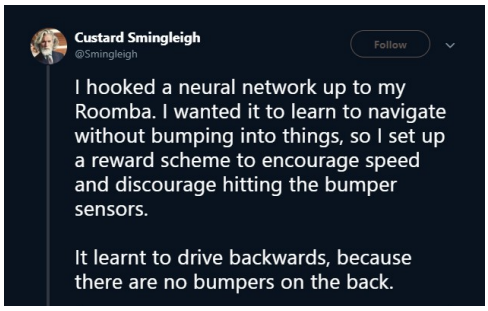
CS 421 Homework #5: Artificial Neural Network

Overview

Your assignment is to create an artificial neural network and teach it your Antics utility function. You'll do this in two steps:

- Part A – creating a working neural network
- Part B – applying it to ReAntics



Part A Specification

Implement a two-layer neural network in Python with four inputs, eight nodes in the hidden layer and and one output node.

Step 1: Create your Weights

With biases this will mean a total of 40 weights in the hidden layer and 9 weights in the output layer. Don't create a Node class like you would for a graph in a data structures class. The nodes of a neural network are conceptual only. The only data structures you should need are lists of floating point numbers to represent the weights and inputs.

When first created, the weights will be random values between -1.0 and +1.0.

Step 2: Output Method

Implement a method that will generate the output of the neural network given the input array and using the current weights. Use a sigmoidal activation function so that the network can be trained using backpropagation.

While not required, this step can be implemented using matrix multiplication. If you wish, you can install the NumPy library (http://www.numpy.org/) to do the matrix multiplications for you. It may make things easier and less error prone.

Step 3: Backpropagation Method

Implement a backpropagation method for adjusting the network's weights using gradient descent so that it can learn. Feel free to use example code from the internet/AI but make sure it matches the network design prescribed by this assignment. Adapting it to your purpose will teach you just as much as writing the code from scratch. *Please be sure to cite your source if you do this.*

Debug your code carefully. While not required, it would be a smart move to *use unit tests*.

Step 4: Train your Network
Given the four inputs a, b, c and d: Train the network using this function:

a	b	c	d	output
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Note: These data have been posted as a Python list on the course website in the file named PartA_TrainingData.txt. You're welcome.

Perform this training by performing a series of epochs. During each epoch, you will give the network 10 inputs chosen randomly from the sixteen possible inputs. For each input, calculate and record the network's error. Then, use your backpropagation algorithm to train the the network to perform better next time for that input.

At the end of each epoch, print the average error for that epoch. This error value should be the only output of your program! If this average is less than 0.05, stop. Otherwise, do another epoch.

If you find that your error does not decrease over time, something is wrong and it's time for debugging.

Part B Specification

Start with the Antics agent that you created for homework #2a. Make the following modifications.

1. Re-architect your neural network to learn a utility function for Antics. You will likely need many more inputs and nodes to do this. Specifically:
    a. *Input:* It should be given the same information as the evaluation function. However, you should map the input to an array of floating point numbers in the

