StARformer: Transformer with State-Action-Reward Representations for Visual Reinforcement Learning

Jinghuan Shang, Kumara Kahatapitiya, Xiang Li, and Michael S. Ryoo

Stony Brook University, NY 11794, USA
{jishang, kkahatapitiy, xiangli8, mryoo}@cs.stonybrook.edu

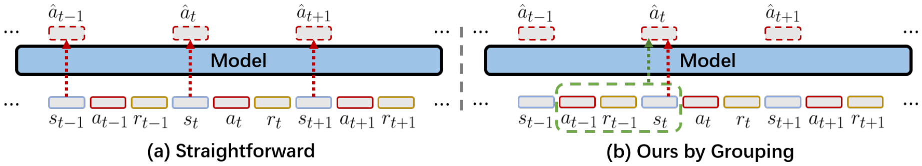
Abstract. Reinforcement Learning (RL) can be considered as a sequence modeling task: given a sequence of past state-action-reward experiences, an agent predicts a sequence of next actions. In this work, we propose **St**ate-**A**ction-**R**eward Transformer (**StAR**former) for visual RL, which explicitly models *short-term* state-action-reward representations (StAR-representations), essentially introducing a Markovian-like inductive bias to improve *long-term* modeling. Our approach first extracts StAR-representations by self-attending image state patches, action, and reward tokens within a short temporal window. These are then combined with pure image state representations — extracted as convolutional features, to perform self-attention over the whole sequence. Our experiments show that StARformer outperforms the state-of-the-art Transformer-based method on image-based Atari and DeepMind Control Suite benchmarks, in both offline-RL and imitation learning settings. StARformer is also more compliant with longer sequences of inputs. Our code is available at `https://github.com/elicassion/StARformer`.

Keywords: Reinforcement Learning, Transformer, Sequence Modeling

1 Introduction

Reinforcement Learning (RL) naturally operates sequentially: an agent observes a state from the environment, takes an action, observes the next state, and receives a reward from the environment. In the past, RL problems have been usually modeled as Markov Decision Processes (MDP). It enables us to take an action solely based on the current state, which is assumed to represent the whole history. With this scheme, sequences are broken into single steps so that algorithms like TD-learning [58] can be mathematically derived via Bellman Equation to solve RL problems. Recent advances such as [9,26] formulate (offline-)RL differently— as a sequence modeling task, and Transformer [64] architectures have been adopted as generative trajectory models to solve it, i.e., given past experiences of an agent composed of a sequence of state-action-reward triplets, a model iteratively generates an output sequence of action predictions.

This new scheme softens the MDP assumption, where an action is predicted considering multiple steps in history. To implement this, methods such as [9,26]


$\hat{a}_{t-1}$
$\hat{a}_t$
$\hat{a}_{t+1}$
Model
s_{t-1} a_{t-1} r_{t-1} s_t a_t r_t s_{t+1} a_{t+1} r_{t+1}
$\hat{a}_{t-1}$
$\hat{a}_t$
$\hat{a}_{t+1}$
Model
s_{t-1} a_{t-1} r_{t-1} s_t a_t r_t s_{t+1} a_{t+1} r_{t+1}

(a) Straightforward

(b) Ours by Grouping

Fig. 1: Illustration of RL as sequence modeling using Transformer: (a) A straightforward approach and, (b) Our proposed improvement. The intuition is to explicitly model local features (green boundary) to help long-term sequence modeling.

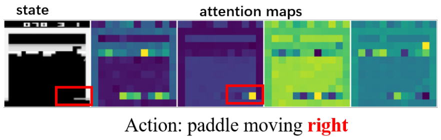


Fig. 2: Attention maps between action token and pixel state patches in our method. In the second attention map from the left, weights in the paddle region are directed towards right (highlighted in red), corresponding to the semantic meaning of "right" action.

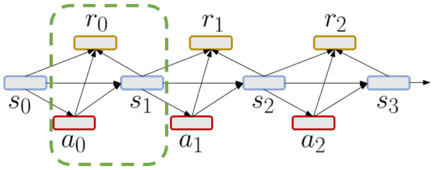


Fig. 3: MDP view of an RL process. Only the connected pairs (directed arrows) are causally-related, whereas others are independent of each other. Green boundary highlights our motivation: explicitly considering a single transition helps long-term modeling.

process the input sequence *plainly* through self-attention (with a causal attention mask) using Transformers [64]. This way, a given state, action, or reward token may attend to any of the (previous) tokens in the sequence, which allows the model to capture long-term relations. Moreover, each image state is usually encoded with convolutional networks (CNNs) *as-a-whole* prior to self-attention.

However, if we consider states, actions, and rewards within adjacent timesteps, they generally have strong connections due to potential causal relations. For instance, states in the recent past have a stronger effect on the next action, compared to states in the distant past. Similarly, the immediate-future state and the corresponding reward are direct results of the current action. In an extreme case— MDP, the relations are far more strong and restricted (see Fig. 3). In the above scenarios, a Transformer attending to all tokens naively may suffer from excess information (making the learning-process harder) or dilute the truly-essential relation priors. This is especially critical when input sequences are quite large, either in spatial [68] or temporal [26] dimension, and when Transformer models become heavy, i.e., contain a large number of layers [62]. Learning Markovian-like dependencies between tokens from scratch is hard and may waste computations [26], as the rest of the dependencies are possibly weaker. Moreover, tokenizing image states as-a-whole based on CNNs further prohibits Transformer models from capturing detailed spatial relations. Such loss of information can be critical especially in RL tasks with fine-grained regions-of-interest.

