Selected Topics in AI-2

Assignment 3

(Notes on everything is included, just be sure to read it all)

Task 1

You are required to write and train a genetic algorithm to learn to play the game Tetris. This algorithm can use any means of calculations that do not involve machine learning related techniques such as SVM or any neural network variants. **Although applying genetic algorithms techniques on machine learning models is a feasible and reasonable idea (may even be better… may), it is not allowed for this assignment and will be penalized**.

To focus on the development of the algorithm and not the gameplay, you will be given a class that you can use to evaluate your algorithm. The class handles playing the game mechanics and provides a simulation environment, and the game is a bit simplified in this assignment. Most importantly, the class will give you what you need and requires one thing from you, a function to evaluate the moves it can make. This will be through your algorithm. And you require one thing from this class and that is how well your algorithm performed and it gives you that with a performance score, along with a debug of the states it went through and ratings it got using your function.

Your algorithm will need to include elements of a genetic algorithm. Meaning you must have values (that you will calculate based on your algorithm) that affect the rating of a current move. How much these calculated values should contribute to the rating usually isn't clear (should it positive or negative, should it be more or less) and so you have to multiply them by contribution factor to control their contribution. These contribution factor will be your chromosomes and you have to generate, test and evolve them until you reach to reach a better setting (possibly optimal) if such setting exists (maybe your algorithm needs more calculated values and the ones you extracted are not enough). Note the factors can be negative, positive or even zero (zero meaning your idea was useless). These factors can be in decimal or integer values (does not matter), but they are **NOT limited** to -1.0 to 1.0 (so you can have bigger numbers, just be aware that it can override the effects of a score)

Required Settings

The minimum number of extracted/calculated values need to be at least 5 (figure something out, there are examples given below). The minimum number of contribution factors is 4.

The number of chromosomes should not be less than 12, apply at least 10 evolutions.

Make a log file of the values. Save the optimal values (values that got you the highest score).

Your algorithm needs to win, this is a part of the task and failing to do so will make you not get the grades on this part. Winning here is a relative term, it means to not lose until the end of the run and to try to score the most you can (trying to improve your score). These two things are kind of correlated. You can choose whatever seed you wish, just keep it in the code and **write it in the report**.

Run the game for 300-500 iterations (pieces/plays) for training and for the final test run 600 iterations using the optimal contribution factors found. Write them down in a separate code

file and run and **save the score for this run**. Note that the game can end early if you get an early game over.

The game itself

The simplest way to know which game is being referred to is through searching it online, you can even see tournaments each year for it.

The game is simply dropping blocks of different shapes into a board from top to bottom. Each shape is made of a group of squares that are arranged in a certain way, these shapes can be rotated and moved. The simplest space component is a square, it defines the minimum movement and how everything is calculated and made. You earn points by filling a whole line of squares with no gaps. The more lines you clear at once the higher the points you get than clearing them individually or in more steps.

There is a **mini easter egg** hidden in the assignment. If found a bonus will be given, but it will be given to only the first student that emails the solution to this easter egg correctly. The grade is not shared (so students in a team have to be greedy and work on it alone). Don't share the solutions or ideas until I announce a winner. **If you've solved it don't tell others, even your teammate. Let them try until the end.** I will not reply to emails regarding this unless the solution emailed is correct and after I've found the winner (given that I've opened and read the email in the first place).

The given gameplay

You will be given pieces to play and are asked to rate them if they were to fall in each column. According to the first best rating and rotation associated with it, the piece will be rotated to match that rotation and fall down in that given place. The left most part of the piece in whatever rotation is always going to be at the given index being tested (this is guaranteed).

Pieces are generated according to a fixed random seed, so each run will regenerate the same sequence (unless the seed is changed). So only change the seed at the beginning of the running the whole code/script, not between each run to achieve a fair and consistent result.

Rating function

Your rating function is required to do a relatively simple, but a bit complex task. You will be given the current state of the board, the current piece to be played and the next piece, and what column I am thinking of playing it on. You will be asked to return what is the best rotation and its score. You can score each rotation and check who has highest and return it and its associated score. You can also try playing the next anywhere in any rotation (the next piece is not constrained by the column, only the current one).

