Towards Better Offline RL with Structured State Space Sequence (S4) Models

Kaustubh Dighe* **Muhender Raj Rajvee*** **Shayan Pardis***
{kdighe, muhender, shayanp}@mit.edu
Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

1 Introduction

Janner, Li, and Levine[1] introduced the idea of viewing reinforcement learning as a sequence generation problem. This allows utilizing the machinery developed for large-scale unsupervised learning. Introduced in Chen *et al.*[2], Decision Transformers (DT) is one of the most successful realization of this idea. Among other metrics, the paper evaluates the model with credit assignment as a metric.

Credit assignment is a fundamental problem in reinforcement learning, that of measuring the influence of an action on future rewards. Figuring out which actions contributed the most to reach the goal or maximum reward distinguishes luck from skill. Credit assignment is therefore a very useful metric for evaluating sparse-reward environments. It requires capturing long-term relationships between actions and rewards. Capturing long-term dependence is also the goal in sequence modeling.

On the sequence models front, there has been work on another class of sequence models - Structured State Space Sequence (S4). S4 models sequences with differential equations and is asymptotically faster than transformers, allowing for much larger sequences.

In this project, we study the impact of using S4 models in place of transformers in DT in various carefully selected different sets of environments. We find that the S4 sequence models perform better in all environments that we tested. They are also several times faster than DT. Additionally, since S4 models the sequences as a differential equations, they show a much more improvement in physical environments like cart pole which have a continuous state space.

2 Background and Relevant Literature

Structured State Space Models (S4)

S4 [3] models the sequence as a differential equation in a latent space and thus is able to capture long range dependencies. Unlike other RNN models where sequentially reading the data is the time bottleneck, the encoding process in S4 can be parallelized using Fourier Transforms. Also, unlike transformers, S4 does not need to keep all the history at inference. Thus S4 showcases better performance in both training and inference.

While we focus on the credit assignment metric which usually applies to environment with discrete tasks, Lu *et al.*[4] are interested in tasks in the control domain, which are non-discrete. S4 naturally lends itself to modeling the dynamics of these systems, the results of which can be seen in their work. We on the other hand try applying S4 models to discrete tasks to look for similar improvements with respect to the credit assignment metric.

*equal contribution

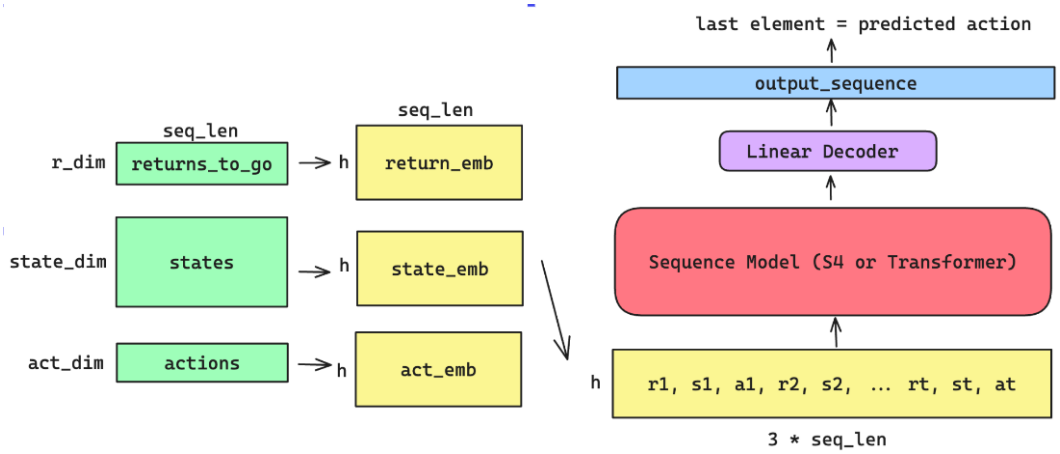


Figure 1: Decision Sequence models include passing the r, s, a for each step in the trajectory. The model tries to predict next action given the sequence of prior state, rewards, and action in the trajectory. We provide an abstraction between S4 and Transformers such that we can use them interchangeably.

Decision Transformers

Since RL can be framed as a sequence modeling task, we can use sequence models like transformers to solve it. As introduced by Chen *et al.*[2], DT can be used to solve offline RL tasks by passing in the trajectory-wise states, actions, and rewards as separate tokens to the model.

Credit Assignment Problem

Previous work like Alipov *et al.*[5] discusses the limitations of existing Hindsight Credit Assignment (HCA) algorithms in deep RL that lead to their poor performance or complete lack of training, then propose several theoretically-justified modifications to overcome them. Venuto *et al.*[6] and decision transformers are ways to do credit assignment better than HCA.

