Performance of Decision Transformer in multi-task offline reinforcement learning

How does the introduction of sub-optimal data affect the performance of the model?

Piotr Bieszczad[1]
Supervisor(s): Matthijs Spaan[1]**, Max Weltevrede**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Piotr Bieszczad
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Max Weltevrede, Elena Congeduti

Abstract

In the field of Artificial Intelligence (AI), techniques like Reinforcement Learning (RL) and Decision Transformer (DT) are utilized by machines to learn from experiences and solve problems. The distinction between offline and online learning determines whether the machine learns from a live environment or simply observes pre-recorded actions. The difference between single-task and multi-task settings indicates whether the machine can handle similar but not identical tasks. Multi-task, offline learning, the focus of this paper, allows machines to address a variety of related tasks, based on a pre-recorded set of experiences. This approach is particularly valuable in situations where traditional training methods are costly or challenging. For instance, in robotics, multi-task, offline learning enables robots to use experiences from various tasks, such as picking up objects, to solve new problems like placing them down. This research paper explores the effectiveness of Decision Transformers in multi-task environments through theoretical discussions and practical examples. It also tries to answer the question, of how introducing sub-optimal training data, affects the performance or generalisation ability of the model.

1 Introduction

Offline reinforcement learning (RL) has garnered attention as a viable approach for training agents in scenarios where real-time interaction with the environment is either impractical or prohibitively costly. While traditional RL methods excel in environments where agents can continuously interact and learn from their experiences, offline RL presents a unique challenge due to the absence of such real-time feedback loops.

Recent studies, such as "The Generalization Gap in Offline Reinforcement Learning" [6], have shed light on a critical issue known as the Generalization Gap in Offline Reinforcement Learning, which underscores the difficulty of effectively transferring policies learned from pre-existing datasets to new, unseen tasks, particularly in multi-task environments.

The goal of this research paper is to explore this difficulty particularly when dealing with Decision Transformer (DT) [3], and investigate possible solutions for that issue. The decision transformer is specific in its architecture, because of the self-attention mechanism. More details can be found in section 2.1. Specifically, the study seeks to produce benchmarks for current performance and explore strategies aimed at improving DT's ability to generalize policies across diverse tasks and environments.

[6] inspires this study, and thus I am going to use a similar approach, to the one taken in [6]. Specifically, I am going to use Behaviour cloning as the benchmark for the model. The reason for choosing BC is that this is a widely used benchmark in the field of offline RL. More details can be found in section 3.2.

I am going to research, whether introducing sub-optimal data can help the model improve its performance and generalisation. While similar tests were already performed in [6], it is important, that the achieved results are reproduced and confirmed. It is also beneficial to look more in-depth into possible solutions for this issue.

From that, the following questions arise:

- How does the offline trained Decision Transformer perform in a multi-task environment?
- How does introducing sub-optimal data affect the performance of the DT?

To answer the above questions, an experiment to assess the performance of DT trained on optimal and sub-optimal data will be conducted. More details can be found in section 4. The results are analysed in section 5.

The significance of this research extends beyond theoretical advancements, particularly in domains such as robotics, where the practical implications of offline RL are profound. Enhancing the performance of DT in offline RL settings can result in significant cost savings as it enables more efficient training of robotic agents without the need for continuous interaction with real-world environments. Through this research endeavour, I aspire to contribute to the ongoing advancement of offline RL methodologies. By providing a deeper understanding of how DT can be optimized in offline RL settings, this study aims to contribute to the development of more efficient, adaptable, and scalable AI systems, ultimately driving progress towards more intelligent and capable autonomous agents.

2 Background

Learning in a Markov decision process (MDP) is considered, characterized by the tuple (S, A, P, R). This tuple includes states $s \in S$, actions $a \in A$, transition probabilities $P(s'|s, a)$, and a reward function $r = R(s, a)$. The state, action, and reward at time step t are denoted by s_t, a_t, and $r_t = R(s_t, a_t)$, respectively. A trajectory is a series of states, actions, and rewards, represented as $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$. The return at time step t, $R_t = \sum_{t'=t}^{T} r_{t'}$, is the cumulative sum of rewards from that point forward. The objective in reinforcement learning is to develop a policy that maximizes the expected return $\mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$ in an MDP. In offline reinforcement learning, rather than generating data through direct interaction with the environment, a predetermined, finite dataset composed of trajectory rollouts from various policies is used. This approach is more challenging because it eliminates the agent's ability to explore the environment and gather new feedback. [3]

2.1 Decision Transformer

Transformers, introduced by Vaswani et al. [12], are designed to efficiently model sequential data. These models are composed of stacked self-attention layers with residual connections. Each self-attention layer processes n embeddings $\{x_i\}_{i=1}^{n}$ corresponding to distinct input tokens and produces n embeddings $\{z_i\}_{i=1}^{n}$, maintaining the input dimensions. Each token i is mapped through linear transformations to a key k_i, query q_i, and value v_i. The output for the i-th token from the self-attention layer is calculated by weighting the values v_j with the normalized dot product between the query q_i and the keys k_j:

