Online Reinforcement Learning as Sequence Modeling

Oliver Dudler   Albin Koshy   Nino Louman

1. Introduction

In the realm of game theory and artificial intelligence, the translation of reinforcement learning (RL) tasks into supervised learning (SL) frameworks presents a novel and under-explored challenge (Li et al., 2023). Our work uses a recent approach to this translation (Schmidhuber, 2020) by implementing the Decision Transformer introduced in Chen et al. (2021) for the full-information game "Connect Four".

Transformers (Vaswani et al., 2017) introduce a paradigm shift in approaching reinforcement learning (RL) tasks, contrasting the traditional RL algorithms which often require careful tuning of hyperparameters (Li et al., 2023) and reward functions—factors that can render them fickle and unpredictable. The translation of RL to a more stable supervised learning task and the well-researched nature of transformer architectures offer a more robust foundation (Li et al., 2023). This reliability, coupled with their proven superior performance in various domains, positions transformers as a promising alternative to conventional RL methods.

Our main contribution is the implementation of a Decision Transformer model that can learn both offline and online. We compare the performance of the Decision Transformer (Chen et al., 2021) to traditional RL methods, a deep Q learning algorithm (Van Hasselt et al., 2016) as well as a more advanced RL model (Kumar et al., 2020). The full source code of this project can be found on GitHub[1].

2. Models and Methods

2.1. Environment, State & Action Space

"Connect Four" is a well-known full-information board game. We implement a custom "Connect Four" environment, representing the board as a 6×7 matrix E, with elements $e_{ij} \in \{0, 1, 2\}$. A zero at position (i, j) means that the position is unoccupied, a one or two represents a chip of the respective player at the position.

The actions our actors can perform in the environment are represented either by numbers $a \in \{0, ..., 6\}$ or by 7-dimensional one-hot encoded vectors representing the selected column for placing the chip.

[1] https://github.com/odudler/dl2023

2.2. Rewards & Training Structure

We experimented with various ideas for reward structures and decided on the following one inspired by other implementations (D'Hulst et al., 2022; Voigt, 2020): For a victory/loss the agent receives a reward of ± 1. For a tie, a reward of -1 is given as we want to encourage winning. For playing any move a reward of 0.05 is given to encourage survival. For playing an invalid move, a reward of -0.1 is given.

With regards to training the traditional agents, we let them play a certain number of games against a specified opponent, during which their traces are collected into a memory containing the most recent n traces. In the learning step, a batch of samples is drawn from the traces. These are used to calculate the loss and perform backpropagation within the agent.

We implement several options for training. To gradually increase the difficulty of the opponent, its randomness can be decreased regularly after a specified number of episodes have been played—if applicable to the type of opponent. This is done to give the agent the possibility of starting simple and gradually learning more complex plays. We also implement the option of replacing the opponent with older versions of the agent after a certain number of bootstrap episodes, i.e. letting the agent perform self-play to improve further.

The agent may play against a random-playing opponent and a Minimax player who chooses the best-predicted action using a pruned tree search up to a specified depth with probability $1 - \varepsilon$ and a random valid action with probability ε. The agent can also play against other agents or its former self.

2.3. Traditional RL Agents

Deep Q learning (DQL) has been around for a number of years (Van Hasselt et al., 2016) and has been successfully used in many applications (Wang et al., 2022). Due to its popularity, we use two different versions of DQL models as our baseline for comparisons. Both models are evaluated using a fully connected feed-forward neural network (Voigt, 2020) and a convolutional neural network (CNN). Inspired by a submission to a Kaggle competition (MacPhee-Cobb, 2023), the latter was implemented due to the inherent grid-like structure of the "Connect Four" board, which a convolution may be able to take advantage of. The CNN converts the game state presented in Section 2.1 into two boards: one for each player, with Boolean indicators 0 and 1, indicating where each player has placed their chips. Splitting the state like this may enable the network to differentiate between players more easily.

We train these networks using two different algorithms, presented in the following sections.

2.3.1. DQL with double Q agents (DDQ)

Standard DQL algorithms tend to overestimate Q-values and thus lead to unstable training and poor-performing policies (Van Hasselt et al., 2016). To mitigate the overestimation bias, the double Q trick is implemented by employing two Q networks. One network, referred to as the target network, is utilized for action selection, while the other is used for action evaluation and prediction. The target network gradually copies the Q network's weights periodically through a soft update mechanism (Van Hasselt et al., 2016). This change leads to overall better performance in DQL.

2.3.2. Conservative Q Learning (CQL)

A similar, but more recent method to mitigate the overestimation of expected rewards is the algorithm introduced in Kumar et al. (2020). It provides theoretical improvement guarantees over the original Q learning algorithm and in practice often substantially outperforms other algorithms (Kumar et al., 2020).

