



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises: Modelling and Simulating Social Systems with Python

Project Report

Evolution of Strategies for Prisoners Dilemma

Gleb Fedorovich, Jan Heldmann, Lukas Schmitt, Fengshi Zheng

Zürich
Nov 2021

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Contents

1	Introduction and Motivations	1
1.1	Historical Background - The Invention of the Prisoner Dilemma	1
1.2	General Prisoner Dilemma	2
1.3	Iterated Prisoner Dilemma	2
1.4	The Tournaments of Axelrod	3
1.5	Evolution of TIT FOR TAT with a Genetic Algorithm	3
1.6	Public Good Game and the tragedy of the commons	4
2	Description of the Model	5
2.1	Agents	5
2.2	Genetic Algorithm	6
2.3	Interaction between Agents	7
3	Implementation	8
3.1	Agents	8
3.2	Interaction	9
3.3	Visualization	9
4	Simulation Results and Discussion	10
4.1	Ring-Structure	10
4.2	Ring-Structure with influence from outside	12
4.3	Grid-Structure	13
4.4	More inputs - more cooperation?	14
5	Summary and Outlook	15
	References	16
6	Appendix	17
6.1	Neural Networks	17
6.2	Genetic Algorithm	18

Abstract

“Bellum omnium contra omnes” is according to Thomas Hobbes the state of nature of human existence. In order to overcome this, a centralized control is needed. However, in the real world collaboration can often form spontaneously.

In this work we analyze how cooperation can evolve in a competitive environment, described here by the prisoner’s dilemma. Using neural networks and genetic algorithm, we are able to show that in order for cooperation to spread, agents have to evolve the strategy Tit For Tat. Analyzing this setting further, we see the importance of local interactions and its connection to cooperation. We then investigate the role of agent-memory. Additionally, we notice that in an cooperating environment, agents unlearn to punish defectors which can give rise to a new cycle of defection. Therefore it can be seen that punishment is important for the formation of a stable society.

Individual contributions

We spread the task according to each team members abilities and preferences and think that everyone contributed in a meaningful and equally important way.

- Gleb Fedorovich: designing of figures, conducting of experiments with ring-agents
- Jan Heldmann: literature research for related work, socio-historical background
- Lukas Schmitt: parts of the implementation and conducting experiments with ring and grid
- Fengshi Zheng: large parts of the implementation and visualization

1 Introduction and Motivations

1.1 Historical Background - The Invention of the Prisoner Dilemma

In the year 1944 the mathematician John von Neumann and the economist Oskar Morgenstern published the book: “Theory of Games and Economic Behavior”. They set the foundation for a new field in mathematics called “Game Theory”. With their book they wanted to approach economical and sociological problems from a new direction [9]. Six years later the mathematician John Nash contributed the definition of an equilibrium point in an N-person Game which is now referred to as Nash equilibrium. If no player can increase their pay-off by changing the individual strategy the Game has reached an equilibrium [1]. The definition of the Nash equilibrium is important for the following problem defined by David Hume in 1739. Two farmers (A and B) meet in harvest season and face a dilemma: The corn of farmer A will be ripe tomorrow and the corn of farmer B is already ripe today. The farmers will not be able to harvest all corn on their own. A has two options: He could cooperate with B and help him with his harvest today or he does not help B, i.e. he defects and B will lose some of his harvest. Farmer B faces the same decision, should he help A tomorrow or defect.

Let’s formalize the problem: For each farmer working on his field on the day of harvest, the owner is rewarded with a utility of two. Each farmer has a cost of one unit for working one day. In addition we assume the farmers face no opportunity costs for helping the other farmer, i.e. their profit for not helping the other farmer is zero. The resulting profit of each farmer depending on their strategy is shown in Table 1. Let’s look at the four different outcomes from the perspective of farmer A:

1. Mutual defection, i.e. A gets a profit of 1 because he had a utility of two and a cost of one for working on his own field.
2. A defects and B cooperates, i.e. A will get a profit of 3 because he just had a cost of one for working one day but a utility of 4 because B was helping him.
3. A cooperates and B defects, i.e. A will get a profit of 0 because he was working both days for himself and for B that costs him 2 units but he also gained a utility of two for working on his own field.
4. Mutual cooperation, i.e. A will get a profit of two because working two days costs him 2 units but his utility is 4 units because he got help from farmer B.

Table 1: Payoff Matrix for the Farmer Example (l) and the general scheme for cooperation (r)

Farmer A	Farmer B		
		Cooperate	Defect
	Cooperate	(2,2)	(0,3)
	Defect	(3,0)	(1,1)

Player A	Player B		
		Cooperate	Defect
	Cooperate	(R,R)	(S,T)
	Defect	(T,S)	(P,P)

From farmer B's perspective the problem is the same way if you just switch the letter A and B. However, the Nash equilibrium is reached when A and B are defecting. To understand why, let's look at the problem from the perspective of farmer B. If A helps him with his harvest, B gets the highest profit by betraying A, i.e. not helping him the next day. If A doesn't help B with his harvest, B also gets the highest profit by not helping A. Independently of A's strategy B gets the highest profit through defection. A knows that and will also defect. The Dilemma is that there would be a better solution for both by cooperating with each other, but because both farmers want to maximize their profit, they will lose some of their harvest [6]. A similar game was invented by two scientist from the RAND Corporation and the mathematician Albert Tucker developed the story of two prisoners, who also needed to decide on defecting or cooperating with their fellow inmate. Since that time, the problem is known as the Prisoner's Dilemma (PD) and has been studied from many different perspectives [5]. The PD shows that the Nash Equilibrium doesn't have to be the perfect solution. There are many fields where we face problems which can be modeled as a PD. One example is protectionism. Two nations could profit more by lowering their tariffs on the goods exported by the other nation. However, the dominant strategy is to have high tariffs because the nation would get more money independently of the other nations strategy. This way it would come to protectionism even though both nations would profit more if the tariffs were lower [2].

