Homework 6; Psych 186B, Winter 2018

Marshall Briggs

26 February 2017

Code explanation

I structured my code as per the following:

MAIN *script* parity.m - This script defines the general workflow for the back propagation exercise:

- Define crucial parameters,
- build the input patterns,
- build the expected output values,
- modify the connection weights such that the model enforces a strong connection between input and output,
  - Within this part of the script, I embedded a while loop to check of the SSE of the model output had converged to < 0.01. If it had not, I would re-execute this step.
- finally, report the output errors before and after and report the network's progress.

1. *function* converge_weights() - This function can be seen as a helper function for the parity script. It takes in its crucial parameters and starting input, output, and connection weights, and returns the final connection weights, the final sse, and some report statistics (the epoch number of each report -¿ the sse of the output at that epoch).

2. *function* epoch() - This function abstracts the contents of each epoch into a single module, taking in the input patterns, the desired output values, the initial connection weights, and some crucial parameters and returning the final connection weights and the final sse after a single epoch. This abstraction exists so that the previous function can play with an arbitrary number of epochs.

3. *function* generate_input() - This function takes in a number of patterns and a number of inputs, and returns a [num_patterns x num_inputs] array of uniformly distributed 0's and 1's.

4. *function* generate_output() - This function accepts a number of patterns, a number of inputs, and an input array generated by the above function and generates an output array corresponding to the input array s.t. Output(i) = the number of 1's in row i of Input.

5. *function* test_model() - This function accepts Input and Output vectors generated by the above functions and a network model represented by two arrays of connection weights and returns (i). The output of the network on the given inputs and (ii). the difference between the desired output and the actual output.

6. *function* activation_fn() - This function acts as the non-linear connection unit for this network model (a monotonically increasing, differentiable, bounded between (-1, 1) sigmoid function).

7. *function* activation_fn_deriv() - This function acts as the derivative of the non-linear sigmoid function.

Figure 1.ii plots magnitude of the output errors for each input pattern (i.) before back propagation (yellow), (ii.) after back propagation on untrained input (orange), and (iii.) after back propagation on trained input (blue). This comparison seems to indicate that, despite how well the model generalizes the trained input (mean error magnitude = 0.030380), it does not generalize well on the new, randomly generated input. In fact, it does worse than it did originally with randomly generated weights, before any back propagation was done (mean error magnitude = 0.484282).

This model performs worse than random on new, randomly generated inputs (mean error magnitude = 0.644465).

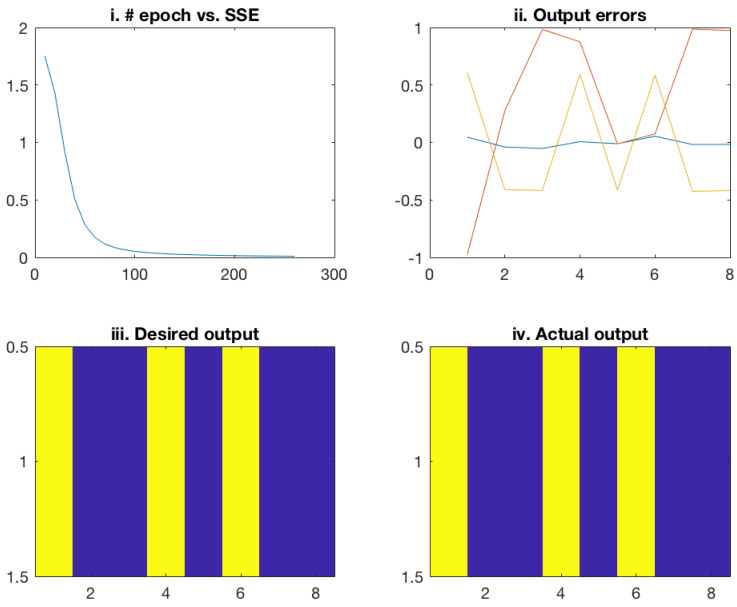

Figure 1: Back prop Neural Network
i. SSE of the network output, as epoch number increases (after SSE convergence)
ii. Absolute value of output error for each input pattern on the trained input (blue), on untrained input (orange), and before back propagation (yellow).
- Average absolute value of output error before backprop: 0.484282
- Average absolute value of output error on trained input: 0.030380
- Average absolute value of output error on untrained input: 0.644465
iii. Parity of desired output
iv. Parity of actual output (after SSE convergence)