Exploring Generalization in Text-Based Reinforcement Learning

Bryon Kucharski
College of Information & Computer Sciences
University of Massachusetts Amherst
bkucharski@umass.edu

1 Introduction

This project explores generalization at the intersection of natural language processing and reinforcement learning through text-based interactive fiction games. All information given to and all actions taken by agents are solely through text. Agents in this domain must contain a general commonsense understanding of the world and the objects around it. Much of the current work in this domain lacks an evaluation of general understanding, making it unknown if learned policies have the ability to generalize to unseen games or different configurations of games.

Section 1.1 will introduce the problem definition and environment used throughout the project. Section 2 explains previous models which serve as inspiration for a new model described in Section 3. An evaluation of generalization is conducted in Section 5 as well as directions of future work in Section 6.

The code can be found at [1]

1.1 Problem Definition

A text-based game can be defined as a Partially Observable Markov Decision Process (POMDP) represented by the tuple $(\mathcal{S}, \mathcal{T}, \mathcal{A}, \Omega, \mathcal{O}, \mathcal{R})$. $\mathcal{S}$ is the set of all possible states, $\mathcal{T}$ is the transition function of the next state given a current state and action, $\mathcal{A}$ is the set of all possible actions, Ω is the set of possible observations, $\mathcal{O}$ defines the probability of the current observation given a state and previous actions, and $\mathcal{R}$ defines the reward function for taking an action in a given state. Additionally, the set of admissible actions A_t is defined to be the set of actions that are recognized by the environment and potentially change the current game state.

Jericho (Hausknecht et al., 2019) is a learning environment for reinforcement learning agents to interface with human-made interactive fiction games, such as the *Zork* series from Infocom. Previous works have used Jericho as a test bench for text-based games, similar to the Atari Learning Environment (Bellemare et al., 2013) for image-based agents. Agents in this domain are trained on instances of a single game and lack a notion of generalization. TextWorld (Côté et al., 2018) is a sandbox learning environment for designing and playing text-based games and is the environment used in this work. TextWorld allows for the creation of many similar instances of the same type of game, which serves as an appropriate environment for testing generalization in reinforcement learning.

The First TextWorld Problems was a competition by Microsoft Research designed to test reinforcement learning agent's ability to generalize in text-based games. The competition provided a training, validation, and test set of games which all shared a common theme of cooking a recipe. Agents must collect the necessary ingredients, process the ingredients as defined in the recipe, and finally eat the meal to solve a game. The dataset was used for pre-training tasks described in Section 3.2.

The same cooking theme from the competition, but a different set of games described in Section 4, is used in this work. At each step, an observation $o_t \in \Omega$ is provided to the agent. The observation o_t consists of a text description of the current room o_{desc}, the current inventory o_{inv}, the feedback from TextWorld from the previous action taken $o_{feedback}$, and the recipe required to solve the game

[1] https://github.com/bryonkucharski/text-based-rl-generalization

Observation	Example Text
Description O_{desc}	-= Kitchen =-If you're wondering why everything seems so normal all of a sudden, it's because you've just shown up in the kitchen. You can see a closed fridge, which looks conventional, right there by you. You see a closed oven right there by you. Oh, great. Here's a table. Unfortunately, there isn't a thing on it. Hm. Oh well You scan the room, seeing a counter. The counter is vast. On the counter you can make out a cookbook and a knife. You make out a stove. Looks like someone's already been here and taken everything off it, though. Sometimes, just sometimes, TextWorld can just be the worst.
Inventory O_{inv}	You are carrying: a banana
Recipe O_{recipe}	Recipe #1 - Gather all following ingredients and follow the directions to prepare this tasty meal. Ingredients: banana Directions: chop the banana fry the banana prepare meal
Action a_{t-1}	take banana from counter
Feedback $O_{feedback}$	You take the banana from the counter. Your score has just gone up by one point.

Table 1: Example of each observation text o_t given to the agent at time step t

o_{recipe}. An example is given in Table 1. The agent's goal is to produce an action a_t which maximizes total discounted reward at every time step t.

