Analysis of Transformer Behaviour in Reinforcement Learning

Miguel Silva Amaral
miguel.s.amaral@tecnico.ulisboa.pt
Instituto Superior Técnico, Lisboa, Portugal

***Abstract*—Decision Transformer (DT) [1] introduces sequence modeling as a method for achieving desired outcomes in reinforcemet learning (RL) problems, while avoiding common issues that cause divergence in this setting [2]. States, actions and future returns are sampled from a dataset in order to condition how the model will plan a certain trajectory, while masking future parts of the sequence to make each timestep attend only to its past in an autorregressive fashion. Some of the DT's bottlenecks are identified in this work, and architectural changes are implemented in order to stabilize how this algorithm performs between each episode of the environment. Because DT is limited by the diversity and quality of its dataset, both an off-policy and an on-policy transformer algorithm were developed as proof-of-concept of how exploration would apply in sequence modeling algorithms. Finally, a shift from using toy problems to working with real-world financial markets was implemented to show that RL can also be used to solve problems that occur in the real world. Experiments demonstrate that there is room for improvement in current sequence modeling solutions, and that transformers have a significant amount of untapped potential when they are used independently for exploration.**

***Index Terms*—Deep Reinforcement Learning; Transformers; Sequence Modeling; Financial Markets;**

I. Introduction

Sutton and Barton [3] draw a parallel between RL and the way living beings learn through persistent observation and interaction with their surrounding environment. This observation-interaction pair opens doors to new ways of ponderation whenever a different challenge, in which past experiences may be applied, is faced.

Although this claim is correct, there are significant differences between how a living being would learn and how a machine would do the same. First of all, the amount of information required for a machine to match the way a human performs on a specific task exceeds the number of perceptions a human uses for extrapolation and decision-making, making the training of any reinforcement learning model a data-intensive process (often requiring millions of environment steps) which also makes it a time-consuming process that requires a large amount of computational power.

Second of all, no system comes close to the ability of generalization that a human has, without having changes implemented into its structure, because all systems lack the capability to generalize concepts and the ability to learn new tasks in a sample-efficient manner.

Lastly, low sample efficiency may provide solutions to the majority of toy problems that are usually treated in this area, such as simulations and games.

The main objective of this paper is to take advantage of parallel processing capabilities of the transformer architecture in order to solve RL problems with reduced time complexity, thus achieving greater efficiency in the learning process and opening doors to the possibility of scaling those algorithmic and architectural choices onto tougher problems.

This research evaluates the capabilities of the DT by investigating the impact of various implementations of context length, positional embeddings, and so on on stability and convergence speed in various datasets, identifying learning bottlenecks that could be improved. For example, depending on the trade-off between quality and diversity, a certain dataset may exhibit superior performance or target modeling capabilities. A sequence modeling adaptation of popular on-policy and off-policy algorithms is also introduced in this work.

Following up on that analysis, this paper offers the following contributions:

- Identification of sequence modeling bottlenecks in the Decision Transformer architecture
- Evaluation of different transformer implementations in offline reinforcement learning
- Application of transformers on model-free online reinforcement learning algorithms
- Evaluation of sequence modeling in a trading environment

II. Theoretical Background

A. Transformer Architectures

Transformers are a model that uses self-attention, a mechanism of correlation between positions in a sequence, to better represent patterns. This architecture takes great advantage of parallel processing, greatly reducing the complexity of its training time.

The transformer may be split into two parts, the encoder and decoder stacks respectively. The main objective of the encoder layers is to encode source sequences and turn them into state vectors, allowing representations to be learned and the dimensions of data to be reduced, while the decoder generates a target conditioned on the vector state.

Multi-head attention allows the model to focus on different positions of a sequence and by giving the attention layer multiple representation spaces, meaning that each token will

be represented in parallel multiple times, allowing for multiple patterns to be identified.The output in position i of the attention layer is given by:

$$Attention(Q, K, V)_i = Softmax\left(\frac{Q_i K_j^T}{\sqrt{d_k}}\right)V_j, \qquad (1)$$

where d_k is the dimension of the key vector and j and i are the indexes used in the example. To mimick causality, j is limited to the location of i, making the attention conditioned on the past where d_k is the dimension of the key vector.

The functioning of the transformer architecture, as presented in 1, begins with the processing of the input sequence in the encoder. The output of each encoder is passed as the attention vectors K and V that will be used in encoder-decoder attention that will help the decoder focus on the appropriate places.

The system ends by taking in the array output of the last decoder and translating it a token at a time into the desired type of information. This is done by running the array through a fully connected layer and through a Softmax layer that will turn each possible token into a probability, choosing the output according to the highest probability possible.

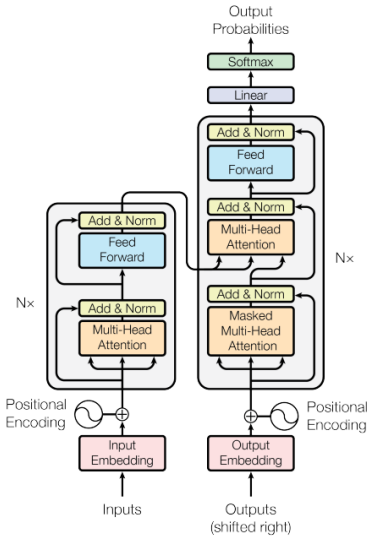


Fig. 1: Transformer architecture (Vaswani et al. 2017)

1) GPT-2 [4] and BERT [5]: innovate by bringing a state-of-the-art solution to natural language processing that are focused on a transformer model built using only a decoder or encoder stack respectively.

In GPT-2 the usage of auto-regressive masking gives the sense of causality that maps a relationship between the token at present time and its previous sequence only, as was described previously.