1.2 General Prisoner Dilemma

In Table 1 the general scheme for cooperation is shown on the right side. We obtain the PD if the Nash equilibrium is mutual defection. The general scheme for cooperation becomes the PD if the following equations hold:

$$T > R > P > S \quad (1)$$

$$R > \frac{S + T}{2} \quad (2)$$

The second equation ensures that the total utility (addition of the utility of player A and B) is highest when it comes to mutual cooperation [2].

1.3 Iterated Prisoner Dilemma

The Iterated Prisoner's Dilemma (IPD) introduces a future to the PD. Instead of just playing the PD once, the game is played multiple times. Each round the participants are rewarded with the same Payoff Matrix. The player can develop strategies based on the previous behavior of the other player. An example would be that our two Farmers would have to decide each year if they help each other with the harvest. This seems also more likely for the case of two neighboring farmers. Now mutual cooperation can appear if the

farmers change their strategy because the future is the insurance of cooperation. Betraying your neighbor is not a good idea if you could use his help in the next years. Additionally, participants shall not know when the last game is played, because this knowledge reduces the IPD to a normal PD with no future. Defection would be the dominant strategy in this situation [7].

1.4 The Tournaments of Axelrod

The political scientist Robert Axelrod developed a tournament where different strategies played the IPD against each other. In the tournament TIT FOR TAT (TFT) was the winning strategy. There were prior tournaments in which TFT already showed its success. TFT is a very simple strategy, it starts with cooperating and then always copies the last move of the other's strategy. Robert Axelrod defines four factors which made TFT the best strategy:

1. TFT starts with cooperation which is nice and prevents trouble;
2. TFT punishes the defection of its opponent which stimulates cooperation;
3. TFT forgives the opponent when they cooperate again;
4. TFT is a clear strategy which is easy for an opponent to anticipate and makes it realize that cooperation is the best way to achieve a high score with TFT.

It is also worth noticing that TFT can never achieve more points than its direct opponent. Since the PD is a non-zero sum game where the goal is not to beat your opponent, but to use it to gain as many points as possible. Of course there are strategies which beat TFT in a one on one game. Always defecting will give you more points since you gain more points in the first move and afterwards there will be mutual defection. But when it comes to the average score against all strategies TFT would outperform the always defecting strategy by far because mutual cooperation is better than mutual defection. Thus, one can refer to TFT as a robust rule rather than the best rule because it almost always performs the best in different environments. However, TFT doesn't try to exploit other rules which makes it less efficient against exploitable rules like the strategy to always cooperate. There were some strategies in Axelrod's experiment which tried to exploit their opponent, but none of them managed to exploit others in such a way that the reward was higher than the punishment [2].

1.5 Evolution of TIT FOR TAT with a Genetic Algorithm

Robert Axelrod and his colleague Stephanie Forrest wanted to see if a Genetic Algorithm (GA - see appendix 6.2) would develop TFT playing the Iterated Prisoner's Dilemma. From the 62 strategies from his second tournament he selected 8 strategies which were enough to model the tournament environment. To model strategies with the string represen-

tation he considered the three last games as an input. Each game has 4 possible outcomes and therefore $4^3 = 64$ bits were necessary to model the reaction of the strategy to each input. The string representation is also referred to as chromosomes. In the genetic algorithm 20 different chromosomes were initialized randomly and each played the IPD in the environment of the 8 strategies. The GA returned strategies which were very similar to TFT or resembled TFT. In some cases the strategies even outperformed TFT because they learned to exploit cooperative strategies. However, Axelrod points out that these strategies only outperformed TFT in this specific tournament setting. Since strategies always depend on their environment TFT might still be the more robust strategy to survive in different environments [3].

1.6 Public Good Game and the tragedy of the commons

PD can be applied to many real life situation but mostly there are more people than two who interact with each other. In this situation we talk about public good games where people contribute to a common public good. We can come up with the following example: Each of the N participants gets 1 CHF and has the possibility to put the money in a common pot or keep it. The money in the pot will be doubled and each participant will get an equal share of the money from the pot. For $N > 2$ participants the best individual strategy is defecting although the total utility the participants get together would be highest if everyone cooperates. In the results in Table 2 it can be seen that if a player has the decision to cooperate or defect (colored fields) defection is rewarded with the highest payoff independently of the other strategies. We call defectors in public good games free riders. In the real life examples are the people's contribution to climate change, tax evasion or cleaning the shared apartment. Everyone profits from the people who cooperate also the free riders. This makes it tempting for participants to become free riders which can lead to the destruction of the public good which is referred to as the tragedy of the commons. If an iterative public good game is played because of free riders the cooperation decays with time and eventually dies out. To avoid the tragedy of the commons a solution is to punish defectors or to reward cooperation with reputation [8].

Table 2: Payoff Matrix for the 4 Person Public Good Game

Decision				Payoff				Total Money
1	2	3	4	1	2	3	4	all
D	D	D	D	1	1	1	1	4
D	D	D	C	1.5	1.5	1.5	0.5	5
D	D	C	C	2	2	1	1	6
D	C	C	C	3.5	1.5	1.5	1.5	7
C	C	C	C	2	2	2	2	8

2 Description of the Model

This section is dedicated to the general description of the considered model. Since we have already talked in detail over the iterated prisoners dilemma, we start with a brief introduction of agents we used. Next, we illustrate the application of a genetic algorithm for the agents and finish with two models of interaction for the iterated prisoners dilemma. For simplicity, we use the letter 'C' for cooperating and 'D' for defecting.

2.1 Agents

A key feature for all of the agents in our model is determined by a strategy which this particular agent follows. In this section, we demonstrate the list of strategies for Prisoner's Dilemma that we implemented in our project. In Table 3 one can see the description of the following strategies: always cooperating, always defecting, tit-for-tat, neural network and a chromosome-strategy (or in short String-Agent).

Name of strategy	Description
Always cooperating	Cooperating all the time
Always defecting	Defecting all the time
Tit-for-tat	Copying the last action of an opponent
Neural-Agent	C or D depending on the output of a neural network
String-Agent	Obeys a set of rules based on previous actions

Table 3: Description of strategies considered in the project.

