Generalization on Text-based Games using Structured Belief Representations

by

Ashutosh Devendrakumar Adhikari

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public

Statement of Contributions

This thesis consists in part of the author's work done as an intern at Mila (Quebec AI Institute) in collaboration with Microsoft Research Montreal, which was accepted as a poster for NeurIPS 2020 [1].

The work is co-first authored by Ashutosh Adhikari, Xingdi Yuan and Marc-Alexandre Côté. Ashutosh Adhikari developed the pre-training approaches, the unsupervised version of the graph-aided transformer agent (GATA) and the probing methods. Marc-Alexandre Côté provided the games and the datasets for pre-training and probing methods and visualization of graphs. Xingdi Yuan scaled the experiments to a large number of games and came up with the supervised version of GATA. All the co-first authors jointly maintained the code base and performed the code reviews.

Abstract

Text-based games are complex, interactive simulations where a player is asked to process the text describing the underlying state of the world to issue textual commands for advancing in a game. Playing these games can be formulated as acting in a partially observable Markov decision process (POMDP), as the player needs to issue actions to reach the goal, by optimizing rewards, given textual observations that may not fully describe the underlying state. Previous art has focused on developing agents to achieve high rewards or faster convergence to the optimal policy for single games. However, with the recent advances in reinforcement learning and representation learning for language we argue it is imperative to start looking for agents that can play a set of games drawn from a distribution of games rather than single games at a time.

In this work, we will be looking at TextWorld [17] as a testbed for developing generalizable policies and benchmarking them against previous work. TextWorld is a sandbox environment for training and evaluating reinforcement learning agents on text-based games. TextWorld is suitable to check the generalizability of agents as it enables us to generate hundreds of unique games with varying levels of difficulties. Difficulty in text-based games are determined by a variety of factors like the number of locations in the environment and length of the optimal walkthrough to name a few. Playing text-based games requires skills in sequential decision making and processing language. In this thesis we evaluate the learnt control policies by training them on a set of games and then observing their scores on unseen games during the training phase. We check for the quality of the policies learnt, their ability to generalize on a distribution of games and their ability to transfer on games from different distributions. We define game distributions based on the difficulty level parameterized by the number of locations in the game, number of objects, etc.

We propose generalizable and transferrable policies by extracting structured information from the raw textual observations describing the state. Additionally, our agents learn these policies in a purely data-driven fashion without using any handcrafted component – a common practice found in prior work. Specifically, we learn dynamic knowledge graphs from raw text to represent our agents' beliefs. The dynamic belief graphs a) allow agents to extract relevant information from text observations and, b) act as memory to act optimally in the POMDP. Experiments on 500+ different games from the TextWorld suite show that our best agent outperforms previous baselines by an average of 24.2%.

Acknowledgements

First and foremost, I would like to thank my advisors, Professor Jimmy Lin and Professor Pascal Poupart, for their constant support, supervision and guidance during the course of my masters. Prof. Jimmy has inspired me to have a great work ethic to carry out research, and I have learnt to always strive for clarity in research from Prof. Pascal.

I would also like to Professor William L. Hamilton, who has taught me to be more confident and ambitious when it comes to motivating and pursuing ideas. The work done in this thesis would not have been possible without his support. I also would like to thank my collaborators from Microsoft Research, especially Xingdi Yuan, Marc-Alexandre Côté, Romain Laroche and Adam Trischler.

I would also like to thank Professor Jesse Hoey and Professor Yaoliang Yu for their comments, and agreeing to be reviewers on my thesis.

I would also like to thank my collaborators from the Data Systems Group, Achyudh Ram and Raphael Tang for the projects done outside the scope of this thesis.

Finally, I would like to thank my family, especially my mom, dad and sister for their constant support during my masters.

Dedication

The thesis is dedicated to my parents and my sister for their constant support, advice and love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Text-based games are complex, interactive simulations in which the game state is described with text and players act using simple text commands (e.g., `chop carrot with knife`). They serve as a proxy for studying how agents can exploit language to comprehend and interact with the environment. Text-based games are a useful challenge in the pursuit of intelligent agents that communicate with humans (e.g., in customer service systems).

Playing text-based games requires a combination of reinforcement learning (RL) and natural language processing (NLP) techniques. For example : a player has to process textual observations to optimize rewards and make a recipe by collecting all the ingredients and processing them correctly. However, inherent challenges like partial observability, long-term dependencies, sparse rewards, and combinatorial state-action spaces make these games very difficult. For instance, [29] show that a state-of-the-art model achieves a mere 2.56% of the total possible score on a curated set of text-based games for human players [11]. On the other hand, while text-based games exhibit many of the same difficulties as linguistic tasks like open-ended dialogue, they are more structured and constrained. Furthermore, the idea behind solving these games is to potentially yield agents which can understand their environments purely by language-informed interactions. However at the current stage, the results obtained on these games should not be directly extended to any real-world application(s), owing to the relative simplicity of these games compared to interactions in the real-world.

To design successful agents for text-based games, previous works have relied largely on heuristics that exploit games' inherent structure. For example, several works have proposed rule-based components that prune the action space or shape the rewards according to *a priori* knowledge of the game dynamics [81, 44, 2, 78]. More recent approaches take advantage of the graph-like structure of text-based games by building knowledge graph (KG) representations of the game state: Ammanabrolu et al., 2019 [6] and Ammanabrolu et al., 2020 [5], for example, use hand-crafted heuristics to populate a KG that feeds into a deep neural agent to inform its policy. They use off-the-shelf open domain information extraction tools (OpenIE [8]) to obtain triplets from text observations at every time step. The set of triplets thus obtained contain many irrelevant triplets which calls for using handcrafted filters for pruning the obtained knowledge graphs. Furthermore, handcrafted components are also required for maintaining the dynamic knowledge graphs to preserve the information from previous time steps – not a functionality provided by OpenIE. Despite progress along this line, we expect more general, effective representations for text-based games to arise in agents that learn and scale more automatically, which replace heuristics with learning [65].

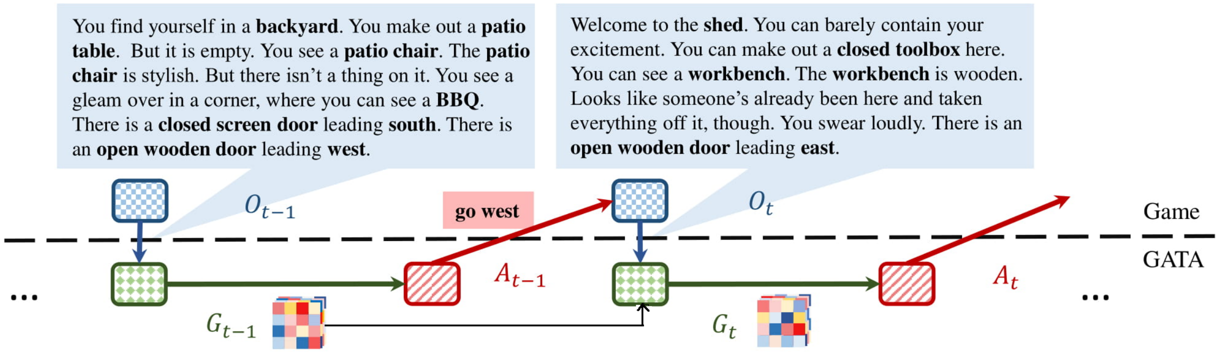
This work investigates how we can learn graph-structured state representations for text-based games in an entirely data-driven manner. We propose the graph-aided transformer agent (GATA) that, in lieu of heuristics, *learns* to construct and update graph-structured beliefs[1] and use them to further optimize rewards. We introduce two self-supervised learning strategies—based on text reconstruction and mutual information maximization by contrastive learning—which enable our agent to learn latent graph representations without direct supervision or hand-crafted heuristics.

We benchmark GATA on 500+ unique games generated by TextWorld [17] (a sandbox environment for text-based games), evaluating performance in a setting that requires generalization across different game configurations. We show that GATA outperforms strong baselines, including text-based models with recurrent policies. In addition, we compare GATA to agents with access to ground-truth graph representations of the game state. We show that GATA achieves competitive performance against these baselines even though it receives only partial text observations of the state. Our findings suggest, promisingly, that graph-structured representations provide a useful inductive bias for learning and generalizing in text-based games.

1.1 Problem Statement

A graph-aided transformer agent (GATA), generalizes on distributions of text-based games better than previous baselines using purely data-driven unsupervised learning regimes.

[1] Text-based games are partially observable environments.



You find yourself in a **backyard**. You make out a **patio table**. But it is empty. You see a **patio chair**. The **patio chair** is stylish. But there isn't a thing on it. You see a gleam over in a corner, where you can see a **BBQ**. There is a **closed screen door** leading **south**. There is an **open wooden door** leading **west**.

Welcome to the **shed**. You can barely contain your excitement. You can make out a **closed toolbox** here. You can see a **workbench**. The **workbench** is wooden. Looks like someone's already been here and taken everything off it, though. You swear loudly. There is an **open wooden door** leading **east**.

Figure 1.1: GATA playing a text-based game by updating its belief graph. In response to action A_{t-1}, the environment returns text observation O_t. Based on O_t and $\mathcal{G}_{t-1}$, the agent updates $\mathcal{G}_t$ and selects a new action A_t. In the figure, blue box with squares is the game engine, green box with diamonds is the graph updater, red box with slashes is the action selector.

1.2 Contributions

The main contributions of this work can be summarized as follows :

- We propose a novel graph-aided transformer agent (GATA) which extracts dynamic belief graphs in an end-to-end manner using unsupervised training regimes, to optimize rewards on distributions of text-based games;
- We propose two unsupervised training regimes, contrastive learning and self-supervised learning-based, to extract and maintain dynamic knowledge graphs from unstructured textual sequences;
- We empirically benchmark all the methods discussed in the work on a wide variety of games to test for their generalization abilities.
- We also probe the dynamic knowledge graph embeddings extracted by GATA to study the nature of information encoded by them.

1.3 Thesis Organization

The thesis is organized as follows, in Chapter 2, we go over related works on playing text-based games, reinforcement learning, graph representation learning, unsupervised representation learning. Chapter 3, the main chapter, describes the model architecture of GATA and the unsupervised training regimes, followed by details about experiments in Chapter 4. In Chapter 5, we summarize the main contributions of the thesis and discuss potential future work.

Chapter 2

Background and Related Work

The approaches discussed in this work are an amalgamation of graph representation learning, deep reinforcement learning, representation learning and text-based games. In this chapter, we first discuss the relevant topics to explain our methodology in Chapter 3.

2.1 A brief review of Reinforcement Learning (RL)

The essence of reinforcement learning is learning through interaction. An RL agent interacts with its environment and learns to optimize rewards by altering its behavior based on these interactions. Such trial-and-error learning has its roots in behavioral psychology which forms one of the core foundations of RL [66]. In an RL setting, an agent observes a state S_t from the environment at gamestep t. The agent then issues an action a_t to interact with the environment based on which it receives a reward R_{t+1} from the environment. The goal of an agent is learn a policy π to maximize the expected return. This goal of an RL agent draws parallels with the literature in optimal control. However, unlike in optimal control, the state transition dynamics of the environment aren't accessible by the agent. As a result, the agent has to rely heavily on the trial-and-error mechanism to learn the consequences of its actions. Every action issued by the agent yields information from the environment, which the agent uses to update its knowledge. See the perception-action-learning loop in Figure 2.1.

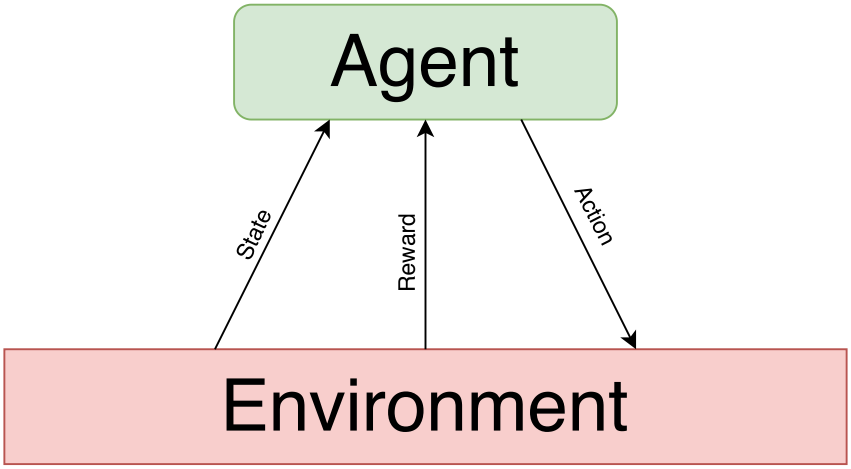


Figure 2.1: An agent interacts with the environment by issuing an action to receive next state and reward.

2.1.1 Markov Decision Processes and RL

Playing games can be represented as a Markov decision process (MDP) parameterized by $< S, p, A, T, R, \gamma >$. Here, S is a set of states, $p(s_0)$ is a distribution of starting states, A is the set of actions, $T(s_{t+1}|s_t, a_t)$ are the state transition dynamics, $R(s_t, a_t, s_{t+1})$ is the reward function and $\gamma \in [0, 1]$ is the discount factor where lower values yield myopic returns.

An agent's policy π maps states to a probability distribution over actions : $\pi : S \rightarrow p(A = a|S)$. In case of episodic MDPs, the state is reset after reaching the terminal state. Further, the sequence of states, actions and rewards is called a rollout of the policy. Every rollout results in a return R defined as : $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$. The goal in RL is to find an optimal policy π^* to achieve the maximum expected return : $\pi^* = argmax_{\pi} \mathbb{E}[R|\pi]$. In case of non-episodic MDPs ($T = \infty$), $\gamma < 1$ prevents infinite returns.

As per the Markov property, the current state s_t is solely dependent on the previous state s_{t-1}. As a result of this property, in RL, actions a_{t-1} are chosen by just observing the current state s_{t-1}. Formulating RL as an MDP requires the environment to be fully observable, which may not be the case. Text-based games for instance are partially observable. Partially observable Markov decision processes (POMDPs) are thus a more general formulation of MDPs which can be used to model partially observable models as well.

In POMDPs the agent receives an observation $o_t \in \Omega$ which is dependent on the current state and the previous action as per the observation probability distribution, $p(o_{t+1}|s_{t+1}, a_t)$.

Algorithms for POMDPs maintain belief over the current state conditioned upon the belief of the previous state, previous action and the current observation. Typically, in a deep learning setting, recurrent neural networks are used to tackle POMDPs as they are dynamical systems [53, 28, 32].

2.1.2 Value functions

In RL, there are two main approaches to achieve an optimal policy: estimating value functions or carrying out policy search. In this work, we mainly rely on value function-based algorithms and hence we restrict the review [10] to only this class of RL algorithms. The state-value function $V^{\pi}(s)$ is defined as the expected return when starting in state s and following the policy π.

$$V^{\pi}(s) = \mathbb{E}[R|s,\pi] \tag{2.1}$$

The optimal policy $\pi^{*}(s)$ has the state-value function $V^{*}(s)$ (and vice-versa) defined as :

$$V^{*}(s) = max_{\pi}V^{\pi}(s) \quad \forall s \in S. \tag{2.2}$$

Access to $V^{*}(s)$ yields the optimal policy as the agent can choose the action a that maximises $\mathbb{E}_{s_{t+1}\sim T(s_{t+1}|s_t,a)}[V^{*}(s_{t+1})]$ at every time step. Similar to the state-value function $V(s)$, we construct the state-action value or quality function $Q^{\pi}(s,a)$. We do so, as the transition dynamics $T(s_{t+1}|s_t,a_t)$ are not accessible by the agent. The $Q^{\pi}(s,a)$ is given as :

$$Q^{\pi}(s,a) = \mathbb{E}[R|s,a,\pi] \tag{2.3}$$

Given a $Q^{\pi}(s,a)$, the agent chooses the action a with highest value in every state : $argmax_{a}Q^{\pi}(s,a)$.

2.1.3 How to learn Q-values?

Q-values are learnt by leveraging the Markov property using the Bellman equation [14]. The Bellman equation adapted for Q-values looks as follows:

$$Q^{\pi}(s,a) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma Q^{\pi}(s_{t+1},\pi(s_{t+1}))]. \tag{2.4}$$

The recursive Bellman equation 2.4 allows us to iteratively improve our Q^{π} values. Such updates forms the core of Q-learning [74] and the state-action-reward-state-action (SARSA) [57] algorithms. Typically the Q-values are updated as :

$$Q^{\pi}(s_t,a_t) \leftarrow Q^{\pi}(s_t,a_t) + \alpha\delta, \tag{2.5}$$

where α is the learning rate, $\delta = Y - Q^{\pi}(s_t, a_t)$ is the temporal difference and Y are the targets.

SARSA is an on-policy learning algorithm which uses the transitions by the behavioral policy (derived from Q^{π}) to improve the Q-value estimates. In SARSA, the targets are thus written as : $Y = r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})$. In Q-learning, targets are necessarily obtained by the behavioral policy, instead, it generally formulates targets as : $Y = r_t + \gamma max_a Q^{\pi}(S_{t+1}, a)$; which estimate Q^*.

The optimal Q^* is achieved from any arbitrary Q^{π} at an intermediate stage by using generalized policy iteration methods. The policy iteration method generally consists of two phases, policy evaluation and policy improvement. The policy evaluation phase aims at iteratively achieving better estimates of the Q-values by minimizing the TD errors. Better Q-value estimates from policy evaluation helps in achieving improved policy (during policy improvement). Generalised policy iteration methods interleave the policy iteration and policy improvement steps for faster convergence to an optimal policy. In a tabular RL setting, generalized policy iteration methods converge Q^{π} to the optimal Q^* as the we increase the number of iterations.

2.2 Deep Reinforcement Learning

The previous chapter describes an RL setting in a tabular form. However, as we move away towards large state and action spaces, it becomes increasingly intractable to learn Q-values for each state-action pair. As a result, we use deep reinforcement learning where value functions $q(s, a)$, $v(s)$ and policies $\pi(s, a)$ are represented by neural networks–whose parameters are trained by gradient descent optimizers. Thus the Q-value estimates are now dependent on the parameters θ of the neural network (or any other function representation), and are defined as, $Q(s, a; \theta) \approx Q^*(s, a)$.