2) Transformer-XL: Often, the defined context length of a certain problem doesn't keep the necessary information to solve it and for that reason Transformer-XL [6] uses Relative Positional Encodings in order to avoid this problem. This attention model would look at repeating parts of a sequence and stores previous segments of a sequence using recurrent units to be used in further segments, making the context length much deeper.

B. Reinforcement Learning

The main components of RL are the agent and the environment, the latter being where the agent acts. At every timestep, the agent observes the current state of the environment and then takes the course of action it finds fitting. With every action the current state of the world will change. The neural network that decides which action the agent should take is called policy.

There are two types of RL Algorithms, those that are on-policy and off-policy. On-policy algorithms are trained using data that was collected according to the most recent version of the policy, while off-policy algorithms may use data that was collected by previous instances of the policy or by data produced outside of the policy. If the data is produced totally outside of the policy the learning process is called Offline RL which is the inverse of Online RL.

1) Off-policy Learning: Q-Learning: Deep Q-Learning [7] is an algorithm that generates an estimate of how beneficial staying in each state is for the agent and which action to take based on that, which is called Q-value.

Deep Q-Learning starts by creating a buffer where each transition in the environment is stored (a tuple with s_t, a, reward and s_{t+1}). After that the next action will be picked by the policy according to the maximum output of the network. Finally the Q-value in the current state will be updated according to the mean squared error between its current amount and a target Q-value that is the sum of the reward obtained and the maximum Q-value obtained in the next state, multiplied by a discount factor γ, as given by:

$$Q(s_t, a_t) + (r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)). \qquad (2)$$

Exploration can be added by exploring with random actions with probability ϵ, making it an $\epsilon - greedy$ policy.

Double Q-learning [8] uses two networks for Q-value estimation, one to generate the target and the other to generate the current Q-value. This ensures stability.

Deep Recurrent Q-networks [9] replace some of their policy layers with recurrent layers like GRUs [10] taking advantage of its hidden state to store information between timesteps and to learn patterns that unfold during a trajectory.

2) On-policy Learning: Proximal Policy Optimization: Proximal Policy Optimization (PPO) [11], succeeds other Actor-Critic methods like A2C [12] that combine both value-optimization like Deep Q-networks, and Policy Optimization by using an Actor that controls how the Agent behaves.

PPO uses a method to avoid large updates to the policy, which is called the Clipped Surrogate Objective, which balances training speed with stability by limiting on how big of an update to do on the network. The update is done via gradient ascent in order to maximize the advantage, A_t, that a certain action, a_t, has against the remaining actions in that state. This is balanced via the log-probability of taking that action, as given by:

$$E_t[log\pi_\theta(a_t|s_t)A_t] \tag{3}$$

The ratio of the update between the old and current Policy is limited between $1-\epsilon$ and $1+\epsilon$, replacing the log-probability in the original equation.

Recurrence can be added similarly to what happened in Deep Q-Networks. The value and policy networks can share their body or not [13], which may be useful to understand features that are relevant for both value estimation and action prediction.

C. Transformers and RL

1) The Gated Transformer-XL architecture (GTrXL) [14]: stabilizes the base transformer architecture by introducing gating mechanisms, that replace the residual connections in the base transformer architecture alongside a change of position of the layer normalization segments. This set of architectural changes are evaluated with an implementation of the V-MPO algorithm [15]. It is claimed that this type of identity mapping allows the model to be initialized closer to a Markovian Policy and that is why it tends to stabilize more easily.

2) Decision Transformer: Some literature tackles RL as a problem that can be solved with Supervised Learning [16] and for that reason Decision Transformer [1] and Trajectory Transformer [17] were applied to the area of Offline RL.

The main focus consists on enabling transformers to learn patterns, as they do so well in NLP, by taking advantage of auto-regressive models that imitate causality like GPT-2.

The model is fed with the objective returns, along with the states and actions of a sequence, that takes in a temporal embedding in order to distinguish positions between States, Actions and Returns-to-go. The predicted action will be compared with the desired action via Mean Squared Error and that is how the model will be trained.

Trajectory transformer incorporates planning by using beam search to condition future states trajectories according to the future desired state in order to get similar results.

3) Further Work: In [18] it is proven that pre-training models on other domains allow for faster convergence and better results in this setting. Bidirectional encoding is also proven to be possible [19] and online exploration with the Decision Transformer was also expanded upon [20].

III. Methodology

A. Outline

For sequence generation in RL, decoder-only architectures prove themselves to be the strongest use case of Transformers, since causality dictates how decision making is made based on the current state of a model and its state on a previous timestep. Future actions would only be modeled by predicting each future state according to a plan of actions.

Even though that Q-Learning and Policy Optimization methodologies are really popular, they remain under-explored when combined with sequence modeling. The main contributions of this paper are heavily focused on developed proof-of-concept methodologies with this objective in mind, while guaranteeing that proper validation and possible improvements were done to Decision Transformer.

This section is divided between the experiments done in the Offline RL setting, that expand and validate Decision Transformer's results and those that tackle exploration and popular RL algorithms.

B. Decision Transformer

1) Network: The base network that was used to reproduce DT's results begins by receiving the numerical order of the timesteps in a sequence as well as a set of actions, states, and returns-to-go (RTG's) in each timestep. The states and actions are drawn from the dataset, while the RTG's are calculated by summing the obtained reward from the end of the trajectory to the current point.

The timesteps pass through an embedding layer, that simply stores embeddings of a fixed size, in this case the maximum number of timesteps possible, producing no gradients. The states, actions and RTG's go through linear layers in order to have the same dimensionality.