To alleviate such issues, we propose to explicitly model single-step transitions, introducing a Markovian-like inductive bias and relieving the capacity to be used for long sequence modeling. We introduce **St**ate-**A**ction-**R**eward Transformer (**StAR**former) for visual RL, which consists of two interleaving components: a Step Transformer and a Sequence Transformer. The Step Transformer learns local representations (i.e., **StAR**-representations) by self-attending state-action-reward tokens *within a window of single time-step*. Here, image states are encoded as ViT-like [18] patches, retaining fine-grained spatial information. The Sequence Transformer then combines StAR-representations with pure image state representations (extracted as convolutional features) *from the whole sequence* to make action predictions. Our experiments validate the benefits of StARformer over prior work in both offline-RL and imitation learning settings, while also being more compliant with longer input sequences

Our contributions are as follows: we (1) propose to model single-step transitions in RL explicitly, relieving model capacity to better focus on long-term relations, (2) present a method to combine ViT-like image patches with action and reward token to retain fine-grained spatial information, and (3) introduce an architecture to fuse StAR-representations over a long-sequence with our interleaving Step and Sequence Transformer layers. In particular, this allows modeling sequences of state-action feature representations at multiple different levels.

2 Related Work

2.1 Reinforcement Learning to Sequence Modeling

Reinforcement Learning (RL) is usually modeled as a Markov Decision Process (MDP). Based on this, single-step value-estimation methods have been derived from the Bellman equation, including Q-learning [66] and Temporal Difference (TD) learning [53,58,60,30], along with their extensions [43,71,24].

More recent directions [9,26] formulate RL a different way — as a sequence modeling task, i.e., given a sequence of recent experiences including state-actions-reward triplets, a model predicts a sequence of next actions. This approach can be trained in a supervised learning manner, being more compliant with offline RL [36] and imitation learning settings [25,61,56]. Zheng et al. [72] adapt this formulation to online settings. Furuta et al. [20] extend DT [9] to match given hindsight information. Reed et al. [52] train a single agent that performs a wide range of RL and language tasks. Sequence modeling can be also viewed as solving RL by learning trajectory representations. Other than methods learning visual representations only [71,70,69,35,34,38,55], our approach combines visual and trajectory representations together, thanks to the power of Transformer.

2.2 Transformers

Transformer architectures [64] have been first introduced in language processing tasks [17,50,51], to model interactions between a sequence of word embeddings, or more generally, unit representations or *tokens*. Recently, Transformers have been adopted in vision tasks with the key idea of breaking down images/videos into tokens [18,5,10,45,23,7,28], often outperforming convolutional networks (CNNs) in practice. Inspired by designs from both Transformers and CNNs, combining the two [15,42] shows further improvements. Transformers also found to be useful in handling sensory information [59] and doing one-shot imitation learning [16]. Chen et al.[9] explore how GPT [51] can be applied to RL under the sequence modeling setting.

Sequence modeling in visual RL is similar to learning from videos in terms of input data, which are composed of sequences of observed images (i.e. states). One challenge of applying Transformers to videos is the large number of input tokens and quadratic computation. These problems have been investigated in multiple directions, including attention approximation [12,65,29], separable attention in different dimensions [5,7], reducing the number of tokens using local windows [40,41], adaptively generating a small amount of tokens [54] or using a CNN-stem to come up with a small amount of high-level tokens [67,46,14].

StARformer shares a similar concept to performing spatial and temporal attention separately as in [5,7]. In contrast to such methods designed to reduce attention computation, our primary target is introducing inductive bias: modeling short-term and long-term contexts separately. Our method also operates on different sets of tokens, in short-term (s-a-r tokens) and in long-term (learned intermediate StAR-representation), which deviates from previous methods.

3 Preliminary

3.1 Transformer

Transformer [64] architectures have shown diverse applications in language [17] and vision tasks [18,5]. Given a sequence of input tokens $X = \{x_1, x_2, ..., x_n\}$, where $x_i \in \mathbb{R}^d$, a Transformer layer maps it to an output sequence of tokens $Z = \{z_1, z_2, ..., z_n\}$, where $z_i \in \mathbb{R}^d$. A Transformer model is obtained by stacking multiple such layers. We denote the mapping for each layer (l) as $F(\cdot)$: $Z^l = F(Z^{l-1})$. We use $F(\cdot)$ to represent a Transformer layer in the remaining sections.

Self-attention [39,48,11,64] is the core component of Transformers, which models pairwise relations between tokens. As introduced in [64], an input token representation X is linearly mapped into query, key and value representations, i.e., $\{Q, K, V\} \in \mathbb{R}^{n \times d}$ respectively, to compute self-attention as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V. \quad (1)$$

Vision Transformer (ViT) [18] extends the same idea of self-attention to the image domain. Given an input image $s \in \mathbb{R}^{H \times W \times C}$, a set of n non-overlapping local patches $P = \{p_i\} \in \mathbb{R}^{h \times w \times C}$ is extracted, flattened and linearly mapped to a sequence of tokens $\{x_i\} \in \mathbb{R}^d$. We extend ViT [18] so that action, reward, and state patches can jointly attend, where we find semantic meanings could be learned within action-patch attention in RL tasks (Fig. 1).

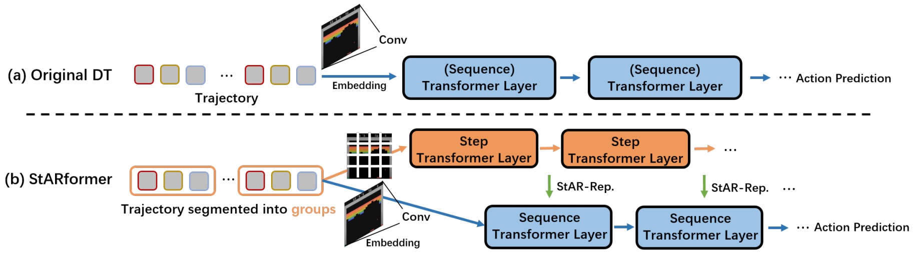


Fig. 4: (a) Structure summary of original DT [9], where the Transformer Layer acts similar as our Sequence Transformer. (b) StARformer consists of Step Transformer and Sequence Transformer, to separately model a single-step and the sequence as-a-whole, respectively. Two types of layers are connected at each level via learned StAR-representations. In terms of state embedding methods, DT uses only convolution, while StARformer uses ViT-like [18] embeddings (patches) in Step Transformer and convolution in Sequence Transformer separately.