In the scope of this work, generalization is broadly defined as the ability to perform well on unseen configurations of the same goal. This includes unseen configurations and layouts of the rooms and objects in the environment, but does not include unseen objects.

2 Related Single Game Models

Template DQN (TDQN) (Hausknecht et al., 2019) and Knowledge Graph Advantage Actor Critic (KG-A2C) (Ammanabrolu & Hausknecht, 2020) are agents trained independently across all Jericho-supported games. Both of these agents depend on the use of *templated action decoding*, where an action can be broken down into a template (*take OBJ from OBJ, eat OBJ*) and corresponding objects (*carrot, fridge*) from the agent's vocabulary. Given a state at each time step, TDQN uses three separate linear layers to produce one Q-Value per template and Q-Values for two possible objects. Actions are constructed by sampling templates and objects with the highest Q-Values and then filling in the blanks of the template.

KG-A2C extends this model by introducing the use of a knowledge graph encoder as part of the state definition. Knowledge graph tuples are extracted from OpenIE (Angeli et al., 2015) and constructed with additional hand crafted rules. An improved action decoding strategy is additionally introduced to sequentially produce an action. Given the state, the model uses a GRU decoder to produce probable templates. A single template is sampled and given to a GRU object decoder to produce a probability distribution over all words in the vocabulary. A single object is sampled and feed into to a second object decoder which finally samples a second object. Therefore, at each time step, three independent samples may be required to produce a single action. The knowledge graph is also used to prune the entire vocabulary of the agent, enabling the agent to only sample objects that are stored in the graph.

An important component of both these models is the requirement of a supervised signal in the loss function. While both use different reinforcement learning algorithms, they both utilize the *admissible commands* to build ground truth template and object targets. A binary cross entropy loss is added to the respective DQN and A2C loss functions. The intuition behind this supervised signal is to guide the agent in the right direction to sample templates and objects that are relevant to the current state. At the same time, the goal of the DQN/A2C loss is to adjust the probabilities to maximise discounted reward. As Hausknecht et al. (2019) mentions, this approach requires the supervised loss to be evenly blended with the reinforcement learning loss.

3 Multi Game Model for Generalization

This section describes an actor-critic model which directly ranks ground truth admissible commands, opposed to constructing supervised targets with the admissible commands. Figure 1 provides an

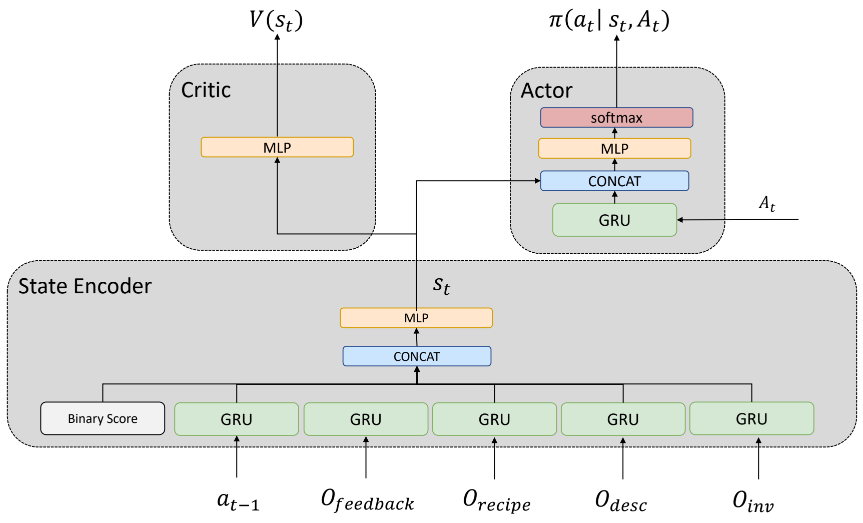

Figure 1: Overview of GT-A2C architecture for a given time step t. Each observation text, the previous action, and the score is encoded, concatenated, and passed through a 2 layer MLP to receive the state vector s_t. Each admissible action is encoded, concatenated with the state, and scored with a linear and softmax layer.

overview of the architecture, named Ground Truth A2C (GT-A2C). Concurrent work was done to develop a model to generate admissible commands at each time step with the intention of removing all ground truth targets and commands at reinforcement learning training and testing time. Section 3.2 describes this model in detail.

