SURVEY ON REINFORCEMENT LEARNING VIA. SEQUENCE MODELING USING DECISION TRANSFORMER

FINAL REPORT

Souradip Pal
pal43@purdue.edu

Neelesh Gopalakrishnan
gopalak4@purdue.edu

December 12, 2023

OBJECTIVE

In this project, we propose to study a new paradigm of approaching reinforcement learning methods by viewing them as a sequence modeling problem where the idea is to predict the next best action given a history of the state, action, reward sequences, and the reward expected as the result of that action. Based on the studies of such modeling approaches in both offline and online settings, our goal is to study the cause of the high variances observed in the results of running transformer-based agents especially *Decision Transformer*[1] for sequence prediction in certain environments and compare the variation observed in the online and offline settings for OpenAI Gym [2] locomotion tasks like Hopper, Walker, Half Cheetah, etc. and Atari games like Breakout, Pong and Qbert [3]. We provide further implementation to run '*Online Decision Transformer*' models, especially on Atari games like Breakout, which were not included in the original study proposed in [4] for comparing between its offline counterpart.

1 Motivation

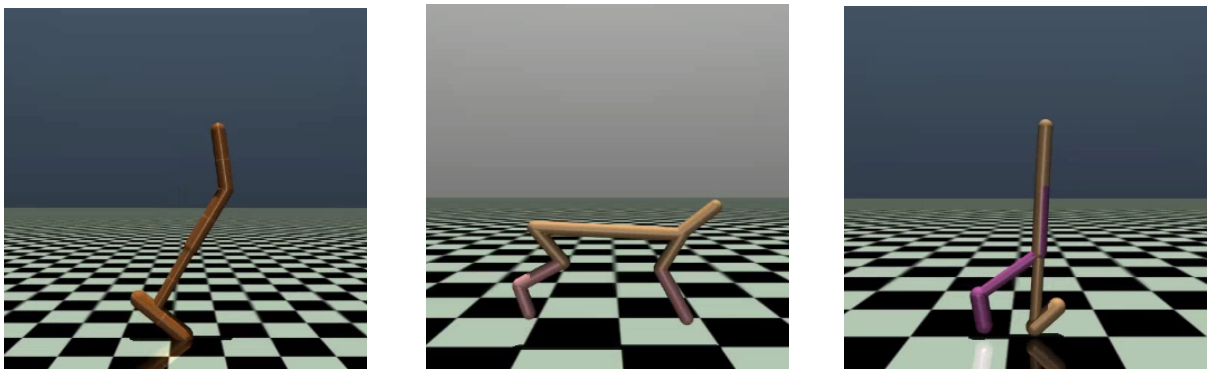
In traditional reinforcement learning tasks, the purpose of training reinforcement learning agents is to find the best policy that will provide the maximum expected future reward. For this reason, most reinforcement learning algorithms follow the idea of estimating some sort of a value function or use policy gradient methods to iteratively find the optimal policy. Here, we move away from the traditional way of viewing the reinforcement learning problem as a problem of maximizing the discounted cumulative rewards and view it more as a sequence modeling problem from which we construct the optimal policy. This idea was first used in the paper '*Offline Reinforcement Learning as One Big Sequence Modeling Problem*'[5] where the distribution of the trajectories was modeled using sequence modeling tools like *Transformers* [6] in an offline setting. This idea was later adopted in the paper '*Decision Transformers*'[1] where given an expected reward also known as *Return-to-Go* and the history of states, actions, and rewards, the model predicts the best action which gives the reward closest to what was expected. In order to model such long sequences, the authors proposed the use of transformer architectures which are more suitable for long-term credit assignment due to the self-attention mechanism. Also, due to this type of modeling, the algorithms do not depend on the discount factor which plays a crucial role in the stabilization of traditional RL algorithms. Later, the algorithm was extended to the online settings in the paper '*Online Decision Transformer*'[4] where the pre-trained models are fine-tuned based on task-specific interaction with the environment. However, these models suffer from high levels of variance when run in certain environments like Walker2D, Breakout, Qbert, etc. Here, we want to find the potential of leveraging such transformer-based deep learning models in these methods and fine-tune them to improve on the variances of the returns acquired.

2 Background

With the success of Transformer architectures in sequence modeling problems especially in natural language processing, there has been an increasing trend to use those techniques in Offline Reinforcement Learning wherein an agent is trained to autoregressively maximize the likelihood of trajectories in the offline dataset. Thus it is natural to view RL

as a supervised learning problem while using state-of-the-art Transformer architectures to model distributions over sequences of states, actions, and rewards. This modeling approach significantly reduces the range of design decisions and provides flexibility to capture long-horizon dynamics. Moreover, using online fine-tuning, which involves data acquisition via exploration, the idea is to improve the policies even further. To discuss these ideas in detail we introduce some basic terminologies related to the discussion below:

- *Transformers*: Introduced by Vaswani et al. [6], it is a powerful technique to efficiently model sequential data using a self-attention mechanism along with residual connections that allow assignment of "credit" by implicitly forming associations via similarity of the query and key vectors.
- *GPT*: GPT(Generative Pre-trained Transformers) [7] is a modification over the Transformers architecture with a causal self-attention mask to enable autoregressive generation.
- *Offline RL*: In offline reinforcement learning [8], instead of obtaining data via environment interactions, we only have access to some fixed limited dataset consisting of trajectory rollouts based on some arbitrary policies. This setting does not allow agents to explore the environment and collect additional feedback.
- *Offline RL with Online Fine-tuning*: Proposed by Nair et al. [9], it is a simple and effective framework that enables us to leverage large amounts of offline data and then quickly perform online fine-tuning of RL policies.



