Upside-down Reinforcement Learning

Yannik Hesse James Liu Amandus Reimer

1. Introduction

Traditional reinforcement learning (RL) algorithms like temporal difference (TD) learning often involve constructs like value functions, which introduce complexities. These include relying on bootstrapping, one of the "deadly triad" known to destabilise RL (Sutton & Barto, 2018), and Bellman backups, which are prone to distractor signals (Hung et al., 2019). In practice, they typically also require an additional target network to stabilise learning, as in Deep Q-Networks (Mnih et al., 2015), which increases the number of parameters and complexity during training. One can bypass these issues through policy gradient methods, like PPO (Schulman et al., 2017), but these are typically online algorithms, requiring expensive environment interactions during training.

A potential solution is to cast RL as a supervised learning task. This has been historically attempted (Jordan et al., 1988), and is especially attractive now given the recent successes of the supervised learning paradigm (Brown et al., 2020). We look to build on UDRL (Schmidhuber, 2019), where we learn to directly map states to actions after conditioning on rewards. More specifically, this method falls under the category of offline reinforcement learning. The model is given a fixed experience dataset, comprising of $(state, action, reward)$ tuples and is tasked with predicting the $action$ corresponding to the $state$ and command signal, $reward$. The command signal is retroactively created from actual observed rewards in the dataset, so that the conditional distribution $p_{A|S,R}(a|s,r)$ is modelled, and can be sampled at test time.

UDRL bears resemblance to behavioral cloning (Torabi et al., 2018). However, a notable drawback is its challenge in exploring the environment effectively, often requiring substantial amounts of pre-generated rollouts for optimal agent training.

In this project, we aim to explore the dynamics of self-play within a one-on-one scenario as a means for exploration. We shall use the game of Connect4 as a challenging task. The baseline will be randomly acting players, and the models will be improving using a selfplay scheme.

In such a setting, rewards manifest in only three categories: 1 (Win), 0 (Draw), or -1 (Lose), and these rewards are allocated solely at the conclusion of each episode. Consequently, numerous actions contribute to the overall outcome, yet the impact is observed only at the episode's end. The central question motivating our research is whether self-play can induce exploration in environments characterized by such sparse feedback. We propose that self-play with only rewards $\in \{-1, 0, 1\}$ allows us to avoid having to sample for rewards that are out-of-distribution. Additionally, we had originally aimed to investigate the applicability of the Decision Transformer (DT) (Chen et al., 2021) in adversarial reinforcement settings. The DT is more focused on sequence modelling and autoregressive token generation. We were unable to obtain convincing results, so this report shall focus on UDRL.

2. Models and Methods

2.1. Environment

The environment in which the game is played is a 6×7 grid, where two players (also called agents) take turns in placing tokens in the different columns. A game is completed when there's 4 tokens belonging to a player next to each other in a row, column, or a diagonal. The rewards that are given to the players at the end of a game is 1 (Win), 0 (Draw), or -1 (Lose).

To be able to rank the players performance an ELO system has been implemented. If player A has an ELO of R_A, and faces player B with an ELO of R_B, the ELO is updated as follows after they have played their games;

$$E_A = \frac{1}{1 + 10^{(R_A - R_B)/400}}, \tag{1}$$

$$R_A^{'} = R_A + G/100 \cdot 10 \cdot (S_A - E_A), \tag{2}$$

where G is the number of games played, S_A is the number of points per game that player A achieved, and E_A is the expected points per game based on the players difference in ELO. The factors of 100 and 10 in the calculation of the new ELO limits the maximum gain/loss after the games are played out.

2.2. MLP

We initially experimented with a straightforward feed-forward network as our first model. The architecture closely resembled that of the original UDRL paper by Schmidhuber et al. (Schmidhuber, 2019), with some portions of code being borrowed. Through testing and development, we determined that the optimal parameters were a layer width of 5,000 and a depth of 5.

A notable departure from Schmidhuber et al.'s model is our disregard for the horizon (remaining episode length). Instead, we chose to input the final reward of the game (1 for a win, -1 for a loss) and the ELO rating of the player executing the current move. Combining the ELO rating, result, and the current state of the board, we provide this information to the model, which subsequently generates a probability distribution over the seven possible Connect4 actions.