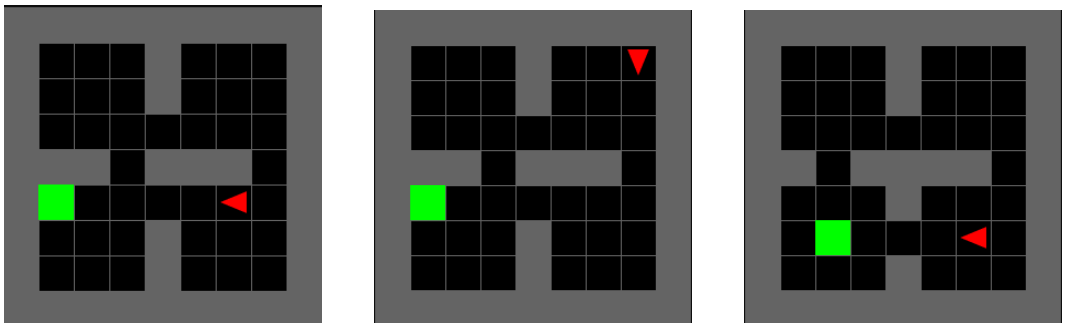
$$z_i = \sum_{j=1}^{n} \text{softmax}\left(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n}\right)_j \cdot v_j. \quad (1)$$

This mechanism allows the layer to implicitly form state-return associations by assigning "credit" through the similarity of the query and key vectors (maximizing the dot product). In

this work, the GPT architecture [7] is employed, which adapts the transformer architecture with a causal self-attention mask to facilitate autoregressive generation. This modification involves replacing the summation/softmax over the n tokens with a summation/softmax over only the preceding tokens in the sequence ($j \in [1, i]$). [3]

2.2 Reach ability

In my work, I am going to use the concept of reachability as described in [13]. To summarise the idea: A reachable state s_r as a state for which there exists a policy whose probability of encountering s_r during training is non-zero. For example:



(a) Original (b) Reachable (c) Unreachable

Figure 1: Reach ability

I have used this distinction, as the reachable states, should be easier to solve for the agent. Usually, the reward obtained by the agent is quite low, therefore to have a better understanding of the results, I used the *train_reachable* for validation and *test_reachable* for testing.

3 Methodology

This study aims to reproduce the findings regarding the DT presented in [6]. Moreover, this research tests how augmenting the training data with sub-optimal behaviour affects the model's performance.

To reproduce and extend the results, the following steps will be undertaken:

1. **Assess the performance of the Decision Transformer trained on an expert dataset** (more about datasets in section 4.2) in a multi-task setting. Compare it to both benchmarks described in section 3.2. A common practice in Multitask Offline RL is to compare the performance of models within a toy environment. A similar methodology can be found in many other related papers, for example, [6]. The idea is to pre-record datasets of interactions, train the model on that data and assess its performance in new, unseen environments. Thus the setup described below was created.

2. **Augment the training data with sub-optimal paths (see section 4.2) and investigate the impact on the model's performance.** The main idea of this approach is to improve the generalisation capabilities of the model. This idea also known as 'path stitching' is visually described in figure 2. To provide an answer to that research sub-question, I have augmented the dataset as described in section 4.2 and repeated the procedure conducted in case of the optimal dataset.

Figure 2: Path stitching [3]

3.1 Environment: 4-Room Grid World

My research utilizes a small grid world composed of four interconnected rooms (illustrated in Figure 3), which is a variant of the Four Rooms environment from the MiniGrid benchmark [4]. In this setting, the entire state space, encompassing all tasks, is fully observable. The agent can perform three actions: moving forward, turning left, and turning right. The primary difference between the training and testing phases lies in the initial states s_0 (or tasks), which can vary based on four key factors: the configuration of the four rooms, the agent's starting position, the agent's initial orientation, and the goal location. During the training phase, the agent is exposed to a fixed set of starting states, while in the testing phase, it encounters new starting states drawn from the same distribution. In this environment, the reach ability of the goal is influenced by changes in the goal's location and the room layout [13].

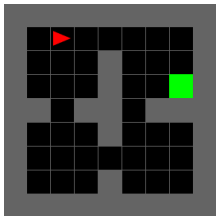


Figure 3: Example of the Four-Room environment.

3.2 Benchmark models

To ensure a reliable comparison of the model performance, the following benchmark models will be used:

- Optimal trajectory - to get an objective measure for the performance of DT, one of the benchmarks is to compare the number of steps taken in the Four-Room environment (see section 3.1) to the number of steps required by the shortest path algorithm. This

approach helps to visualise better the performance of the model across different tasks. (see example in figure 10)

- Behaviour Cloning - even though Optimal Trajectory is a more objective measure of the model performance, for convenient comparison of the methods across different environments, Behaviour Cloning [5] is also going to be used as a benchmark. This method is a well-known baseline in the field of Offline RL. It was also used as a benchmark in the following papers: [1], [9], [14], [8].

3.3 Models source code

For the experiment run the d3rlpy library [10] was used. This library supports the DT that is compatible with discrete tasks. Also for the benchmark the Behaviour Cloning (BC) model from that same library was used.

4 Experimental Setup and Results

4.1 Environments

The environment was used in the following 3 configurations (Configuration - location of doors, agent starting point and the goal at the beginning of the episode):

- *train* - initial subset of different configurations. This configuration was used to create the training datasets.
- *validation_reachable* - a reachable subset of different configurations (only agent start position and goal position changed). This configuration was used as a validation environment.
- *test_reachable* - a reachable subset of different configurations (only agent start position and goal position changed). This configuration was used a reachable test environment.
- *test_unreachable* - an unreachable subset of different configurations (the door position/goal pose was changed, so the agent could not have achieved this state during training). This configuration was only used for testing purposes.