The hypothesis is that the removal of the supervised loss will lead to improved control polices as there is no need to find the hyperparameters to evenly mix the supervised and A2C losses. An addition hypothesis is that scoring entire actions opposed to sequentially scoring each template/object as in KG-A2C may also lead to better and more generalized control policies.

3.1 Control Policy

Following the previous works of KG-A2C, the agent is given an observation o_t and produces a state representation s_t at each timestep t. Each textual description is encoded with a corresponding GRU, and the last hidden states are concatenated and feed through a two layer MLP with a ReLU non linearity to produce the final state vector s_t. A single layer MLP is used to obtain the value function $V(s_t)$.

The list of admissible commands A_t and the state vector s_t are given to the action scorer to produce a distribution over all admissible actions $\pi(a_t|s_t, A_t)$ where $a_t \in A_t$. Each admissible action is encoded with a GRU and concatenated with the state vector. A linear and softmax layer is used to score each state-action pair. All GRUs contain separate parameters and word embeddings are initialized to random at the start of training.

Following the standard Advantage Actor Critic (Mnih et al., 2016), the control policy is updated over T transitions, where the policy update is given as

$$L_p = \log \pi(a_t|s_t, A_t) * Adv(s_t, a_t) \tag{1}$$

and the advantage function is defined using the temporal difference error

$$Ret = r_t + \gamma V(s_{t+1}) \tag{2}$$
$$Adv(s_t, a_t) = Ret - V(s_t) \tag{3}$$

where γ is the discount factor. The value function is learned using the mean squared error between the return and the predicted value

$$L_v = MSE(Ret, V(s_t)) \tag{4}$$

Level	#I	#R	S	Cut	Cook	Walkthrough
1	1	1	4	✓	✗	open fridge, take pork chop from fridge, take knife from counter, slice pork chop with knife
2	1	1	5	✓	✓	take banana from counter, cook banana with stove, take knife from counter, chop banana with knife
3	1	9	3	✗	✗	go west, open screen door, go west, go west, take red onion, go east, open screen door, go east, go east
4	3	6	11	✓	✓	go north, go east, take banana from counter, take red potato from counter, open fridge, take white onion from fridge, cook banana with stove, cook red potato with oven, cook white onion with stove, take knife from counter, dice banana with knife, dice red potato with knife, dice white onion with knife, drop knife,

Table 2: Statistics of games and walkthrough example of each game. #I is the number of ingredients in a recipe. #R is the **max** number of locations in a game. S is the max score an agent can achieve. Note each walkthrough also requires *prepare meal, eat meal* commands to complete the game. Adapted from Adhikari et al. (2020)

An additional entropy term is used to prevent early convergence

$$L_e = \pi(a_t|s_t, A_t) * \log \pi(a_t|s_t, A_t) \tag{5}$$

The final loss function is given as the sum over each transition in T as

$$L = \sum_{t=0}^{T} L_p + L_v - \alpha * L_e \tag{6}$$

where α is a hyperparameter to control the importance of entropy.

3.2 Command Generation

A model was developed to produce the set of admissible actions A_t given the textual description of a room o_{desc} and inventory o_{inv}. For simplicity, these will be called the context vector $c =< o_{desc}, o_{inv} >$. Since actions are view as the combination of templates and objects, the task is to generate the joint distribution

$$P(t, o_1, o_2|c) \tag{7}$$

from a known set of templates T and objects O, where $t \in T, o_1 \in O$, and $o_2 \in O$. The assumption is that there are only two possible objects to fill per template. This joint distribution can be rewritten as

