Towards Hybrid Deep Reinforcement Learning with Decision Transformers for Large Discrete Action Spaces

Kevin Rohner [*1] **Philipp Brodmann** [*1] **Bastian Schildknecht** [*1]

Abstract

In this study, we explore the use of offline and online decision transformers in the context of deep reinforcement learning. We apply a decision transformer to the problem of solving the game 1010!. We aim to compare the performance of the offline decision transformer to the online decision transformer. We find that the unexpected high complexity and large action space of 1010! poses significant challenges to simpler decision transformer architectures and likely requires larger models and more training data to achieve good performance. Additionally, we explore the use of custom CNN-based observation encoders and find that they can accelerate training as shown in overfitting experiments.

1. Introduction

Since the publication of the famous Google paper "Attention Is All You Need" in 2017, the transformer architecture has experienced a rocket-like rise and found its way into almost every field of deep learning. Whether it is computer vision with vision transformers as proposed by (Dosovitskiy et al., 2020) surpassing the performance of CNNs, biological chemistry with AlphaFold working on protein folding applications (Jumper et al., 2021), or the most prominent example ChatGPT in NLP, transformers have "transformed" the field they entered.

This project, however, addressed a more recently conquered area of the sequence-to-sequence framework, reinforcement learning. Building on the sequence-to-sequence approach used by LSTMs, where the environment is modeled as a sequence of action, reward, and observation triplets, the concept of Decision Transformers and Online Decision Transformers emerged. While both share the concept of pre-training, they differ in the formulation of the objective function and, as the name suggests, the latter provides online fine-tuning with replay buffers.

For our approach, we tried to create a hybrid framework where pre-training of a decision transformer is followed by fine-tuning in an online setting.

[*]Equal contribution [1]Department of Computer Science, ETH Zürich, Switzerland. Correspondence to: Kevin Rohner <rohnerk@student.ethz.ch>, Philipp Brodmann <philipbr@student.ethz.ch>, Bastian Schildknecht <sbastian@student.ethz.ch>.

Student Project of the Deep Learning Course at ETH Zürich, Switzerland, 2023.

2. Related Work

The core concept that our hybrid approach tries to combine comes from two papers that build on each other. In the following, we present the main ideas from these papers and clarify the difference between the two approaches.

Decision Transformer. The groundwork for the whole project is based on the Decision Transformer paper, which gave us the framework to formulate the offline reinforcement learning problem as a context-dependent sequence modeling problem (Chen et al., 2021). This is in contrast to the more conventional formulation of a policy optimization problem.

Online Decision Transformer. While the viability of decision transformers was shown in training on a pre-recorded trajectory data set, the online decision transformer (Zheng et al., 2022) can be used in online fine-tuning after the initial pre-training. It uses a replay buffer initialized with carefully chosen trajectories. It implements a FIFO replacement strategy of trajectories with the online runs achieved by the transformer. The loss objective is reformulated from cross-entropy overpredicted and target action to a maximum-entropy approach over a stochastic policy in the form of a multivariate conditional Gaussian, as detailed in Section 3.

3. Models and Methods

In this section, we outline the models and methods used for our training. Firstly, expert trajectories are created by human players playing the game and recording their steps. These trajectories are then used by the offline decision transformer both for training and validation. In order to evaluate the model, it plays in a live gym environment afterwards. Lastly, an online decision transformer uses a pretrained model as a start and then plays in a gym to further train.

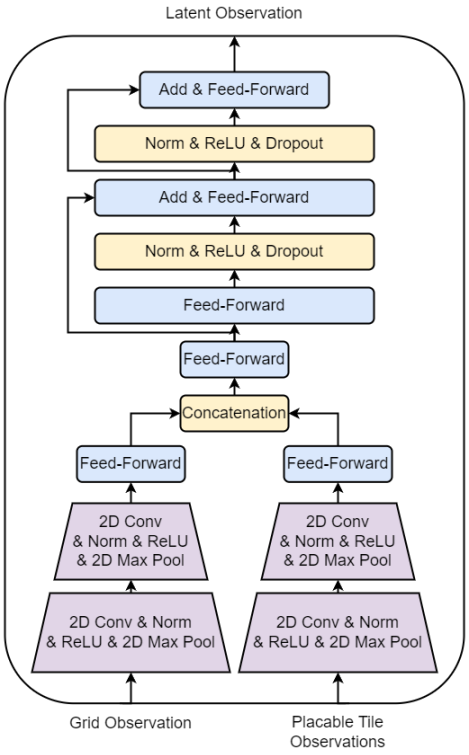


Figure 1. The observation encoder as used in our model consists of two separate 2D convolutional preprocessing stages, one for the grid and one for the three placable tiles, followed by a residual feed-forward network aimed at creating rich latent representations for the complex observation space presented by the game.