3.2 RL as Sequence Modeling

We consider a Markov Decision Process (MDP), described by tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R})$, where $s \in \mathcal{S}$ represents the state, $a \in \mathcal{A}$, the action, $r \in \mathcal{R}$, the reward, and P, the transition dynamics given by $P(s'|s, a)$. In MDP, a trajectory (τ) is defined as the past experience of an agent, which is a sequence composed of states, actions, and rewards in the following temporal order:

$$\tau = \{s_1,\ a_1,\ r_1,\ s_2,\ a_2,\ r_2,\ \ldots,\ s_t,\ a_t,\ r_t\}. \tag{2}$$

Sequence modeling for RL is making action predictions from past experience [9,26]:

$$Pr(\hat{a}_t) = p(a_t|\ s_{1:t},\ a_{1:t-1},\ r_{1:t-1}). \tag{3}$$

Recent work [9,26] try to adopt an existing Transformer architecture [51] for RL with the formulation as above. In [9,26], states (s), actions (a), and rewards (r) are considered as input tokens (see Fig. 1a), while using a causal mask to ensure an autoregressive output sequence generation (i.e. following Eq. 3), where a token can access any of its preceding tokens through self-attention.

In contrast, our formulation attends tokens with (potentially) strong causal relations *explicitly*, while attending to long-term relations as well. To do this, in this work, we break a trajectory into small groups of state-action-reward tuples (i.e., s, a, r). It learns local relations within the tokens of each group through self-attention (see Fig. 1b, and Fig. 2), followed by long-term sequence modeling.

4 StARformer

4.1 Overview

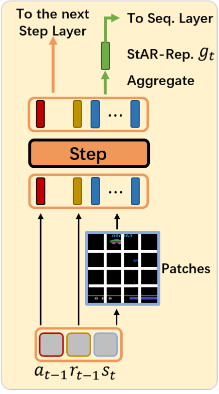
StARformer consists of two basic components: Step Transformer and Sequence Transformer, together with interleaving connections (see Fig. 4b). Step Transformer learns StAR-representations from strongly-connected local tokens *explicitly*, which are then fed into the Sequence Transformer along with pure state representations to model the whole input trajectory. At the output of the final Sequence Transformer layer, we make action predictions via a prediction head. In the following subsections, we will introduce the two Transformer components, and their corresponding token embeddings in detail.

4.2 Step Transformer

Grouping State-Action-Reward: Our intuition of grouping is to model strong local relations explicitly. To do so, we first segment a trajectory (τ) into a set of groups, where each group consists of previous action (a_{t-1}), reward (r_{t-1}) and current state (s_t) (Fig. 5). Each element within a group has a strong causal relation with the others.

Patch-wise State Token Embeddings: In Step Transformer, we tokenize each input image state by dividing it into a set of non-overlapping spatial patches z_{s_t} along its spatial dimensions, following ViT [18] (Fig. 5). Our motivation for using patch embeddings is to create fine-grained state embeddings. This allows the Step Transformer to model the relations of actions and rewards with local-regions of state (Fig 2). Such local correspondences provide more information compared to highly-abstracted convolutional features in this single-step modeling, which is empirically validated in our ablation studies (Sec. 6.4).

Action and Reward Token Embeddings: We embed the action and reward tokens with a linear layer as in [9].

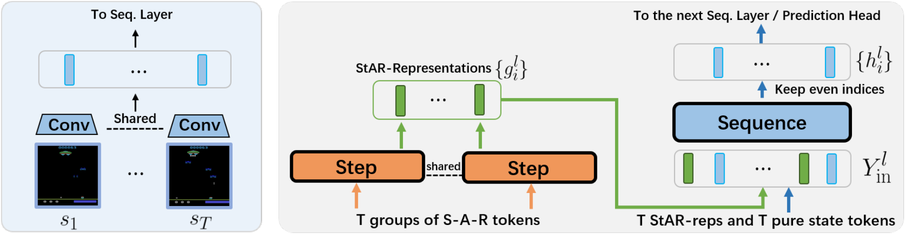


S-A-R Embedding and Step Transformer

Fig. 5: Overview of Step Transformer. Output tokens are (1) sent to the next Step Transformer layer and (2) aggregated to produce StAR-representation.

S-A-R embeddings: Altogether, we get a collection of state, action, and reward embeddings as the input to the initial Step Transformer layer which is given by: $Z_t^0 = \{z_{a_{t-1}},\ z_{r_t},\ z_{s_t^1},\ z_{s_t^2},\ \ldots,\ z_{s_t^n}\}$. We have T groups of such token representations per trajectory, which are simultaneously processed by the Step Transformer with shared parameters.

Step Transformer Layer: We adopt the conventional Transformer design from [64] (Sec. 3.1) as our Step Transformer layer. Each group of tokens from the previous layer Z_t^{l-1} is transformed to Z_t^l by a Step Transformer layer with the mapping F_{step}^l: $Z_t^l = F_{\text{step}}^l(Z_t^{l-1})$.



(a) Pure State Embedding

(b) Sequence Transformer and Step-to-Seq. connection

Fig. 6: (a) Pure state embeddings are learned from shared convolutional layers. (b) Sequence Transformer takes StAR-representation and the pure state tokens and generate output tokens.

StAR-representation: At the output of each Step Transformer layer l, we further obtain a State-Action-Reward-representation (StAR-representation) $g_t^l \in \mathbb{R}^D$ by aggregating output tokens $Z_t^l \in \mathbb{R}^{n \times d}$ (see green flows in Fig. 4b):

$$g_t^l = \text{FC}([Z_t^l]) + e_t^{\text{temporal}}. \tag{4}$$

Here $[\cdot]$ represents concatenation of the tokens within each group and $e_t^{\text{temporal}} \in \mathbb{R}^D$, the temporal positional embeddings for each timestep. Finally, the output StAR-representation g_t^l is fed into the corresponding Sequence Transformer layer for long-term sequence modeling.