$$P(t, o_1, o_2|c) = P(t|c) * P(o_1|t, c) * P(o_2|t, o_1, c) \tag{8}$$

To generate each distribution, a set of 190,000 tuples of (o_{desc}, o_{inv}, A_t) were collected from the FirstTextWorld problems dataset and used to fine tune a 12 layer BERT (Devlin et al., 2018) base-cased model. The context vector is encoded with BERT to receive e_c. Templates and objects are encoded with embeddings initialized at random at the beginning of training. First, the embedded context vector e_c is paired with each template embedding and passed through a 2 layer MLP to obtain $P(t|e_c)$. To produce $P(o_1|t, e_c)$, each template embedding is paired with each object embedding, concatenated with e_c and passed through another 2 layer MLP. Lastly, a combination of embeddings from all pairs of templates, o_1, and o_2 are concatenated with e_c and passed through a third MLP to produce $P(o_2|t, o_1, e_c)$. To generate the admissible actions list A_t, the three probability distribution are first thresholded. The probable templates are extracted from $P(t|e_c)$. For each probable template, the objects are extracted from either $P(o_1|t, e_c)$ or $P(o_2|t, o_1, e_c)$ depending on the number of objects required for the corresponding template.

The previous works have used the admissible commands to obtain non-conditional targets for templates and objects y_t, y_{o_1}, y_{o_2} for use in policy training. The same approach is followed to generate conditional targets $y_{t|c}, y_{o_1|t,c}, y_{o_2|t,o_1,c}$ and trained using the binary cross entropy during pre-training. $y_{t|c}$ is a one hot encoded vector over all possible templates, $y_{o_1|t,c}$ is a two dimensional one-hot encoded matrix for every template-object pair, and $y_{o_2|t,o_1,c}$ is a three dimensional one-hot encoded matrix for every template-object-object pair.

Algorithm 1 Training Strategy for GT-A2C - pseudocode for each training environment

```
1: Input: training games $\mathcal{X}$, evaluation games $\mathcal{Z}$, update frequency $F$, evaluation frequency $E$, averaging window $W$.
2: Initialize transition cache $C$, policy $\pi$, Episode Scores $R$, $score_{t-1} \leftarrow 0$
3: Sample a game $x \in \mathcal{X}$, $m_s \leftarrow$ max possible score for game, steps for current game $t_g \leftarrow 0$
4: Reset game to get state $s_t$
5: for $t \leftarrow 0$ to TOTAL_STEPS do
6:     Get admissible commands $A_t$
7:     Perform $a_t$ according to policy $\pi(a_t|s_t, A_t)$
8:     Receive $score_t$ and $s_{t+1}$
9:     $r_t \leftarrow score_t - score_{t-1}$
10:    push transition into $C$, $t \leftarrow t+1$, $t_g \leftarrow t_g + 1$
11:    if done then push score into R, reset game
12:    if average score in $R$ from previous $W$ episodes $= m_s$
       or $t_g$ = MAX_STEPS_PER_GAME then
13:        Sample new game $x \in \mathcal{X}$
14:        $m_s \leftarrow$ max possible score for game
15:        Reset game to get state $s_t$
16:        clear $R$, $score_{t-1} \leftarrow 0$, $t_g \leftarrow 0$
17:    end if
18:    if $t\%F = 0$ then Update($\pi$), reset $C$
19:    if $e\%E = 0$ then Evaluate each game $z \in \mathcal{Z}$
20: end for
```

4 Training and Dataset

A set of games introduced by Adhikari et al. (2020) is used for training and evaluating agents across several level of difficulties. Table 2 provides an overview for each level of difficulty. The agent is trained using the same set of 100 unique training games introduced in the DQN based agent in Adhikari et al. (2020). The use of an A2C agent in this work allows for multiple environments to run in parallel, making it possible to train on multiple games at the same time. The main training algorithm is outlined in Algorithm 1.

The agent trains on a batch amount of games at a single time, each collecting experiences independently, but updating and using the same network. A *convergence check* heuristic is implemented (Line 12 of Algorithm 1) to switch games during training and ensure the agent is not training on a single game for too many steps. Each game is allowed to run for a maximum number of t_g steps. If t_g amount of steps is reached, it is assumed the agent is stuck in a local optima and should sample a new game. In contrast, if the agent has already learned how to play a game, training steps should not be wasted by continuing to train on this one game. Therefore, an agent is assumed to have learned a single game if it can achieve max possible score m_s over the last window W episodes. The convergence check is performed at every time step for each training environment.