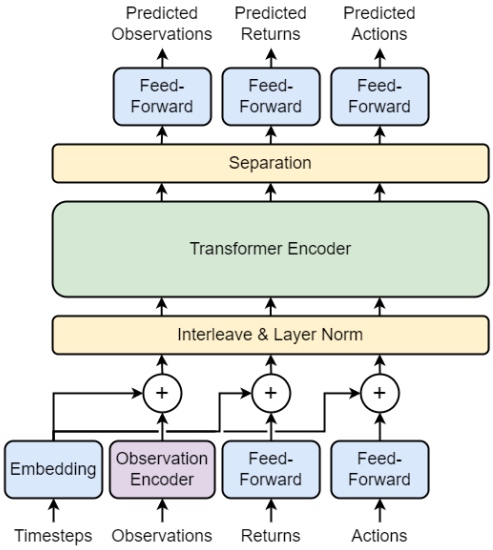


Figure 2. Overview of the complete model consisting of the linear encoders for returns and actions as well as the encoder for observations described in Figure 1, combined with the transformer encoder as described in Figure 6. The encoded observations, returns, and actions are added together with the timestep embedding. The result is then interleaved and normalized before being fed into the decision transformer.

3.1. Creating Expert Trajectories

The game is implemented in Unity. Using the play mode, a human player can play the game while recording every state, action and reward. These trajectories are eventually saved to JSON files, which allow for relatively simple merging for bigger datasets

3.2. Offline Decision Transformer

Our offline decision transformer follows the implementation of (Chen et al., 2021) for the most part. It takes a batch of sequences of states, actions, returns-to-go and time steps as input, which will be embedded and run through a transformer. In the final step, the output of the transformer is used to return predictions for states, actions and returns-to-go (Figure 2).

Input Embedding. Time steps are embedded using an embedding layer, which needs to know how long the maximum episode can be. For both actions and returns, a linear layer is used, mapping from their respective dimensions onto the hidden dimension the transformer works on. Since 1010! works on a discrete action space, we additionally perform a one-hot encoding ahead of passing them to the decision transformer. As for the state, two different methods were tested: The first one follows the implementation of the paper and uses a simple linear layer. Since the state is 2-dimensional, we tested a second method that uses 2D convolutions, on the assumption that this might better capture the state (Figure 1).

Transformer. The transformer used in our decision transformer is a PyTorch TransformerEncoder, which is different to the custom GPT2 transformer used in the original paper. A transformer encoder typically consists of multiple layers (Figure 6), each consisting of a self attention block, whose output is added together with its input and fed into a normalization layer. Its outputs are in turn given to a linear layer and then added and normed, again (Figure 5).

It is given our interleaved sequence of returns, states and actions, and produces a new sequence which is used for the predictions in the final step of the decision transformer. Additionally, an upper triangular attention mask is given to the transformer, making current tokens attend only to previous tokens.

Predictions. States, actions and returns are predicted using linear layers. For the actions, a Tanh layer can be used in addition to the linear one.

Loss Function. The loss is only calculated on the actions. In contrast to the games played in the original paper, which all had continuous actions, our action space is discrete, consisting of a block selection, an x- and a y-coordinate. Therefore, we one-hot encode actions, and this also means that we use a different loss function than the MSE, namely a cross-entropy loss. This loss is calculated for each of the three encoded actions and then added together for the total action loss, which is used for the backwards step in the model.

Validation and Evaluation. After training on a large dataset of expert trajectories, the decision transformer model is also run on a smaller validation dataset to see whether it is overfitting to the training set. Using either the current model or a previous pretrained model, it is also possible to evaluate them in a Unity gym environment, which allows examining the actual performance with both rewards and the visible score in-game.

3.3. Online/Hybrid Decision Transformer

The online decision transformer is in some ways very similar to and in others very different from the offline decision transformer. Our implementation is essentially a wrapper of the offline decision transformer. It needs a pretrained one as an input, and then changes its parameters as needed and calls its methods.

Action Prediction. One such parameter is the action prediction layer. It is no longer a linear layer, but instead returns a multivariate conditional Gaussian distribution. In order to get an actual action out of this distribution for evaluation and online training purposes, it is possible to sample from the distribution.

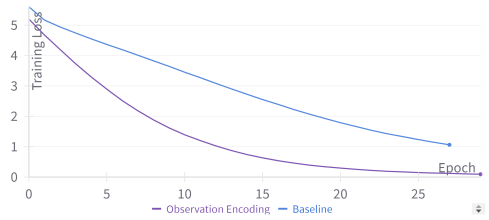
Loss Function. Another thing that is different is the loss function. Instead of the cross-entropy loss, we use a negative log-likelihood loss to evaluate how similar a target action is to the predicted distribution. In the paper of (Zheng et al., 2022), they made the decision to use this loss for both the offline and online decision transformer, which most likely leads to better results when the model switches from offline to online training. In our implementation, this is different, and the decision transformer needs to switch from a model pretrained on cross-entropy loss to the Gaussian distribution.