2.2.1 Deep Q-Networks (DQN)

DQN [51] successfully combined deep neural networks and reinforcement learning to predict action values for a given state S_t. Specifically, they use convolutional neural networks to encode the states of an Atari game (stack of frames of raw pixels) to predict Q value for every action at every time step t. Further, at each game step t, the agent selects an action in an ϵ-greedy manner and adds a transition tuple $< S_t, A_t, R_{t+1}, \gamma_{t+1}, S_{t+1} >$ to a

Algorithm 1 Training DQN

```
Initialize replay buffer $B$
Initialize the Q-network with random weights
for episode = 1, M do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for t=1, T do
        With probability $\epsilon$ select a random action $a_t$
        or select $a_t = max_a Q^*(\phi(s_t), a_t; \theta)$
        Execute action $a_t$ to receive reward $r_t$ and next image $x_{t+1}$
        Set next state $s_{t+1} = s_t, x_{t+1}, a_t$ and preprocess it as $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $< \phi_t, a_t, r_t, \phi_{t+1} >$ in $B$
        Sample random batch $< \phi_j, a_j, r_j, \phi_{j+1} >$ from $B$
        if $\phi_{t+1}$ is terminal then
            $y_j = r_j$
        else
            $y_j = r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta)$
        end if
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
    end for
end for
```

replay buffer [45]. The replay buffer is used to sample transitions to optimize the DQN by minimizing the loss

$$(R_{t+1} + \gamma_{t+1} max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t))^2, \tag{2.6}$$

where $Y = R_{t+1} + \gamma_{t+1} max_{a'} q_{\bar{\theta}}(S_{t+1}, a')$ are the targets, and $\bar{\theta}$ are the parameters of the target network–a frozen copy of the behavioral network θ from a few iterations of the policy iteration before. Algorithm 2 explains the training procedure of the DQN [51] in detail.

2.2.2 Double Deep Q-Networks

The maximizaton step in Equation 2.6 leads to an overestimation bias that can further lead to sub-optimal policies [26]. Double Q-Learning [26] addresses this bias by decoupling the selection of action from its evaluation. In deep Q-learning setting, this is done by using the greedy action from the online network to get the estimated target Q-value from the target network. This can be formulated as:

$$(R_{t+1} + \gamma_{t+1} q_{\bar{\theta}}(S_{t+1}, argmax_{a'} q_\theta(S_{t+1}, a')) - q_\theta(S_t, A_t))^2. \tag{2.7}$$

Doing this was experimentally demonstrated to reduce overestimations to avoid suboptimal policies on Atari games [25].

2.2.3 Prioritized Experience Replay

Naive DQN uniformly samples a batch from the replay buffer B to optimize the Q-network. A DQN with prioritized experience replay [58] samples transitions from which there is a much to learn with higher priority than other transitions. Prioritized experience replay uses the following variant of the temporal difference (TD) error [66], to sample a transition with a probability p_t :

$$p_t \propto |R_{t+1} + \gamma_{t+1} max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_{\theta}(S_t, A_t)|^w. \tag{2.8}$$

The intuition here is to sample transitions with higher TD-errors more frequently to capture the correct state-representations and Q-values rapidly.

2.3 Text-based Games

Text adventure games or interactive fiction (IF) games are interactive simulations where text describes the states of the game, and require players to enter textual commands in order to progress. IF games form a useful testing ground for grounding in natural language and reinforcement learning policies. Formally, text-based games can be described as partially observable Markov decision processes (POMDPs) [17] as the text-only observations, O_t, provided by the game engine at time step t may not fully describe the underlying game state S_t for the agent. Note that the observations O_t are deterministic given the state S_t in the games explored in this work. However, same O_t can be generated for more than one S_t. Using O_t, the agent interacts with its environment by issuing short text commands A_t as actions—based on which rewards R_t are provided at every time step. Figure 1.1 depicts an agent interacting with its environment by issuing `go west` command to receive next observation O_t. During training an agent learns from the rewards provided by the environment to achieve the highest possible score (total rewards). The agent is then evaluated based on the total score it achieves on the testing game (which is different from the game(s) used for training, in this work). A game is considered solved (or won) if the agent achieves 100% score, or in other words the maximum score. When evaluating on a distribution of games, we report the average of the fraction of the maximum possible score achieved by the agent (during both training and testing).

2.3.1 Formats of Text-based Games

Depending upon the type of inputs A_t expected by the game engine, text-based games can be classified into the following three types: [31]:

- **Parser-based Games :** Here, the agent types the short textual commands A_t word by word at every time step to receive the next observation O_{t+1}.
- **Choice-based Games :** Here, the agent has to choose an action A_t from a list of multiple actions provided along with the current observation O_t.
- **Hyptertext-based Games :** The observations O_t consist of clickable links embedded inside them. The agent then clicks on one of these links to progress in the game.

Note that hypertext-based games can be dealt as choice-based games as the hyperlinks can just end up being choices for the agent to choose an action from [83]. Further, even parser-based games with finite number (although large) of actions can be converted to choice-based games, where an agent can choose from all the possible actions at every time step [52]. The agents discussed in this work (are adapted to) deal with choice-based games.

2.3.2 What makes Text-based Games Challenging?

Text-based games come in all genres (e.g. cooking games, treasure hunting games, Zork, etc.), which dictate the content of actions, rewards and observations. Further, even within a genre, one can differentiate games based on their parameters which decide a game's difficulty.

Partial Observability

The textual observations provided by the game engine describes only the locality (temporal and/or spatial) of the agent and are almost always insufficient to convey the entire information about the underlying state to the agent. Thus, text-based games can be seen as discrete partially observable Markov decision processes [38, 17] (POMDPs) defined by the tuple $< S, T, A, \Omega, O, R, \gamma >$. Here S is the set of environment states, A are the textual commands, T is the conditional transition function between states, Ω is the set of observations, O are conditional observation probabilities, $R : S \times A \rightarrow \mathbb{R}$ is the reward function, with $\gamma \in [0, 1]$ being the discount factor.

Large State and Action Space

While there have been a lot of work to find [51, 33, 60] optimal policies in non-tabular environments, it still is a fairly active area of research. In TextWorld, the space of all word strings can be quite large resulting into rich state and action spaces. Further, the validity of commands depends on the current underlying state–which is not characteristic of other game environments like the Atari benchmark used for evaluating RL algorithms.

A widely-used approach to alleviate the issue of large action space in text-based games is to use a list of admissible commands C_t at every game step for the agent to choose from. We adopt the same approach for training the agents in this work.

Sparse Rewards and Long term Credit Assignment

Sparse rewards are an inherent property of text-based games. Sparse rewards also lead to the long-term credit assignment problem–where an agent needs to know which action in the past led to the sparse positive (or negative) reward. For example, dropping an unnecessary ingredient to be able to pick up the right ingredient might be essential to make a recipe.

2.3.3 TextWorld Learning Environment

In this paper, we use TextWorld [17] as a testbed for conducting our experiments. TextWorld is a sandbox Python framework that enables generation of game distributions parametrized by the number of objects, the size of the map, size of the action space, richness of textual descriptions. These parameters dictate the difficulty and the distribution of games in this work (and not the genre of the games), as we aim to evaluate for in-distribution generalization. Generalization across genres or even difficulty levels. Refer to Table 4.1 for how we leverage TextWorld to generate a distribution of games based on the difficulty levels.

While TextWorld's ability to generate a large number of games is desirable for our work, the textual observations O_t are generated using templates (refer [17]). Templated language is not as rich and diverse as natural language making these games a bit limited compared to their natural language counterparts. Jericho [27], a Python-based environment which provide human-made IF games can be a possible alternative to TextWorld in the future. However, Jericho's ability to contain large number of games is extremely limited (50 games) compared to TextWorld–making it unfit for our work. Further, Hausknecht at al. 2019 [29], show that previous state-of-the-art agents perform very poorly on the Jericho framework (achieving a mere score of 2.56% of the maximum possible score).

2.4 Problem Setting

The focus of this work is to evaluate (and train) agents on a distribution of text-based games. Text-based games have a variety of difficulty levels determined mainly by the environment's complexity (i.e., how many locations in the game, and how many objects are interactive), the game length (i.e., optimally, how many actions are required to win), and the verbosity (i.e., how much text information is irrelevant to solving the game).

We use TextWorld [17] to generate unique *choice-based* games of varying difficulty. All games share the same overarching theme: an agent must gather and process cooking ingredients, placed randomly across multiple locations, according to a recipe it discovers during the game. The agent earns a point for collecting each ingredient and for processing it correctly, and finally for completing the recipe. The game is won upon completing the recipe. Processing any ingredient incorrectly terminates the game (e.g., `slice carrot` when the recipe asked for a *diced carrot*). To process ingredients, an agent must find and use appropriate tools (e.g., a knife to `slice`, `dice`, or `chop`; a stove to `fry`, an oven to `roast`).

We divide generated games, all of which have unique recipes and map configurations, into sets for training, validation, and test. Adopting the supervised learning paradigm for evaluating generalization, we tune hyperparameters on the validation set and report performance on a test set of previously unseen games. Testing agents on unseen games (within a difficulty level) is uncommon in prior RL work, where it is standard to train and test on a single game instance. Our approach enables us to measure the robustness of learned policies as they generalize (or fail to) across a "distribution" of related but distinct games. Throughout the paper, we use the term *generalization* to imply the ability of a single policy to play a distribution of related games (within a particular difficulty level).

2.5 Playing Text-based Games

Recent years have seen a host of work on playing text-based games. Various deep learning agents have been explored [52, 31, 27, 82, 36, 5, 83, 77]. Fulda et al., 2017 [21] use pre-trained embeddings to reduce the action space. Other works [82, 62, 36] explicitly condition an agent's decisions on game feedback. Most of this literature trains and tests on a single game without considering generalization. Urbanek et al., 2019 [69] use memory networks and ranking systems to tackle adventure-themed dialog tasks. Yuan et al., 2018 [81] propose a count-based memory to explore and generalize on simple unseen text-based games. For baselines which only use text observations O_t to optimize rewards, we introduce Tr-DRQN which is a DQN-based version with a transformer to encode O_t and Tr-DRQN+ based on Yuan et al., 2018 [81] which used count-based memory. We also introduce Tr-DQN, which is basically LSTM-DQN [52], but with a transformer model to represent O_t. Madotto et al., 2020 [48] use GoExplore [19] with imitation learning to generalize. Adolphs and Hofmann 2019 [2] and Yin and May 2019 [78] also investigate the multi-game setting. These methods rely either on reward shaping by heuristics, imitation learning, or rule-based features as inputs. We aim to minimize hand-crafting, so our action selector is optimized only using raw rewards from games while other components of our model are pre-trained on related data. Recent works [6, 5, 78] leverage graph structure by using rule-based, untrained mechanisms to construct KGs to play text-based games.

2.6 The *FTWP* dataset

Previously, Trichler et al., 2019 [68] presented the *First TextWorld Problems (FTWP)* dataset, which consists of TextWorld games that follow a cooking theme across a wide range of difficulty levels. Although this dataset is analogous to what we use in this work, it has only 10 games of cooking genre per difficulty level. This is insufficient for reliable experiments on generalization, so we generate new game sets for our work. As we will explain in Section 3.3, we use a set of transitions collected from the *FTWP* dataset. To ensure the fairness of using this dataset, we make sure there is no overlap between the *FTWP* and the games we use to train and evaluate our action selector.

2.7 Graphs and Text-based Games

We expect graph-based representations to be effective for text-based games because the state in these games adheres to a graph-like structure. The essential content in most observations of the environment corresponds either to entity attributes (e.g., the state of the `carrot` is `sliced`) or to relational information about entities in the environment (e.g., the `kitchen` is `north_of` the `bedroom`). This information is naturally represented as a dynamic graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, where the vertices $\mathcal{V}_t$ represent entities (including the player, objects, and locations) and their current conditions (e.g., closed, fried, sliced), while the edges $\mathcal{E}_t$ represent relations between entities (e.g., `north_of`, `in`, `is`) that hold at a particular time-step t. By design, in fact, the full state of any game generated by TextWorld can be represented explicitly as a graph of this type [84]. For example, the image in Figure 2.2 shows the adjacency tensor for the `is` relation at an intermediate stage in the game. The aim of our model, GATA, is to estimate the game state by learning to build graph-structured beliefs from raw text observations. In our experiments, we benchmark GATA against models with direct access to the ground-truth game state rather than GATA's noisy estimate thereof inferred from text.

2.7.1 Extracting Ground-truth Graphs from *FTWP* Dataset

Under the hood, TextWorld relies on predicate logic to handle the game dynamics. Therefore, the underlying game state consists of a set of predicates, and logic rules (i.e. actions) can be applied to update them. TextWorld's API allows us to obtain such underlying state S_t at a given game step t for any games generated by the framework. We leverage S_t to extract both $\mathcal{G}_t^{\text{full}}$ and $\mathcal{G}_t^{\text{seen}}$.

In which, $\mathcal{G}_t^{\text{full}}$ is a discrete KG that contains the full information of the current state at game step t; $\mathcal{G}_t^{\text{seen}}$ is a discrete partial KG that contains information the agent has observed from the beginning until step t.

Figure 2.3 shows an example of consecutive $\mathcal{G}_t^{\text{seen}}$ as the agent explores the environment of a *FTWP* game. Figure 2.4 shows the $\mathcal{G}^{\text{full}}$ extracted from the same game.

2.8 Graph Representation Learning

In this section, we review the graph representation learning approaches which become relevant once we have extracted the (belief or ground-truth) graphical representations of the underlying states.

Graph neural networks (GNNs) based on neural message passing [15, 41] between nodes to learn high-dimensional representations have been shown as effective graph representation learning methods. As shown in the figures 2.4 and 2.3, the underlying graphs are multi-relational. As a result we decide to use the relational graph convolutional networks (RGCNs) [59] which are a relation-aware version of the graph convolutional networks (GCNs) [41, 15].

2.8.1 Relational Graph Convolutional Networks

Relational Graph Convolutional Network (RGCN) [59] augment the aggregation function for message passing in GCNs to be relation-aware as :

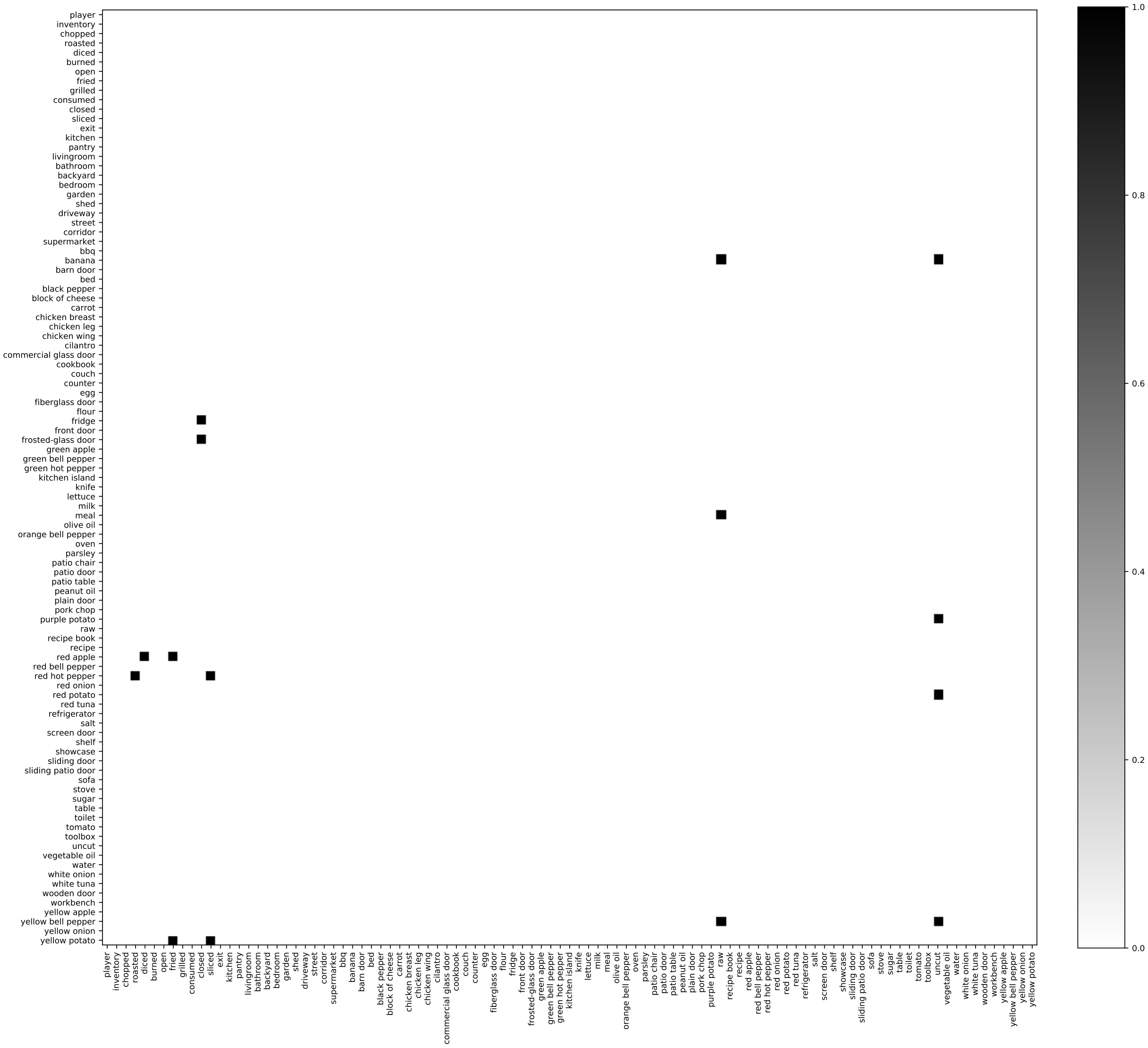
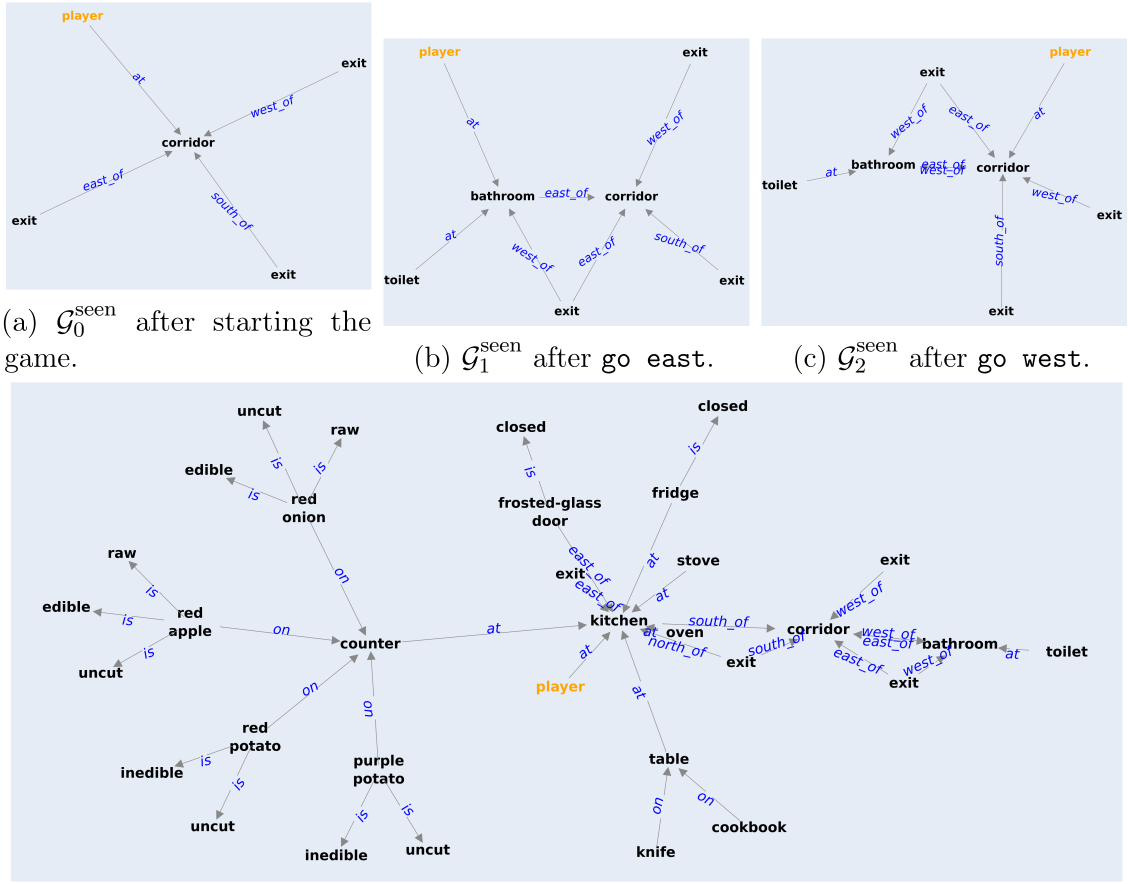