4.2 Datasets

For the exploration of how training data influences the generalisation of the model, the following datasets were created:

- *train_optimal* - This is a dataset, that was created using the *train* configuration. To create this dataset, the agent was performing only the optimal actions.
- *train_sub-optimal* - This is a dataset, that was created using the *train* configuration. To create the dataset, the agent 50% of the time took the optimal action, and 50% of the time, the action was random.

4.3 Performance metric

The metric used for evaluation is the average reward returned by environments. For evaluation the reachable *test_reachable* and unreachable *test_unreachable* were used. Each of them consists of 40 different configurations. The environment returns 1 if the agent reaches the goal within 20 steps. Otherwise, it returns 0.

For the evaluation, the model is run on 40 different configurations. The *Returned reward* is the fraction of environments, that the model solved. E.g if the model solves $\frac{11}{40}$ configurations it receives the reward of 0.275

4.4 Process plan

To conduct the experiment the following steps were undertaken:

1. Create the datasets (see section 4.2): (*train_optimal*, *train_sub-optimal*).
2. Run hyperparameter sweep to elicit correct parameters for each dataset.
3. Run the models on datasets.
4. Gather and discuss the results.

4.5 Hyperparameters DT

For the hyperparameter tuning, the performance was evaluated on *validation_reachable*. The y-axis *"returned reward"* relates to the fraction of environments solved by the model.

To better understand the impact hyperparameters have on the model, first a linear search was conducted. Each hyperparameter is represented by 5 seeds. In this part, the model was trained on *train*. The following hyperparameters were tuned:

- Learning rate - the model uses a cosine learning curve. The model was evaluated for the following values of learning rate: $[5 \times 10^{-3}, 1 \times 10^{-4}, 5 \times 10^{-4}, 6 \times 10^{-5}]$

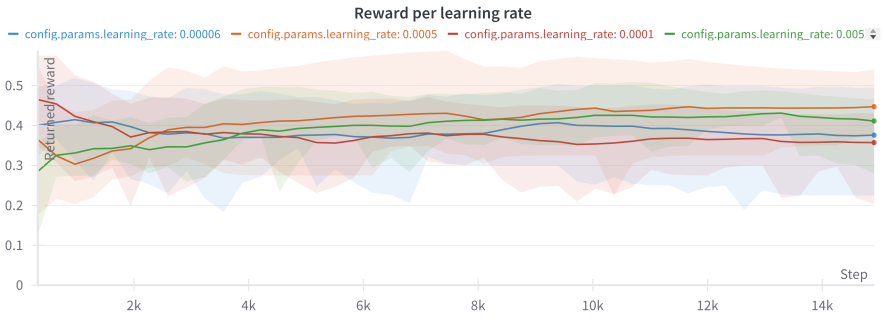


Figure 4: Reward per learning rate

As can be seen in figure 4 learning rate of 0.0005 performed best, although it is important to emphasize, that there is no significant correlation between the value of the learning rate and returned reward. (The changes are minor.)

- Batch size - for small datasets, the batch size close to the size of the dataset is usually considered a good choice. Therefore the batch sizes of $[127, 324, 526]$ were considered.

- Context length - Decision transformer uses 'context length' as an important hyper-parameter [3]. The model was evaluated for the following values of the context size: $[5, 10, 15, 20, 30]$. As can be seen in figure 5 the value of 15 performed best, although it is important to emphasize, that there is no significant correlation between the value of the learning rate and the returned reward. (The changes are minor.)

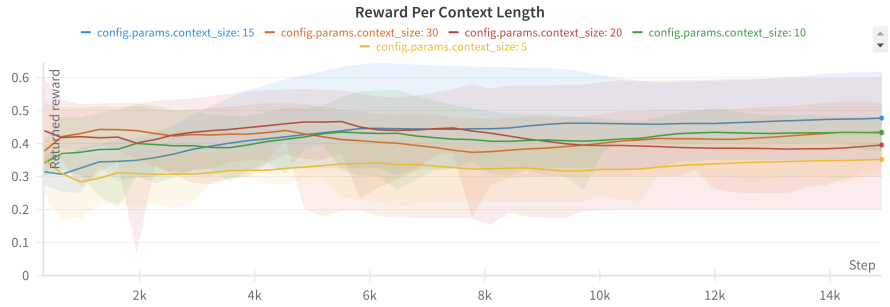


Figure 5: Reward per context length

- Target reward - decision transformer uses target reward to generate the next action that achieves the given reward [3]. Through experimentation, the target reward of 0 was chosen, as it performed best. Notice, that the top 5 performance graphs correspond to a target reward of 0, as shown in figure 6.

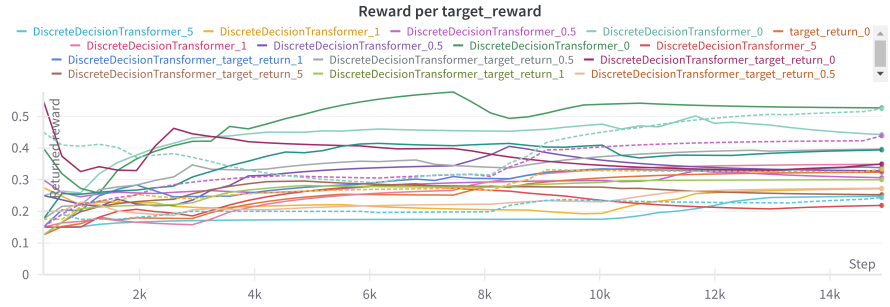


Figure 6: Reward per target reward

The value of *target_return* equal to 0 may seem counter-intuitive because we want the reward to be maximized, however, it makes more sense when considering, that the reward in Four-Room Environment is very sparse, so most of the time, the reward of 0 'perfectly fits' in the context of making decision-based on the training data. Also as described in [3] blindly increasing the target reward does not actually improve the performance of the model.