(a) Hopper (b) Half Cheetah (c) Walker2D

Figure 1: Open AI Gym Environments

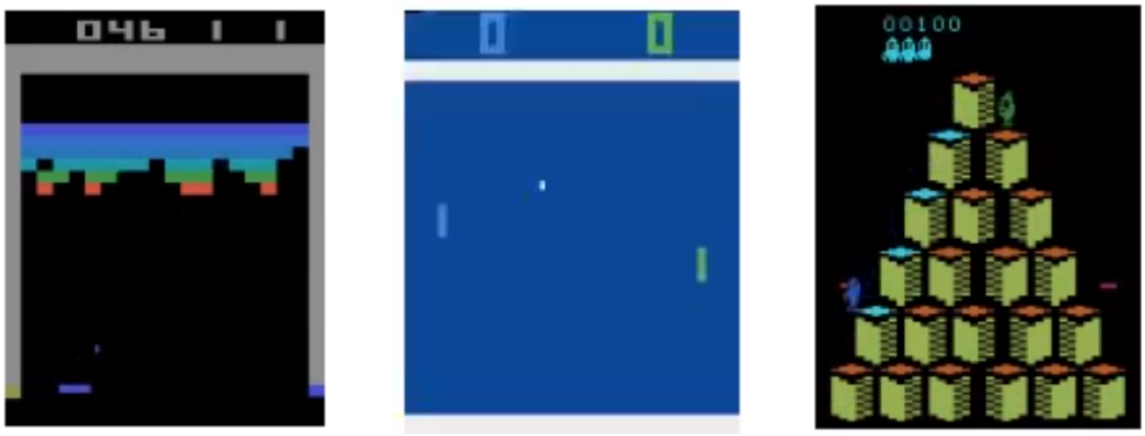
2.1 Literature Review

The following gives a brief overview of some of the related works that have led to the core idea of viewing RL as a sequence modeling problem.

Upside Down Reinforcement Learning (UDRL) The paper '*Reinforcement Learning Upside Down: Don't Predict Rewards - Just Map Them to Actions*' [10] introduces an unconventional approach that inverts traditional RL by using rewards as inputs rather than outcomes to be predicted. In UDRL, rewards, along with time horizons and other functions of historical and desired future data, are treated as task-defining commands. The system learns to map these commands to actions or action probabilities, using supervised learning techniques on past experiences.

Offline RL as One Big Sequence Modeling Problem The paper '*Offline Reinforcement Learning as One Big Sequence Modeling Problem*' [5] explores the application of sequence modeling techniques, specifically using Transformer architectures, to solve reinforcement learning (RL) problems. Here the RL problem is framed as a generic sequence modeling problem, where the objective is to produce a sequence of actions leading to a sequence of high cumulative rewards. By leveraging the Transformer architecture to model distributions over trajectories, the authors used a technique called beam search for model planning. Considering, a trajectory τ consisting of N-dimensional states, M-dimensional actions, and scalar rewards: $\tau = \{s_t^0, s_t^1, \cdots, s_t^{N-1}, a_t^0, a_t^1, \cdots, a_t^{M-1}, r_t\}_{t=0}^{T-1}$, viewed as tokens, the approach is to maximize a joint likelihood function of the induced conditional probabilities (P_θ) over the tokens in the trajectory given as $\mathcal{L}(\bar{\tau}) = \sum_{t=0}^{T-1} \left(\sum_{i=0}^{N-1} \log P_\theta(\bar{s}_t^i | \bar{s}_t^{<i}, \bar{\tau}_{<t}) + \sum_{j=0}^{M-1} \log P_\theta(\bar{a}_t^j | \bar{a}_t^{<j}, \bar{\tau}_{<t}) + \log P_\theta(\bar{r}_t | \bar{a}_t, \bar{s}_t, \bar{\tau}_{<t}) \right)$ where θ is the transformer parameters and $\bar{\tau}_{<t}$ is the shorthand for a tokenized trajectory from timesteps 0 through $t-1$.

Reward-Conditioned Policies This paper '*Reward-Conditioned Policies*' [11] proposes an innovative approach to bridge the gap between reinforcement learning and supervised learning. Traditional RL algorithms, while powerful, often suffer from instability, sensitivity to hyper-parameters, and a need for extensive data samples. On the other hand, Supervised Learning which is similar to imitation learning, requires near-optimal expert data, which is not always feasible to obtain. In this method, non-expert trajectories from sub-optimal policies are considered as optimal supervision, not for maximizing reward, but for achieving the reward level of the given trajectory. By conditioning the policy on the numerical value of the reward, it aims to create a policy that can generalize to higher returns.



(a) Breakout (b) Pong (c) Qbert

Figure 2: Atari Games

2.2 Decision Transformer

In the paper '*Decision Transformer*' [1], the idea is to use a model to predict the best action to maximize the *future* desired returns rather than the past returns. Hence instead of feeding the immediate rewards to the model, it is fed with the returns-to-go $\hat{R}_t = \sum_{t'=0}^{t} r'_t$ considering a Markov decision process(MDP) formulation, described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$, $r_t = \mathcal{R}(s_t, a_t)$ are the state, action and reward at timestep t respectively and the transition probabilities are defined as $\mathcal{P}(s'|s, a)$ and the reward function as $r = \mathcal{R}(s, a)$. This leads to the following trajectory representation $\tau = \{\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, ..., \hat{R}_T, s_T, a_T\}$, in contrast to the original trajectory $\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T, a_T, r_T\}$, which is then used in the training and generation of the optimal actions autoregressively. Initially, the agent starts from a starting state and a desired total return-to-go, and in each step after taking the action the return-to-go is reduced by the current reward and the process continues until the episode terminates.

```
# main model
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t)  # per-timestep (note: not per-token)
    s_embedding = embed_s(s) + pos_embedding
    a_embedding = embed_a(a) + pos_embedding
    R_embedding = embed_R(R) + pos_embedding

    # interleave tokens as (R_1, s_1, a_1, ..., R_K, s_K)
    input_embeds = stack(R_embedding, s_embedding, a_embedding)

    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions

    # predict action
    return pred_a(a_hidden)
```

Figure 3: Decision Transformer Model (Chen et al.)

The core idea of the approach relies on the use of Transformer architectures for predicting the next best action. Here, the trajectory information of the last K timesteps is fed into the Decision Transformer model having $3K$ tokens (Return-to-go, state, action) for each timestep. After encoding the inputs using Linear layers, the embedded tokens are then processed by a GPT [7] model, which predicts future action tokens via autoregressive modeling (Fig. 3). This algorithm was evaluated using Offline RL benchmarks on OpenAI Gym environments like Hopper, Walker, Half Cheetah, and Reacher, Atari games like Breakout, Pong, Qbert, and Seaquest, and some Key-to-Door tasks.

