POLICY GRADIENT REINFORCEMENT LEARNING APPROACH FOR MULTI AGENT COOPERATIVE GAME

Francesco Cerri
DEI - Alma Mater Studiorum Università di Bologna
Bologna, IT
`francesco.cerri2@studio.unibo.it`

Abstract

In this work, I describe a simulation of a cooperative and competitive game, with complete information available, based on rugby. The main interest was to find an approach that could make cooperative strategies between the actors emerge, but also the possibility to use the learned experience on a single actor only. It is presented a solution based on the REINFORCE algorithm, using a neural network to compute policy parametrization.

1 Introduction

1.1 Game structure

Game consists in simplified version of the rugby game with two teams competing against each other; simplification affects the possible situations of the game: the attacking team can only win if it is able to bring the ball beyond the goal line (*try line*), while the defending one will win in any other case. Field has a rectangular form, with length spanning in the vertical direction (y) and width in the horizontal one (x). Coordinates origin is in the upper-left corner and axis point toward right and downwards. Dimensional values can be modified at run-time and the field size is by default of 13 long by 9 wide.

Both teams consists in 3 players cooperating with each other, starting in symmetrical positions on the game field: defending teams is placed in a line with y equal to 2, while attackers on a line symmetrical to the previous one around the field's middle line. Each player and its corresponding opponent are placed in a given column: the *middle* ones stay on the central vertical line, while the external ones (*wings*) on columns having same distance from it.
Try line is an horizontal line, placed right before each team; the *defense try line* by default is placed at $^{1.5}/_{5}$ of field's length.

Basic rules of the game only consists in:

- the game is limited to a constant number of *time steps*; by default 20;
- each actor can only act once for each step and is able to do it only for a certain amount of time; if it is not able to do so, it will *hesitate*;
- no action can bring any actor outside the game field;
- also, players cannot share a *field's tile*;
- it is possible for the defenders to steal the ball to the attackers and hence save the score.[1]

[1] This is a result of checking for termination conditions at the beginning of turn

Autonomous and Adaptive Systems M course, academic year 2021/2022.

At the beginning of each turn, the game check for one of the *terminal conditions*:

1. *time step* > steps permitted for a game
2. ball is still beyond *defense try line*

First one will result in defending team winning, second one will make attackers score a point.
If not verified, it plays *attack phase*, followed by *defense phase*.

1.2 Agents behaviour

Player entities represent the actors in the game. Their motion is limited in both trajectory (only a straight line) and intensity (constant max motion distance) and can be further limited by a specific action described in the following paragraph.1.2 They also store their *position, role and possess of the ball*, which the game will query when updating its state.
The most practical way to differentiate this entities is using both *role* and *ball possess*.
In each of the teams' phases, a player chooses an *action*, following a certain probability distribution; if conditions are not met, it tries again, until its timeslice is concluded.

Actions embed the real complexity of the game. Their effects is here described:

- advance: let the player move along the y coordinate, toward the direction of the *opponents' end line*; intensity is arbitrary. Preventable;[2]
- stepBack: same as *advance*, but towards its *own end line*. It will also cancel the *blocking effect* of *tackle* action;
- dodge: moves the player in x direction of a given direction and intensity;
- ballPass: if the *maximum distance conditions* is met, transfer ball possess to another player. Additional condition is available.[3]
- tackle: if the *motion and distance conditions* are met, *charging* player will steal the ball to the *charged* one, in the case it has the ball; otherwise is block its ability to *advance*.

Different situations have different available actions, for each team. Following table describes the availability:

Table 1: Actions availability to players

	Availability	
Name	Attacker	Defender
advance	always*	always*
stepBack	always	always
dodge	always	always
ballPass	with ball**	with ball
tackle	without ball**	without ball

* see *tackle* action for cases,
** in *multi actor mode*2.2.1 always available.