- #steps - the training dataset is very small, (324 steps for optimal interactions). Thus an optimal number of steps needs to be chosen to balance the training and overfitting. It can be seen, that all models achieve their full potential around 15k steps on validation set. On the training environment, because of its small size, the optimal performance is achieved after about 1000 steps. Thus 15k steps was chosen as the base metric for the experiments.

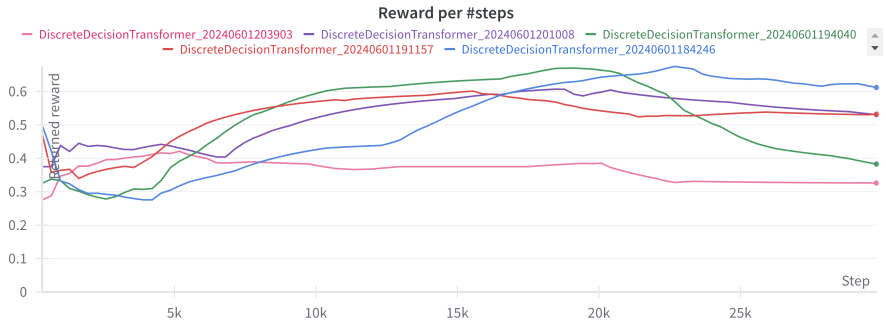


Figure 7: Reward per #steps

After the initial linear search, a Bayesian search was performed separately for *train_optimal* and *train_stumble*. The search for both datasets consisted of 40 randomly chosen models.

4.5.1 Bayesian Search for *train_optimal*

Based on the search described in Appendix A, the following hyperparameters were elicited:

Table 1: DT Optimal Parameters

Parameter	Value
target_return	0
learning_rate	0.0005
context_size	20
#steps	15k
batch_size	324

4.5.2 Bayesian search for *train_suboptimal*

Based on the Bayesian parameter search described in Appendix A, the following hyperparameters were elicited:

Table 2: DT Sub-optimal Parameters

Parameter	Value
target_return	0
learning_rate	0.0005
context_size	20
#steps	15k
batch_size	324

4.6 Hyperparameters BC

For the hyperparameter search, the same methodology described in Appendix A was carried out, with the exception, that the BC was optimized only for *batch_size* and *learning_rate*, as the rest of hyperparameters were DT-specific. Based on the search, the following values were found:

Table 3: BC Optimal Parameters

Parameter	Value
learning_rate	0.0005
#steps	15k
batch_size	324

4.6.1 Grid search for *train_stumble*

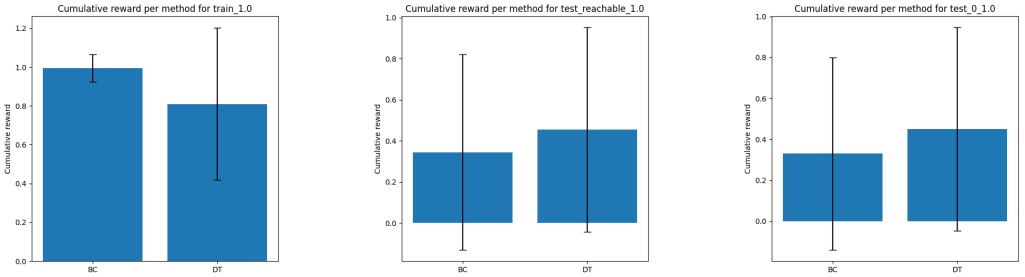
Based on the search, the following hyperparameters were elicited:

Table 4: DT Sub-optimal Parameters

Parameter	Value
learning_rate	0.0005
#steps	15k
batch_size	324

4.7 Optimal dataset performance

After running the evaluation of the models trained on 5 seeds on optimal data, the following results were found:

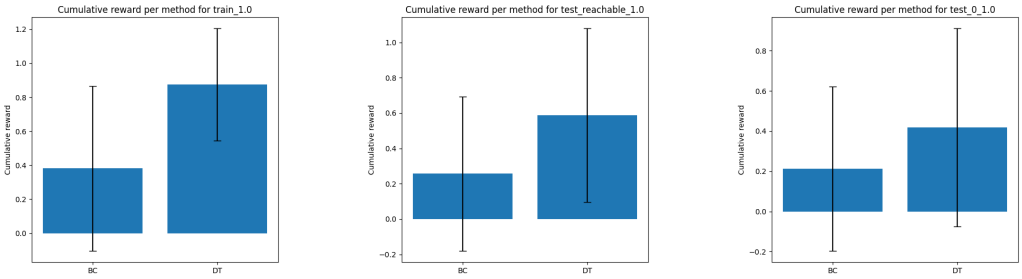


(a) Train data (b) Reachable test data (c) Unreachable test data

Figure 8: Results of Optimal Evaluations

4.8 Sub-optimal dataset performance

After running the evaluation on of the models trained on 5 seeds on sub-optimal data, the following results were found:



(a) Train data (b) Reachable test data (c) Unreachable test data

Figure 9: Results of Sub-optimal Evaluations

4.9 Example runs

The figures below show the number of steps for each model in each environment compared to the optimal policy. The number of steps equal to -1, indicates, that the level was not solved.

