Lab Report #1
CSE 779
Andrew D Yates

Abstract

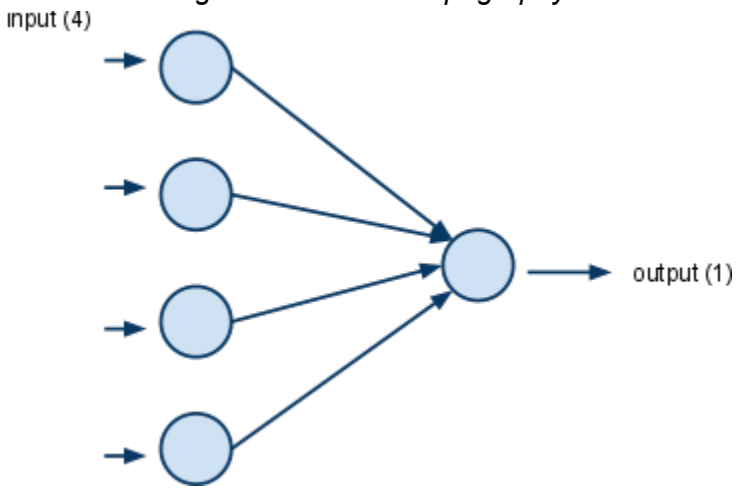
Neural networks are self-training software programs which can arbitrarily approximate functions similar to how a function can be approximated by several polynomial sums. In this lab, we implement a simple fully-connected neural network with four binary inputs, one output, and one hidden layer (4-1-1 network) which is trained to approximate the 4 bit parity function with an absolute error of less than 0.05 for all input patterns. Then, we systematically modify learning rate and momentum parameters of this network to test how network performance is related to these network parameters.

Method

The neural network in this lab is defined as follows:

- 4 input nodes
- 1 hidden layer of 4 nodes
- 1 output node with one output bit
- Activation function $\phi = 1/(1+e^{-av(n)})$ with $a = 1$
- Network topography as in Figure 1
- All weights initialized to random numbers between -1 and 1

Figure 1: Network Topography



This network was programmed using the Python programming language on OSX and executed in the operating system terminal. See the attached source code in the appendix as Appendix C: Source Code.

The “4 Bit Parity Function” is defined with four inputs and one output as in Figure 2. The set of

all inputs and outputs forms one epoch.

Figure 2: 4 Bit Parity Function

(0,0,0,0) => 0
(0,0,0,1) => 1
(0,0,1,0) => 1
(0,0,1,1) => 0
(0,1,0,0) => 1
(0,1,0,1) => 0
(0,1,1,0) => 0
(0,1,1,1) => 1
(1,0,0,0) => 1
(1,0,0,1) => 0
(1,0,1,0) => 0
(1,0,1,1) => 1
(1,1,0,0) => 0
(1,1,0,1) => 1
(1,1,1,0) => 1
(1,1,1,1) => 0

The neural network was trained over the epoch repeatedly until the maximum absolute error for all desired outputs was less than or equal to 0.05. At this point, training ended, and the training handle printed the total number of training epochs and the final network function table. To record the results and to monitor real time network training, the network training program regularly printed network statistics to STDOUT including the current epoch, the average error, the maximum error, and less frequently, the current result set of the network. Because training can take several minutes to hours for over many iterations, the output was divided into a collection of files of output piped from STDOUT. The final version of the training handle trains 5 copies of the network for each of twenty of parameter permutations: the training rate *eta* = 0.05 to 0.50 in 0.05 increments, and the momentum scalar *alpha* = 0 or 0.9.

Results

Generally, the neural network trained faster with a higher training rate *eta* and with non-zero momentum *alpha*. Training was obviously slowest when *eta*=0.05 and *alpha*=0 (1,371,119 epochs) and generally fastest when *eta*=0.35 and *alpha*=0.9 (15,439 epochs). Training was generally faster with momentum, *alpha*=0.9, (275,973 epochs) than without momentum, *alpha*=0, (62,127 epochs) for all values of *eta*. Variance was high for all permutations, and outlying training samples could be quite large. Also, due to limited computing time, only about 3 networks were trained for each permutation of network parameters. Thus, given such a limited sample set, high variance, the possibility of high outliers, and stochastic influence, the exact values for the results should not be considered as relevant to the results except to sketch the general trend that this network trains fastest for a “medium” value of eta of about 0.3 with momentum and slowest for a very “low” value of eta of about 0.05 without momentum.

In addition to the overall behavior of the network as described above, some features of network training could be observed during operation. Specifically, the network seemed to quickly converge to the “ambiguous” solution of about 0.5 for all inputs, suddenly diverge, and then very slowly converge to an “all but one” solution in which almost all the inputs were exactly correct except for the following input sets and corresponding outputs:

```
input        => d ~= y         Difference
[0, 1, 1, 1] => 1 ~= 0.799731, D: 0.200
[1, 0, 1, 1] => 1 ~= 0.799724, D: 0.200
[1, 1, 0, 1] => 1 ~= 0.799718, D: 0.200
[1, 1, 1, 0] => 1 ~= 0.799714, D: 0.200
[1, 1, 1, 1] => 0 ~= 0.804331, D: 0.804
```

Eventually, after enough epochs, this solution would also diverge and then quickly converge to the correct and final solution. In the case of low eta with no momentum, the network could train for millions of epochs while it very slowly converged to this “all but one” interim solution before finally diverging and converging to the correct solution.

This neural network could be improved beyond the scope of this lab. First, the program could be rewritten in C or using speed-optimized Python libraries to improve training time. Second, the program could be modified to generate and train networks of variable inputs, hidden layers, and outputs. Third, additional heuristics to improve training performance could tested including the addition of a subroutine to attempt to break slow convergence to local solutions by adding noise to neuron weights, testing an increased range of network parameters including the activation function and its parameters, and testing other network topographies and configurations.

Conclusion

Neural networks can be trained approximate functions, but they may require very many epochs to train, and their training performance can be unpredictable. In this case, network training performance was improved by increasing the training factor *eta* to about 0.30 and by adding a momentum scalar *alpha*. However, these improvements were not guaranteed to always improve network training performance although they generally did so when compared to the original configuration of the network of *eta*=0.05, *alpha*=0 which could run for millions of epochs for almost an hour before converging to a final solution.

Appendix

An easily digested output table has been compiled from the collection of raw output data and included in the appendix as Appendix A: Table of Results. Because the complete output is several pages, only an excerpt of program output is provided in the appendix as Appendix B: Program Output Excerpt. This excerpt clearly demonstrates the function of the program without burdening the reader with redundant information already compiled in Appendix A: Table of

Results. Finally, the two files of source code, the neural network definition in nn.py and the training handler script in test.py, are included as Appendix C: Source Code.