It is important to highlight the challenges between the four level of difficulties described in Table 2. Agents are designed to receive reward for taking actions that progress towards the end goal. Picking up ingredients relevant the recipe, processing ingredients (*chop, slice, dice*), cooking relevant ingredients, and prepare/eat meal commands always give the agent +1 reward. To introduce reward sparsity, taking a knife, opening a fridge/door, and navigation commands (*go north, go south*) do not give any reward but may still be required for solving a game. Additionally, processing an ingredient with the wrong command (*chop* when recipe calls for *slice*) and cooking with the wrong appliance (*cook with stove* when recipe calls for *cook with oven*) will end the game with no negative reward. Additionally, games are ended early if the correct cook command is called on an object that is already cooked, as the object will burn and disappear.

Games with single rooms in Level 1 and Level 2 are designed to require the agent to have more commonsense reasoning since ingredients must be processed and cooked correctly. Multi-room games in Level 3 require the agent to focus more on exploring the environment to find the kitchen. There is no processing of ingredients, meaning the episode will only end if the agent reaches the max amount of steps per episode or if the game is solved. Level 4 is designed to combine the commonsense reasoning, sparse reward, and navigation challenges together. Table 2 gives examples

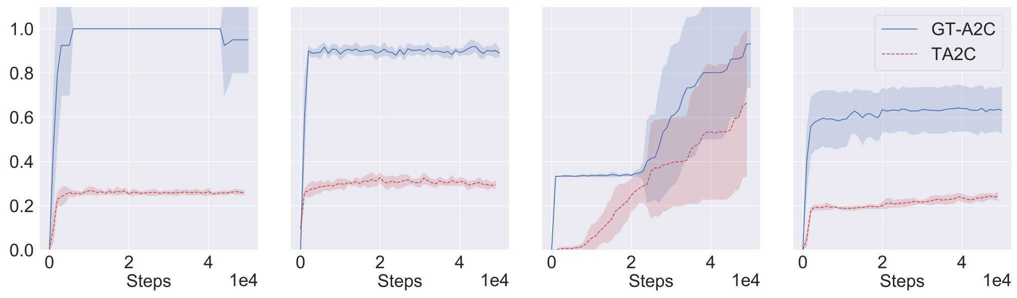

Figure 2: Average **single game** training curves over 10 seeds with standard deviation error bars. Level 1, 2, 3, 4 are from left to right. The y-axis is the normalized rewards (score/max score).

of the walkthroughs of each level of difficulty, but it is also important to note games in the dataset vary slightly within each difficulty (don't require opening the fridge, less number of rooms, etc.).

5 Results

As a first step, the GT-A2C model was used to isolate and evaluate the control policy without the use of the command generation model. This section discusses results from the GT-A2C model and Section 6.2 discusses the use of the command generation model.

5.1 Single Game Training

Since KG-A2C serves as state-of-the-art at the time of this project, the first step was to modify this model for use in TextWorld and cooking games. Since the OpenIE tuples would need to be extracted *per game* as they are in the original model, and concurrent work by Adhikari et al. (2020) involves learning to extract knowledge graph tuples from text observations in TextWorld, this work removes the knowledge graph component and evaluates the sequential action decoder. The same state encoder is used as in Figure 1. The original model also uses a graph mask to prune the agent's large vocabulary space. To allow for a more fair comparison, the action decoding of TA2C only scores over a set of known templates and objects opposed to the entire vocabulary. This version of the algorithm will be referenced as Template-A2C (TA2C).

Figure 2 provides training curves on a single game across all four difficulties for TA2C and GT-A2C models. The best hyperparameters were found after a gridsearch for both models. Each model is ran for a total of 50,000 steps and updated every 8 steps. Every 1000 steps, the average score over the last 6 episodes is recorded and shown in the figure.

