Introduction to AWESOME

Report by

Jannik Petersen
Matriculation number: 3461587
jannik.petersen@stud.uni-heidelberg.de
July 19, 2019

Artificial Intelligence for Games

Prof. Dr. Ullrich Köthe

Abstract

This report explains a multiagent learning algorithm, which converges in self play to a Nash equilibrium and learns a best response against stationary opponents. AWESOME makes much less assumptions as previous learning agents. It was the first algorithm which is guaranteed to achieve these two properties 1. Converging to Nash equilibrium in self-play, 2 learn best response against stationary opponents.

Contents

Introduction:

First we will explain some basics, so that we stake out the conditions for AWESOME.
Which includes Nash equilibria, learning in games in general, the setting (which means we are defining a stage game) and how a stage game should be played.
Then extract two properties form these conditions, first that the algorithm converges to a Nash equilibrium in self-play and second, that the algorithm learn to play a best response strategy against stationary opponents. These properties are minimal, so that every multiagent learning algorithm which fails one of these is unsatisfactory.
We point out why AWESOME is more preferably than another learning agent called WoLF-IGA, which has to make some more assumptions to learn properly, which makes WoLF-IGA less general.
Nevertheless, AWESOME also have to make a few assumptions.
Like it's only able to play repeated games, for example one of the 2-person games (rock-paper-scissors), and it assumes that the game is learned already. So AWESOME can focus on learn to play optimally. And last it assumes that the algorithm can compute a Nash equilibrium.

Which leads us to the methods of AWESOME to achieve both of these minimal properties.
So we are introducing AWESOME's null hypotheses.
There are two null hypotheses, between which AWESOME cycles via a type of state machine.
This means AWESOME uses them to check if it should play the equilibrium or a best response against a stationary opponent.
When both of these hypotheses are rejected AWEOSME restarts itself completely.
The truth of a hypothesis is tested at the end of each epoch, so AWESOME didn't adapt too fast.
To guarantee the convergence over time the epoch length is increased and the criteria for rejecting a hypothesis are tightened.
AWESOME also restarts, when it detects that its own actions signal non-stationary, for synchronization with the other agents.

Later we show the implementation of AWESOME and explain the code with the previous discussed logic.
Then we will compare AWESOME against another learning agent named Fictitious-Play in some meaningful cases.

Basics

Learning in games:

Learning in games is a key capability in AI research, because it allows you to train an algorithm in a virtual world without having to be mindful of all real world conditions. In addition the training phase can be repeated very easy, which gives the possibility to make many different training attempts.
Learning is especially important in multiagent settings, where the other agent's behavior is not known. The agents can learn to play against copies of itself, which is called self-play.

Learning in games can be splitted into two important aspects, first learning the game itself and second learning how the opponent is behaving and play optimally against him.

Learning the game means, that the agent don't know the game structure, the rules and has to explore the game itself.
Learning how to play optimally against opponents is the second aspect.
We assume that the agent know the game already and can focus on that the agent just need to learn how to play optimally against its opponents.

Nash equilibria:

As mentioned before in the introduction, AWESOME needs to compute a Nash equilibrium of every game it is playing.
Basic idea of a Nash equilibrium is to characterize a rational decision for players in which none player knows how the others choose their decisions. The player can only consider how the others might play.
A Nash equilibrium describes game situations between two or more player in which none of the players can improve by changing their choice of actions. Nash equilibria are given in non-cooperative games, which means that the players compete against each other. It is therefore also said that a Nash equilibrium is in a sense 'stable'. Every player agrees with his strategy, even in hindsight and would choose these actions again.
It comes to a Nash equilibrium, when all players play the best possible response against the behavior of the opponents. So a Nash Equilibrium is a combination of strategies of more than just one player.
A strategy can consist of more than just one action.

Basically we differentiate Nash equilibria into two different types, pure strategies and mixed strategies.
In pure strategies the player will always responds with the same best response to the chosen strategies of the opponents strategies. These types of Nash equilibria are characterized by constant or lesser reward for all players. Which means a player cannot gain more reward by only changing his own strategy.
In contrast to pure strategies the player makes no direct decisions in mixed strategies. He leaves his choice to a probability distribution over his action set. The mixed strategy is a Nash equilibrium when none of the players can improve his reward by only changing his own strategy.