Neural-Agent

The Neural-Agent uses a feed-forward-network with one or multiple Input-Neurons (see Appendix 6.1). The number of Hidden-Layers and their dimensions are adjustable as well. The agent's decision is based on the output of the neural network given an input of the last action(s) of his opponent and his own last action(s).

String-Agent

We also implemented a string based agent that is first proposed by [2] and compared it to our Neural-Agent. Inspired by the concepts of chromosome from biology, the strategy of an agent can be fully modeled by a single string which tells it what action to take given every possible IPD tournament history. So this agent's actions are deterministic, compared to our Neural-Agent.

2.2 Genetic Algorithm

The genetic algorithm selects the *fittest* agents after a round of IPD, modifies these agents with a certain probability and replaces the old agents with the fitter ones. In the first variation of this algorithm, we replace every agent with their new children. In later variations not every agent is replaced.

In this model, we use the following techniques.

Fitness-function

In order to determine the fitness of an agent, we add the accumulated points of one round of IPD and scale them. For more sophisticated methods we refer to [10]. We choose the agents for reproduction in a probabilistic manner weighted by their fitness.

Crossover

In order to mimic some aspects of nature, our genetic algorithm chooses always two agents, which act as parents. Then, using parts of these two agents, we produce two children. In case of the neural networks, this procedure is done, by mixing the weights of layers as can be seen in Fig. 1.

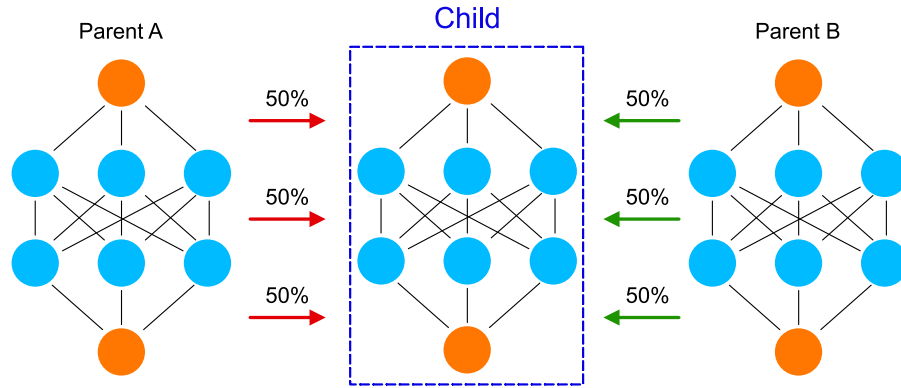


Figure 1: *Crossing over process. Two arbitrary agents act as parents for a child in a new generation. The weights of the child are defined by mixing the weights of its parents.*

For the String-Agent the crossover is simpler. We randomly pick a pivot in the chromosome and let the two chromosomes a, b (length n) swap at the pivot p to get two new chromosomes.

Mutation

Since *selection* and *crossing-over* cannot improve the agents indefinitely, and also to prevent the system from converging into local optima, it is necessary to introduce mutations.

In case of the Neural-Agent, this corresponds to some random perturbation of weights - ergo to a random-walk in parameter space of the neural network.

For the String-Agent case, we iterate through every character in the chromosome and negate it with a small probability.

Global genetic algorithm

One variant of the used genetic algorithm selects the fittest parents in the whole population in a stochastic manner.

Local genetic algorithm

Instead of choosing the parents for new agents in the whole population, we select, in a probabilistic manner, the fittest adjacent agents.

Local genetic replacement To simulate the concept of “the survival of the fittest” in a more dramatic way, we also implemented another version of local genetic algorithm. The idea is to replace the weaker agents with the children of fitter parents.

2.3 Interaction between Agents

Now we need to determine possible ways how our Agents interact with each other. We start with a simple configuration, in the following referred as *Ring-Structure*, where every Agent has only two neighbors - right and left (Fig. 2a)). Extending this model, we place our Agents on a two-dimensional grid, thus, each Agent can interact with either 2, 4 or the 8 adjacent Agents (Fig. 2b)). The idea of considering these two structures is the following: First, we want to study a one-dimensional system, which is easier to understand, and only after that proceed to the more complex two-dimensional case.

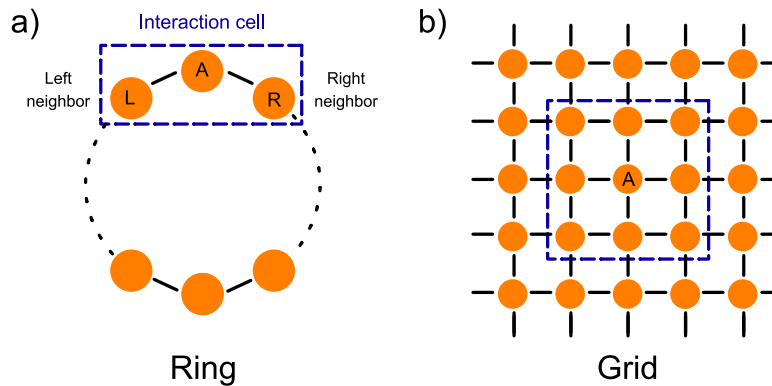


Figure 2: a) Ring structure of agents. Each agent interacts only with its right and left neighbors. b) Grid structure of agents. Each agent interacts only with its 4 or 8 nearest neighbors.

3 Implementation

3.1 Agents

For better code reusability and maintainability, we defined an abstract base class to describe the common behaviors of all agents in our system. So the simulation system can treat the agents equally regardless of their strategies. We then define agents with concrete strategies as derived classes of the base class.

Both Neural-Agent and String-Agent in our implementation have a finite memory of length m . I.e. it can only remember the tournament history of up to m previous rounds. This can be simply achieved by a queue of maximum length m .

Meanwhile, the agent has multiple neighbors to interact with during each IPD tournament, so for each agent we keep a dictionary of memory queues to keep track of the pair-wise interaction histories with its opponents.

