Decision Transformers for Robotic Control

Aidan Beery Nathan Funckes Ali Martz
Oregon State University
{beerya, funckesn, martzal}@oregonstate.edu

Abstract

Traditional methods in reinforcement learning iteratively update a policy function to optimize over non-continuous reward spaces. Recent work has demonstrated the possibility of reformulating these reward-based sequential decision problems as sequence learning problems. Simultaneously, the recent dominance of transformer models for sequence learning tasks has spurred research efforts to evaluate what domains can benefit from the powerful long-distance relationship learning possible with this architecture. The Decision Transformer seeks to unify these two principles with a generative trajectory modeling approach and a framework for large, interdependent sequence inputs. In this work, we extend the use of decision transformers to the robotics control domain. Specifically, we show the feasibility for decision transformer to learn the four tasks of the Fetch environment provided by gymnasium-robotics (slide, pick & place, reach, push) in which a 7-degrees-of-freedom manipulator is used to move a simulated object from one state to another.

1. Introduction

Traditional methods in reinforcement learning rely on learning a policy by taking a series of iterative steps toward the maximization of an often non-continuous reward function. Based on this principle of sequences of actions which each create optimal rewards, recent work has formulated sequential decision-making problems often solved by these reinforcement learning paradigms into a sequence learning task, and applied a rich body of established sequence learning techniques to the domain [6] [7]. Simultaneously, transformer models have proven to be highly effective models for learning semantically relevant representations of the interactions between features in a space, particularly in a variety of natural language processing tasks. A decision transformer seeks to unify these two principles by taking a generative trajectory modeling approach to deep reinforcement learning problems, and applying a popular framework for handling large, highly interdependent sequence inputs [3].

Prior work has demonstrated the efficacy of decision transformers across a variety of benchmark RL tasks, including Atari 2600 [1] and D4RL [2]. In this paper, we extend the use of decision transformers to the robotics control domain, specifically targeting the Fetch environment provided by `gymnasium-robotics` [3]. The Fetch environment proposes four tasks: Slide, Pick & Place, Reach, and Push. Each one involves the use of a 7-DoF robotic manipulator to successfully move an object from one state to another in a simulated environment. An example of the slide task is presented in Figure 1.

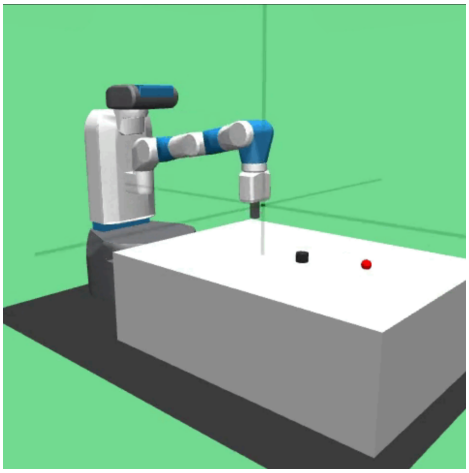


Figure 1. An example of the Fetch robotic manipulator, rendered in the MuJoCo physics simulator

We demonstrate the feasibility of applying decision transformers to robotic control tasks. We find that decision transformers are able to learn somewhat successful policies for 3 of the 4 Fetch tasks, even when trained on completely random data. When learning from expert data, we find that our model achieves a success rate greater than 70% for Reach, Push, and Pick & Place. Overall, we show promise in the use of sequence modeling techniques to learn behavior policies in continuous-valued environments using offline

[1] https://research.google/resources/datasets/dqn-replay/
[2] https://github.com/Farama-Foundation/D4RL
[3] https://robotics.farama.org/envs/fetch/

data without the need for conventional reinforcement learning algorithms.

2. Related Work

A fundamental challenge of many reinforcement learning problems is representing the action and observation spaces. Successful methods in deep reinforcement learning, such as Deep Q-Learning, are designed for discrete environments. However, robotics applications often involve continuous-valued action spaces. Previous work attempting to apply Deep Q-Learning to continuous environments discretize the action space which may lead to numerical instability in neural function approximators [8].