Algorithm 1: Online Decision Transformer

Input: offline data $\mathcal{T}_{\text{offline}}$, rounds R, exploration RTG g_{online}, buffer size N, gradient iterations I, pretrained policy π_θ
Initialization: Replay buffer $\mathcal{T}_{\text{replay}} \leftarrow$ top N trajectories in $\mathcal{T}_{\text{offline}}$.
for *round* $= 1, \ldots, R$ **do**
  // use randomly sampled actions
  Trajectory $\tau \leftarrow$ Rollout using $\mathcal{M}$ and $\pi_\theta(\cdot|\mathbf{s}, \mathbf{g}(g_{\text{online}}))$.
  $\mathcal{T}_{\text{replay}} \leftarrow \{\mathcal{T}_{\text{replay}} \backslash \{\text{the oldest trajectory}\}\} \bigcup \{\tau\}$.
  $\pi_\theta \leftarrow$ Finetune ODT on $\mathcal{T}_{\text{replay}}$ for I iterations via Algorithm 2.

(a) Algorithm 1

Algorithm 2: ODT Training

Input: model parameters θ, replay buffer $\mathcal{T}_{\text{replay}}$, training iterations I, context length K, batch size B
Compute the trajectory sampling probability $p(\tau) = |\tau| / \sum_{\tau \in \mathcal{T}} |\tau|$.
for $t = 1, \ldots, I$ **do**
  Sample B trajectories out of $\mathcal{T}_{\text{replay}}$ according to p.
  for *each sampled trajectory* τ **do**
    // Hindsight Return Relabeling
    $\mathbf{g} \leftarrow$ the RTG sequence computed by the true rewards: $\mathbf{g}_t = \sum_{j=t}^{|\tau|} r_j,\ 1 \leqslant t \leqslant |\tau|$.
    $(\mathbf{a}, \mathbf{s}, \mathbf{g}) \leftarrow$ a length K sub-trajectory uniformly sampled from τ.
  $\theta \leftarrow$ one gradient update using the sampled $\{(\mathbf{a}, \mathbf{s}, \mathbf{g})\}$s.

(b) Algorithm 2

Figure 4: ODT Algorithm (Zheng et al.)

2.3 Online Decision Transformer

The paper '*Online Decision Transformers*'[4] approaches the reinforcement learning problem by merging offline and online learning methods. In offline RL, an agent learns from a fixed dataset in a supervised manner, but this limits the policy's quality to the dataset's quality and necessitates the need to finetune the model based on the task of interest via online interactions. Online Decision Transformer (ODT) overcomes this by incorporating online fine-tuning, where the agent adapts through direct environment interactions, enabling it to gather new data through exploration.

ODT's innovations include shifting from deterministic to stochastic policies for improved exploration and creating a new replay buffer that stores data from online interactions, aiding in continuous learning. To maximize the log-likelihood of the trajectories over the training dataset defined as $J(\theta) = \frac{1}{K}\mathbb{E}_{(a,s,g)\sim\mathcal{T}}[-\log(\pi_\theta(a|s,g)]$, the paper takes a probabilistic approach to learn a stochastic policy by modeling the action distributions conditioned on states($s_{-K,t}$) and Return-to-go's($g_{-K,t}$) as a multivariate Gaussian distribution with a diagonal covariance matrix i.e. $\pi_\theta(a_t|s_{-K,t}, g_{-K,t}) = \mathcal{N}(\mu_\theta(s_{-K,t}, g_{-K,t}), \Sigma_\theta(s_{-K,t}, g_{-K,t}))$ where θ is the policy parameters. ODT enhances the efficiency of policy optimization, especially where environment interactions are expensive or limited by employing sample-efficient online fine-tuning. This is achieved by maximizing the log-likelihood under a policy entropy constraint defined as $H_\theta^{\mathcal{T}}(a|s,g) \geq \beta$ where $H_\theta^{\mathcal{T}}(a|s,g) = \frac{1}{K}\mathbb{E}_{(s,g)\sim\mathcal{T}}[H[\pi_\theta(a|s,g)]]$ and β is a hyper-parameter, to balance the exploration vs exploitation trade-off. Hence the objective function to be minimized is represented as $\mathbb{E}_{(a,s,g)\sim\mathcal{T}}[-\log(\pi_\theta(a|s,g)] - \lambda\mathbb{E}_{(s,g)\sim\mathcal{T}}[H[\pi_\theta(\cdot|s,g)]]$ where the first term can be considered as the NLL loss and the second term can be considered as the cross entropy loss. It also employs a method called Hindsight Return Relabeling to relabel the agent's trajectories with the achieved goal, as opposed to the intended goal. If the return achieved during a policy rollout and the induced Return-to-Go(RTG) differs from the intended Return-to-Go, then the RTG token for the rolled-out trajectory is relabeled with the achieved returns, such that the RTG token at the last timestep is exactly the reward obtained by the agent. This is how it combines offline pre-training with online adaptation, showing not only competitive performance with top algorithms but also notable improvements in the fine-tuning phase for applications needing both pre-training and ongoing interaction-based learning. This algorithm has shown state-of-the-art results on OpenAI Gym environments like Hopper, Walker, Half Cheetah, and Ant and also on Ant-Maze tasks.

3 Proposed Methodology

3.1 Datasets

Offline Settings For offline learning, the D4RL benchmark dataset for OpenAI Gym environments like Hopper, Walker, and Half Cheetah was downloaded using the D4RL framework [1]. Each dataset has three variations Medium,

[1] https://github.com/Farama-Foundation/D4RL

Medium-Replay, and Medium Expert. Currently, in our experiments, we are using Medium and Medium-Expert datasets. The Medium dataset contains one million timesteps generated from a policy trained to about one-third the efficiency of an expert policy. In contrast, the Medium-Replay dataset comprises data from the replay buffer of a policy trained up to the performance level of a medium agent.

In the case of Atari games, we have downloaded the DQN Replay dataset using the D4RL Atari framework[2]. Currently, we are using a mixed dataset with frame stacking (4 frames) which consists of 1 million 84×84 image samples from games like Breakout, Qbert, and Pong.