After the Linear layers, the temporal embedding is added to the state, action, RTG sequences that are interleaved in temporal order. Layer-normalization is applied before sending the sequence to the decoder stack.

Each decoder is composed by a multi-head attention block and a Multi-layer Perceptron with GELU as its activation function. The end of each of those sub-layers is followed up with residual connections and layer-normalization. Finally a Linear Layer will change the output dimensionality into the number of possible actions and an hyperbolic-tangent activation function was used to limit the actions according to the environment.

Attention is masked in an auto-regressive manner in order to make sure that each timestep attends to past timesteps only.

The models were trained from scratch in three different seeds and using the default hyperparameters from the original paper, often obtaining a slightly higher standard deviation between runs, but an equivalent mean performance.

Evaluation is done with a fixed RTG, that is determined according to the max reward obtained in the environment multiplied by a constant. The only change in hyperparameters was here, since the presented results in some datasets weren't achievable with the provided target RTG's.

2) Evaluation of DT: To properly validate the architectural implementation and to expand upon it, the used environments had to be some of the used in the original paper. For that reason, MuJoCo environments [21] were chosen due to their Observation Space being linear, which makes them more efficient to train, as seen in Coberl [19] and for the difficulty that their continuous action-space adds to the equation.

The environments used were **half-cheetah, hopper** and **walker**, that feature a set of robotic simulators, whose objective is to move fast and in a stable manner toward a certain goal. To do that the model needs to observe a set of

coordinates, angles, velocities and others in order to decide how much torque to apply to each of the robot's hinges.

Datasets for these environments were obtained from the python library D4RL [22], and were of three types, **medium, medium-replay** and **medium-expert**. The medium dataset is generated by a policy that achieves one third of the score of the expert policy. The medium-expert dataset is the concatenation of the medium dataset with a dataset generated by an higher quality policy. The medium-replay dataset is generated from the Replay-Buffer of a medium policy making it more diverse that the remaining ones.

The best scores are expected from the medium-expert datasets, but since medium-replay datasets are the most diverse there is some interest in comparing how both of these would model a less than optimal objective, like not going to fast or too slow for example.

For that reason an evaluation of how DT models targets will be done in a more extensive fashion than what was done in the original article. This will establish how sequence modeling performs across datasets. To complement this, a comparison between how the Transformer architecture would perform with and without its context length is also developed. This provides a study on how data impacts Offline RL and concludes on whether or not the introduction of exploration is necessary in sequence modeling.

Architectural improvements were attempted in order to improve performance and increase stability between runs. The main objective was to make sure that the same policy wouldn't obtain too different of a reward between two randomly generated episodes of the environment, making the model more reliant to random initialization.

C. Transformer Q-Learning

1) Trajectory input and Replay Buffer: Since the algorithm is now using trajectories and not a single observation, the replay buffer will no longer be a storage of the tuple $(s_t, s_{t+1}, a_t, r_t, done)$ and for that reason it became necessary to implement some changes in order to harbor sequences of a fixed context length. For that reason a two replay buffer strategy was implemented.

The first one is a simple first in, first out (FIFO) array that stores the last K timesteps in each tuple in order to be replaced whenever the agent moves a timestep forward. This is reset with each environment run, and while the current timestep is less than the value of K, padding will be done with zeros.

The second one is the classical replay buffer where alterations were performed in order to store each of the trajectories obtained from the FIFO buffer and to sample them according to a randomized batch size, instead of sampling timesteps.

An example of how both buffers are connected is present on figure 2. This image is simplified to only present stored states, but for each state its respective action, next state and reward are also stored.

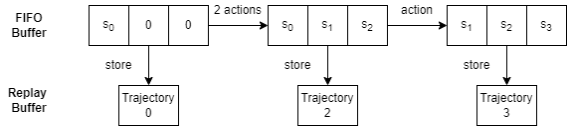


Fig. 2: Double Replay Buffer Architecture

2) Architecture: Two transformers with the implemented improvements from the previous chapter were initialized with the same weights in order to work similarly to double deep Q-Learning, one will be the Policy Transformer and the other the Target Transformer.

A batch of trajectories is fed into the Policy Transformer, where the sinusoidal positional embedding used in the DT evaluation is used in order to retain a sense of order between timesteps, that are not defined formally unlike on DT.

The same decoder blocks with gating that were applied for increased stability were also re-utilized for this experiment and the output layer produces K Q-values to be used in training from a set of K timesteps that compose a trajectory. A small representation is shown on figure 3.

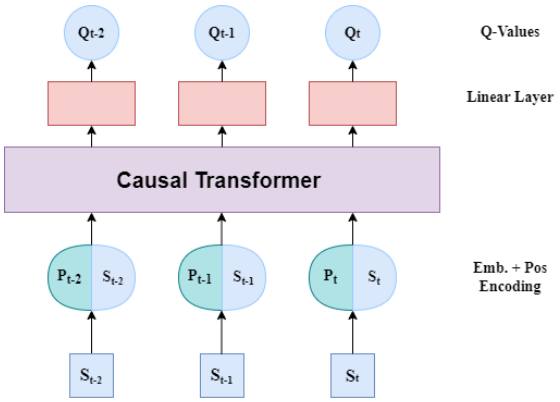


Fig. 3: Transformer Architecture for Q-learning

3) Training: The replay buffer is slowly filled according to an ϵ-greedy policy using only the maximum Q-value, obtained from the last timestep of the policy network or from a randomized choice.

After obtaining enough trajectories, training starts by computing the maximum Q-values for the action-space of the current K states, Q_t, with the policy network. After that the maximum Q-values from the following states are obtained from the target network ,Q_{t+1}.

The target will be given by the rewards obtained ,r_t, summed to $Q_{t+1} * \gamma$, with γ being the discount factor. The loss function will be the mean squared error between the target and Q_t.