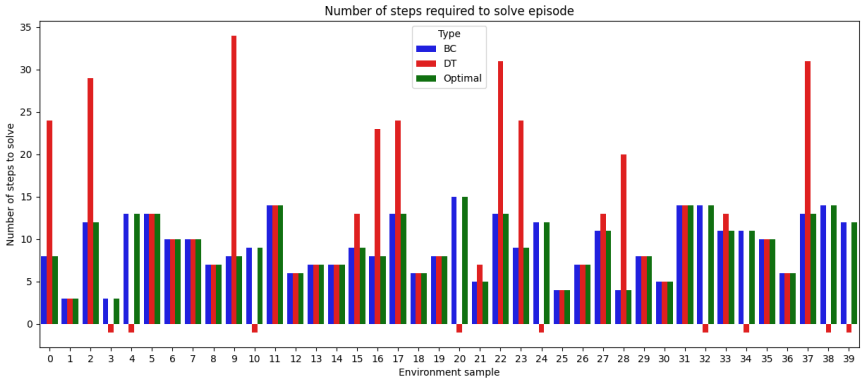


Figure 10: Runs for models trained on optimal train data, evaluated on train environment

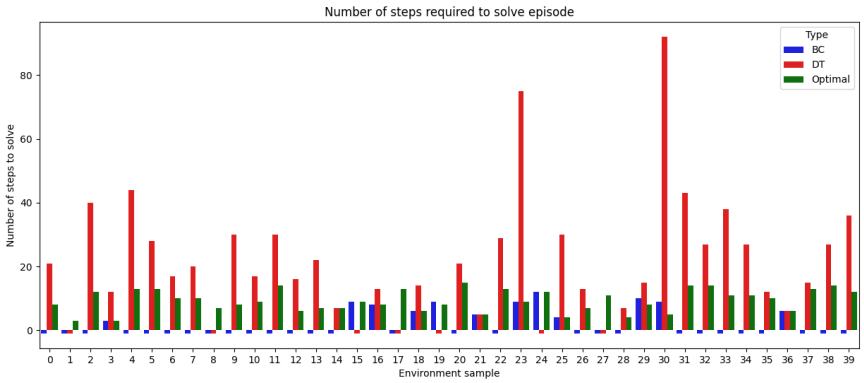
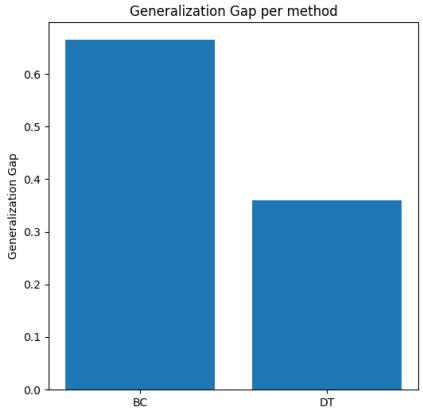


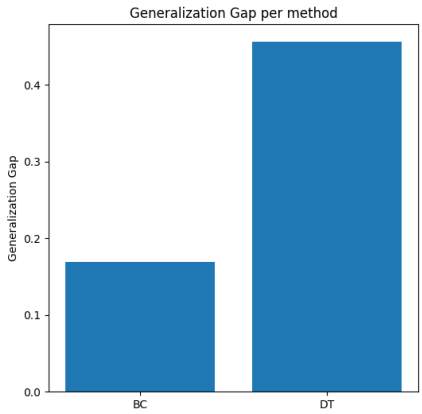
Figure 11: Runs for models trained on sub-optimal train data, evaluated on train environment

4.10 Generalization Gap

The generalization gap of the model was computed as the difference in average performance between results obtained in the training environment, and results obtained in an unreachable environment.



(a) Generalization gap when trained on optimal data



(b) Generalization gap when trained on sub-optimal data

Figure 12: Generalization Gap

5 Conclusions

5.1 Optimal dataset performance

As was expected, and also confirming the results from [6] BC outperforms DT when trained on the optimal dataset when evaluated on the training data. When evaluated on unseen data (Reachable and Unreachable), both models experience a significant drop in performance. (About 40% drop in accuracy for DT and 60% drop in accuracy for BC). As can be seen in figure 8 The variance in average results for both models is quite big, which indicates the involvement of randomness, which is typical in those scenarios. The only reliable performance was obtained by BC when trained on optimal data and evaluated on the same data. In this experiment, the DT outperformed BC on unseen data, which is an interesting result. This might be due to the ability of the DT to capture longer contexts, and not only focus on the current state, which was an advantage in case of unseen problems.

5.2 Sub-optimal dataset performance

When the model was trained on the sub-optimal dataset (see section 4.2), the performance increased for DT when evaluated on train and reachable data, and dropped slightly when evaluated on unreachable data. That behaviour is caused by the introduction of additional (although sub-optimal) data in the form of a mixed dataset. The DT has seen more examples of "the same" (meaning reachable) environments and therefore was able to use this data to solve testing examples. The performance however decreased for BC for all the testing environments. The decrease in the performance of BC (especially when evaluating the training set), is in line with the expectations. As shown in [6], the performance of BC decreases when sub-optimal data is introduced. This is due to the fact, that while BC focuses on cloning behaviour, it doesn't capture longer contexts, and the sub-optimal data only made it more difficult for this model to learn, as the training examples had conflicting actions for the same state.