Figure 2.2: Slice of a ground-truth adjacency tensor representing the `is` relation.



(a) $\mathcal{G}_0^{\text{seen}}$ after starting the game.

(b) $\mathcal{G}_1^{\text{seen}}$ after `go east`.

(c) $\mathcal{G}_2^{\text{seen}}$ after `go west`.

(d) $\mathcal{G}_3^{\text{seen}}$ after `go south` that leads to the kitchen which contains many objects.

Figure 2.3: A sequence of $\mathcal{G}^{\text{seen}}$ extracted after issuing three consecutive actions in a *FTWP* game.

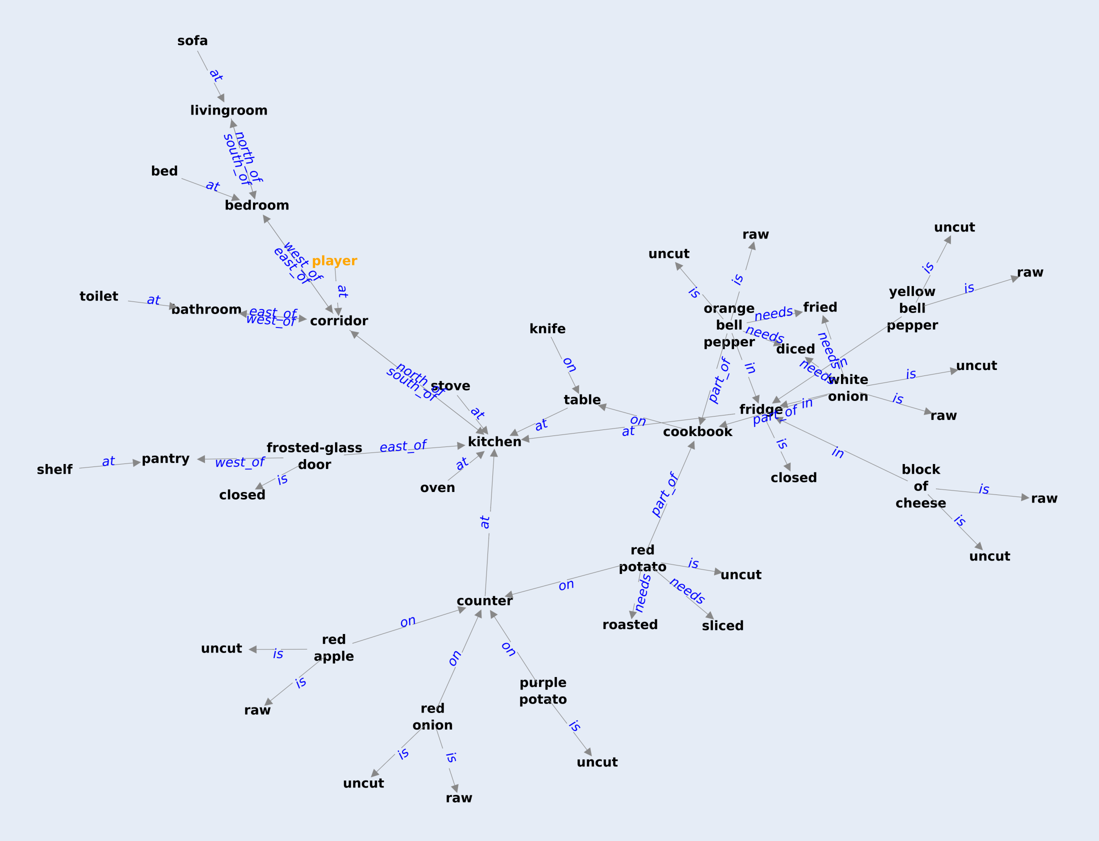


Figure 2.4: $\mathcal{G}^{\text{full}}$ at the start of a *FTWP* game.

$$\tilde{h}_i = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^l h_j^l + W_0^l h_i^l\right), \tag{2.9}$$

where W_r^l is the learnable weight matrix for the $l^{(th)}$ layer and relation $r \in \mathcal{R}$, $\mathcal{N}_i^r$ is the set of neighbouring nodes for the i^{th} node for the relation $r \in \mathcal{R}$. $c_{i,r}$ is the normalizing constant which can be chosen or learnt.

As described in Equation 2.9, the RGCN accumulates messages for every node per relation–making it relation aware. However, this makes naive RGCNs vulnerable to overfitting as the number of relations increase as they have weights W for every relation. Schlichtkrull et al., 2017 [59] provide two regularization methods to address the issue described below:

Block-diagonal Decomposition: In block-diagonal, each W_r^l is defined as a direct sum over a set of low-dimensional matrices given by

$$W_r^l = \oplus_{b=1}^{B} Q_{br}^l, \tag{2.10}$$

where W_r^l are block-diagonal matrices given as $diag(Q_{1r}^l, \ldots, Q_{Br}^l)$ and $Q_{br}^l \in \mathbb{R}^{(d^{l+1}/B) \times (d^l/B)}$.

Basis decomposition: In basis regularization, each W_r^l is decomposed as a linear combination of basis transformations $V_b^l \in \mathbb{R}^{d^{l+1} \times d^l}$. The decomposition can be formulated as :

$$W_r^l = \sum_{b=1}^{B} a_{rb}^l V_b^l \tag{2.11}$$

where a_{rb}^l are the coefficients to combine the basis vectors.

2.8.2 Dynamic graph extraction

Numerous recent works have focused on constructing graphs to encode structured representations of raw data, for various tasks. Kipf et al., 2020 [40] propose contrastive methods to learn latent structured world models (C-SWMs) as state representations for vision-based environments. Their work, however, does not focus on learning policies to play games or to generalize across varying environments. Das et al., 2018 [18] leverage a machine reading comprehension mechanism to query for entities and states in short text passages and use a dynamic graph structure to track changing entity states. Fan et al., 2019 [20] propose to encode graph representations by linearizing the graph as an input sequence in NLP tasks. Johnson 2016 [37] construct graphs from text data using gated graph transformer neural networks. Yang et al., 2018 [76] learn transferable latent relational graphs from raw data in a self-supervised manner. Compared to the existing literature, our work aims to infer multi-relational KGs dynamically from partial text observations of the state and subsequently use these graphs to inform general policies. Concurrently, Srinivas et al., 2020 [63] propose to learn state representations with contrastive learning methods to facilitate RL training. However, they focus on vision-based environments and they do not investigate generalization.

More generally, we want to note that compared to traditional knowledge base construction (KBC) works, our approach is more related to the direction of neural relational inference [39]. In particular, we seek to generate task-specific graphs, which tend to be dynamic, contextual and relatively small, whereas traditional KBC focus on generating large, static graphs.

Chapter 3

Graph-Aided Transformer Agent (GATA)

In this chapter, we introduce GATA, a novel transformer-based neural agent that can infer a graph-structured belief state and use that state to guide action selection in text-based games. As shown in Figure 3.1, the agent consists of two main modules: a graph updater and an action selector. Note that the graph updater and action selector share some structures but not their parameters (unless specified). At game step t, the graph updater extracts relevant information from text observation O_t and updates its belief graph $\mathcal{G}_t$ accordingly. The action selector issues action A_t conditioned on O_t and the belief graph $\mathcal{G}_t$. Figure 1.1 illustrates the interaction between GATA and a text-based game.

3.1 Belief Graph

We denote by $\mathcal{G}$ a belief graph representing the agent's belief about the true game state according to what it has observed so far. We instantiate $\mathcal{G} \in [-1, 1]^{\mathcal{R} \times \mathcal{N} \times \mathcal{N}}$ as a real-valued adjacency tensor, where $\mathcal{R}$ and $\mathcal{N}$ indicate the number of relation types and entities. Each entry $\{r, i, j\}$ in $\mathcal{G}$ indicates the strength of an inferred relationship r from entity i to entity j. We select $\mathcal{R} = 10$ and $\mathcal{N} = 99$ to match the maximum number of relations and entities in our TextWorld-generated games. In other words, we assume that GATA has access to the *vocabularies* of possible relations and entities but it must learn the structure among these objects, and their semantics, from scratch.

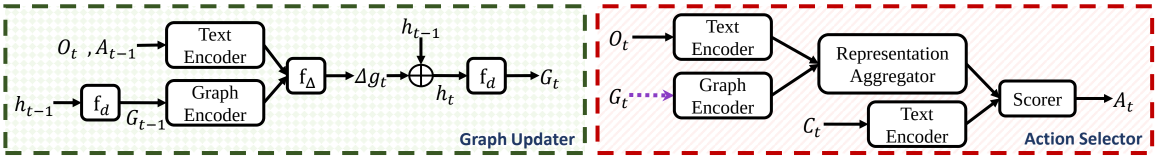


Figure 3.1: GATA in detail. The coloring scheme is same as in Figure 1.1. The graph updater first generates Δg_t using $\mathcal{G}_{t-1}$ and O_t. Afterwards the action selector uses O_t and the updated graph $\mathcal{G}_t$ to select A_t from the list of action candidates C_t. Purple dotted line indicates a detached connection (i.e., no back-propagation through such connection).

3.2 Graph Updater

The graph updater constructs and updates the dynamic belief graph $\mathcal{G}$ from text observations O_t. Rather than generating the entire belief graph at each step t, we generate a graph update, Δg_t, that represents the change of the agent's belief after receiving a new observation. This is motivated by the fact that observations O_t typically communicate only incremental information about the state's change from time step $t-1$ to t. The relation between Δg_t and $\mathcal{G}$ is given by

$$\mathcal{G}_t = \mathcal{G}_{t-1} \oplus \Delta g_t, \tag{3.1}$$

where $\oplus$ is a graph operation function that produces the new belief graph $\mathcal{G}_t$ given $\mathcal{G}_{t-1}$ and Δg_t. We formulate the graph operation function $\oplus$ using a recurrent neural network (e.g., a GRU [16]) as:

$$\begin{aligned}\Delta g_t &= \mathrm{f}_\Delta(h_{\mathcal{G}_{t-1}}, h_{O_t}, h_{A_{t-1}});\\ h_t &= \mathrm{RNN}(\Delta g_t, h_{t-1});\\ \mathcal{G}_t &= \mathrm{f_d}(h_t).\end{aligned} \tag{3.2}$$

The function f_Δ aggregates the information in $\mathcal{G}_{t-1}$, A_{t-1}, and O_t to generate the graph update Δg_t. $h_{\mathcal{G}_{t-1}}$ denotes the representation of $\mathcal{G}_{t-1}$ from the graph encoder. h_{O_t} and $h_{A_{t-1}}$ are outputs of the text encoder (refer to Figure 3.1, left part). The vector h_t is a recurrent hidden state from which we decode the adjacency tensor $\mathcal{G}_t$; h_t acts as a memory that carries information across game steps—a crucial function for solving POMDPs [28]. The function $\mathrm{f_d}$ is a multi-layer perceptron (MLP) that decodes the recurrent state h_t into a real-valued adjacency tensor (i.e., the belief graph $\mathcal{G}_t$).

3.2.1 Graph Encoder

GATA utilizes a graph encoder which is based on the R-GCN [59]. Further, to better leverage information from relation labels, when computing each node's representation, we also condition it on a relation representation E:

$$\tilde{h}_i = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r} W_r^l[h_j^l; E_r] + W_0^l[h_i^l; E_r]\right), \tag{3.3}$$

in which, l denotes the l-th layer of the R-GCN, $\mathcal{N}_i^r$ denotes the set of neighbor indices of node i under relation $r \in \mathcal{R}$, $\mathcal{R}$ indicates the set of different relations, W_r^l and W_0^l are trainable parameters. Since we use continuous graphs, $\mathcal{N}_i^r$ includes all nodes (including node i itself). To stabilize the model and preventing from the potential explosion introduced by stacking R-GCNs with continuous graphs, we use Tanh function as σ (in contrast with the commonly used ReLU function).

As the initial input h^0 to the graph encoder, we concatenate a node embedding vector and the averaged word embeddings of node names. Similarly, for each relation r, E_r is the concatenation of a relation embedding vector and the averaged word embeddings of r's label. Both node embedding and relation embedding vectors are randomly initialized and trainable.

To further help our graph encoder to learn with multiple layers of R-GCN, we add highway connections [64] between layers:

$$\begin{aligned} g &= L^{\text{sigmoid}}(\tilde{h}_i), \\ h_i^{l+1} &= g \odot \tilde{h}_i + (1-g) \odot h_i^l, \end{aligned} \tag{3.4}$$

where $\odot$ indicates element-wise multiplication.

We use a 6-layer graph encoder, with a hidden size H of 64 in each layer. The node embedding size is 100, relation embedding size is 32. The number of bases we use is 3.

3.2.2 Text Encoder

We use a transformer-based text encoder, which consists of a word embedding layer and a transformer block [72]. Specifically, word embeddings are initialized by the 300-dimensional fastText [50] word vectors trained on Common Crawl (600B tokens) and kept fixed during training in all settings.

The transformer block consists of a stack of 5 convolutional layers, a self-attention layer, and a 2-layer MLP with a ReLU non-linear activation function in between. In the block, each convolutional layer has 64 filters, each kernel's size is 5. In the self-attention layer, we use a block hidden size H of 64, as well as a single head attention mechanism. Layernorm [12] is applied after each component inside the block. Following standard transformer training, we add positional encodings into each block's input.

We use the same text encoder to process text observation O_t and the action candidate list C_t. The resulting representations are $h_{O_t} \in \mathbb{R}^{L_{O_t} \times H}$ and $h_{C_t} \in \mathbb{R}^{N_{C_t} \times L_{C_t} \times H}$, where L_{O_t} is the number of tokens in O_t, N_{C_t} denotes the number of action candidates provided, L_{C_t} denotes the maximum number of tokens in C_t, and $H = 64$ is the hidden size.

3.2.3 Representation Aggregator

The representation aggregator aims to combine the text observation representations and graph representations together. Therefore this module is activated only when both the text observation O_t and the graph input $\mathcal{G}_t$ are provided. In cases where either of them is absent, for instance, when training the agent with only $\mathcal{G}^{\text{belief}}$ as input, the aggregator will be deactivated and the graph representation will be directly fed into the scorer.

For simplicity, we omit the subscript t denoting game step in this subsection. At any game step, the graph encoder processes graph input $\mathcal{G}$, and generates the graph representation $h_{\mathcal{G}} \in \mathbb{R}^{N_{\mathcal{G}} \times H}$. The text encoder processes text observation O to generate text representation $h_O \in \mathbb{R}^{L_O \times H}$. $N_{\mathcal{G}}$ denotes the number of nodes in the graph $\mathcal{G}$, L_O denotes the number of tokens in O.

We adopt a standard representation aggregation method from question answering literature [79] to combine the two representations using attention mechanism.

