Data Augmentation for Learning to Play in Text-Based Games

Jinhyeon Kim[1,2] and **Kee-Eung Kim**[1]
[1] Kim Jaechul Graduate School of AI, KAIST
[2] Skelter Labs
jhkim@ai.kaist.ac.kr, kekim@kaist.ac.kr

Abstract

Improving generalization in text-based games serves as a useful stepping-stone towards reinforcement learning (RL) agents with generic linguistic ability. Data augmentation for generalization in RL has shown to be very successful in classic control and visual tasks, but there is no prior work for text-based games. We propose *Transition-Matching Permutation*, a novel data augmentation technique for text-based games, where we identify phrase permutations that match as many transitions in the trajectory data. We show that applying this technique results in state-of-the-art performance in the *Cooking Game* benchmark suite for text-based games.

Observation	-= Kitchen =- You've just shown up in a kitchen. You make out a closed fridge in the corner. You make out a counter. The counter is vast. On the counter you can see a cookbook and a knife. [...]
Action	**> open fridge**
Observation	You open the fridge, revealing a carrot.
Action	**> take carrot**
Observation	You take the carrot from the fridge. Your score has just gone up by one point.

Figure 1: An example gameplay from the text-based game, Cooking Game. Both the observation and action are in the form of natural language.

1 Introduction

The *text-based game*, a game played through a text-based user interface, serves as a promising testbed towards building an intelligent agent that learns how to interact with the environment through natural language. In order to play text-based games, the agent needs to understand the state of the game via natural language processing (NLP), as well as make appropriate action choices via reinforcement learning (RL) [Hausknecht *et al.*, 2020]. An example gameplay from a text-based game is shown in Figure 1.

In order to ultimately build RL agents with generic linguistic ability, one must tackle the problem of generalizability. In this context, Côté *et al.* [2018] proposed a procedurally generated text-based game framework called *TextWorld* to help the research on generalization for text-based games. Using this framework, a number of techniques have been proposed to integrate the observations into more generalizable representations; Adhikari *et al.* [2020] construct a belief graph from the observations using pre-trained graph neural networks and Chaudhury *et al.* [2020] devised a pruning method that captures essential features.

On the other hand, data augmentation is a highly effective technique for improving generalizability when training machine learning models. Cobbe *et al.* [2019] report that data augmentation is one of the crucial regularization techniques required for generalizing well via extensive experiments conducted on procedurally generated RL environments. However, to the best of our knowledge, no prior work exists for data augmentation in text-based games.

In this paper, we propose a novel data augmentation technique for text-based games, *Transition-Matching Permutation*, which permutes the actions and observations while preserving equivalence to the original game dynamics. Utilizing a set of pre-collected trajectory data, the method does not require any other extra information or instrumentation of the game engine. The method first discovers phrase pairs that exhibit similarity in the transitions with the respective phrases from trajectory data. We regard the permutations of such phrase pairs as capturing the *symmetry* in the environment. Once the permutations are identified, we randomly sample a permutation at the beginning of every episode in the RL loop as the data augmentation. This permutation is used to transform observations and actions, effectively creating a randomly modified, yet equivalent game instance from the original environment.

We experiment on top of the state-of-the-art RL algorithms for text-based games, and show that our data augmentation techniques further improve their generalization performance. Moreover, we observe that even the simple DQN with a straightforward pre-processing of observations can achieve state-of-the-art performance in the TextWorld's *Cooking Game* benchmark [Côté *et al.*, 2018; Adhikari *et al.*, 2020]. The code is publicly available at https://github.com/KAIST-AILab/transition-matching-permutation.

2 Background

2.1 Partially Observable Markov Decision Process (POMDP)

In many real-world environments for RL, including text-based games, the agent can access only a portion of the environment state. Partially observable Markov decision processes (POMDP) are an extension of MDPs for these partially observable environments. Formally, a POMDP is a 7-tuple $(S, A, T, R, \Omega, O, \gamma)$, where

- S is a set of states,
- A is a set of actions,
- $T(s'|s, a)$ is a transition probability distribution,
- $R : S \times A \to \mathbb{R}$ is the reward function,
- Ω is a set of observations,
- $O(o|s, a)$ is an observation probability distribution,
- $\gamma \in [0, 1]$ is the discount factor.

