Sequence Modeling for Objective Robustness in Reinforcement Learning

Elian Belot
McGill University
elian.belot@mail.mcgill.ca

Abstract

Recent work has shown that reinforcement learning (RL) can be solved by language models when formulated as a sequence modeling problem (Janner et al., 2021; Chen et al., 2021). This enables the use of transformer models, where capacity for few-shot learning and out-of-distribution generalization can be transferred in the context of RL. We conduct an empirical study of objective robustness failures in sequence modeling agents, a type of failure that occurs when the agent pursues an incorrect objective while retaining its capabilities out-of-distribution. We hypothesize that the generalization capacities of transformers can mitigate these types of failure while performing comparably to state-of-the-art RL algorithms.

1 Introduction

We study the effectiveness of sequence modeling agents in mitigating objective robustness failures in reinforcement learning agents. Specifically, we investigate whether decision transformers can outperform traditional reinforcement learning algorithms that fail on environments designed to induce objective robustness failures.

We hypothesize that the few-shot learning capacities of transformers (Brown et al., 2020) as well as their capacity for out-of-distribution (OOD) generalization (Hendrycks et al., 2020) allows for effective mitigation of objective robustness failures in adversarial variants of training environments.

To test this hypothesis, we conduct a series of experiments aimed at comparing sequence modeling agents with traditional online RL algorithms as they are tested for objective robustness.

2 Related work

2.1 Objective robustness

The concept of objective robustness has its root in AI safety research and stems from the work of Hubinger et al. (2019) on learned optimization. It studies the implications of machine learning models solving tasks by performing search according to an objective function. With sufficiently advanced machine learning models, it postulates that misalignment between training loss and learned objective function could lead to significant risks and undefined behaviors.

The term *objective robustness* was subsequently introduced to differentiate it from *capability robustness*, where trained models fail to generalize capably in out-of-distribution environments. Objective robustness is a subset of the more general problem of out-of-distribution (OOD) robustness, which is concerned with models performing well on test data sampled from a distribution different than the one used in training. Objective robustness failures arise when an agent performs capably in a new distribution, yet optimizes for a different objective.

For example, consider an environment where an is agent trained to find keys in order to unlock doors, which leads to a reward. When placed in a similar environment but with more keys than doors, the agent might attempt to collect all the keys and disregard the doors: it optimizes for number of keys collected, an incorrect proxy of the real reward function. The agent is still capable of collecting keys and opening doors (capability robustness), but fails to pursue the correct objective.

This concept was empirically observed in the context of reinforcement learning by Koch et al. (2021), in which experiments were conducted on variants of training environments designed to induce objective robustness failures. These environments induce selective randomness in elements susceptible to be learned as proxies of the reward function by an agent and will form the basis of the experiments conducted in this paper.

2.2 Sequence modeling

Reinforcement learning tasks are typically contextualized in a Markov decision process (MDP) described a $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ tuple representing states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transitions $p(s'|s,a) \in \mathcal{P}$ and rewards given by the function $r = \mathcal{R}(s,a)$. A trajectory represents a sequence of states, actions and rewards over a fixed number of timesteps t from 0 to T, the final timestep, and can be represented as $\mathcal{T} = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_T, r_T)$.

Sequence modeling is the task of generating a sequence of values by analyzing a series of input values and has greatly benefitted from the introduction of deep learning architectures such as LSTM models (Hochreiter and Schmidhuber, 1997) or transformers (Vaswani et al., 2017).

Reinforcement learning as a sequence modeling problem based on trajectories was first introduced by Janner et al. (2021) through a *Trajectory transformer*, in an attempt to apply the transformer architecture to achieve long-term planning without traditional single-step policy-based methods. This approach leverages offline reinforcement learning, an approach concerned with training reinforcement learning agents from offline data, rather than through interaction with the environment (online).

A number of methods have been devised for effective use of offline data and outperforming online models without the ability to explore or interact (Prudencio et al., 2022). Subsequent improvements to the use of transformers for sequence modeling are presented by Chen et al. (2021) with the Decision transformer architecture, which forms the basis for several experiments conducted in this paper.

The Decision transformer learns to predict actions sequentially based on a sequence of states, actions and reward represented through modality-specific embeddings combined with positional timestep embeddings to effectively situate the model in time. Action tokens are predicted autoregressively using a causal self-attention mask.

