F20/F21RO Intelligent Robotics assignment 2 - Evolving a robot controller using evolutionary robotics techniques

Yibo Liang
School of Mathematical and
Computer Sciences
Heriot-Watt University
H00194508
Email: yl9@hw.ac.uk

Kevin Klein
School of Mathematical and
Computer Sciences
Heriot-Watt University
H00233447
Email: kgk1@hw.ac.uk

***Abstract*—Elevated plus-mazes (EPM) are often used to analyse and evaluate the behaviour of rats. In this document, a e-puck robot will be used within the webots software simulator to emulate such behaviour by evolving an Artificial Neural Network (ANN), or to be more specific, a Recurrent Neural Network (RNN). Recurrent Neural Networks are a class of Artificial Neural Networks where connections between layers are bidirectional to form a loop. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. An RNN, in comparison to a classic feed-forward network, can make usage of it's internal memory on order to process arbitrary sequences of input. This way, the robot can explore the test field environment in a similar way as the rat would and learn from it's errors and mistakes in an exploring but careful manner. To mimic the rats approach, further biologically inspired techniques such as Evolutionary Algorithms and Genetic Algorithms are also used to fulfill the requirements for this assignment.**

I. Introduction

In this assignment, a robot controller will be evolved using evolutionary robotics techniques. The controller will be modeled as a Recurrent Neural Network (RNN) to control a e-puck robot on an elevated plus-maze (EPM) arena.
For the task, the robot should move in the elevated plus-maze and mimic as close as possible the behaviour of a rat. Basically, the robot should move around within the maze and explore it as quick as possible.
The behavior of a rat in the EPM is a result of the approach-avoidance conflict. Therefore, the robot will have to deal with 2 different types of behaviour which stand in contrast to each other: exploratory and fear.

A. Literature review

1) Genetic algorithms: Genetic algorithms (GAs) are search algorithms, based on the idea of natural selection methods and genetics. They are used to solve optimisation problems using random search methods. When referring to the term random search, it is referred to an approach where the search space is moved closer towards the region which in previous generations delivered better performance.

The natural selection methods which inspire Genetic computing are ones such as 'survival of the fittest', where individuals compete against each other with the motivation that the fitter individuals dominate the weaker ones and survive. In GA, each population consists of a population of character strings which are analogue to chromosomes that we can observe in our human DNA. Each individual character in the string represents a specific location in the search space and therefore a possible solution.
This means that each individual within a generated population competes with its other individuals for resources and possible partners. The individuals which survive each generation will then breed more offspring than the weaker individuals. With every completed run of a generation, the genetic algorithm makes the population more suited for it's environment.
At first, an initial population is generated randomly. The genetic algorithm therefore generates three operators as in selecting the parents for the survival of the fittest procedure, paring them together with crossover and mutating them, meaning random computations are made.
The idea behind the selection operator is to preferably select the strongest individuals which will breed children for the next generation.
What separates GA from a lot of other optimisation techniques is the crossover operator. This operator takes all the values up to a fixed crossover point in both of the selected parent strings and swaps these two data sets with each other. The

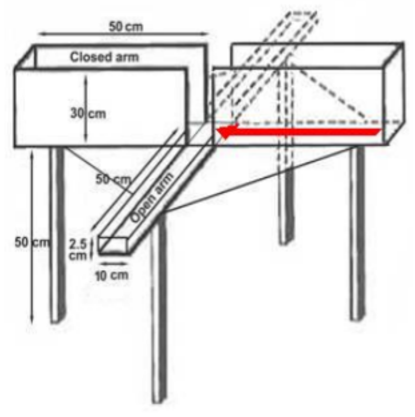

Fig. 1. Elevated plus-maze (EPM)

two offspring created as in the resulting strings will be put into the next generation. With each generation completed, the population will most likely consist of stronger, better individuals.

2) Artificial Neural Networks: In general, a network usually consists of two main components: a set of nodes, also referred to as computational units, and connections between these nodes. The nodes usually receive an input, process these inputs and finally produce an output. The connections between the nodes define the data flow within the network.
Within Artificial Neural Networks (ANN), these nodes are often referred to as neurons. ANNs are biologically inspired by the human brain as well as brains of animals, where the neurons receive signals through synapses which are located on the membrane of each neuron. Usually, the input signals are multiplied with weights, which define the strength of the corresponding input signals. These are then computed by a mathematical activation function and lead to the corresponding neuron to activate (to fire) if they are strong enough, meaning their values reached or surpassed a certain threshold. This generated output may at the same time be the input signal for the next neuron and may contribute to this neuron to activate the same.
In order to make ANNs learn, an algorithm, most often a evolutionary algorithm is implemented to enable the network to learn to then finally achieve the desired output.

