NTHU EE6530 Neural Networks

HW3: Back propagation for function approximation

Printed report due April 16 , 2015 in class. **Page limit = 5**.

Lecturer: Yi-Wen Liu

In this homework, you are asked to implement the back-propagation learning rule and apply it for function approximation. Our desired function is $y_{\text{des}} = x^2$. You can try to set up a single-input, single-output, feed-forward network with depth = 2. To illustrate, I have tried the following configuration,

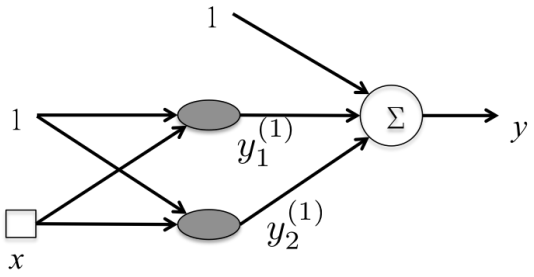

Fig. 1: System configuration

where each arrow represents a linear weight, the dark oval represents a neuron with logistic nonlinearity, and the summation unit Σ represents linear filtering. Under this configuration, we have 7 weights or biases to learn adaptively. Since this system is rather simple, I suggest you should derive your back-propagation algorithm from scratch before you start coding. When you are done coding and ready to test your system, I suggest you can follow these steps:

0. Initialize the system with random weights and biases.
1. Randomly pull a sample of x, e.g., between ± 1.
2. Set $y_{\text{des}} = x^2$, and calculate the error $e(n) = y_{\text{des}} - y(n)$, where y denotes the output of the system at this instance.
3. Adjust the weights (and biases) using the back propagation method.
4. Repeat steps 1-3.

Successful learning should cause the weights to converge so the final weights and biases produce near perfect relation $y = x^2$ for all x within the learned range. The following figure demonstrates the process and the final results of learning achieved by my system, where each red dot represents the instantaneous output given one example of x, the blue line shows the system's output using the final set of weights (and biases) obtained after 5000 iterations, and the green dashed line shows the desired I/O relation $y = x^2$.

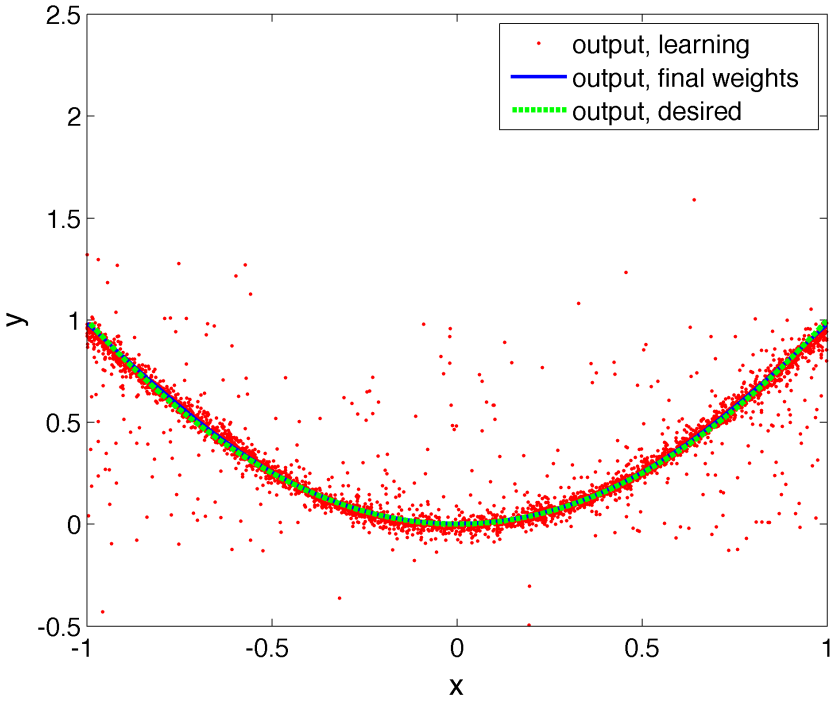

Fig. 2: The output produced while learning, to be compared against the desired output and the output produced by the weights that are learned after 5000 iterations of online-learning.

You can see that, although the instantaneous output is scattered at the beginning, after "seeing" one thousand examples or so, the system finds a proper set of weights that approximate the function $y = x^2$.

Following are a few big questions for you to consider.

1. Does initialization matter?
2. Does randomization matter?
3. How does the learning rate affect the system's behavior?
4. Does the system always converge to the same set of weights or does the system sometimes "get stuck" in a sub-optimal region in the weight space?
5. How well does your system learn a different function, say $y = \cos^2\left(\frac{\pi}{2}x\right)$, $y = \sin kx$, $y = |x|$, or $y = |x|^{1/2}$?
6. Continuing from 3, does it help to increase the number of hidden neurons if the system shown in Fig. 1 does not learn a function well?

Your written report can include any other discussions you have in mind. Support your arguments with numerical experiments. Have fun back-propping!