Deep Reinforcement Learning with Transformers for Text Adventure Games

Yunqiu Xu*, Ling Chen*, Meng Fang†, Yang Wang‡ and Chengqi Zhang*
*Centre for Artificial Intelligence
University of Technology Sydney, Sydney, Australia
Email: Yunqiu.Xu@student.uts.edu.au, {Ling.Chen, Chengqi.Zhang}@uts.edu.au
†Tencent Robotics X
Email: mfang@tencent.com
‡Key Laboratory of Knowledge Engineering with Big Data (Ministry of Education)
Hefei University of Technology, China
Email: yangwang@hfut.edu.cn

***Abstract*—In this paper, we study transformers for text-based games. As a promising replacement of recurrent modules in Natural Language Processing (NLP) tasks, the transformer architecture could be treated as a powerful state representation generator for reinforcement learning. However, the vanilla transformer is neither effective nor efficient to learn with a huge amount of weight parameters. Unlike existing research that encodes states using LSTMs or GRUs, we develop a novel lightweight transformer-based representation generator featured with reordered layer normalization, weight sharing and block-wise aggregation. The experimental results show that our proposed model not only solves single games with much fewer interactions, but also achieves better generalization on a set of unseen games. Furthermore, our model outperforms state-of-the-art agents in a variety of man-made games.**

***Index Terms*—text-based games, reinforcement learning, natural language processing**

I. Introduction

Text-based games, or interaction fiction (IF) games, are software environments where both states and actions are described by textual descriptions [1]. To solve the games, the agents are required to understand natural language to deduct the objectives. Compared with typical Natural Language Processing (NLP) tasks, sequential decision making is usually involved in these games, where there's no predictive label to specify whether an action is correct or not, and the effect of an action may last long. Text-based games are usually seen as Reinforcement Learning (RL) tasks in the context of natural language processing [2], [3]. Existing works on text-based games have been focusing on the challenges including but not limited to building representation from text [4], partial observability [5], combinatorial action space [6] and sparse rewards [7].

For reinforcement learning, building an effective representation generator is one of the most crucial steps to solve the tasks. However, current models are limited in their ability to represent varying textual data. Compared with video games [8]–[10] or robotics tasks [11]–[13] or other NLP tasks [14], [15], where Convolutional Neural Networks (CNNs) and Multilayer Perceptrons (MLPs) are commonly used for input processing, the states in text-based games are usually processed via recurrent modules, such as LSTMs [4], [7], [16] and GRUs [1], [17]. Recently, it has been empirically validated that the transformer architecture with self-attention mechanism [18] deals better with longer temporal horizons and thus provides significant performance gains over the recurrent modules [19]–[21]. However, they are seldom deployed in the domain of text-based games.

In this paper, we propose a novel representation generator to build state representation effectively and efficiently for text-based games. Inspired by the success of transformer in many sequential information processing domains, we argue that this architecture is also powerful in building representations for text-based games. We first replace the recurrent encoder in representation generator with the vanilla transformer architecture to investigate whether it is more powerful. Then, we make a set of technical changes to the vanilla transformer to further improve its performance. In order to make convergence faster, we reorder the layer normalization by placing it within residual connection [22] to provide an identity map of input towards output. We then share the weights among all transformer blocks to reduce the number of learnable weights. Our third modification is to enhance residual connection via gating mechanisms [23], [24]. Specifically, we apply block-wise gate layer to aggregate the input and output of a block.

We evaluate our model on both text-based games generated from TextWorld CoinCollector Challenge [2] and man-made games supported by Jericho [1]. The experimental results show that the proposed model achieves higher data efficiency by requiring fewer interactions to solve the games. In addition to the single game setting, we also evaluate the model's generalizability on multiple unseen games. The results show that the proposed model succeeds to learn a general strategy to solve the games even though the scenes have not been observed during the training stage. In terms of man-made games, the proposed model outperforms current agents in a variety of games.

Our main contributions are summarized as follows:

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

- We are one of the first to study the transformer architecture for RL framework in text-based games.
- We show that the transformer architecture is powerful for building representations from text-based states.
- We develop a novel lightweight and easy-to-use transformer variant for RL games.
- We show that our model achieves good performance on not only generated games but also man-made games. Besides single games, our model exhibits promising generalizability in unseen games.

II. Related work

A. Methods for Text-based games

Text-based games have captured great attention from natural language processing, natural language understanding and reinforcement learning. The Text-Based Adventure AI Competition, launched in 2016, aims to find game agents that are capable of solving different text-based games [25]. Some competitors, such as BYU16Agent [26], Golovin [27], CARL [25] and NAIL [28], achieve competitive performance in the competition by injecting game playing heuristics instead of letting the agent learn from interaction only. For example, NAIL [28], which won first place in 2018's competition, is designed with strong heuristics for exploring the game, interacting with objects, and forming the game map to track the states. Despite being effective as game solvers, these agents require huge amount of prior knowledge for ideal rules and heuristics, which is more challenging than encoding states and learning policies from data [1].