2.2.1. Training

To train our agent, we employed the Adam optimizer and formed batches from previously generated games. This process involved randomly selecting a game and then, more crucially, choosing a random timestamp within that game. This approach aimed to mitigate the bias toward the initial states in most games, which tend to be similar and can be over-represented in the training data. To address this imbalance and introduce greater diversity in our training set, we sampled timestamps from a random distribution that favoured mid-game and end-game timestamps over early-game states. We consistently stacked 1000 states before performing an optimization step. Our early stopping criterion dictated that if we did not observe improvement in the last 200 iterations, we would halt and terminate the training process. As a loss function we use PyTorch's cross entropy method.

2.2.2. League play

We implement the training and evaluation of the agents by a league based system. In the beginning there are two random players created. For 10 iterations they play 10000 games against each other. After the initialization, the games proceed as follows; a new player is created and trained until convergence or early stopping on games from the replay buffer. Afterwards this player is added to the list of players. Then two players are selected at random from the player list. These selected players play 1000 games against each other. This selection and playing is repeated 50 times.

After all of these games are completed the new player plays against itself for 5000 times.

After each iteration of playing games the ELOs of the players are recalculated.

The training flow can be seen in figure 1.

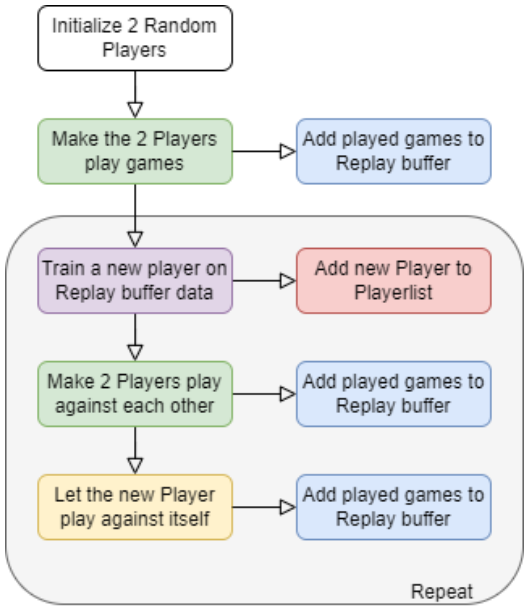

Figure 1. Flow of play and initialization during league play.

3. Results

3.1. Baseline

The baseline used in the project is a random agent playing against another random agent. The win percentage in this case is close to 50%. For the other models the win percentage should thus be above 50% when playing against a random player, to be able to deduce that an improvement has occurred.

We start of with two different random players. The first random player (uniRand) plays all actions based on a uniform distribution, while the second random player (greedyRand) plays the middle column with 50% of the time, otherwise it places a token in another column with uniform probability. If these two agents face each other greedyRand wins 65% of the time. These two players can then be used to evaluate the performance of our own trained agents.

3.2. MLP results

The results from the league play system, which can be seen in figure 2, shows that the performance of the MLP agents appears to quite quickly figure out how to beat the random opponents, and the new agents all outperform the random players fairly directly. The ELO diminishment of the random players is similar, even if the greedyRand player has a higher ELO throughout the whole league. This might be attributed to the limiting factor of ELO gain/loss per game as was described previously, and that the random players

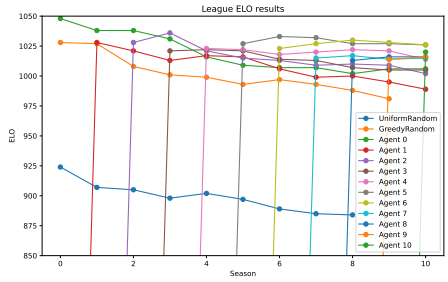

Figure 2. ELO distribution per season for a 10 season league where one new agent is added to the player list per season.

initially had a quite large difference in between themselves in ELO.

The new agents all achieve similar result, they have roughly the same ELO, which is to be expected when agents of similar skills play plenty of games against each other.

The difference in ELO between the trained agents and the pure random agent is about 100 ELO points, which corresponds to the trained agents winning roughly 65% of games when matched against the random players. For the greedyRand player, its ELO deficit is between 10 - 50 points to the trained agents, which means a winrate of a bit lower than 50% when pitted against the trained agents. Hence the trained agents can be said to outperform the set baseline.

In the interest of investigating the trained agents decision making, two specific situations were studied after the league play was completed. The study was made with the final created agent, since this agent was trained on data from more recent games, with other trained agents, and fewer of the initial random games.

The first of these situations is the start of the game, where one expects the agent to want to place a token in the middle column. This is based on the fact that more possibilities of winning can be made containing tokens in the middle rather than in the corners, as well as one of the first papers solving the game (Allis, 1988). There it was proven that the first agent can always win if they placed their first token in the middle.

