Go correlated Q-learning, scooooore!

Khalil Nouili

Abstract

This paper aims to replicate some results from "Correlated Q-Learning" by Amy Greenwald and Keith Hall (Greenwald, Hall, and Serrano 2003) [Amy03]

s

Introduction

Learning to predict from past experience is an ever evolving topic. Until today there are new published papers that suggests cutting edge techniques in order to deal with this problem. In this paper, I will be replicating the results in figure 3 of "Correlated Q-Learning" paper by Amy Greenwald and Keith Hall (Greenwald, Hall, and Serrano 2003) while comparing the results between the different methods. This paper contains:

- Problem description: understanding the soccer environment.
- Theories behind these experimentations.
- Multi-agent Q-Learning.
- Experiments.
- Conclusion.

Environment description: Soccer game

The soccer game is the game environment in which we are doing our experiments.

The game rules

The environment represents 2 players, finite space, zero sum space. The goal of each player is to put the ball inside their opponent's goal. The game is over once one player scores a goal.

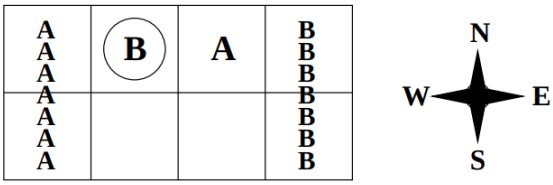

Figure 1: Soccer game environment

In this part, I am going to formulate this problem into a Markov decision process for a better understanding:
Agents: The player A and player B
States: We are in a 2*4 grid space. The player's position is defined by a number between 0 and 7 mapping each grid position.
While implementing Q-learning, the states are :

- player's position.
- opponent's position.
- boolean value describing if the player has the ball or not.

While implementing friend-q, foe-q and correlated-q, the states are :

- player's position.
- opponent's position.
- boolean value describing if the player has the ball or not.
- player's previous action.

Actions: Moving to the north, east, west, south or standing still or left.
The players' actions are executed in random order. If this sequence of actions causes the players to collide, then only the first moves. But if the player with the ball moves second, then the ball changes possession.
Rewards: Rewards are received once terminal states are reached by scoring a goal. reward = 100 if player score else reward = -100.

Implementation

Each player's initial position is represented as the figure XX suggests. Hence the initial position of the ball is randomly

chosen.

Pseudocode

Algorithm 1: Soccer game environment step function

Input : both players' positions, both players' actions
output: rewards, new players' positions, whether the game is over or not
if *random$_n umber < 0.5$* **then**
  player A goes first
  if *not new position A == new position B* **then**
    player B keeps his new position
  end
  else
    player B stands still
    player A has ball
  end
end
else
  player B goes first
  if *not new position B == new position A* **then**
    player A keeps his new position
  end
  else
    player A stands still
    player B has ball
  end
end
if *(position player A == goal player B and player A has ball) or (position player B == goal player B and player B has ball)* **then**
  reward player A = 100
  done = True
  return output
end
else if *(position player B == goal player A and player B has ball) or (position player A == goal player A and player A has ball)* **then**
  reward player B = 100
  done = True
  return output
end
else
  return output
end

Theories behind these experimentations

Nash equilibrium

Every player in an non-cooperative game opts for a strategy to play with. Nash equilibrium is when both players are better with not switching their strategies given that they know each other strategy.

Correlated equilibrium

The correlated equilibrium represents the probability distribution of the action space. An agent that uses the correlated equilibrium to optimize his strategy and the action to take, takes into consideration the other players actions. In order to calculate a correlated equilibrium, solving a linear program representing the states is needed.

Multi-agent Q-Learning

The idea behind multi agents Q-learning is updating both q and value functions for all the multiple agents at the same time using states and actions reached or done at each step. Technically speaking, an equilibrium function f is fed to the algorithm that computes both the value and Q functions given the Q-table.