Every finite game has a Nash equilibrium in mixed strategies. While in pure strategies it is possible that there is no equilibrium. [1]

Setting & playing a stage game:

The multiagent learning algorithm in this paper only deals with repeated stage games. That's why we have to define the environment for AWESOME.
For our discussion we use finite stage games with a finite number of agents. Finite stage game means, that there is a limited number of actions an agent can choose from.
Which leads us to: a stage game is defined by N agents who play repeatedly. Each agent has its own action set A_i for agent i, which in general not must be the same for all agents. Furthermore each agent has its own utility function u_i. The utility function describes the usefulness of each played action.
In a stage game an agent chooses a strategy to play. A strategy is a probability distribution π_i over the action set A_i of agent i. It's indicating which action the agent will choose in this round. It is important to notice, that all agents choose their actions independently. Similar to the Nash equilibria a strategy can be divided into two different types. First the pure strategies, second mixed strategies. Pure strategies are characterized by the fact that all their probability mass, of the probability function, relates to just one single action. Mixed strategies consists of more than one action, each one with an own probability. Which means the agent plays a different distribution every epoch. Every non-pure strategy is a mixed strategy.

As mentioned above each agent choose one action in each round, many rounds sum up to one epoch. And a game consists of many epochs, since it is a repeated game.

There is an example of an unnamed stage game. It's a two-player three-action game, which AWESOME is able to play.
Each block of figure 1 consists of two numbers, which represents the reward for a player if he chooses this action and win the round. The second number is for player One, the one who chooses the actions on the upper side of the diagram. The first is for player Two.

	A	B	C
A	1, 2	2, 1	3, 0
B	2, 1	1, 2	4, 0
C	0, 3	0, 4	2, 2

Figure 1, example of a stage game

Most of the time we will focus on the first two actions A and B.
There is a Nash equilibrium in mixed strategies for the actions A and B, because they are complete symmetrically.

If we assume that player one (the player on the upper side of the diagram) choose action A the best response for the second player is to play action B. Because his reward will be two, and vice versa.

There is also a Nash equilibrium for a pure strategy when we take in the third action C.
We see that, when both players choose action C, none of the two players can increase their reward by changing only his own action.

Figure 2 describes the case that both players are playing the mixed strategy equilibrium, and since both actions are symmetrically the probability of each action is 50%.
When both players are completely rational, none of them would change their probability distribution. But when we are playing against a bit less clever opponent, who chooses his probability function as described in figure 3, we can see that we should always choose action A. Just because we are gaining the highest reward in most cases by choosing action A.

		50% A	50% B	C
50%	A	1, 2	2, 1	3, 0
50%	B	2, 1	1, 2	4, 0
	C	0, 3	0, 4	2, 2

Figure 2, example of a mixed strategy Nash equilibrium

		49% A	51% B	C
100%	A	1, 2	2, 1	3, 0
	B	2, 1	1, 2	4, 0
	C	0, 3	0, 4	2, 2

Figure 3, example of a less clever opponent

AWESOME faces this problem, because AWESOME plays the equilibrium or adapt when it detects that everybody is playing stationary.
And in the case described above AWESOME is playing against a stationary opponent with a mixed strategy, which means AWESOME will adapt to its best response.

Method

Properties and removed assumptions:

The example form above shows why we want, that a satisfactory multiagent learning algorithm should learn to play optimally against stationary opponents or converge to a Nash equilibrium in self-play.
From this we extract two minimal properties for our agent.
1) Learning to play optimally against stationary opponents, or even opponents, which might become stationary, or called rationality
2) Converge to a Nash equilibrium in self-play.
Intuitively, the properties formalize that an agent should learn a best response when possible.
An agent that at least fails one of these minimal properties is unsatisfactory, that's why they are our absolute minimum we want to achieve.