In figure 3 we see the first token placed by a agent trained by self play, along with the probability distribution that caused the decision. As expected the distribution shows a clear peak in the middle, with some diminishing probability of placing the token in another column closer to the edges, as expected.

The second situation that was studied was if the agent learned to block a game winning move by the opponent. Figure 4 shows that the agent realized that it was most beneficial to place a token in the first column to block the

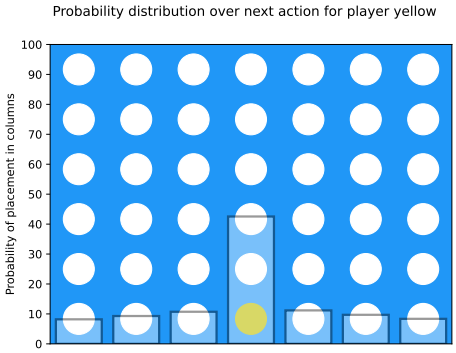

Figure 3. How a self play trained agent distributes the first piece.

opponent from winning.

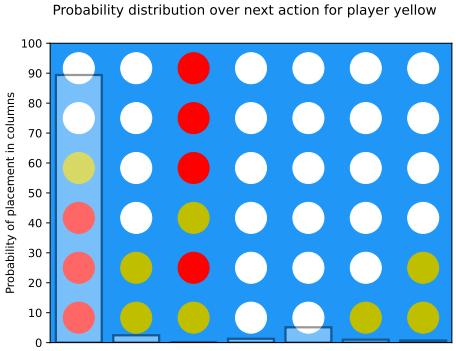

Figure 4. The agent learns to block a potential win of red by playing in the first column.

4. Discussion

4.1. The problem of randomness

During our initial experiments, we grappled with an intriguing challenge related to the issue of unpredictability, where certain actions predicted by agents could result in a losing game state. This phenomenon was well-documented in an article by (Paster et al., 2022), which delved into the intricacies of the problem. The key concern was that if an agent executed action x and it led to a favourable outcome 75% of the time, the agent remained oblivious to the fact that 25% of the time, the same action could lead to failure. The significant obstacle stemmed from the absence of inference in identifying the link between actions and their potential undesirable outcomes.

This challenge is less pronounced in scenarios with a large or continuous action space where actions can be distinctly categorized as either good or bad. However, in Reinforcement Learning, such clear distinctions are often absent, and ac-

tions may be deemed merely ”okay,” with their true impact becoming evident only after several states. The issue is particularly evident in games like Connect4, where a player's single action can result in 7 varied game states depending on the seemingly ”random” moves of the opponent.

While (Paster et al., 2022) adeptly outlines and illustrates this problem, it doesn't propose a concrete solution. Notably, (Schmidhuber, 2019) acknowledges this limitation in the approach. Their suggested solution involved training on the average outcome of an action rather than the rewards of a single rollout. However, implementing this solution is challenging, especially in cases like Connect4, where the immense number of possible states (4.531.985.219.092) and actions per state make straightforward storage on current hardware unfeasible.

In response to this challenge, we devised a strategy involving querying our model for actions it would take to win and actions as if it intended to lose. By obtaining distributions for both scenarios, we could compute the difference. This approach aimed to encourage solutions leading to favourable outcomes and discourage those resulting in unfavourable ones, offering a practical workaround to the inherent complexities of the problem.

4.2. Limited Improvement

The agent's exploration of the state space is constrained, and there is no inherent incentive for it to do so in the design of UDRL. We anticipated that self-play would compel the agent to adapt its strategy and discover novel ways to outmanoeuver opponents. However, this expectation was not met, and instead, we observed more of a behavioral cloning phenomenon rather than genuine learning about the game dynamics. The agents could be seen to overfit to their own policies, and converging to local minima.

This tendency became especially apparent as we initiated training with simple random agents whose actions were initially unrelated to the game's state. The multilayer perceptron (MLP) adapted to mimic these random actions. Given that Connect4 is a solved game with known optimal moves for any state, we expected the agent to acquire this knowledge. Nevertheless, our observations, illustrated in Figure 5, reveal a limited understanding of the game on the agent's part.

While it might be unfair to evaluate this agent against the standards of an informed minimax player, it is evident that the agent's game knowledge is restricted. We stopped the playing after 10 seasons as the improvement was no longer significant and the newest agent frequently lost (47%) against older ones.