The starting action of the agents can be adjusted to C, D, or random.

Neural-Agent

Each layer of the neural network is modeled as a weight matrix and a bias vector according to equation 3. The topology of the network can be defined using an array that specifies the dimension of each layer. For activation function we use `sigmoid`, since it maps the output to a number between 0 and 1. We then identify 0 with C and 1 with D. In the case of a non-integer output, we have two ways to interpret the result. In the *stochastic*-approach, we associate the output with a probability to C or D, while in the *deterministic*-approach, the action of the agent is determined by rounding the output of the neural network.

String-Agent

For an IPD round, the outcomes (CC, CD, DC, DD) can be represented by a number from 0 to 3. So if a String-Agent has a memory of length m , there are only 4^m possible IPD histories when the agent's memory queue is full. For each history case, the chromosome should dictate it to take an action C or D. So String-Agent's chromosome can be represented by a CD-string of length 4^m . More formally, the character a at index i means the agent would take action a under the i th history circumstance. For starting actions (the first m actions where agents' memory queue is not full), we encode the strategies using an additional length- $(2^m - 1)$ string, following the idea from [7]. In practice, we used a 01 array of length $2^m - 1 + 4^m$ to store the two chromosomes.

3.2 Interaction

In order to manage the interaction of each agent we use the package *Mesa*. This library provides us with a scheduler that can step agents simultaneously, gives the possibility to place agents on a grid, and collects the interaction history for us to do statistical analysis. We made our model highly configurable. The parameters of the spatial society such as height/width of the grid, is the grid torus or not (A grid is torus if its top and bottom, left and right are connected. In 1D case it's a ring.), how many neighbors each agent have, how the agents are set up in the initial grid etc.

3.3 Visualization

With the help of *Mesa*, we can record the statistics of the population after each generation. They include but are not restricted to: The average score of the the whole population; The average score of the best and worst 10% agents; The number of different types of agents still standing; The number of cooperating agents (an agent is defined as cooperating if it has more cooperations than defections with its neighbors throughout the rounds in a generation); The feature-vector which characterizes the actions of a Neural-Agent. (will be explained later)

Feature-Vector

In order to characterize the behavior of a Neural-Agent we store the network output for different input-vectors in a so called *feature-vector* (in short FV). E.g. for an agent with memory 1, we have a FV0, which shows the output of an agent if its opponents last action was 0 (C) and a FV1 defined respectively. For agents with more memory, the FV gives the output for a combination of last actions. E.g. FV001 describes the output of the neural network if the last action of the opponent was D and the two prior actions were C.

Web-Application

For the sake of visualization, we have written various functions in order to plot and visualize the results using graphs and color maps. With the help of *Mesa* we created an interactive web-application, where it is possible to change the configuration of the system and see how it evolve over time. The application supports displaying various attributes of each agent such as the scores, types, and feature-vectors.

4 Simulation Results and Discussion

4.1 Ring-Structure

In this section we consider agents placed in a ring-structure. We start with 20 neural agents with random start action and probabilistic behavior. These agents play PD for 20 rounds per generation. After that, we use the global genetic algorithm in order to choose which agents reproduce. In this scenario, we use a neural network with just one input, two hidden layers with three neurons and one output. The results can be seen in Fig. 3.

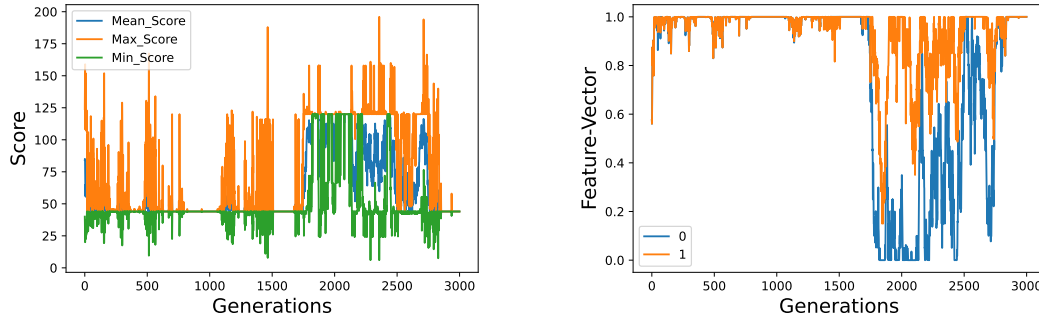


Figure 3: a) Score and b) average FV of a ring with 20 neural network agents. Max- and Min-Score are averaged over the top or bottom 10%, respectively.

In Fig. 3 a) we can see, that at the beginning the average score is around 85. However, after a small amounts of generations, the score starts to rapidly decline until it reaches its lowest score at 40 points. This is the case, when every agent is constantly defecting ($2 \cdot 20 \cdot 1$).

If we compare this with Fig. 3 b), we can see that both feature-vectors are rapidly approaching 1. Therefore everyone defects in every situation.

In the next 1500 generations, we can see little peaks in the average score from time to time. If we compare with the feature-vectors, we see that these peaks correspond to a drop of FV0. This means, that every time some agents try to cooperate, they get defected by the others, which results in a temporarily increase of the average score ($5 + 0 > 1 + 1$).