5.3 Steps per environment run

Another interesting insight that can be found in this study is shown when comparing figure 10 and figure 11. When trained on optimal data, BC usually performs perfectly, and the DT either fails to complete the task or does not complete it optimally. However, when trained on sub-optimal data, the BC is rarely able to solve any of the instances of the training environment, however, the DT manages the majority of them, even though its behaviour is most of the time far from optimal.

5.4 Answer research question

To answer the main research question of this project, overall the performance of the decision transformer in Offline RL is quite poor. When trained on the seen examples, it reaches a 'decent' score of at most around 80% accuracy, therefore the model is able to learn from examples and reproduce them. However, when tested on unseen environments, the performance does not consistently cross 60% of solved environments. That makes the decision transformer not a reliable solution for most applications. Especially considering, that the DT is a complex model. Training time is 10 times longer (on average) compared to BC, while the results are similar.

5.5 Answer sub-question

To answer the sub-question, introducing sub-optimal data improved the performance of the Decision Transformer, when evaluating on train and reachable environment, and didn't really affect it, when evaluating on unseen environments. This shows, that the performance of the model is heavily data dependent (it is starving for data) and introducing new data (even sub-optimal) can increase the performance of the model. This is an interesting finding, as introducing sub-optimal data to the model can be cheap in many scenarios, thus creating an approachable option for many cases.

5.6 Model stability & Hyperparameters

Based on the results presented in Appendix A, there is one very significant conclusion I would like to discuss, and that is the stability of the model. The offline RL algorithms are known to be quite unstable. [11] From the plots shown in figures 14, and 17, one can see, that the seed has importance **that is comparable or higher**, to the importance of other hyperparameters. That means, that random initialization of the model **has higher or comparable significance** to its hyperparameters. I believe the reason for this problem is that there is not enough data. However, this is the very core of offline RL, thus (significant) augmentation of dataset size was not considered in this experiment, as it is not the goal of Offline RL. To mitigate the issue of model instability, hyperparameter searches, as well as the actual test runs, were performed on multiple seeds, and the average result was taken.

5.7 Generalization gap

It is important to correctly interpret the results shown in figure 12. It is important, to consider not only the gap itself but also the performance of the model. For example, the drop in gap for the BC is really only correlated to the drop in the performance of BC in the training environment when trained on sub-optimal data. The generalization gap For the DT, even though it seems to increase, it is again correlated only to change in performance in only one of the testing environments. In this case, it was caused by an improvement in performance when evaluating the training dataset.

6 Future work

The future work for this study could include two possible research questions:

1. How does understanding of the task scale with the amount of training data?" - As I mentioned in section 5.6 the models are greatly unstable because of insufficient data, it would be interesting to see how that changes when more training data is introduced.

2. What alternatives are there for Multitask Offline RL since the traditional methods seem to fail? - I have found [8] as a promising solution. It would be interesting to see how it performs against the benchmarks introduced in this paper.

3. The Performance of DT increased when new, suboptimal data was introduced. It would be interesting to see if there is any cut-off point, where introducing more sub-optimal data does not further increase the performance of the model.

7 Responsible Research

In my study, I used the d3rlpy library [10] and the Four Room environment (see section 3.1), both of which are open-source, ensuring reproducibility and transparency in my research endeavours.

Improving the generalization of offline learning methods is crucial for cultivating more resilient and dependable autonomous agents capable of making rational decisions even in unfamiliar states—a necessity for their deployment in real-world applications. However, deploying such agents in high-stakes scenarios requires careful consideration, as it can entail negative repercussions. Therefore, it's imperative to implement additional safety measures when contemplating real-world deployment.

Given that my findings are derived from a simulated environment (Four-Room), which offer simplified representations compared to real-world settings, I anticipate no direct adverse effects on society. These simulations serve as useful proxies for problem-solving scenarios, allowing to explore potential outcomes without posing any immediate harm.

7.1 Use of LLMs

ChatGPT was used as a 'proofreader' for this research paper. The author is aware of the limitations of LLMs, that they make mistakes and should not be relied on.

A Optimal Hyperparameters

To find the best combination of hyperparameters for DT trained on the optimal dataset, after the initial linear search I have done a hyperparameter sweep. The sweep was performed using the Bayesian method, to avoid the unpromising combinations in order to save computational time and the environment. There was a total of 40 different models trained. The run per model can be seen in figure 13. From the runs, using visualisation tools of wandb [2], I have used the figure 14 to understand the importance of the hyperparameters, and also figure 15 to elicit the best hyperparameters. The same procedure was repeated:

- For DT trained on suboptimal: $[16, 17, 18]$
- For BC trained on optimal: $[19, 20, 21]$
- For BC trained on sub-optimal: $[22, 20, 24]$