However, long time there has been no algorithm that was able to satisfy both of these properties in general repeated games. The best one, who came closest to achieve both of these properties, was WoLF-IGA. Described in the paper "Multiagent learning using a variable learning rate" by Bowling & Veloso, 2002.
The so called WoLF-principle is varying the learning rate of the multiagent. That's why the agent is learning fast when he is losing and slowly when he is winning. The name WoLF means, win or learn fast. Which is a very natural method, to learn quickly when the agent realizes he performs too poor.
The second part is IGA, which simply stands for infinitesimal gradient ascent.

AWESOME's advantage is that it removes the following assumptions from WoLF-IGA completely:
a) there is a maximum of two players
b) each player has just two, or less, actions to choose from
c) the agent can observe the strategy of the opponent
d) the agent can use the gradient ascent of infinitesimal small step sizes
Not to forget, that AWESOME also fulfill the two minimum properties, for every finite number of actions and/or agents and AWESOME doesn't need to observe the complete strategy of its opponents. The agent can deal with just recognizing the played actions. [2]

AWESOME:

This section will present the methods to implement the basic ideas described before.

Notice that AWESOME makes some assumptions, like:
a) it only deals with repeated games
b) it assumes that the structure of the game is known
c) the agents can compute a Nash equilibrium of the stage game

To achieve this AWESOME has two null hypotheses.

The algorithm of AWESOME is also self-aware, which means AWESOME is able to detect the continuity of its own actions and restart itself completely, if it detects nonstationarity.

Null Hypotheses:

A null hypothesis is the assumption about the population to be tested in the context of a hypothesis test. Or in other words, a null hypothesis gives a general statement of whether two measured phenomena are related to each other. A null hypothesis is always assumed to be true, until the contrary was shown.

The basic idea for AWESOME is trying to adapt to the other agents when they appear to be stationary, but otherwise retreat to a precomputed equilibrium strategy. That's why AWESOME maintains at every point one of these two null hypotheses.

As shown in figure 4 AWESOME will start and assume, in the first null hypothesis, that everybody is playing the precomputed equilibrium strategy for the stage game. When the agent detects, that it's not true AWESOME reject the equilibrium hypothesis and moves to the stationary hypothesis, which means that AWESOME assume that all other agents are playing a stationary strategy. When the second strategy is rejected too, AWESOME will restart itself completely.

The current hypothesis is evaluated in every epoch. [3]

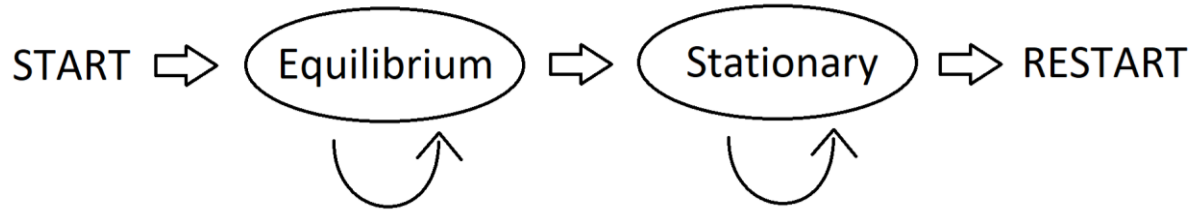

Figure 4, null hypotheses structure

Equilibria:

AWESOME's goal is to learn an equilibrium of a repeated game. The basic advantage of learning an equilibrium of a repeated one-shot game, which is a game that naturally is not repeated, is that the equilibria are natural and simple, always exists and are robust to changes in the averaging schemes. (Indirect quote from original paper, S. 3)
A repeated one-shot game is also called a supergame, which are divided into finite games and infinite games.

But there are also equilibria in general repeated games and it is possible that these equilibria are completely different from the one-shot repetition equilibrium. These equilibria are based on the competition between the players, because the players are tuning their future gameplay behavior on the currently chosen actions.

First Approach:

Because AWESOME can't observe the other players actions, just the played actions, we have to specify how to reject a hypothesis.
Since AWESOME gains its 'rewards' in epochs it can compare, at the end of each epoch, the actual played actions against the equilibrium, which was precomputed before. Usually the algorithm assumes that the actions are chosen because of the strategy, when the distance between the two action distributions is small enough.