Deep reinforcement learning has been applied to text-based games. Specifically, LSTM-DQN [4] uses a recurrent neural network to build state representations, and computes the Q values of objects and verbs respectively via two output layers. LSTM-DRQN [7] combines this idea with DRQN [29]. In particular, a recurrent layer is incorporated into the decision making process for partial observations. Other extensions include the Deep Reinforcement Relevance Network (DRRN) [16] and Template-DQN (TDQN) [1], where both the states and the actions are processed to learn representations. Besides RNN encoders, CNN encoders have also been applied to process text sequences [6], [30].

Other methods engage knowledge graphs [5], [17], [28], [31], [32], where the states and the entities are memorized and tracked explicitly. The knowledge graph serves as a complement of the current state, thus being helpful to handle partial observability. However, the rules for building and updating a knowledge graph need to be pre-defined manually, which hinders the usage of knowledge graph-based solutions. In contrast, we build state representations without any hand-crafted rules, while our method can be integrated with a knowledge graph to achieve better performance.

B. Transformers

The transformer architecture, based on the multi-head self-attention mechanism, has been shown to outperform RNNs in a wide range of NLP tasks such as machine translation [18], text summarization [33], question answering [34] and language modeling [20]. Nevertheless, it is far less applied in the RL area where recurrent modules are used dominantly. Zambaldi et al. [35] applied the multi-head self-attention mechanism to capture the relations between entities in multi-agent reinforcement learning tasks. Parisotto et al. [36] replaced LSTM with the transformer architecture, and proposed further modifications to tackle the optimization problem of a vanilla transformer. However, both LSTM and transformer serve as memory modules instead of state representation generators. Although the transformer is applied as part of the state encoder in [32], it is used straightforwardly without any analysis or optimization because the focus of the work is to explore the usage of knowledge graphs in text-based games.

Different from the aforementioned efforts, we focus on studying the effectiveness of the transformer in text-based games. In particular, we investigate 1) whether the transformer architecture could be used for generating state representations; 2) whether the optimization problem of a vanilla transformer [37] still exists when it is used as the representation generator; and 3) how to improve its performance in text-based games.

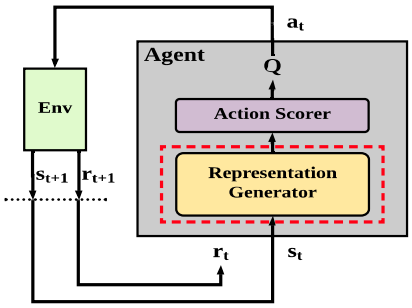

Fig. 1. The RL framework, which consists of a representation generator and an action scorer. In this work we focus on the representation generator part (shown in red dashed box).

III. Preliminaries

A. Overview

Similar to previous work [1], [7], [28], we formulate the text-based games as Partially Observable Markov Decision Processes (POMDPs). The POMDPs can be defined as a tuple $(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \boldsymbol{T}, \boldsymbol{O}, \boldsymbol{\Omega}, \boldsymbol{R}, \gamma)$, where the state set $\boldsymbol{\mathcal{S}}$, action set $\boldsymbol{\mathcal{A}}$, conditional transition probabilities between states $\boldsymbol{T}$, reward function $\boldsymbol{R}$ and discount factor $\gamma \in (0, 1]$ are same as the setting of MDPs. The $\boldsymbol{\Omega}$ is the observation set and $\boldsymbol{O}$ is conditional observation probabilities. Different from MDPs, the state $\boldsymbol{s}_t$ in POMDP cannot be directly observed. Instead, at each time step, the agent will receive an observation $\boldsymbol{o}_t \in \boldsymbol{\Omega}$, depending on the current state and previous action via the conditional observation probability $O(\boldsymbol{o}_t|\boldsymbol{s}_t, \boldsymbol{a}_{t-1})$. By executing an action $\boldsymbol{a}_t \in \boldsymbol{\mathcal{A}}$, the environment will transit into a new state based on the state transition probability $T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, and the agent will receive the reward $\boldsymbol{r}_{t+1} = R(\boldsymbol{s}_t, \boldsymbol{a}_t)$. Same as MDPs, the goal of the agent in this work is to learn an optimal