A.1 Hyperparameter search DT optimal

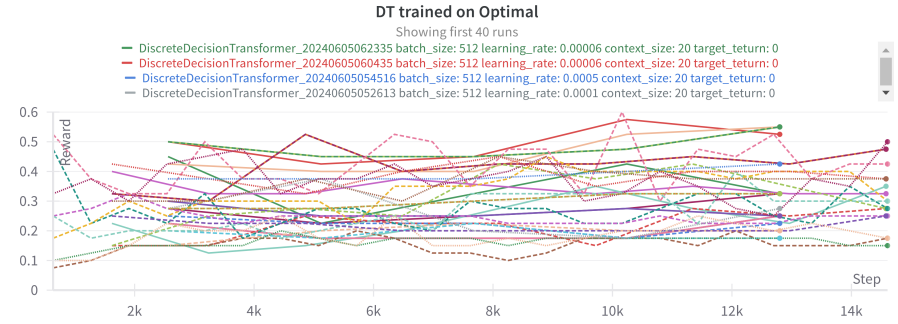


Figure 13: DT optimal runs

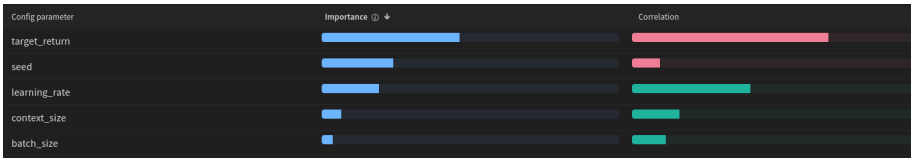


Figure 14: DT optimal Parameter Importance

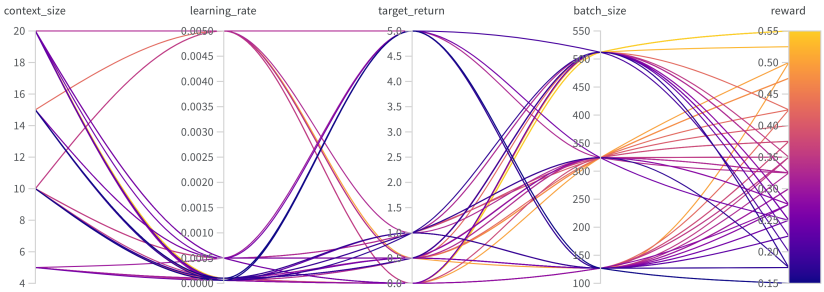


Figure 15: DT reward, hyperparameter correlation.

A.2 Hyperparameter search DT sub-optimal

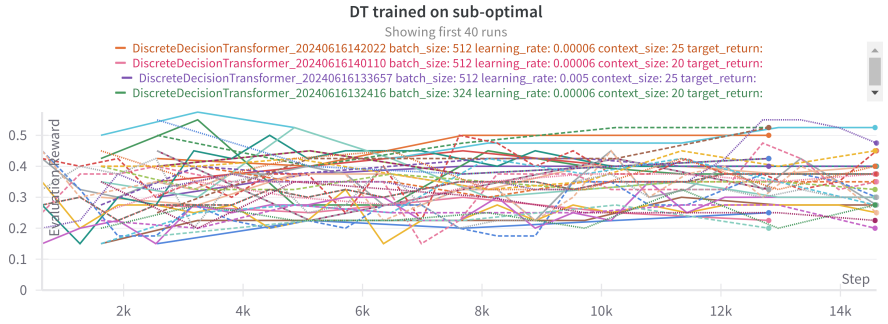


Figure 16: DT suboptimal runs

Figure 17: DT suboptimal Parameter Importance

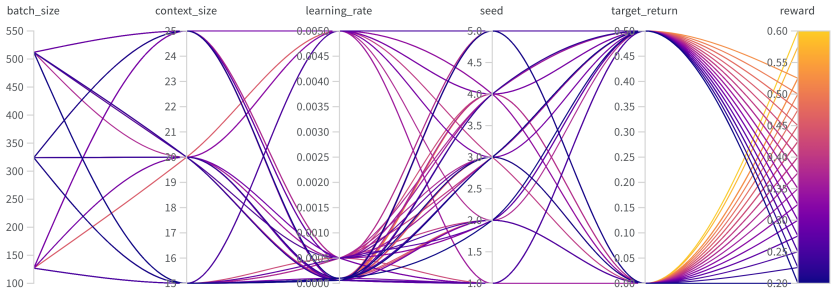


Figure 18: DT reward, hyperparameter correlation.