For this purposes we define:
P_h^a: percentage of action a which comes from the historical distribution
P_π^a: percentage of action a which comes from the mixed strategy
$P_{h^{prev}}^a$: percentage of action a which comes from the historical distribution, without current epoch

As mentioned above AWESOME computes:

$$\max_{a_i} \left| P_h^{a_i} - P_\pi^{a_i} \right| < \epsilon_e$$

For accepting or rejecting the equilibrium hypotheses, ϵ_e is a threshold for acceptance.
Analog for the stationary hypothesis:

$$\max_{a_i} \left| P_h^{a_i} - P_{h^{prev}}^{a_i} \right| < \epsilon_s$$

Where we are computing the difference between the percentage of actions chosen from the historical distribution and the actions of the history without the current episode. We are also using a different threshold ϵ_s.

This first approach keeps the number of iterations (games per epoch) constant, as well as our two thresholds ϵ_e and ϵ_s for rejecting a hypothesis.
The main problem with this is, that there is a constant probability that the chosen actions do not appear from the equilibrium strategy. Which leads AWESOME to eventually restarting itself, so AWESOME will never converge to its equilibrium.
And second AWESOME can't detect if a strategy is drawn from the precomputed equilibrium or not.

Fixing & Solution:

Both problems can be fixed easily.
By adjusting the thresholds ϵ_e and ϵ_s we can change the maximum distance to choose an action from.
When we decrease the thresholds and increase the number of iterations for each epoch, we gain the desired effect of tightened boundaries. So the probability that an action is chosen from a non-hypothesis distribution will decrease. As shown in figure 6.

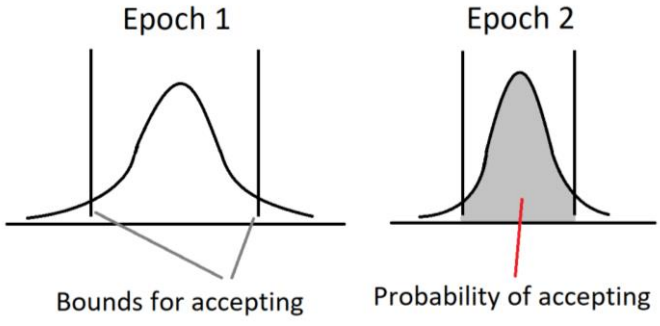

Figure 5, bounds of acceptance

The Implementation:

In this section I present the implementation of the AWESOME algorithm, but first I have to explain some of the needed variables and functions, which are used in the implementation.

π_p^*: player p's equilibrium strategy
$APPE$: 'all players playing equilibrium', true when assuming equilibrium hypothesis
APS: 'all players stationary', true when assuming stationary hypothesis
β: true if equilibrium is rejected just before, to get an extra chance
t: index of epoch
ϕ: current strategy of the AWESOME algorithm
$\epsilon_e^t, \epsilon_s^t, N^t$: thresholds and epoch length for epoch t
a: is an action
μ: difference between best and worst outcome
$|A|$: number of actions
n: number of players
V: return the utility of a played action or strategy, for a the strategy of the opponent

First the algorithm starts to compute the equilibrium strategy for each player p. Then AWESOME will start its infinity loop, because theoretically AWESOME will play the game forever, and initializes all its action lists, hypothesis and the epoch counter.
Next, while APS is *true*, it will play N^t times its strategy ϕ and update the history for each player.