At each time step $t = 0, 1, 2, \ldots$, the agent takes an action $a_t \in A$ in the environment state $s_t \in S$, which transitions the environment state to s_{t+1} with probability distribution $T(s_{t+1}|s_t, a_t)$ and yields a reward $r_t = R(s_t, a_t)$ and an observation $o_{t+1} \in \Omega$ with probability distribution $O(o_t|s_t, a_{t-1})$. The goal is to maximize the expected cumulative reward $\mathbb{E}\left[\sum_t \gamma^t r_t\right]$.

Unlike in an MDP, the information of the underlying state in a POMDP is partially and indirectly provided through observations. Hence, the agent must accumulate the history of observations for inferring an estimate of the state or the *belief state*.

2.2 Permutable POMDP

Doshi and Roy [2008] recognized that a POMDP environment with an inherent symmetry can reduce its computational complexity. The symmetry is described by state permutations $\rho : S \to S$ such that there exist their corresponding action permutation $\sigma_\rho^{act} : A \to A$ and observation permutation $\sigma_\rho^{obs} : \Omega \to \Omega$ satisfying the following conditions,

$$\begin{aligned} T(s'|s, a) &= T(\rho(s')|\rho(s), \sigma_\rho^{act}(a)), \\ O(o|s, a) &= O(\sigma_\rho^{obs}(o)|\rho(s), \sigma_\rho^{act}(a)), \text{and} \\ R(s, a) &= R(\rho(s), \sigma_\rho^{act}(a)), \end{aligned}$$

for all $s, s' \in S, a \in A, o \in \Omega$.

In the era of deep reinforcement learning, it is often unclear how to incorporate the symmetry into the deep neural networks. Therefore, based on this formulation, we will utilize the permutations that describe the environment symmetry for data augmentation. Unfortunately, the set of states S and the models T, O, and R are not directly accessible under the RL setting. Hence, we will address this restriction by leveraging the trajectory data, as we will explain in Section 5.2.

3 Related Works

3.1 Generalization in Text-Based Games

The *text-based game* is a game that is played through a text-based user interface. Many popular classic computer games, such as *Zork* and *Rogue*, are examples of text-based games. Learning how to play in text-based games lends an insight on how to interact with the environment through the use of natural language.

Traditional RL settings assume that the agent is trained and tested in the same environment. In a more realistic scenario, however, one must expect to encounter unseen circumstances. To this end, we consider the setting where an agent is trained in one set of games and evaluated in another unseen set of games. Without a proper treatment, the agent is tempted to memorize the best-performing strategies specific to each particular training game instances rather than learning a general policy that explores for such information.

Adhikari *et al.* [2020] and Chaudhury *et al.* [2020] tackle the generalizability problem of text-based games by building a more generalizable representation of the belief state from the observations, discouraging the agent from identifying the training game instance. On top of these pioneering works, we will take an orthogonal approach of *data augmentation* to further boost their generalization capabilities.

Graph-Aided Transformer Agent

Adhikari *et al.* [2020] criticized the use of heuristics of prior works for building belief states and proposed *Graph-Aided Transformer Agent* (GATA), which infers the belief state in an unsupervised manner. Specifically, GATA models the belief state as a graph representing the inferred attributes of the entities as well as their relationships. The agent uses this belief graph as the input.

Updating the belief graph from text observations is done by a pre-trained relational graph convolutional network (R-GCN) [Schlichtkrull *et al.*, 2018]. They use the following pre-training schemes:

- Observation generation (OG), trained to minimize the reconstruction loss between the original observation and the observation reconstructed from the predicted belief graph and the previous action.
- Contrastive Observation Classification (COC), trained to minimize the contrastive loss by differentiating the original observation from random observations, given the predicted belief graph and the previous action.

Context Relevant Episodic State Truncation

Chaudhury *et al.* [2020], on the other hand, propose *Context Relevant Episodic State Truncation* (CREST), which prunes the observation so that it only contains information relevant to the gameplay. The main motivation for CREST is that the presence of irrelevant information facilitates the identification of the training game instance, leading to the overfitting.