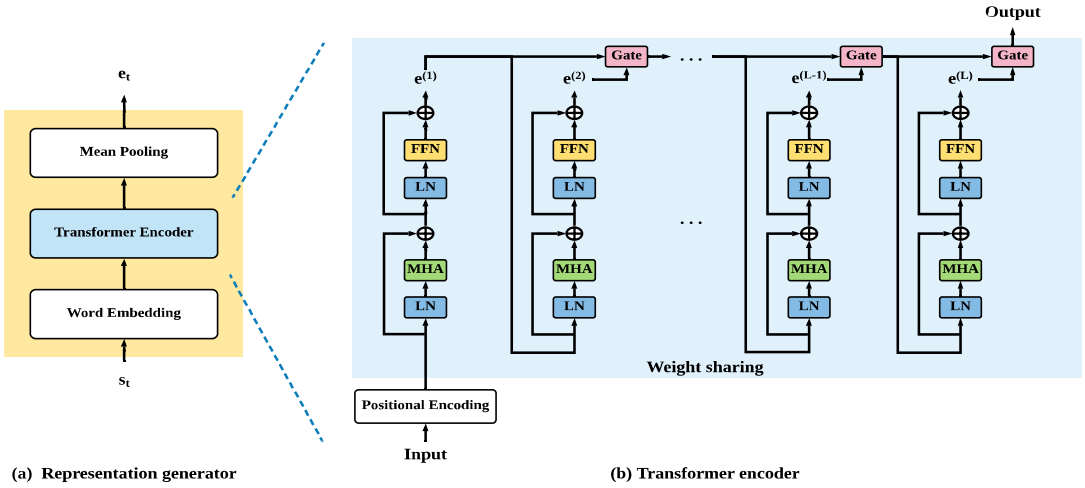

Fig. 2. (a) The proposed representation generator. (b) The modified transformer architecture, where L denotes the number of transformer blocks, LN denotes layer normalization, multi-head attention (MHA) represents multi-head attention sub-module, and feed forward network (FFN) means feed-forward network sub-module. We reorder the layer normalization by filling it within residual connection. All the modules within blue region share the weights. We then propose a block-wise gate layer after each block (except the first block) to aggregate its input and output.

policy π^* to maximize the expected future discounted sum of rewards from each time step: $\boldsymbol{G}_t = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k \boldsymbol{r}_{t+k+1}]$.

We adopt a basic Q-learning framework as the backbone, which is shown in Fig. 1. We use Deep Recurrent Q-Network (DRQN) to solve POMDPs [29]. For simplicity, we denote the input of agent as s_t. The agent consists of two parts: a representation generator, which is our main focus in this work, and an action scorer. The learning objective is to estimate the Q value $Q(s_t, a_t)$, which is the long-term expected return for choosing an action a_t at state s_t. Following the Bellman equation, the Q values can be learned iteratively with the current reward and the max Q value over all actions at next state:

$$Q_{i+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) := Q_i(\boldsymbol{s}_t, \boldsymbol{a}_t) + \boldsymbol{\alpha}(\boldsymbol{r}_{t+1} + \boldsymbol{\gamma} \max_{\boldsymbol{a}' \in \boldsymbol{\mathcal{A}}} Q_i(\boldsymbol{s}_{t+1}, \boldsymbol{a}') - Q_i(\boldsymbol{s}_t, \boldsymbol{a}_t)). \quad (1)$$

where i denotes i-th update step.

Depending on different frameworks and environment settings, the action $a \in A$ for text-based games can be either word combinations (e.g. "go east" is obtained by selecting "go" then "east"), or templates with words (e.g. "take key from table" is obtained by selecting template "take _ from _" first, then selecting word "key" and "table"). Following [1], [7], our policy estimates Q-values for all action components. For example, for template-based action space, we estimate $Q(\boldsymbol{s}, \boldsymbol{u})$ for all templates $\boldsymbol{u} \in \boldsymbol{\mathcal{T}}$ and $Q(\boldsymbol{s}, \boldsymbol{p})$ for all words $\boldsymbol{p}$ in word vocabulary $\boldsymbol{\mathcal{V}}$.

IV. Methodology

As shown in Fig. 2, we consider a transformer-based representation generator for RL algorithms. Unlike vanilla transformer [18] that consists of both encoder and decoder, we apply only the encoder part and refer to it as transformer hereafter. Based on the vanilla transformer, we propose three techniques to improve its performance in RL tasks. First, we reorder the layer normalization to be within the residual connection. Then we share the weights among all transformer blocks. Finally, we add a block-wise gate layer after each block (except the first one) to aggregate the input and output of this block. Similar to blocks, all gate layers share the weights. The output of transformer will be fed into a mean pooling layer to generate vector representation.

A. Normalization layers

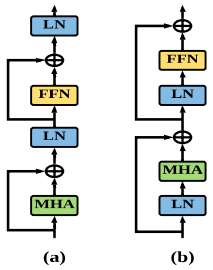

Fig. 3. (a) The vanilla transformer block. (b) The proposed block, where the layer norm is placed within residual connection.

Similar to vanilla transformer, each transformer block in our model consists of two sub-modules: multi-head attention (MHA) and feed forward network (FFN). Our first contribution is layer normalization reordering, which is motivated by the success of QA-NET [19] and TrXL-I [36]. As shown in Fig. 3, we incorporate the layer norm into the residual connection, yielding an identity map between the input and the output of a sub-module.