Online Settings The paper [4] utilizes the same datasets from the D4RL benchmark as in the case of the offline paper, specifically focusing on two types of tasks. The first type includes Gym locomotion tasks: Hopper, Walker, Half Cheetah, and Ant. These are characterized by dense reward environments. For evaluation, the paper uses the Medium and Medium-Replay datasets from these environments. The second type of task considered in the paper is the goal-reaching task, AntMaze. In this task, an Ant robot is tasked with reaching a target location, and the rewards are sparse: the agent receives a reward of 1 for reaching the goal, and 0 otherwise. The datasets used here are the *antmaze-umaze* and *antmaze-umaze-diverse*. For our current experiment, we are specifically using the Hopper, Walker, and Half Cheetah Medium dataset from the D4RL benchmark for pre-training and then utilizing the environments to generate observations from trajectories during online fine-tuning. In the case of Atari games, we have used the same DQN Replay dataset as in the offline method.

3.2 Model Specification & Experiments

In this project, we investigate the effect of certain hyper-parameters(e.g. context length) of the Decision Transformer model on the variance observed in the returns averaged over several episodes. Also, we study the differences in the variance observed in online vs offline settings. To achieve this goal we plan to conduct several experiments with a GPT2-based Transformer model as our backbone on a subset of data extracted from the aforementioned datasets.

3.2.1 Decision Transformer

We run the currently available implementation of the paper '*Decision Transformer*'[1] on the OpenAI Gym[2] environments and Atari games. The implementation is available on GitHub[3]. Here, we were able to run some preliminary evaluations using the available pre-trained models based on GPT2 architecture. Moreover, we modify the hyper-parameters mainly the context length of the GPT2 [12] model, and compare them with the existing results to observe the effect of context length in the variances obtained in the accuracy results and map those correlations with specific environments. The detailed changes in the implementation can be accessed via. the GitHub link [4].

Gym Experiments: For this experiment, we collect offline datasets to run the RL algorithm on standard locomotion environments (Half Cheetah, Hopper, and Walker2D) provided by OpenAI Gym as part of the D4RL benchmark [13]. These datasets come with different variations e.g. Medium, Medium-Replay, and Medium-Expert. Firstly we run evaluations on existing pre-trained Decision Transformer models available via the HuggingFace library [5]. Then we train different models by varying the context length hyper-parameter ($K = 10, 20, 30$) of the GPT2 model and observe the results for each of the Gym environments mentioned above. Linear layers are used to first encode the state, action, and returns-to-go into tokens which are then passed to the GPT2 model for inference. Table 2 shows the essential hyper-parameters involved.

Atari Experiments: We collect offline datasets to run the RL algorithm on Atari games (Breakout, Pong, and Qbert) which can be accessed via. the DQN Replay dataset provided by Google [8]. These datasets come with different variations as well e.g. Mixed, Medium, and Expert with or without frame stacking. Firstly we run evaluations on existing pre-trained Decision Transformer models available via the HuggingFace library. Since the DQN Replay dataset consists of grayscale images of dimension 84×84, we pass those images through a convolutional encoder model having $32, 64$ and 64 channels with filter sizes of $\{8 \times 8, 4 \times 4, 3 \times 3\}$ and strides of $\{4, 2, 1\}$ respectively to get the state embeddings. Linear embeddings are used to encode the action and rewards. These embeddings are then fed into the GPT2 model as tokens. Table 3 shows the other essential hyper-parameters involved.

[2]https://github.com/takuseno/d4rl-atari

[3]https://github.com/kzl/decision-transformer

[4]https://github.com/souradipp76/decision-transformer/pull/1

[5]https://huggingface.co/blog/decision-transformers

Table 1: Essential Hyper-parameters for Decision Transformer

Table 2: OpenAI Gym Experiments

Hyperparameter	Value
Number of layers	3
Number of attention heads	1
Embedding dimension	128
Context Length K	20
Return-to-Go conditioning	3600 - Hopper 5000 - Walker 12000 - Half Cheetah
Nonlinearity	ReLU
Learning Rate	10^{-4}
Batch Size	8

Table 3: Atari Experiments

Hyperparameter	Value
Number of layers	6
Number of attention heads	8
Embedding dimension	128
Context length K	30
Return-to-Go conditioning	90 - Breakout 14000 - Qbert 20 - Pong
Nonlinearity	ReLU
Max Timesteps	2654 - Breakout 3901 - Qbert 4701 - Pong
Warmup tokens	512×20
Final tokens	$2 \times 500000 \times K$
Learning rate	6×10^{-4}
Batch Size	8

3.2.2 Online Decision Transformer

Here, we run the currently available implementation of the paper '*Online Decision Transformer*'[4] on the previously mentioned OpenAI Gym [2] environments. The implementation is available on GitHub[6]. Here we modify the hyper-parameters mainly the context length of the GPT2 model and compare them with the existing results to observe the effect of context length in the variances obtained in the results in the case of online fine-tuning. In this experiment, instead of using the hyper-parameters presented in the paper [4], we resort to the hyperparameters used in the original Decision Transformer [1] paper so that the comparison is consistent with the model architecture. We also check the effect of the algorithm with and without offline pretraining to understand how the algorithm behaves when run in an online way.

Gym Experiments: In this experiment, we use the same offline D4RL benchmark datasets mentioned in the previous section for offline pre-training. The pre-training phase consists of 5 training epochs followed by 100 iterations of online fine-tuning. Here we use a replay buffer size of 1000 to store the sampled trajectories via exploration. The essential hyper-parameters are shown in Table 4.