4.3 Sequence Transformer

Our Sequence Transformer models long-term sequences by looking at the learned StAR-representations and the *pure state tokens* (introduced below) over the whole trajectory (See Fig. 6). Notice that, as illustrated in Fig. 4 (b), this happens with multiple intermediate StAR representations, allowing the Sequence Transformer to capture detailed information.

Pure State Token Embeddings: In addition to the patch-wise token embeddings in Step Transformer, we embed the input image state s_t as-a-whole, to create pure state tokens h_t^0. Each such token represents a single state representation, describing the state globally in space. We do this by processing each state through a CNN encoder, since the convolutional layers mix features spatially:

$$h_t^0 = \text{Conv}(s_t) + e_t^{\text{temporal}}, \tag{5}$$

where $e_t^{\text{temporal}} \in \mathbb{R}^D$ represents the temporal positional embeddings exactly the same as we add to g_t for each timestep. The convolutional encoder is from [44].

Sequence Transformer Layer: Similar to Step Transformer, we use the conventional Transformer layer design from [64] for our Sequence Transformer. The input to the Sequence Transformer layer l consists of representations from two sources: (1) the learned StAR-representations $g_t^l \in \mathbb{R}^D$ from the corresponding Step Transformer layer, and (2) $h_t^{l-1} \in \mathbb{R}^D$ from the outputs of the previous Sequence Transformer layer. Here, as mentioned above, we set h_t^0 to be the pure state representation. The two types of token representations are merged to form a single sequence, preserving their temporal order (as elaborated below):

$$Y_{\text{in}}^l = \{g_1^l,\ h_1^{l-1},\ g_2^l,\ h_2^{l-1},\ \ldots,\ g_T^l,\ h_T^{l-1}\}. \tag{6}$$

We place g_t^l before h_t^{l-1}— which originates from s_t— because g_t^l contains information of the *previous* action a_{t-1}, which comes prior to s_t in the trajectory. We also apply a causal mask in the Sequence Transformer to ensure that the tokens at time t cannot attend any future tokens (i.e., $> t$).

Here, Sequence Transformer takes StAR-representations generated from each intermediate Step Transformer layer, rather than taking the final StAR-representations after all Step Transformer layers. In this way, the model gains an ability to look at StAR-representations at multiple abstraction levels. In Section 6.4, we empirically validate the benefit of this layer-wise fusion.

Sequence Transformer computes an intermediate set of output tokens as in: $Y_{\text{out}}^l = F_{\text{sequence}}^l(Y_{\text{in}}^l)$. We then select the tokens at even indices of Y_{out}^l (where indexing starts from 1) to be the pure state tokens $h_i^l := y_{\text{out};2i}^l$, which are then fed into the next Sequence Transformer layer.

Action Prediction: The output of the last Sequence Transformer layer is used to make action predictions, based on a linear head: $\hat{a}_t = \phi(h_t^l)$.

4.4 Training and Inference

StARformer is a drop-in replacement of DT [9], as training and inference procedures remain the same. StARformer can easily operate on step-wise reward without a performance drop (detailed discussed in 6.4). In contrast, it is critical to design a Return-to-go (RTG, target return) carefully in DT, which needs more trials and tuning to find the best value.

5 Experimental Setup

5.1 Settings

We consider offline RL [36] and imitation learning (behavior cloning) in our experiments. In offline RL, we have a fixed memory buffer of sub-optimal trajectory rollouts. Offline RL is generally more challenging compared to conventional RL due to the shifted distribution problem [36].

In our Imitation learning setting, the agent is not exposed to reward signals and online-collected data from the environment. This is an even harder problem due to provided trajectories being sub-optimal, compared to traditional imitation learning that could collect new data and do Inverse Reinforcement

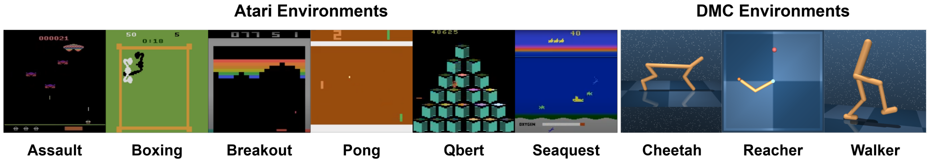


Fig. 7: Environments used: Atari is with a discrete action space, and DMC is with a continuous action space. We use gray-scale input similar to prior work [43,9].

Learning [47,1]. We simply remove the rewards in the dataset used in offline RL to come up with this setting. Both DT [9] and proposed method can operate without reward, by simply removing T reward tokens in DT [9] (T is trajectory length), or removing the reward token in Step Transformer in our model.

5.2 Environments and Datasets

We consider image-based Atari [6] (discrete action space) and DeepMind Control Suite (DMC) [63] (continuous action space) to evaluate our model in different types of tasks, listed in Fig. 7 with image examples. We pick 6 games in Atari: Assault, Boxing, Breakout, Pong, Qbert, and Seaquest. Similar to [9] we use 1% (500k steps) of the DQN replay buffer dataset [2] to perform a thorough and fair comparison. We select 3 continuous control tasks in DMC [63]: Cheetah-run, Reacher-easy, and Walker-walk. In DMC, we collect a replay buffer (i.e. sub-optimal trajectories) generated by training a SAC [21] agent from scratch for 500k steps for each task. Note that these continuous control tasks are with image inputs, which previous work [9] does not cover (originally using Gym [8]).

We report the absolute value of episodic returns (i.e., cumulative rewards). Results are averaged across 7 random seeds in Atari and 10 seeds in DMC, each seed is evaluated by 10 randomly initialized episodes.