4. Results

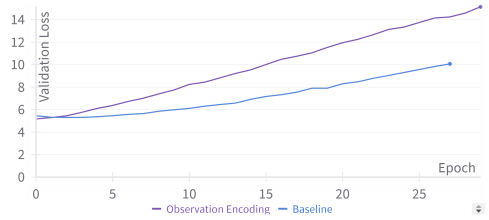
Preface. We regrettably preface this section with the confession that our work and experiments did not yield any result. We failed to reproduce the implementations of the offline and online decision transformers found in the referenced papers.

Attribution. We attribute this first and foremost to the lack of data to train on. Based on the rapidly decreasing training loss in Figure 3b and rising validation loss in Figure 3a, we assume that the model is heavily over fitting to the small amount of trajectories and transitions. This, unfortunately, renders any guessing on performance useless.

Effect of State Convolution. Besides all this, we some results from the observation encoder network for state embedding, which we tried in place of the linear layer used in both the other papers. Using the custom embedding, the training loss decreased quicker, indicating a faster learning. On the downside, the validation loss, which increases for both methods, increased **Functionality.** The provided framework within our code base is functional with regard to validated training loops for the offline and online decision transformers, allows for multiple ways of visualizing training, evaluation and replay of single trajectories or datasets. We provide in figure Figure 4 a histogram comparison of the cumulative rewards between a trained offline decision transformer and part of the used dataset constructed by a human player.



(a) Comparison of the training loss between the same model with and without the observation encoder.

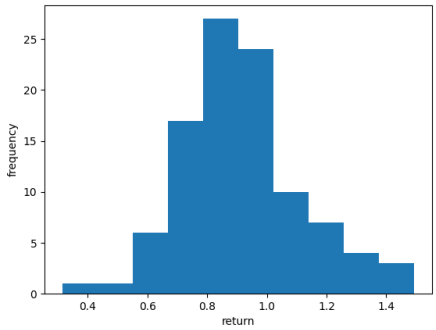


(b) Comparison of the validation loss between the same model with and without the observation encoder.

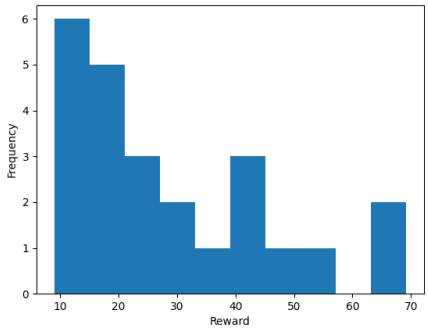
Figure 3. Comparison of the training dynamics for the same model based on the offline decision transformer, with and without observation encoder.

5. Discussion

In this section, we want to discuss possible reasons and explanations for our shortcoming of providing any established results. In the same breath we also want to add pointers for improvement, possible fixes and even suggest further research areas to arrive at a conclusion to the question of



(a) Final cumulative rewards of the trained agent over 100 episodes.



(b) Final cumulative rewards of an experienced human player over 24 episodes.

Figure 4. Histograms showing the final cumulative rewards achieved in 100 episodes by the agent and an experienced human player in 24 episodes. Note the very different x-axis scales.

compatibility.

5.1. Dataset Size and Game Complexity

As mentioned in the section 4, the training dynamic graphs suggest a fast over fitting to the given dataset. This is further underlined by the corresponding rise in validation loss. Due to this, it is nearly impossible to compare the chosen features in our model architecture or even contribute any meaning to different choices.
In addition, a short consideration of the action and state space hints at the unexpected complexity of the game (Coelho, 2016). This could be taken as yet another hint at how much data is needed to draw any conclusions on the success of the implemented architecture.
As an obvious solution, we suggest creating much more data with our Unity implementation of the game.

5.2. Compatibility of Approaches

Since the online decision transformer builds on a pretrained decision transformer, we yet again can only speculate over the observed behaviour.
Our approach to overwrite the action prediction with the multivariate conditional Gaussian distribution surely leads to a vast amount of relearning in earlier layers, let alone the impact of having a new loss objective. If one finds results that suggest this relearning to destruct too much of the pretraining, we propose to freeze the pretrained layer weights and prohibit trajectory replacement in the replay buffer for a certain number of epochs. After this initial warm-in period, the weights can be released to allow for the fine-tuning

6. Summary

This project tried to delve into the application of decision transformers in reinforcement learning, using the modeling of environment interactions as sequences. We specifically take alookt at the context of a hybrid approach combining pre-training with online fine-tuning. Drawing inspiration from the Decision Transformer and Online Decision Transformer papers, we implemented an offline decision transformer for pre-training on expert trajectories and an online decision transformer for further refinement. Our experiments, to our regret, did not yield conclusive results, which we attribute to limited dataset size that lead to over fitting issues.Despite the lack of concrete results, our framework provides a functional basis for future research in this domain.
We propose to counter this limitation by generating more training data, for example by using our Unity game implementation. Additionally, we discuss potential improvements, such as freezing the weights of retained layers of the pretrained model during the initial online fine-tuning phase to preserve valuable knowledge.