To identify which tokens are relevant to the gameplay, CREST first trains an agent with raw observation. This agent is expected to generate a well-performing trajectory specific to each training game instance but may not generalize to other game instances. Let K denote the number of training game instances and $\mathcal{T}_k$ denote the trajectory of this agent for k-th game, $k = 1, \ldots, K$. Let $\mathcal{V}$ denote the vocabulary set and $\mathcal{V}^k \subset \mathcal{V}$ denote the tokens used in the actions of $\mathcal{T}_k$. Note that the tokens in $\mathcal{V}^k$ are essential for playing k-th game. Hence,

any words that are similar to these tokens can be considered *relevant* to the gameplay.

Given a word similarity function $D : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$, CREST defines *Token Relevance* of words $w \in \mathcal{V}$,

$$\mathcal{C}(w; \mathcal{V}^k) \doteq \max_{v \in \mathcal{V}^k} D(w, v).$$

They prune an observation of k-th training game by removing tokens w with token relevance $\mathcal{C}(w; \mathcal{V}^k)$ below a certain threshold θ_{CREST}. The final agent is trained on this pruned observation, encouraging generalization by curating only the relevant information. For evaluation, they prune the tokens according to the global token relevance $\mathcal{C}(w; \bigcup_k \mathcal{V}^k)$. For our purpose, we use the global token relevance for training as well to reduce the train-test distribution shift and simplify implementation.

3.2 Data Augmentation

Data augmentation is a generic technique that applies a set of modifications to generate more training instances from the original data to increase the size and the diversity of the dataset. It is one of the most popular and effective approaches for improving generalization when training machine learning models.

Cobbe *et al.* [2019] report that data augmentation is one of the crucial regularization techniques required for generalizing well in RL. RAD [Laskin *et al.*, 2020] and DrQ [Yarats *et al.*, 2020] devise sample-efficient methods to incorporate data augmentation techniques into RL training loop. However, all these works have focused on classical control and visual tasks, and to the best of our knowledge, no prior works exist for data augmentation in RL for text-based games.

Data augmentation is also actively explored in building dialogue systems, where the dialogue corpus is enriched with synthesized data to improve generalization. Hou *et al.* [2018] proposed paraphrasing delexicalized sentences to generate synthesized utterances. Campagna *et al.* [2020] designed a dialogue synthesis scheme for domain transfer using ontology. These works have shown improvement in generalization performance under limited training data. Compared to our work in this paper, these data augmentation methods deal with more complex and diverse utterances in human dialogues, while relying on manually tagged annotations to synthesize semantically coherent utterances. Our data augmentation approach deals with more regular sentences generated by the game script, while being agnostic to the game engine and working with raw text.

4 Tasks

TextWorld [Côté *et al.*, 2018] is a framework, based on Jericho [Hausknecht *et al.*, 2020], that allows procedural generation of text-based games of varying difficulty and features. It serves as a useful tool for studying generalization in text-based games.

In particular, we use the *Cooking Game* suite, which is one of the standard sets of games provided by TextWorld. The goal of the *Cooking Game* is to collect and process all the necessary objects, called *ingredients*, placed across different rooms. The player should discover a `cookbook` that contains the recipe describing the necessary ingredients and the exact methods for processing them. A reward of +1 is given for gathering each necessary ingredient and for each step of correctly processing the ingredients; in all other cases, the reward is zero. The game ends either when the recipe is completed or when any ingredients are processed incorrectly. Note that inspecting the `cookbook` does not earn a reward directly. Below are some of the actions in the game:

- `examine` <obj> provides information related to the object. Especially, the content of the recipe is retrieved by the action `examine cookbook`.
- `go north` (and so on) navigate in the specified direction.
- `take` <obj> `(from` <obj>) lets the player carry it.
- `chop/dice/slice/cook` <obj> `with` <obj> process the ingredient in the respective manner.

