Simulation Trained Agents

Sriram Shanmuga

Department of Electronics and Communication Engineering, Anna University, Chennai, sriramshanmugacf@gmail.com

Introduction

Space robots are tested and validated in environments which resemble the target environment. Training the robots involves the collection of immense amounts of data with an additional overhead of time. If there is no environment which resembles the target environment, an artificial environment which resembles the target environment is created. Problems arise when there are no methods at disposal to model an artificial environment. Though there haven't been any major issues, an alternative to train robots, preferably a software solution, would save resources, time and capital.

Hypothesis

If we train intelligent agents in artificial software environments which resemble the target environment using Neuroevolutionary techniques, they should perform in the desired way in the actual target environment itself.

Methodology

There are two steps to test the hypothesis. (1) Building the artificial software environment, (2) Training the agents. The software environment has little to no constraints on the choice of the programming language. While designing the environment, utmost attention has to be paid to the constraints of the target environment. This is crucial because we are trying to mimic the target environment; we are trying to *simulate* the physics and geographics of the target environment. The agent would be receiving the *same* percepts in the target environment. For training the agents in the software environment, we make use of NEAT (Neuroevolution of Augmenting Topologies). NEAT is a genetic algorithm used for the generation of *evolving* artificial neural networks. Traditionally, the topology of a neural network is designed by a human experimenter and the connection weight values are learnt through a training procedure, for example, Stochastic Gradient Descent and Backpropagation. NEAT, on the other hand, is a TWEANN (Topology and Weight Evolving Artificial Neural Network), which aims to achieve a balance between the connection weights and the topology of the neural network. We start out with a very simple neural network – usually consisting of only two layers viz, Fully connected Input and Output layers. The neural network is *complexified* if and only if required. Thus, preserving the compactness and simplicity while ensuring functionality, resulting in very efficient neural networks. There is one key component to evolutionary algorithms – the Fitness Function. Every agent has associated to it a fitness score. The fitness score projects the agent's performance. This is utilitarian because this helps identify the best genotypes (i.e. neural networks), which would then be crossed and mutated to produce new genotypes. Mutations take place in the form of an addition of a connection or a node or a layer. The new genotypes are used to populate the next generation. The design of the fitness function is as crucial as the design of the software environment, because it directly influences the actions of an agent. The fitness function should be designed in such a way that it conveys the objective to the agents in an effective way. The actual running of the simulations takes time – it usually takes epochs in the order of hundreds of millions. Contrary to expectation, the fitness graph is not linear. This is due to the fact that mutations do lead to decreased performances occasionally. But the ultimate neural network will perform exponentially better than the initial neural network.

Experimentation

As a small experiment, a rover that would explore a simple target environment was considered. The target environment consists only of rocks. The rocks' locations and dimensions (width and height) are randomized. The percepts: 360 inputs convey the distance between the rover and the rocks in the rover's field of view (if there are any), 360+360 inputs convey the width and height of the rocks in the rover's field of view (if there are any), 360 inputs convey the angle between the rover's horizontal and the rocks in the rover's field of view (if there are any), 4 inputs convey the distance to the boundary, 2 inputs convey the rover's position, adding to a total of 1446 inputs. These 1446 nodes form the input layer of the agents' neural network. The output layer of the agents' neural network consists of 4 nodes: move forward, move backward, move right, move left. The fitness function: (1) Punish the agent if it stays in one place, (2) Punish the agent if it leaves the boundary, (3) Punish the agent if it collides with a rock, (4) Reward the agent if it visits a new location, (5) Reward agent if it does not collide with a rock, (6) Reward the agent a score proportional to its lifetime.

Results

Owing to the lack of availability of computing resources available at my disposal, I was not able to train the agents. Thus, the project is still in the initial stages of the second step.

Discussions

However, interesting behaviour by the agents were observed. There was an insidious bug in the first version of the fitness function which did not contain (1) and (4). The other conditions were satisfied even if the agent decided to stay in one place. The agents were quick to identify this loophole and they cleverly exploited it. As there was no node in the output layer which would enable the agent to stay in one place, they moved a tiny bit to the left and in the next frame, a tiny bit to the right or the moved a tiny bit forward and in the next frame, a tiny bit backward. Due to flaw in the fitness function, the genotypes of these agents were selected as the best ones. As a result, all the progenies stayed in one place. Though the agents did not mature due to the very limited amount of training duration, some of the agents exhibited a different movement pattern – moving diagonally.

Keywords: Simulation, Genetic Algorithms, Deep Learning

Author for correspondence
Name: Sriram Shanmuga
Affiliation: Department of Electronics and Communication Engineering, Anna University, Chennai
Email: sriramshamugacf@gmail.com
Mobile phone: 9361351733