The target network will copy the parameters of the policy network after a certain amount of steps and the exploration factor ϵ will slowly decay to 0. This ensures stability in the recurrent updates to the model.

D. Transformer Proximal Policy Optimization

1) Trajectory input: For an adaptation of an on-policy algorithm like PPO, a small amount of changes are necessary

in order to mantain the same objective as in transformer Q-learning.

- The replay buffer needs to be cleared after the policy is updated in order to train exclusively on newly generated trajectories.
- The log-probabilities produced by the policy are now part of the replay buffer, while outcome states (s_{t+1}) become unnecessary. Trajectories are still stored as a whole.
- The FIFO array remains unchanged as the last K timesteps remain necessary in order to train the network to model a sequence.

2) Architecture: A shared network using the improved transformer from previous section was chosen for this type of implementation, representing both the Policy and Value function. Even though separate networks may avoid interference between both of these objectives, certain characteristics of the environment may be of interest to both sections of training, meaning that a certain degree of sample efficiency may be achieved by training in this fashion.

For this reason a transformer similar to the one used in Q-Learning is introduced with two different heads. The **value head** produces a single value for the present state of the network. The **policy head** outputs the probabilities for each action, using a Softmax activation function. Those probabilities are then used to create a Categorical distribution from where log-probabilities and entropy are taken from.

For this model the outputs are not of sequential fashion, since that originated problems with advantage estimation that could not be solved.

A brief overview of the network is present in figure 4:

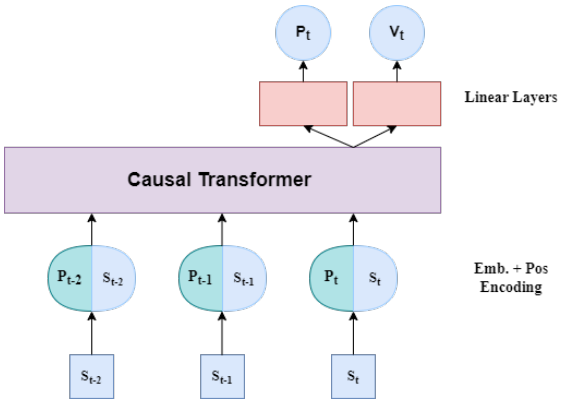


Fig. 4: Architecture for Policy Optimization

3) Training: A limited amount of environment episodes are generated by using the policy head of the model to select the following action. The entropy and log-probabilities of picking each action are stored along with the obtained reward, a terminal signal and the last K states of the model.

After reaching the necessary amount of timesteps the model will be updated. In order to do that the return-to-go from each timestep needs to be computed. That is achieved in the same fashion of what was done in the DT implementation, but here the returns-to-go are normalized according to their mean and standard deviation across one episode.

The current parameters of the network are stored and new log-probabilities, entropy and values will be generated in each training epoch, after the model is updated.

Advantages between the newer log-probabilities and the old ones will be calculated using Monte-Carlo estimation and used to calculate the Surrogate Loss, that will be clipped between $[1-\epsilon, 1+\epsilon]$ times the ratios between the current log-probabilities and the old ones.

An entropy bonus will be added to the loss function to ensure exploration and the value policy will be compared with the returns-to-go via mean-squared-error, composing our loss function.

After the gradient descent step, the buffer is cleared and the process starts over.

IV. Experimental Results and Discussion

A. Original Architecture Validation

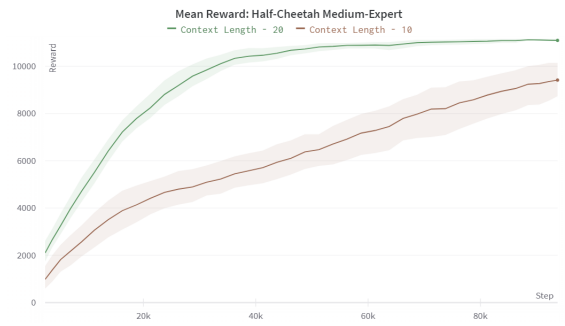
The baseline implementation used to develop this body of work is not only comparable, but often surpasses the original implementation of Decision Transformer having either a higher mean reward or a lower standard deviation between runs.

Since everything was implemented in the same fashion as Decision Transformer, the difference between models may only be attributed to three main differences

- Different implementation of certain functions like multi-head attention or positional embeddings.
- By replacing ReLU activation functions by GELU.
- By training the model from scratch instead of loading a pre-trained model.

1) Impact of the Context Length: In this section an attempt at understanding whether or not it is really necessary to use context or previous timesteps to model a sequence is made. In order to achieve the following results a baseline model was trained with a context length of 20 and another with a context length of 1, which is the same as having no context and simply working with timesteps.

Results are displayed in figure 5.



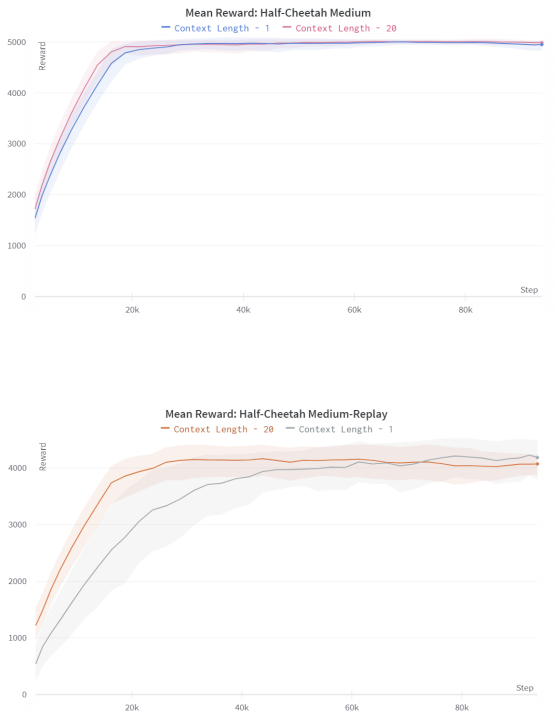


Fig. 5: Context Length: Mean Reward and Standard Deviation across Half-Cheetah Datasets