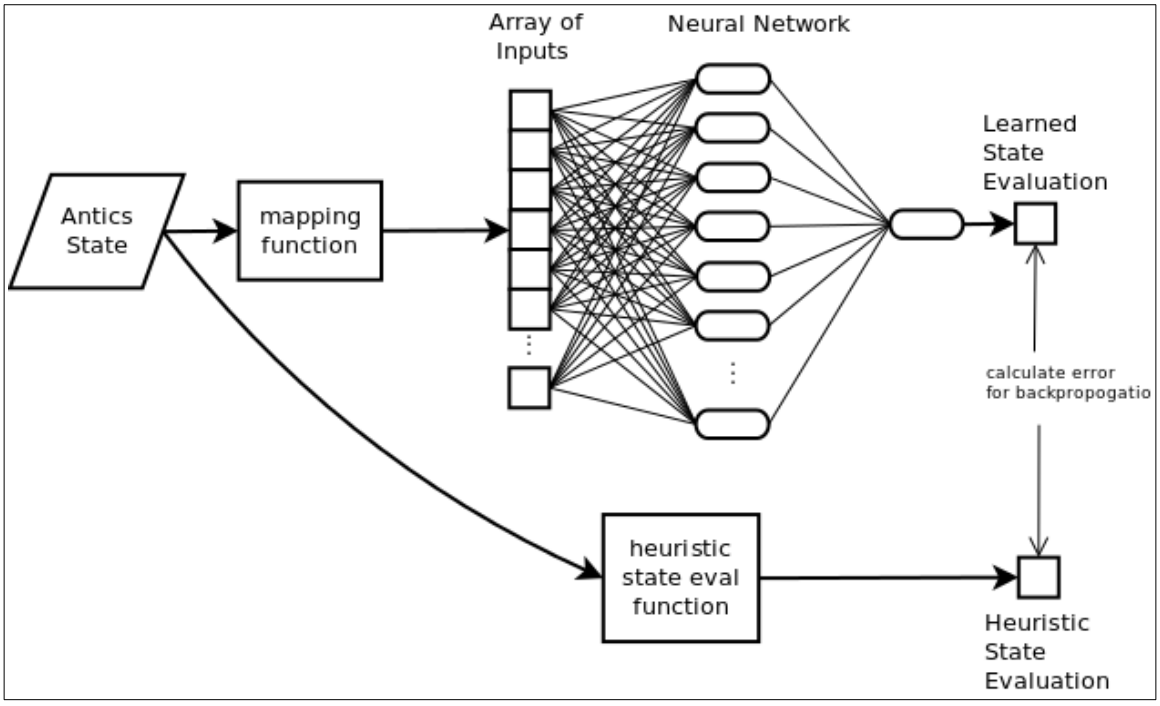
range 0..1 to make it more accessible to a neural network. How you do this mapping is up to you and there are many ways you could do it.

b. *Output:* Your output of the network should be the same as your evaluation function: a single number between 0 and 1 that represents how "good" the current state is for the player. Your network should have at least two layers. The last layer will only have one node since there is only one output value (the overall utility of the state).

Remember: Designing a neural network is as much a black art as a science. There is no "correct" design, only designs that are more effective than others.

2. Train your neural network using the heuristic utility function you wrote for Homework #2a. The result from your heuristic function is the "target" (or "correct") answer that you will pass to the backpropagation algorithm.

   This diagram may be helpful (or terrifying):



3. Getting your network to learn a function this complicated is not a trivial affair. You will likely find you need to tune some parameters – like the learning rate, the number of hidden nodes, the design of the input mapping function, etc. – to get get a good result.
4. Once the network's error is sufficiently low, modify your code so that your original heuristic function is never called. Instead, the neural network is used to evaluate each state using the weights that you generated during the training. Just hard-code those weights in. (The backpropagation function won't be called anymore but you should leave the code in place for grading.)

5. Your resulting agent should defeat each of the three given agents in 7 games out of 10. Furthermore, your agent should be able to play 10 games against the random agent in less than 10 minutes on a lab computer.

Hint: (from Stephen Robinson, class of 2016):

At first it learned very quickly so that it was perfectly accurate by the end of every game, but when the next game started, the neural network was completely wrong. The problem was that the neural network was almost completely ignoring my input and just returning a number which represented the average-ish score at that point in the game. Every time it just went "Oh, this number is about .006 larger than last time, I'll just make my weights a tad bit bigger." But at the start of the next game they were all wrong. To fix this, I saved a copy of every game state in that game and its training score. Then I randomized that list and did all the training at the end in a random order. After that it worked [much better].

Turning In Your Assignment:

Follow the steps defined in homework #1.

Grading Guidelines:

40% Part A Functionality	Does your code for Part A correctly learn the given function?
20% Part B Functionality	Does your code for Part B meet the given performance specifications in terms of games won and time spent?
40% Design	Did you follow the instructions in this homework correctly? Is the program well designed? Have you followed the CS421 coding standard? (The CS421 coding standard is posted on the course website.)
Teamwork	As per usual. See HW#1.