An avenue to explore is to use self-play with Decision Transformers, modelling the joint distribution of states, actions and rewards as an autoregressive sequence. The self-attention mechanism allows for long-term credit assignment, even in sparse reward settings. (Chen et al., 2021) showed that DTs were able to model a wide range of behaviours, so we hypothesise that generating a larger variety of behaviours with DTs in our league situation could result in more exploration, and improved learned behaviours.

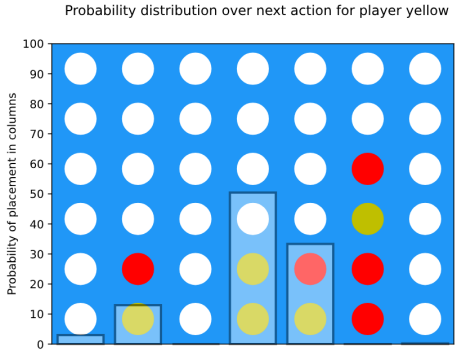

Figure 5. The agent is not aware of a lose condition, that it could have won if it chose to place a token in column 3 instead of in column 4.

4.3. Impact of commands

We provide the agent with commands on how to play the game. One part of this command is the ELO it should play at, the other is if it should aim to win or lose the game. How well our agent follows these commands is interesting to see. For one it is hard to determine how a certain agent would play at a certain ELO and that in itself is entirely based on its training data. Nonetheless its interesting to look at to scenarios and how it would play them at different ELO levels. Examples of this is shown in appendix A, where the first placement is studied for agents trying to play with high/low ELO, attempting to win or lose.

5. Conclusion

We explored the use of self-play as a means for exploration in the context of UDRL. The results showed that self-play was able to allow a MLP to learn basic strategies, such as beginning the game in the center, and to block near attempts at four-in-a-row. However, after a few rounds of self-play, the inability of the MLP to generate varied behaviours caused overfitting, and behaviour improvement slowed. We then outlined an avenue for future investigation, utilising self-play with Decision Transformers with the hypothesis of improved generation of varied behaviours, helping to alleviate the exploration problem.

References

Allis, V. A knowledge-based approach of connect-four. *ICGA Journal*, 11:165, 1988.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Hung, C.-C., Lillicrap, T., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):5223, 2019.

Jordan, M. I. et al. *Supervised learning and systems with excess degrees of freedom*. University of Massachusetts at Amherst, Department of Computer Science, 1988.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Paster, K., McIlraith, S., and Ba, J. You can't count on luck: Why decision transformers and rvs fail in stochastic environments, 2022.

Schmidhuber, J. Reinforcement learning upside down: Don't predict rewards - just map them to actions. *CoRR*, abs/1912.02875, 2019. URL http://arxiv.org/abs/1912.02875.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

A. Low ELO agents start

Two cases of trained agents placing their first token while playing as a low ELO player. In figure 6 the agent attempts to play at ELO 100, with the goal of losing. In figure 7 the agent attempts to win, playing as it would have 100 ELO as well.

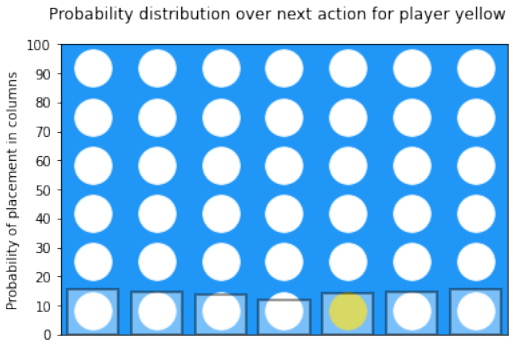

Figure 6. A trained agents probability distribution when asked to make the first move, with the aim of losing playing as a low ELO player. Notably it tries to avoid the central column, which is theoretically the best starting move.

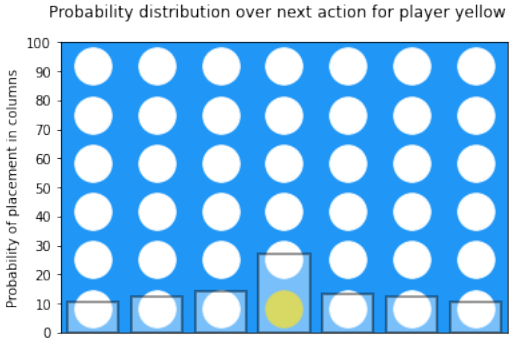

Figure 7. A trained agents probability distribution when asked to make the first move, with the aim of winning playing as a low ELO player. The probability distribution is similar in shape to the higher ELO agent, see figure 3, but with more spread toward the edges.