3) Recurrent Neural Networks: Compared to Artificial Neural Networks (ANN), which consist of a feed-forward architecture, whereas Recurrent Neural Networks (RNN) additionally provide a feed-back connection from the output layer to the hidden layer, creating a activation loop. This brings the advantage that the network can produce a sequence reproduction.
The Recurrent Neural Network implemented for this coursework consists of a standard Multi-Layer Perceptron (MLP) with additionally added loops, which provides some additional memory functionality to the network.

4) e-puck Robot: The e-puck robot is a mini robot which comes along with a relatively large amount of 8 distance sensors, a dsPIC processor with 60MHz, various LEDs, Bluetooth and is completely open source in terms of hardware and software. It is programmable in C and provides access to all it's low level electronics.

5) Behaviour Based Robotics vs Evolutionary Robotics: When referring to Behaviour Based Robotics, the controller can be programmed directly towards a desired behaviour, where as in Evolutionary Robotics, the autonomous robot learns from the environment and evolves it's own intelligence.

II. Program rationale

A. Design of topology for RNN

For this coursework, our network is basically a recurrent network built on MLP, that has bidirectional connection from one layer of neurons to another. The neurons will additionally have a connection to themselves, meaning the neuron value of previous step is added for the calculation for new value, this feature gives the ability of memory to the neural network.

The topology of our RNN is initially set as following: 1 input layer of 8 neurons, 2 hidden layers with 5 neurons each, and 1 output layer with 2 neurons that controls the speed of the wheels of E-puck robot. Each layer is connected to another layer like MLP, but every connection is bidirectional, moreover, each neuron in hidden layer and output layer is connected to itself. This topology is later modified to 8 input neurons, 1 hidden layer with 5 neurons and 1 output layer with 2 neurons.

B. Implementation

1) Genetic Algorithm: We use genetic algorithm to search for best weights for the RNN. The webot simulator already comes along with a genetic algorithm. We implement our RNN with a encoding function that convert the neural network to an array of float number, which can be directly used by this genetic algorithm implementation. Then modification are done to the genetic algorithm implementation to adapt our task. Finally, We designed the exact maze in Webot shown in Figure 1.

2) Encoding: The neural network is encoded to an array of float numbers that each correspond to the value of a connection in the network. This array is sent to E-Puck controller and decoded to a neural network, which uses E-Puck robot's distance sensor to control the robot. Each RNN is run on robot for 60 seconds and the fitness is valuated for further process.

3) Evolutionary process: Every generation, a population of 50 neural networks are evaluated, and their fitness is sorted in descending order. Elitism is used and top 10 percent of the population are cloned, and then tournament selection with a tournament size of 2 is used to choose parents for reproducing next generation.

When the 2 parents are chosen, they are encoded as 2 arrays of float numbers, then each pair of value, namely V1 and V2, are used to calculate Vnew=V1-k+ (V2-V1+2k)*RAND, where k is a small value equals to 0.1, RAND is a randomly generated number ranging from 0 to 1. In this way the child is an array of float numbers that lies within its parents, but with a possibility to exceed its parent.

Finally a mutation function is used on new child, that each of its float numbers have a probability of 0.06 percent change of being added with a random value varying from -0.3 to 0.3.

A maximum generation of 120 is used for best results.

III. Problem encountered

A. Webot bugs

The first problem we ran into was the webot collision bug. As we finished setting up the world and coding the program, we started running tests where the evolutionary process was managed by supervisor component of webot. The bug occurs occasionally that when the robot fell off the maze from unsafe area and whose position and rotation was reset by the supervisor, it kept falling off through the ground infinitely. We had to stop and reset the entire evolving process for it to work again. We spent many days just trying to solve this problem.

While in the process of solving above problem, we found another problem with the E-puck robot in Webot. When the robot collided with the wall and get stuck, its wheel came off and was rolling in the air. When this happens, the robot will keep turning in one direction and the world has to be reset.

After digging into Webot documentation, we decided to abandon the way to reset the E-puck robot given by default and implement our own. The robot is removed from the world after each run, and reloaded from a wbo file by supervisor, in this way, the above 2 bugs are avoided since the robot will be fully reset, rather than only resetting its position and rotation.

B. Slow Evolution

The evolution process itself contributes to the majority of difficulties of this coursework, because Webot can only have one robot evaluating one RNN at one time. It is possible to set multiple E-Puck robot with multiple mazes, but the whole program will have to be rewritten because the use of global variables of the given genetic algorithm. We then decide only to use the default one and wait for the results.