2.4. Decision Transformer

One of our main contributions is the implementation of an agent based on the Decision Transformer (DT) architecture (Chen et al., 2021). We use a DT implementation by Huggingface (hug) which is based on the original paper's code.

Instead of training a policy through conventional RL algorithms as with the CQL Agent, we train the agent on collected experience using a sequence modeling objective. The DT requires complete trajectories of episodes of the form $(s_0, g_0, a_0, s_1, g_1, a_1, \dots, s_T, g_T, a_T)$ where s_i represents the state, a_i the action and g_i the "return-to-go" at time step i in an episode of length T. The DT then learns to correlate sequences of states and actions and expected returns with future outcomes via the attention mechanism. Figure 1 shows an illustration of the DT and its input.

A DT has a context window that determines how far the model can look into the past. A context window of size K means that the last K prior time steps are passed into the DT as input. The choice of K is important for the performance of the model and is a trade-off between performance and computational cost. We chose a context window of 21, as in a field of six rows and seven columns (not considering invalid moves) an episode lasts at most 21 time steps. It is not always possible to have entire episodes in the context window. For example in cases where the episodes may be much longer (consider for example the Atari game Starship).

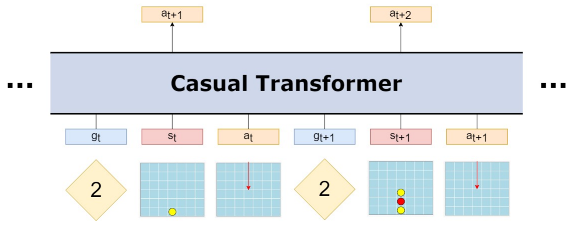

Figure 1. Illustration of the Decision Transformer. The trajectory of the episode up to the current time step is the input, and the next action is the output. The Input is embedded and a positional encoding corresponding to the respective time step is added.

The first major step in the implementation was to collect trajectories of thousands of episodes for training. We collected a set of $100'000$ trajectories where two random players played against each other and we did the same for two Minimax agents. For the Minimax agents, we had to set a probability of the agents performing a random action, such that not all episodes would be equal (as the agents play deterministically). Having collected the trajectories, we performed what is referred to as "Hindsight Return Relabeling". This is the act of labeling each step in each trajectory with its corresponding return-to-go in that episode, which we of course know in hindsight. The following fields are then passed to the DT per trajectory for training: states, actions, rewards, returns-to-go, time steps (for positional encoding) and an attention mask, which isolates only the relevant time steps for prediction.

The actual training of the model is non-trivial as we are in a two-player game setting where we require an opponent for the model. The training is split into two parts: An offline pre-training phase and an online fine-tuning phase. The concept is similar to what Zheng et al. used in their paper "Online Decision Transformer" (2022). In the offline pre-training phase, we feed the model with our generated trajectories T. In the online fine-tuning phase, the goal is to store the best trajectories (according to "return-to-go") in a replay buffer and then repeatedly generate trajectories with the current model and replace the worst trajectories with the new ones in the buffer. We then continuously train the model with the new trajectories. The reason this stage is called fine-tuning is because we always select the same expected return-to-go for each generated trajectory (namely 2) and in that sense, we fine-tune the model's ability to play well and aim for

Algorithm 1 Offline pre-training

```
Input: Trajectories $T$, number of epochs $E$
for each epoch do
    Train the model on trajectories $T$
end for
```

Algorithm 2 Online fine-tuning

```
Input: Trajectories $T$, pre-trained model $M$, buffer size $N$, number of rounds $R$
Initialize replay buffer of size $N$
Store top $N$ trajectories into buffer
for each round in $R$ do
    Generate trajectories with current model
    Remove worst traces from buffer and add new ones
    Train model with replay buffer trajectories
end for
```

that score.

Algorithm 1 and Algorithm 2 give pseudo-code of this procedure. There are a few problems and shortcomings when performing such training with DTs. To train our model online, using this upside-down approach, we need to have a pretty good guess of the expected reward that we want as we need to give an initial "return-to-go" to the model as input. In our case, we can only approximate this value. We know that a win gives a reward of $+1$, but as we also give a reward for each survived move and a negative reward for each invalid move, we don't know the exact value. For our online training, we chose an initial value of 2 as an upper approximation as it corresponds to a win in 20 moves without performing invalid actions. Zheng et al. (2022) propose to adapt this value depending on the resulting total rewards on the future episodes, however, we chose to keep it constant for simplicity.

The main problem we faced while training was that it was difficult to find a proper opponent for training our model. If the opponent plays too poorly, the model will exploit this by following overly simplistic policies. Figure 2 shows an example of such a policy.