Specifically, the aggregator first uses an MLP to convert both $h_{\mathcal{G}}$ and h_O into the same space, the resulting tensors are denoted as $h'_{\mathcal{G}} \in \mathbb{R}^{N_{\mathcal{G}} \times H}$ and $h'_O \in \mathbb{R}^{L_O \times H}$. Then, a trilinear similarity function [61] is used to compute the similarities between each token in h'_O with each node in $h'_{\mathcal{G}}$. The similarity between ith token in h'_O and jth node in $h'_{\mathcal{G}}$ is thus computed by:

$$\text{Sim}(i, j) = W(h'_{O_i}, h'_{\mathcal{G}_j}, h'_{O_i} \odot h'_{\mathcal{G}_j}), \tag{3.5}$$

where W is trainable parameters in the trilinear function. By applying the above computation for each pair of h'_O and $h'_{\mathcal{G}}$, a similarity matrix $S \in \mathbb{R}^{L_O \times N_{\mathcal{G}}}$ is resulted.

Softmax of the similarity matrix S along both dimensions (number of nodes $N_{\mathcal{G}}$ and number of tokens L_O) are computed, producing $S_{\mathcal{G}}$ and S_O. The information contained in the two representations are then aggregated by:

$$\begin{aligned} h_{O\mathcal{G}} &= [h'_O; P; h'_O \odot P; h'_O \odot Q], \\ P &= S_{\mathcal{G}} h'^{\top}_{\mathcal{G}}, \\ Q &= S_{\mathcal{G}} S^{\top}_O h'^{\top}_O, \end{aligned} \tag{3.6}$$

where $h_{O\mathcal{G}} \in \mathbb{R}^{L_O \times 4H}$ is the aggregated observation representation, each token in text is represented by the weighted sum of graph representations. Similarly, the aggregated graph representation $h_{\mathcal{G}O} \in \mathbb{R}^{N_{\mathcal{G}} \times 4H}$ can also be obtained, where each node in the graph is represented by the weighted sum of text representations. Finally, a linear transformation projects the two aggregated representations to a space with size H of 64:

$$\begin{aligned} h_{\mathcal{G}O} &= L(h_{\mathcal{G}O}), \\ h_{O\mathcal{G}} &= L(h_{O\mathcal{G}}). \end{aligned} \tag{3.7}$$

3.3 Training the Graph Updater

We pre-train the graph updater using two self-supervised training regimes to learn structured game dynamics. After pre-training, the graph updater is fixed during GATA's interaction with games; at this time it provides belief graphs $\mathcal{G}$ to the action selector. We train the action selector subsequently via RL. Both pre-training tasks share the same goal: to ensure that $\mathcal{G}_t$ encodes sufficient information about the environment state at game step t. For training data, we gather a collection of transitions by following walkthroughs in *FTWP* games. To ensure variety in the training data, we also randomly sample trajectories off the optimal path. Next we describe our pre-training approaches for the graph updater.

3.3.1 Observation Generation

Our first approach to pre-train the graph updater involves training a decoder model to reconstruct text observations from the belief graph. Conditioned on the belief graph, $\mathcal{G}_t$, and the action performed at the previous game step, A_{t-1}, the observation generation task aims to reconstruct $O_t = \{O_t^1, \ldots, O_t^{L_{O_t}}\}$ token by token, where L_{O_t} is the length of O_t. We formulate this task as a sequence-to-sequence (Seq2Seq) problem and use a transformer-based model [72] to generate the output sequence. Specifically, conditioned on

$\mathcal{G}_t$ and A_{t-1}, the transformer decoder predicts the next token O_t^i given $\{O_t^1, \ldots, O_t^{i-1}\}$. We train the Seq2Seq model using teacher-forcing to optimize the negative log-likelihood loss:

$$\mathcal{L}_{\text{OG}} = -\sum_{i=1}^{L_{O_t}} \log p_{\text{OG}}(O_t^i | O_t^1, ..., O_t^{i-1}, \mathcal{G}_t, A_{t-1}), \tag{3.8}$$

where p_{OG} is the conditional distribution parametrized by the observation generation model.

As shown in Figure 3.2, given a transition (O_{t-1}, A_{t-1}, O_t), we use the belief graph $\mathcal{G}_t$ and A_{t-1} to reconstruct O_t. $\mathcal{G}_t$ is generated by the graph updater, conditioned on the recurrent information h_{t-1} carried over from previous data point in the transition sequence.

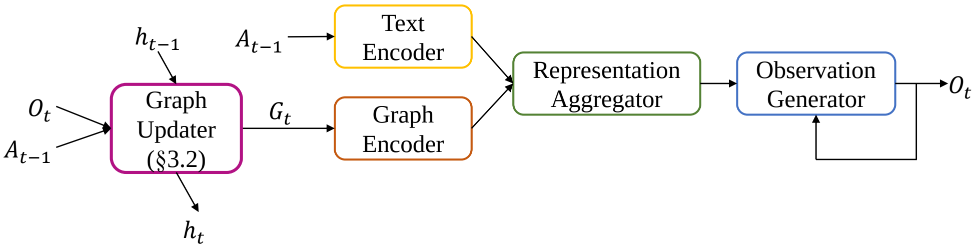


Figure 3.2: Observation generation model.

Observation Generator Layer

The observation generator is a transformer-based decoder. It consists of a word embedding layer, a transformer block, and a projection layer.

Similar to the text encoder, the embedding layer is frozen after initializing with the pre-trained fastText [50] word embeddings. Inside the transformer block, there is one self attention layer, two attention layers and a 3-layer MLP with ReLU non-linear activation functions in between. Taking word embedding vectors and the two aggregated representations produced by the representation aggregator as input, the self-attention layer first generates a contextual encoding vectors for the words. These vectors are then fed into the two attention layers to compute attention with graph representations and text observation representations respectively. The two resulting vectors are thus concatenated, and they are fed into the 3-layer MLP. The block hidden size of this transformer is $H = 64$.

Finally, the output of the transformer block is fed into the projection layer, which is a linear transformation with output size same as the vocabulary size. The resulting logits are then normalized by a softmax to generate a probability distribution over all words in vocabulary. Following common practice, we also use a mask to prevent the decoder transformer to access "future" information during training.

3.3.2 Contrastive Observation Classification (COC)

Inspired by the literature on contrastive representation learning [70, 34, 73, 13], we reformulate OG mentioned above as a contrastive prediction task. We use contrastive learning to maximize mutual information between the predicted $\mathcal{G}_t$ and the text observations O_t. Specifically, we train the model to differentiate between representations corresponding to true observations O_t and "corrupted" observations $\widetilde{O}_t$, conditioned on $\mathcal{G}_t$ and A_{t-1}. To obtain corrupted observations, we sample randomly from the set of all collected observations across our pre-training data. We use a noise-contrastive objective and minimize the binary cross-entropy (BCE) loss given by

$$\mathcal{L}_{\text{COC}} = \frac{1}{K}\sum_{t=1}^{K}\left(\mathbb{E}_{O}\left[\log\mathcal{D}\left(h_{O_t}, h_{\mathcal{G}_t}\right)\right] + \mathbb{E}_{\widetilde{O}}\left[\log\left(1-\mathcal{D}\left(h_{\widetilde{O}_t}, h_{\mathcal{G}_t}\right)\right)\right]\right). \tag{3.9}$$

Here, K is the length of a trajectory as we sample a positive and negative pair at each step and $\mathcal{D}$ is a *discriminator* that differentiates between positive and negative samples. The motivation behind contrastive unsupervised training is that one does not require to train complex decoders. Specifically, compared to OG, the COC's objective relaxes the need for learning syntactical or grammatical features and allows GATA to focus on learning the semantics of the O_t.

The contrastive observation classification task shares the same goal of ensuring the generated belief graph $\mathcal{G}_t$ encodes the necessary information describing the environment state at step t. However, instead of generating O_t from $\mathcal{G}_t$, it requires a model to differentiate the real O_t from some $\tilde{O}_t$ that are randomly sampled from other data points. In this task, the belief graph does not need to encode the syntactical information as in the observation generation task, rather, a model can use its full capacity to learn the semantic information of the current environmental state.

We illustrate our contrastive observation classification model in Figure 3.3. This model shares most components with the previously introduced observation generation model, except replacing the observation generator module by a discriminator.

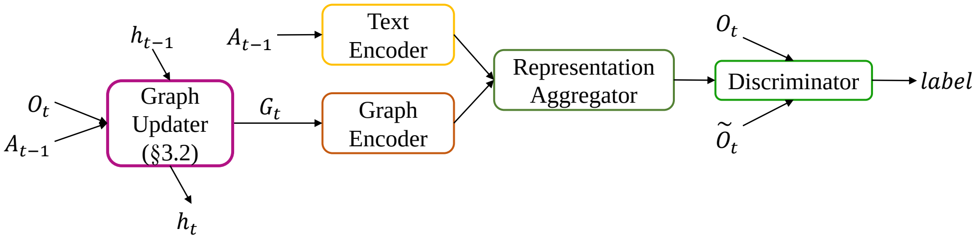


Figure 3.3: Contrastive observation classification model.

3.4 Action Selector

The graph updater discussed in the previous section defines a key component of GATA that enables the model to maintain a structured belief graph based on text observations. The second key component of GATA is the *action selector*, which uses the belief graph $\mathcal{G}_t$ and the text observation O_t at each time-step to select an action. As shown in Figure 3.1, the action selector consists of four main components: the *text encoder* and *graph encoder* convert text inputs and graph inputs, respectively, into hidden representations; a *representation aggregator* fuses the two representations using an attention mechanism; and a *scorer* ranks all candidate actions based on the aggregated representations.

3.4.1 Graph Encoder

GATA's belief graphs, which estimate the true game state, are multi-relational by design. Therefore, we use relational graph convolutional networks (R-GCNs) [59] to encode the belief graphs from the updater into vector representations. We also adapt the R-GCN model to use embeddings of the available relation labels, so that we can capture semantic correspondences among relations (e.g., `east_of` and `west_of` are reciprocal relations). We do so by learning a vector representation for each relation in the vocabulary that we condition on the word embeddings of the relation's name. We concatenate the resulting vector with the standard node embeddings during R-GCN's message passing phase. Our R-GCN implementation uses basis regularization [59] and highway connections [64] between layers for faster convergence.

3.4.2 Text Encoder

We adopt a transformer encoder [72] to convert text inputs from O_t and A_{t-1} into contextual vector representations. The architecture of the text encoder used for the action selector is same as the one for graph updates as described in Section 3.2.2.

3.4.3 Representation Aggregator

To combine the text and graph representations, GATA uses a bi-directional attention-based aggregator [79, 61]. Attention from text to graph enables the agent to focus more on nodes that are currently observable, which are generally more relevant; attention from nodes to text enables the agent to focus more on tokens that appear in the graph, which are therefore connected with the player in certain relations. The representation aggregator's model remains the same as in the graph updater (Section 3.2.3).

3.4.4 Scorer

The scorer consists of a self-attention layer cascaded with an MLP layer. First, the self-attention layer reinforces the dependency of every token-token pair and node-node pair in the aggregated representations. The resulting vectors are concatenated with the representations of action candidates C_t (from the text encoder), after which the MLP generates a single scalar for every action candidate as a score.

The scorer consists of a self-attention layer, a masked mean pooling layer, and a two-layer MLP. As shown in Figure 3.1 and described above, the input to the scorer is the action candidate representation h_{C_t}, and one of the following game state representation:

$$s_t = \begin{cases} h_{\mathcal{G}_t} & \text{if only graph input is available,} \\ h_{O_t} & \text{if only text observation is available, this degrades GATA to a Tr-DQN,} \\ h_{\mathcal{G}O_t}, h_{O\mathcal{G}_t} & \text{if both are available.} \end{cases}$$

First, a self-attention is applied to the game state representation s_t, producing $\hat{s}_t$. If s_t includes graph representations, this self-attention mechanism will reinforce the connection between each node and its related nodes. Similarly, if s_t includes text representation, the self-attention mechanism strengthens the connection between each token and other related tokens. Further, masked mean pooling is applied to the self-attended state representation $\hat{s}_t$ and the action candidate representation h_{C_t}, this results in a state representation vector and a list of action candidate representation vectors. We then concatenate the resulting vectors and feed them into a 2-layer MLP with a ReLU non-linear activation function in between. The second MLP layer has an output dimension of 1, after squeezing the last dimension, the resulting vector is of size N_{C_t}, which is the number of action candidates provided at game step t. We use this vector as the score of each action candidate.

3.5 Training the Action Selector

We use Q-learning [74] to optimize the action selector on reward signals from the training games. Specifically, we use Double DQN [71] combined with multi-step learning [66] and prioritized experience replay [58]. To enable GATA to scale and generalize to multiple games, we adapt standard deep Q-Learning by sampling a new game from the set of training games to collect an episode. Consequently, the replay buffer contains transitions from episodes of different games.

The overall training procedure of GATA's action selector is shown in Algorithm 2.

We report two strategies that we empirically find effective in DQN training. First, we discard the underachieving trajectories without pushing them into the replay buffer (lines 10–12). Specifically, we only push a new trajectory that has an average reward greater than $\tau \in \mathbb{R}_0^+$ times the average reward for all transitions in the replay buffer. We use $\tau = 0.1$, since it keeps around some weaker but acceptable trajectories and does not limit exploration too severely. Second, we keep track of the best performing policy Π on the validation games. During training, when GATA stops improving on validation games, we load Π back to the training policy π and resume training. After training, we report the performance of Π on test games. Note these two strategies are not designed specifically for GATA; rather, we find them effective in DQN training in general.

We use a prioritized replay buffer with memory size of 500,000, and a priority fraction of 0.6. We use ϵ-greedy, where the value of ϵ anneals from 1.0 to 0.1 within 20,000 episodes. We start updating parameters after 100 episodes of playing. We update our network after every 50 game steps (update frequency F in Algorithm 2) Note that 50 is the total steps performed within a batch. For instance, when the batch size is 1, we update per 50 steps; whereas when the batch size is 10, we update per 5 steps. Note the batch size here refers to the parallelization of the environment, rather than the batch size for backpropagation. During update, we use a mini-batch of size 64. We use a discount $\gamma = 0.9$. We update target network after every 500 episodes. For multi-step learning, we sample the multi-step return $n \sim \text{Uniform}[1, 3]$. We refer readers to Rainbow-DQN [33] for more information about different components of DQN training.

Algorithm 2 Training Strategy for GATA Action Selector

```
Input: games $\mathcal{X}$, replay buffer $B$, update frequency $F$, patience $P$, tolerance $\tau$, evaluation frequency $E$.
Initialize counters $k \leftarrow 1, p \leftarrow 0$, best validation score $V \leftarrow 0$, transition cache $C$, policy $\pi$, checkpoint $\Pi$.
for $e \leftarrow 1$ to NB_EPISODES do
    Sample a game $x \in \mathcal{X}$, reset $C$.
    for $i \leftarrow 1$ to NB_STEPS do
        play game, push transition into $C$, $k \leftarrow k + 1$
        if $k\%F = 0$ then sample batch from $B$, Update($\pi$)
        if done then break
    end for
    if average score in $C > \tau \cdot$ average score in $B$ then
        for all item in $C$ do push item into $B$
    end if
    if $e\%E \neq 0$ then continue
    $v \leftarrow$ Evaluate($\pi$)
    if $v >= V$ then $\Pi \leftarrow \pi$, $p \leftarrow 0$, continue
    if $p > P$ then $\pi \leftarrow \Pi$, $p \leftarrow 0$
    else $p \leftarrow p + 1$
end for
```

In our implementation of the Tr-DRQN and Tr-DRQN+ baselines, following Yuan et al., 2018 [81], we sample a sequence of transitions of length 8, use the first 4 transitions to estimate reasonable recurrent states and use the last for to update. For counting bonus, we use a $\gamma_c = 0.5$, the bonus is scaled by an coefficient $\lambda_c = 0.1$.

For all experiment settings, we train agents for 100,000 episodes (NB_EPISODES in Algorithm 2). For each game, we set maximum step of 50 (NB_STEPS in Algorithm 2). When an agent has used up all its moves, the game is forced to terminate. We evaluate them after every 1,000 episodes (evaluation frequency E in Algorithm 2). Patience P and tolerance τ in Algorithm 2 are 3 and 0.1, respectively. The agents are implemented using PyTorch [54].

3.6 Variants Using Ground-Truth Graphs

In GATA, the belief graph is learned entirely from text observations. However, the TextWorld API also provides access to the underlying graph states for games, in the format

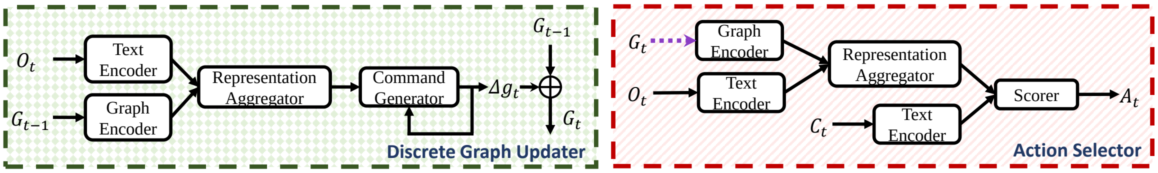


Figure 3.4: GATA-GTP in detail. The coloring scheme is same as in Figure 1.1. The discrete graph updater first generates Δg_t using $\mathcal{G}_{t-1}$ and O_t. Afterwards the action selector uses O_t and the updated graph $\mathcal{G}_t$ to select A_t from the list of action candidates C_t. Purple dotted line indicates a detached connection (i.e., no back-propagation through such connection).

of discrete KGs. Thus, for comparison, we also consider two models that learn from or encode ground-truth graphs directly.

3.6.1 GATA-GTP: Pre-training a *discrete* graph updater using ground-truth graphs

We first consider a model that uses ground-truth graphs to pre-train the graph updater, in lieu of self-supervised methods. GATA-GTP uses ground-truth graphs from *FTWP* during pre-training, but infers belief graphs from the raw text during RL training of the action selector to compare fairly against GATA. Here, the belief graph $\mathcal{G}_t$ is a discrete multi-relational graph. To pre-train a discrete graph updater, we adapt the command generation approach proposed by Zelinka et al., 2019 [84].

The GATA-GTP has the same action scorer as GATA, but equipped with a discrete graph updater. We show the overview structure of GATA-GTP in Figure 3.4.