The *Cooking Game* can be configured for the procedural generation via specifying the following settings:

- The number of ingredients in the recipe (Recipe size)
- The number of locations in the game (#Locations)
- Whether some ingredients require cutting (Need cut)
- Whether some ingredients require cooking (Need cook)

Defined on top of the *Cooking Game* suite, the *First TextWorld Problems* (FTWP) competition [Trischler *et al.*, 2019] encouraged generalization over unseen set of games. They split the processing methods (i.e. cooking and cutting) for each ingredient into training, validation, and test set. For instance, a training game instance may require carrot to be `sliced` or `chopped` but never `diced`, which may appear in the validation or test game instance.

Adhikari *et al.* [2020] proposed another benchmark from the *Cooking Game* suite. They procedurally generated four sets of game instances of varying difficulty levels, independent from the FTWP setup, as described by Table 1. It is notable that, when generating the new benchmark, they used only the FTWP's training split of the processing methods for all of training, validation, and test set games. Therefore, in this configuration, a suboptimal policy that memorizes the processing methods for each ingredient from the

	Level 1	Level 2	Level 3	Level 4
Recipe Size	1	1	1	3
#Locations	1	1	9	6
Max Score	4	5	3	11
Need Cut	Yes	Yes	No	Yes
Need Cook	No	Yes	No	Yes

Table 1: Configuration of the *Cooking Game* benchmark.

Level 1	Level 2	Level 3	Level 4
66.2	36.0	58.3	27.7

Table 2: A summary of the best normalized test scores on the *Cooking Game*, reported by Adhikari *et al.* [2020].

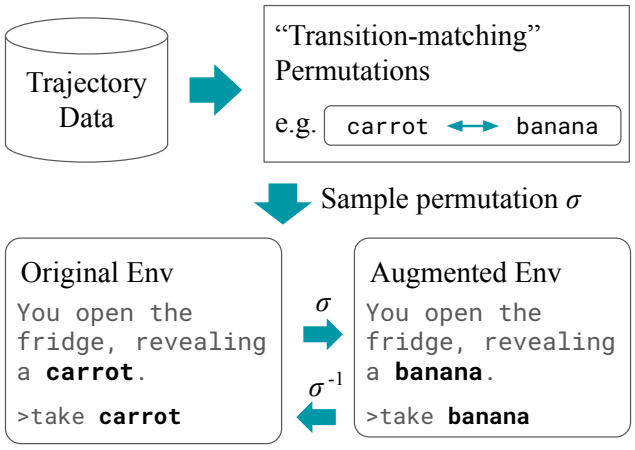

Figure 2: An illustration of our data augmentation technique, *Transition-Matching Permutation*.

training set may perform well in the validation and the test game instances, although it is more appropriate to consult the `cookbook` before processing the ingredients. Indeed, we observed that the baseline methods often converged to this suboptimal policy, which we discuss more in the experiments section. Despite the issue, we will use this particular configuration as the *Cooking Game* benchmark for the sake of reproducing the results from the baseline methods.

Table 2 summarizes the best of the test scores from Adhikari *et al.* [2020]. We regard these scores as the previous state-of-the-art on the *Cooking Game* benchmark.

5 Data Augmentation via Transition-Matching Permutation

We consider data augmentations of the following form. Given an observation and a set of admissible actions, we perform certain transformation on them before presenting them to the agent. When the agent selects one of the transformed actions, we perform the original action that is associated with it on the original environment. To ensure the soundness, we restrict the transformation to be injective. Otherwise, we may have two actions a_1 and a_2 that are mapped to a single augmented action a'. When the agent chooses to act a', it becomes unclear whether a_1 or a_2 should be performed on the original environment.

To this end, we adopt the Permutable POMDP formulation. We consider the case where the difference in two states can be identified via the differences of certain phrases in the textual observations and actions. In such cases, the observation and action permutation corresponding to the state permutation are text replacement of such phrases. For example, if one game has a "carrot" and another game has a "banana" with other things being equal, the observation and action permutation are text replacement operation from "carrot" to "banana" and vice versa, as shown in Figure 2.

Before delving into details, we remark that the following information is available, as in Adhikari *et al.* [2020]:

- The trajectory data $\mathcal{T}$, obtained by issuing 10 random actions on each step of ground-truth trajectories of the *First TextWorld Problems* games. It was utilized to pretrain GATA.
- The admissible actions for both the *First TextWorld Problems* games and the *Cooking Game* benchmark.

