Group 5 Intermediate Report

Egger, Maximilian — Pelat, Guillaume — Pilger, Laura

June 3, 2021

Abstract

Following our proposal, we investigated two agents, one based on Dynamic Programming and one based on Q-learning while using different rewards approaches. To come up with desired policies, we also optimized the feature space. The initial findings concerning the state to feature conversion, the reward function composition and the algorithms can be found in this document together with ideas for future improvements.

1 Introduction

After one and a half month of training agents within the Haxball environment, we successfully accomplished to come up with an agent that is able to shoot into the opponent's goal. This report should give insights into the current status of the project, point out the changes that were necessary compared to our initial plans and introduce future ideas.

Our proposal was based on the assumption that the agents are rewarded for shooting in either one of the goals. However, the feedback made clear that the agents should in fact only aim for the right goal. That is why we had to modify our planned feature space. The changes are described in section 2. In contrast to our initial approach, we are now using this feature space for both algorithms, dynamic programming and Q-learning. Furthermore, we worked on our two reward approaches that are further explained in section 3. The progress for the different agents can be found in section 4 and section 5. Finally, future improvements are described in section 6.

2 Feature Changes

In the initial report, we introduced our feature space consisting of three major components: the ability to shoot, the relative directions from the player towards the current and the future ball position and the discretized angles between the player and either one of the goals. However, this idea was based on the assumption that the agent should be trained on shooting in any goal. As the agent should in fact only aim for the right goal, we first added a boolean describing if the agent is located in the left or the right half of the field. This information should help the agent to avoid shooting into the opponent's goal. Additionally, we removed the discretized angle between

the agent and the left goal from the feature space, since this information is not relevant anymore in contrast to the initial assumption.

With these changes applied, the agents were still not aiming for the right goal exclusively. Therefore, we tried to train the agents with very different feature spaces, including, for example, a completely discretized set of the state space. Currently, a modified version of the initial approach is the most promising. In particular, we removed the boolean describing the side of the field in which the agent is located in and also the angle relating to the right goal. To enable the agent to learn more advanced strategies, such as scoring goals by using the walls, we added two features describing the discretized agent's position, in both horizontal and vertical direction. Therefore, we divided the field into a grid of 10×20 subfields which corresponds to a resolution of 0.4. This set of features eventually results in $2 \cdot 3^4 \cdot 11 \cdot 21 = 37422$ possible states which is an even smaller state space than the one described in the initial report. By now, this set of features led to the best results. However, we are planning to further investigate optimizations, e.g., including the distance between the agent and the future ball position.

3 Reward Changes

At first, we chose to use a reward with fixed values, each associated with a component:

1. 1 for getting closer to the ball: $||p_{ball}(t+1) - p_{player}(t+1)|| < ||p_{ball}(t+1) - p_{player}(t)||$
2. 50 for shooting in direction of the goal
3. 100 for scoring in the right goal

We wanted to try two approaches: one with using all rewards at once (the *scaling* approach), and one where the training would proceed in three phases, each using one component (the *sequential* approach). The challenge for the first one was that if the scaling was off, the agent would not learn the intended policy. For the second approach, the difficulty was to estimate the length needed for training before proceeding to the next stage.

During testing, the scaling approach proved to be efficient enough, so we gave up using the sequential approach at all. The chosen values seemed good enough such that no counterproductive behavior was learned (e.g., like wedging the ball in a corner). However, we found out that our approach with fixed values had two problems: firstly, if the incentive to aim for the right goal was not large enough, our agent would end up scoring equally in both goals. Adding a big punishment for this undesired action only seemed to deter scoring at all. Secondly, there was no incentive to use diagonal movements compared to moving along one axis, and thus the agent's movements could have been faster.

Hence, we decided to use the unweighted sum of the following components as our reward function:

1. Getting closer to the ball:

$$r_1 = ||p_{ball}(t+1) - p_{player}(t)|| - ||p_{ball}(t+1) - p_{player}(t+1)||$$

2. Getting the ball closer to the right goal:

$$r_2 = ||p_{ball}(t) - p_{rgoal}|| - ||p_{ball}(t+1) - p_{rgoal}||$$

3. Staying at the left side of the ball:

$$r_3 = \begin{cases} p_{ball}(t+1)|_x - p_{player}(t+1)|_x & \text{if } p_{ball}(t+1)|_x < p_{player}(t+1)|_x \\ 0 & \text{else.} \end{cases}$$

   Additionally, this component is doubled if $p_{ball}(t+1)|_y \approx p_{player}(t+1)|_y$ to help the agent circumventing the ball.

4. Scoring goal (the sign accounts for right/wrong goal):

$$r_4 = \pm 1000$$

Although the agent still has trouble scoring, it is visually much better at moving and shooting roughly in the right direction. In particular, the addition of r_3 was probably the most efficient change to avoid shooting in the wrong goal. Therefore, our reward function should not change much during the second phase of the project, and we will focus on honing our features and algorithms.

4 Recursion and Dynamic Programming Agents

As explained in our first report, we wanted to use Dynamic Programming (DP) as a baseline in order to validate our choices for the feature space and the reward function. In each iteration, however, this algorithm should run once through the whole state space, or in our case, the feature space. With our setting, this is rather difficult to realize, since we would need to convert our feature space back to the state space in order to set the environment state appropriately. Due to the previously performed dimensionality reduction in the state-feature mapping, this re-transformation, in turn, would add inaccuracy. Therefore, we tried to solve the problem by implementing three different approaches. For each approach, we utilized multi-threading to speed up the convergence.