Table 4: Essential Hyper-parameters of Online Decision Transformer for Gym Experiments

Table 5: Offline Pretraining

Hyperparameter	Value
Number of layers	3
Number of attention heads	1
Embedding dimension	128
Context length K	20
Return-to-Go conditioning	7200 - Hopper 10000 - Walker 12000 - Half Cheetah
Nonlinearity	ReLU
Learning rate	10^{-4}
Batch Size	8

Table 6: Online Finetuning

Hyperparameter	Value
Number of layers	3
Number of attention heads	1
Embedding dimension	128
Context length K	20
Updates between rollouts	300
Return-to-Go conditioning	3600 - Hopper 5000 - Walker 6000 - Half Cheetah
Nonlinearity	ReLU
Target Entropy β	$-dim(\mathcal{A})$
Batch Size	8

[6] https://github.com/facebookresearch/online-dt

3.3 Extending Online Decision Transformers for Atari Games

The paper '*Online Decision Transformer*' [4] only focuses their experiments on locomotion tasks-based environments like Hopper, Walker2D, and Half Cheetah. Thus, in order to draw parallels between the observed returns from the Decision Transformer in the offline and online settings, we modify its available implementation to run the algorithm on Atari games like Breakout, Pong, and Qbert. Moreover, we try to gain some insights by training and experimenting with the algorithms on those environments and observing the variance of the rewards averaged over several episodes. Similar to the experiments in the offline case, we use the DQN Replay dataset for the offline pre-training phase and successively perform online fine-tuning. Here, we modify the architecture to accommodate for the 84×84 grayscale images with frame stacking of 4 before feeding it to the transformer network. Since ODT uses stochastic policies we convert the discrete action space of Atari games to one-hot encodings. Also, to pre-train the model we select trajectories having the top $k(= 0.2\times$ Total number of trajectories) RTG so that it is much faster and more efficient. The other hypermeters are kept to be same as the offline settings as shown in Table 7. The detailed changes can be viewed via. this GitHub link[7].

```
# Load DQN replay dataset
import d4rl_atari
import gym

env = gym.make(env_name, stack=True) # -v{0, 1, 2, 3, 4} for datasets with the other random seeds

# dataset will be automatically downloaded into ~/.d4rl/datasets/[GAME]/[INDEX]/[EPOCH]
dataset = env.get_dataset()

...

## Change state embedding layer
self.embed_state = nn.Sequential(nn.Conv2d(4, 32, 8, stride=4, padding=0), nn.ReLU(),
                        nn.Conv2d(32, 64, 4, stride=2, padding=0), nn.ReLU(),
                        nn.Conv2d(64, 64, 3, stride=1, padding=0), nn.ReLU(),
                        nn.Flatten(), nn.Linear(3136, hidden_size), nn.Tanh())
```

Table 7: Essential Hyper-parameters of Online Decision Transformer for Atari Experiments

Table 8: Offline Pretraining

Hyperparameter	Value
Number of layers	6
Number of attention heads	8
Embedding dimension	128
Context length (K)	20
Return-to-Go conditioning	180 - Breakout 40 - Pong 28000 - Qbert
Nonlinearity	ReLU
Learning rate	10^{-4}
Batch Size	8

Table 9: Online Finetuning

Hyperparameter	Value
Number of layers	6
Number of attention heads	8
Embedding dimension	128
Context length (K)	20
Updates between rollouts	300
Return-to-Go conditioning	90 - Breakout 20 - Pong 14000 - Qbert
Nonlinearity	ReLU
Target Entropy β	$-dim(\mathcal{A})$
Batch Size	8

4 Results & Discussion

4.1 Pretrained Model Evaluation

As a preliminary step, we evaluated already existing pre-trained GPT2 transformer models trained on the Medium and Expert datasets of Half Cheetah, Hopper, and Walker2D OpenAI Gym environments. The hyper-parameters of the model are specified in Table 2. Also, we evaluated the existing pre-trained GPT2 transformer models trained on the Mixed dataset of the Breakout Atari game. The hyper-parameters of the model are specified in Table 3. Table 10 and 11 shows the mean and standard deviation of the return and episode length obtained for the corresponding evaluation

[7] https://github.com/souradipp76/online-dt/pull/1

experiments using those pre-trained models. The normalized scores with 100 representing an expert policy as per [13] for Gym environments and [1] for Atari games are also included in both cases. It is worth noticing in the case of OpenAI Gym environments the models trained on Expert datasets are far better and much more stable in terms of the standard deviation obtained e.g. the return std. value for Hopper trained on the expert dataset is ~ 56 whereas the value in the case of the medium dataset is ~ 107. It is also to be kept in mind that even if return std. value is higher in the case of Half Cheetah and Qbert, the return mean is far higher. In general, it can be inferred that the variance is higher if the return expected is higher for a particular type of dataset used.

Table 10: Pretrained Model Evaluation on OpenAI Gym Environments

Environment	Model	Target	Return Mean	Return Std.	Length Mean	Length Std.	Normalized Score
Hopper	Medium	3600	1708.18	107.02	540	30.51	52.81
Hopper	Expert	3600	3629.12	55.83	996	8.0	112.2
Walker	Medium	5000	2947.68	792.11	818.8	206.52	64.18
Walker	Expert	5000	4009.89	50.15	1000	0.0	87.31
Half Cheetah	Medium	12000	5064.32	55.29	1000	0.0	41.73
Half Cheetah	Expert	12000	10875.49	350.72	1000	0.0	89.62

Table 11: Pretrained Model Evaluation on Atari

Game	Model	Target	Return Mean	Return Std.	Length Mean	Length Std.	Normalized Score
Breakout	Mixed	90	41.8	16.48	1103.8	172.28	139.33
Qbert	Mixed	14000	5015	4079.48	816.8	283.29	37.27
Pong	Mixed	20	4	1.17	875.4	45.25	26.7

(a) Mean of Training Losses vs Number of Epochs(Steps)

(b) Standard Deviation of Training Losses vs Number of Epochs(Steps)

(c) Action Error vs Number of Epochs(Steps)

Figure 5: Training Decision Transformer on OpenAI Gym Environments

4.2 Training Comparison

Offline Training Here, we perform training and evaluation of the Decision Transformer model with GPT2 architecture using the D4RL benchmark datasets on OpenAI Gym environments like Hopper, Walker2D, and Half Cheetah. We ran the training for a maximum of 5 epochs with batch size 8 and averaged the returns over 100 evaluation episodes. The trained models are evaluated under two evaluation conditioning targets as shown in Table 12. Here we consider a maximum episode length of 1000. Fig. 5 shows the plot of training error vs the number of epochs averaged over 3 runs. From Table 12, it can be seen the results obtained while evaluating the models after training follow a similar pattern as those obtained from the pre-trained model evaluations. This is evident from the high standard deviation seen in the case of Walker2D agent training on the Medium dataset same as before. Overall, the values are quite close to the results from the pre-trained models.