Discrete Graph Updater

In the discrete graph setting, following Zelinka et al., 2019 [84], we update $\mathcal{G}_t$ with a set of discrete update operations that act on $\mathcal{G}_{t-1}$. In particular, we model the (discrete) Δg_t as a set of update operations, wherein each update operation is a sequence of tokens. We define the following two elementary operations so that any graph update can be achieved in $k \geq 0$ such operations:

- `add(node1, node2, relation)`: add a directed edge, named `relation`, between `node1` and `node2`.

- `delete(node1, node2, relation)`: delete a directed edge, named `relation`, between `node1` and `node2`. If the edge does not exist, ignore this command.

Given a new observation string O_t and $\mathcal{G}_{t-1}$, the agent generates $k \geq 0$ such operations to merge the newly observed information into its belief graph.

Table 3.1: Update operations matching the transition in Figure 1.1.

```
<s> add player shed at <|> add shed backyard west_of <|> add wooden door shed
east_of <|> add toolbox shed in <|> add toolbox closed is <|> add workbench
shed in <|> delete player backyard at </s>
```

We formulate the update generation task as a sequence-to-sequence (Seq2Seq) problem and use a transformer-based model [72] to generate token sequences for the operations. We adopt the decoding strategy from [49], where given an observation sequence O_t and a belief graph $\mathcal{G}_{t-1}$, the agent generates a sequence of tokens that contains multiple graph update operations as subsequences, separated by a delimiter token `<|>`.

Since Seq2Seq set generation models are known to learn better with a consistent output ordering [49], we sort the ground-truth operations (e.g., always `add` before `delete`) for training. For the transition shown in Figure 1.1, the generated sequence is shown in Table 3.1.

Pre-training Discrete Graph Updates

As described above, we frame the discrete graph updating behavior as a language generation task. We denote this task as command generation (CG). Similar to the continuous version of graph updater in GATA, we pre-train the discrete graph updater using transitions collected from the *FTWP* dataset. It is worth mentioning that despite requiring ground-truth KGs in *FTWP* dataset, GATA-GTP does not require any ground-truth graph in the RL game to train and evaluate the action scorer.

For training discrete graph updater, we use the $\mathcal{G}^{\text{seen}}$ type of graphs provided by the TextWorld API. Specifically, at game step t, $\mathcal{G}_t^{\text{seen}}$ is a discrete partial KG that contains information the agent has observed from the beginning until step t. It is only possible to train an agent to generate belief about the world it has seen and experienced.

In the collection *FTWP* transitions, every data point contains two consecutive graphs, we convert the difference between the graphs to ground-truth update operations (i.e., `add` and `delete` commands). We use standard teacher forcing technique to train the transformer-based Seq2Seq model. Specifically, conditioned on the output of representation aggregator, the command generator is required to predict the k^{th} token of the target sequence given all the ground-truth tokens up to time step $k-1$. The command generator module is transformer-based decoder, similar to the observation generator described in Section **??**. Negative log-likelihood is used as loss function for optimization. An illustration of the command generation model is shown in Figure 3.5.

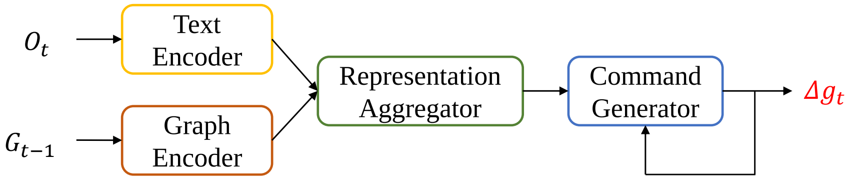


Figure 3.5: Command Generation Model.

During the RL training of action selector, the graph updater is detached without any back-propagation performed. It generates token-by-token started by a begin-of-sequence token, until it generates an end-of-sequence token, or hitting the maximum sequence length limit. The resulting tokens are consequently used to update the discrete belief graph.

Pre-training a Discrete Graph Encoder for Action Scorer

In the discrete graph setting, we take advantage of the accessibility of the ground-truth graphs. Therefore we also consider various pre-training approaches to improve the performance of the graph encoder in the action selection module. Similar to the training of graph updater, we use transitions collected from the *FTWP* dataset as training data.

In particular, here we define a transition as a 6-tuple $< \mathcal{G}_{t-1}, O_{t-1}, C_{t-1}, A_{t-1}, \mathcal{G}_t, O_t >$. Specifically, given $\mathcal{G}_{t-1}$ and O_{t-1}, an action A_{t-1} is selected from the candidate list C_{t-1}; this leads to a new game state $\mathcal{S}_t$, thus $\mathcal{G}_t$ and O_t are returned. Note that $\mathcal{G}_t$ in transitions can either be $\mathcal{G}_t^{\text{full}}$ that describes the full environment state or $\mathcal{G}_t^{\text{seen}}$ that describes the part of state that the agent has experienced.

In this section, we start with providing details of the pre-training tasks and their corresponding models, and then show these models' performance for each of the tasks.

Action Prediction (AP) Given a transition $< \mathcal{G}_{t-1}, O_{t-1}, C_{t-1}, A_{t-1}, \mathcal{G}_t, O_t, r_{t-1} >$, we use A_{t-1} as positive example and use all other action candidates in C_{t-1} as negative examples.

A model is required to identify A_{t-1} amongst all action candidates given two consecutive graphs $\mathcal{G}_{t-1}$ and $\mathcal{G}_t$.

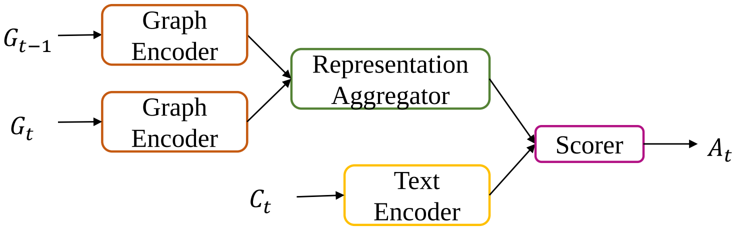


Figure 3.6: Action Prediction Model.

We use a model with similar structure and components as the action selector of GATA. As illustrated in Figure 3.6, the graph encoder first converts the two input graphs $\mathcal{G}_{t-1}$ and $\mathcal{G}_t$ into hidden representations, the representation aggregator combines them using attention mechanism. The list of action candidates (which includes A_{t-1} and all negative examples) are fed into the text encoder to generate action candidate representations. The scorer thus takes these representations and the aggregated graph representations as input, and it outputs a ranking over all action candidates.

In order to achieve good performance in this setting, the bi-directional attention between $\mathcal{G}_{t-1}$ and $\mathcal{G}_t$ in the representation aggregator needs to effectively determine the difference between the two sparse graphs. To achieve that, the graph encoder has to extract useful information since often the difference between $\mathcal{G}_{t-1}$ and $\mathcal{G}_t$ is minute (e.g., before and after taking an apple from the table, the only change is the location of the apple).

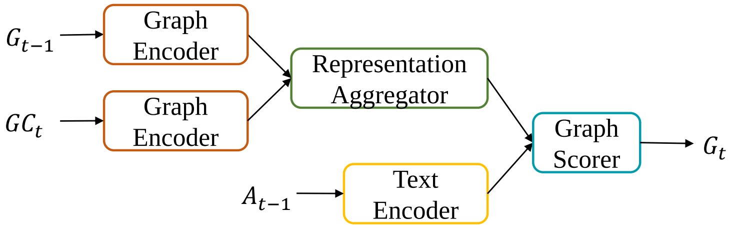


Figure 3.7: State Prediction Model.

State Prediction (SP) Given a transition $(\mathcal{G}_{t-1}, O_{t-1}, C_{t-1}, A_{t-1}, \mathcal{G}_t, O_t, r_{t-1})$, we use $\mathcal{G}_t$ as positive example and gather a set of game states by issuing all other actions in C_{t-1} except A_{t-1}. We use the set of graphs representing the resulting game states as negative samples. In this task, a model is required to identify $\mathcal{G}_t$ amongst all graph candidates GC_t given the previous graph $\mathcal{G}_{t-1}$ and the action taken A_{t-1}.

As shown in Figure 3.7, a similar model is used to train both the SP and AP tasks.

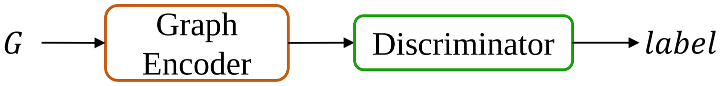


Figure 3.8: Deep Graph Infomax Model.

Deep Graph Infomax (DGI) This pre-training method is inspired by Velickovic et al., 2018 [73]. Given a transition $< \mathcal{G}_{t-1}, O_{t-1}, C_{t-1}, A_{t-1}, \mathcal{G}_t, O_t, r_{t-1} >$, we map the graph $\mathcal{G}_t$ into its node embedding space. The node embedding vectors of $\mathcal{G}_t$ is denoted as H. We randomly shuffle some of the node embedding vectors to construct a "corrupted" version of the node representations, denoted as $\tilde{H}$.

Given node representations $H = \{\overrightarrow{h_1}, \overrightarrow{h_2}, ..., \overrightarrow{h_N}\}$ and corrupted representations of these nodes $\tilde{H} = \{\overrightarrow{\tilde{h}_1}, \overrightarrow{\tilde{h}_2}, ..., \overrightarrow{\tilde{h}_N}\}$, where N is the number of vertices in the graph, a model is required to discriminate between the original and corrupted representations of nodes. As shown in Figure 3.8, the model is composed of a graph encoder and a discriminator. Specifically, following [73], we utilize a noise-contrastive objective with a binary cross-entropy (BCE) loss between the samples from the joint (positive examples) and the product of marginals (negative examples). To enable the discriminator to discriminate between $\mathcal{G}_t$ and the negative samples, the graph encoder must learn useful graph representations at both global and local level.

3.6.2 GATA-GTF: Training the action selector using ground-truth graphs

To get a sense of the upper bound on performance we might obtain using a belief graph, we also train an agent that uses the full ground-truth graph $\mathcal{G}^{\text{full}}$ during action selection. This agent requires no graph updater module; we simply feed the ground-truth graphs into the action selector (via the graph encoder). The use of ground-truth graphs allows GATA-GTF to escape the error cascades that may result from inferred belief graphs. Note also that the ground-truth graphs contain full state information, relaxing partial observability of the games. Consequently, we expect more effective reward optimization for GATA-GTF compared to other graph-based agents. GATA-GTF's comparison with text-based agents is a sanity check for our hypothesis—that structured representations help learning general policies.

Chapter 4

Experiments and Analyses

We conduct experiments on generated text-based games (Section 2.3) to answer two key questions:
Q1: Does the belief-graph approach aid GATA in achieving high rewards on unseen games after training? In particular, does GATA improve performance compared to SOTA text-based models?
Q2: How does GATA compare to models that have access to ground-truth graph representations?

4.1 Experimental Setup and Baselines

We divide the games into four subsets with one difficulty level per subset. Each subset contains 100 training, 20 validation, and 20 test games, which are sampled from a distribution determined by their difficulty level. To elaborate on the diversity of games: for easier games, the recipe might only require a single ingredient and the world is limited to a single location, whereas harder games might require an agent to navigate a map of 6 locations to collect and appropriately process up to three ingredients. We also test GATA's transferability across difficulty levels by mixing the four difficulty levels to build level 5. We sample 25 games from each of the four difficulty levels to build a training set. We use all validation and test games from levels 1 to 4 for level 5 validation and test. In all experiments, we select the top-performing agent on validation sets and report its test scores; all validation and test games are unseen in the training set. Statistics of the games are shown in Table 4.1.

As baselines, we use our implementation of LSTM-DQN [52] and LSTM-DRQN [81], both of which use only O_t as input. Note that LSTM-DRQN uses an RNN to enable

Table 4.1: Games statistics (averaged across all games within a difficulty level).

Level	Recipe Size	#Locations	Max Score	Need Cut	Need Cook	#Action Candidates	#Objects
1	1	1	4	✓	✗	8.9	17.1
2	1	1	5	✓	✓	8.9	17.5
3	1	9	3	✗	✗	4.9	34.1
4	3	6	11	✓	✓	10.8	33.4
5	Mixture of levels {1,2,3,4}						

an implicit memory (i.e., belief); it also uses an episodic counting bonus to encourage exploration [81]. This draws an interesting comparison with GATA, wherein the belief is extracted and updated dynamically, in the form of a graph. For fair comparison, we replace the LSTM-based text encoders with a transformer-based text encoder as in GATA. We denote those agents as Tr-DQN and Tr-DRQN respectively. We denote a Tr-DRQN equipped with the episodic counting bonus as Tr-DRQN+. These three text-based baselines are representative of the current top-performing neural agents on text-based games.

Additionally, we test the variants of GATA that have access to ground-truth graphs (as described in Section 3.6). Comparing with GATA, the GATA-GTP agent also maintains its belief graphs throughout the game; however, its graph updater is pre-trained on *FTWP* using ground-truth graphs—a stronger supervision signal. GATA-GTF, on the other hand, does not have a graph updater. It directly uses ground-truth graphs as input during game playing.

Q1: Performance of GATA compared to text-based baselines

In Table 4.2, we show the normalized test scores achieved by agents trained on either 20 or 100 games for each difficulty level. The normalized test scores are basically average of the fraction of the maximum possible scores achieved by an agent on a testing set. Equipped with belief graphs, GATA significantly outperforms all text-based baselines. The graph updater pre-trained on both of the self-supervised tasks (Section 3.2) leads to better performance than the baselines (⋆ and ∞). We observe further improvements in GATA's policies when the text observations (♠) are also available. We believe the text observations guide GATA's action scorer to focus on currently observable objects through the bi-attention mechanism. The attention may further help GATA to counteract accumulated errors from the belief graphs. In addition, we observe that Tr-DRQN and

Table 4.2: Agents' normalized **test** scores and averaged relative improvement (% ↑) over Tr-DQN across difficulty levels. An agent m's relative improvement over Tr-DQN is defined as $(\mathrm{R_m} - \mathrm{R_{Tr\text{-}DQN}})/\mathrm{R_{Tr\text{-}DQN}}$ where R is the score. All numbers are percentages. ♢represents ground-truth full graph; ♣represents discrete $\mathcal{G}_t$ generated by GATA-GTP; ♠represents O_t. ★and ∞are continuous G_t generated by GATA, when the graph updater is pre-trained with OG and COC tasks, respectively.

	20 Training Games						100 Training Games						Avg.
Difficulty Level	1	2	3	4	5	% ↑	1	2	3	4	5	% ↑	% ↑
Agent	Text-based Baselines												
Tr-DQN	66.2	26.0	16.7	18.2	**27.9**	——	62.5	32.0	38.3	17.7	34.6	—-	—-
Tr-DRQN	62.5	32.0	28.3	12.7	26.5	+10.3	58.8	31.0	36.7	21.4	27.4	-2.6	+3.9
Tr-DRQN+	65.0	30.0	35.0	11.8	18.3	+10.7	58.8	33.0	33.3	19.5	30.6	-3.4	+3.6
Input	GATA												
★	70.0	20.0	20.0	18.6	26.3	-0.2	62.5	32.0	46.7	**27.7**	35.4	**+16.1**	+8.0
★♠	66.2	**48.0**	26.7	15.5	26.3	+24.8	**66.2**	**36.0**	**58.3**	14.1	**45.0**	**+16.1**	+20.4
∞	**73.8**	42.0	26.7	**20.9**	24.5	+27.1	62.5	30.0	51.7	23.6	36.0	+13.2	+20.2
∞♠	68.8	33.0	**41.7**	17.7	27.0	**+34.9**	62.5	33.0	46.7	25.9	33.4	+13.6	**+24.2**
	GATA-GTP												
♣	56.2	26.0	40.0	17.3	17.7	+16.6	37.5	31.0	45.0	13.6	18.7	-18.9	-1.2
♣♠	65.0	32.0	41.7	12.3	23.5	+24.6	62.5	32.0	51.7	21.8	23.5	+5.2	+14.9
	GATA-GTF												
♢	48.7	61.0	46.7	23.6	28.9	+64.2	95.0	95.0	70.0	37.3	52.8	+99.0	+81.6

Tr-DRQN+ outperform Tr-DQN, with 3.9% and 3.6% relative improvement (% ↑). This suggests the implicit memory of the recurrent components improves performance. We also observe GATA substantially outperforms Tr-DQN when trained on 100 games, whereas the DRQN agents struggle to optimize rewards on the larger training sets.

Q2: Performance of GATA compared to models with access to the ground-truth graph

Table 4.2 also reports test performance for GATA-GTP (♣) and GATA-GTF (♢). Consistent with GATA, we find GATA-GTP also performs better when given text observations (♠) as

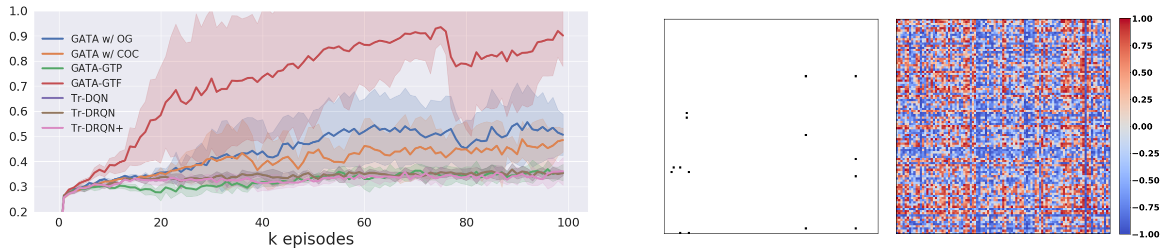


Figure 4.1: **Left:** Training curves on 20 level 2 games (averaged over 3 seeds). **Right:** Density comparison between a ground-truth graph (binary) and a belief graph $\mathcal{G}$ generated by the COC pre-training procedure. Both matrices are slices of adjacency tensors corresponding the `is` relation.

additional input to the action scorer. Although GATA-GTP outperforms Tr-DQN by 14.9% when text observations are available, its overall performance is still substantially poorer than GATA. Although the graph updater in GATA-GTP is trained with ground-truth graphs, we believe the discrete belief graphs and the discrete operations for updating them make this approach vulnerable to an accumulation of errors over game steps, as well as errors introduced by the discrete nature of the predictions (e.g., round-off error). In contrast, we suspect that the continuous belief graph and the learned graph operation function (Eqn. 3.2) are easier to train and recover more gracefully from errors.