At first, we tried applying a conventional recursive algorithm that tries to maximize the sum of all rewards given a certain number of steps. For every state, which is randomly chosen by resetting the environment, it iterates through every possible action and computes the maximizing action and value by recursively applying the algorithm for every subsequent state. This resulted in an agent that was able to find the ball, shoot and even score goals. However, even though the agent behaved very well on certain positions, it also scored in its own goals regardless of the size of its punishment. Therefore, we assumed that the recursion depth is insufficient to observe important consequences of its behaviors. Since the memory occupied for the recursion is growing exponentially with this depth parameter, this algorithm is not feasible to learn the desired behavior. In this case, the agent was limited to be aware of 20000 steps in the future, resulting from the maximum recursion depths for our hardware. It once again showed the limitations of recursive algorithms.

Next, we wanted to try out two slightly modified approaches of Dynamic Programming for which we exploited two properties of our environment: firstly, resetting the environment leads to a

random state and secondly, the resetting to a certain state is always possible. The idea was to cover all states often enough randomly by resetting the environment to ensure convergence. Therefore, for each randomly chosen state, we iterated through every action, calculated the Bellman operator and saved the maximizing action and value. In practice, however, this algorithm took a very long time to visit all states and was therefore not able to converge in a reasonable amount of time.

To speed up the convergence behavior, we implemented an algorithm that saves a value for each state-action pair and uses a huge amount of short trajectories for learning. This set-up would allow us to initialize with a random policy and a randomly chosen starting state, calculate the Bellman operator for every state in the trajectory based on the current policy and change the values accordingly. Since this modification of DP became rather close to Q-learning, we then decided that we will give up on DP for this task and focus our efforts on Q-learning.

5 Q-learning Agent

In parallel, we trained a second agent with Q-Learning. The first version used the basic implementation of what was seen in class, thereby using multi-threading to allow for faster training. A notable improvement was the implementation of an eligibility trace: while each epoch is slightly longer, the overall training time was greatly reduced. We decided to keep the eligibility weights only if they are greater than $1e^{-3}$, which with an intermediate λ (according to the examples in Sutton & Barto's book [1], $\lambda \approx 0.7$ seems to work best) leads to $\sim$ 20 coefficient updates in the whole Q matrix per iteration. When storing the eligibility trace in Eigen's Sparse Matrix, it should become thread-safe[1] while improving the speed by a factor of 64.000 compared to using a single-thread and a dense matrix.

We will not expand too much on the results here, as they were the cause for changes made on the features and reward components stated earlier. To summarize it, the agent has no trouble following the ball (and even circumvent it if he is in front of the ball), and shoots mostly in the right direction. Currently, the main problem encountered by our agent is that it oscillates a lot between two pairs of states: for example, the chosen action for state s_t is "go left" and the action for s_{t+1} is then "go right". Such cases happen a lot and we have to change the way our ϵ-greedy policy works: instead of randomly choosing an action from time to time for one step, we should stick to that action for multiple steps to be sure to provoke a change in the underlying states.

Finally, although we did not attempt much hyper-parameter tuning (by hyper-parameter, we mean the discount factor, learning rate, eligibility parameter and the epsilon greedy policy), we tried several ideas to change the exploration of our agent. Currently, we settled for $\epsilon = \frac{1}{2i}$, where i is the current epoch. This way, an agent beginning its training will spend most of its time exploring, while the latter epochs are mostly dedicated to validate previous findings. After discussing it with our supervisors, we will dedicate the time that we originally planned for hyper-parameter tuning in our schedule to the future improvements described in the next section, since the default values usually work very well for non-deep RL algorithms.

[1] We are not entirely sure of the way the addition is implemented in Eigen, but did not encounter any problems while trying to do so.

6 Future Improvements

Initially, we planned to finish implementing the algorithms until the 18^{th} week this year. For the DP agent, it took us longer since we changed our strategy on the way. However, we finished investigating different reward approaches in time and were able to validate the results obtained so far, meaning we are pretty much on schedule with respect to our initial plans. Considering the results achieved by our agents, we now want to focus on the following topics:

Further try to optimize feature space: The features we currently use are still not definitive. In particular, we want to investigate if adding information about the distance to the ball is useful or not.

Investigate situation-based agent: When computing the Q-matrix, we wondered if it would not be easier to separate the cases where the agent is in shooting range and where it is not, as in the second case half of the actions (those involving shooting) are meaningless and only add to the computational complexity. We could take it further and have one algorithm using only parts of the features to get in shooting range, and a second one handling only the actual shooting. [2] indicates that this situational partitioning is a valid approach.

Implement greedy multi-step Q-learning: When looking at the state-of-the-art, we discovered a recent paper (not yet published) describing a variation of Q-learning that would be worth to investigate. [3] proposes an improved version of the Bellman Operator, which supposedly converges faster than the original operator, and introduces a new bootstrapping method.

Train agents with opponent enabled: Everything we did up to now was designed with our agent being the only player on the field. In this new situation, our features would be enhanced by the discretized position of the opponent (similarly to our own agent's position). Furthermore, a reward component that penalizes getting the ball close to the opponent might become necessary. Nevertheless, adding an enemy increases the challenge and is a first step towards winning the world cup.

References

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

[2] Ure, N. & Yavas, U. & Alizadeh, A. & Kurtulus, C. (2019). Enhancing Situational Awareness and Performance of Adaptive Cruise Control through Model Predictive Control and Deep Reinforcement Learning. 626-631. doi: 10.1109/IVS.2019.8813905.

[3] Wang, Y. & He, P. & Tan, X. (2021). Greedy Multi-step Off-Policy Reinforcement Learning.