Decision making as language generation

Roland Memisevic, Sunny Panchal & Mingu Lee
Qualcomm AI Research[*]
{rmemisev,sunnpanc,mingul@qti.qualcomm.com}

Abstract

Decision transformers are a recently proposed approach to offline reinforcement learning that leverages transformer-based auto-regressive sequence models. We discuss challenges associated with fine-tuning a given, pre-trained language model on a decision making task. We propose solutions to these challenges and study their viability on a shortest path problem. We also show how given language model allows us to bring to bear data-centric approaches to improving the model and how it opens up the possibility to treat the decision transformer objective as one task alongside others to perform transfer learning.

1 Introduction

Decision transformers [1, 4] define the policy in an offline reinforcement learning task as a transformer-based, auto-regressive sequence model. They are trained supervised on sequences of triplets of actions, states and returns from roll-outs of an expert or other (including random) policy. Prompting with a high return at test-time encourages a decision transformer to follow action trajectories associated with high rewards and thereby to perform well.

Although decision transformers are a highly successful new approach to offline RL (cited several hundred times within few months of the publication of the basic approach by [1] and [4]), applications of the model come with a significant shortcoming: the decision transformer is typically conceived of as an RL-specific architecture, with task-specific action-, state- and reward-heads, and is, as common in RL, trained on a narrowly defined set of target tasks [16, 5]. This is unlike foundation models in NLP, where the use of a single, canonical architecture applied to a wide range of tasks unlocks transfer learning and generalist models.

In this work, we study several improvements to the decision transformer based on treating the model as a pure language model. This makes it possible to pre-train and/or regularize the model with NLP tasks, while fine-tuning it on the task-specific RL objective. It also shifts the focus in RL tasks away from architecture search toward data-centric development. As an application of this data-centric view, we describe a novel approach to addressing the notorious problem of *reward-mismatch*: in many decision making tasks, the achievable return for an episode cannot be known ahead of generating the roll-out of the policy at run-time. This makes it impossible to know the appropriate return with which to prompt the model. We show that a simple data pre-processing step, that biases the model toward attempting to approximate the achievable return, allows us to achieve near-optimal performance by simply always prompting with a high desired reward and letting the model correct to the highest achievable reward.

2 Re-purposing a pre-trained language model as decision transformer

We follow [1] and consider the decision making task as the task of modeling sequences of triplets (a, s, r), based on training data $\{(a_t^i, s_t^i, r_t^i)\}_{i=1}^N$ where a_t^i is an observed action at time t, s_t^i is an

[*]Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

observed state at time t, and r_t^i is the cumulative return (or “return-to-go”) that the given training trajectory incurred from time t onward. At inference time, the agent is prompted with a start state s and desired return r and generates an action a (followed by further states, returns and actions).

2.1 Encoding

When using a pre-trained language model, it is necessary to commit to the model’s tokenizer and vocabulary, and to express actions, states, returns as language. We restrict the encoding of these to ones that can be detected by a regular expression (as discussed below). Within these confines, we experimented with a variety of encoding schemes, including white-space separated values, XML-notation (“$\langle s \rangle 3 \langle / s \rangle \langle r \rangle 5 \langle / r \rangle \langle a \rangle \ldots$”), comma separated values, etc.

We found simple comma-separated notation (“$s = 3, r = 5, a = \ldots$”) to work well and use that in all our experiments below. We note in passing that this encoding permits interleaving data from other tasks, including simple language, with the task-specific data.

2.2 Environment–model interaction

In this work, we model the environment as a Markov decision process (MDP). Our goal is to absorb the decision making task solely within the language modeling objective rather than solving it using a specialized architecture. Therefore, the model architecture does not have any task-specific heads (unlike, for example, [1]), nor any task-specific tokens or word embeddings (unlike [11]).

In the absence of a task-specific output, it is necessary for the environment to continuously monitor and parse the model output to detect any well-formed actions within the model’s output stream. We use a regular expression to this end, that matches on action outputs. While this provides sufficient flexibility in the encoding of actions (including all encoding schemes discussed above), it is simple and can admit highly efficient implementations.

To use a general-purpose language model we need to commit to the model’s tokenizer. As a result, the output sequence generated by the model is not guaranteed to advance one character at a time. For example, it is common to train auto-regressive language models using sub-word representations, such as byte-pair encoding [12], in which case a single model inference typically generates multiple characters. For optimal flexibility, we let the environment interact with the model on the character level instead.

We propose letting the environment continuously monitor the model output by maintaining (i) a buffer of characters emitted by the model so far and (ii) a pointer to the end of the last well-formed action detected so far. Each emission by the model is then parsed with the regular expression, and the pointer is updated, if applicable. Whenever a well-formed action is detected, the environment generates a state and a reward, which are encoded and appended to the model’s output stream. Writing back into the model’s buffer at run-time has been referred to as *environment forcing* by [9], and it is used also by [6, 13, 2] and others since.