We denote the number of transformer blocks as L. For the l-th transformer block ($l \in [1, L]$), the input $\boldsymbol{e}^{(l-1)} \in R^{T \times D}$ can be either the state after word embedding and positional encoding (the first block), or the output of previous block

after aggregation, where T is the sequence length and D is the embedding dimension. In MHA sub-module, after layer normalization (LN), the embedding will be used to compute key $\boldsymbol{K}_i \in R^{H\times T\times d}$, query $\boldsymbol{Q}_i \in R^{H\times T\times d}$ and value $\boldsymbol{V}_i \in R^{H\times T\times d}$, where H is the number of heads, i means i-th head, and $d = D/H$. Then $\boldsymbol{K}$, $\boldsymbol{Q}$ and $\boldsymbol{V}$ will be used to compute H scaled dot-product attention in parallel, and be combined as the output embedding $e^{(l)}_{\text{MHA}}$, defined as

$$\boldsymbol{e}^{(l)}_{\text{MHA}} = \text{MHA}(\text{LN}(\boldsymbol{e}^{(l-1)})). \tag{2}$$

Finally, a residual connection will be used to aggregate the output and the identity map of the input:

$$\widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}} = \boldsymbol{e}^{(l-1)} + \text{ReLU}(\boldsymbol{e}^{(l)}_{\text{MHA}}). \tag{3}$$

In FFN sub-module, there are similar operations after LN. The input will be processed by fully connected layers, and then aggregated with its identity map to obtain the output of this block $\boldsymbol{e}^{(l)} \in R^{T\times D}$, defined as

$$\boldsymbol{e}^{(l)} = \widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}} + \text{ReLU}(\text{FFN}(\text{LN}(\widetilde{\boldsymbol{e}^{(l)}}_{\text{MHA}}))). \tag{4}$$

B. Weight sharing

Our second effort is to seek methods to construct a lightweight transformer. The transformer variants used in general NLP tasks are complex with large amounts of learnable weights [20], making it less efficient to optimize. This problem could be more salient in RL, where the dataset is not fixed so that the agent has to spend more interactions to get enough training data. Inspired by ALBERT [38], we apply the weight sharing to reduce the learnable weights, for the purpose of improving parameter efficiency and data efficiency. As shown in Fig. 2 (b), the blue region indicates that we share the weights among all transformer blocks. In other words, this modification makes the representation extraction process along L blocks similar to extracting information recurrently for L times by only one block. In this way, we can increase the depth of model while keeping the number of weights friendly to optimize. In addition to the transformer blocks, we also share the weights among block-wise gate layers.

C. Block-wise gate layer

Besides the modifications within the transformer block, we further enhance residual connection via block-wise aggregation operations after each block (except the first one). We get insight from the GTrXL [36], where gating methods are used to substitute the adding aggregation in the block to enhance learning stability. Different from their work, we keep the adding operation and apply the gating mechanism between the blocks. Specifically, we apply a gated output connection, where the weight factor computed from the input stream is assigned to the output stream. For the l-th block with input stream $\boldsymbol{e}^{(l-1)}$ and output stream $\boldsymbol{e}^{(l)}$, the output of gate layer is computed as:

$$\boldsymbol{e}^{(l)} = \boldsymbol{e}^{(l-1)} + \sigma(\boldsymbol{W}_g\boldsymbol{e}^{(l-1)} + \boldsymbol{b}_g) \odot \boldsymbol{e}^{(l)}, \tag{5}$$

where $\boldsymbol{W}_g$ and $\boldsymbol{b}_g$ are learnable variables, and σ is the Sigmoid function. We restrict that both the input and the output have been processed by at least one block. Consequently, there is no gate layer after the first block.

V. Experiments

A. Experiment setting

We evaluate our model on two text-based game domains: procedurally-generated games supported by TextWorld's CoinCollector Challenge [2] and man-made games supported by Jericho [29]. Each game in CoinCollector domain contains a number of randomly connected rooms, where a coin is placed in one of them, and the goal is to find and pick up this coin within the step limit. The tasks are associated with sparse reward so that the reward will be 1.0 only if the coin is collected successfully, otherwise 0.0. An episode will be terminated if the coin is collected or the step limit is exceeded. The difficulty depends on the number of rooms, the quest length (i.e., the length of optimal trajectory) and the number of distractor rooms leading to dead ends. We define a game by its *level* and *difficulty mode*, where the level denotes the length of the optimal trajectory. We consider two difficulty modes: Easy − no distractor room, and Hard − each room in the optimal path has two distractor rooms. For example, "L30E" denotes level 30, mode easy.

The games generated by CoinCollector are relatively simple for initial study and parameter tuning because the quest length is customizable and the size of action set is small. Besides, the generated games share the same goal and the same solving strategy (with different layouts such as room connectivities), making it feasible to evaluate the generalizability of the proposed method on unseen games. We thus perform two branches of experiments in this domain: single game and multiple unseen games. The single game is similar to general RL settings, where the agent will be trained and validated on the same game. In terms of multiple unseen games, we generate two non-overlapping sets of games for training and evaluation, respectively. Table I summarizes the games. For each branch, we conduct experiments with varied difficulties.

TABLE I
THE DETAILS OF GENERATED GAMES IN COINCOLLECTOR DOMAIN.