The conclusions that can be derived from these results show the importance of using a certain type of dataset when training a reinforcement learning model by using supervised learning. Distinguishing each dataset allows for the following judgements:

- **Medium-expert**: It becomes easier to model complicated trajectories with a bigger context. The more complex the environment, the more important it becomes to pay attention to previous timesteps.
- **Medium**: Displays faster convergence and smaller standard-deviation with context. The trajectories are not polished enough for this difference to become significant in the end results when compared with Medium-Expert datasets.
- **Medium-replay**: The trajectories to reproduce are so simple that the model can achieve the same conclusions without context, both in mean reward and in standard-deviation across runs.

The dataset quality demonstrates that the importance of context increases with the complexity of the solution. Understanding a sequence is much more important than understanding a timestep and that is the most important aspect of dealing with difficult environments.

2) Target Modeling: In this subsection, a visualization of how a trained model reacts to a requested target result is displayed in order to establish the behaviour of the agent, when facing different objectives in the same environment.

This demonstrates if the agent is capable of understanding its environment outside of a determined goal and whether or not overfitting occurs in this form of learning. The main premise of this experiment is to question whether or not is an agent really solving an environment if it does not understand how to achieve multiple objectives inside of it.

In the **medium-expert** dataset an interesting phenomenon happens. Due to its lack of bad quality runs, the model only understands part of the given targets. This is evident according to the spike in quality that happens when the target achieves about 50% of its maximum value.

This would be defined as an overfitting boundary, since the model does not understand what is happen beyond that target as displayed in figure 6.

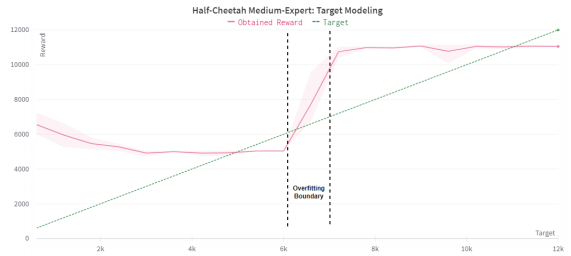


Fig. 6: Overfitting boundary in Medium-Expert datasets

Since the **medium** dataset is an even more reduced dataset than the medium-expert dataset, the limits of its understanding are even more reduced, showing that the model has almost no reaction to a change in target. This would mean that the RTG training signal is even unnecessary in this model, since there is no distinction between good and bad runs inside the same dataset.

In the **medium-replay** dataset the situation changes since the model samples a replay buffer that faced the progression of going from zero understanding of the environment to a full understanding. For that reason and thanks to the presence of diversity the model has an almost linear behaviour across target objectives, being of course limited by not knowing any expert strategies for the best solution.

It is easy to conclude that the medium-replay dataset gives the model a better understanding of its environment than any other dataset and that for that reason the introduction of more diversity to higher quality datasets is a way to solve this diversity bottleneck found in higher quality datasets.

B. Architectural Changes

In this section the results of performing multiple changes to DT are presented.

The metrics used to analyze models take into account the mean reward obtained and its standard deviation between runs. Since DT is a method that uses supervised learning, the training loss is also taken into account, even though it will not be the main metric of evaluation.

1) Positional Embeddings: As previously mentioned positional embeddings were changed from hash-tables to sinusoidal positional embeddings that are learnable and not frozen. The following results will be presented according to environment and dataset. Their analysis will be global in order to identify behavioural patterns.

Figure 7 represents the obtained results in the Half-Cheetah domain.

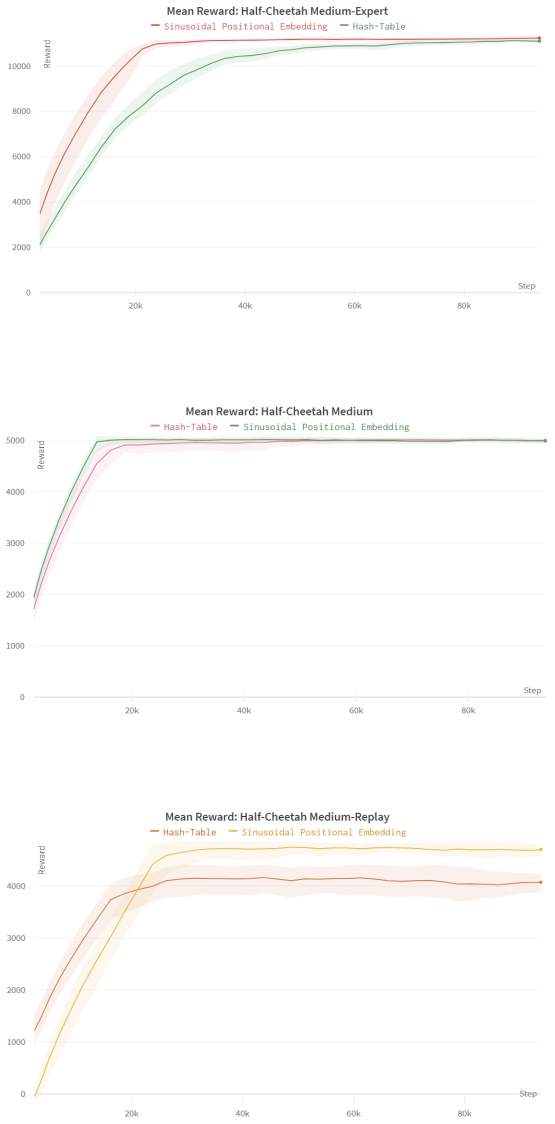


Fig. 7: Positional Embedding: Mean Reward and Standard Deviation across Half-Cheetah Datasets

It is easy to conclude that according to the increasing complexity of the datasets used, the more important it becomes to use a more complex type of positional embeddings. Learnable sinusoidal positional embeddings allow for flexible modeling of timesteps and a better distinction between the beginning and end of long episodes.

