Neural Network, Genetic Algorithm, Robot Algorithm Creation Report

IMAT2801 CI and Control Systems
De Montfort University

Artem Bobrov (P2547788)

May 15 2021

Contents

Introduction

Computational Intelligence and **Control Systems** began to gain popularity with unbelievable rates. Having said that both of them are intended for problem solving yet they are quite different.

Computational Intelligence introduced the ability of computers to "learn" and use given information (datasets) to solve specific problems. It can be used to fulfil simple tasks such as training it on a small datasets (this was accomplished within this project), as well as more complicated problems like monitoring the haloketones presence in drinking water. It is believed that linear/log regressions models cannot well predict haloketone levels but backpropagation algorithm is good to predict haloketone levels in internal validation (Deng et al. 2021).

Control Systems on the other hand are mostly used to regulate problems on a physical layer. For example a control system can affect the behaviour of another device based on the gathered information. A simple example is a thermostat controlling a boiler. A maze-solving algorithm is a part of control systems. It is believed that needs in a control system that can control a robot in an efficient way are growing (Dang et al. 2010).

This report provides information and description of the working process carried out in order to complete the coursework for **IMAT2801 CI and Control Systems**. Three tasks that were to be undertaken are:

- Neural Network with backpropagation mechanism
- Genetic Algorithm
- Maze-solving robot in CoppeliaSim

All three tasks required not only good knowledge of programming languages such as **Python** and **Lua** but also a well thought out design of those complex programs.

Neural Network solution (written in Python)

Overview

The neural network created for this assignment features a backpropagation mechanism and is a feedforward type of a network. Backpropagation is a basic tool for pattern classification application of Artifical Neural Networks medical diagnosis and remote sensing (Amrutha & Remya Ajai 2018). In spite of the fact that this neural network is not used for medical researches, backpropagation plays a big role in all-purpose ANNs. In simpler terms backpropagation refers only to the algorithm for computing the gradient (Wikipedia 2021). Neural networks mimic the human brains and neuron connections specifically. Human learning reflects on the connections between neurons and changes the strength of the bonds between them. Neural networks use term "weights" to describe the strength of these bonds (Amrutha & Remya Ajai 2018).

Coding

Results

Genetic Algorithm (written in Python)

Overview

Genetic Algorithms are a part of evolutionary computing and biological theory is used as a base in their design. They are used to solve different optimisation problems as well as search problems. Those algorithms mimic the principles of nature. Survival of the fittest is the main goal of genetic algorithms. Yet not all of them stick to the same rule. There are some other genetic algorithms called nomadic which do not discard the weakest species (Siva Sathya & Radhika 2013).

The assigned task was to design a genetic algorithm for **Ackley Function** optimisation. The algorithm in the appendix 2 introduces a mutation function in order to maintain genetic diversity between the generations. Respectively, it follows the principles of the biological mutation.

Coding

The genetic algorithm solution is written in **Python** and PyCharm IDE by JetBrains was used in order to complete the task. The whole project was split in separate classes (and different files) to maintain code readability and maintainability. Random and Math libraries were used during creation of this project. The genetic algorithm is made universal and can work with every given function. Passed a number of tests for a sphere function and Ackley Function. Planned tests include: Rastrigin function, Rosenbrock function.

Difficulties: Mutation function was quite complex to implement and designing the offset change in particular. The main complexity in writing the code was to translate every rule and thought into code.

Results

Created genetic algorithm was tested optimising Ackley Function and it showed the following results: -0.06783551718545366 as for the X coordinate, -0.014311762013534637 as for the Y coordinate and 0.31929690272986244 as for the best score value with parameters as follows: number of individuals = 500, crossover rate = 0.5, mutation steps = 15, chance of mutation = 0.4, number of epochs = 15, result limited between -5 and 5. Those results can vary in different runs and iterations therefore allow some difference.

Maze-solving Algorithm (written in Lua)

Overview

Robots are essential in science and especially in undertaking tasks that are impossible to be accomplished by a human being. The task set by the assignment was related to designing the behaviour of the pioneer robot which can be used to explore, document, transport, research and many other things. In order the robot to work properly it has to be programmed for a specific task.

The main algorithm of the robot presented in this report was to solve a given maze. In the presented code it runs a modified Breitenberg algorithm that uses multipliers to control velocity of the wheels and an additional variable used to detect a wall in front of it.