5.1 Candidate Phrase Extraction

First, we extract candidate phrases that will be used as building blocks for permutations. The key idea here is that we generate symmetric versions of the game by replacing phrases of the observations and the actions by candidate phrases, e.g. replace `carrot` by `banana`, `front door` by `plain door`, etc. Without access to the game engine, we need to identify these candidate phrases from the observations and the actions. However, we found that the observations may contain many phrases not relevant to the gameplay. For instance, the sentence `The counter is vast.` in Figure 1 is not useful for playing the game. The actions, on the contrary, are always relevant to the gameplay. Therefore, we search for candidate phrases from the actions.

Let $\mathcal{V}$ denote the vocabulary set and $\mathcal{V}^*$ the set of all word sequences in $\mathcal{V}$. Let $\tilde{A} \subset \mathcal{V}^*$ denote the set of all the actions in the trajectory data $\mathcal{T}$. For instance, $\tilde{A}$ may contain actions such as `go south` and `take knife from table`. We extract all the noun phrases $E \subset \mathcal{V}^*$ found in the actions of $\tilde{A}$ using SpaCy [Honnibal *et al.*, 2020]. For the action `take knife from table`, the phrases `knife` and `table` are extracted. We use the extracted phrases E as the candidate phrases for the remaining steps.

5.2 Filtering from Trajectory Data

Next, we use the extracted phrases to define permutations. Specifically, for candidate phrases $x, y \in E$ and a word sequence $z \in V^*$, let $\sigma_{x,y}(z)$ denote the operation that replaces all the occurences of x found in z with y and vice versa. For example, $\sigma_{\texttt{carrot,banana}}$ maps an observation "The fridge contains a carrot and a banana" to "The fridge contains a banana and a carrot." For every candidate phrase pairs $x, y \in E$, we want to verify if the mapping $\sigma_{x,y}$ reflects the symmetry in the environment.

If the trajectory data $\mathcal{T}$ contained all the possible trajectories, then it would suffice to check if each trajectory in $\tau = \{(a_i, o_i, r_i)\}_i \in \mathcal{T}$ is mapped to some trajectory $\tau' = \{(\sigma_{x,y}(a_i), \sigma_{x,y}(o_i), r_i)\}_i \in \mathcal{T}$, according the permutable POMDP formulation. However, since only a subset of trajectories is available, we relax the criterion to find *as many* matching transitions as possible.

This boils down to measuring how many transitions are matched by $\sigma_{x,y}$ over all the transitions relevant to any of $x, y \in E$. We consider the following sets of transitions:

$$T \doteq \{(a, o, r) : \exists \tau \in \mathcal{T} \text{ s.t. } (a, o, r) \in \tau\}$$
$$T_x \doteq \{(a, o, r) \in T : a \text{ contains } x\}$$
$$T_y \doteq \{(a, o, r) \in T : a \text{ contains } y\}$$
$$T_{x,y} \doteq \{(a, o, r) \in T_x : (\sigma_{x,y}(a), \sigma_{x,y}(o), r) \in T_y\}$$

Here, T is the set of all transitions found in $\mathcal{T}$ and T_x and T_y are the sets of transitions whose action contains the phrase x

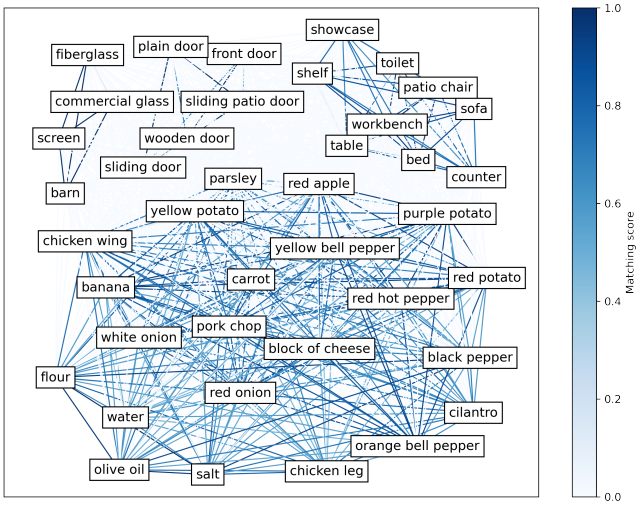

Figure 3: Clusters of the extracted phrases for the *Cooking Game*. The color of the edge represents the matching score.