2.3 Training on crops

Unfortunately, the character sequences at the transitions between model output and environment output are not guaranteed to contain token sequences that are common (or even contained at all) in the pre-training or fine-tuning dataset, and they depend on the encoding for actions, states and rewards. We observed in our experiments that this can prevent the pre-trained model from generating any reasonable decision sequences. This is a form of *exposure bias* (see, for example, [7]: at inference time, the model sees token sequences that are rarely or never observed during training.

We found that the following, simple fix resolves this issue: for fine-tuning the model on decision sequences, we populate the training dataset with randomly generated crops of *proper sub-sequences* as a pre-processing step. The crops are generated at the character level. This ensures that (in the limit) each character occurs as the first or last character in a sequence and exposes the model to every character sequence that environment forcing may generate at inference time. We perform this step in all our experiments below and do no further processing nor any task-specific architecture adaptations to the model.

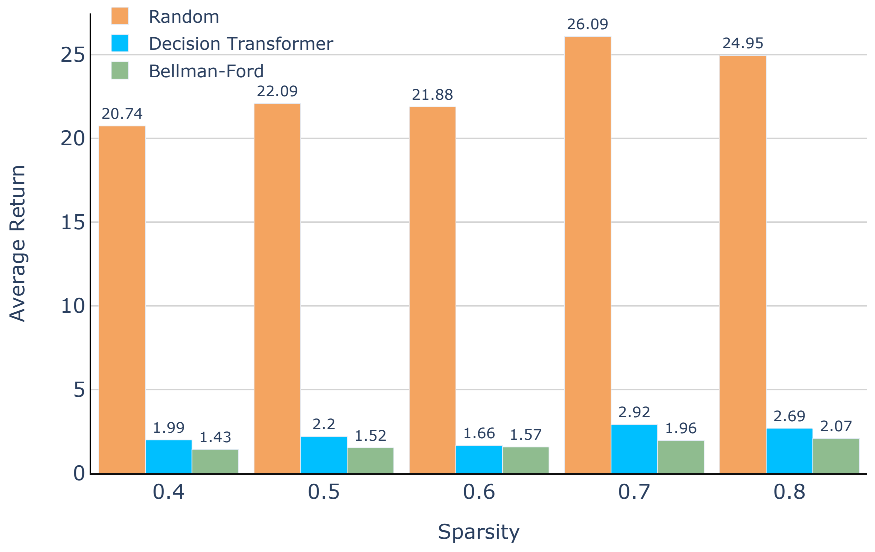

Figure 1: Performance of language-only decision transformer as compared to random walk and Bellman-Ford on a shortest path problem with varying levels of sparsity. The average number of steps to the goal are measured with 100 randomly generated 20-node graphs of each sparsity level.

3 Experiments

We study the viability of the language-based decision transformer on the shortest path problem. The shortest path problem [14] is a classic search problem, where the goal is to find the shortest path from a start node to a goal node in a given graph. It is common to use it as a reinforcement learning test bed by conditioning on a given, fixed graph (usually randomly generated) and presenting traversals of the graph alongside rewards. The goal of the agent is to reach the goal node through a sequence of transitions. Since the graph is not fully connected, not all attempted transitions are valid. We adopt the reward structure proposed by [1], which amounts to generating a reward of -1 for each attempted transition, including those that are invalid, and generating a reward of 0 after the goal state is reached.

Following [1], we train the model on random traversals (of length 30 steps) of randomly generated graphs with 20 nodes and varying levels of connectivity. We use $60k$ sequences for training and train for a single epoch. We use the data pre-processing step discussed in the next section, and at inference time prompt the model with a random initial state and an optimal reward request of -1. The model interacts with the environment using regular expression parsing as discussed in section 2.2. Figure 1 compares the performance of the language-based decision transformer ("DT" in the figure) with the performance of a random walk ("Random") and the optimal performance obtained using Bellman-Ford ("Bellman-Ford").

3.1 Turning reward prompts into reward requests

Reward conditioning in a decision transformer amounts to "selecting" from the trajectories learned by the model one which is associated with high return. When a conditioned reward is not achievable given the state, this "selection" fails and can generate behaviours not encountered during training and thereby generate trajectories with very bad performance [1, 16].

Unfortunately, conditioning on an achievable return is not possible at inference time because it is not usually known which return is achievable for any given state. [1] propose resolving this issue by incorporating a prior into the sampling procedure used by the model.

