Domain Randomization with Offline Data Using Decision Transformers

Romil Sonigra
Dept. of Electrical & Computer Engineering
Texas A&M University
College Station, TX 77843
romils@tamu.edu

Durward Cator
Dept. of Electrical & Computer Engineering
Texas A&M University
College Station, TX 77843
tac464@tamu.edu

Moeez Akmal
Dept. of Electrical & Computer Engineering
Texas A&M University
College Station, TX 77843
moeez.akmal@tamu.edu

Debajoy Mukherjee
Dept. of Computer Science and Engineering
Texas A&M University
College Station, TX 77843
debajoym98@tamu.edu

Abstract

The application of Reinforcement Learning (RL) to Real World scenarios is often times challenging due to the complexity and partial observability of the environments. Simulators for training RL algorithms can aid in this challenge, but can possess difficulties of their own in sim-to-real transfer such as differences in physics, appearance, or noise. This creates a need for robustness in RL training to ensure differences in application do not cause agents to fail. Domain randomization (DR) is a popular technique to address this challenge by training agents on a diverse set of simulated environments with randomized properties. The idea behind DR is that if the agent can handle enough variation in simulation, it can also generalize to the real world without requiring any fine-tuning or adaptation.
Another complication with Real World applications of RL is the difficulty in obtaining training data. By the time a strong policy has been learned from standard online algorithms, the agent collecting the data may have been damaged or destroyed due to suboptimal behavior. Thus RL algorithms that can train with offline data pose attractive learning techniques. However, collection of offline data is similarly challenging, so algorithms which can contextualize well from small or poor data sets are desired. Decision Transformers present a promising approach to these challenges due to their ability to contextualize well from small data. These architectures transition from the classical Markov Decision Process (MDP) formulation of RL to a sequence modeling formulation and can therefore adapt to small or poor trajectory data sets. In this paper, we present the robust learning abilities of decision transformers trained on offline domain randomized trajectories.

Keywords: Domain Randomization, Decision Transformer, Offline Learning

1 Introduction

Reinforcement Learning (RL) is a type of Machine Learning which encompasses the field of stimulation based learning. Here an agent (the machine) is placed inside an unknown environment and given a certain objective that it needs to satisfy. It learns the appropriate sequence of actions that would help it achieve its objective by receiving stimulus from the environment after it takes some actions. Thus, RL can be defined as a framework for learning optimal policies from trial-and-error

Preprint. Under review.

interactions with an environment.
In today's modern lifestyle, RL has seen several software based applications such as in recommendation systems, etc. However, in general, RL algorithms often suffer from high sample complexity and poor generalization to new tasks or domains. To address these challenges, we need to incorporate robustness while training the agent so that it doesn't overfit to the underlying parameters of the environment but instead is able to adapt to minor changes in the environment parameters. Sim-to-real transfer and meta learning are promising approaches to solve this problem.
Sim-to-real transfer aims to train agents in simulation and deploy them in the real world without any adaptation. Sim-to-real transfer can reduce the cost and risk of data collection on real robots, but it requires overcoming the reality gap between simulation and reality. This gap can arise due to differences in physics, appearance, or noise. A similar problem can occur when transferring agents from one simulation to another with different parameters. This can be seen as a sim-to-sim transfer problem with a diverse parameter set, which is the problem our project will revolve around.
Generalization is particularly interesting because it enables agents to learn from limited or imperfect data and apply knowledge to new situations. Generalization is essential for solving real-world problems that are dynamic, uncertain, and diverse. Generalization also reflects the intelligence and adaptability of the agents and their ability to cope with complexity and novelty.
Broad ML research in the last few years has demonstrated the capability of transformers to model large-scale high-dimensional distributions of semantic concepts, enabling effective zero-shot generalization in language and out-of-distribution image generation. These successes lead to an investigation into their potential in sequential decision-making problems that are formulated as RL. However, instead of incorporating transformers as a component within conventional RL algorithms, this project aims to apply them for generative trajectory modeling as initially proposed by Chen et al. [1], which involves modeling the joint distribution of states, actions, and rewards, as a substitute for traditional RL algorithms. Here, we train the transformers using a rewards-to-go approach to incorporate a sequence modeling structure in place of the traditional Q-function estimation approach. The benefit of using this approach is that we avoid the deadly triad of RL since we no longer need bootstrapping for reward assignment. It also avoids the inherent weakness of using a discounted cumulative reward model which can substantially devalue a future reward especially when combined with bootstrapping.
Now, for incorporating robustness into the policy learnt by this transformer, we use a Domain Randomized offline data-set for training the transformer. The methodology for generating these trajectories, the characteristics of these trajectories and the policy used to generate these trajectories is mentioned in Section 4. We've compared our model against a transformer model trained on D4RL Hopper-medium v2 and Hopper medium-expert data-set to show the robustness provided by Domain Randomization (which inherently acts as a Multi-task learning problem) in Section 5.3. Also, to explicitly show the power of transformers, we've compared the performance of the transformer agent against the performance of an agent trained using Implicit Q-Learning (IQL) algorithm in Section 5.5.

