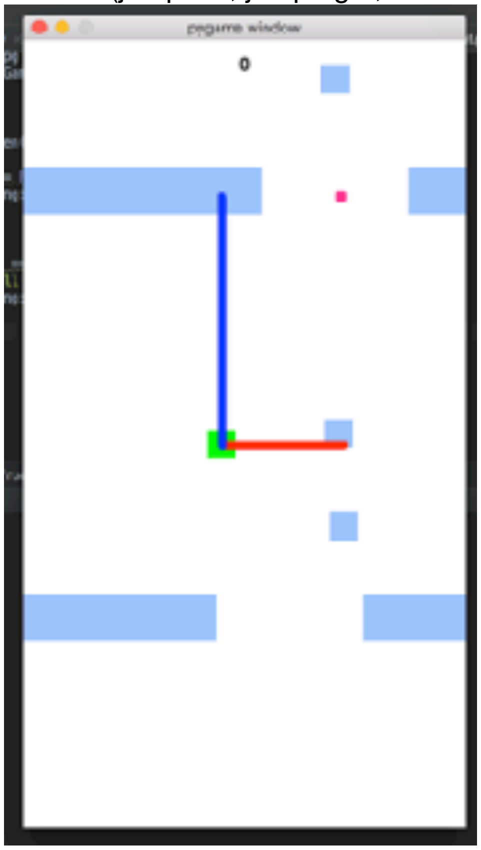
Plan for How to Make a Neural Network (NN) Bot to Play Atom Jumper
Mack Tang

This document is my detailed plan for how I will use machine learning to play the game I have created in pygame.

Note: In order to simplify things for the NN bot, I would like to initially delete all square brick obstacles, and only have gates be obstacles. If the NN bot works, I plan to re-add the brick obstacles to make the game harder, and retrain it.

1-Inputs and Outputs

Neural nets take an input and return an output. My input will be information about the state of the game. My output will be an action (jump left, jump right, or do nothing).



Caption: Red Line is Horizontal distance to next gate, Blue Line is Vertical distance to next gate

Inputs:
V = Vertical distance to the next gates
H = Horizontal distance to middle of next gates

Outputs:
jump left by calling jump(left)
jump right by calling jump(right)
do nothing

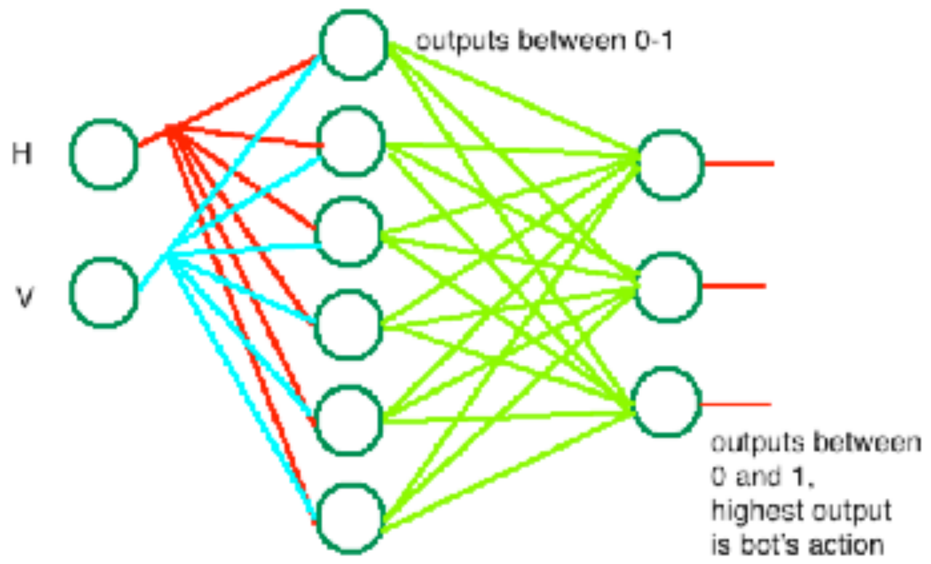
2-Pre-processing Inputs

Based on my research, neural nets require all inputs to be between 0 and 1. For example a grayscale image with a pixel value between 0 and 255 should be scaled to between 0 and 1. I plan to do this by dividing all values by the largest possible value:

<u>Pre processed Inputs between 0 and 1</u>
V = V / (maximum possible value for V)
H = H / (maximum possible value for H)

Neural Network Layout

The number of layers and number of nodes within each layer are varied. In general, more layers are supposedly capable of more subtle decisions. To start I plan to use an input layer, a hidden layer with 6 nodes, and 3 output nodes. I am basing this off of someone else's neural network on automating "Flappy Bird" [1], I believe that as the games are similar, it is a good starting point.



Math in each Node

Every node has a weight associated with each of its inputs. The node takes the weighted sum of all the inputs. A "bias" is added to this. Then, it takes this output and runs it through the "Sigmoid Function" to get the final output between 0 and 1.

Training the Neural Net
To train the neural net, a cost function measuring the performance of the NN is usually used. The distance traveled up or the score may be used as our cost function, which in this case should be maximized. After making random variations and playing the game to characterize many points on the cost function, an optimal way to change the weights and biases can be determined to maximize the score. The method for finding this optimal change is called “Backpropagation” and is similar to finding the local minimum of a function in calculus. The changes to the weights and biases are made, and the process repeats. In this way the NN can be trained. The game ’s fps will be sped up during training. Hopefully after training, the NN will have the correct weights and biases to play the game well.