In this work, we explore a simple and entirely data-centric alternative, which does not amount to modifying the model architecture or sampling mechanism at all. Our approach amounts to pre-processing the training data used for fine-tuning the model to encourage the model to generate the

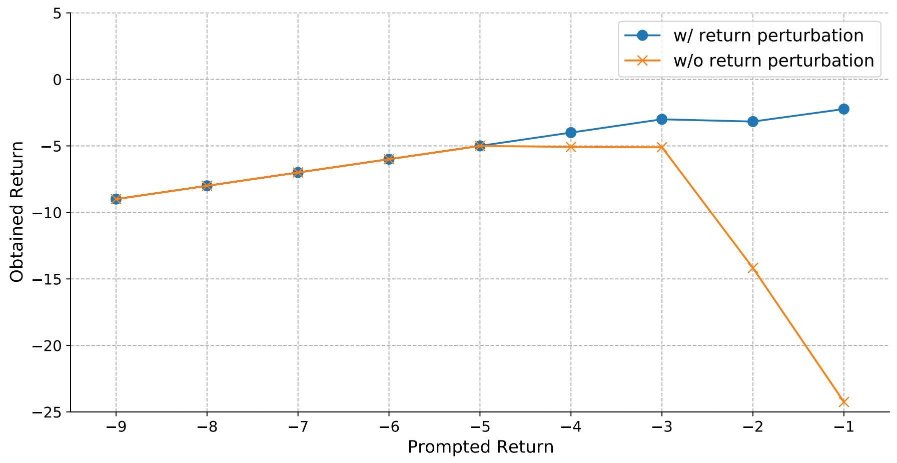

Figure 2: Comparison of obtained return (y-axis) for varying return prompts (x-axis) for a model trained without perturbed reward requests (“w/o return perturbation”) and a model trained with perturbed reward requests (“w/ return perturbation”).

best possible approximation for a given return. It amounts to effectively changing the semantics of return prompts from “trajectory selection” into “return request”.

To this end, after generating the initial training data, we replace the *first* return encoding to a random integer (which essentially plays the role of a “return request”) r with a random perturbation $\hat{r}$ such that $\hat{r} < r$. The rest of the sequence remains untouched. This results in an overly optimistic return request, which the model learns to replace with a more realistic return estimate in the subsequent time-step. This allows the model to learn to replace overly demanding return requests by the next best *achievable* return estimate and to subsequently follow the resulting trajectory. We perform this pre-processing step before the random crop generation discussed above.

Figure 2 shows the result. For state/return values that are achievable by the model, conditioned and obtained returns are highly correlated, but for the vanilla decision transformer, obtained returns degrade sharply for overly large return prompts. Performing the data pre-processing step before training (labelled “w/ return perturbation”in the figure) changes this behaviour and allows us to simply prompt with the best achievable return after training.

4 Discussion

We discussed the challenges associated with turning a pre-trained, *fixed* language model into a decision transformer, as well as solution to those challenges. We demonstrated the viability of these on a shortest path problem. While our results show both the possibility and advantages of using pre-trained language models for decision making (for example, by allowing for flexible return prompts), in future work we aim to study these in more challenging tasks. Pre-trained language models may come with additional advantages, such as the ability to mix in with other, RL or non-RL, objectives. This may greatly benefit the current push towards more generalist models (for example, [10]). It would also be interesting to explore data-driven (as opposed to architectural) approaches to deal with environment stochasticity [8, 15].

Another avenue for future research is to mix action emissions with other textual outputs. The interactions with the environment via regular expressions would allow the model to interleave language-based reasoning with the emission of actions. In complex tasks, it would be possible for the model to allocate varying numbers of tokens (and hence, computation-time) to the task, making it possible to study the benefits of adaptive computation time [3] in the context of modern foundation models.

References

[1] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[2] K. Cobbe, V. Kosaraju, M. Bavarian, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[3] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[4] M. Janner, Q. Li, and S. Levine. Reinforcement learning as one big sequence modeling problem. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021.

[5] S. Li, X. Puig, Y. Du, C. Wang, E. Akyurek, A. Torralba, J. Andreas, and I. Mordatch. Pre-trained language models for interactive decision-making. *arXiv preprint arXiv:2202.01771*, 2022.

[6] R. Nakano, J. Hilton, S. Balaji, J. Wu, L. Ouyang, C. Kim, C. Hesse, S. Jain, V. Kosaraju, W. Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[7] R. Y. Pang and H. He. Text generation by learning from demonstrations. In *International Conference on Learning Representations*, 2021.

[8] K. Paster, S. McIlraith, and J. Ba. You can't count on luck: Why decision transformers fail in stochastic environments. *arXiv preprint arXiv:2205.15967*, 2022.

[9] G. Recchia. Teaching autoregressive language models complex tasks by demonstration. *arXiv preprint arXiv:2109.02102*, 2021.

[10] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[11] M. Reid, Y. Yamada, and S. S. Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[12] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[13] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[14] Wikipedia contributors. Shortest path problem — Wikipedia, the free encyclopedia, 2022. [Online; accessed 24-September-2022].

[15] M. Yang, D. Schuurmans, P. Abbeel, and O. Nachum. Dichotomy of control: Separating what you can control from what you cannot. *arXiv preprint arXiv:2210.13435*, 2022.

[16] Q. Zheng, A. Zhang, and A. Grover. Online decision transformer. *arXiv preprint arXiv:2202.05607*, 2022.