If the opponent plays too well, such as a Minimax opponent with a high depth, the model has no chance to properly learn as it never wins (at least not within our training time). Another issue is the stochasticity of our setting. Given a state and a chosen action, the next state is, from the model's point of view, not deterministic, as it can't predict the behavior of the opponent. This stochasticity is a known problem for DTs as the model doesn't know if a successful move it made was a good move or if it was just luck (due to a mistake by the opponent). This issue along with a potential solution has been discussed by Paster et al. in their paper "You Can't Count on Luck" (2022), but implementing their solution is outside the scope of this project.

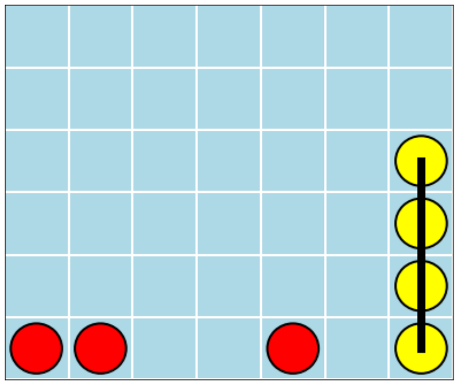

Figure 2. The DT model (yellow) simply learns to stack their chips in one row, as the chance of winning with this policy is pretty high if the opponent (red) plays randomly.

3. Results

3.1. Evaluation Setup

The FCNN and CNN networks are trained for 2000 episodes against a Minimax opponent set to a depth of 2 and choose a random action with a probability of 0.3. All evaluations against the Minimax algorithm are done using the same parameters. The initial epsilon of the agent during training is set to $\varepsilon = 1$ with a decay factor of 0.997 per move and a minimum of $\varepsilon = 0.01$. For evaluating agents against opponents other than random and Minimax, we keep a small randomness factor of 0.1. This is done to prevent them from playing purely deterministically, as that would mean that the same game is played every time during evaluation, which doesn't give very meaningful results. The transformer is trained as described in Section 2.4.

3.2. Transformer Performance

When looking at Table 1, it is apparent that the DT performs markedly better than the other models, achieving the highest winning percentage against 5 out of the 7 agents we evaluated. This shows that it is indeed feasible to perform reinforcement learning using transformers and that they can outperform traditional RL methods. With the setup we are using, it even outperforms the Minimax algorithm. The winning rate quickly decreases though with increasing search depth and decreasing randomness.

Table 2 shows the results of further training the offline trained transformer online against a Minimax opponent with $\varepsilon = 0.0$. We stipulate that it learns to play better against that specific opponent, but due to the deterministic nature of pure Minimax, training against it worsens the performance

Agent	RANDOM	MINIMAX	DQN-FCNN	CQL-FCNN	DQN-CNN	CQL-CNN	DT
RANDOM	55.83 ± 2.23	4.87 ± 1.39	43.50 ± 9.18	23.47 ± 2.20	50.83 ± 14.06	43.27 ± 1.72	27.80 ± 1.21
MINIMAX	$\mathbf{97.77 \pm 0.55}$	65.53 ± 0.96	95.87 ± 0.81	85.57 ± 1.35	96.30 ± 3.29	91.83 ± 1.91	40.13 ± 1.91
DQN-FCNN	69.07 ± 7.45	8.63 ± 4.55	50.73 ± 10.79	30.57 ± 37.28	54.57 ± 27.02	35.40 ± 28.50	28.3 ± 44.60
CQL-FCNN	83.93 ± 2.64	9.27 ± 0.31	72.43 ± 36.95	58.40 ± 2.36	93.07 ± 0.42	48.10 ± 19.23	$\mathbf{50.07 \pm 41.62}$
DQN-CNN	53.23 ± 8.98	4.53 ± 2.15	28.70 ± 12.17	52.00 ± 24.15	59.70 ± 4.85	58.97 ± 2.41	4.73 ± 2.28
CQL-CNN	81.00 ± 1.15	8.63 ± 1.10	93.83 ± 2.41	79.53 ± 2.74	94.60 ± 1.55	52.60 ± 2.29	24.00 ± 37.95
DT	85.40 ± 0.95	$\mathbf{94.33 \pm 0.35}$	$\mathbf{98.13 \pm 1.32}$	$\mathbf{97.37 \pm 2.55}$	$\mathbf{98.6 \pm 0.72}$	$\mathbf{97.70 \pm 1.40}$	–
Parameters	–	–	23.1k	23.1k	72.8k	72.8k	1264.1k
FLOPs	–	–	11.3k	11.3k	51.5k	51.5k	199.3k

Table 1. Mean and standard deviation of the winning ratios in percent out of 1000 games across 3 random seeds. The agent on the left (first column) always starts the game. FLOPs are given as the FLOP count of the underlying network for one forward pass with a batch size of 1. The highest values in each column are marked in boldface.

against opponents with some inherent randomness.