A.3 Hyperparameter search BC optimal

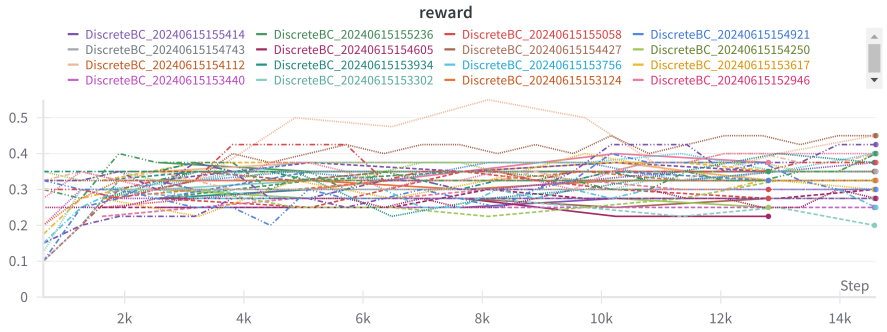


Figure 19: BC optimal runs

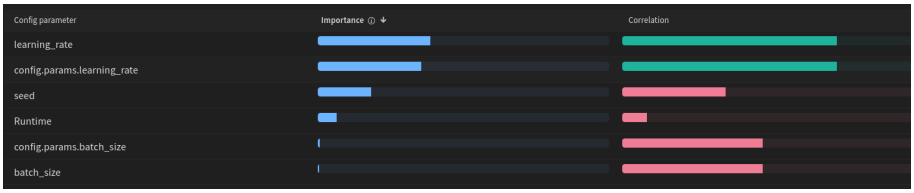


Figure 20: BC optimal Parameter Importance

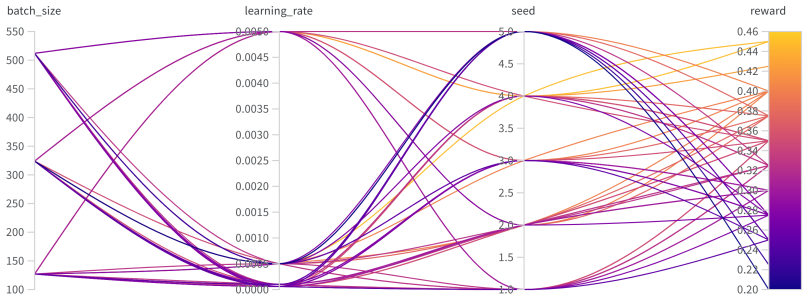


Figure 21: BC reward, hyperparameter correlation.

A.4 Hyperparameter search BC sub-optimal

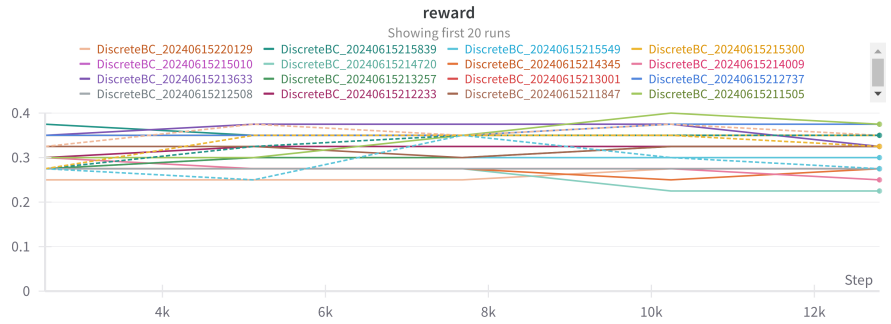


Figure 22: BC optimal runs

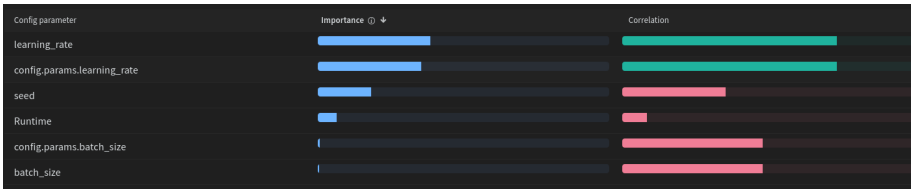


Figure 23: BC optimal Parameter Importance

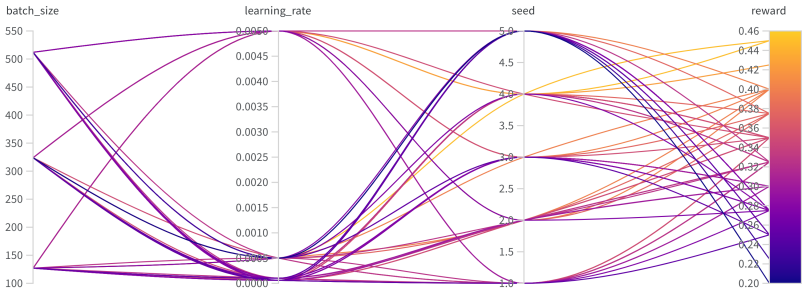


Figure 24: BC reward, hyperparameter correlation.

References

[1] S. Akshay, Nathalie Bertrand, Serge Haddad, and Loïc Hélouët. The Steady-State Control Problem for Markov Decision Processes. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Kaustubh Joshi, Markus

Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio, editors, *Quantitative Evaluation of Systems*, volume 8054, pages 290–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. Series Title: Lecture Notes in Computer Science.

[2] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[3] Lili Chen and et al. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in neural information processing systems 34*, pages 15084–15097. arXiv, 2021.

[4] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym, 2018. arXiv preprint arXiv:1802.09464.

[5] Wonjoon Goo and Scott Niekum. Know Your Boundaries: The Necessity of Explicit Behavioral Cloning in Offline RL, June 2022. arXiv:2206.00695 [cs].

[6] Ishita Mediratta and et al. The generalization gap in offline reinforcement learning, 2023. arXiv preprint arXiv:2312.05742.

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018. OpenAI.

[8] Alfredo Reichlin, Miguel Vasco, Hang Yin, and Danica Kragic. Goal-Conditioned Offline Reinforcement Learning via Metric Learning, February 2024. arXiv:2402.10820 [cs].

[9] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can Wikipedia Help Offline Reinforcement Learning?, July 2022. arXiv:2201.12122 [cs].

[10] Takuma Seno and Michita Imai. d3rlpy: An offline deep reinforcement learning library. *Journal of Machine Learning Research*, 23(315):1–20, 2022.

[11] Zizhou Su. The Least Restriction for Offline Reinforcement Learning, July 2021. arXiv:2107.01757 [cs].

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].

[13] Max Weltevrede, Matthijs T. J. Spaan, and Wendelin Böhmer. The Role of Diverse Replay for Generalisation in Reinforcement Learning, August 2023. arXiv:2306.05727 [cs].

[14] Zilai Zeng, Ce Zhang, Shijie Wang, and Chen Sun. Goal-Conditioned Predictive Coding for Offline Reinforcement Learning, October 2023. arXiv:2307.03406 [cs].