TA2C is only able to achieve close to max score on Level 3. Observing the trajectories learned during training, TA2C utilizes a nuance in the TextWorld simulator. Commands such as *take carrot from fridge* and *take knife from counter* register in the simulator if only *take carrot* and *take knife* are given. TA2C favors these short length commands since each word of the action is being produced sequentially. TA2C struggles across all difficulties when it is required to generate two objects (*cook OBJ with OBJ* and *slice/chop/dice OBJ with OBJ*). Across all four difficulties, GT-A2C is able to outperform TA2C. This is largely due to the fact that GT-A2C only requires one sample per action, regardless of the length of the action.

5.2 Multi Game Training Results

For these reasons stated in the previous section, it was determined that a more simple control policy with only one sample would be a better model to use for multi-game training and evaluation. GT-A2C is trained for 100,000 steps with a batch size of 8 independent games at a time. Games are switched when the *convergence check* is reached. Figure 3 provides distributions of raw scores during training for each level of difficulty. Each time a new game is sampled, the average reward over a window size of 6 previous episodes is recorded.

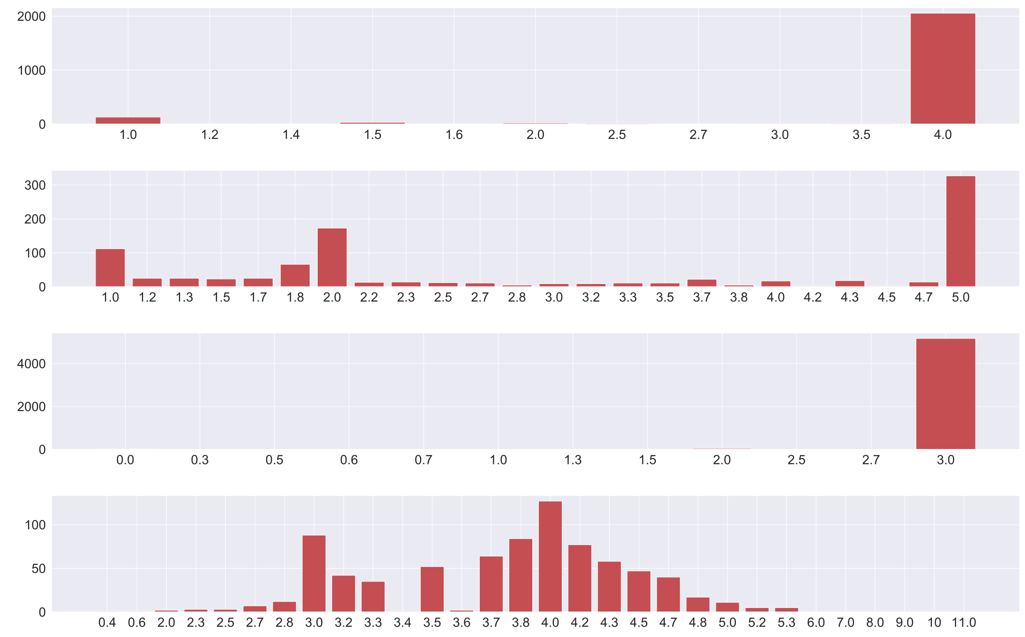

Figure 3: Training Statistics of a single trial of GT-A2C model. Each row represents Level 1/2/3/4, each with a max score of 4/5/3/11. Each time a new game is sampled, the average reward over the last 6 episodes plotted. The x axis represents the average reward and the y axis represents the frequency of the reward.

Level 1 and 3 are almost always able to achieve the max score of 4 and 3, respectively. As explained in Section 4, Level 1 requires the agent to pick up the knife (sparse reward) and how to properly use the knife (chop, cut, or dice), and is able to so on a regular basis. Level 3 always achieves max score, which is likely due to the fact that there is no way to end an episode early and the agents have been given enough steps to always navigate to the kitchen and take the correct item.