Algorithm 2: Multi-agent Q learning

Input : total episodes
output: value functions, Q functions
for *episode = 1...total episodes* **do**
  $a_1 ... a_n \leftarrow$get the new actions to execute for each agent
  $r_1 ... r_n, s_1 ... s_n \leftarrow$the rewards and states for each agent after executing the actions A_t
  for *i = 1 to n* **do**
    update Value function using the equilibrium function f with Q function of all other agents as input
    $Q_i(s_i, a_i) = (1-\alpha)Q_i(s_i, a_I) + \alpha * (1-\gamma) * r_i + \gamma V_i(s_i)$
    decay α
  end
end

Q-learning

The agent here aims to maximize the total reward by performing the set of actions generated from the cheat sheet. This value is called eventually $Q - value$ defined by:

$$Q(s, a) = r(s, a) + \gamma max_a(Q(s', a))$$

Initially it starts with making a random assumptions for all Q-values related to each state and action, after some time it will converge to the optimal policy.
In practice the policy update function is written :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(s_t, a_t)]$$

with α as the learning rate

Friend Q learning

Friend Q is an updated version of Q-learning where the agents have to take into consideration the other agents' strategies and behaviours as well. This will be done by adding another dimensions in the agent's state containing other agents' actions.

Foe Q learning

Foe Q learning is an updated version of friend Q learning. The agent here takes other agents' strategies into consideration as well. The only difference is the formulation of the equilibrium function f. Here the agent follows minmax approach to maximize his rewards even in the worst situations.

Correlated Q learning

The correlated Q learning units all the approaches seen in the previous methods. Every agent is allowed to see all the actions and rewards of other agents and uses it as a way to optimize its strategy. As suggested by Correlated Q-Learning paper, there are 4 variations of Correlated Q learning. By the choice of the correlated Q learning variations, we choose the equilibrium function as well.

Experimentations

In this part, I am going to talk about the 4 experimentations done to replicate the paper's result as well as a comparison between my results and the actual results.

Q-learning

Experimentation setup:

- α = 0.9.
- α_{min} = 0.001
- α_{decay} = 0.9995
- γ = 0.9
- ϵ = 1.0
- ϵ_{min} = 0.001
- ϵ_{decay} = 0.9995

Experimentation result :
The graphs are a little bit similar as the paper suggests: In the beginning, there is no sign of convergence until the 600000 episode.
The gap between the Q values at each step starts to get smaller but not small enough to converge.
The difference between both graphs is that the difference between the Q-values is smaller in my graph and it looks like it's going to converge if I give it more time.

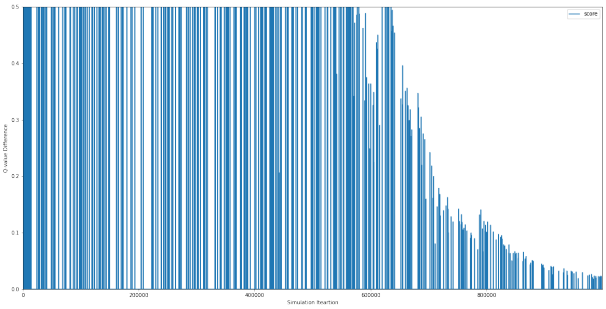

Figure 2: Replication of figure 3-d results from Greenwald's paper

Friend Q-learning

Experimentation setup:

- α = 0.9.
- α_{min} = 0.001
- α_{decay} = 0.9999905
- γ = 0.9
- ϵ = 1.0
- ϵ_{min} = 0.001
- ϵ_{decay} = 0.9995

Experimentation result :
I tried to mimic the paper's graph as much as possible with tweaking the hyper-parameters. Both graphs shares the fact that their Q-values converge very fast, but mine converge slower. As the name of the algorithm suggests, the agent here will assume a potential collaboration with the agent to pick the best strategy that suits it, and for that the agent will pick the perfect counter attack for the best action that the opponent might take.