Task name	L30E	L30H	L40E
Rooms	30	90	40
Quest length	30	30	40
Single	√	√	
Multiple unseen	√		√

We also conduct experiments on a variety of classic man-made games. These games have much larger state space and action space, involving more complex logic. For example, the game zork1 contains 697 words in the action vocabulary such that a four-word-action consists of $|697|^4$ choices, while the walkthrough contains 130 actions used in 345 steps. Current agents, even when being integrated with huge external knowledge and engineering tricks, are still far from being able to solve most of these games [1], [6], [28], [30]. In this work,

we aim to provide our transformer-based state representation generator as an easy-to-use plugin to help existing and future agents to achieve better performance.

B. Baselines

For CoinCollector domain, we use LSTM-DRQN [7] as the backbone module[1] and name our proposed model as Trans-v-DRQN. We construct the following three baselines:

- LSTM-DRQN [7]: LSTM-based representation generator.
- CNN-DRQN: CNN-based representation generator modified from CNN-DQN [30].
- Trans-DRQN: Vanilla transformer-based representation generator with parameter tuning only.

For Jericho domain, we use DRRN [16] as the backbone module [2] and name our model as Trans-v-DRRN. We construct the following three baselines:

- DRRN [16]: an extension of LSTM-DQN where the valid actions are presented at every state.
- TDQN [1]: an extension of LSTM-DQN with template-based action spaces.
- KG-A2C[3] [17]: an extension of TDQN combined with knowledge graph and auxiliary task of valid action prediction.

All of the models are trained from scratch. We leave the combination with pre-trained language models as a part of future work.

C. Implementation details

For CoinCollector domain, the hyper-parameters of LSTM-DRQN are same as those in the original paper [7]. Both Trans-DRQN and Trans-v-DRQN are modified based on the transformer example in Pytorch's documentation[4], where we simplify the encoder by reducing the word embedding dimension, the number of heads in MHAs, the dimension of the linear layer in FFNs, as well as the number of repeated blocks. Considering that the word vocabulary size is 1250 and the word embedding size is 20, which are relatively small, we use single-head self-attention in the blocks. The FFN part consists of a fully-connected layer with the hidden dimension of 100. The Trans-v-DRQN contains 4 blocks while the Trans-DRQN achieves better performance with 2 blocks. The encoder part of CNN-DRQN is modified from [30] so that it contains 3 convolutional layers with kernel size of 3, 4, 5, respectively. Following the original paper [7], all models apply the same DRQN action scorer and the episodic discovery bonus. The 1.0 reward bonus will be assigned if the current state is seen for the first time.

For Jericho domain, the hyper-parameters of DRRN, TDQN and KG-A2C are taken from [1], [17]. The architecture of Trans-v-DRRN is similar to DRRN, except that we replace the 3 GRUs, which are used for processing the narrative text, the players inventory and the location description, with a shared transformer encoder equipped with proposed modifications. The word embedding size is 128 and the transformer is configured with 4 heads, 4 blocks and 128 hidden dimension.

D. Training details

For CoinCollector domain, all models use Adam optimizer with a learning rate of 0.001 and a gradient clip of 5. We apply prioritized replay buffer with size 500000, where the batch size during updating is 32. For all games, we set step limit as 50 and denote an epoch as the model running on each environment for one episode. For single games, we generate 10 same environments. We use the ϵ-greedy method to select the action, where ϵ is the probability to choose a random action and it anneals from 1.0 to 0.2 in 1000 epochs during the training. For multiple unseen tasks, we follow similar settings in [5] to generate a training set of 200 games and a testing set of 20 games, where one environment is generated for each game (i.e., 200 episodes per epoch), and the ϵ anneals from 1.0 to 0.2 in 50 epochs. All the experiments are run with 3 random seeds.

For Jericho domain, we select 15 games and set step limit as 100 valid steps or game over/victory. Similar to the baselines, we tune the hyper-parameters on zork1, then hold them fixed across other games. For each game, we generate 8 environments in parallel and train Trans-v-DRRN for 100000 update steps. We use Adam optimizer with a learning rate of 0.0001 and a gradient clip of 5. We apply replay buffer with size 500000, where the batch size during updating is 64. The setting for baselines is same as original implementation. All of the experiments are run with 5 random seeds.

E. Evaluation metrics

Same as the existing RL-related works, we measure all games with episodic sum of rewards. For Jericho domain, we report the average result of last 100 episodes. For single games in CoinCollector domain, we also report the number of training epochs required to achieve maximum reward sum (i.e., 1.0).

VI. Results and discussions

A. CoinCollector: single game

Fig. 4 shows the performance of single games in CoinCollector domain. While the learning process of LSTM-DRQN is slow and lacking of stability, the CNN-based and transformer-based models achieve higher learning speed. Since all the compared models succeed to reach the maximum reward sum, we report the number of epochs required to achieve this. As shown in Table II, the proposed Trans-v-DRQN requires the fewest number of interactions among all models, showing high data efficiency. The advantage is more obvious in hard tasks with distractor rooms. A possible reason is that the modified transformer helps to extract key information to recognize and understand the state, such as the previous action and the direction related words in description.