and y, respectively. $T_{x,y}$ denotes the set of transitions of T_x and T_y that are matched by $\sigma_{x,y}$. For example, $\sigma_{\texttt{carrot,banana}}$ may match the transition (`dice carrot with knife`, `You dice the carrot with knife`, 0) with (`dice banana with knife`, `You dice the banana with knife`, 0).

We define the matching score $f(x, y)$ similarly to the Jaccard similarity,

$$f(x, y) \doteq \frac{|T_{x,y}|}{|T_x| + |T_y| - |T_{x,y}|},$$

where higher matching score indicates that the mapping $\sigma_{x,y}$ matches more transitions associated with phrases x, y, and thus is more likely to represent the symmetry of the environments.

Using the matching score, we build a graph G of phrases where two phrases have an edge when their matching score is above threshold. We discard phrases with no edges. We also remove phrases containing any other phrase in G to avoid ambiguity in mapping the word sequences. The matching scores of the pairs of extracted phrases for the *Cooking Game* are illustrated in Figure 3.

5.3 Training Procedure

We will perform permutation both on the observation and the actions to ensure coherence between them. Let N denote the number of connected components of G and C_i denote each connected component, $i = 1, \ldots, N$. For each phrase cluster C_i, let P_i denote the set of all permutations on C_i. We will use $P \doteq \{\rho_1 \circ \ldots \circ \rho_N : \rho_i \in P_i\}$ as our permutation set.

For a word sequence $x \in V^*$ and a phrase permutation $\rho \in P$, let $\sigma_\rho(x)$ denote the operation that replaces any occurences of each entity $e \in E$ found in x with $\rho(e)$. For example, if ρ swaps `carrot` with `banana` and `table` with `counter`, $\sigma_\rho($`take carrot from table`$)=$`take banana from counter`. Algorithm 1 describes the pseudo-code for applying relabeling as data augmentation for learning to play text-based games.

Algorithm 1 Applying relabeling for text-based games

for each episode **do**
  Reset the environment.
  Randomly sample a permutation $\rho \in P$.
  for each time step t **do**
    Let $o'_t \doteq \sigma_\rho(o_t)$ denote the permuted observation.
    Let $A' \doteq \{\sigma_\rho(a) : a \in A\}$ denote the permuted action set.
    Choose $a'_t \in A'$ according to the behavior policy.
    Act $a_t \doteq \sigma_{\rho^{-1}}(a'_t)$ on the environment and receive a reward r_t.
    Update the model parameters of the agent.
  end for
  Collect the permuted transitions $\{(o'_t, a'_t, r_t, o'_{t+1})\}_t$ into the replay buffer.
end for

6 Experiments

6.1 Training Details

In reporting our experiment results, we used the mean of trials with five different seeds. For each trial, we select the best model in terms of the validation set performance and report its test set performance. We used 100 training games instances for each difficulty level.

We mostly follow the setup by Adhikari *et al.* [2020]. However, we found that our data augmentation allowed convergence to higher performance. Therefore, to emphasize the effect of our methods, we modified the setup as follows.

We train the agents for 300,000 episodes with a linear decay on the learning rate from 10^{-3} to 10^{-6}. We use ϵ-greedy with the values of ϵ annealed from 1.0 to 0.1 over 200,000 episodes. We use the batch of size 256 for the parameter update. The update takes place after each game step of the batch environment interaction of size 64.

For training GATA models with our data augmentation, we also applied our method in the pre-training process of the graph encoders as well. Otherwise, the performance significantly degraded due to the data distribution shift on the input to the graph encoders.