At generation 1700, something interesting happens. In a short period, the score grows to 120, which is the highest possible average score ($2 \cdot 20 \cdot 3$). For the next 600 generations, it stays that high, even though some small drops can be noticed in between. After that, the score start to decline again. In order to understand what is happening here, we take a closer look at the feature-vectors Fig. 4

At the beginning, we can see, that FV0 declines. Notice, that at this time FV1 is still close to 1. This means, that our agents are effectively playing tit for tat. If the last action of

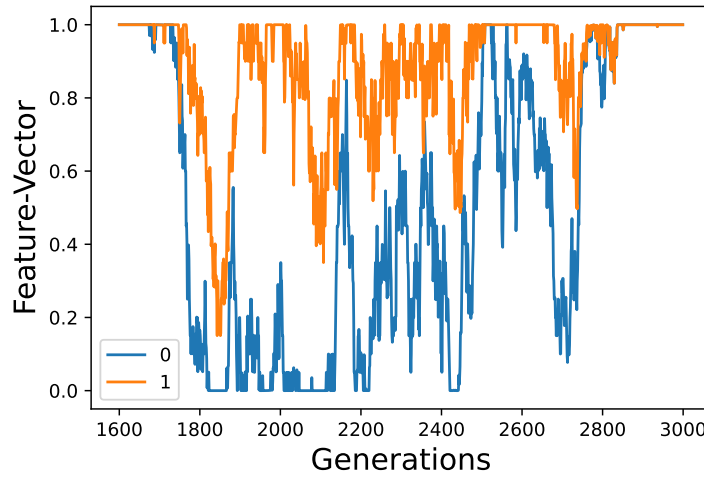


Figure 4: Feature-vectors, starting from generation 1600

their opponent was a cooperation, they cooperate and in case of defection, they defect. Interestingly, FV1 starts to drop a short time after the start of cooperation. We believe that there are two reasons for this. First of all, since the number of defecting agents was heavily reduced, punishing defectors (FV1 being 1) is not an important trait anymore. Therefore the genetic algorithm does not specifically select for this characteristic anymore and it “mutates” away. In fact, it could be even advantageous to forgive (FV1 being 0) other agents. Since the average FV0 is very close to 0. Nearly every agent cooperates in case of a earlier cooperation. Therefore a forgiving agent can initialize cooperation.

Around generation 1900, we can see that FV0 rises. This means, some agents are starting to defect again. This makes sense, since at this point most of the agents are cooperating all the time. As soon as the defectors start to spread, feature vector 1 starts to grow again. Since being able to punish defectors is in this situation an important trait, the genetic algorithm starts to select for these agents again. A race between defectors and defending agents starts. Notice, that the growth of FV0, (aka defecting agents) is larger than the one of FV1. But the defending agents have a head start, since not every agent has forgotten the ability to defend. Therefore a critical amount of defending agents can be reached, before the defectors take over. This point is around 50% of the generation, but depends largely on the spatial distribution of the agents.

The system can therefore return to a situation where everyone is cooperating and playing tit for tat. As long as there are some defectors, the agents do not forget the ability to defend, but around generation 2100, the previous scenario repeats and defectors start to take over. But again, the agents are able to learn fast enough to defend again and cooperating agents

are able to recover. This goes on until at some point not enough cooperating agents remain and we end in a tragedy of the commons.

In the following sections, we want to investigate this behavior further.

4.2 Ring-Structure with influence from outside

As was discussed before, we simulated the evolution of $N_r = 20$ Neural-Agents for $N_g = 700$ generations. Every generation consist of 20 rounds of PD game, and after the game, agents were updated following the genetic algorithm. In Fig.5 a)-b) horizontal dashed red and green lines represents the cases when all the agents defect and cooperate all the time, respectively. When all of agents defects, they earn only $20 \cdot (1 + 1) = 40$ points for every generation. At the same time, if anyone cooperates, the number of points for every agent

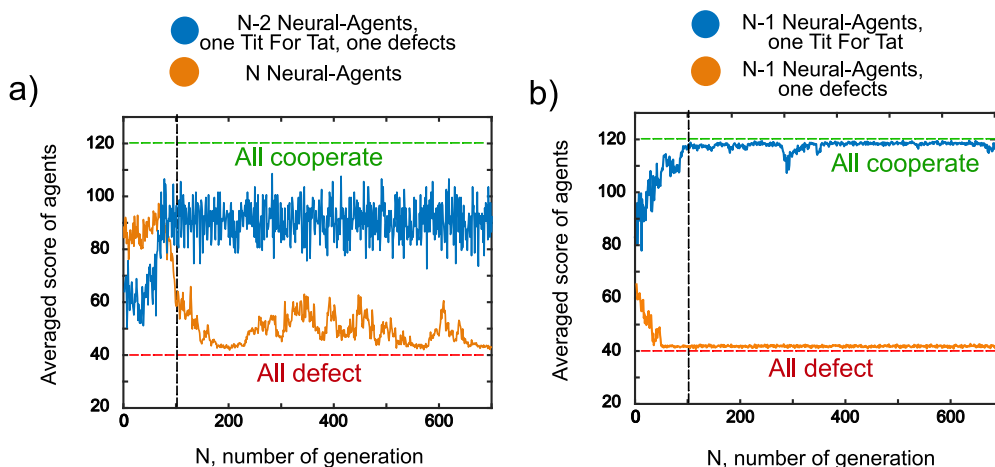


Figure 5: Averaged score of agents depending on number of generation N for different types of agents. a) Orange line represents averaged score of N Neural-Agents. Blue line corresponds to $N - 2$ Neural-Agents, one TFT and one always defects. b) Orange line represents averaged score of $N-1$ Neural-Agents and one always defects. Blue line corresponds to $N - 1$ Neural-Agents and one TFT.

equals $20 \cdot (3 + 3) = 120$ points. We used these limited cases to distinguish whether our agents tend to cooperate or defecting after some time.

In Fig.5 a) one can see that N Neural-Agents after approximately 100 generations mostly tends to defect (orange line). To strengthen this statement, we replaced one of our Neural-Agent to a simple one, which always defects. As can be seen from Fig.5 b), this change leads to complete defecting for all agents. At the same time, if we replaced one of N neural network agents to the one following tit-for-tat strategy, it results in total cooperation of

agents (blue line in Fig.5 b)).

A reasonable question may arise: what would happen if we replace two Neural-Agents by one tit-for-tat and one always defecting at the same time? As was mentioned earlier, these changes separately leads to completely different behavior of neural network agents, namely, in one case, they fully cooperate, in the other - always defect. To answer this question, we simulated a chain of $N_A = 18$ neural network Agents, one tit-for-tat and one simple, placed oppositely to each other. One can see in Fig.5 that in this particular case Agents starts to defect and cooperate almost equally likely, so the average score is approximately 80 points for each generation after the hundredth.

