THE PLAN

1 Coding Connect-Four

Not writing anything here because it seems pretty easy.

2 The ComputerPlayer Class

- Function *init* - Instantiate a NeuralNet object (see below)
- Function *takeTurn* - Input the current gameBoard (which should be an element of the Game class). Find the list of possible moves. For each possible move, take the corresponding game board and put it through the NeuralNet. Note that the NeuralNet somehow needs to know who it is optimizing for (should it find the move that makes player1 win or player2?). My proposal is to save one board for each player, where a 1 in some locations means "my piece", -1 means "their piece", and 0 means empty. The reason I suggest this is so that we can handle the issue before it ever reaches the NeuralNet (so we don't need to add currentPlayer as an input to the NeuralNet).
- var *gameBoard* - Read the above discussion for why I want this player to have his own gameboard. If we wind up agreeing on this then we should move around how variables are stored (the Game class shouldn't store the board I don't think).

3 The Neural Net Part

Note: A lot of this is subject to change, especially since I'm writing this up in a plane with no internet access so it could wind up being bogus. But here's my first pass at putting together the neural net structure.

The overall plan is for each neuron to accept a linear combination of values from the previous layer, pass it through our activation function (my first vote is ReLU it's so easy), and send this value with the corresponding weights onto the next layer. I propose we implement this with the following structure

3.1 NeuralNet Class

- Function *init* - The constructor. From a separate file it loads the neuron structure, along with each of the connections and weights.
- Var *numLayers* - Number of layers, including input and output layer
- Var *neurons* - a 2d list. First index is which layer, second index is which neuron in layer. Actual list elements are from the Neuron class, which is below.
- Function *doTheThing* - The function that calculates the value based on a given input. This should load up the first layer (index 0) of the network with the input values, *fire* each layer (one at a time), and collect the output from the last layer.

3.2 Neuron Class

- Function *init* - Inputs the *nextNeurons* and *weights*. Sets *value* to zero.
- Var *value* - Holds the input from the previous layer. Initialize this to zero, then update it as the nodes from the previous layer get fired off. Once the previous layer is finished this is the final value.
- Var *nextNeurons* - A list of pointers to the next layer of neurons
- Var *weights* - A list of weights corresponding to *nextNeurons* above
- Function *fire* - Apply the activation to *value* and send it (with appropriate weights) to each guy in *nextNeurons*, updating the *value* in that dude.

4 Update Process

Hahahhahaha I don't know how this works. This is where most of the studying will come in. But we'll somehow update the file that stores all of our network weights.

There's even a decision to be made between updating after every turn of the game, or only updating after a full game has been played out.