3 Methods

We have tried running the official decision transformers repository (github.com/kzl/decision-transformer) and some other reference implementations from HuggingFace. However, each of them have issues with package compatibility and more importantly they do not have support for easily replacing sequence models. Hence, we decided to implement our own decision transformer architecture that is generalizable to S4 as well as seen in Figure 1. We anticipated GPT training to be a bottleneck, but it did not turn out to be a big issue as we implemented our own small transformer instead of using the heavy GPT. We also implemented dataset generation classes for all the following environments. A lot of logging, training config etc. classes had to be implemented as well for ease of training and recording results.

4 Experiments

As we have mentioned in passing several times by now, we have a lot of factors to vary and we plan to experiment them. We have picked a variety of environments to benchmark performance. Each of the environments is carefully chosen to measure performance of algorithms from different aspects. In addition to that, we will vary the sequence model inside the decision transformer.

Another confusion is on how to view the trajectory as a sequence. The way the decision transformer paper does it is to send state, action and return as 3 separate tokens one by one. It works well, however we have a concern that the transformer needs to spend additional effort on learning this relationship between adjacent triplets of tokens. We want to see if sending all 3 concatenated into one token works well or not. The objective function as prescribed by DT is just the mean loss between the predicted

action tokens by the DT and the actions in our trajectory. The loss can be L2 for continuous actions and cross entropy for discrete.

Random Walk Environment

In this environment the goal is reaching the goal node in a weighted graph G starting from a node $s \sim D_0$. The reward is the negative weighted length of the path traversed, so essentially the agent needs to find the shortest path in this graph given some random walks. The state space would be all vertices in the graph and actions would be traversing any outgoing edge from the vertex the agent is currently at. The objective function is the cross entropy loss between the predicted actions by the DT and the actions in our trajectory.

The challenge in this environment is that agent needs to stitch paths that are traversed in dataset and find a path that is better than the shortest path in the dataset. In [2] author suggested that DT is able to achieve this.

The Umbrella Length

This task, which is a part of the Behavior Suite for RL (bsuite) [7], is also specifically for testing credit assignment. As described in Venuto *et al.*[6], The task involves a long sequential episode where only the first observation (a forecast of rain or shine) and action (whether to take an umbrella) matter, while the rest of the sequence contains random unrelated information with random rewards except the last state. The state conveys information about having an umbrella and the forecast. More specifically, it has 3 pieces of information: need_umbrella, have_umbrella, and time_to_live, and n distractors that are randomly sampled from a binomial distribution. The action space is whether to take an umbrella or not. A reward of +1 is given at the end of the episode if the agent chooses correctly whether to take the umbrella or not depending on the forecast, and -1 otherwise. The "length" part comes from the amount of distracting information in the sequence in the form of the chain length, which is the number of states until the final state. The objective function is the cross entropy loss between the predicted actions by the DT or S4 policy and the actions in our trajectory.

We ran experiments with chain lengths of 1, 2, 4, and 8, with 20 distractor states.

Door Key Environment

This is a minigrid environment where the agent has a 7x7 view of stuff in front of it. The agent has to find and pick up a key, open the door, and reach the goal after going through the door. The agent can move up, down, left, right, pick up the key and open the door. It gets a reward = 1 - number of steps taken / max steps possible.

Cartpole Environment

In this environment, a cart has a vertical pole hinged on it. The goal is to keep the pole from falling down. Actions are discrete - pushing the cart forward or backward. The reward is +1 for each time step that the pole does not fall. When the pole falls or we are above 500 steps, the episode ends. The observation space is continuous, with the cart position, velocity, pole angle, and pole angular velocity.

5 Comparison between S4 and Transformers

Our experiments showed that Decision S4 (our model with S4 instead of transformer) is more robust than Decision Transformers in terms of both performance and training time while having fewer parameters allowing the training process to be faster (Figure 3). Although we did not explore this in the project, another significant improvement achieved by using S4 is the sequential decoding capability that allows the time complexity to be $O(n)$ as opposed to $O(n^2)$ in transformers.