Meanwhile, GATA-GTF, which uses ground-truth graphs $\mathcal{G}^{\text{full}}$ during training and testing, obtains significantly higher scores than does GATA and all other baselines. Because $\mathcal{G}^{\text{full}}$ turns the game environment into a fully observable MDP and encodes accurate state information with no error accumulation, GATA-GTF represents the performance upper-bound of all the $\mathcal{G}_t$-based baselines. The scores achieved by GATA-GTF reinforce our intuition that belief graphs improve text-based game agents. At the same time, the performance gap between GATA and GATA-GTF invites investigation into better ways to learn accurate graph representations of text.

4.2 Additional Results

We also show the agents' training curves and examples of the belief graphs $\mathcal{G}$ generated by GATA. Figure 4.1 (**Left**) shows an example of all agents' training curves. We observe consistent trends with the testing results of Table 4.2 — GATA outperforms the text-based baselines and GATA-GTP, but a significant gap exists between GATA and GATA-GTF (which uses ground-truth graphs as input to the action scorer). Figure 4.1 (**Right**) highlights the sparsity of a ground-truth graph compared to that of a belief graph $\mathcal{G}$. Since generation of $\mathcal{G}$ is unsupervised by any ground-truth graphs, we do not expect $\mathcal{G}$ to be interpretable nor sparse. Further, since the self-supervised models learn belief graphs directly from text, some of the learned features may correspond to the underlying grammar or other features useful for the self-supervised tasks, rather than only being indicative of relationships between objects. However, we show $\mathcal{G}$ encodes useful information for a relation prediction probing task in Section 4.2.4.

We report the training curves of all our mentioned experiment settings. Figure 4.2 shows the GATA's training curves. Figure 4.3 shows the training curves of the three text-based baseline (Tr-DQN, Tr-DRQN, Tr-DRQN+). Figure 4.4 shows the training curve of GATA-GTF (no graph updater, the action scorer takes ground-truth graphs as input) and GATA-GTP (graph updater is trained using ground-truth graphs *from the* FTWP *dataset*, the trained graph updater maintains a discrete belief graph throughout the RL training).

4.2.1 Performance on Graph Encoder Pre-training Tasks

We provide test performance of all the models described above for graph representation learning. We fine-tune the models on validation set and report their performance on test set.

Additionally, as mentioned in Section 3.4, we adapt the original R-GCN to condition the graph representation on additional information contained by the relation labels. We show an ablation study for this in Table 4.3, where R-GCN denotes the original R-GCN [59] and R-GCN w/ R-Emb denotes our version that considers relation labels.

Note, as mentioned in previous sections, the dataset to train, valid and test these four pre-training tasks are extracted from the *FTWP* dataset. There exist unseen nodes (ingredients in recipe) in the validation and test sets of *FTWP*, it requires strong generalizability to get decent performance on these datasets.

From Table 4.3, we show the relation label representation significantly boosts the generalization performance on these datasets. Compared to AP and SP, where relation label information has significant effect, both models perform near perfectly on the DGI task. This suggests the corruption function we consider in this work is somewhat simple, we leave this for future exploration.

Table 4.3: Test performance of models on all pre-training tasks.

Task	Graph Type	R-GCN	R-GCN w/ R-Emb
Accuracy			
AP	full	0.472	**0.891**
	seen	0.631	**0.873**
SP	full	0.419	**0.926**
	seen	0.612	**0.971**
DGI	full	0.999	**1.000**
	seen	**1.000**	**1.000**

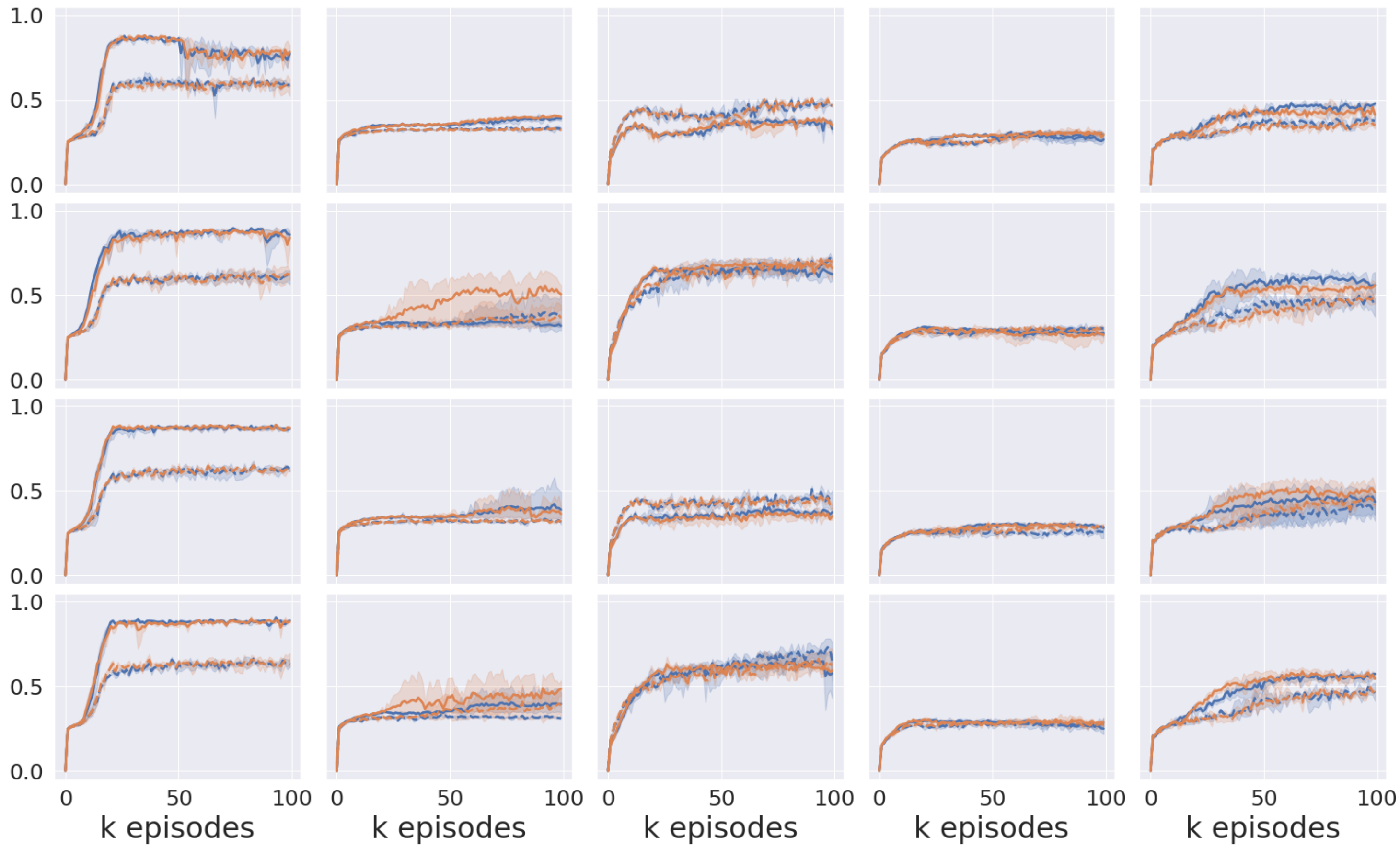


Figure 4.2: GATA's training curves (averaged over 3 seeds, band represents standard deviation). Columns are difficulty levels 1/2/3/4/5. The upper two rows are GATA using belief graphs generated by the graph updater pre-trained with observation generation task; The lower two rows are GATA using belief graphs generated by the graph updater pre-trained with contrastive observation classification task. In the 4 rows, the presence of text observation are False/True/False/True. In the figure, blue lines indicate the graph encoder in action selector is randomly initialized; orange lines indicate the graph encoder in action selector is initialized by the pre-trained observation generation and contrastive observation classification tasks. Solid lines indicate 20 training games, dashed lines indicate 100 training games.

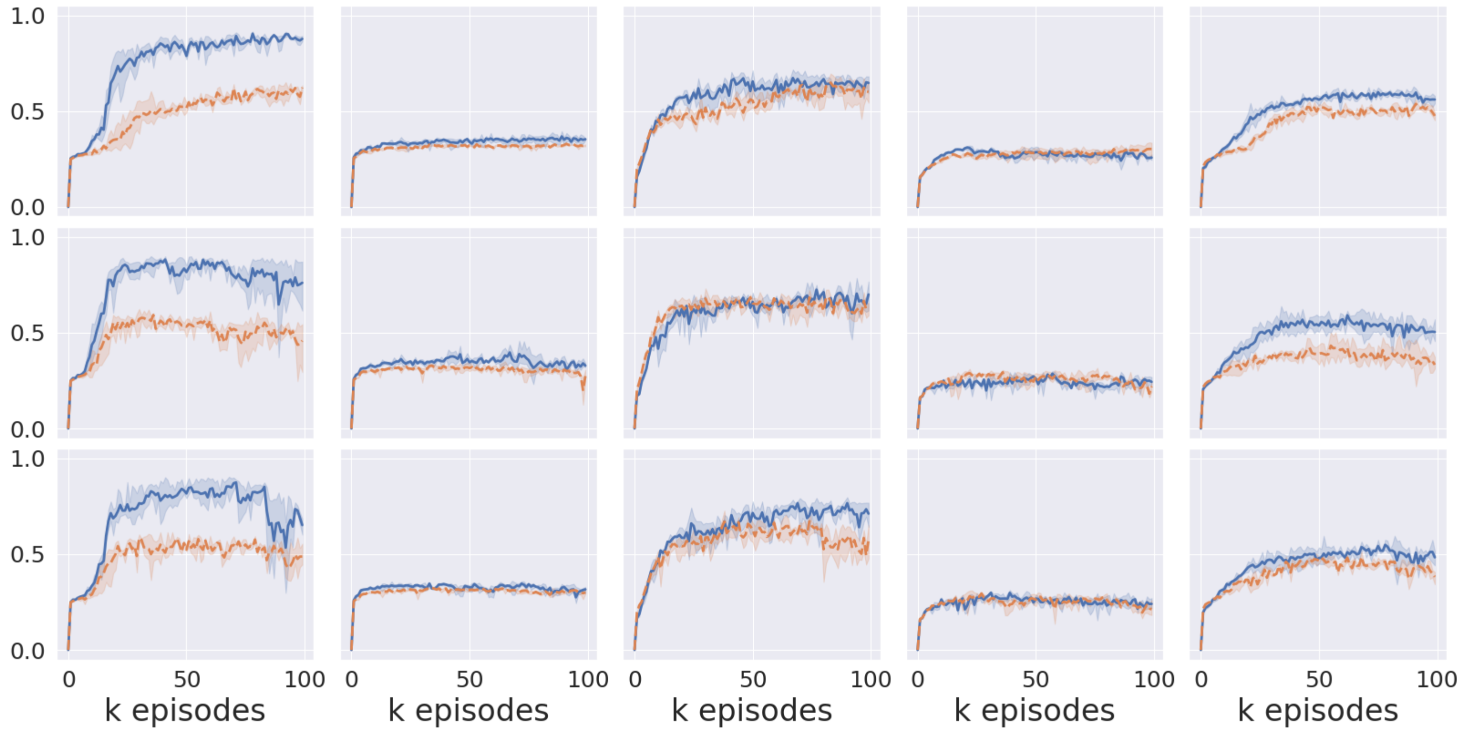


Figure 4.3: The text-based baseline agents' training curves (averaged over 3 seeds, band represents standard deviation). Columns are difficulty levels 1/2/3/4/5, rows are Tr-DQN, Tr-DRQN and Tr-DRQN+, respectively. All of the three agents take text observation O_t as input. In the figure, blue solid lines indicate the training set with 20 games; orange dashed lines indicate the training set with 100 games.

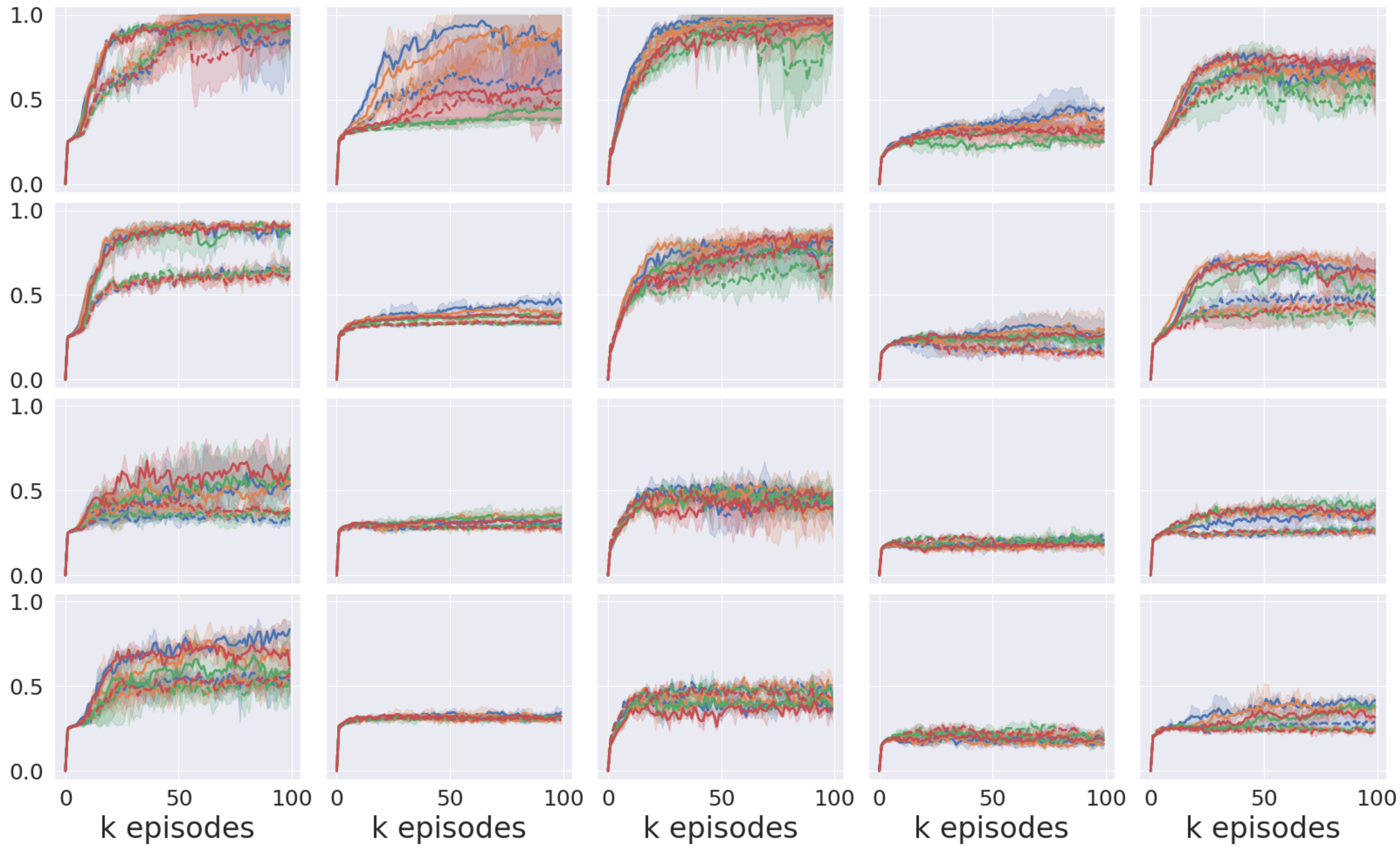


Figure 4.4: GATA-GTP and GATA-GTF's training curves (averaged over 3 seeds, band represents standard deviation). Columns are difficulty levels 1/2/3/4/5. The upper two rows are GATA-GTF when text observation is absent and present as input; the lower two rows are GATA-GTP when text observation is absent and present as input. In the figure, blue/orange/green indicate the agent's graph encoder is initialized with AP/SP/DGI pre-training tasks. Red lines indicate the graph encoder is randomly initialized. Solid lines indicate 20 training games, dashed lines indicate 100 training games.

4.2.2 Training Scores

In Table 4.4 we provide all agents' max training scores, each score is averaged over 3 random seeds. All scores are normalized. Note as described in Section 3.4, we use ground-truth KGs to train the action selector, $\mathcal{G}^{\text{belief}}$ is only used during evaluation.

4.2.3 Test Results

In Table 4.5 we provide all our agent variants and the text-based baselines' test scores. We report agents' test score corresponding to their best validation scores.

Observations

Pre-training graph encoder helps. In Table 4.4 and Table 4.5, we also show GATA, GATA-GTP and GATA-GTF's training and test scores when their action scorer's graph encoder are initialized with pre-trained parameters as introduced in Section 3.2 (for GATA) and Section 3.6.1 (for GATA-GTP and GATA-GTF). We observe in most settings, pre-trained graph encoders produce better training and test results compared to their randomly initialized counterparts. This is particularly obvious in GATA-GTP and GATA-GTF, where graphs are discrete. For instance, from Table 4.5 we can see that only with text observation as additional input (♣♠), and when graph encoder are initialized with AP/SP/DGI, the GATA-GTP agent can outperform the text-based baselines on test game sets.

Note however, that pre-training the graph encoder with OG and COC do not help GATA. This is because, OG and COC, unlike AP and SP are much different objectives than what the action selector is required to do, i.e. optimize rewards for RL. As a result, we find OG/COC not necessarily helping for the RL task when used to initialize the graph encoder in the action selector.

Fine-tuning graph encoder helps. For all experiment settings where the graph encoder in action scorer is initialized with pre-trained parameters (OG/COC for GATA, AP/SP/DGI for GATA-GTP), we also compare between freezing vs. fine-tuning the graph encoder in RL training. By freezing the graph encoders, we can effectively reduce the number of parameters to be optimized with RL signal. However, we see consistent trends that fine-tuning the graph encoders can always provide better training and testing performance in both GATA and GATA-GTP.