Early convergence, lower standard deviation and an overall increase in stability are some of the important traits found across environments on an implementation of learnable sinusoidal positional Embeddings. This does not reflect on a significant improvement on the end results of the model, but it does help the model become more sample-efficient and stable when using medium or medium-expert datasets.

Even though that convergence speed is hurt on the medium-replay dataset, the overall performance benefits greatly from using sinusoidal positional embeddings. The reason for this may be related to the fact that a learnable positional embedding may learn patterns that are not otherwise present in the dataset, which allows for a small amount of extrapolation from an otherwise simple dataset that produces mediocre results when compared with the remaining options.

2) Gating: With the objective of finding a stable solution, attempts to initialize the model in a way that would behave similarly to a Markov Decision Process meant a change in the model architecture that would now include gating mechanisms instead of residual connections.

The following results, that remain coherent across datasets reveal the impact that gating truly has on training, that also translate themselves on an increase of parameters by the transformer network of 11%, getting most of the architectures in the 2 million parameter range.

Differences between both architectures across datasets in the Half-Cheetah domain are represented in figure 8.

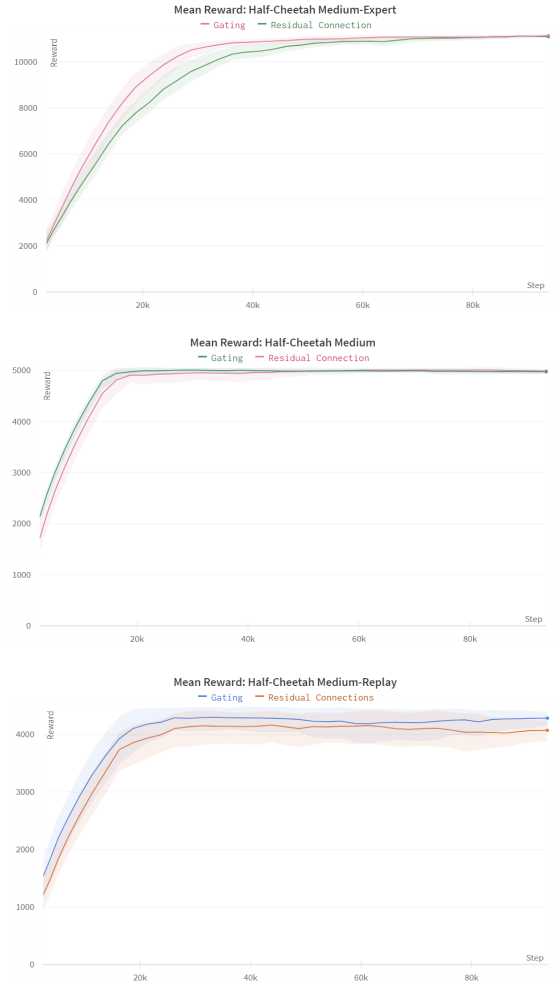


Fig. 8: Gating: Mean Reward and Standard Deviation across Half-Cheetah Datasets

Even though training would take a toll in temporal efficiency due to the increase in parameters, having increased between 35 and 42% for a single run, the powerful performance improvements in standard deviation and sturdiness to random initialization cannot be understated.

Convergence is slightly faster and even if it was not the rapid decrease in standard deviation would at least mean that

the model now behaves more coherently across runs meaning that the peak performances may become slightly lower, but that the worst runs become less of an issue.

Standard deviation would amount to hundreds of reward points in some runs and this is no longer is the case with gating, meaning that the initialization of the network with gating mechanisms really translates in a behaviour similar to a Markov Decision Process.

C. Transformer Q-Learning

After applying the adapted Q-Learning algorithm and fine-tuning the necessary hyperparameters, the model was evaluated in three environments and compared with it's counterpart with no context in order to properly understand if the presence of a sequence is relevant to the solution of this type of problems.

In this case only reward will be displayed, since standard deviation is directly connected to the context length in the same way as the mean reward. The main point to retain is that the concept of off-policy learning with sequence modeling can be applied and solve problems in RL with great efficiency, as shown in figure 9

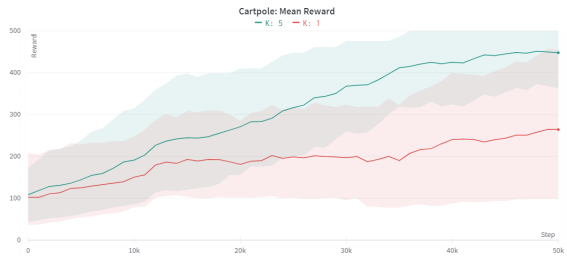


Fig. 9: Cartpole Results for Transformer Q-learning

In the CartPole environment, not only is a solution achieved in 10 thousand training steps, but a maximum score is easily obtained within a small temporal boundary, while retaining sturdiness in each evaluation. This proves that not only is it possible to use Transformers on off-policy learning, but that they are also an efficient method to learn.

With an increased amount of training steps an MLP or a transformer without context would definitely converge to the same result, but that would prove unfeasible in real problems where learning on the go and fast is essential.