5.3 Baselines

We select Decision-Transformer (DT) [9], a SOTA Transformer-based sequence modeling method for RL. We notice there is also Trajectory-Transformer [26], which however, is not designed for image inputs. We use most of the same hyper-parameters as in DT [9] for Atari environments without extra tuning (details in Supplementary Table 4 and 5). As for DMC environments, since they are not covered by DT [9], we carefully tune the baseline first and then use the same set of hyper-parameters in our method. We also compare with SOTA non-Transformer offline-RL methods including CQL [33], QR-DQN [13], REM [3], and BEAR [31]. For imitation (behavior cloning), we only compare with DT [9] and straightforward behaviour cloning with ViT (referred to as BC-ViT).

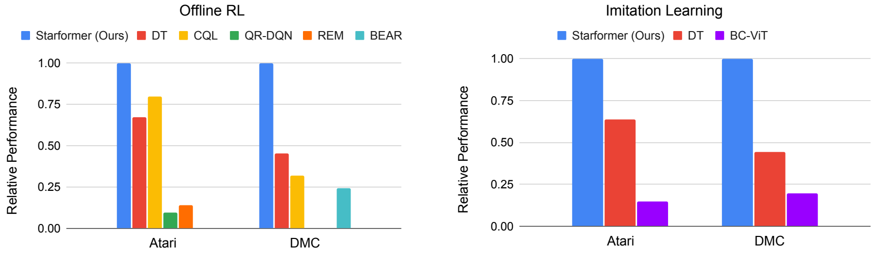


Fig. 8: Relative performance of episodic returns. The results are averaged across all environments and random seeds (same in later experiments), and normalized w.r.t. the performance of StAR. Please refer Table 1 in supplementary for details.

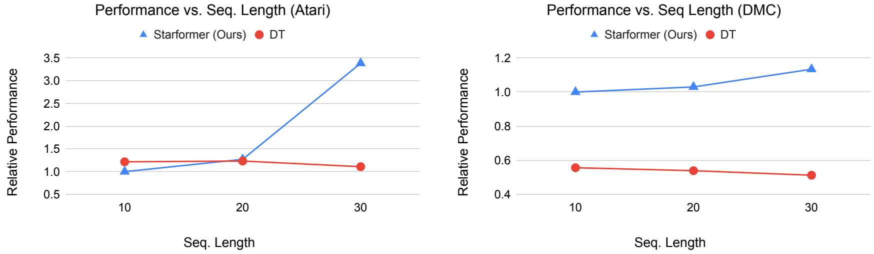


Fig. 9: Change in performance with the length of input sequence, $T \in \{10, 20, 30\}$, in Atari and DMC (averaged across tasks), under offline-RL. Please refer to Fig.1 in supplementary for per-task result.

6 Results

6.1 Improving Sequence Modeling for RL

We first compare our StARformer (StAR) with the state-of-the-art Transformer-based RL method in Atari and image-based DMC environments, under both offline RL and imitation learning settings. We select the Decision-Transformer proposed in [9], (referred as DT) as our baseline. Here, we keep $T = 30$ for all environments, which is the number of time-steps (length) of each input trajectory. We also compare our method to CQL [33], a SOTA non-Transformer offline-RL method. Fig. 8 shows that our method outperforms baselines, in both offline RL and imitation learning settings, suggesting that our method can better model reinforcement sequences with images.

6.2 Scaling-up to Longer Sequences

In this experiment, we evaluate how StARformer and DT perform with different input sequence lengths, specifically $T = \{10, 20, 30\}$, under offline-RL setting.

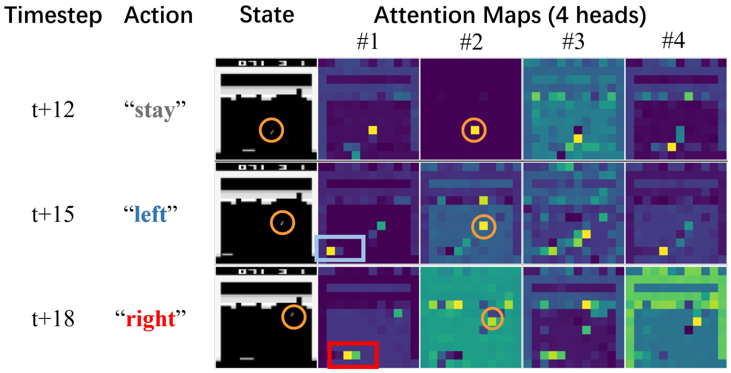


Fig. 10: Visualization of attention maps in our Step Transformer, extracted for Breakout game. Attention weights are computed between the action token and state patch tokens. We highlight the ball (orange circle) in the input for convenience. Please find out more visualizations in our supplementary material.

In Fig. 9, we see that StARformer gains performance with longer trajectories, whereas DT [9] saturates as early as $T = 10$. This validates our claim that considering short-term and long-term relations separately (and then fusing) help models to scale-up to longer sequences. Instead of learning Markovian pattern attentions [26] implicitly, we model it explicitly in our Step Transformer. This acts as an inductive bias, relieving the capacity of Sequence Transformer to better focus on long-term relations. In contrast, DT takes off-the-shelf language model GPT [51], in which Markov property is not considered.

6.3 Visualization

We show attention maps between action and state patches in Step Transformer at several timesteps extracted from a trajectory in Breakout (see Fig. 10). In this game, the agent should move the paddle to bounce the ball back from the bottom, after the ball falls down while breaking the bricks on the top. In the presented attention maps, the regions with a high attention score (highlighted) mainly overlap with the locations of the ball, paddle, and potential target bricks. We find the attention maps in head #1 to be particularly interesting. Here, the focused regions corresponding to the paddle show a directional pattern, corresponding to the semantic meaning of actions "moving the paddle right", "left" or "stay". This validates that Step Transformer captures essential spatial relations between actions and state patches, which is important for decision making. Moreover, in head #2, we observe that the focused regions correspond to the locations of the ball, except when the ball is out of the boundary, too-close to the paddle or indistinguishable within bricks. Overall, these attention maps suggest how our model can show a basic understanding of the Breakout game.