4.3 Grid-Structure

In order to analyze the the role of spatial position for the formation of cooperation, we place the agents on a grid and let them locally interact with each other. After some tests, it became clear, that the global genetic algorithm makes cooperation very unlikely, since cooperating agents cannot accumulate at one position. Therefore the local genetic algorithm is used in the following experiment.

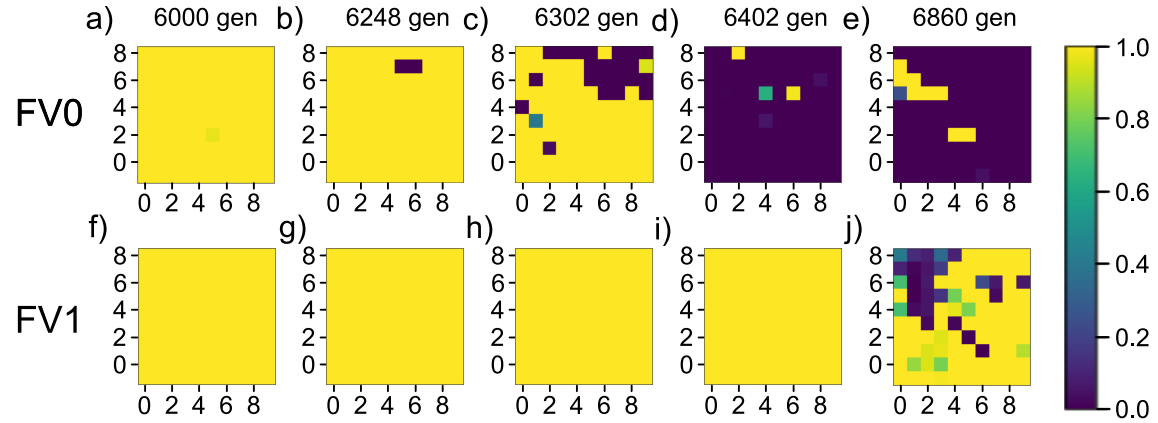


Figure 6: Spatial representation of the feature-vectors. The color yellow corresponds to the value 1, therefore defection and the color violet to cooperation (0). Pictures a)-e) show FV0 starting at generation 6000 and f)-j) show FV1.

We prepare a population of 100 agents on a 10 by 10 grid and let them interact with their 4 adjacent neighbors. The agents play 10 rounds of IPD before the next generation takes over. The spatial evolution of the feature-vector can be seen in Fig. 6.

It should be noted, that in general the agents need a long time to evolve cooperation in this setting. In this run, the agents were basically always defecting for the first 6000 generations. Only after two neighboring agents mutated their FV0 to cooperate, cooperation

could spread. Even then, it was necessary, that the off springs would be placed adjacent to each other. Only under these circumstances a cooperating community could evolve. Note, that FV1 stays 1 for the whole time of this process. Therefore, the new agents play effectively tit for tat. If this would not be the case, the surrounding defectors would strongly profit from the cooperating agents and because of their higher fitness make it impossible for the cooperating agents to spread.

After 100 generations, the Tit For Tat playing agents have taken over (apart from some spontaneous mutations). In the next 400 generations, cooperation stays stable. But, due to mutations, FV1 drops to 0 in some cases. This inevitably leads to defecting agents (see Fig. 6 e) and j)) who profit from the non-defending agents. Note, that since the new defectors descended in most cases from the always cooperating players, they often have a forgiving FV1. Only when they also start to evolve a punishing nature, they have a advantage.

In comparison with the ring, cooperation takes much longer to evolve. The most significant reason for this is the number of interaction between the agents. Interestingly, when cooperation forms on the grid, it is much more stable than on the ring. A reason for this could be that FV1 does not drop as fast as on the ring, since random defecting mutations “remind” more agents to defend against defectors. Also, spreading defectors are more likely to be stopped by the last tit for tat playing agents before they have a critical mass.

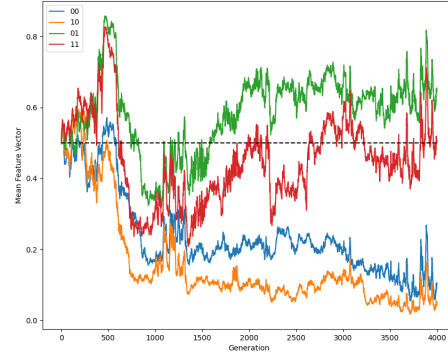
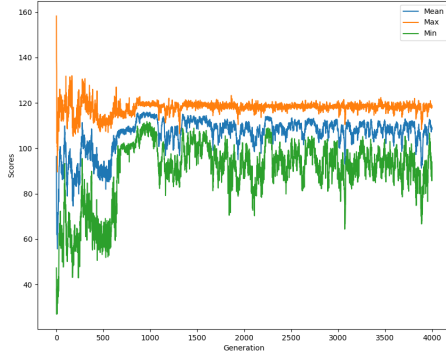
4.4 More inputs - more cooperation?

In order to improve the ability of the agents to form cooperation, the effect of more input variables for the neural network was tested. Instead of receiving only the last action of the opponent, the network now gets the last 2,3,4 actions and the own corresponding outputs. This makes the networks recurrent and therefore able to memorize certain things.

Additionally, the topology of the network was changed. In order to speed up the process of the genetic algorithm, the complexity of the neural network, respectively the number of hidden layers was reduced. This lead to astonishing results. Using the last two actions (own and opponent), the agents were in every setting able to learn to cooperate (see Fig. 7).

By analyzing the FVs, we can infer that the agents learn to cooperate relatively fast and to even forgive defecting agents to some extends. Note, that since we are in the deterministic case, a value smaller than 0.5 means cooperation. We postpone a deeper analysis of the FVs for future works.