In Minigrid, the use of sparse rewards produces similar results, proving that the model can be agnostic to its reward function, as the results in the MiniGrid $7x7$ environment display in figure 10.

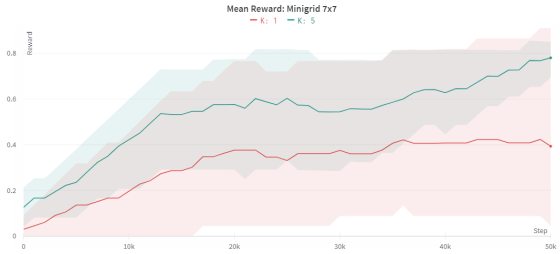


Fig. 10: Minigrid Results for Transformer Q-learning

The gap in learning between a model with and without context is more evident in this setting, where a model with context is able to find the solution to the maze 80% of the time on average, while a model without context is only able to do it 40% of the time. This is what the mean reward means in this case, since a normalized reward of 1 is presented every time the environment is solved.

Finally, in the trading environment, the model begins by losing money or performing in a neutral manner, gradually learning the critical positions of long and short actions in a sequence. Eventually, the context-aware model learns to obtain a peak profit of 129.84 times the original investment, outperforming the value of simply holding the stock until its peak value, which was 30 times its original value, while the model without context only obtained a profit of 115.21 times the original value.

The learning results are displayed in figure 11.

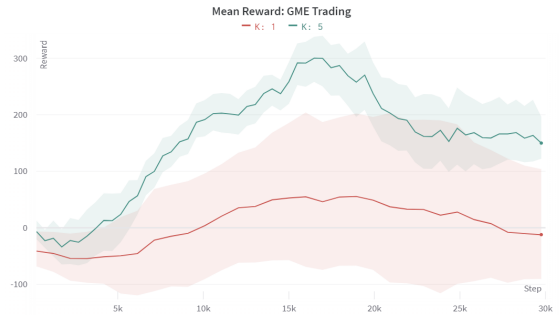


Fig. 11: Gamestop Trading Results for Transformer Q-learning

The average reward may be more unstable than expected, since a single change in one timestep may snowball into a difficulty in obtaining more profit. Less money early would mean less profit later, and that may not easy to interpret from the reward function alone.

In this case the model without context loses money, marking the importance of temporal modeling in financial markets, where understanding the valorization of a certain stock is important.

D. Transformer Proximal Policy Optimization

Using the same evaluation standards as in Transformer Q-Learning, the transformer adaptation of policy proximal optimization bears similar results in the CartPole environment due to its easier reward function.

However, the model is slower than the Q-learning version in solving the environment and does not show as big as of an advantage in using context, in a problem that might be attributed to this model having a single head of prediction at each timestep, while Transformer Q-Learning uses as many heads as its context length to do backpropagation.

Figure 12 shows how this type of algorithm behaves in the CartPole setting.

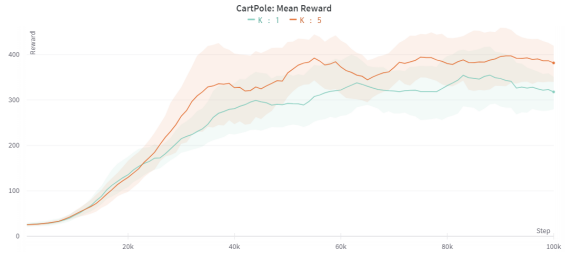


Fig. 12: Cartpole Results for Transformer PPO

When facing a sparse reward function, the on-policy algorithm struggles to learn. Getting a reward becomes a rare situation, since reaching the goal according to random actions is difficult. This becomes apparent in figure 13, where learning is possible even if marked by unstable results.

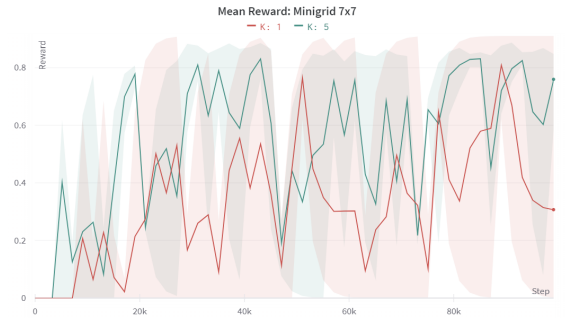


Fig. 13: Minigrid Results for Transformer PPO

Once the goal is reached, the trajectory that was taken cannot be sampled via replay buffer repeatedly, meaning that the model tends to have difficulties getting enough learning signals to fully grasp the goal. This makes the model get stuck unless the unlikely phenomenon of finding the goal multiple times by chance happens.

Because this model uses a more expressive reward function than the previous environments, the results on the anytrading environment become more promising than on Q-learning, making it a good benchmark to evaluate the Transformer PPO adaptation.

Figure 14 shows that the model obtains a slightly higher reward when using context that is not very meaningful, which is similar to the behavior observed in CartPole. However, because the reward function is not linear with profit, this method yielded a maximum profit of 150.21 times the original value, while no context only had a profit of 115.21 times the original value.

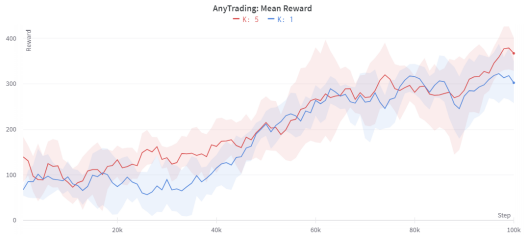


Fig. 14: Anytrading Results for Transformer PPO

V. Discussion

This section presented the results of 171 environment training and evaluation sessions across 6 environments and 3 specific algorithms that were implemented and trained from scratch in order to properly validate the current state of the art and to expand upon it.