Since $APPE$ is *true* AWESOME will compare the strategies against the currently played actions of player p (equilibrium hypotheses test). If AWESOME detects a non-equilibrium strategy, it will reject the equilibrium ($APPE := false$), and choose a random action as new strategy.
Since $APPE$ is *false* now, AWESOME will check the stationary hypotheses for now and eventually reject it as well and restart or choose a better action a to play.

```
AWESOME()
 1. for each p
 2.   π*_p := ComputeEquilibriumStrategy(p)
 3. repeat {// beginning of each restart
 4.   for each player p {
 5.     InitializeToEmpty(h_p^prev)
 6.     InitializeToEmpty(h_p^curr)
 7.   }
 8.   APPE := true
 9.   APS := true
10.   β := false
11.   t := 0
12.   φ := π*_Me
13.   while APS { // beginning of each epoch
14.     repeat N^t times {
15.       Play(φ)
16.       for each player p
17.         Update(h_p^curr)
18.     }
19.     if APPE = false {
20.       if β = false
21.         for each player p
22.           if (Distance(h_p^curr, h_p^prev) > ε_s^t)
23.             APS := false
24.       β := false
25.       a := arg max V(a, h_-Me^curr)
26.       if V(a, h_-Me^curr) > V(φ, h_-Me^curr) +n|A|ε_s^{t+1}μ
27.         φ := a
28.     }
29.     if APPE = true
30.       for each player p
31.         if (Distance(h_p^curr, π*_p) > ε_e^t) {
32.           APPE := false
33.           φ := RandomAction()
34.           β := true
35.         }
36.     for each player p {
37.       h_p^prev := h_p^curr
38.       InitializeToEmpty(h_p^curr)
39.     }
40.     t := t + 1
41.   }
42. }
```

Figure 6, implementation of the AWESOME algorithm

Comparison:

Now we are comparing AWESOME against an algorithm called fictitious play.
Fictitious play is a very simple algorithm for learning in games, it just plays the best response to the opponent's play, which is calculated form the historical distribution. The algorithm can be applied to random games, because it only requires to see the opponent's actions, instead of the full strategy, and fictitious play will converge to a best response for all stationary opponents.

For comparison we use the game Rock-Paper-Scissors, with an opponent who plays the mixed strategy 0.4-0.6-0 (rock-paper-scissors). Which means he chooses with a probability of 40% the rock.

In the diagrams of figure 7 we see that we compare 3 different types of agents.
1) the empirical distribution, for the complete history of actions
2) the empirical distribution, only the last 500 round
3) the mixed strategy (only AWESOME)

In the first graph we see, that fictitious play converges to the best response really fast, just because the algorithm is optimized to play against stationary opponents. While AWESOME plays the equilibrium strategy and reject it at the 700th epoch and then plays the best response. For both of AWESOME historical distributions it also takes many rounds to converge. Because of the other actions it played before it rejected the equilibrium hypotheses.

In the second graph are the results for a situation in which each agent plays against a copy of itself.
Both algorithm perform quite well, while fictitious play converges to the equilibrium, AWESOME has never rejected it. But both 500 round moving distributions cannot converge, because they choose from a mixed strategy a fixed number of times. And the empiric distribution will not equal with the distribution of actions.