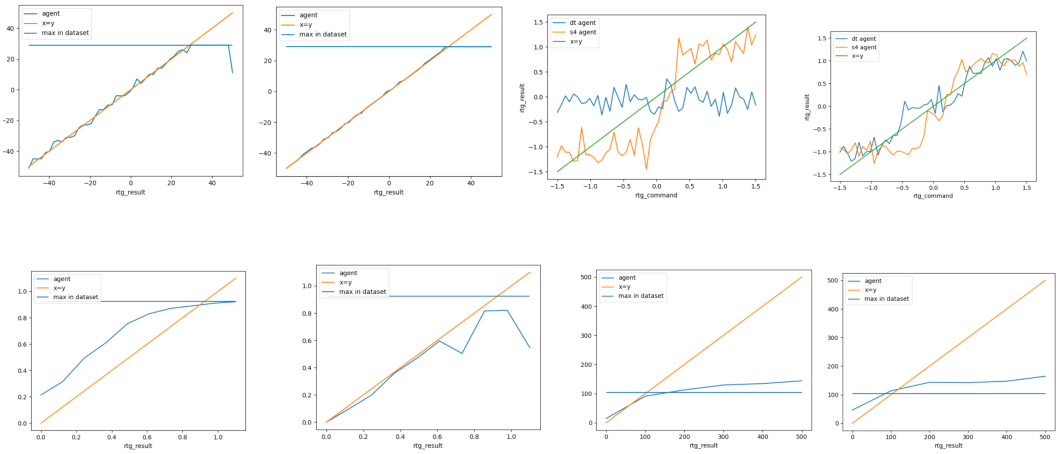
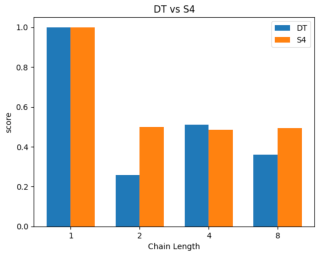


Figure 2: Comparison of performance in different environments. From upper left to upper right, we have random walk S4 and DT, umbrella length with chain length 8 and 4, and from bottom left to bottom right, we have doorkey S4 and DT, and cartpole S4 and DT.

Environment	Performance		Parameters		Training Time	
	S4	DT	S4	DT	S4	DT
CartPole	**129** steps without pole falling	107 steps without pole falling	**142k**	170k	**6 min**	30 min
DoorKey	uses **11%** of the max_steps	uses 18% of the max_steps	**600k**	1.1M	**7 min**	20 min
Umbrella Length	**0.4925** score on chain length 8	0.36 score on chain length 8	**600k**	1.1M	**2:45 min**	3:15 min
Graph Walk	Finds the shortest path	Finds the shortest path	**600k**	1.1M	**9 min**	23 min

(a) Summary of the results



(b) Comparison of performance in different chain lengths. Blue is DT and orange is S4. The score is the fraction of the 2000 iterations where the reward is greater than 0.5.

Figure 3: Summary and a plot of umbrella length results.

6 Implementation

We followed Chen *et al.*[2] in order to implement Decision Transformer but as their codebase was not flexible we implemented our version that is accessible at `https://github.com/Shayan-P/RL-credit-assignment-experiment`. Initially, we were planning to use minGPT [8], however later we decided to use min_decision_transformer notebook [9] as it allowed for more flexibility.

We implemented a similar interface for S4 and Decision Transformer that allows us to seamlessly switch between those. Additionally, we create an abstraction that allows the Sequence Model Policy to be composed with other Pytorch models (e.g. using CNN as a feature extractor) and easily apply the policy to any gym environment. Additionally, we also explored different batching techniques that improved the training process of S4.

References

[1] M. Janner, Q. Li, and S. Levine, *Offline reinforcement learning as one big sequence modeling problem*, 2021. arXiv: 2106.02039 [cs.LG].

[2] L. Chen, K. Lu, A. Rajeswaran, *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," *CoRR*, vol. abs/2106.01345, 2021. arXiv: 2106.01345. [Online]. Available: https://arxiv.org/abs/2106.01345.

[3] A. Gu, K. Goel, and C. Ré, *Efficiently modeling long sequences with structured state spaces*, 2022. arXiv: 2111.00396 [cs.LG].

[4] C. Lu, Y. Schroecker, A. Gu, *et al.*, *Structured state space models for in-context reinforcement learning*, 2023. arXiv: 2303.03982 [cs.LG].

[5] V. Alipov, R. Simmons-Edler, N. Putintsev, P. Kalinin, and D. Vetrov, *Towards practical credit assignment for deep reinforcement learning*, 2022. arXiv: 2106.04499 [cs.LG].

[6] D. Venuto, E. Lau, D. Precup, and O. Nachum, "Policy gradients incorporating the future," *CoRR*, vol. abs/2108.02096, 2021. arXiv: 2108.02096. [Online]. Available: https://arxiv.org/abs/2108.02096.

[7] I. Osband, Y. Doron, M. Hessel, *et al.*, "Behaviour suite for reinforcement learning," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rygf-kSYwH.

[8] A. karpathy, *Mingpt: A minimal pytorch re-implementation of the openai gpt training*, 2020. [Online]. Available: https://github.com/karpathy/minGPT.

[9] *Min_decision_transformer*, https://colab.research.google.com/github/nikhilbarhate99/min-decision-transformer/blob/master/min_decision_transformer.ipynb?fbclid=IwAR2wiYd9wFQJb6i77uIw-_XaEhSpo0bSRxz9YYwRSBwbAhVfPlNT79nde_U#scrollTo=9TpGEYTblzQc.