With a memory of the last three rounds, the agents could also form a cooperating society consistently. But here the average score is in general a bit lower than in the case of a memory of 2. This trend proceeds for higher amounts of memory 10. An observation of the FVs hints that the lower average score is caused by the agent’s attempt to exploit each other.



h

Figure 7: Score a) and average FVs b) of a grid of 10x10 agents interacting with 4 neighbors deterministic.

5 Summary and Outlook

With the Genetic Algorithm our neural agents are able to learn to cooperate with each other in a one dimensional and in a two dimensional world. In the Axelrod's tournaments the connection between the success of TFT and its characteristic that it gives an incentive for cooperation was made[2]. We could show that the Agents need to develop the strategy TFT in order to get to a state of mutual cooperation. In the one dimensional world mutual cooperation was achieved even faster by adding one TFT agent to the Ring-Structure.

The cooperation was endangered when the neural agents unlearned to punish defectors which made them exploitable. In public good games with humans we can also see a decay of cooperation when there is no punishment of the defecting free riders [8]. Therefore punishing defectors is a key factor for cooperation. This can be also observed in the real world, e.g. in shared apartments and International treaties.

Another key factor for cooperation was to use local interactions (see also Ref.[11]) and the local genetic algorithm rather than the global genetic algorithm. In order for cooperation to emerge, two adjacent agents need to be playing TFT so they can profit from each other and perform better than their environment.

Larger agent-memory led to a more stable cooperation but only to some extends. Increasing the size of the neural network did even worsen the results. As Axelrod already pointed out, one reason for the success of TFT is its simplicity [2].

For future work it would be interesting to investigate the agents' behavior if they are able to move around on the grid and select the agents they want to cooperate with. Also a deeper analysis of the FVs of agents with multiple inputs would be interesting and the impact of the network structure on the behavior.

References

- [1] J. F. Nash, *Equilibrium Points in N-Person Games*, 36(1). 1950, **volume** 322, **pages** 48–49. doi: <https://doi.org/10.1073/pnas.36.1.48>.
- [2] R. M. Axelrod **and** W. D. Hamilton, *The Evolution of Cooperation*. Basic Books, 1984, ISBN: 0-465-02121-2.
- [3] R. Axelrod, *The Evolution of Strategies in the Iterated Prisoner's Dilemma*. Lawrence Davis, London: Pitman, **and** Los Altos, CA: Morgan Kaufman, 1987, **pages** 32–41.
- [4] J. H. Holland, *Genetic Algorithms*, 1. Scientific American, a division of Nature America, Inc., 1992, **volume** 267, **pages** 66–73. url: <http://www.jstor.org/stable/24939139>.
- [5] P. Kollock, *Social Dilemmas: The Anatomy of Cooperation*, 1. 1998, **volume** 24, **pages** 183–214. doi: 10.1146/annurev.soc.24.1.183. eprint: <https://doi.org/10.1146/annurev.soc.24.1.183>. url: <https://doi.org/10.1146/annurev.soc.24.1.183>.
- [6] H. Esser, *Soziologie: Soziales Handeln*. Campus Verlag, 2002, **volume** 3.
- [7] A. Errity, *Evolving Strategies for the Prisoner's Dilemma*, 2003.
- [8] E. Fehr **and** U. Fischbacher, *The Nature of Human Altruism*. **november** 2003, **volume** 425, **pages** 785–91. doi: 10.1038/nature02043.
- [9] J. von Neumann **and** O. Morgenstern, *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press, 2007, ISBN: 9781400829460. doi: doi:10.1515/9781400829460. url: <https://doi.org/10.1515/9781400829460>.
- [10] L. D. Andrew L. Nelson Gregory J. Barlow, *Fitness functions in evolutionary robotics: A survey and analysis*, http://www.nelsonrobotics.org/paper_archive_nelson/nelson-jras-2009.pdf, 2009.
- [11] Dirk Helbing, Attila Szolnoki, Matjaz Perc, Gyorgy Szabo, *Evolutionary establishment of moral and double moral standards through spatial interactions*. 2010. arXiv: 1003.3165.

6 Appendix

6.1 Neural Networks

A neural network can be understood as an emulation of the human brain. It consists of many single neurons, which are connected to each other. Each neuron is able to do simple calculations based on the other neurons. In this project we used a *Feed-Forward-Network* (see Fig. 8).

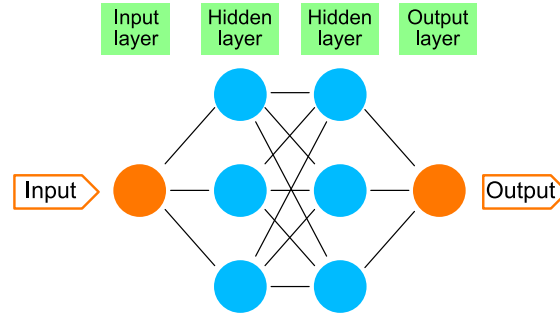


Figure 8: Schematic representation of a *Feed-Forward-Network*.

Neuron

The building-block of the neural network is the neuron. Each neuron can *weight* a vector of input-data. To do this, it multiplies every element of the vector with a specific number w_i , that depends on the strength of the connection between the neurons. After that, it sums over every scaled element and is able to add or subtract an intern scalar b , also called *bias*. At the end, it applies an activation-function σ to the sum (see Fig. 9). This result is then used as an input for the next neurons.

There are many possibilities for activation-functions, but in this project we use the *sigmoid*-function, since it maps the output to a value between 0 and 1. This output is going to correspond to either *cooperating* or *defecting*.

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

Layers

As seen in figure 8, the networks is divided in different layers. One can summarize the calculation of all neurons in a layer as a matrix multiplication (Eq: 3).

$$\mathbf{y}_{out} = \sigma(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}) \quad (3)$$

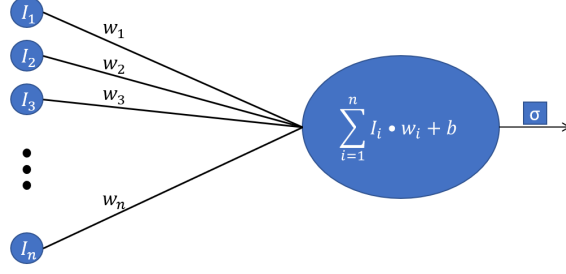


Figure 9: Schematic representation of a neuron.

y_{out} is here the output of a layer, written as a vector, \mathbf{W} is a matrix, consisting of the weights of each neuron in the layer and \mathbf{b} is a vector of the biases. σ is defined over the elementwise application of the activation-function σ .