To systematically validate our model, we perform ablation study to investigate the importance of each of the three

[1]https://github.com/xingdi-eric-yuan/TextWorld-Coin-Collector

[2]Both DRRN and TDQN are based on https://github.com/microsoft/tdqn

[3]https://github.com/rajammanabrolu/KG-A2C

[4]https://pytorch.org/docs/stable/_modules/torch/nn/modules/transformer.html

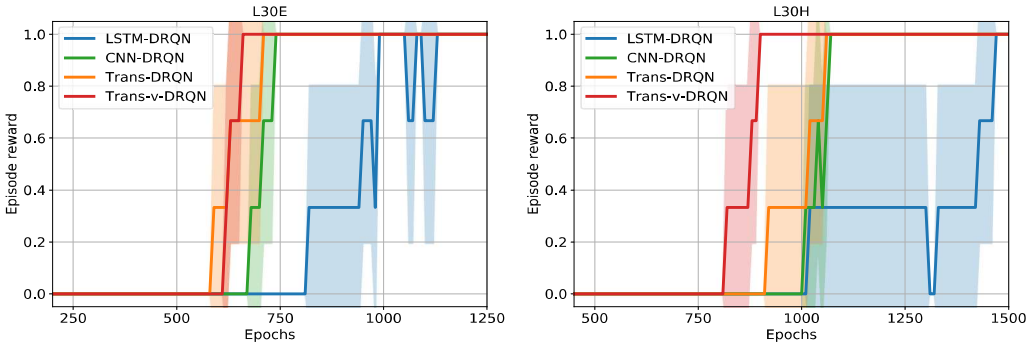

Fig. 4. The performance of models on CoinCollector: single games.

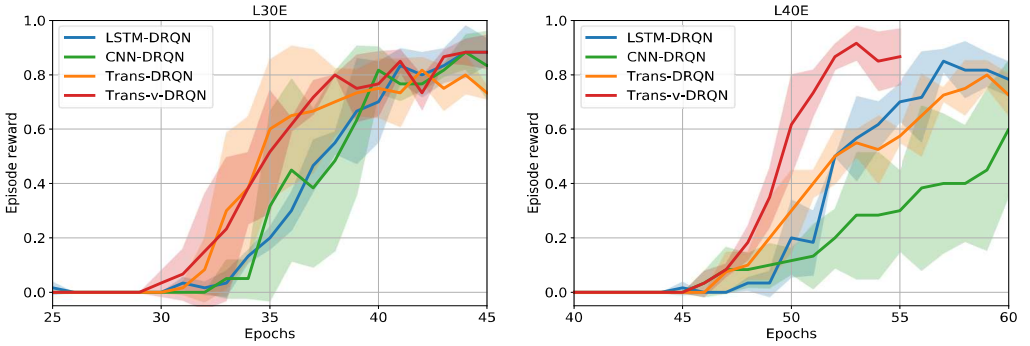

Fig. 5. The performance of models on CoinCollector: multiple unseen tgames. In experiment "L40E", we early stop "Trans-v-DRQN" at epoch 55 since it has reached high performance before that.

TABLE II
THE REQUIRED EPOCHS TO REACH THE MAXIMUM REWARD SUM.

Model	L30E	L30H
LSTM-DRQN	920.00 ± 72.57	1306.67 ± 203.36
CNN-DRQN	710.00 ± 24.49	1040.00 ± 24.49
Trans-DRQN	643.33±49.89	1000.00 ± 58.88
Trans-v-DRQN	**636.67 ± 17.00**	**866.67 ± 33.99**

TABLE III
THE PERFORMANCE OF MODELS WITH DIFFERENT MODIFICATIONS.

Model	L30H
Trans-DRQN	1000.00 ± 58.88
Trans-v-DRQN (full)	**866.67 ± 33.99**
-gate	920.00 ± 29.44
-gate -reorderLN	940.00 ± 58.88
-gate -shareW	1050.00 ± 92.74
-gate -reorderLN -shareW	1016.67 ± 37.71

TABLE IV
THE MAXIMUM REWARDS ACHIEVED IN MULTIPLE UNSEEN GAMES.

Model	L30E	L40E
LSTM-DRQN	0.88 ± 0.09	0.85 ± 0.04
CNN-DRQN	0.88 ± 0.06	0.60 ± 0.25
Trans-DRQN	0.82 ± 0.05	0.80 ± 0.05
Trans-v-DRQN	**0.88 ± 0.06**	**0.92 ± 0.06**

proposed modifications. The results are reported in Table III, where "-X" means that the modification of "X" is removed from Trans-v-DRQN. We observe that weight sharing contributes the most – the performance drops most significantly if removing this technique (e.g., comparing "-gate" *v.s.* "-gate -shareW", and "-gate -reorderLN" *v.s.* "-gate -reorderLN -shareW"). The block-wise gate layer also helps to improve performance (e.g., comparing "Trans-v" *v.s.* "-gate"). Our model is benefited least from layer normalization reordering. Slight improvement can be observed when it's applied together with weight sharing (e.g., comparing "-gate -reorderLN" *v.s.* "-gate"). However, the performance even degrades when it is applied alone (e.g., comparing "-gate -shareW" *v.s.* "-gate -reorderLN -shareW").