In response to the challenges of learning policies for continuous control tasks, Lillicrap et. al. present an extension to Deterministic Policy Gradients (DPG) using neural networks to learn a critic function $Q(s, a)$ and an actor function $\mu(s|\theta^{\mu})$ [8]. Deep Deterministic Policy Gradients (DDPGs) use neural function approximators to enable learning higher dimension Q functions which permits an agent to learn optimal actions in high dimensional or continuous spaces. Similarly to Deep Q-Learning, DDPG implements a target function for both μ and Q. It also applies a replay buffer to apply existing optimization techniques on function approximators without producing numerical instability when training.

During training, some noise term $\mathcal{N}$ is added to the actor policy μ which enables exploration in continuous environments. DDPG matches or exceeds the performance of regular DPG when evaluated against a series of control tasks. It also can learn an effective policy in motion control-based environments in significantly fewer steps than DPG.

DDPG is still limited by an environment's reward space despite its success in continuous action spaces with low and high dimensional observation spaces. Many reinforcement learning problems are best characterized by binary rewards where a given trajectory only receives a positive reward signal in the case where it reaches the goal state. However, for large state spaces, heuristic-driven exploration policies may never encounter a trajectory where it receives any reward information. Thus, DDPG and other off-policy reinforcement learning algorithms are unable to gain any understanding of an environment's dynamics.

Hindsight Experience Replay (HER) is a technique that augments off-policy learning algorithms by modifying the replay buffer such that, for a given trajectory $s_1, ..., s_T$, and some goal $g \notin \{s_1, ..., s_T\}$, it randomly sets $g = s_T$ for this trajectory before inserting it into the replay buffer [1]. This goal-modification principle operates on the assumption that the dynamics of the environment are invariant to the goal state. Therefore, providing a positive reward signal for some trajectories at some end-trajectory states s_T can condition our model on the properties of the transition function governing the environment and learn what state-action pair are associated with what end-trajectory states.

To evaluate the efficacy of HER in complex, continuous, multi-task environments, Andrychowicz et. al. propose the Fetch environment, a simulated 7-DoF robotic manipulator with a two-fingered end effector [10]. The Fetch robotic arm simulation is implemented in a MuJoCo physics simulator and consists of four tasks: Reach, Pick and Place, Push, and Slide (see figure XYZ). The observation space is defined by the velocities and torques at each joint in the manipulator as well as the global coordinates of the box, goal state, and end effector. Fetch's action space is defined by the desired global coordinate of the end effector at the next timestep and constrained by the physics simulator's dynamics. Fetch uses a binary, sparse reward where the return is -1 in all states except the goal state, where it is 0. For Slide, Pick and Place, and Push, this goal state is achieved by moving a box from a randomly generated initial coordinate to the goal location. Reach only needs the manipulator to move the box to the goal state.

For Push, Slide, and Pick and Place, Andrychowicz et. al. find that DDPG is unable to solve these tasks without HER [1]. Using HER and a sparse binary reward space, an agent can achieve a 100% success rate in any of these three environments. RL practitioners often design reward functions to guide an agent towards the desired goal without requiring it to reach the goal state before receiving a reward signal. A dense reward function for Fetch is presented as $r(s, a, g) = \lambda|g - s_{\text{object}}|^P - |g - s'_{\text{object}}|^P$ and encourages the agent to take actions which reduce the distance between the box and the goal state. However, this shaped reward is not sufficient to allow a DDPG agent to solve any of the Fetch tasks. Furthermore, even when incorporating HER into a DDPG agent with a dense continuous-valued reward, in no case is a success rate of greater than 20% achieved. By hand-crafting a shaped reward function, our learning algorithm optimizes for a new objective function which may or may not be sufficiently similar to our desired outcome. Using HER avoids the need for reward engineering in this environment by implicitly developing a curriculum to learn the dynamics of the environment.

