Artificial neural nets

Preface

We use Artificial Neural Nets(ANNs) to control a “cart” within a simulated environment. The environment is essentially a game, where the performance of the ANN is scored by how long it can use its control of the cart to balance a pole on top of it. We then use the ANNs with the best scores to make new generations of ANNs, and repeat the process in an attempt to evolve the ANNs by way of natural selection.

Introduction

An Artificial Neural Net is a collection of nodes, and connections, that are put together to simulate the theoretical workings of neurons found in biological entities. From a programming perspective, a ANN receives a number of inputs from a number of sources that are given different weights, the ANN then decides on an output. In the Simulated environment in this project there are two possible inputs the ANN can output: move to the left or move to the right. For every frame of the “game”, the ANN is given all the parameters of the scene, and give an output.

ANN

Crossover

When the GA class defined in *Genetic\_algoithm.py* calls the *new\_generation(self)* method, it will call breed(self,parent1,parent2) and *select\_n\_from\_distribution(2)*. *select\_n\_from\_distribution(2)* randomly picks two ANNs as parents from the current generation from the population with probability defined by calling *get\_relative\_fitness(self).* *breed(self,parent1,parent2)* is run *self.n*/2 times where n is the size of the generation set when the GA was constructed.

Parent1 and parent2 are cloned to child1 and child2. Child1 is then used to call *crossover(self, other, \*args)* from the ANN class defined in *neural\_network.py.* The *coefs* and *intercepts* defined by the *MLPClassifier,* representing all weights and biases, are merged into rows, one from each parent. *crossover(self, other, \*args)* then calls *\_crossover\_rows(self, row1, row2, crossover\_method, ravel)* to do one of three defined crossover operations. The input ravel is used to process matrices into rows before running a crossover method.

The code has three crossover methods available.

"one-point":

The two rows are split in two somewhere at random, and two new rows are made using these four parts. One part is made with the first part of row 1 and the second part of row 2, the other is made with the opposite.

"two-point":

Much like "one-point”, except the rows are divided into three parts at random. One of the new rows would be made from the first part of row 2, the second part of row 1 and the third part of row 2. Vice versa for the other new row.

"uniform":

Uniform crossover assigns a 50% chance to each variable in the row that they be one of two parts used to make the new rows. The new rows are then made much like how the one-point split is made.

In our code, these methods are realized by using a mask of index numbers using slice() for one or two point crossover, and using an array of boolean values for the uniform crossover.

Mutation method

Summary