The reason you are not evaluating all the columns at once for the current piece is that it can eventually be the same. And this will require less code on your part.

Example of Values

The max height in a given state. The difference between max height in this state and the next. The minimum height. The number of gaps (depending on each type of gap). Score gained (if any) for this play. Height of the deepest valley (this is not exactly min height). All of

these can be for current state/board and after playing the current piece, and you can even evaluate playing the next piece (remember the next piece can be played anywhere).

You can add whatever calculations comes to mind (Just remember, no machine learning, it needs to be in steps through an algorithm), the above are counting variables or variables that rate something, you should at the end return single rating value based on these variables. Remember that the contribution factors are what control them, and you evolve the factors not the calculated values.

Code

You will be given a code and a **video will be posted explaining how to use the code on google classroom** (along with the code of course). You are required to submit the code file you've used and a final code for the optimal run. Separate the optimal run in another code file. Put in it the same code, without the evolution part and with the optimal found values of the factors and run it for the requested number of iterations (in the final test run mentioned above).

Please note that you are not allowed to play with the core functionality of the code. You can add more printing and debugging, but not how the parameters are passed. If there is a major issue you can tell me.

Also, write your own separate two scripts. One for the evolution run (the training part), and one for the final run with optimal parameters (the best chromosome found). Include the imports as needed to your code. The main (main encapsulated condition) given in the TetrisSIE script is for demonstration purposes only, you can take ideas from it and understand the code, but I want your actual work in separate files.

Report

Explain your algorithm in detail and how you think it gets you a better gameplay (this can help if you could not achieve a winning condition). Show the progress of the best two chromosomes in all states (show their score in a graph).

Write the evolution/mutation strategy you followed, explain how you choose your chromosomes and how does it evolve to generate the next ones.

Write the seed you chose, the two best chromosomes/factors and their corresponding score at the end of the whole run. Also write the score of the optimal factor after doing the final test run (the one that has more iterations, done in a separate file).

You can include any additional findings you wish to share.

Deliverables

- Code (Submit the whole project with your new added scripts)
- Report

Task 2

You are required to use Ant colony optimization to solve a variant of the traveling salesman problem. In the variant, you have *n* cities. You have to start at a city (doesn't matter which one) and visit all the other cities and come back to the starting city once you're finished

using the lowest cost (or shortest path) possible. Making a full loop through all of the cities. You will be using ant colony optimization (ACO) to tackle this problem.

Problem Configuration

You have to run the ACO on a given set of cities under different configurations.

You'll run ACO on a set of 10 cities with a different amount of ant agent each time. The given number of ant agents you'll use is 1, 5, 10, 20. The amount of ant agents does not change in the run. Each run consists of 50 iterations, a single iteration consists of having all the ant agents complete their full loop (going through each city once and back to the start). After you finish, you'll repeat the same runs when the number of cities is 20 (on each given ant agent amount for each run).

You will need to use the ants to figure out the shortest path going through all the cities and back to the starting city.

Generate the distances between the cities such that the path between a city is between 3 and 50 (inclusive and integers). Generate those distances only once (once for the 10 and once for the 20) and use the same generated distances for the different amount of ant agents.

Code

The code should contain the generation function used and the generated distances for the set of 10 cities and the set of 20 (The simplest way is to use a 2D matrix to represent the from city x to city y distance).

Show the values used for the distances (you will need them for the runs after the generation, so keep them in the code and show them)

The code should contain the logic of the ants traveling finding their way according to the ACO, the process and the extraction of the final solution.

Report

- First of all, explain how the ACO will work in the problem. Then show the chosen distances between the cities in each given configuration.
- Show the development of the pheromone map (This can be done by a graph or a table of values where i , j is the path from city i to j)and current optimal path for every 10 iteration (do not include iteration 0, start from 10).
- Show the results relating to the set of 10 cities first in a separate section, then the set of 20 cities.
- In each set of cities, include the results asked above per amount of ant agents (for the 1 ant agent run, then the 5 ant agents run, etc…).
- Comment on the progress and optimal solution of the runs of all the sets of ant agents at the end of section of the city set and write your conclusion.
- At the end of the report write if there were any differences when you used the same amount of ant agents on a larger set of cities (10 cities vs 20 overall)?

Deliverables

- Code
- Report