Practical 2 - Evolution of strategy populations

Multi-agent learning 2017-2018

TA's: Maaike Burghoorn, Wouter van 't Hof

Publication Date: 5 January, 2018
Last edited: 5 January, 2018

1 Installation

The programming language used in the framework is C# with the IDE Visual Studio. To complete the assignment, it is advised to download Visual Studio 2015 or a later version. Visual Studio can be downloaded from the following websites.

Windows
`https://www.visualstudio.com/downloads/`
Mac
`https://www.visualstudio.com/vs/visual-studio-mac/`
Linux
For Linux users we refer to Windows or Mac.

The framework can be found here:
`https://drive.google.com/open?id=1z7qWnxtCICu4hBj3dkSZLXhTABRkoucK`

You can load the framework in Visual Studio by opening the sln file.

2 Scenario

In this practical a pool of agents play the repeated prisoner's dilemma with all other agents. Every agent has its own strategy, which he uses to choose an action. Strategies grow or shrink with respect to their relative fitness within the pool. The purpose of the practical is to observe and analyze the

evolutionary dynamics among these strategies. Use the payoff below as the PD game.

	C	D
C	3,3	0,5
D	5,0	1,1

3 Framework

There are two main components of the framework. One being the score table and the second being the population state plot. The score table shows how each strategies fairs against another. This is partly dependent on the parameters, which are nrrounds, nrrestarts and noise. These parameters can be varied with the sliders on the left. The plot represents the simulation of the evolution, where the start state is specified with the sliders on the top right. These sliders represent the fractions of the strategies, they will be normalized on setup. The box on the bottom right is a numeric representation of the plot. The Calculate button will recalculate the score table, this can be useful when you change the score table parameters on the left. The Setup button will reset the replicator dynamics with new proportions, it won't recalculate the table.

4 Assignment

The assignment has to be made in groups of 2. The google groups mentioned in Section 5 can be used to find a partner if needed.
Do not change any code in the framework other than the appointed classes. The theoretical questions have to be written in the Latex template provided in the google drive. Always motivate your answers.

For any questions on the assignment, we refer to Section 5.

4.1 Implementation

The implementation part consists of three components:

- Implementing the strategies

- Creating the score table
- Applying the replicator equation

You can decide what to implement first yourself. But we recommend to implement at least a few simple strategies first to test the functionality of your scoretable implementation. Each of the three components will be briefly described below.

4.1.1 Strategies

The strategies have to be implemented in the Strategy.cs file. Every strategy class has the method getAction. The function should return the integer 0 or 1 (cooperate or defect respectively). For the strategies that depend on the history, their first action should be 0 (cooperate).

There is a total of 11 strategies:

1. All_C : Always cooperate
2. All_D : Always defect
3. Randomly : Choose a random action
4. TFT : Tit-for-tat
5. TFT2 : Tit-for-2-tats
6. Pavlov : Also known as win-stay lose-shift
7. Unforgiving : Cooperate until your opponent defects
8. Smooth FP : Smooth (proportional) fictitious play
9. Prop NR : Proportional no-regret learning
10. Majority : Play the opponent's most played action
11. RL : Proportional RL with averaged estimates

4.1.2 Score table

As mentioned before the score table represents how each strategy fairs against another. This is done by displaying the average reward of a row strategy playing the repeated PD against a column strategy. For example, see the score table for All_C and All_D below (with noise = 0).

	All_C	All_D
All_C	3	0
All_D	5	1

The payoff matrix used in the PD games is shown in Section 2. This matrix is not yet implemented, this is part of your task. Please note that the strategies play a *repeated* PD game, this means that the PD is played for N rounds. This is determined by the *nrrounds* parameter. To obtain a more precise estimate the repeated PD is played multiple times as well. The number of replays is determined by the nrrestarts parameter.
In short, each strategy plays *nrrestarts* amount of times a *nrrounds* repeated PD game against every strategy.
In reality, an environment is not always deterministic. To simulate this, there is a noise parameter. This parameter determines the probability that a random action is selected.

The score table creation has to be implemented in the UpdateLogic.cs in the scoreTable method. You are advised to create helper methods to structure your code.