For more details refer to code in appendix.A

2 Policy gradient approach

The proposed approach used an algorithm based on the **REINFORCE**[3][1][2] by *preferences parametrized with softmax in action*[1][2].
My interest came from the fact that in this game some actions are not available for all players, and i

[2] As described in *tackle* action.

[3] Back passage only can be forced. Currently unused.

wanted the actors to learn to automatically mask that ones thanks to the ability of emphsoftmax in action proferences approximated policy to approximate a deterministic one. To be more precise, it can learn an *arbitrary stochastic policy*; [1][2]; moreover, for this reason it has good convergence properties.[1][2]

2.1 Background: policy gradient methods and REINFORCE

These are methods that learn a parametrized policy without referring to value-functions; for policy we refer to π(a | s, θ)=$\{..., Prob(A_t = a|S_t = s, \theta), ...\}$ the probability that action a is taken at time t given that the environment is in state s at time t with $\theta \in \mathbb{R}^d$ for the policy's parameter vector.[1] Learning θ is based on the maximization of some performance measure J(θ), hence the update approximizes *gradient ascent* in J: θ_{t+1}= θ_t + $\alpha \nabla \hat{J(\theta_t)}$.

A typical **policy parametrization**[4] can be obtained using softmax in action, namely $\pi(a|s\theta) = \frac{e^{h(a|s,\theta)}}{\sum_b e^{(h(b|s,\theta)}}$, with h(als,$\theta$) is a scalar utility function expressing preferences, that is higher Prob of being selected, for (any) (s,a) for the policy. A typical to way to express the preferences is using linear features[1].

In my work, preferences are computed using the probabilities of action selection **given by a neural network**.

Using the **Policy Gradient Theorem**, we can advantajeously approximate the gradient to a form easy to compute: for the episodic case it is obtained[1][2] that ∇J(θ) $\propto \sum_s \mu$(s) $\cdot \sum_a q_\pi$(s,a)$\nabla\pi$(a | s, θ).

REINFORCE [3] is a Monte Carlo, policy gradient control algorithm[2]. Using the theory discussed above, we can obtain the following results: the update of the parameters (the net, in my case) is θ_{t+1}= θ_t + $\alpha \cdot G_t \cdot \nabla ln\pi(A_t|S_t, \theta_t)$, where the loss function is J(θ) $= -G_t \cdot \nabla ln\pi(A_t|S_t, \theta_t)$. Please note that thanks to α scaling we can find $\alpha*$ | $\alpha*\cdot\nabla\hat{J}(\theta)$=$\nabla$J($\theta$), hence changing the proportionality to a real equivalence.

2.1.1 Theoretical performances and issues

If α is small, then improve in performance is guaranteed; if α diminishing, then convergence (at least to local optimum) too. Yet, REINFORCE is a Monte Carlo method, hence it will perform sometimes random actions in a wrong direction, hurting learning speed (high variance)[2].

2.2 Multi agent setting

The biggest difficulty in this setting is the *coordination problem* and the effective *optimality of joint*[5] *action* [4][5][6]. To simplify the problem, and to focus on the real goal of this work, I implemented a *centralized control and information unit*, such that all information is available to all team members. Moreover, I assumed that all the possible combinations of actions are optimal, i.e. that that particular combination is the only optimal one that exists.

2.2.1 Behaviour modes

Learning is structured in a way that it is possible that the whole team learns (the so called *multi-actor mode*), namely every particular player has its own distribution for action sampling[6], or only the ball bearer learns, while the others use a *random policy*. This let me verify how the position influenced the scoring capabilities.

2.3 Reward structure

Reward, which is assigned at each time step as result of *attack and defense phase*, is shared by whole attacking team (**cooperative setting**): the first may assign positive or negative rewards (see below), while the latter will only assign negative rewards. In the case of *single-actor mode*, *punishment*

[4] It is required that the policy is differentiable.

[5] i.e. considering all team members jointly

[6] In this case, role inside the team (*mid,wing* it's important.

resulting from defense phase is discounted, to take into account the fact that 2 attackers act randomly. Please note that, if multiple *on policy actions* within the attack phase may result from a ball pass, even in this particular behaviour mode.

