User Manual for Neural Net Implementation

Sanath Govi, Adam Bertelli

In our final project, we constructed an artificial neural network capable of machine learning. Two example data sets which the network can perform well on were used for testing, and the attached programs TesterPoly and TesterXOr can run the network on these data sets.

How to run these programs using command prompt:

1. Move the folder “neuralnet” to desktop.
2. Open Command Prompt
3. Change the directory to neuralnet with “cd desktop/neuralnet”
4. Use the command “javac <filename>.java” for each of the 7 files in neuralnet. This should create seven .class files in the folder.
5. Run either “java TesterPoly” or “java TesterXOr”. This should create a text document called percentages.txt

If you can’t run programs from command prompt, you can also create a project in eclipse and paste the code for each file into a new class, and run it from eclipse.

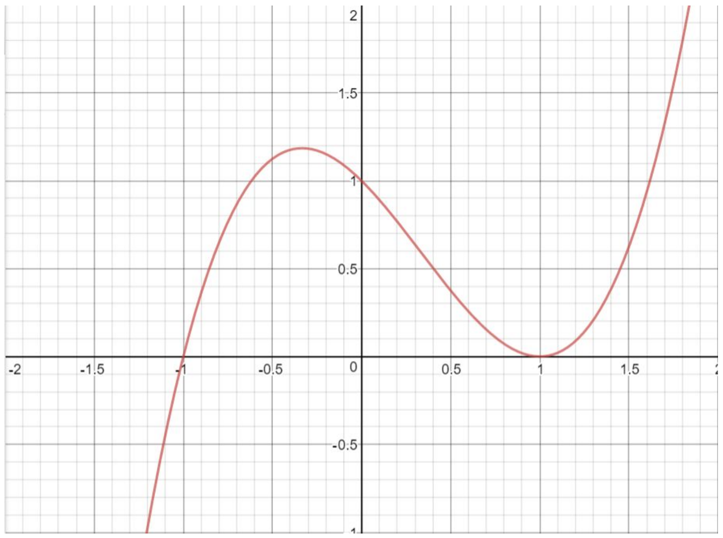
The file percentages.txt will show the percentage of data samples the neural net got correct for every 50 new trials. As the number of trials increases, the neural net gets more accurate, so these numbers start at around 50% (a random guess), and go all the way up to high 90% or even 100%.

The neural net also uses a value called learningRate to determine how much to adjust the weights between neurons based on the most recent data input. We found that changing the learning rate often affects the eventual accuracy of the neural net, so we created an example program LearningOptimizer to test what learning rate gives the most accurate neural net after many trials, in this case for the PolynomialTester class. This program will output to command prompt the learning rate and the final accuracy of the net for each value it tests, and then output the resulting optimal value.

Explanation of the two data sets:

XOr is a binary operation taking two bits as input (each bit can be 0 or 1). It returns 1 only if the two input bits have different values, and returns 0 if they have the same value. Since this is a relatively simple function, the neural net quickly gets 100% accuracy on the data set.

The PolynomialTester tries to determine whether a given point is located above or below a curve in the 2D plane, specifically the cubic polynomial $y = x^3 - x^2 - x + 1$. The region looks like this:



We randomly generated points within this region, and the PolynomialTester usually ended up determining which half they were in with an accuracy above 90%, which is significant, considering the two halves are not split in a simple way, and the areas near the middle require machine learning to predict accurately.