4.1.3 Replicator equation

Make sure your score table is implemented before you implement this. In order to simulate evolution of the population, the replicator equation is used. This equation grows strategies proportionally to their fitness. The fitness of a strategy is determined by the accumulated payoff they get from playing against the population. In the next time step / generation, the proportion of a strategy is based on its current proportion and its fitness. A complete replicator equation also contains a birthrate, which should be used in your implementation. The replicator equation with birthrate is defined as follows.

$$P_i(t+1) = \frac{P_i(t) * (1 + \beta * S_i(t))}{1 + \beta * S(t)}$$

In this equation, $P_i(t)$ represents the proportion of the population with strategy i at time t, $S_i(t)$ represents the score of strategy i against the population at time t, $S(t)$ represents the average score of the population.

Figure 1 shows an example of the replicator dynamics on a population of AllC, AllD, Random and TFT strategies.

The method you have to implement is the replicator method in UpdateLogic.cs. It is again advised to create at least one helper method.

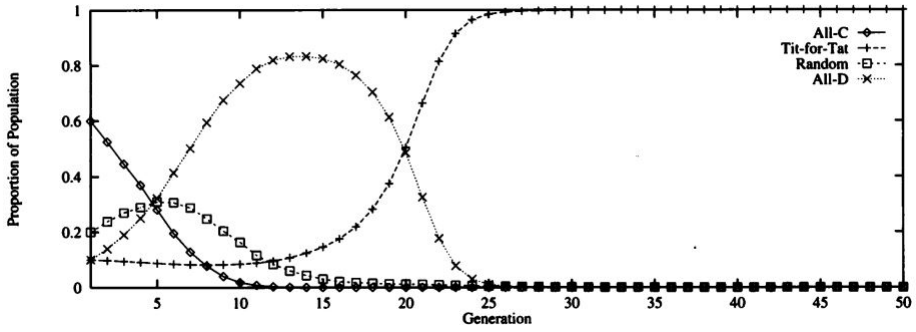

Figure 1: [Flake, 1998], p. 300

4.2 Problems

Question 1

a) Show the score table with all strategies without noise and with a small noise (noise = 0.05). Which algorithms are strongly affected by the noise parameter? Explain your answer.

b) Explain the usefulness of the score table.

c) Why is it necessary to have both the parameter *nrrounds* amd *nrrestarts*?

Question 2

For this question, set the parameters to: noise = 0.05; nrrounds = 200; nrrestarts = 15, birthrate = 0.05. The setups should contain 3 strategies. Motivate your answers.

a) What is the influence of the birthrate parameter on the replicator dynamics?

b) Show a setup where the elimination mechanism is present. [Bouzy and Métivier, 2010]

c) Show a setup where the proportions still fluctuate after 2000 iterations.

Question 3

For this question, run the program with the following parameters: noise = 0.0; nrrounds = 200; nrrestarts = 15, birthrate = 0.1. Set the strategies Randomly, TFT2 and All_D to equal starting proportions and the rest to 0. Run the algorithm multiple times (recalculate the table) with at least 1000 iterations.

a) Show the replicator dynamics of one of the runs and explain them.

b) Does the population always end up with the same strategy with the highest proportion? Explain your answer.

Question 4

a) How does TFT2 differ from TFT? In which situations would TFT2 perform better than TFT and vice versa?

b) Do the learning strategies (FP, NR and RL) perform better than the reactive strategies in this scenario? Explain your answer.

c) Do you think FP or NR would perform better in their pure form compared to the implemented forms? Explain your answer.

5 Support

Any questions regarding the practical assignment can be posted to the Q&A Google discussion group found here:
`https://groups.google.com/forum/#!forum/mal20172018`

NB: You can only join the group using a google account. Unfortunately, your student email does not have access to the google group services.

6 Submit

Deadline: **Friday 19 January 2018, 23:59h.**

The only modified files should be **Strategy.cs and UpdateLogic.cs**. Only

submit these files of the program. The report should be made using the provided Latex template and should contain your names, students numbers and answers to the questions. The length of the report should be 10 pages at maximum.

Submit link: `http://www.cs.uu.nl/docs/submit/`

The following items have to be submitted to acquire a passing grade:
Strategy.cs, UpdateLogic.cs and a report in pdf.

7 Version changes

No changes made yet.

References

[Bouzy and Métivier, 2010] Bouzy, B. and Métivier, M. (2010). Multi-agent learning experiments on repeated matrix games. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 119–126.

[Flake, 1998] Flake, G. W. (1998). *The computational beauty of nature: Computer explorations of fractals, chaos, complex systems, and adaptation*. MIT press.