Table 2: Reward values

Event	Value	Attribution
Game lost	-200	1
Game won	+300	1
Time step passed	-10	1
Formation too broad	-30	2
Ball moved toward goal	+8	2
Ball moved away from goal	-2	2
Ball lost	-5	3
Ball regained	0	3
Ball pass completed	0	3
Player hesitated	-80	3

Rewards attribution situations are the following:

1. when episode termination conditions are checked. Win and loss rewards are attributed once per episode, while the third every time step that is not terminal;
2. at the end of attack phase. Punishment for motion away from try line is scaled by the effective vertical distance between that line and team barycenter's y, while the reward for moving toward it is scaled by the effective vertical motion. Punishment for keeping an average distance between players too high (thus preventing exploiting of ballPass action) is scaled by that value;
3. at the very moment this event happens.

2.4 Implementation of training procedure and network

Implementation of the training procedure uses Tensorflow and Keras[7][8]. It starts by creating the requested number of episodes: for each of them, the history of states, actions and rewards are stored in lists, which are fed to the training method which computes the current *diminished training rate* and, for each time step in that episode, the *expected return* (actualized sum of future rewards for a given step).

Algorithm 1 Learning Algorithm

```
1. Input softmax in action preferences parametrized, differentiable policy $\pi$(a | s, $\theta$)
2. Parameters learning rate $\alpha > 0$, reward discount factor $\gamma > 0$
3. Initialize policy parameters $\theta \in \mathbb{R}^d$
4. Loop forever (episodes number):
    (a) generate episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$, following $\pi$
    (b) Loop for each step of the episode t= 0,1,...T-1:
        i. G$_t = \sum_{k=t+1}^{T} \gamma^t R_k$
        ii. $\theta_{t+1}$= $\theta_t$ + $\alpha \cdot G_t \cdot \nabla ln\pi(A_t|S_t, \theta_t)$
```

Loss function computation is performed following the gradient ascent rule, using the aforementioned expected return, and the softmax in action preferences computed on action probabilities obtained from the stored states. Their product is then passed to tensorflow.nn.log-softmax activation and it's gradient, with respect to all trainable variables, computed using tensorflow.GradientTape object .(For the optimization step, see below 2.4).

Optimization and parameter update is performed by Keras' Adam (adaptive moments) optimizer routine.

Network is model using Keras. Input consists in a numpy 2-dimensional array representing the state of the field (players and ball position); output it's the probabilities distrubution for the actions selection, which is then normalized to unitary probability. Please note that single-actor mode has a different output shape (1 by 5) than multi-actor (3 by 4) The following table shows the different layers of the network.