Online Training According to our experimental plan, we also ran the available implementation of the '*Online Decision Transformer*' [4] and evaluated the model using the D4RL benchmark datasets on OpenAI Gym environments for Hopper, Walker2D, and Half Cheetah. Fig. 6 shows the plots of the variation of training loss (mean and standard deviation) along with the number of epochs. Furthermore, we trained the model for Atari environments as well

Table 12: Decision Transformer Training on OpenAI Gym Environments

Environment	Dataset	Target	Return Mean	Return Std.	Length Mean	Length Std.
Hopper	Medium	1800	1803.40	190.35	568.28	56.37
		3600	1866.73	220.01	583.25	66.19
Walker	Medium	2500	1945.94	1200.23	806.7	288.67
		5000	2832.92	1433.73	554.66	361.33
Half Cheetah	Medium	6000	4839.06	652.13	1000.0	0.0
		12000	4908.62	430.49	1000.0	0.0

e.g. Breakout, Pong, and Qbert by using the modified implementation of the ODT algorithm. Table 14 shows the corresponding mean and standard deviation of the return obtained in Breakout, Pong and Qbert respectively along with their normalized scores. The results obtained in this case was not optimal due to the fact that limited training was performed only using a subset of trajectories.

Table 13: Online Decision Transformer Training on OpenAI Gym Environments

Environment	Dataset	Phase	Target	Return Mean	Return Std.	Length Mean	Length Std.
Hopper	Medium	Offline Pretrain	7200	1903.03	179.10	609.5	52.11
		Online Finetune	3600	1998.04	289.01	641.2	84.64
Walker	Medium	Offline Pretrain	10000	2630.36	479.95	1000	0.0
		Online Finetune	5000	3335.59	216.94	985.1	44.69
Half Cheetah	Medium	Offline Pretrain	120000	4184.64	1383.92	1000.0	0.0
		Online Finetune	6000	5026.85	212.92	1000.0	0.0

(a) Mean of Training Losses vs Number of Epochs(Steps)

(b) Standard Deviation of Training Losses vs Number of Epochs(Steps)

(c) NLL Loss vs Number of Epochs(Steps)

Figure 6: Training Online Decision Transformer on OpenAI Gym Environments

Table 14: Online Decision Transformer Training on Atari Games

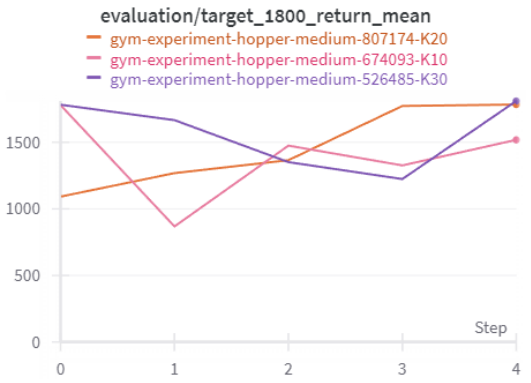
Game	Model	Target	Return Mean	Return Std.	Length Mean	Length Std.	Normalized Score
Breakout	Mixed	90	7.2	16.48	84.37	84.37	24
Qbert	Mixed	14000	367.5	143.63	407	72.59	2.73
Pong	Mixed	20	0.6	1.64	916.1	42.86	4

4.3 Effect of Context Length

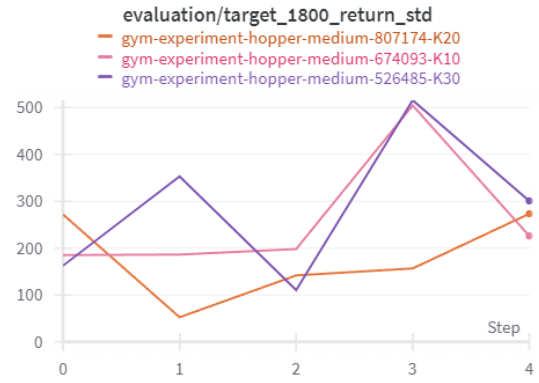
Context length is a crucial hyper-parameter whenever Transformers come into play in most of the sequential modeling problems. To understand the effect of context length on the observed variances in the cumulative returns, we trained multiple Decision Transformer models with different context lengths K e.g. 10, 20, and 30. Fig. 7, 8 and 9 shows the variation of mean and standard deviation of the returns obtained during evaluation under two sets of evaluation targets for context length $K = 10, 20, 30$ in case of Gym environments. It can be noticed that in the case of the

Walker2D environment, the standard deviation observed is quite high (~ 1000) even though the actual average return is low(~ 2000) in comparison to other Gym environments for both the higher and lower evaluation return targets. Further, for $K = 20$ and $K = 30$, the mean return is generally higher in all three environments than for $K = 10$. Thus it can be understood that a higher context length would generally lead to a higher return. Also, it can be seen that for $K = 20$, the standard deviation is generally lower for higher evaluation targets than for $K = 10$ and $K = 30$. However, no general correlation is visible in the values of return variance with respect to the context length over the different environments.

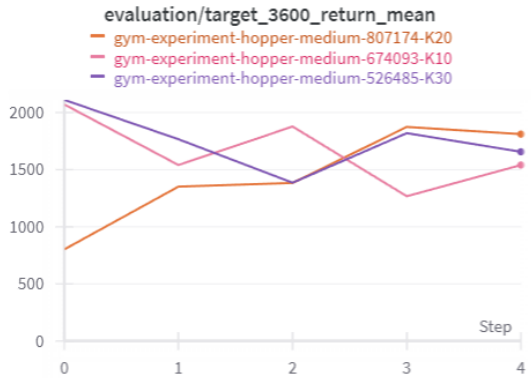
If we look at the Fig. 10, 11 and 12 for the ODT case, the average return remains almost constant after the pretraining phase. Unfortunately, the values of the standard deviation are quite erratic without showing any useful trend over the different environments.