B. *CoinCollector: multiple unseen games*

Fig. 5 shows the generalization performance for multiple unseen games in CoinCollector domain. For "L30E" tasks, the transformer based models make progress faster than LSTM and CNN based models. However, the performance of Trans-DRQN starts to drop after the 40th epoch and eventually fails to reach the levels of other models. One possible reason is that, the vanilla transformer, when being deployed as a

TABLE V
THE PERFORMANCE ACROSS 15 JERICHO SUPPORTED GAMES.

Game	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{V}\rvert$	TDQN	KG-A2C	DRRN	Trans-v-DRRN	MaxReward
acorncourt	151	343	1.6	0.3	10	**10**	30
adventureland	156	398	0	0	20.6	**25.6**	100
affilicted	146	762	1.3	-	2.6	2.0	75
detective	197	344	169	207.9	197.8	**288.8**	360
enchanter	290	722	8.6	12.1	20.0	**20.0**	400
library	173	510	6.3	14.3	17	**17**	30
ludicorp	187	503	6	**17.8**	13.8	16	150
pentari	155	472	17.4	**50.7**	27.2	34.5	70
reverb	183	526	0.3	-	8.2	**10.7**	50
spellbrkr	333	844	18.7	21.3	37.8	**40**	600
temple	175	622	7.9	7.6	7.4	**7.9**	35
tryst205	197	871	0	-	9.6	**9.6**	350
zork1	237	697	9.9	34	32.6	**36.4**	350
zork3	214	564	0	0.1	**0.5**	0.19	7
ztuu	186	607	4.9	9.2	**21.6**	4.8	100

representation generator, is prone to be overfitting which leads to performance drop in unseen tasks. For "L40E" tasks with longer quest length, the advantage gained by Trans-v-DRQN becomes more significant. While its score starts to exceed 0.6 at the 50th epoch and finally goes over 0.8, the baselines learn much slower (one epoch contains approximately 10,000 interaction steps) and show worse performance. Additionally, we observe that while CNN-DRQN performs the worst and fails to reach convergence within 60 epochs, our model as well as other two baselines encounters performance drop after reaching the highest performance, which is probably caused by overfitting. Table IV reports the maximum scores that can be be achieved by the compared models, which shows again the proposed Trans-v-DRQN performs best.

C. Jericho-supported games

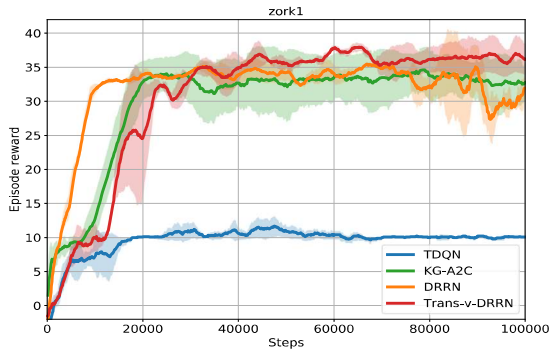

Fig. 6. The performance with respect to update steps on zork1.

Table V shows the performance across 15 man-made games in Jericho domain, where $|\mathcal{T}|$ and $|\mathcal{V}|$ denote the number of templates (for template-based methods such as TDQN and KG-A2C) and the size of action vocabulary. Our model, Trans-v-DRRN, achieve the highest score in 10 games. When compared with the backbone model DRRN, which is already a strong baseline in Jericho domain, our model brings performance improvement in 8 games. In another 4 games, Trans-v-DRRN's performance is comparable with DRRN by achieving equivalent results. We further compare the learning curve for zork1. As shown in Fig. 6, while Trans-v-DRRN starts to make progress a little slower than DRRN and KG-A2C, its performance keeps increasing and becomes the highest at the end of learning. Besides, we observe that the backbone model DRRN encounters performance fluctuation along with the progress of the learning process. The integration of our transformer-based representation generator helps to overcome this problem, which in turn boosts the performance.

VII. CONCLUSION

In this paper, we investigated how the transformer architecture could be used for enhancing deep reinforcement learning in solving text-based games. We focused on building state representation and introduced a lightweight transformer-based representation generator featured with layer normalization reordering, weight sharing and block-wise gate layers. The experimental results on generated games show that our model not only solves the single task with fewer interactions, but also achieves better generalizability in solving unseen tasks. Furthermore, our model outperforms current state-of-the-art agents in a variety of man-made games. With respect to the future work, we would like to focus on the zero/few-shot generalizability on more complex tasks. While current model is trained from scratch, in the future we would also like to combine our model with pre-trained language models [20], [38] to obtain better commonsense reasoning ability, thus further exploit its potential.