6.2 Genetic Algorithm

Genetic Algorithms (GA) were inspired by evolution in nature. Invented in the mid 1960s by John H. Holland the GA resembles natural selection and sexual reproduction. In Computers natural selection is modeled by fitness functions. The creature performing well achieves a high fitness score and is more likely to sexually reproduce itself. The generated offspring includes genetic code from both parents in case of bisexual reproduction. Additionally, there is also a probability of mutation in the generated offspring. The bisexual combination of well performing genetic codes outperformed asexual reproduction where the genetic code only mutated but didn't mix with other genetic codes. The genetic code was modeled by Holland similar to a string of DNA in the so called classifier system. Any written computer program could be modeled with a classifier system which is a string representation of 1's and 0's. If the program has a certain characteristics the corresponding string position is set to 1 otherwise to 0. If a classifier should for example recognize a human the corresponding bits to the attributes "mammal", "4 extremities" and "Omnivore" would be set to 1. Bits corresponding to attributes like "reptile", "8 extremities", "Carnivore" would be set to 0. The Algorithms developed through this process can solve very complex problems which can not even be solved by humans analytically. Examples for such problems are huge infrastructures like gas pipelines or the energy grid.[4]

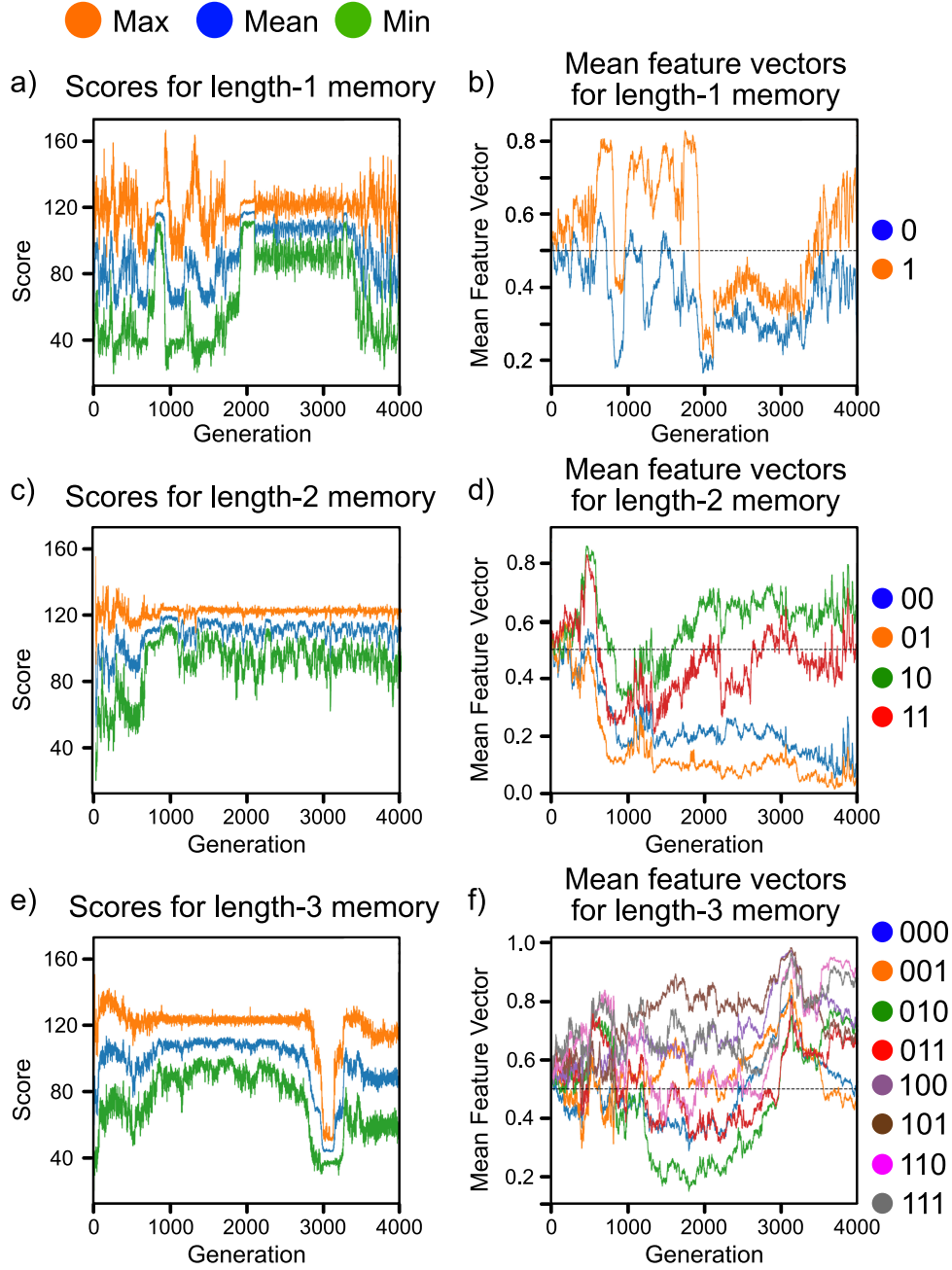


Figure 10: Plots of scores and feature vectors for different memory sizes. The binary numbers i in the label of the feature vector plots denote the i th element of the feature vector. For example, in the case of length-3 memory, “001” means the opponent has cooperated for two rounds but defected in the latest round.



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

First name(s):



With my signature I confirm that

- I have committed none of the forms of plagiarism described in the '[Citation etiquette](#)' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Signature(s)

	
	
	Fengshi Zheng

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.