2 Related work

One of the main challenges of sim-to-real transfer is to account for the discrepancies between simulation and reality, such as differences in physics, appearance, or noise. Domain randomization (DR) is a popular technique to address this challenge by training agents on a diverse set of simulated environments with randomized properties [2]. The idea behind DR is that if the agent can handle enough variation in simulation, it can also generalize to the real world without requiring any fine-tuning or adaptation. DR can also be trained with purely offline data [3] excluding the need for online RL algorithms and simulators. DR has been shown to be effective for various tasks in robotics and autonomous driving, such as object detection and pose estimation [2], robotic grasping [4], and locomotion [5]. Similarly, domain randomization can also be applied to sim-to-sim transfer problems, where the agent needs to adapt to different simulations with varying parameters.
In order to be robust against real world variations from training data, RL models must learn to contextualize from the data. The inclusion of attention masks in transformer architectures [6] allows for machine learning techniques to do just that. To transition from the classical Markov Decision Process (MDP) formulation of RL to this transformer architecture, the MDP can be transformed into a sequence modeling formulation [1]. Further robustness can be achieved with the inclusion of online fine turning [7] to tune the offline trained model to better fit the new task at hand.

3 Problem Formulation

Problem Statement: RL agents perform well in an environment with static parameters. However the real world environment for the same task likely has different underlying parameters that may vary. Traditional RL agents are unable to perform well when such parameter changes occur. Thus, the idea here is to allow for robustness in the policy learnt by the RL agent to allow them to perform well in similar real world scenarios.
We consider learning over the set of episodic MDPs $\mathcal{U}$ with distribution ρ given by $(\mathcal{U}, \rho)$. Each MDP $\mathcal{M} \in \mathcal{U}$ is represented by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}_{\mathcal{M}}, \mathcal{R}, T, \mu_0)$. The tuple consists of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition dynamics $\mathcal{P}_{\mathcal{M}}(s'|s, a)$, a reward function $r = \mathcal{R}(s, a)$, episode length T, and initial state distribution $s_0 \sim \mu_0(\cdot)$. Offline trajectories of these MDPs are generated where $\mathcal{M} \in \mathcal{U}$ is randomly sampled according to the distribution ρ at the start of each episode. Given this formulation, our objective is to find the optimal policy π^* in the (possible history dependent) policy space Π where π^* is given by:

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{\mathcal{M} \sim \rho, s \sim \mu_0} \left[\sum_{t=1}^{T} r(s_t, a_t) | s_0 = s \right] \tag{1}$$

4 Research Approach and Methodology

Data-set	Total timesteps	Avg Epi len	Avg Cumu Reward	Max reward	Min Reward	Reward std
Medium DR	5,000,000	109	223.41	243.16	170.93	7.59
Expert DR	2,000,000	328	1034.79	1709.15	209.19	366.60
D4RLdataset	999906	457	1422.06	3222.36	315.87	378.95

Table 1: Different Data-sets used for performance comparison

4.1 Transformer Architecture

The Decision Transformer model is a mini GPT based architecture that can learn from variable length trajectories using auto regressive sequence modeling. It uses blocks made up of feed-forward layers and causal self-attention layer which avoids us from including future information while training. The main ideology is to send the states and actions and returns to go which are nothing but summation of rewards through a fixed length horizon, we predict the action embedding. This is used to then calculate the loss function which is mean squared error if actions are real valued or binary cross entropy if its discrete. This is how the transformer learns effectively to stick optimal zones in sub-optimal trajectory to do in-context learning and hence perform way better than what it fed on.
We have trained a feed-forward based MiniGPT architecture for transformers from block length varying in multiples of 3 like 3,6,9. We have also varied the context length and the number of heads. The context length is the no of (s,a,r,s') samples we intend to attend to before making a prediction. In each bock of the transformers we have couple of feedforward layers in each block and we have used GELU non linearity. We have used embedding size of 128 and batch size of 64 to train the architecture.