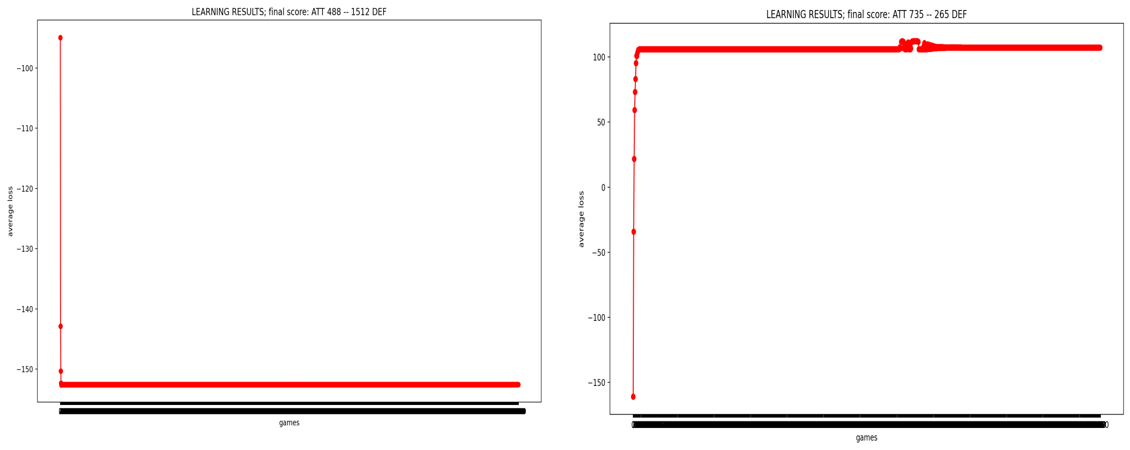
Table 3: Network layers

Number	Description
input	flattening
H1	dense, 32 nodes, softmax activation
H2	dropout (20%)
H3	dense, units depending on actor mode[7], softmax activation
output	reshape to matrix form (see above 2.4

3 Results and conclusions

3.1 Comparison with random policy

With respect to a random policy, which is able to win $30\%$of the times, the **learnt policy is able to win up to** 90% **of the times**.
From Figure 11 it is possible to observe the convergence properties showed in this case; it must be noted however, that speed of convergence was really fast, possibly falling in a local optimum.



(a)

(b)

Figure 1: History of Loss during episodes: (a) random policy (b) learnt policy

3.2 Emerging strategies

The actor which had the ball often learnt that the most rewarding action was to sprint toward the line. Since the opponents will sometimes stole the ball, it learnt to wait until it had enough free space, taking advantage of all opponents stepping back. See: Video

Some learnt behaviours where indeed bad: sometimes one of the wings moved to the opposite end of the field: I tried to punish this behaviour using tight formation rewards. See Video

3.2.1 Team strategies

Being a cooperative game, my main interest was to observe the emergence of some behaviour, where multiple agents cooperated to obtain a win. I present here the two most interesting ones observed:

Left wing dash: in this case, mid player learnt to give the ball to the sprinting left wing, who than proceeded to go in a straight line towards goal. See: Video

Formation control: in this case the whole team learnt to go toward the goal, trying to keep constant distances between them. See: Video

3.3 Limitations

It is not unusual that solution stalls to a value that appears "non optimal": this is caused by a bad starting set of moves (initial condition), which make the solution converge to a "bad local optimum. Nevertheless, there are no cases where the solution diverges. Even with many different sets of *hyperparameters* (i.e. learning rate, discount factor, dropout share), this problem still persists sometimes.

3.4 Future work

The problem presented before can be solved trying to force exploration in the cases where the loss stagnates.
Creating a second network to teach defenders how to act is also a good setting to really test what the cooperation between attackers can develop new strategies. Moreover, this could be a way to force exploration.

References

[1] Sutton, S. & Barto, A.G. (2018) Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press.

[2] Musolesi, M. (2022) Lectures from the course "Autonomous and Adaptive Systems M". Bologna: Alma Mater Studiorum Università di Bologna.

[3] Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn 8, 229–256 (1992). `https://doi.org/10.1007/BF00992696`

[4] Buşoniu, L., Babuška, R. & De Schutter, B. (2010) Multi-agent reinforcement learning: An overview. Delft Center for Systems and Control Technical report 10-003 `https://www.dcsc.tudelft.nl/~bdeschutter/pub/rep/10_003.pdf`

[5] Buşoniu, L., Babuška, R. & De Schutter, B. (2018) A Comprehensive Survey of Multiagent Reinforcement Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part C, Volume 38, Issue 2 `https://ieeexplore.ieee.org/document/4445757`

[6] Zhang, K., Yang, Z. & Basar ,T. (2021) Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. arXiv:1911.10635v2 `https://arxiv.org/pdf/1911.10635.pdf`

[7] Tensorflow documentation`https://www.tensorflow.org/api_docs/`

[8] Keras documentation `https://keras.io/api/`

A Code and definitions

For the code and documentation, please refer to
`https://github.com/gnoccoalpesto/rugby_rl`
Please note: all video materials require access to Unibo OneDrive Server.