References

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling, 2021. URL https://github.com/kzl/decision-transformer/tree/master.

Coelho, F. 1010! analysis, 2016. URL https://blog.coelho.net/games/2016/07/28/1010-game.html.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image

recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

Jumper, J. M., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Zídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D. A., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021. URL https://api.semanticscholar.org/CorpusID:235959867.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer, 2022. URL https://github.com/facebookresearch/online-dt/tree/main.

A. Appendix

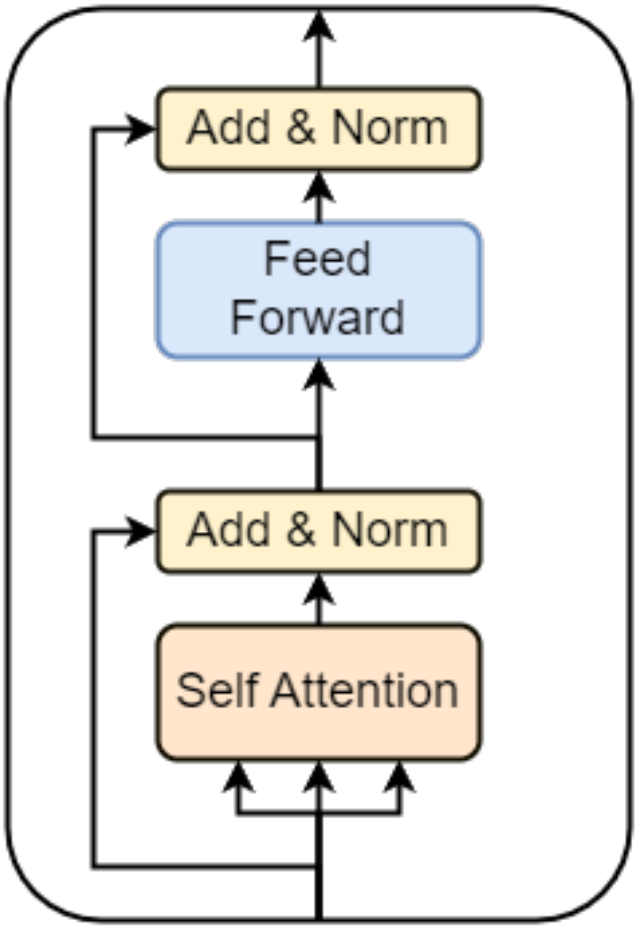


Figure 5. A single layer of the used transformer encoder model consisting of masked self-attention, layer normalization blocks, and a feed-forward neural network. This architecture is identical to the one proposed by (Vaswani et al., 2017) except just using a single self-attention head.

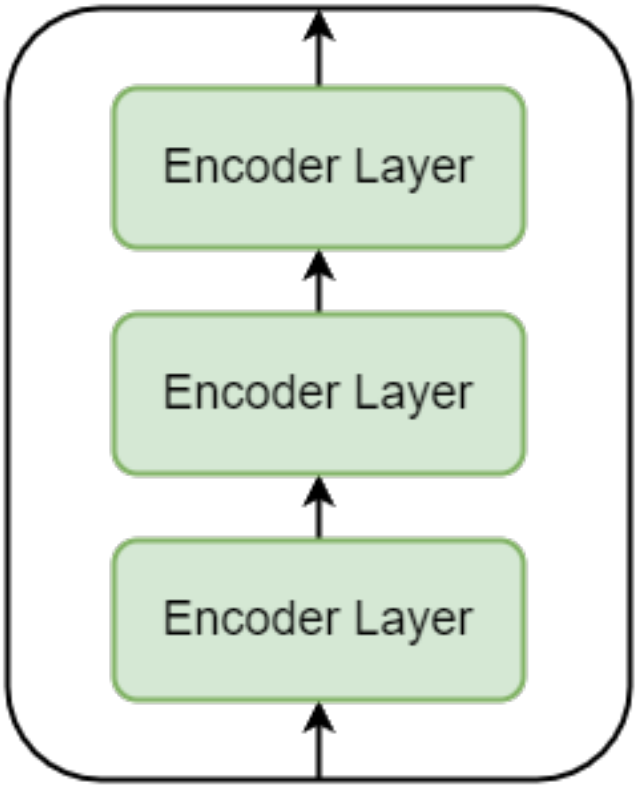


Figure 6. The used transformer encoder architecture consists of three identical layers as described in Figure 5.