Level 2 is also able to regularly achieve the max score of 5, but there are also smaller peaks at a scores 1 and 2. The four major components of Level 2 are taking the ingredient (giving score 1), processing (giving score 2) and cooking the ingredient (giving score 3), and preparing and eating meal (giving score 5). Learning each new component of the level requires enough exploration of the action space. Processing or cooking the ingredient incorrectly will cause the episode to end, which explains the peaks at score 1 and 2. Once there is enough exploration, agents are able to learn how to properly process and cook ingredients and consistently achieve max score of 5.

There is enough exploration in Level 2 with a single ingredient, but not in Level 4 with three ingredients. In Level 4, agents must learn to take three ingredients (score 3), how to process (score 6) and cook each ingredient (score 9), then prepare and eat the meal (score 11). Figure 3 shows the majority of games ending with a score of 3 or 4 which indicates agents are able to regularly take all ingredients but have difficulties when correctly processing each ingredient.

5.3 Multi Game Evaluation Results

Agents are evaluated across the same 20 unseen games from Adhikari et al. (2020). Two methods of generalization are tested in this work. The first is **zero shot generalization**, where agents are given unseen games and not allowed to perform any updates. The second method is an **adaption generalization**, where agents are allowed to update parameters, but are given a small amount of steps to do so.

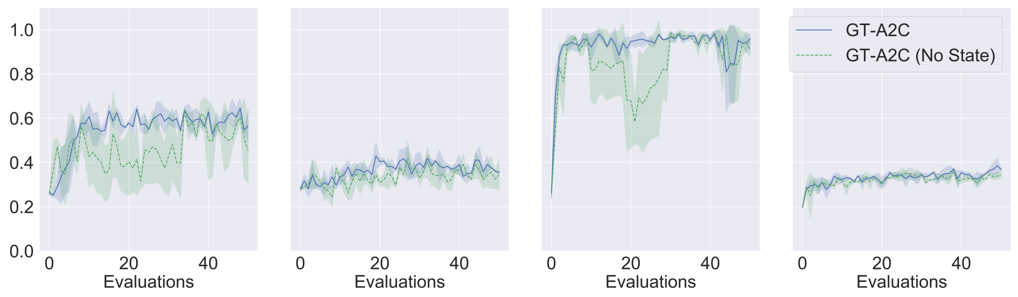

Figure 4: Average evaluation scores averaged across 20 evaluation games over 100,000 training steps. Curves are over 3 seeds with standard deviation error bars. Level 1, 2, 3, 4 are from left to right. The y-axis is the normalized rewards (score/max score).

5.3.1 Zero Shot Generalization

Figure 4 provides zero-shot evaluation results over the course of 100,000 steps. An evaluation is ran every 2000 steps across all 20 games. For each game during evaluation, the agent runs three episodes without any updates, and the final scores of each episode are averaged into a final score. Lastly, the final scores across all 20 games are averaged and reported in Figure 4.

Level 1 converges at around 60 percent evaluation score on average, even though Figure 3 showed almost all training games reached max score. Since there is no repercussion for taking and processing items that are not required for the recipe, during training the agent learns to continuously take and process items from the fridge and counter until it discovers which item is needed for the reward. The policy learned clearly does not generalize well to unseen games as there is no evidence of commonsense reasoning between the given state and scored actions. The recipe is directly given both at train and test time, which contains exactly which items that are needed to solve the game, it is clear this information is not being properly utilized by the agent.

The same explanation holds for Level 2 where commonsense reasoning is needed to understand which ingredients are in the current state which actions should be taken. The same issues are only magnified in Level 4, but the agent was not able to ever train to max score so it would not be expected that the agent would perform well on unseen games.

Level 3 is able to achieve close to 100 percent zero-shot evaluation score. While this game is characterized as the third hardest game by Adhikari et al. (2020), it is the easiest game for this agent to learn because of the lack of commonsense reasoning needed to solve the games. The agents are given enough steps to effectively explore the environment to and find the kitchen. Once in the kitchen, agents can take all items until it finds which item gives the reward.