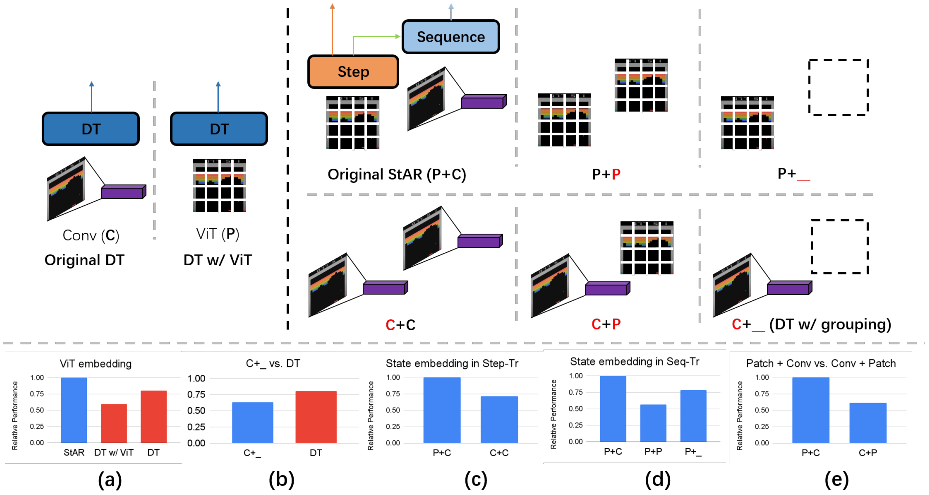


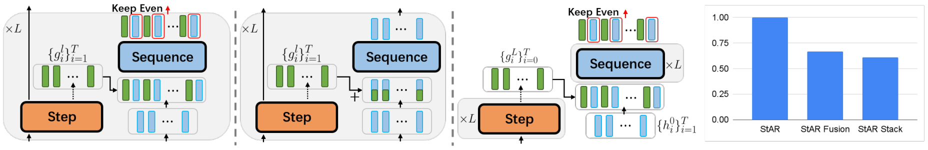
Fig. 11: (Top): Embedding methods used in original DT, StARformer (StAR), and their variants. We label ViT (patches) as **P**, convolution as **C**, and None (not using the corresponding embedding) as “__”. (Bottom): (a-e) Performance comparisons between variants. Per-task results are in Supplementary Table 2.

6.4 Ablations

StARformer has three design differences compared to baseline DT [9]: it (1) learns StAR-representation from single-step transitions (grouping), (2) uses both ViT-like [18] patch embeddings and convolutions for state representation, and (3) merges these two types of embeddings in Step Transformer *layer-wise*.

StAR-representation and State Representations: We first discuss designs of StAR-representation and state representation methods ((1) and (2) mentioned above) jointly, as they can be unified into variants shown in Fig. 11. We vary state embedding methods used to learn s_t in Step Transformer and h_t in Sequence Transformer. Namely, we consider: (1) ViT features (patch embeddings, labeled as **P**), (2) Convolutional features (labeled as **C**), or (3) None (not having the corresponding embedding, labeled as “__”). The original StARformer can be represented by **P+C** (patch embeddings for s_t and convolutional embeddings for h_t). Other variants include: **P+P**, **P+__**, **C+P**, **C+C**, and **C+__**. We note variant **C+__** could be viewed as DT + grouping, where we simply adapt DT to our framework, and learn StAR-representation from convolutional features only. We also implement a variant of DT using ViT for state embedding (noted as DT w/ ViT), to match our method in terms of having a similar embedding method and capacity (13M parameters vs. 14M parameters in ours).

When comparing StARformer with original DT and DT w/ ViT (Fig. 11(a)), we see a performance drop in DT when used with ViT, which suggests that re-



(a) StAR (b) StAR Fusion (c) StAR Stack (d) Performance Comparison

Fig. 12: Variants of our model w/ different connections. Two variants, (b) StAR Fusion and (c) StAR Stack are shown in comparison to our original (a) StAR model. (d) Experiments (offline-RL) show original structure, which is a layer-wise fusion, works best. Please refer to Supplementary Table 3 for per-task results.

placing convolutional features with ViT-like features naively would not benefit the model, despite the increased capacity (similar to ours). StARformer, however, does not benefit only from the larger capacity, but also from its better structural design, as verified in following experiments (see Fig. 11(c)(d)(e)). From Fig. 11(b), we see that **C**+__ which only uses convolutional features at Step Transformer, performs worse compared to DT. This is because convolutional features are highly abstracted, which makes them not well-suited for single-step transition (i.e., fine-grained) modeling.

When comparing **P**+**C** with **C**+**C** (Fig. 11(c)), the lower performance of **C**+**C** suggests that patches embeddings are better suited to model single transitions in Step Transformer. In Fig. 11(d), we compare **P**+**C** with **P**+**P** and **P**+__. We find convolution features work best in Sequence Transformer, validating that they provide abstract global information which is useful for long range modeling (coarse), in contrast to patch embeddings. The observations from above comparisons of StARformer variants suggest that our method benefits from fusing patch and convolutional features. We further evaluate this by comparing **P**+**C** and **C**+**P** (Fig. 11(e)), where **P**+**C** performs better, confirming this fusion method of “fine-grained (patches) to high-level (conv)” best matches with our sequence modeling scheme of “single-transition followed-by long-range-context”.

Step-to-Sequence Layer-wise Connections: In our model, we model whole trajectory using representations from two sources: StAR-representations g from Step Transformer , and pure state representation h from previous layer of Sequence Transformer. We combine g and h in a layer-wise manner (i.e., at each corresponding layer). We investigate two other variants: (1) g_t^l is fused with h_t^l by summation (referred as StAR Fusion, see Fig. 12(b)), and (2) the Sequence Transformer is “stacked” on-top of the Step Transformer (referred as StAR Stack, see Fig. 12(c)). Results of these configurations are shown in Fig. 12(d) and StAR works the best. We see that attending to all tokens is better than token summation at Sequence Transformer. Also, having StAR-representations from different abstraction levels is beneficial compared to having one.