Coding

The code features a state machine using an improvisation in order to mimic a switch statement present in other programming languages (e.g. C++, Java, PHP). We have a set of states which correspond to the robot's current action. Multipliers are used to control the velcity of the motors so the turns can be manipulated precisely. In the initialisation function the lab "skeleton" was used to facilitate the coding.

Difficulties: Many maze-solving algorithm were applied but not implemented correctly in Lua. Wall-following Algorithm, Left-hand Algorithm, standard Breitenberg Algorithm among others were applied with no success.

Results

The robot (P3DX model) had a task to complete the maze without being hard-coded to do so. The time is not recorded with a command but is displayed with a built-in graph object in CoppeliaSim ”attached” to the robot. The graph below shows the time taken by the robot to complete the maze. As it can be seen, the time taken is approximately 80 seconds. The pioneer follows a ”right-hand” algorithm as if there’s space it turns right. If not right, then left. The robot doesn’t store any information about visited tiles and follows a simpler set of instructions.

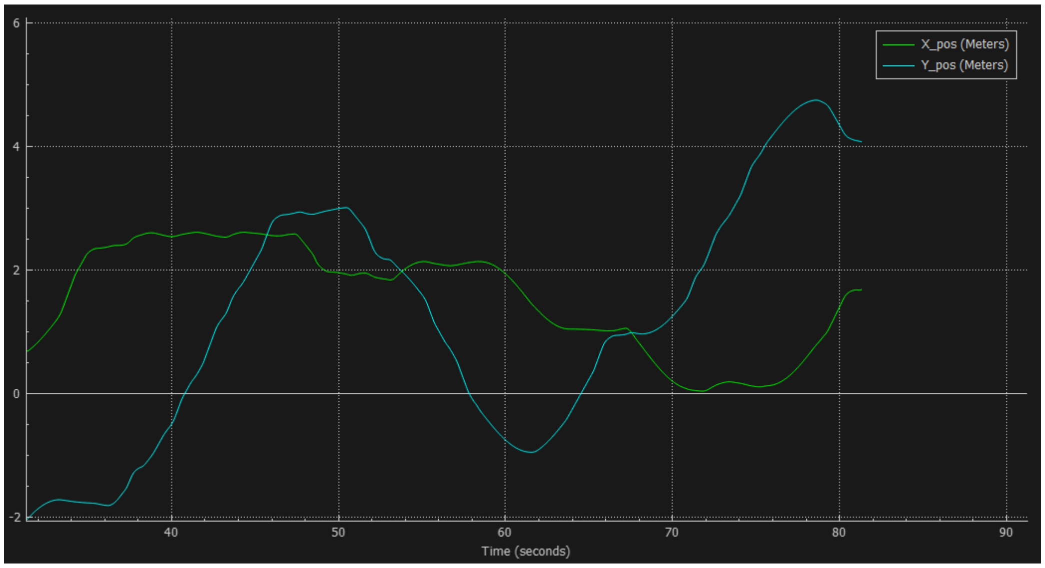

Figure 1: A graph showing the absolute position relation to time.

The robot has been tested multiple amount of time in the given maze only and it’s not certain that it will be able to solve other more complex mazes but it’s believed that the algorithm is modifiable and can be used as a base in creating more complicated systems to solve more complicated problems (mazes). An attempt was made to use a ”left-hand” algorithm to solve the given maze but the peciluliar properties of this maze didn’t allow the ”left-hand” algorithm. The first turn in the maze has a catch and is very hard to solve by a robot using only its supersonic sensors to scan the environment.