Text input helps more when graphs are imperfect. We observe clear trends that for GATA-GTF, using text together with graph as input (to the action selector) does not provide obvious performance increase. Instead, GATA-GTF often shows better performance when text observation input is disabled. This observation is coherent with the intuition of using text observations as additional input. When the input graph to the action selector is imperfect (e.g., belief graph maintained by GATA or GATA-GTP itself), the text observation provides more accurate information to help the agent to recover from errors. On the other hand, GATA-GTF uses the ground-truth full graph (which is even accurate than text) as input to the action selector, the text observation might confuse the agent by providing redundant information with more uncertainty.

Learning across difficulty levels. We have a special set of RL games — level 5 — which is a mixture of the other four difficulty levels. We use this set to evaluate an agent's generalizability on both dimensions of game configurations and difficulty levels. From Table 4.4, we observe that almost all agents (including baseline agents) benefit from a larger training set, i.e., achieve better test results when train on 100 level 5 games than 20 of them. Results show GATA has a more significant performance boost from larger training set. We notice that all GATA-GTP variants perform worse than text-based baselines on level 5 games, whereas GATA outperforms text-based baselines when training on 100 games. This may suggest the continuous belief graphs can better help GATA to adapt to games across difficulty levels, whereas its discrete counterpart may struggle more. For example, both games in level 1 and 2 have only single location, while level 3 and 4 games have multiple locations. GATA-GTP might thus get confused since sometimes the direction relations (e.g., `west_of`) are unused. In contrast, GATA, equipped with continuous graphs, may learn such scenario easier.

Table 4.4: Agents' **Max** performance on **Training** games, averaged over 3 random seeds. In this table, ♠, ◇represent O_t and $\mathcal{G}_t^{\text{full}}$, respectively. ♣represents discrete belief graph generated by GATA-GTP (trained with ground-truth graphs of *FTWP*). ⋆and ∞indicate continuous belief graph generated by GATA, pre-trained with observation generation (OG) task and contrastive observation classification (COC) task, respectively. Light blue shadings represent numbers that are greater than or equal to Tr-DQN; light yellow shading represent number that are greater than or equal to all of Tr-DQN, Tr-DRQN and Tr-DRQN+.

		20 Training Games						100 Training Games						Avg.
Difficulty Level		1	2	3	4	5	% ↑	1	2	3	4	5	% ↑	% ↑
Input	Agent	Text-based Baselines												
♠	Tr-DQN	90.8	36.9	69.3	31.5	61.2	——	63.4	33.2	66.1	31.9	55.2	——	——
♠	Tr-DRQN	88.8	41.7	76.6	29.6	60.7	+2.9	60.8	33.7	71.7	30.6	44.9	-3.4	-0.2
♠	Tr-DRQN+	89.1	35.6	78.0	30.9	58.1	+0.0	61.1	32.8	70.0	30.0	50.3	-2.8	-1.4
	Pre-training	GATA												
⋆	N/A	87.9	40.4	40.1	30.8	50.1	-11.2	65.1	34.6	51.2	32.0	41.8	-7.9	-9.6
⋆	OG	88.8	40.8	40.1	32.1	48.2	-10.6	63.8	33.9	51.9	32.6	39.3	-9.1	-9.8
⋆♠	N/A	90.2	35.4	69.0	32.0	62.8	-0.2	63.9	41.2	72.2	32.2	50.8	+5.4	+2.6
⋆♠	OG	90.0	57.1	70.6	31.7	57.9	+10.2	64.2	38.9	72.5	32.4	50.1	+4.1	+7.1
∞	N/A	89.0	43.1	41.2	31.8	48.7	-9.0	65.8	33.4	51.5	29.1	44.0	-9.4	-9.2
∞	COC	89.6	41.0	39.2	31.9	54.4	-8.7	65.8	33.4	48.6	31.2	47.2	-7.8	-8.2
∞♠	N/A	90.9	41.4	66.1	31.3	58.8	+0.6	67.1	33.1	73.6	29.9	51.2	+0.7	+0.6
∞♠	COC	90.2	50.8	66.4	31.9	59.3	+6.2	67.4	41.5	66.6	31.4	50.8	+4.5	+5.4
		GATA-GTP												
♣	N/A	73.3	34.5	50.5	21.7	43.5	-22.6	49.3	31.1	54.3	25.1	28.8	-23.1	-22.8
♣	AP	68.4	34.8	61.3	23.8	43.1	-19.2	40.9	31.3	55.4	24.8	28.8	-25.7	-22.4
♣	SP	62.7	38.1	57.5	23.5	44.1	-19.6	50.2	30.8	55.0	23.7	28.0	-24.0	-21.8
♣	DGI	64.9	37.0	55.6	25.7	47.4	-17.8	43.4	31.8	58.3	25.3	30.2	-22.7	-20.3
♣♠	N/A	77.5	33.9	45.6	26.3	40.2	-21.6	59.5	32.3	55.4	29.1	27.6	-16.8	-19.2
♣♠	AP	87.5	35.8	50.4	22.3	45.4	-17.8	61.3	32.2	56.3	25.3	33.0	-16.4	-17.1
♣♠	SP	80.0	35.5	50.2	23.5	44.0	-19.4	57.3	32.1	58.3	27.1	29.2	-17.4	-18.4
♣♠	DGI	70.3	33.9	51.4	26.3	42.1	-20.9	57.7	32.7	55.6	28.8	29.8	-16.4	-18.6
		GATA-GTF												
◇	N/A	98.6	58.4	95.6	36.1	80.9	+30.3	96.0	53.4	97.9	36.0	76.4	+42.3	+36.3
◇	AP	98.7	97.5	98.3	48.1	79.3	+59.4	97.1	74.7	98.3	44.5	75.9	+60.8	+60.1
◇	SP	100.0	96.9	98.3	44.9	76.6	+56.5	98.6	90.5	99.0	38.9	73.4	+66.6	+61.5
◇	DGI	96.9	45.4	95.3	28.7	72.6	+15.4	98.2	39.1	90.1	33.0	62.4	+25.1	+20.2
◇♠	N/A	91.7	55.9	80.9	33.6	63.2	+15.8	73.5	48.1	67.7	31.8	56.7	+13.1	+14.5
◇♠	AP	87.9	62.4	78.8	32.4	62.8	+17.0	76.8	54.0	73.7	34.1	55.6	+20.6	+18.8
◇♠	SP	90.7	55.8	83.8	30.7	64.2	+14.9	60.4	40.1	67.4	31.1	51.5	+1.8	+8.3
◇♠	DGI	88.1	38.1	73.0	32.5	62.5	+2.2	66.4	35.5	59.1	30.4	49.6	-2.8	-0.3

Table 4.5: Agents' performance on **test** games, model selected using best validation performance. **Boldface** and underline represent the highest and second highest values in a setting (excluding GATA-GTF which has access to the ground-truth graphs of the RL games). In this tabel, ♠, ◇represent O_t and $\mathcal{G}_t^{\text{full}}$, respectively. ♣represents discrete belief graph generated by GATA-GTP (pre-trained with ground-truth graphs of *FTWP*). ⋆and ∞indicate continuous belief graph generated by GATA, pre-trained with observation generation (OG) task and contrastive observation classification (COC) task, respectively. Light blue shadings represent numbers that are greater than or equal to Tr-DQN; light yellow shading represent number that are greater than or equal to all of Tr-DQN, Tr-DRQN and Tr-DRQN+. Note that this table is an elaborate version of Table 4.2 to compare amongst the pre-training methods.

		20 Training Games						100 Training Games						Avg.
Difficulty Level		1	2	3	4	5	% ↑	1	2	3	4	5	% ↑	% ↑
Input	Agent	Text-based Baselines												
♠	Tr-DQN	66.2	26.0	16.7	18.2	**27.9**	—	62.5	32.0	38.3	17.7	34.6	—	—
♠	Tr-DRQN	62.5	32.0	28.3	12.7	26.5	+10.3	58.8	31.0	36.7	21.4	27.4	-2.6	+3.9
♠	Tr-DRQN+	65.0	30.0	35.0	11.8	18.3	+10.7	58.8	33.0	33.3	19.5	30.6	-3.4	+3.6
	Pre-training	GATA												
⋆	N/A	70.0	20.0	20.0	18.6	26.3	-0.2	62.5	32.0	46.7	**27.7**	35.4	+16.1	+8.0
⋆	OG	66.2	28.0	21.7	15.9	24.3	+2.4	66.2	34.0	40.0	21.4	34.0	+7.2	+4.8
⋆♠	N/A	66.2	34.0	30.0	12.7	24.3	+13.5	66.2	**38.0**	36.7	27.3	36.1	+15.8	+14.6
⋆♠	OG	66.2	**48.0**	26.7	15.5	26.3	+24.8	66.2	36.0	**58.3**	14.1	**45.0**	+16.1	+20.4
∞	N/A	73.8	42.0	26.7	**20.9**	24.5	+27.1	62.5	30.0	51.7	23.6	36.0	+13.2	+20.2
∞	COC	66.2	29.0	30.0	18.2	27.7	+18.1	66.2	34.0	41.7	19.1	40.3	+9.1	+13.6
∞♠	N/A	68.8	33.0	41.7	17.7	27.0	+34.9	62.5	33.0	46.7	25.9	33.4	+13.6	+24.2
∞♠	COC	66.2	44.0	16.7	20.0	21.7	+11.4	**70.0**	34.0	45.0	12.3	36.2	+2.0	+6.7
		GATA-GTP												
♣	N/A	56.2	23.0	41.7	11.4	22.1	+13.0	45.0	32.0	30.0	10.5	17.4	-28.0	-7.5
♣	AP	50.0	20.0	25.0	9.5	24.3	-11.7	45.0	31.0	50.0	15.9	24.4	-8.0	-9.9
♣	SP	45.0	25.0	38.3	11.8	22.6	+7.9	42.5	32.0	50.0	11.4	22.5	-14.4	-3.3
♣	DGI	56.2	26.0	40.0	17.3	17.7	+16.6	37.5	31.0	45.0	13.6	18.7	-18.9	-1.2
♣♠	N/A	73.8	31.0	28.3	8.2	22.5	+5.2	62.5	29.0	38.3	13.2	19.8	-15.5	-5.2
♣♠	AP	62.5	32.0	**46.7**	12.3	21.1	+28.1	58.8	30.0	40.0	10.9	29.2	-12.4	+7.9
♣♠	SP	65.0	32.0	41.7	12.3	23.5	+24.6	62.5	32.0	51.7	21.8	23.5	+5.2	+14.9
♣♠	DGI	**75.0**	27.0	33.3	17.3	24.3	+19.7	62.5	31.0	46.7	19.5	24.7	+0.1	+9.9
		GATA-GTF												
◇	N/A	83.8	53.0	33.3	23.6	24.8	+49.7	100.0	90.0	68.3	37.3	52.7	+96.5	+73.1
◇	AP	85.0	39.0	26.7	26.4	27.5	+36.4	92.5	88.0	63.3	53.6	51.6	+108.0	+72.2
◇	SP	48.7	61.0	46.7	23.6	28.9	+64.2	95.0	95.0	70.0	37.3	52.8	+99.0	+81.6
◇	DGI	85.0	27.0	31.7	14.1	22.1	+15.7	100.0	40.0	70.0	31.8	50.6	+58.7	+37.2
◇♠	N/A	92.5	39.0	30.0	15.9	23.6	+28.3	96.3	56.0	55.0	14.5	46.6	+37.9	+33.1
◇♠	AP	73.8	36.0	46.7	25.9	23.9	+51.5	85.0	42.0	68.3	36.4	47.5	+57.7	+54.6
◇♠	SP	62.5	24.0	36.7	14.5	28.6	+17.7	60.0	43.0	46.7	25.0	47.9	+26.4	+21.1
◇♠	DGI	81.2	30.0	25.0	16.8	30.7	+18.0	73.8	39.0	48.3	15.0	40.6	+13.6	+15.8

4.2.4 Probing Task and Belief Graph Visualization

In this section, we investigate whether generated belief graphs contain any useful information about the game dynamics. We first design a probing task to check if $\mathcal{G}$ encodes the existing relations between two nodes. Next, we visualize a few slices of the adjacency tensor associated to $\mathcal{G}$.

Probing Task

We frame the probing task as a multi-label classification of the relations between a pair of nodes. Concretely, given two nodes i, j, and the vector $\mathcal{G}_{i,j} \in [-1,1]^{\mathcal{R}}$ (in which $\mathcal{R}$ denotes the number of relations) extracted from the belief graph $\mathcal{G}$ corresponding to the nodes i and j, the task is to learn a function f such that it minimizes the following binary cross-entropy loss:

$$\mathcal{L}_{\text{BCE}}\left(f(\mathcal{G}_{i,j}, h_i, h_j), Y_{i,j}\right), \tag{4.1}$$

where h_i, h_j are the embeddings for nodes i and j, $Y_{i,j} \in \{0,1\}^{\mathcal{R}}$ is a binary vector representing the presence of each relation between the nodes (there are R different relations). Following Alain and Bengio 2017 [4], we use a linear function as f, since we assume the useful information should be easily accessible from $\mathcal{G}$.

We collect a dataset for this probing task by following the walkthroughs of 120 games. At every game step, we collect a tuple $(\mathcal{G}, \mathcal{G}^{\text{seen}})$ (see Section 2.7.1 for the definition of $\mathcal{G}^{\text{seen}}$). We used tuples from 100 games as training data and the remaining for evaluation.

From each tuple in the dataset, we extract several node pairs (i,j) and their corresponding $Y_{i,j}$ from $\mathcal{G}^{\text{seen}}$ (positive examples, denoted as "+"). To make sure a model can only achieve good performance on this probing task by using the belief graph $\mathcal{G}$, without overfitting by memorising node-relation pairs (e.g., the unique relation between player and kitchen is `at`), we augment the dataset by adding plausible node pairs (i.e., $Y_{i,j} = \vec{0}^{\mathcal{R}}$) but that have no relation according to the current $\mathcal{G}$ (negative examples, denoted as "−"). For instance, if at a certain game step the player is in the bedroom, the relation between the player and kitchen should be empty ($\vec{0}^{\mathcal{R}}$). We expect $\mathcal{G}$ to have captured that information.

We use two metrics to evaluate the performance on this probing task:

- **Exact match** represents the percentage of predictions that have all their labels classified correctly, i.e., when $f(\mathcal{G}_{i,j}, h_i, h_j) = Y^n_{i,j}$.
- **F$_1$ score** which is the harmonic mean between precision and recall. We report the macro-averaging of F$_1$ over all the predictions.

Table 4.6: Probing task results showing that belief graphs obtained from OG and COC do contain information about the game dynamics, i.e. node relationships.

	Exact Match						F_1 score					
	Train			Test			Train			Test		
Model	+	−	Avg	+	−	Avg	+	−	Avg	+	−	Avg
Random	0.00	0.99	0.49	0.00	0.99	0.49	0.00	0.99	0.49	0.00	0.99	0.49
Ground-truth	0.98	0.96	0.97	0.97	0.96	0.97	0.98	0.96	0.97	0.98	0.96	0.97
Tr-DRQN	0.61	0.84	0.73	0.61	0.83	0.72	0.61	0.84	0.73	0.61	0.83	0.72
GATA (OG)	0.69	0.86	**0.78**	0.70	0.86	**0.78**	0.71	0.85	**0.78**	0.72	0.86	**0.79**
GATA (COC)	0.65	0.86	0.75	0.65	0.84	0.75	0.67	0.85	0.76	0.67	0.84	0.75

To better understand the probe's behaviors on each settings, we also report their training and test performance on the positive samples (+) and negative samples (−) separately.

From Table 4.6, we observe that belief graphs $\mathcal{G}$ generated by models pre-trained with either OG or COC do contain useful information about the relations between a pair of nodes. We first compare against a random baseline where each $\mathcal{G}$ is randomly sampled from $\mathcal{N}(0, 1)$ and kept fixed throughout the probing task. We observe the linear probe fails to perform well on the training set (and as a result also fails to generalize on test set). Interestingly, with random belief graphs provided, the probe somehow overfits on negative samples and always outputs zeros all the time. In both training and testing phases, it produces zero performance on positive examples. This baseline suggests the validity of our probing task design — there is no way to correctly predict the relations without having the information encoded in the belief graph $\mathcal{G}$.

Next, we report the performance of using ground-truth graphs ($\mathcal{G}^{\text{seen}}$) as input to f. We observe the linear model can perform decently on training data, and can generalize from training to testing data — on both sets, the linear probe achieves near-perfect performance. This also verifies the probing task by showing that given the ground-truth knowledge, the linear probe is able to solve the task easily.

Given the two extreme cases as upper bound and lower bound, we investigate the belief graphs $\mathcal{G}$ generated by GATA, pre-trained with either of the two self-supervised methods, OG and COC (proposed in Section 3.2). From Table 4.6, we can see $\mathcal{G}$ generated by both OG and COC methods help similarly in the relation prediction task, both provide more than 75% of testing exact match scores.

In Section 4, we show that GATA outperforms a set of baseline systems, including Tr-DRQN (as described in Section 4.1), an agent with recurrent components. To further investigate if the belief graphs generated by GATA can better facilitate the linear probe in this relation prediction task, we provide an additional setting. We modify the Tr-DRQN agent by replacing its action scorer by a text generator (the same decoder used in OG training), and train this model with the same data and objective as OG. After pre-training, we obtain a set of probing data by collecting the recurrent hidden states produced by this agent given the same probing game walkthroughs. Since these recurrent hidden states are computed from the same amount of information as GATA's belief graphs, they could theoretically contain the same information as $\mathcal{G}$. However, from Table 4.6, we see that the scores of Tr-DRQN are consistently lower than GATA's score. This is coherent with our findings in the RL experiments (Section 4), except the gap between GATA and Tr-DRQN is less significant in the relation prediction task setting.