Despite its success in the Fetch environment, researchers have identified that the technique can be numerically unstable and is highly sensitive to hyperparameters. Furthermore, in the case of stochastic environments, end-trajectory states reached due to random noise may be assigned as goal states and introduce bias to the learned policy. As an alternative to hindsight based methods, Ma et. al. propose an offline RL algorithm that uses a goal-conditioned state occupancy matching objective function to learn a policy. GoFAR [9] attempts to learn a policy with a state-occupancy distribution most similar to the distribution of states that satisfy the goal condition as generated by some expert agent which is able to "teleport" to the goal state. By reducing this

to a weighted regression problem, GoFAR is able to guarantee optimal goal-weighting distribution. When evaluated against Fetch, the authors find that GoFAR is able to achieve significantly greater average returns than DDPG + HER despite a sparse binary reward space. Furthermore, GoFAR performs slightly worse with HER than without, indicating that hindsight is not necessary for a state-occupancy matching based learning algorithm.

Continued efforts have been made to handle high-dimension continuous control problems in reinforcement learning without relying on hindsight methods. By treating Q function estimation as a representation learning problem instead of strictly as a reward optimization problem, Eysenbach et. al. the abundance of existing work in representation learning models to design effective policy learning agents [4]. Specifically, the authors investigate the use of contrasting learning as a mechanism for directly estimating a Q function to learn a goal-conditioned policy. When compared against TD3 + HER on Fetch Reach and Slide tasks, a contrastive-learning-based model matches or exceeds the performance of hindsight-based models, even without reward shaping.

Two trends in the broader machine learning research community seemed to coincide - the reformulation of RL as a modeling problem and by proxy the use of deep learning techniques for policy estimation, and the meterotic rise of the transformer architecture for sequence representation learning [7]. Trajectory Transformers were an early attempt at fusing these two trends by reformulating reinforcement learning as a sequence learning problem [6]. Episodes are represented as trajectories consisting of a state, action, and reward token at each timestep $\tau = (s_1, \mathbf{a}_1, r_1, s_2, \mathbf{a}_2, r_2, ..., s_T, \mathbf{a}_T, r_t)$. The goal is to learn a sequence of actions that leads to the goal condition. A decoder model architecture based on GPT is used for the transformer and actions are decoded using a beam search algorithm at inference time. Janner et. al. leverage existing datasets of offline reinforcement learning problems, namely the D4RL control benchmark tasks, and find that a GPT-style transformer is able to match or exceed current state-of-the-art offline RL algorithms on these tasks.

One of the primary challenges facing reinforcement learning for robotic control is the expense of gathering data. Despite the robustness of the aforementioned representation learning techniques, many still rely in part on expert inputs to build a dataset of trajectories to learn from [2]. Similarly to Janner et. al., the Decision Transformer (DT) uses a GPT-style transformer to learn optimal action sequences. Unlike the Trajectory Transformer, Decision Transformers predict action sequences autoregressively. Both Trajectory Transformer and Decision Transformer use action sequence generation methods which pick actions based on future potential reward, instead of reward at the next state. Instead of directly learning reward embeddings in each trajectory, it uses Return To Go (RTG) and associates early states with the cumulative discounted reward of all future states in that action sequence.

During training, a similar trajectory formulation as used in the Trajectory Transformer is utilized, tokenizing states, actions, and RTGs at each timestep. However, the autoregressive action generation of Decision Transformer only predicts actions $[a_1, ..., a_T$ instead of predicting the state and RTG as well. In the Atari 2600 benchmark using an image observation space, the Decision Transformer is able to match or exceed the current state of the art - conservative Q-learning - in 3 of the 4 games tested. In the D4RL control benchmark suite, Decision Transformer outperforms conventional reinforcement learning algorithms in most cases. Crucially, Decision Transformer also demonstrates the ability to learn reward-optimizing behavior even when only presented with randomly generated trajectories, enabling vastly superior scaling of trajectory datasets than what is currently available in offline RL datasets.