Training	Traces	vs. MINIMAX	vs. RANDOM
Offline	Random	$\mathbf{77.10 \pm 18.92}$	$\mathbf{78.80 \pm 7.30}$
Offline	Minimax	76.00 ± 17.95	77.08 ± 4.81
Online	Random	34.73 ± 14.53	62.02 ± 2.97
Online	Minimax	34.73 ± 14.53	62.02 ± 2.97

Table 2. Ablation study for the decision transformer. Win rates are given in percent out of 2000 games with the agent and opponent starting 1000 times each.

3.3. Comparison of Traditional Methods

For the traditional agents, a high correlation between winning rates against random opponents and Minimax opponents can be observed. Training with the more recent CQL algorithm (Section 2.3.2) allows for higher performance compared to the DQN algorithm (Section 2.3.1). The same correlation doesn't hold when they play against other traditional agents—there the winning rates vary much more between different random seeds as well, leading to a high standard deviation and less reliable results. Further testing would be needed to decrease the uncertainty of these results.

3.4. Computational Efficiency

The DT requires almost 18 times more compute than the FCNN models, and almost four times as much as the CNN models. It uses 55 and 17 times as many parameters as the FCNN and CNN models, respectively. This difference is offset due to the better performance and reliability of the DT. For low-power and energy-efficient devices, the traditional methods may be the preferred choice, but we believe that on modern computers the DT offers good results at an acceptable payoff. Using 8-bit integer quantization, we could feasibly reduce memory usage by $4\times$ and decrease inference time without a significant drop in accuracy (Jacob et al., 2018).

4. Discussion and Summary

The main difficulty in our project was getting the agent to learn more than the local minimum of attempting to build towers of four and instead aiming to connect four diagonally and horizontally as well. This was a challenge for all the evaluated models, including the DT.

We also faced challenges when training the DT, particularly in opponent modeling, where too simplistic or advanced opponents hindered nuanced strategy learning. This underscores the delicate balance required in environmental design for effective training and disagrees with the proposed stability of transformer training in RL. Paster et al. (2022) also argue that the DT has problems with learning in stochastic environments due to the element of luck involved. Still, our DT model achieved much more reliable results with a smaller standard deviation when compared to the traditional algorithms.

Online training of the DT worsened its performance, but we stipulate that with further optimization of the training procedure and by applying the method proposed in Paster et al. (2022), this trend could be mitigated or possibly reversed. Evaluating our agents on a partial information game would affirm the transformer's ability to learn in complex RL environments. Training other types of models such as an LSTM (Hochreiter & Schmidhuber, 1997) using the same method as with the DT (Chen et al., 2021) would allow for a direct comparison of models unaffected by the training method.

In conclusion, we show that the RL problem can be reformulated as a sequence modeling task and that the Decision Transformer (Chen et al., 2021) can successfully learn to play and even outperform traditional agents, showcased for the strategic full-information game "Connect Four". This illuminates the potential for broader applications of this method in more complex full- as well as partial-information games.

References

Hugging face. URL https://huggingface.co. Last accessed: 20.12.2023.

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. *CoRR*, abs/2106.01345, 2021. URL https://arxiv.org/abs/2106.01345.

D'Hulst, L., Gupta, S., Fabbri-Garcia, C., and Stefan, D. Training a deep Q learning network for Connect 4, 2022. URL https://medium.com/@louisdhulst/training-a-deep-q-learning-network-for-connect-4-9694e56cb806. Last accessed: 03.01.2024.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative Q-learning for offline reinforcement learning. *CoRR*, abs/2006.04779, 2020. URL https://arxiv.org/abs/2006.04779.

Li, W., Luo, H., Lin, Z., Zhang, C., Lu, Z., and Ye, D. A survey on transformers in reinforcement learning, 2023.

MacPhee-Cobb, L. Reinforce RL - Kaggle ConnectX competition, 2023. URL https://www.kaggle.com/code/wrinkledtime/reinforce-rl. Last accessed: 06.01.2024.

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, 2020.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Voigt, S. Connect Four - Deep reinforcement learning, 2020. URL https://www.voigtstefan.me/post/connectx/. Last accessed: 04.01.2024.

Wang, X., Wang, S., Liang, X., Zhao, D., Huang, J., Xu, X., Dai, B., and Miao, Q. Deep reinforcement learning: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27042–27059. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/zheng22c.html.