6.2 Models

We demonstrate the effectiveness of our approach by comparing the test performance of various models with and without our data augmentation techniques. We adopt the following baseline methods of Adhikari *et al.* [2020],

- Tr-DQN [Adhikari *et al.*, 2020]: A DQN baseline [Mnih *et al.*, 2015] with Transformer [Vaswani *et al.*, 2017] encoder for the observation.
- Tr-DRQN [Adhikari *et al.*, 2020]: A DRQN baseline [Van Hasselt *et al.*, 2016], which is a recurrent variant of Tr-DQN.
- GATA OG and GATA COC [Adhikari *et al.*, 2020].

We additionally experiment on the following model,

- Tr-DQN-cat: Tr-DQN with the concatenation of the last few observations observation as input. Specifically, we use the last 256 tokens of the concatenated observations.

	Level 1			Level 2			Level 3			Level 4		
	Score	Rel.Imp.	Read	Score	Rel.Imp.	Read	Score	Rel.Imp.	Read	Score	Rel.Imp.	Read
Tr-DQN	63.8	-11.5%	30%	42.5	-2.6%	72%	42.4	-13.4%	10%	21.5	**+3.7%**	4%
+Ours	56.5		8%	41.4		34%	36.7		6%	22.3		0%
Tr-DRQN	64.0	-15.9%	4%	45.0	-21.5%	8%	42.0	-6.8%	10%	12.9	**+51.6%**	4%
+Ours	53.8		9%	35.3		19%	39.2		3%	19.6		21%
Tr-DQN-cat	63.3	-11.9%	3%	49.6	**+29.0%**	1%	31.7	**+2.1%**	5%	12.6	**+2.2%**	6%
+Ours	55.8		10%	64.0		50%	32.3		3%	12.8		4%
GATA OG	64.4	-22.2%	20%	33.9	-4.0%	38%	38.0	**+1.7%**	5%	11.4	**+56.3%**	1%
+Ours	50.1		15%	32.6		29%	38.6		1%	17.8		6%
GATA COC	66.4	-25.4%	1%	41.3	-20.2%	6%	32.1	**+25.4%**	1%	15.2	**+23.6%**	1%
+Ours	49.6		5%	32.9		1%	40.2		4%	18.8		7%
CREST	63.2	-25.3%	8%	34.2	**+0.5%**	21%	39.6	**+3.4%**	6%	18.8	-4.5%	0%
+Ours	47.2		5%	34.4		52%	41.0		14%	18.0		17%
CREST-cat	65.8	**+28.3%**	8%	40.4	**+87.0%**	6%	39.7	**+9.1%**	11%	18.2	**+36.1%**	1%
+Ours	84.4		59%	75.5		86%	43.3		10%	24.7		0%

Table 3: Normalized scores on the test set games with and without our data augmentation technique (Score) and the relative improvement of the scores by the augmentation (Rel.Imp.). Bold indicates positive relative improvements. We also list the portion of test episodes that contain the action `examine cookbook` (Read).

Lastly, we use the following two variants of CREST,

- CREST: Tr-DRQN with the pruning of CREST.
- CREST-cat: Tr-DQN-cat with the pruning of CREST.

We use the CREST threshold θ_{CREST} of 0.5 for Level 1 and 0.4 for Level 2, 3, and 4.

Tr-DQN, Tr-DRQN, and CREST use the following format:

$$\texttt{<sep>}\ a_{t-1}\ \texttt{<sep>}\ o_t$$

For Tr-DQN-cat and CREST-cat, we use the following:

$$\texttt{<sep> restart <sep>}\ o_0\ \texttt{<sep>}\ a_0$$
$$\texttt{<sep>}\ o_1\ \texttt{<sep>}\ \ldots\ \texttt{<sep>}\ a_{t-1}\ \texttt{<sep>}\ o_t$$

6.3 Results

Table 3 shows the normalized scores on the test set games before and after applying our data augmentation technique and the relative improvement of the scores by the augmentation.[1] The relative improvement of a score m over a reference score m_{ref} is calculated as $(m - m_{ref})/m_{ref}$. Table 3 also lists the ratio of test episodes that contain the action `examine cookbook`, which is a necessary condition for acquiring a generalizable policy in the *Cooking Game*.