3. Methods

We train a Decision Transformer to learn a policy for four tasks from the Fetch environment - Reach, Pick & Place, Slide, and Push. To achieve this, we use the offline RL dataset for Fetch provided by Ma et. al. [9], consisting of 40,000 trajectories generated from a random action policy, as well as 4,000 from an expert policy.

Each trajectory represents a single episode in the environment with 50 steps per episode. At each timestep, the state is represented by the velocity and torque values at each joint of the arm as well as an indicator as to if the end effector is opened or closed. The Fetch environment is goal-aware, so the current location of the box and the coordinate of the goal state are provided along with the observation for Slide, Pick & Place, and Push. For Reach, the current global coordinates of the end effector replace the box coordinates. We append this goal information to the state vector for each observation. In total, our observation space contains 31 dimensions for all tasks besides Reach, where the observation vector is of length 16. The action space is represented as the global 3-dimensional coordinates of the manipulator at the next timestep.

Reward in the Fetch environment can be modeled as either a sparse binary reward or a continuous-valued dense reward. The sparse reward space assigns each action-state pair with a reward of -1, unless it is within an Euclidian distance of $\epsilon = 0.05$ of the goal coordinate, in which case the reward is 0. The shaped reward function for Fetch proposed in [1] measures the difference in distance between the box location and the goal coordinate in the current timestep in addition to the anticipated distance after the next action according to tunable hyperparameters. We use a simplified

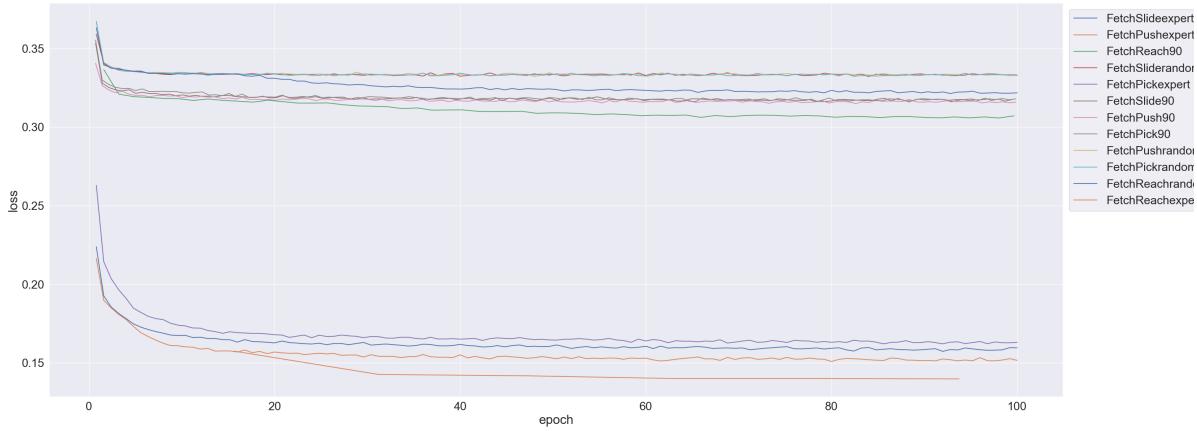


Figure 2. Training loss across instances of Decision Transformer.

version of this dense reward that takes the negative Euclidian distance between the box location and the goal coordinate at each timestep[4]. As recommended in [3], we calculate returns-to-go and assign future earned reward to each state-action pair.