An effort to evaluate every design choice and to justify every experiment before its realization was done in order to obey to scientific rigor. Other experiments that did not convey positive results were also developed like the use of ConViT [23], Swin Transformer [24], Relative Positional Encodings [6] and even GANs [25].

It can be considered than 96.5% of the experiments were successful, since the evaluation on PPO with transformers in a sparse setting produced far too unstable results to properly conclude anything.

Validating the Decision Transformer proved that its results are reproducible and that there are still some bottlenecks to tackle in offline reinforcement learning, and that there is still room for improvement when designing the neural networks that employ the methods of the original paper.

Transformer Q-learning and Proximal Policy Optimization adaptations were successful as proof-of-concept applications of popular off-policy and on-policy algorithms in sequence modeling with large room for improvement and potential in the area.

Looking at table I, a comparison between both algorithms show that they are competitive in all environments and that solutions to all problems are always found, even if less than ideal.

TABLE I: Results Comparison of Online Reinforcement Learning with Sequence Modeling

	PPO	DQN
CartPole	485.3	500
Minigrid (solve %)	80	80
Anytrading (profit)	129.84	150.21

VI. Conclusions and Further Investigation

Decision Transformer was used as a starting point to achieve the proposed objectives ion order to understand how sequence modeling and offline reinforcement learning were interconnected. Its objectives were translated to online reinforcement learning and the algorithmic achievements are a great proof-of-concept of what can be achieved in this area.

A. Achievements

With the development of this dissertation, the following objectives were achieved:

- Identification of sequence modeling bottlenecks in the Decision Transformer architecture and their direct connection to the used datasets. Closed datasets are deemed problematic and diversity and exploration are proposed as a solution that allows for sturdy learning.
- Evaluation of different transformer architectural choices in offline reinforcement learning. Gating mechanisms grant added stability thanks to their initialization as a Markovian policy, while learnable sinusoidal positional embeddings greatly improve understanding of long sequences and allow for more flexibility than a fixed table.
- Application of transformers on model-free, off-policy and on-policy reinforcement learning algorithms, making methodologies for sequence modeling in RL available as a starting point to develop further work in the area.

B. Future Work

This body of work would greatly benefit from development in the following directions:

- It is possible that exploration and generalization may have an interconnection with sequence modeling. Improvements in this point may help create agents that are capable of multi-tasking and faster adaptation.
- Pre-trained models may allow for a smaller amount of training and simplify future design choices. Offline pre-training into off-policy training may achieve an efficient combination when tackling sample efficiency.
- More complex environments need to be used to evaluate the proposed algorithms. Due to computational limitations environments like the Atari benchmark or real robotic environments that use actuators were not evaluated.
- To be back-propagated, the Transformer Proximal Proximal Optimization must produce K values and K log-probabilities of the action space. Because we only train on one action and value, the training speed becomes limited.
- Finally world models, sequential versions of Muzero, EfficientZero and other models that were not updated to sequential models may be explored to fully understand whether or not there is also room for improvements in those settings.

REFERENCES

[1] L. Chen, K. Lu, Rajeswaran *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," 06 2021.

[2] H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," 2018. [Online]. Available: https://arxiv.org/abs/1812.02648

[3] R. S. Sutton and A. G. Barton, *Reinforcement Learning: An Introduction*, 2nd ed. MIT press, 2018, iSBN:978-0387303031.

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[6] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *CoRR*, vol. abs/1901.02860, 2019. [Online]. Available: http://arxiv.org/abs/1901.02860

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[8] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," 2015.

[9] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, vol. abs/1507.06527, 2015. [Online]. Available: http://arxiv.org/abs/1507.06527

[10] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[13] K. Cobbe, J. Hilton, O. Klimov, and J. Schulman, "Phasic policy gradient," *CoRR*, vol. abs/2009.04416, 2020.

[14] E. Parisotto, H. F. Song, J. W. Rae *et al.*, "Stabilizing transformers for reinforcement learning," *CoRR*, vol. abs/1910.06764, 2019. [Online]. Available: http://arxiv.org/abs/1910.06764

[15] H. F. Song, A. Abdolmaleki, J. T. Springenberg *et al.*, "V-MPO: on-policy maximum a posteriori policy optimization for discrete and continuous control," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=SylOlp4FvH

[16] J. Schmidhuber, "Reinforcement learning upside down: Don't predict rewards - just map them to actions," *CoRR*, vol. abs/1912.02875, 2019. [Online]. Available: http://arxiv.org/abs/1912.02875

[17] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Advances in Neural Information Processing Systems*, 2021.

[18] M. Reid, Y. Yamada, and S. Gu, "Can wikipedia help offline reinforcement learning?" 01 2022.

[19] A. Banino, A. P. Badia, J. C. Walker *et al.*, "Coberl: Contrastive BERT for reinforcement learning," *CoRR*, vol. abs/2107.05431, 2021. [Online]. Available: https://arxiv.org/abs/2107.05431

[20] Q. Zheng, A. Zhang, and A. Grover, "Online decision transformer," in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 27 042–27 059. [Online]. Available: https://proceedings.mlr.press/v162/zheng22c.html

[21] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[22] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2020.

[23] S. d'Ascoli, H. Touvron, M. L. Leavitt *et al.*, "Convit: Improving vision transformers with soft convolutional inductive biases," *CoRR*, vol. abs/2103.10697, 2021. [Online]. Available: https://arxiv.org/abs/2103.10697

[24] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *CoRR*, vol. abs/2103.14030, 2021. [Online]. Available: https://arxiv.org/abs/2103.14030

[25] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661