First of all, most of the models improved the performance on Level 3 and 4. Furthermore, the performance of CREST-cat has increased significantly at Level 1 and 2, setting up the state-of-the-art on the benchmark. As manifested in the relatively high portion of episodes with `examine cookbook`, CREST-cat succeeded in learning the generalizable policy of examining and following the recipe on these difficulty levels, after applying our augmentation. Tr-DQN-cat also accomplished the generalizable policy on Level 2.

On the contrary, other models experienced performance drop on Level 1 and 2. Note the relatively low portion of episodes with `examine cookbook` for these models. As mentioned in Section 4, each ingredient is associated with slightly different distribution of desired processing methods in the *Cooking Game* banchmark. Without data augmentation, these models learned a non-generalizable suboptimal policy that exploits this distribution without reading the recipe. Unfortunately, our data augmentation technique prohibits this strategy. Unlike CREST-cat, these models could not acquire the generalizable policy whether our data augmentation is applied or not. As a result, they learned neither of the aforementioned policies, yielding poorer performance.

Nevertheless, it is remarkable that the acquisition of the generalizable policy on Level 1 and 2 was possible only with our data augmentation method. This confirms that the Transition-Matching Permutation is essential in achieving higher level of generalizability. However, on Level 3 and 4, the low portion of episodes with `examine cookbook` shows that our data augmentation technique was not sufficient to acquire the generalizable policy. It is because our data augmentation technique is not perfect in discovering the symmetries. For instance, each ingredient also has a designated set of initial locations in the Cooking Game; e.g. `carrot` can appear only in the `fridge` or the `garden`. Consequently, the Transition-Matching Permutation hinders learning more efficient policy of examining the cookbook and visiting the associated location, although such policy was not acquired without augmentation as well.

One possible direction to handle these problems would be to consider the permutation on the n-tuples of the phrases for some n instead of single phrases. In this case, the efficient search algorithm for such tuples would be the main challenge. Another possible direction is to utilize belief states. It would be possible to capture symmetry that is manifested over multiple time-steps. In this case, the challenge would be to construct belief states which are well-suited to work together with the Transition-Matching Permutation.

[1] The figures of Tr-DQN, Tr-DRQN, and GATA in Table 3 are our reproduction.

7 Conclusion

In this work, we proposed *Transition-Matching Permutation*, a novel data augmentation technique for text-based games, based on the Permutable POMDP formulation. We experimentally verified the effectiveness of our approach by training various agents with our data augmentation technique. Moreover, with our technique, a simple variant of DQN with a reasonable observation pruning attained the state-of-the-art performance on Level 1 and 2 of the *Cooking Game* benchmark.

One limitation of our work is that it does not handle non-trivial relationships between the state, action, and observation permutations. Discovering such complex relations from the trajectory data can be a promising future direction.

Acknowledgements

This work was supported by the National Research Foundation (NRF) of Korea (NRF-2019R1A2C1087634, NRF-2021M3I1A1097938), the Ministry of Science and Information communication Technology (MSIT) of Korea (IITP No. 2020-0-00940, IITP No. 2019-0-00075, IITP No. 2021-0-02068), and Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (No. 22ZS1100).

References

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems*, 33, 2020.

Giovanni Campagna, Agata Foryciarz, Mehrad Moradshahi, and Monica Lam. Zero-shot transfer learning with synthesized data for multi-domain dialogue state tracking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 122–132, 2020.

Subhajit Chaudhury, Daiki Kimura, Kartik Talamadupula, Michiaki Tatsubori, Asim Munawar, and Ryuki Tachibana. Bootstrapped Q-learning with context relevant observation pruning to generalize in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3002–3008, 2020.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A learning environment for text-based games. *arXiv preprint arXiv:1806.11532*, 2018.

Finale Doshi and Nicholas Roy. The permutable POMDP: Fast solutions to POMDPs for preference elicitation. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '08, page 493–500. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength natural language processing in Python. 2020.

Yutai Hou, Yijia Liu, Wanxiang Che, and Ting Liu. Sequence-to-sequence data augmentation for dialogue language understanding. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1234–1245, 2018.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems*, 33, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

Adam Trischler, Marc-Alexandre Côté, and Pedro Lima. First TextWorld problems, the competition: Using text-based games to advance capabilities of AI agents. 2019.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2020.