We use the HuggingFace Decision Transformer implementation[5] based on the architecture proposed in [3]. The D4RL control environments benchmarked in the original Decision Transformer paper consist of trajectories with episode lengths of 1000, while our Fetch trajectories are only of length 50. We aim to reduce overparameterization and improve training stability via reducing the number of transformer layers from 3 to 2. We also increase the number of self-attention heads from 1 to 4, as Fetch's observation space is larger than that of the D4RL environments. Thus, there may be additional relationships between features that a single attention head cannot encode appropriately. Our model's loss is defined as the mean squared error of the predicted action from the target. We use an Adam optimizer with a learning rate of 1×10^{-4}.

We compare three different dataset preparation schemas for each of our tasks. Models are trained first on only random trajectories, as described in [3]. The collection of expert trajectories requires considerably more effort than random sampling of the action space, so the ability to learn an environment's dynamics from only random trajectories is of great interest for high-dimensionality control problems, like those frequently solved by conventional RL algorithms. We also evaluate our models when trained on a dataset with 90% random data and 10% expert trajectories, comparable to the dataset regimen used for GoFAR [9]. We then undersample our random subset to create a dataset of 36,000 random trajectories and 4,000 expert trajectories. Finally, we include models trained on exclusively expert data, assessing the ability for Decision Transformers to emulate behavioral cloning. Since behavioral cloning of expert policies in complex environments is already achievable using feed-forward neural networks, we anticipate that Decision Transformers should handle this task with ease despite having access to significantly fewer trajectories.

Each model is evaluated using the `gymnasium-robotics` Fetch implementation [6]. A trained Decision Transformer predicts the next action at each time step in the environment. Success is measured by percentage of trials which are able to successfully reach the goal state within a 0.05 margin of error. For each model, we test 10 different training depths, from 10 epochs to 100, executing 100 episodes per training depth and recording the number of successful trials as well as the average return.

4. Results

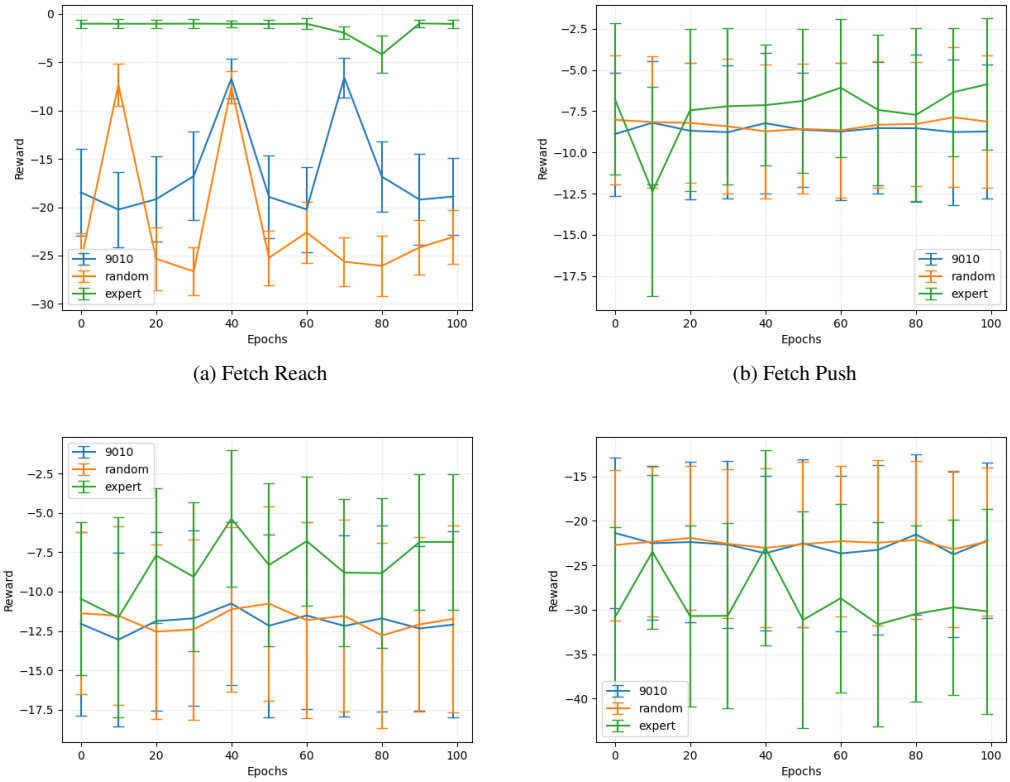
To evaluate how well decision transformers have learned the tasks in the Fetch environment, we report the loss of our models during training, the average cumulative rewards across 100 trials at different points during training, and the success rate of our models as a percentage of successful trials.