4.2 Transformer Training

We begin by shuffling the training dataset to minimize dependency between sampled trajectories. We then set the transformer optimizer using AdamW with learning rate $1e-4$ and weight decay $1e-4$ and begin looping over training steps. Every 100 timesteps, we evaluate the performance of the transformer on 5 RandomHopper environments set with in-distribution parameters shown in Table 2. If the mean of this evaluation is greater than previous evaluations, we save the model weights for testing. This process is summarized in Algorithm 1.

Algorithm 1 Transformer Training

Require: Offline Dataset; Optimizer hyperparameters: lr, wt_decay; T > 0; Transformer model
Ensure: Trained Transformer
  Shuffle Dataset
  optimizer = AdamW(model.parameters(), lr, wt_decay)
  eval_envs = [env.sample(seed1),...,env.sample(seed5)]
  for $t = 0, 1, ..., T$ **do**
    pred_action = model.forward(Dataset(t))
    loss = MSE(pred_action, target_action)
    optimizer.step()
    value = evaluate(model, eval_envs).mean()
    if value > max_value **then**
      torch.save(model.state_dict())
      max_value = value
    end if
  end for

4.3 Testing simulators

We evaluated the performance of each model on an environment sample taken from 5 different distributions. This was done to check the range of robustness that the models can provide.
To perform randomization in the underlying environment, we varied the mass of the different components of the Hopper's anatomy.

Mass dis	Torso Mass	Thigh Mass	Leg Mass	Foot Mass
in	[0.9, 1.1]	[1.9, 2.1]	[2.9, 3.1]	[3.9, 4.1]
slightly out	[0.7,0.9]	[1.7,1.9]	[3.1,3.3]	[4.1,4.3]
inverted mass	[3.9,4.1]	[2.9,3.1]	[1.9,2.1]	[0.9,1.1]
extremities	[3.9,4.1]	[0.9,1.1]	[0.9,1.1]	[3.9,4.1]
Scaled mass	[0.9X, 1.1X]	[1.9X, 2.1X]	[2.9X, 3.1X]	[3.9X, 4.1X]

Table 2: Different Mass Distributions used

The mass distribution chosen is uniformly distributed over the respective bounds for the respective distribution names, where the tuples represents the [lower bound, upper bound] for respective anatomical parts of the Hopper i.e. the torso, the thigh, the leg and the foot.

5 Results

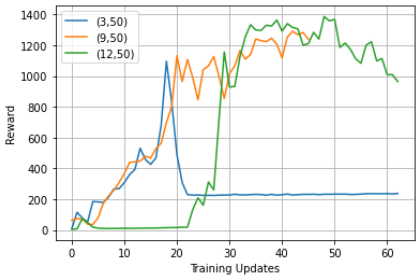
5.1 Using Medium Data: Comparing different Transformer architectures

		Context Length	
		30	50
Number of Blocks	3	470	1018
	9	253	1293
	12	217	1327

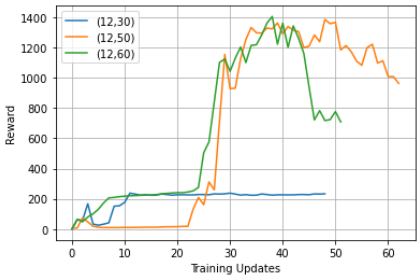
Table 3: Transformer Architecture Performance Evaluation with medium data set. Test evaluation score is given for each tuple (# blocks, context length).

As is clearly visible from Table 3, increasing context length improves the overall reward that can be generated by the agent. Increases in the number of blocks as shown in Fig 1a, on the other hand, creates more interesting behavior. For a context length of 30, increases in block number causes the transformer to more rapidly converge to the medium data mean. Whereas a context length of 50 allows increases in block number to improve the performance.

The impact of context length is dominant as is visible from Fig 1b. This is due to the fact that larger context length allows the transformer to better learn the underlying MDP sequence information. It