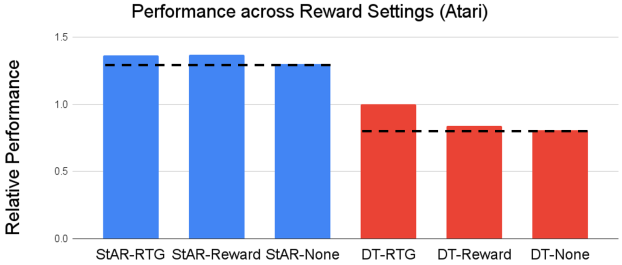


Fig. 13: Performance in different reward settings: return-to-go (RTG), stepwise reward, or no reward at-all (labeled as 'None') settings.

Reward setting: Return-to-go, stepwise reward, or no reward at-all? We investigate how different reward settings affect sequence modeling, specifically, return-to-go (RTG) [9], stepwise reward, and no reward at-all. Decision-Transformer [9] originally uses RTG $\hat{R}_t$, which is defined as the sum of future step-wise rewards: $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$, widely used in [27,4,49,57,32,37,19]. Stepwise reward r_t is the immediate reward generated by an environment in each step, which is generally used in most RL algorithms [43,24]. StARformer uses r_t by default, guided by the motivation of modeling single-step transitions. No reward at-all corresponds to imitation (behavior cloning).

Results are shown in Fig. 13. StARformer and DT behaves differently when reward settings are varied. Both methods show performance gains with reward. Also, StARformer performs similarly regardless of RTG or stepwise reward, whereas DT relies more on RTG, and StARformer-None can still outperform DT-RTG, even without reward. These observations tell sequence modeling can even work on state-action-only trajectories when the model has enough capacity. Such observation is consistent with Dreamerv2 [22], where no-reward setting performs as well as having reward due to the strong dynamics model.

7 Conclusion

In this work, we introduce StARformer, which models strong local relations explicitly (Step Transformer) to help improve the long-term sequence modeling (Sequence Transformer) in Visual RL. Our extensive empirical results show how the learned StAR-representations help our model to outperform the baseline. We further demonstrate that our method successfully models trajectories, with an emphasis on long sequences.

Acknowledgements. We thank members in Robotics Lab at Stony Brook for valuable discussions. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Ministry of Science and ICT (No.2018-0-00205, Development of Core Technology of Robot Task-Intelligence for Improvement of Labor Condition.

References

1. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML). p. 1 (2004)
2. Agarwal, R., Schuurmans, D., Norouzi, M.: An optimistic perspective on offline reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 104–114 (July 2020)
3. Agarwal, R., Schuurmans, D., Norouzi, M.: An optimistic perspective on offline reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 104–114. PMLR (2020)
4. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W.: Hindsight experience replay. In: Advances in Neural Information Processing Systems (NeurIPS) (2017)
5. Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lučić, M., Schmid, C.: ViViT: A video vision transformer. In: Proceedings of the International Conference on Computer Vision (ICCV) (Oct 2021)
6. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. J Artif Intell Res . **47**(1), 253–279 (May 2013)
7. Bertasius, G., Wang, H., Torresani, L.: Is space-time attention all you need for video understanding? In: Proceedings of the International Conference on Machine Learning (ICML) (July 2021)
8. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI gym (2016), arXiv:1606.01540
9. Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., Mordatch, I.: Decision transformer: Reinforcement learning via sequence modeling. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2021)
10. Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Dhariwal, P., Luan, D., Sutskever, I.: Generative pretraining from pixels. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 1691–1703 (Jul 2000)
11. Cheng, J., Dong, L., Lapata, M.: Long short-term memory-networks for machine reading (2016), arXiv:1601.06733
12. Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L., Weller, A.: Rethinking attention with performers. In: Proceedings of the International Conference on Learning Representations (ICLR) (Apr 2020)
13. Dabney, W., Rowland, M., Bellemare, M., Munos, R.: Distributional reinforcement learning with quantile regression. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). vol. 32 (2018)
14. Dai, R., Das, S., Kahatapitiya, K., Ryoo, M.S., Bremond, F.: Ms-tct: Multi-scale temporal convtransformer for action detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 20041–20051 (2022)
15. Dai, Z., Liu, H., Le, Q.V., Tan, M.: CoAtNet: Marrying convolution and attention for all data sizes. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2021)
16. Dasari, S., Gupta, A.: Transformers for one-shot visual imitation. In: Conference on Robot Learning (CoRL) (2020)

17. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding (2019), arXiv:1810.04805
18. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. In: Proceedings of the International Conference on Learning Representations (ICLR) (Apr 2020)
19. Eysenbach, B., Geng, X., Levine, S., Salakhutdinov, R.: Rewriting history with inverse rl: Hindsight inference for policy improvement. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
20. Furuta, H., Matsuo, Y., Gu, S.S.: Distributional decision transformer for hindsight information matching. In: Proceedings of the International Conference on Learning Representations (ICLR) (2022)
21. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 1861–1870 (Jul 2018)
22. Hafner, D., Lillicrap, T., Norouzi, M., Ba, J.: Mastering atari with discrete world models. arXiv preprint arXiv:2010.02193 (2020)
23. Han, K., Xiao, A., Wu, E., Guo, J., Xu, C., Wang, Y.: Transformer in transformer. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2021)
24. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2018)
25. Ho, J., Ermon, S.: Generative adversarial imitation learning. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2016)
26. Janner, M., Li, Q., Levine, S.: Offline reinforcement learning as one big sequence modeling problem. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2021)
27. Kaelbling, L.P.: Learning to achieve goals. In: International Joint Conference on Artificial Intelligence (IJCAI) (1993)
28. Kahatapitiya, K., Ryoo, M.S.: Swat: Spatial structure within and among tokens. arXiv preprint arXiv:2111.13677 (2021)
29. Kitaev, N., Kaiser, L., Levskaya, A.: Reformer: The efficient transformer. In: Proceedings of the International Conference on Learning Representations (ICLR) (May 2019)
30. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2000)
31. Kumar, A., Fu, J., Soh, M., Tucker, G., Levine, S.: Stabilizing off-policy q-learning via bootstrapping error reduction. Advances in Neural Information Processing Systems (NeurIPS) **32** (2019)
32. Kumar, A., Peng, X.B., Levine, S.: Reward-conditioned policies. arXiv preprint arXiv:1912.13465 (2019)
33. Kumar, A., Zhou, A., Tucker, G., Levine, S.: Conservative q-learning for offline reinforcement learning. Advances in Neural Information Processing Systems (NeurIPS) **33**, 1179–1191 (2020)
34. Laskin, M., Srinivas, A., Abbeel, P.: Curl: Contrastive unsupervised representations for reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML). pp. 5639–5650. PMLR (2020)

35. Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., Srinivas, A.: Reinforcement learning with augmented data. Advances in Neural Information Processing Systems **33**, 19884–19895 (2020)
36. Levine, S., Kumar, A., Tucker, G., Fu, J.: Offline reinforcement learning: Tutorial, review, and perspectives on open problems (2020), arXiv:2005.01643
37. Li, A.C., Pinto, L., Abbeel, P.: Generalized hindsight for reinforcement learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)
38. Li, X., Shang, J., Das, S., Ryoo, M.S.: Does self-supervised learning really improve reinforcement learning from pixels? arXiv preprint arXiv:2206.05266 (2022)
39. Lin, Z., Feng, M., dos Santos, C.N., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding (2017), arXiv:1703.03130
40. Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., Guo, B.: Swin transformer: Hierarchical vision transformer using shifted windows. Proceedings of the International Conference on Computer Vision (ICCV) pp. 10012–10022 (Oct 2021)
41. Liu, Z., Ning, J., Cao, Y., Wei, Y., Zhang, Z., Lin, S., Hu, H.: Video swin transformer (2021), arXiv:2106.13230
42. Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T., Xie, S.: A convnet for the 2020s. arXiv preprint arXiv:2201.03545 (2022)
43. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013), arXiv:1312.5602
44. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)
45. Neimark, D., Bar, O., Zohar, M., Asselmann, D.: Video transformer network (2021), arXiv:2102.00719
46. Neimark, D., Bar, O., Zohar, M., Asselmann, D.: Video transformer network. In: Proceedings of the International Conference on Computer Vision (ICCV). pp. 3163–3172 (2021)
47. Ng, A.Y., Russell, S.J., et al.: Algorithms for inverse reinforcement learning. In: Proceedings of the International Conference on Machine Learning (ICML). vol. 1, p. 2 (2000)
48. Parikh, A.P., Täckström, O., Das, D., Uszkoreit, J.: A decomposable attention model for natural language inference (2016), arXiv:1606.01933
49. Pong, V., Gu, S., Dalal, M., Levine, S.: Temporal difference models: Model-free deep rl for model-based control. Proceedings of the International Conference on Learning Representations (ICLR) (2018)
50. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I.: Improving language understanding by generative pre-training (2018)
51. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI blog **1**(8) (2019)
52. Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S.G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J.T., et al.: A generalist agent. arXiv preprint arXiv:2205.06175 (2022)
53. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems, vol. 37. Citeseer (1994)
54. Ryoo, M.S., Piergiovanni, A., Arnab, A., Dehghani, M., Angelova, A.: TokenLearner: Adaptive space-time tokenization for videos. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2021)

55. Shang, J., Das, S., Ryoo, M.S.: Learning viewpoint-agnostic visual representations by recovering tokens in 3d space. arXiv preprint arXiv:2206.11895 (2022)
56. Shang, J., Ryoo, M.S.: Self-supervised disentangled representation learning for third-person imitation learning. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 214–221. IEEE (2021)
57. Srivastava, R.K., Shyam, P., Mutz, F., Jaśkowski, W., Schmidhuber, J.: Training agents using upside-down reinforcement learning. arXiv preprint arXiv:1912.02877 (2019)
58. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine learning **3**(1), 9–44 (Aug 1988)
59. Tang, Y., Ha, D.: The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. arXiv preprint arXiv:2109.02869 (2021)
60. Tesauro, G., et al.: Temporal difference learning and TD-Gammon. Commun. ACM **38**(3), 58–68 (Mar 1995)
61. Torabi, F., Warnell, G., Stone, P.: Generative Adversarial Imitation from Observation (2019), arXiv:1807.06158
62. Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., Jégou, H.: Going deeper with image transformers. In: Proceedings of the International Conference on Computer Vision (ICCV). pp. 32–42 (Oct 2021)
63. Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., Tassa, Y.: dm_control: Software and tasks for continuous control. Software Impacts **6**, 100022 (Nov 2020)
64. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems (NeurIPS) (Dec 2017)
65. Wang, S., Li, B.Z., Khabsa, M., Fang, H., Ma, H.: Linformer: Self-attention with linear complexity (2020), arXiv:2006.04768
66. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3-4), 279–292 (May 1992)
67. Xiao, T., Singh, M., Mintun, E., Darrell, T., Dollár, P., Girshick, R.: Early convolutions help transformers see better. Advances in Neural Information Processing Systems (NeurIPS) **34**, 30392–30400 (2021)
68. Yang, J., Li, C., Zhang, P., Dai, X., Xiao, B., Yuan, L., Gao, J.: Focal self-attention for local-global interactions in vision transformers (2021), arXiv:2107.00641
69. Yarats, D., Fergus, R., Lazaric, A., Pinto, L.: Mastering visual continuous control: Improved data-augmented reinforcement learning. arXiv preprint arXiv:2107.09645 (2021)
70. Yarats, D., Kostrikov, I., Fergus, R.: Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In: Proceedings of the International Conference on Learning Representations (ICLR) (2020)
71. Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., Fergus, R.: Improving sample efficiency in model-free reinforcement learning from images. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 10674–10681 (May 2021)
72. Zheng, Q., Zhang, A., Grover, A.: Online decision transformer (2022)