In Figure 2 we present a graph all our models' losses during training. We observe that each models' loss is quickly reduced after only a few epochs. After this initial rapid falloff the loss of each model shows little to no further improvement. This failure to reduce loss after a few early

[4]This is the dense reward function recommended in https://robotics.farama.org/envs/fetch/slide/#rewards

[5]https://huggingface.co/docs/transformers/main/model_doc/decision_transformer

[6]https://robotics.farama.org/envs/fetch/



(a) Fetch Reach

(b) Fetch Push

(c) Fetch Pick and Place

(d) Fetch Slide

Figure 3. Average Cumulative Rewards over 100 trials

epochs suggests that our loss may not be representative of our target function, or that our model may not be adequately learning from the target data.

We observe in Table 1 the average cumulative rewards for each of our dataset subsets. Using the average reward of the trajectories in our dataset, we compare the rewards from model-generated trajectories to assess our model's performance on Fetch tasks relative to the policies used to generate the dataset.

Following this idea we evaluate our models at every ten training epochs and calculate the average cumulative reward across 100 trials as seen in Figure 3. We find that for three of the four tasks, the models trained on the expert data have a greater cumulative reward than each of the other models for their respective tasks. This behavior likely follows from the higher average reward found in the expert training data as observed in Table 1. However, for most of the models, including the expert models, the average cumulative reward of the models seem to be consistently less than or equal to that of the data it was trained on.

In Table 2 we present the success rate of our models on each of the environments. We include the success rate for both a max length of 50 timesteps and 100 timesteps. These

Task	Random	90-10	Expert
Reach	-9.51	-8.76	-2.03
Push	-8.59	-8.09	-3.86
Pick and Place	-11.76	-11.35	-7.91
Slide	-22.69	-21.88	-20.37

Table 1. Average Cumulative Reward for each Data Distribution

values were calculated by running each model for 100 trials on the selected task. For the models trained on the expert data, we observe good performance on 3 of the 4 tasks with a success rate of 100% on reach, 82% on pick and place and 55% on push. The model trained for the slide task, on the other hand, had a success rate of only 4%. Additionally, the models trained on all random data and the ninety-ten split data all had low success rates when limited to 50 timesteps. However, when the number of steps is doubled, there is a noticeable increase in the success rate of the models on reach, push, and pick and place tasks. The model trained for pick and place on the ninety-ten split increased its success rate from 6% to 50% and the model trained on expert went from an already high success rate of 82% to a

Task	Random (50 steps)	90/10 (50 steps)	Expert (50 steps)	Random (100 steps)	90/10 (100 steps)	Expert (100 steps)
Reach	9%	9%	100%	100%	100%	100%
Push	14%	13%	55%	13%	24%	73%
Pick and Place	7%	6%	82%	7%	50%	96%
Slide	0%	1%	4%	2%	3%	4%

Table 2. Success rate over 100 trials, best epoch

success rate of 96%. The most significant of these increases is seen for the reach task where the models trained on random data and the ninety-ten split both increased their success rate from 9% to 100%. The results in Table 2 clearly show that a decision transformer trained on random data in the Fetch environment is able to only successfully complete three out of four of the tasks in some cases. Furthermore, a policy for the Reach task is learned which achieves a 100% success rate when given enough time.