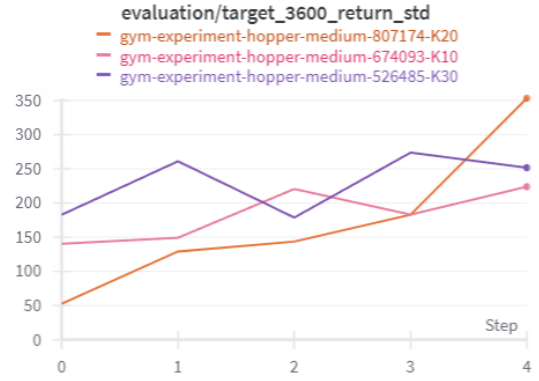
(a) Return Mean vs Number of Epochs(Steps) for RTG target 1800



(b) Return Std. vs Number of Epochs(Steps) for RTG target 1800



(c) Return Mean vs Number of Epochs(Steps) for RTG target 3600

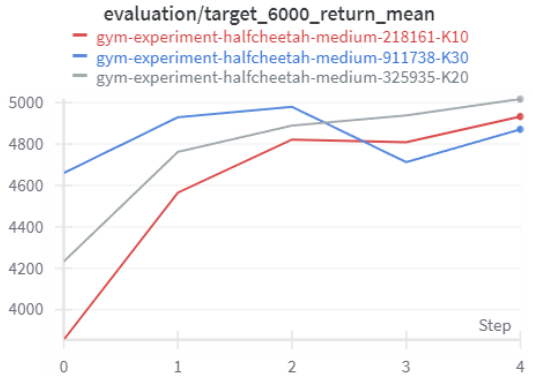


(d) Return Std. vs Number of Epochs(Steps) for RTG target 3600

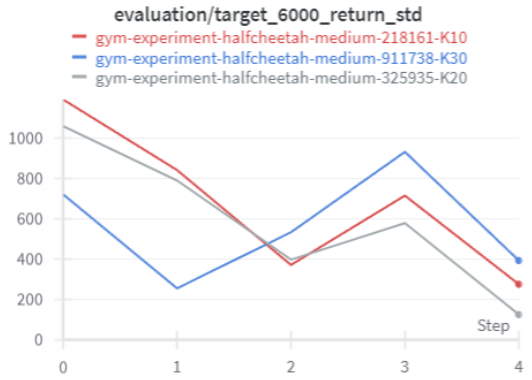
Figure 7: Variation of Return Mean and Std. for DT with context length $K = 10, 20, 30$ in Hopper

4.4 Offline vs Online

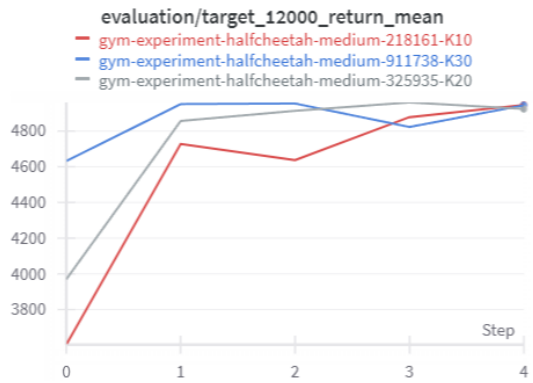
From the Tables 12 and 13, it can be seen that with online fine-tuning, the standard deviation obtained is much more controlled in comparison to the offline decision transformer. This lower standard deviation can be attributed to the usage of stochastic policy instead of fully deterministic policy. The inclusion of a lower bound on the policy entropy while training encourages exploration and thus serves as a regularization factor for the loss function. It is to be noticed that the return variances are much more controlled right from the pre-training phase. Thus it is evident that putting a lower bound on the policy entropy is essential in achieving consistent returns over several episodes. Moreover, we show the variation of the return with respect to the number of samples generated and it shows an obvious increasing trend in the initial stages and remains constant thereafter. A steep initial slope indicates rapid early learning which is a sign of an effective exploration strategy, followed by exploitation as the agent begins to saturate its learning capacity.



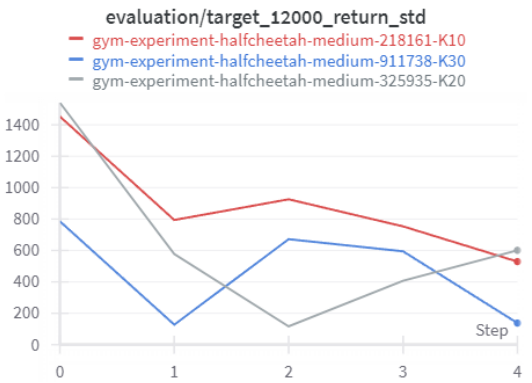
(a) Return Mean vs Number of Epochs(Steps) for RTG target 6000



(b) Return Std. vs Number of Epochs(Steps) for RTG target 6000



(c) Return Mean vs Number of Epochs(Steps) for RTG target 12000



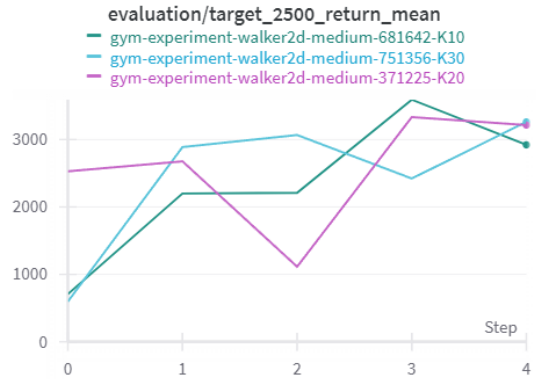
(d) Return Std. vs Number of Epochs(Steps) for RTG target 12000

Figure 8: Variation of Return Mean and Std. for DT with context length $K = 10, 20, 30$ in Half Cheetah

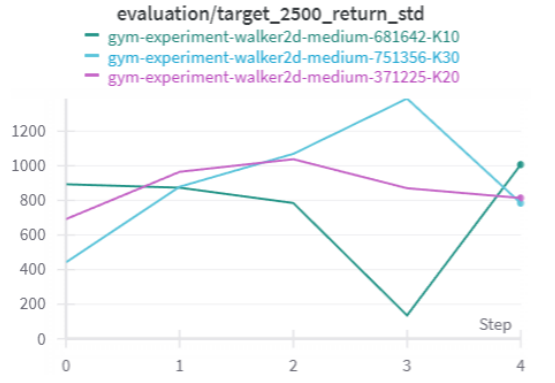
5 Conclusion & Future Work

In this project, our primary goal of understanding how transformer architectures are being leveraged to approach a general RL problem has been achieved. This included understanding the code, setting up the environment, and gathering all the data and other components to run the experiments using Decision Transformer in both online and offline settings. Moreover, the initial experimental results showed that there is indeed a high variance in the returns observed even with the pre-trained models as can be seen from Table 10. The standard deviation in the case of the Walker2D agent trained on the Medium dataset is quite high in particular. Through online fine-tuning, this variance was observed to be reduced significantly and training was observed to be much more stable. Although there wasn't any significant improvement in the returns obtained in the online fine-tuning phase, the agents were much more stable and reliable. Hence online fine-tuning proves to be a good way to improve the return variance over the different environments used.