Conclusion

Appendix 1 - NN Code

Appendix 2 - GA Code

Appendix 3 - Robot Code

```
-- Switch for velocity multipliers
switch = {
    [1] = function()
        leftVelMultiplier=0.55
        rightVelMultiplier=0.9
    end,
    [2] = function()
        leftVelMultiplier=0.9
        rightVelMultiplier=0.55
    end,

    [3] = function()
        leftVelMultiplier=0.99
        rightVelMultiplier=1
    end,
    [4] = function()
        leftVelMultiplier=1.5
        rightVelMultiplier=1.5
    end,
    [5] = function()
        leftVelMultiplier=-0.5
        rightVelMultiplier=0.5
        -- Reset it
        gen=0
    end,
    [6] = function()
        leftVelMultiplier=2
        rightVelMultiplier=0.03
    end
}

--Initializing the system
function sysCall_init()
    -- Creating an array of sensors (all -1 for now)
    usensors={ -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
    ↪  }
    -- Populating the sensor array
    for i = 1, 16, 1 do
        usensors[i] =
        ↪  sim.getObjectHandle("Pioneer_p3dx_ultrasonicSensor"..i)
    end
    -- Assigning values to variables
    motorLeft = sim.getObjectHandle("Pioneer_p3dx_leftMotor")
    motorRight = sim.getObjectHandle("Pioneer_p3dx_rightMotor")
    -- No detection distance is set to 0.5 as a standard of
    ↪  Breitenberg
    noDetectionDist = 0.5
    -- Max detection distance is set to 0.2 as a standard of
    ↪  Breitenberg
    maxDetectionDist = 0.2
    -- Detection array is empty for now
    detect = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    -- Breitenberg arrays
    braitenbergL={ -0.2,-0.4,-0.6,-0.8,-1,-1.2,-1.4,-1.6,
    ↪  0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0 }
    braitenbergR={ -1.6,-1.4,-1.2,-1,-0.8,-0.6,-0.4,-0.2,
    ↪  0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0 }
    -- Velocity
    v0 = 5
    -- Multipliers
    leftVelMultiplier = 1
    rightVelMultiplier = 1
    -- Distance arrays
    dist = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    res = { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
    gen = 0

    func = switch[state]
end

function sysCall_cleanup()

```

```lua
end

function sysCall_actuation()
    -- Read only necessary sensors
    for i=5,9,1 do
        res[i],dist[i]=sim.readProximitySensor(usensors[i])
        if dist[i]==nil then
            dist[i]=1
        end
    end

    -- Multiplying the velocity by its multiplier
    vLeft=v0*leftVelMultiplier
    vRight=v0*rightVelMultiplier

    -- Maze algorithm
    -- Basically when certain distance to an object is
    ↪  achieved,
    -- we change the multiplier of the speed and the robot
    ↪  turns.
    -- Not perfect multipliers, might tweak them later.

    --1
    if dist[7]<0.32 then
        state = 1
    end

    --2
    if dist[7]>0.42 and dist[7]<1 or res[7]==0 then
        state = 2
    end

    --3
    if dist[7]>0.32 and dist[7]<0.42 then
        state = 3
    end

    --4
    if res[6]==1 and res[7]==1 and dist[6]>0.35 and dist[7]>0.35 and gen>20 then
        state = 4
    end

    --5
    if dist[5]<0.6 and res[6]==1 and res[7]==1 and res[8]==1 then
        state = 5
    end

    --6
    if res[8]==0 then
        state = 6
    end

    -- Every iteration adds one more, see functions above (one checks for it, the other resets it)
    gen=gen+1

    print("Gen: " .. gen)

    func = switch[state]

    if(func) then
        func()
    else
        print("Something's wrong")
    end

    -- Setting the velocities of the motors
    sim.setJointTargetVelocity(motorLeft,vLeft)
    sim.setJointTargetVelocity(motorRight,vRight)
end
```

Bibliography

Amrutha, J. & Remya Ajai, A. S. (2018), Performance analysis of backpropagation algorithm of artificial neural networks in verilog, *in* '2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)', pp. 1547–1550. [Online; accessed 18/05/2021].

Dang, H., Song, J. & Guo, Q. (2010), An efficient algorithm for robot maze-solving, *in* '2010 Second International Conference on Intelligent Human-Machine Systems and Cybernetics', Vol. 2, pp. 79–82. [Online; accessed 16/05/2021].

Deng, Y., Zhou, X., Shen, J., Xiao, G., Hong, H., Lin, H., Wu, F. & Liao, B.-Q. (2021), 'New methods based on back propagation (bp) and radial basis function (rbf) artificial neural networks (anns) for predicting the occurrence of haloketones in tap water', *Science of The Total Environment* **772**, 145534. [Online; accessed 16/05/2021].
URL: *https://www.sciencedirect.com/science/article/pii/S0048969721006021*

Siva Sathya, S. & Radhika, M. (2013), 'Convergence of nomadic genetic algorithm on benchmark mathematical functions', *Applied Soft Computing* **13**(5), 2759–2766. [Online; accessed 16/05/2021].
URL: *https://www.sciencedirect.com/science/article/pii/S1568494612004954*

Wikipedia (2021), 'Backpropagation — Wikipedia, the free encyclopedia', `http://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=1013872709`. [Online; accessed 18/05/2021].