5. Discussion

We demonstrate the feasibility of extending the Decision Transformer model to the Fetch continuous control tasks. Without the use of hindsight, a replay buffer, or target networks, transformer networks are able to learn decision-making policies in complex continuous-valued environments. However, our results clearly indicate that our model struggles to converge to an optimal policy in many cases.

As found in [1, 10], Reach appears to be the easiest of the Fetch tasks for a learning algorithm to solve. Despite this, our Decision Transformer only achieves a 9% success rate when using random trajectories and limiting to 50 epochs. When trained exclusively on expert data, however, our model achieves 100% success on Reach. Furthermore, when the maximum episode length is extended to 100 steps, models across data strategies achieve 100%.

Similarly, consistent with findings in prior work, we find Slide to be the most challenging task because our model fails to learn an effective behavior policy. Even when extending the maximum episode length to 100 steps and using exclusively expert trajectories, our model only achieves a 4% success rate in the best case. This large of a performance difference between Slide and other tasks, even on expert data, demonstrates the difficulty of learning from actions with long-distance effects on reward. Since the manipulator is not in contact with the box as the box approaches the goal state, the learning agent must learn what early action-state pair are associated with a goal state despite a reward that is decoupled from these earlier states and only sends a positive signal after the manipulator can no longer effect change on the outcome of the episode. However, transformers are known for their ability to model long-distance relationships between tokens in a sequence, and as such we would anticipate that our model is in fact best suited for a task such as Slide. Therefore, we believe our model's failure on this task to be an engineering failure, rather than a limitation of the modeling approach.

Except for in the case of Slide, we find that models trained on exclusively expert trajectories outperform their random counterparts across all tasks and episode lengths. This indicates that the Decision Transformer is in fact capable of behavior cloning. We also see in Figure 2 that models trained on expert data tend to achieve smaller mean squared errors with respect to action vectors than their random-model counterparts, seeming to indicate that Decision Transformers can more easily learn to emulate the state distribution of the action spaces of expert datasets. However, especially in the case of Push, we find that these models do not achieve 100% accuracy, and do not outperform other behavioral cloning algorithms. Given the computational complexity of transformer models, it would be ill advised to use such a model for behavioral cloning. However, an all-expert model provides a point of comparison to measure the performance impact of learning from random data.

When comparing the cumulative returns of our models in Figure 3 to those of our dataset in Table 1, we find that our models tend to perform similarly to the reward distribution of their training dataset. In the case of Reach and Pick & Place, the model trained on expert trajectories outperforms all other models considerably. Table 1 also indicates a similar trend with expert data having a considerably smaller mean cumulative return relative to other subsets. Our models seem to match the reward distribution of the source trajectories. However, these models are not learning some aspect of the environment or task sufficiently to actually exceed the average returns found in the source data.

5.1. Limitations & Future Work

With an maximum episode length of 50 steps, our random and 90-10 split models fail to learn consistently effective behavior policies. Given the robust properties of the Decision Transformer model presented in [3], we find this behavior to be unusual. However, when the episode length is extended to 100 steps, our random models achieve considerably more successful trials. We find that our Decision Transformer is learning the dynamics of the Fetch environ-

ment, and is learning to accomplish goals in said environment, even from random data. However, the learned policy is inefficient, and takes a long time to approach the goal state, despite the training data only consisting of 50 steps per episode.

We suspect the cause of this suboptimal learning behavior is twofold. First, we hypothesize that our loss function does not capture enough information about the environment and long-term rewards to effectively inform the optimizer about what decisions are and are not relevant to the model for learning our task. Naively measuring mean squared error between action vectors is appropriate for environments where an explicit goal state is not known, such as the D4RL benchmark tasks. However, since we are able to perceive achieved goal and desired goal with each observation, alternative loss functions should be explored. Contrastive learning can be used to align embeddings generated from transformer models [5]. Applying a contrastive objective function to our decision transformer could allow for the alignment of successful trajectories from the dataset. Another possible reformulation of the Decision Transformer loss function would be to incorporate state occupancy matching - measuring the KL divergence between the distribution of possible states and the distribution of goal-satisfying states in a given time step. However, this function would likely be significantly more expensive to compute than mean squared error over an elementwise difference of vectors, introducing additional issues of scalability.