3 Method

We study the comparative performance of sequence modeling and traditional online RL algorithms when evaluated in environments designed to induce objective robustness failures. To do so, we first train two models on the same set of environments and evaluate them on adversarial variants.

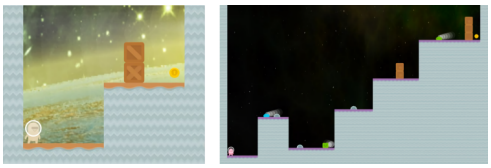
3.1 Environments

We use a modified subset of the Procgen environments (Cobbe et al., 2019) to train and evaluate our agents. Procgen consists of 16 procedurally generated game-like environments, and can be used to benchmark sample efficiency and generalization in reinforcement learning agents.

We use variants of three games from that suite from Koch et al. (2021) CoinRun, Maze, and Heist. After the initial phase of training and evaluation in these environments, we evaluate both agents on adversarial variants designed to induce objective robustness failures. The details of the environments used and their variants are described below.

CoinRun

Figure 1: The procedurally generated CoinRun environment on an easy level (left) and hard level (right).

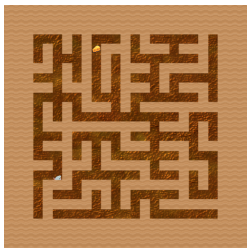


CoinRun is a procedurally generated platformer environment. The goal is to collect a coin at the right side of the level, while avoiding saw obstacles, enemies, and chasms. The player spawns at the left side of the level.

The adversarial variant places the coin randomly on the ground rather than on the far side of the level.

Maze

Figure 2: The Maze environment.

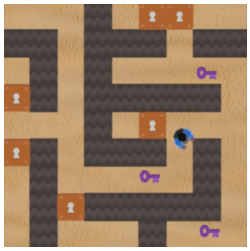


The player, a mouse, must navigate a maze to find a unique piece of cheese and earn a reward. Mazes are generated procedurally and range in size from 3x3 to 25x25. The player may move up, down, left or right to navigate the maze.

The adversarial variant places two objects in the maze: a red gem, and a yellow star. The objective can be configured to either be the gem, or the star.

Heist

Figure 3: The modified Heist environment.



In this modified version of the Heist environment, the player must collect keys to open chests, and is only rewarded for opening chests. There are twice as many chests as there are keys. In the adversarial variant, the environment is the same, but there is now twice as many keys as there are chests.

3.2 Proximal Policy Optimization (PPO)

For our online RL agent, we use an Actor-Critic model implementing Proximal Policy Evaluation (PPO) (Schulman et al., 2017). The agent is implemented in Pytorch using code from Lee [1]. The hyperparameters used are reported in Appendix A. The model is trained for 40M timesteps on 20k procedurally generated levels.

After training, we run the model on 1M timesteps on 200 levels while generating a set of trajectories that will serve as a basis for training the decision transformer. The dataset consists of a set of trajectories of the form $\mathcal{T} = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_T, r_T)$. The states $s \in \mathcal{S}$ represent the current frame in RGB colors with dimension $64 \times 64 \times 3$. Both the action space and reward space are discrete integers.

3.3 Decision Transformer

We train a decision transformer model using the codebase associated with the original paper by Chen et al. (2021)[2]. The model we use is based on the GPT model (Radford et al., 2019), with positional embeddings modified to represent timesteps. The model is trained on the set of trajectories described above. Hyperparameters are found in the Appendix. Following the original paper, we only predict actions rather than states or rewards. During preprocessing of trajectories, we replace every reward r_t by the corresponding returns-to-go $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$, which allows us to generate actions based on future desired returns, rather than past rewards.

[1] https://github.com/joonleesky/train-procgen-pytorch
[2] https://github.com/kzl/decision-transformer

4 Experiments

[...]

5 Discussion

[...]

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. 2021. Decision transformer: Reinforcement learning via sequence modeling.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. 2019. Leveraging procedural generation to benchmark reinforcement learning.

Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. 2020. Pretrained transformers improve out-of-distribution robustness.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. 2019. Risks from learned optimization in advanced machine learning systems.

Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline reinforcement learning as one big sequence modeling problem.

Jack Koch, Lauro Langosco, Jacob Pfau, James Le, and Lee Sharkey. 2021. Objective robustness in deep reinforcement learning.

Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. 2022. A survey on offline reinforcement learning: Taxonomy, review, and open problems.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.

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

A Hyperparameters

[...]