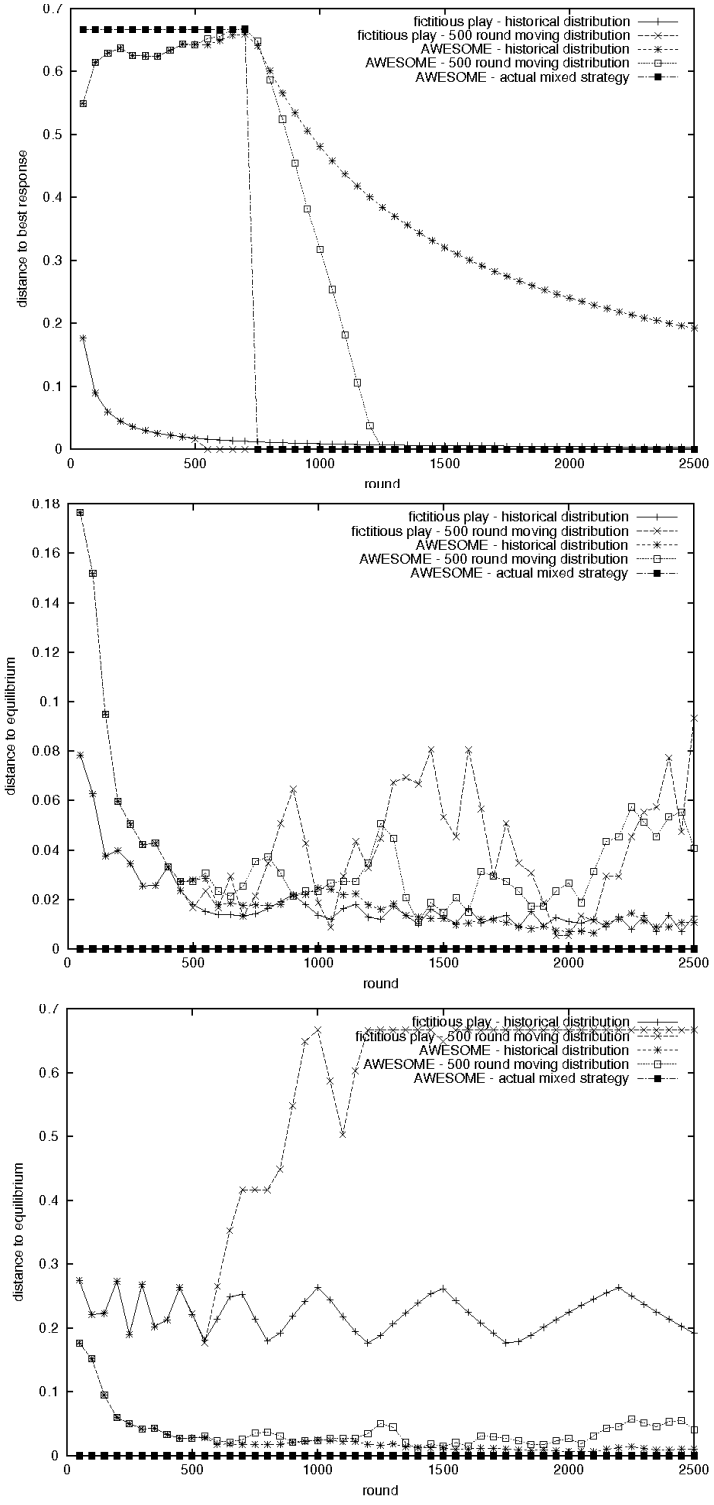

Figure 7, comparison of AWESOME and fictitious play

The third graph shows result for a different game named Shapley's game, see figure 8. With this game we can illustrate how fictitious play will fail in self-play situations.
Both fictitious play agents start to cycle, the length of the cycles increases and that's why the 500-round moving distribution will never converge, because it probably set all its probability on a single action is therefore quite far away from the equilibrium.
AWESOME for his part is still playing the equilibrium strategy.

	0, 1	0, 0	1, 0
	0, 0	1, 0	0, 1
	1, 0	0, 1	0, 0

Figure 8, Shapley's game

Conclusion:

As mentioned in this report, AWESOME is the first learning algorithm, which provides both minimal desired properties for the discussed multiagent settings. AWESOME is able to converge to a Nash equilibrium in self-play situations and otherwise plays a best response against stationary opponents.
At any point in the ongoing game AWESOME assumes one of two null hypothesis, first that everybody is playing the equilibrium, or second that everybody is playing stationary. Otherwise AWESOME will restart itself completely.
AWESOME's criteria are evaluated at the end of each epoch.
The length of the epochs is getting longer and longer, and the thresholds for rejecting a hypothesis are tightened.
AWESOME was a theoretical work, but experiments has shown, that AWESOME performs quite well in actual game situations.

References

[1] Osborne, Martin J., & Rubinstein, Ariel (12 Jul 1994), A Course in Game Theory. Cambridge, MA: MIT. p. 14.

[2] Stockburger D.W. (2007), Hypothesis and hypothesis testing, *Encyclopedia of Measurement and Statistics* (editor—Salkind N.J.), Sage Publications.

[3] Bowling, M., & Veloso, M. (2002), Multiagent learning using a variable learning rate. *Artificial Intelligence, 136*, 215–250.

[4] Conitzer, Vincent & Sandholm, Toumas (2007), AWESOME: A General Multiagent Learning Algorithm that Converges in Self-Play and Learns a Best Response Against Stationary Opponents