Furthermore, we believe our choice of reward function for our model was inadequate for the task. As discussed in [1], shaped or dense rewards in environments where the desired behavior is typically represented with a binary reward risks introducing bias into the policy learned by the agent. In the case of negative Euclidian distance from box to goal coordinate, our model pursues actions which bring the box closer to the goal, but lacks the ability to quickly solve a task in the environment instead needing significantly longer episode lengths to demonstrate model performance. In the future, we would investigate alternative dense reward functions which also provide a slight incentive to approach the object with the gripper, allowing the model to better learn optimal actions during earlier steps in the episode. We believe that a reward function which would encourage these steps earlier in the episode would lead to a model which would be more likely to reach the goal state within 50 epochs.

Future experiments with Decision Transformers in the Fetch environment should investigate the use of sparse reward functions. The aforementioned modifications to the dense reward function may encourage optimal actions early on, however the hand-crafted reward function risks introducing bias into the learning algorithm. By using a sparse reward space, we allow the learning agent to optimize for the desired outcome as defined by the goal state, instead of optimizing for a function which we believe to be an appropriate heuristic of the goal. Training Decision Transformers on a sparse reward environment would enable more direct comparison to existing models as well, such as DDPG + HER [1], GoFAR [9], and Contrastive Learning [4].

We evaluate the Decision Transformer as a single-task model, in which we take a dataset of pre-computed trajectories for a given task and train the model exclusively on data for the task for which we evaluate it against. However, much of the robustness of transformer models has been derived from their ability to fine-tune on downstream tasks after being pretrained on large amounts of unstructured data. Future experiments should begin to assess the feasibility of transferring this pre-training and fine-tuning paradigm to reinforcement learning domains. Furthermore, it is yet to be seen if a Decision Transformer can function as a multi-task learner, transferring knowledge about the dynamics of the environment learned from one task's trajectories to previously unseen tasks in that environment.

6. Conclusion

We extend the Decision Transformer model to a series of robotics control problems. We find that Decision Transformers are able to effectively clone expert policies in an offline context. Furthermore, we demonstrate that our model begins to learn successful behavioral when trained on trajectories generated from a random policy. This model formulation shows promise at being able to generalize to a variety of continuous control tasks, allowing environments with continuous-valued action spaces to be represented without needing discretization or hindsight. We identify two primary limitations of our approach and suggest future explorations into alternative loss functions which are better able to capture the desired properties in a trajectory sequence representation learner.

Our code can be found at https://github.com/Aidan-B1409/AI535_DecisionTransformer

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. (arXiv:1707.01495), Feb 2018. arXiv:1707.01495 [cs]. 2, 3, 6, 7

[2] Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, and Sergey Levine. Actionable models: Unsupervised offline reinforcement learning of robotic skills. (arXiv:2104.07749), Jun 2021. arXiv:2104.07749 [cs]. 3

[3] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind

Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021. 1, 4, 6

[4] Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. 3, 7

[5] Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, and Daniel P. W. Ellis. Mulan: A joint embedding of music audio and natural language, 2022. 7

[6] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem, 2021. 1, 3

[7] Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A survey on transformers in reinforcement learning, 2023. 1, 3

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. (arXiv:1509.02971), Jul 2019. arXiv:1509.02971 [cs, stat]. 2

[9] Yecheng Jason Ma, Jason Yan, Dinesh Jayaraman, and Osbert Bastani. How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression. (arXiv:2206.03023), Nov 2022. arXiv:2206.03023 [cs]. 2, 3, 4, 7

[10] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. (arXiv:1802.09464), Mar 2018. arXiv:1802.09464 [cs]. 2, 6