(a) Comparison of performance across different transformer block lengths



(b) Comparison of performance across different context lengths

Figure 1: Performance comparison for different hyperparameters

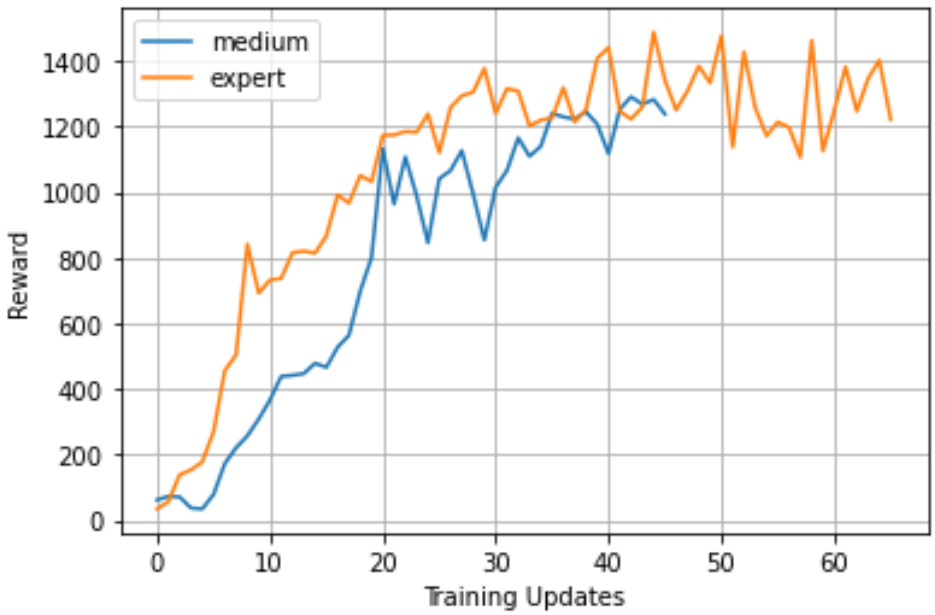


Figure 2: Learning rate comparison between a poor dataset and expert dataset on (9,50) transformer

also slows the transformer's rate of convergence allowing the model to learn good trajectories before converging on the dataset. This allows greater block numbers to improve contextualization and learn higher scoring policies.

5.2 Comparison of Transformer architecture performance based on underlying dataset

In this section we discuss the impact of the offline trajectories on the performance of the transformer. As is visible from Fig. 2, the transformer model trained with trajectories generated from an expert underlying dataset achieves a higher cumulative reward and converges faster. However, the model trained on the medium dataset is able to generalize policies better, whereas the expert data fails to generalize. This failure is because optimizing on the difference in actions gives it enough cumulative reward leading to overfitting. The change in the dynamic parameters with poor trajectories and a very high returns to go increased the in-context learning power of the transformer making it a more generalizable policy. This can be thought of a exploration and exploitation trade off, doing well on the expert trajectories is a greedy choice of rote learning while exploring better actions to do well on the medium data helps to overall achieve robustness.

5.3 Comparison between D4RL trained and Domain Randomized Trained Transformer

Here we verify the Robustness of the Transformer architecture and the benefits of Domain Randomization towards the goal of robustness.

distribution	D4RL-med	(9,50)-med	(12,50)-med	(12,60)-med	SAC	IQL
in	504.7±74.04	1271.86±54.50	1317.28±53.18	**1376.53±43.29**	1242.79	676.73±126.24
out of	474.16±133.94	**1206.76±164.48**	924.54±126.87	1091.04±154.40	212.71	501.70±152.56
inverted mass	591.74±40.17	475.82±2.89	**872.48±5.77**	689.72±8.39	571.54	100.30±54.87
extremities	580.92±48.15	568±27.50	599.82±98.78	**668.17±25.15**	535.91	272.22±46.63
Scaled mass	517.62±39.68	**1344.34±31.86**	906.72±222.51	825.98±41.01	1323.74	390.52±91.46

Table 4: Comparison of Domain Randomized training and D4RL training under different random mass distribution environments

In the above table, all the bold values represent the max reward for that mass distribution.

5.4 In distribution

As seen in Table 4 the transformer architectures trained on a uniform mass distribution do very well in that environment. This shows the basic robustness ability of Domain Randomized data and of transformers as they are able to learn a good policy despite poor underlying trajectories. Also, the transformer architectures are more robust to the in-distribution samples as compared to IQL.

5.4.1 Out of distribution

- **Slightly Out of Distribution Mass**: This represents minor deviation from the trained distribution. This allows for us to check the robustness of the transformer model in scenarios which may vary more than what was modeled. Since, all the transformer models do well in the slightly out of distribution case, hence we can be sure that transformers are robust to minor changes in the environment parameters and can continue to perform well making them viable and important candidates for sim-to-real transfer.
- **Inverted Mass distribution**: This makes the Hopper have an inverse mass distribution making the torso the heaviest and the foot the lightest. This makes the distribution move further away from the initial distribution thereby making it difficult for the transformer to perform robustly for this distribution.
- **Heavier Extremities Distribution Mass**: Here, we made the torso and the foot much heavier compared to the thigh and the leg. This was another check for how the actions of the trained transformer model would be affected when the masses are substantially changed. Here, as expected the performance is poor since this distribution is far from the underlying training distribution and applying excess torque breaks the angle requirements of the Hopper environment.
- **Scaled Mass distribution**: Here, we scaled the mass of each component by a factor X (where $1 \leq X \leq 2$ due to boundary conditions in Hopper environment). This was done to ensure the ratio of the masses remained same however the overall mass would actually change thereby requiring a significantly different amount of torque to be applied on the hinge joints. This is another check for whether the transformer is able to estimate this change and adapt accordingly. As Table 4 shows, the transformer in general adapts okayishly to this scaling except for the (9,50)-med model which is able to adapt extremely well.