ACKNOWLEDGMENT

This work is partly supported by ARC Discovery Project DP180100966. Yang Wang is supported by the National Natural Science Foundation of China under Grant 61806035, Grant U1936217.

REFERENCES

[1] M. Hausknecht, P. Ammanabrolu, M.-A. Côté, and X. Yuan, "Interactive fiction games: A colossal adventure," *arXiv preprint arXiv:1909.05398*, 2019.

[2] M.-A. Côté, Á. Kádár, X. Yuan, B. Kybartas, T. Barnes, E. Fine, J. Moore, M. Hausknecht, L. E. Asri, M. Adada *et al.*, "Textworld: A learning environment for text-based games," 2018.

[3] J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel, "A survey of reinforcement learning informed by natural language," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 6309–6317. [Online]. Available: https://doi.org/10.24963/ijcai.2019/880

[4] K. Narasimhan, T. D. Kulkarni, and R. Barzilay, "Language understanding for text-based games using deep reinforcement learning," in *Conference on Empirical Methods in Natural Language Processing, EMNLP 2015*. Association for Computational Linguistics (ACL), 2015, pp. 1–11.

[5] P. Ammanabrolu and M. Riedl, "Playing text-adventure games with graph-based deep reinforcement learning," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 3557–3565.

[6] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3562–3573.

[7] X. E. Yuan, M.-A. Côté, A. Sordoni, R. Laroche, R. Tachet des Combes, M. Hausknecht, and A. Trischler, "Counting to explore and generalize in text-based games," in *European Workshop on Reinforcement Learning (EWRL)*, October 2018. [Online]. Available: https://www.microsoft.com/en-us/research/publication/counting-to-explore-and-generalize-in-text-based-games-2/

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop*, 2013.

[9] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[10] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik *et al.*, "Deepmind lab," *arXiv preprint arXiv:1612.03801*, 2016.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[12] M. Fang, C. Zhou, B. Shi, B. Gong, J. Xu, and T. Zhang, "DHER: Hindsight experience replay for dynamic goals," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Byf5-30qFX

[13] M. Fang, T. Zhou, Y. Du, L. Han, and Z. Zhang, "Curriculum-guided hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2019, pp. 12 602–12 613.

[14] K. Narasimhan, A. Yala, and R. Barzilay, "Improving information extraction by acquiring external evidence with reinforcement learning," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2355–2365.

[15] M. Fang, Y. Li, and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 595–605.

[16] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf, "Deep reinforcement learning with a natural language action space," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1621–1630.

[17] P. Ammanabrolu and M. Hausknecht, "Graph constrained reinforcement learning for natural language action spaces," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=B1x6w0EtwH

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[19] A. W. Yu, D. Dohan, Q. Le, T. Luong, R. Zhao, and K. Chen, "Fast and accurate reading comprehension by combining self-attention and convolution," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B14TlG-RW

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners."

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in neural information processing systems*, 2015, pp. 2377–2385.

[25] T. Atkinson, H. Baier, T. Copplestone, S. Devlin, and J. Swan, "The text-based adventure ai competition," *IEEE Transactions on Games*, 2019.

[26] N. Fulda, D. Ricks, B. Murdoch, and D. Wingate, "What can you do with a rock? affordance extraction via word embeddings," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 1039–1045. [Online]. Available: https://doi.org/10.24963/ijcai.2017/144

[27] B. Kostka, J. Kwiecieli, J. Kowalski, and P. Rychlikowski, "Text-based adventures of the golovin ai agent," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2017, pp. 181–188.

[28] M. Hausknecht, R. Loynd, G. Yang, A. Swaminathan, and J. D. Williams, "Nail: A general interactive fiction agent," *arXiv preprint arXiv:1902.04259*, 2019.

[29] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.

[30] X. Yin and J. May, "Comprehensible context-driven text game playing," *2019 IEEE Conference on Games (CoG)*, pp. 1–8, 2019.

[31] P. Ammanabrolu and M. O. Riedl, "Transfer in deep reinforcement learning using knowledge graphs," *arXiv preprint arXiv:1908.06556*, 2019.

[32] M. Zelinka, X. Yuan, M.-A. Côté, R. Laroche, and A. Trischler, "Building dynamic knowledge graphs from text-based games," *arXiv preprint arXiv:1910.09532*, 2019.

[33] Y. Liu and M. Lapata, "Hierarchical transformers for multi-document summarization," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 5070–5081. [Online]. Available: https://www.aclweb.org/anthology/P19-1500

[34] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser, "Universal transformers," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HyzdRiR9Y7

[35] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals, and P. Battaglia, "Deep reinforcement learning with relational inductive biases," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HkxaFoC9KQ

[36] E. Parisotto, H. F. Song, J. W. Rae, R. Pascanu, C. Gulcehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury *et al.*, "Stabilizing transformers for reinforcement learning," *arXiv preprint arXiv:1910.06764*, 2019.

[37] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1DmUzWAW

[38] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.