1) Choice of fitness function: The most difficult problem was to design a good fitness function so that we push the RNN to evolve to desired direction. We created 3 different fitness functions that are used to evolving our RNN.

The first fitness function is calculated as F = E * (1-P) - S, where E is the effective exploration distance, and P is the percentage of time that the robot being in dangerous place each run. E is calculated as the distance travelled from one end of the maze to another, for example, the robot is place in the centre of the maze, when it reaches one safe end, it travelled approximately 0.5 meter, then E=0.5, and if it turns around and arrives another end that is 1 meter away, E=0.5+1=1.5. S is the time of second the robot stay still. Using this fitness function, the robot is supposed to explore as much as possible in the maze while avoiding dangerous area (routes that are not protected by walls).

The second fitness function is more complex: 10 points Pn (n=1 to 10) are distributed on the maze with reward value (Vn), each time the robot approach near one, namely Pn, the fitness E is added by Vn, and Vn which was initially 1.0, is then decreased by 0.1. Then the final fitness is F=E*(1-P)-S same as previous function.

The last one we used was F=E*N*(1-P), where $N = 0.996^S$, and S is the number of time steps the robot being still (or stuck). This one is different from the second one in the sense that it does not easily give negative fitness. This prevent the problem that the robot learned to roaming in very small area in order to avoid being punished to a very large negative value which happens if we use the second function.

IV. Evaluation

A.

It take more than 2 hours to evolve and evaluate 120 generations of RNN. As shown in the figure 2, the fitness increased drastically to 12 in first 20 generations followed by a drop to 6, and then with fluctuation it gradually increased to 17 in the following 30 generations.

The final RNN exhibits the desired rat-like behavior, it goes to both ends of the maze, turns around when it approaches to

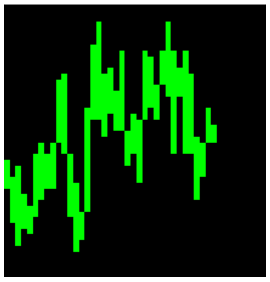

Fig. 2. Fitness graph from Webots

the end, and avoids going to dangerous route. This shows the successful design of our program and rationale, although the robot does not always goes to the very end of the maze. We believe further evolution will give it better performance.

V. Conclusion

This coursework give us good understanding of Recurrent Neural Network, Genetic algorithm, and the experience of using Webot simulator as well as programming in C. The successful approach of using RNN with GA in simulator in this coursework implies that it is practical to use RNN to solve real robotic problems. The memory property of RNN enable robot to accomplish tasks in complex environment.

March 30, 2016

References

[1] IEEE, *A Comprehensive Review of Stability Analysis of Continuous-Time Recurrent Neural Networks*, IEEE, July 2014
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=6814892

[2] Dorado, Julin; Rabual, Juan R.; Rivero, Daniel; Santos, Antonino; Pazos, Alejandro *A Comprehensive Review of Stability Analysis of Continuous-Time Recurrent Neural Networks*, IEEE, 2002
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=1007748

[3] Shimo, H.K.; Roque, A.C.; Tinos, R.; Tejada, J.; Morato, S. *Use of Evolutionary Robots as an Auxiliary Tool for Developing Behavioral Models of Rats in an Elevated Plus-Maze*, IEEE, 23-28 Oct. 2010
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=5715240

[4] Grzesiak, L.M.; Meganck, V.; Sobolewski, J.; Ufnalski, B.; Morato, S. *Genetic Algorithm for Parameters Optimization of ANN-based Speed Controller*, IEEE, 9-12 Sept. 2007
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=4400689

[5] Shibata, Junko; Okuhara, Koji; Shiode, Shogo; Ishii, Hiroaki *The Self-Organizing Map Applying the "Survival of the Fittest Type" Learning Algorithm*, IEEE, 26-28 Nov. 2008
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=4696444

[6] Ganapathy, Velappa; Yun, Soh Chin; Ng, Jefry *Fuzzy and Neural controllers for acute obstacle avoidance in mobile robot navigation*, IEEE, 14-17 July 2009
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=5229761

[7] Al-Shareef, A.J.; Abbod, M.F. *Fuzzy and Neural controllers for acute obstacle avoidance in mobile robot navigation*, IEEE, 24-26 March 2010
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/xpls/icp.jsp?arnumber=5481067

[8] Hiroshi Nakamura, Akio Ishiguro, Yoshiki Uchikawa *Evolutionary Construction of Behavior Arbitration Mechanisms Based on Dynamically-Rearranging Neural Networks*, IEEE, 2000
URL: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/stamp/stamp.jsp?tp=&arnumber=870290