_j(t)$). This change of coordinates can be found by applying the chain rule:

$$\frac{\partial R}{\partial x_j(t)} = \sum_{it_1} \frac{\partial R}{\partial b_i(t_1)} \frac{\partial b_i(t_1)}{\partial x_j(t)}$$

Denote $s_i(t) = \frac{\partial R}{\partial b_i(t)}$ and compute $\frac{\partial R}{\partial x_j(t)}$ as a function of $D^{-1}(t)$, W_{ij}, and δ_{tt_1} where δ_{tt_1} is the kronecker function *i.e.* $\delta_{tt_1} = 1$ if $t = t_1$ and 0 else.

(b) From there derive Eq. 3 and ΔW.

- Train a fully-connected recurrent network with 2 visible and 3 hidden units (this means that the network contrains 5 units total) using the backpropagation-through-time algorithm.

  Hint: Verify that your W is 5×5. Your program should contain an outer loop (2000 epochs) and 2 inner loops: One for the forward pass where you compute and store the $x(t)$ and the $D(t)$ for all T time steps. You should have a second inner loop for the backward pass where you compute the $y(t)$ also for all T time steps. $D(t)$, $y(t)$ and $x(t)$ should all be $5 \times T$. After the forward and backward pass, you have all you need to update W and b. If your training is getting stuck in a local minima, try training on a small fraction of the trajectory first. Submit your MATLAB code, training error plots, and a state-space plot of the actual trajectory superimposed on the desired trajectory. Describe in words the function of the hidden units in the trained networks.