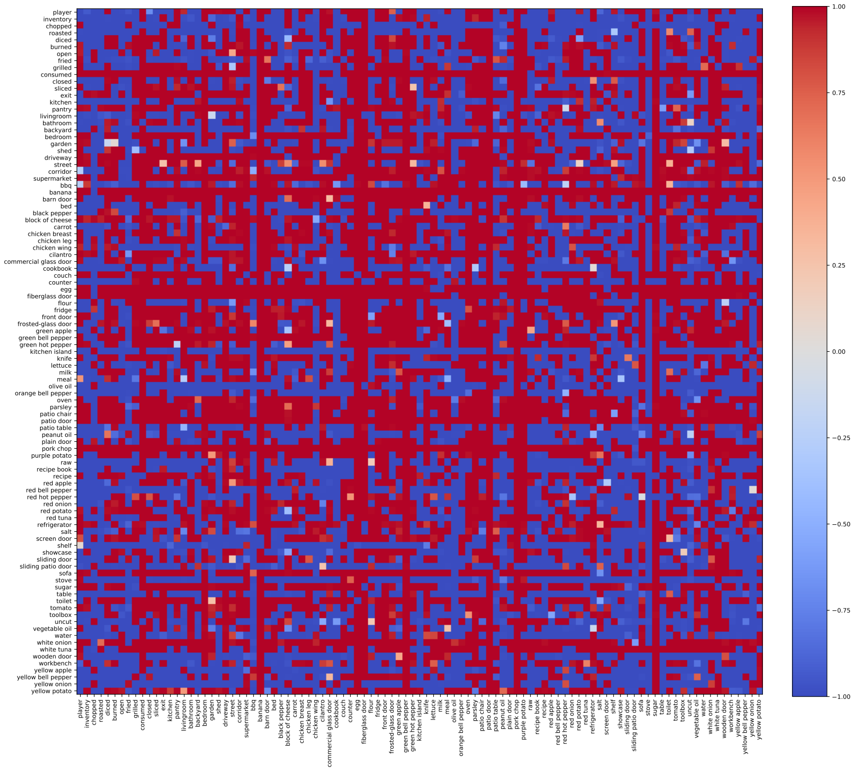
While being able to perform the classification correctly in a large portion of examples, we observe a clear performance gap comparing GATA's belief graphs with ground-truth graphs. The cause of the performance gap can be twofold. First, compared to ground-truth graphs that accurately represent game states without information loss, $\mathcal{G}$ (iteratively generated by a neural network across game steps) can inevitably suffer from information loss. Second, the information encoded in $\mathcal{G}$ might not be easily extracted by a linear probe (compared to ground-truth). Both aspects suggest potential future directions to improve the belief graph generation module.

We optimize all probing models for 10 epochs with Adam optimizer, using the default hyperparameters and a learning rate of 0.0001. Note in all the probing models, only parameters of the linear layer f are trainable, everything else (including node embeddings) are kept fixed.

Belief Graph Visualization

In Figure 2.2, we show a slice of the ground-truth adjacency tensor representing the `is` relation. To give context, that tensor has been extracted at the end of a game with a recipe requesting a fried diced red apple, a roasted sliced red hot pepper, and a fried sliced yellow potato. Correspondingly, for the same game and same time step, Figure 4.5 shows the same adjacency tensor's slice for the belief graphs $\mathcal{G}$ generated by GATA pre-trained on observation generation (OG) and contrastive observation classification (COC) tasks.

For visualization, we found that subtracting the mean adjacency tensor, computed across all games and steps, helps by removing information about the marginal distribution of the observations (e.g., underlying grammar or other common features needed for the self-supervised tasks). Those "cleaner" graphs are shown in Figure 4.6–which represent the same graphs from the Figure 4.5 subtracted with the respective baselines. One must keep in mind that there is no training signal to force the belief graphs to align with any ground-truth graphs since the belief graph generators are trained with pure self-supervised methods. Further, the higher values in these adjacency tensors may not necessarily imply strength of a relation between two nodes.



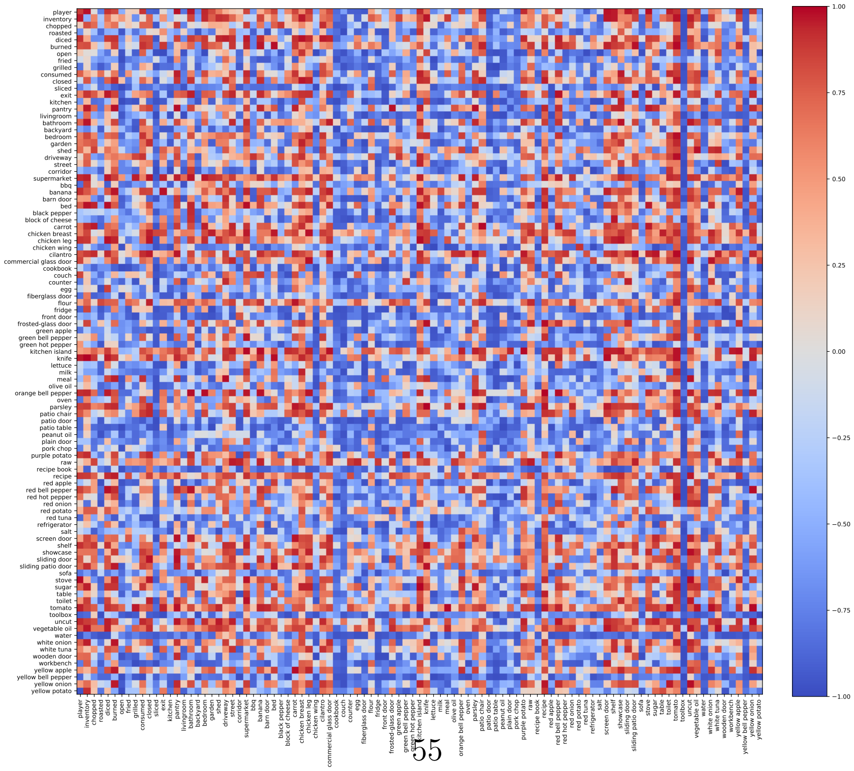


Figure 4.5: Adjacency tensor's slices for $\mathcal{G}$ generated by GATA, pre-trained with OG task **(top)** and COC task **(bottom)**.

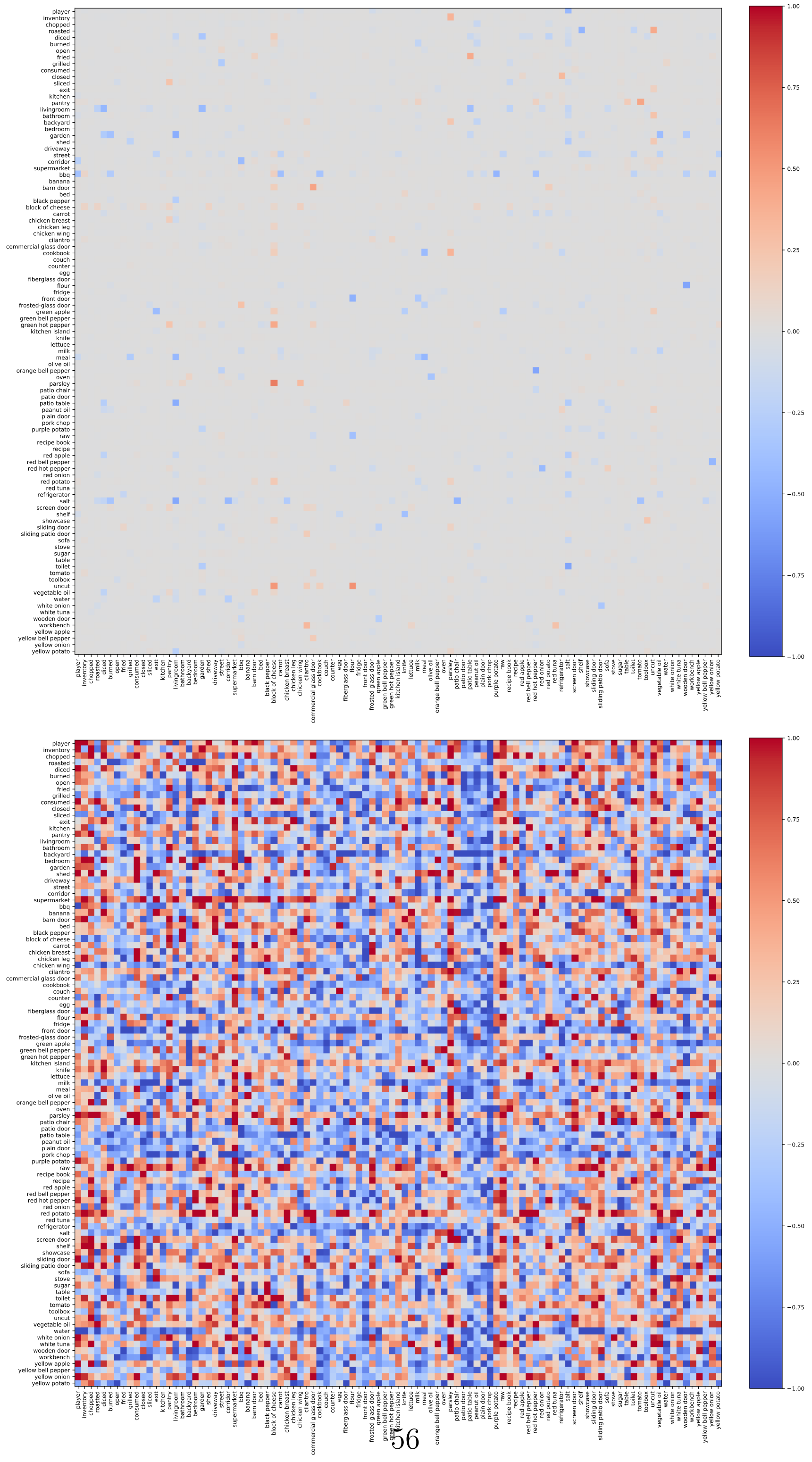


Figure 4.6: Adjacency tensor's slices after subtracting the mean for $\mathcal{G}$ generated by GATA, pre-trained with OG task **(top)** and COC task **(bottom)**.

Chapter 5

Conclusion

In this work, we investigate how an RL agent can play and generalize within a distribution of text-based games using graph-structured representations inferred from text. We introduce GATA, a novel neural agent that infers and updates latent belief graphs as it plays text-based games. We use a combination of RL and self-supervised learning to teach the agent to encode essential dynamics of the environment in its belief graphs. We show that GATA achieves good test performance, outperforming a set of strong baselines including agents pre-trained with ground-truth graphs. This evinces the effectiveness of generating graph-structured representations for text-based games.

5.1 Future Directions

As mentioned in Section 4.2.4, the belief graphs generated by GATA lack interpretability because the training is not supervised by any ground-truth graph. Technically, they are recurrent hidden states that encode the game state, we only (weakly) ground these real-valued graphs by providing node and relation vocabularies (word embeddings) for the message passing in R-GCN.

Therefore, there can be two potential directions deriving from the current approach. First, it would be interesting to investigate regularization methods and auxiliary tasks that can make the belief graph sparser (without relying on ground-truth graphs to train). A sparser belief graph may increase GATA's interpretability, however, it does not guarantee to produce better performance on playing text-based games (which is what we care more about).

Second, it would also be interesting to see how GATA can be adapted to environments where the node and relation names are unknown. This will presumably make the learned belief graphs even far away from interpretable, but at the same time it will further relax GATA from the need of requiring any prior knowledge about the environments. We believe this is an essential property for an agent that is generalizabile to out-of-distribution environments. For instance, without the need of a pre-defined node and relation vocabularies, we can expand GATA to the setting where training on the cooking games, and testing on games from another genre, or even text-based games designed for humans [27].

References

[1] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikul'avs Zelinka, M. Rondeau, R. Laroche, P. Poupart, J. Tang, Adam Trischler, and William L. Hamilton. Learning dynamic belief graphs to generalize on text-based games. *arXiv: Computation and Language*, 2020.

[2] Leonard Adolphs and Thomas Hofmann. Ledeepchef: Deep reinforcement learning agent for families of text-based games. *CoRR*, abs/1909.01646, 2019.

[3] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 5074–5082. Curran Associates, Inc., 2016.

[4] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *ArXiv*, abs/1610.01644, 2017.

[5] Prithviraj Ammanabrolu and Matthew Hausknecht. Graph constrained reinforcement learning for natural language action spaces. In *International Conference on Learning Representations*, 2020.

[6] Prithviraj Ammanabrolu and Mark 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)*, pages 3557–3565, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[7] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm. Unsupervised state representation learning in atari. In *NeurIPS*, 2019.

[8] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. Leveraging linguistic structure for open domain information extraction. In *ACL*, 2015.

[9] Ghulam Ahmed Ansari, P SagarJ., A. P. Sarath Chandar, and Balaraman Ravindran. Language expansion in text-based games. *ArXiv*, abs/1805.07274, 2018.

[10] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[11] Timothy Atkinson, Hendrik Baier, Tara Copplestone, Sam Devlin, and Jerry Swan. The text-based adventure ai competition. *IEEE Transactions on Games*, 11:260–266, 2018.

[12] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[13] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pages 15509–15519, 2019.

[14] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[15] Joan Bruna, W. Zaremba, Arthur Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2014.

[16] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[17] 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. *CoRR*, abs/1806.11532, 2018.

[18] Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. Building dynamic knowledge graphs from text using machine reading comprehension. In *International Conference on Learning Representations*, 2019.

[19] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *ArXiv*, abs/1901.10995, 2019.

[20] Angela Fan, Claire Gardent, Chloé Braud, and Antoine Bordes. Using local knowledge graph construction to scale Seq2Seq models to multi-document inputs. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4186–4196, Hong Kong, China, November 2019. Association for Computational Linguistics.

[21] Nancy Fulda, Daniel Ricks, Ben Murdoch, and David 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*, pages 1039–1045, 2017.

[22] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1994.

[23] Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 140–149, Berlin, Germany, August 2016. Association for Computational Linguistics.

[24] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. `https://worldmodels.github.io`.

[25] H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[26] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[27] Matthew Hausknecht, Prithviraj Ammanabrolu, Côté Marc-Alexandre, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[28] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (AAAI-SDMIA15)*, November 2015.

[29] Matthew J. Hausknecht, Ricky Loynd, Greg Yang, Adith Swaminathan, and Jason D. Williams. Nail: A general interactive fiction agent. *CoRR*, abs/1902.04259, 2019.

[30] Matthew J. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposia*, 2015.

[31] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari 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)*, pages 1621–1630, Berlin, Germany, August 2016. Association for Computational Linguistics.

[32] N. Heess, J. Hunt, T. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *ArXiv*, abs/1512.04455, 2015.

[33] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[34] Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR 2019*. ICLR, April 2019.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[36] Vishal Jain, William Fedus, Hugo Larochelle, Doina Precup, and Marc G. Bellemare. Algorithmic improvements for deep reinforcement learning applied to interactive fiction. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[37] Daniel D Johnson. Learning graphical state transitions. In *International Conference on Learning Representations (ICLR)*, 2017.

[38] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1–2):99–134, May 1998.

[39] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

[40] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020.

[41] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[42] Donald Knuth. *The TEXbook.* Addison-Wesley, Reading, Massachusetts, 1986.

[43] Leslie Lamport. *LATEX — A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, second edition, 1994.

[44] Pedro Lima. First textworld challenge - first place solution, 2019.

[45] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3–4):293–321, May 1992.

[46] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*, April 2020.

[47] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *IJCAI*, 2019.

[48] Andrea Madotto, Mahdi Namazifar, Joost Huizinga, Piero Molino, Adrien Ecoffet, Huaixiu Zheng, Alexandros Papangelis, Dian Yu, Chandra Khatri, and Gokhan Tur. Exploration based language learning for text-based games, 2020.

[49] Rui Meng, Xingdi Yuan, Tong Wang, Peter Brusilovsky, Adam Trischler, and Daqing He. Does order matter? an empirical study on generating multiple keyphrases as a sequence. *CoRR*, 2019.

[50] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of*

the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[52] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[53] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2790–2799. JMLR.org, 2016.

[54] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[55] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[56] Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alex Peysakhovich, Kyunghyun Cho, and Joan Bruna. Backplay:" man muss immer umkehren". *arXiv preprint arXiv:1807.06919*, 2018.

[57] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.

[58] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

[59] 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.

[60] John Schulman, F. Wolski, Prafulla Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[61] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[62] Mathieu Seurin, Philippe Preux, and Olivier Pietquin. "i'm sorry dave, i'm afraid i can't do that" deep q-learning from forbidden action. *CoRR*, abs/1910.02078, 2019.

[63] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

[64] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[65] Richard Sutton. *The Bitter Lesson*, 2019.

[66] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[67] Chen Tessler, Tom Zahavy, Deborah Anne Cohen, Daniel J. Mankowitz, and Shie Mannor. Action assembly: Sparse imitation learning for text based games with combinatorial action spaces. *ArXiv*, abs/1905.09700, 2019.

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

[69] Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. Learning to speak and act in a fantasy text adventure game. *CoRR*, abs/1903.03094, 2019.

[70] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018.

[71] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2015.

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[73] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. In *International Conference on Learning Representations*, 2019.

[74] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[75] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

[76] Zhilin Yang, Jake Zhao, Bhuwan Dhingra, Kaiming He, William W Cohen, Russ R Salakhutdinov, and Yann LeCun. Glomo: Unsupervised learning of transferable relational graphs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8950–8961. Curran Associates, Inc., 2018.

[77] Xusen Yin and Jonathan May. Comprehensible context-driven text game playing. In *2019 IEEE Conference on Games (CoG)*, pages 1–8. IEEE, 2019.

[78] Xusen Yin and Jonathan May. Learn how to cook a new recipe in a new house: Using map familiarization, curriculum learning, and bandit feedback to learn families of text-based adventure games. *CoRR*, abs/1908.04777, 2019.

[79] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. In *International Conference on Learning Representations*, 2018.

[80] Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Christopher Pal, Yoshua Bengio, and Adam Trischler. Interactive language learning by question answering. 2019.

[81] Xingdi Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. Counting to explore and generalize in text-based games. *arXiv preprint arXiv:1806.11525*, 2018.

[82] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3562–3573, 2018.

[83] Mikulás Zelinka. Baselines for reinforcement learning in text games. *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 320–327, 2018.

[84] Mikulas Zelinka, Xingdi Yuan, Marc-Alexandre Cote, Romain Laroche, and Adam Trischler. Building dynamic knowledge graphs from text-based games. *arXiv preprint arXiv:1910.09532*, 2019.