5.5 Comparison between Transformer, IQL and SAC based online trained policy

Here we compare the performance of the transformer architectures we have used above against the state of the art offline training algorithm: Implicit Q-Learning as shown in Fig 3. Implicit Q-learning is the state of the art offline Q-learning policy, hence we compare our model's performance against it. Here, IQL was trained on the same medium data-set that we trained our transformer models on. As is visible from Table 4 the transformer models outperform the IQL model thereby indicating stronger performance of transformers among the offline robustness algorithms.
Also, we compare the Transformer model with the Soft Actor-Critic based online model. This model

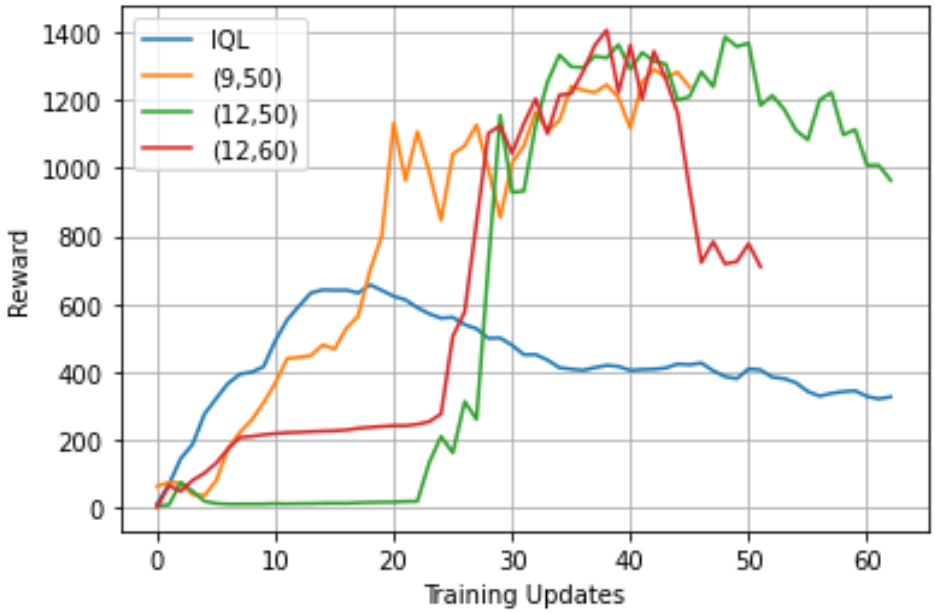


Figure 3: Learning rate comparison between transformer models and IQL

is used for generating the expert trajectories. Again from Table 4, it is clear that despite being trained in an offline setting the transformer models are able to outperform or perform comparably to the SAC based agent.

6 Conclusions and Future Work

Our main contribution has been applying Transformers with Domain Randomization and analysing their robustness. They have been very robust in the in-distribution and slightly out of distribution case making them a viable candidate for real world applications. The problems that we faced while working were that even though the transformers were excellent in-context learners, it was not able to work across crooked domains which needed completely different physics. In fact, none of the domain randomized models could do well in the inverted masses domain, which is mainly due to the physics of the system being completely inverted from the in-distribution training. This is why we need online fine-tuning and which is a future work that we plan to do. A second direction for future work would be using a larger transformer architecture such as adaptive GPT-4, and then easily generalize the model. Additionally, we can add more environments like half-cheetah and other domains so we could do meta learning and not be restricted with multitask learning.

7 Existing Implementation

These are some of the github links for the current implementations.

1. https://github.com/eric-mitchell/macaw
2. https://github.com/kzl/decision-transformer
3. https://github.com/facebookresearch/online-dt
4. https://github.com/tinkoff-ai/CORL
5. https://github.com/moeezakmal/random-envs-DR

References

[1] 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.

[2] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[3] Gabriele Tiboni, Karol Arndt, and Ville Kyrki. Dropo: Sim-to-real transfer with offline domain randomization, 2023.

[4] Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks, 2019.

[5] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots, 2018.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[7] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer, 2022.