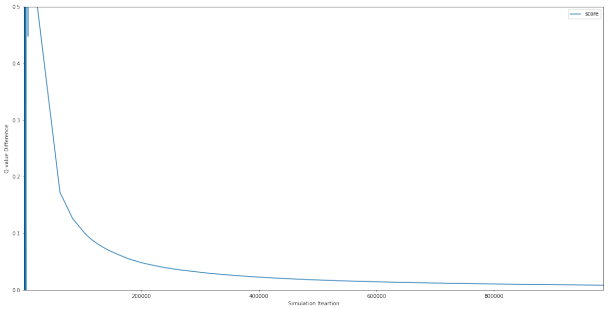

Figure 3: Replication of figure 3-c results from Greenwald's paper

Foe Q-learning

Experimentation setup:

- α = 0.9.
- α_{min} = 0.001
- α_{decay} = 0.9999905
- γ = 0.9
- ϵ = 1.0
- ϵ_{min} = 0.001
- ϵ_{decay} = 0.9995

Experimentation result :
Both graphs shows that the algorithm takes some time to converge unlike friend Q learning. The difference of the Q-values starts to get lower from 100000 step and approach to zero from 600000 step. This result proves the fact that min-max Q is guaranteed to converge.

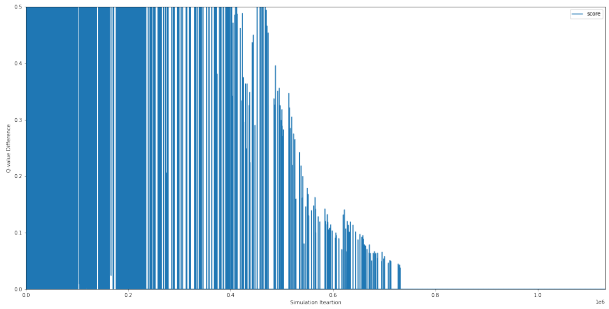

Figure 4: Replication of figure 3-b results from Greenwald's paper

Correlated Q-learning

Experimentation setup:

- α = 0.9.
- α_{min} = 0.001
- α_{decay} = 0.9999905
- γ = 0.9
- ϵ = 1.0
- ϵ_{min} = 0.001
- ϵ_{decay} = 0.9995

Experimentation result :
Both graphs shows that the algorithm takes some time to converge unlike friend Q learning. The difference of the Q-values starts to get lower from 100000 step and approach to zero from 600000 step. The results in this graph are similar to the foe q learning. The explanation might be since we are in a zero sum environment, the min-max equilibrium function behaves the same way as any other variation of the correlated q learning.

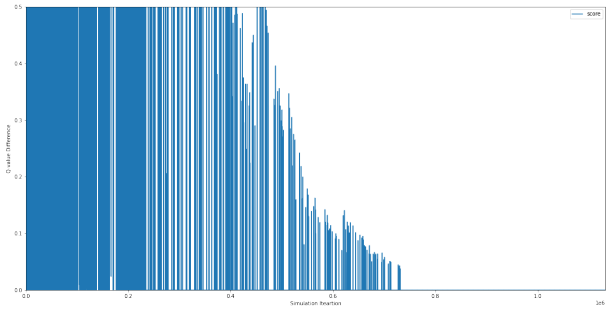

Figure 5: Replication of figure 3-a results from Greenwald's paper

Conclusion

In this paper, I tried to replicate found in the correlated q-learning paper published by Amy Greenwald and Keith Hall(Greenwald, Hall, and Serrano 2003. In the beginning, an explanation about the soccer game environment is needed to understand and to formulate the problem. Then I tried to explain briefly what Nash equilibrium is and it's relation with the correlated equilibrium. After defining the multiple Q-agents while explaining the different algorithms associated with, an implementation and analysis of the results were made.

References

[Amy03] Keith Hall Amy Greenwald. "Correlated-Q Learning". In: (2003). DOI: https://www.aaai.org/Papers/ICML/2003/ICML03-034.pdf.