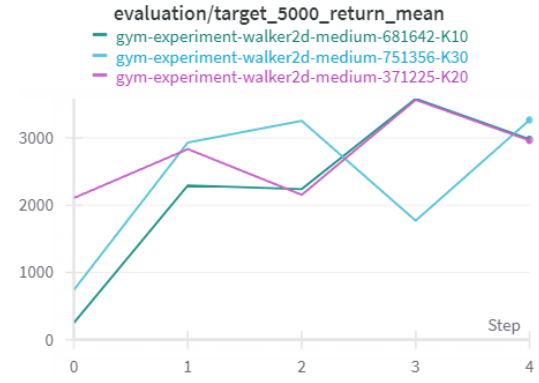
Due to the complexity and scale of the datasets used it takes several GPU hours to train such models. Thus with limited time and GPU resources, we resorted only to training multiple '*Online Decision Transformer*' models in OpenAI Gym environments. For Atari games, even though we modified the implementation to make it as efficient as possible it took more than the intended GPU hours of training to get any viable results. In the future, this study can be continued even further by conducting more experiments with the ODT algorithm on Atari games and inspecting the variance by varying the context length. Ablation studies can also be done by changing other hyper-parameters like the embedding dimension or the non-linear activation function used. Since we mainly used the GPT2 architecture in our project, investigating the effects of a different transformer architecture like BERT is also a possible direction of extending the current work future. In essence, these analyses will reveal the huge potential that Transformers hold in effectively modeling more general and complex RL problems.



(a) Return Mean vs Number of Epochs(Steps) for RTG target 2500



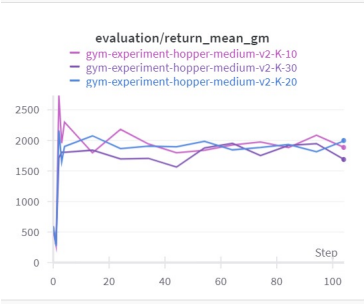
(b) Return Std. vs Number of Epochs(Steps) for RTG target 2500



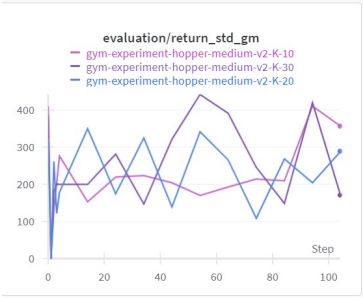
(c) Return Mean vs Number of Epochs(Steps) for RTG target 5000

(d) Return Std. vs Number of Epochs(Steps) for RTG target 5000

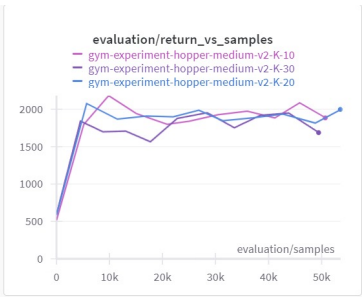
Figure 9: Variation of Return Mean and Std. for DT with context length $K = 10, 20, 30$ in Walker2D



(a) Return Mean vs Number of Epochs(Steps) for RTG target 3600

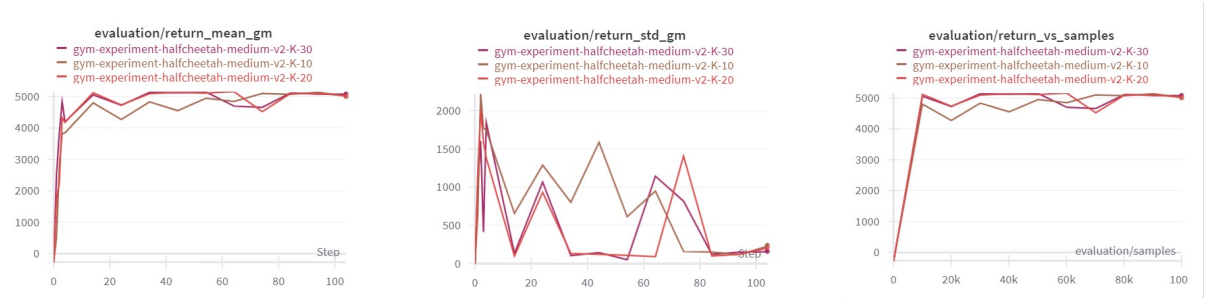


(b) Return Std. vs Number of Epochs(Steps) for RTG target 3600



(c) Return Mean vs Number of Samples for RTG target 3600

Figure 10: Variation of Return Mean and Std. for ODT with context length $K = 10, 20, 30$ in Hopper



(a) Return Mean vs Number of Epochs(Steps) for RTG target 6000

(b) Return Std. vs Number of Epochs(Steps) for RTG target 6000

(c) Return Mean vs Number of Samples for RTG target 6000

Figure 11: Variation of Return Mean and Std. for ODT with context length $K = 10, 20, 30$ in Hopper

(a) Return Mean vs Number of Epochs(Steps) for RTG target 5000

(b) Return Std. vs Number of Epochs(Steps) for RTG target 5000

(c) Return Mean vs Number of Samples for RTG target 5000

Figure 12: Variation of Return Mean and Std. for ODT with context length $K = 10, 20, 30$ in Walker2D

References

[1] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[4] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *international conference on machine learning*, pages 27042–27059. PMLR, 2022.

[5] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[7] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. URL `https://api.semanticscholar.org/CorpusID:49313245`.

[8] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning, 2020.

[9] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021.

[10] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards – just map them to actions, 2020.

[11] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies, 2019.

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

[13] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021.