5.3.2 Ablation Study with No State

Since it is apparent that agents are not utilizing the given recipe, an ablation is performed to determine which observations provide the most benefit for the agent. The state vector is completely removed during training and evaluated in the same zero-shot fashion. As Figure 4 shows, the agent is able to achieve similar evaluation scores, showing the state provides very little benefit. This provides insight to the lack of commonsense reasoning as agents are able to solve Levels 1, 2, and 3 during training by simply encoding given actions and ignoring information given as a state. Level 4 games are unable to achieve higher scores because it is simply too difficult for agents to perform three sequential processing and three sequential cook commands without commonsense reasoning.

5.3.3 Adaption Evaluation

Another evaluation method was designed to first measure if agents can adapt to unseen games when updates are allowed, and second to understand the importance of the number of games in the training set. For each unseen evaluation game, the agent is trained for 500 steps on 8 instances of the same

Level	1		2		3		4	
Training Size	20	100	20	100	20	100	20	100
Steps								
0	.264	.264	.382	.382	.495	.495	.301	.301
25k	.766	**.994**	.496	**.683**	.915	.915	.368	**.407**
50k	.697	**.962**	**.579**	.368	.865	**.903**	.357	**.393**
75k	.816	**.962**	**.606**	.456	.500	**.895**	.339	**.382**
100k	.737	**.998**	.659	**.775**	**.883**	.559	.357	**.397**

Table 3: Normalized adaption evaluation scores for a single trial. Episodes are ran on unseen games for 500 steps with parameter updates every 8 steps. For each game, the normalized score of the last 6 episodes are averaged. Scores are averaged over 20 unseen games. Evaluations are performed during training on 20 games and 100 games and done every 25,000 steps during training.

game. After the 500 training steps, the average normalized score over the last 6 episodes is recorded. The average across all 20 evaluation games is shown in Table 3. This evaluation is ran once at the beginning of training and then every 25,000 steps.

As expected, agents across all levels are able to achieve higher adaption scores as training progresses. This provides insight as agents can adapt within 500 steps, but further experimentation is needed to determine *how quickly* agents can adapt in terms of evaluation steps needed for convergence. For Level 1, 3, and 4 there is a clear increase in adaption scores when training on 20 games versus 100 games which justifies the use of a larger training set. A logical next step would be to determine how large this training size should be before performance stops increasing.

6 Discussion and Future Work

As Section 5.2 shows, with the exception of Level 4, agents are generally to achieve max score during training. Section 5.3 shows agents unable to achieve good performance on zero-shot evaluation. This is largely attributed to the policies learning during training, where agents learn to take and processes all items in the room until reward is found. With enough exploration in Levels 1, 2, and 3, agents are able to learn how to properly process and cook each ingredient.

At zero-shot test time, agents are expected to utilize information from the state. For example, if is a carrot on the counter, the stove is in the kitchen, and the recipe calls for a cooked carrot, the agent should be able to sequentially predict *take carrot from counter* and *cook carrot with stove*. As the ablation study with no state reveals, removing the state produces the nearly identical zero-shot results. There is a clear lack of connection between state and actions, so much so that a state is not needed to achieve max score on Levels 1, 2, and 3 during training. As many previous works do (Narasimhan et al., 2015; Hausknecht et al., 2019; Ammanabrolu & Hausknecht, 2020; Zahavy et al., 2018; Yuan et al., 2018), encoding each word in the observation to producing a single state vector s_t may not be an effective way to represent a state.

To provide insight to the previous statement, consider the fact detailed in Section 4 that a game will end early if the correct cook command is called twice on the same object. The agent gets +1 reward for the first cook command, but the second cook command will end the episode with no negative reward. If a *cook carrot with stove* command is called once during an episode, the agent must recognize that there is now a *fried carrot* in the inventory opposed to just a *carrot* and should not call this cook command again during the episode. As the encoded text is aggregated together to a single vector, this minimal difference in text may lead to information being lost. In future work, in order for good performance on zero-shot evaluation, it is not sufficient to only observe agents are training to max score. It is also needed to ensure agents are utilizing each component of the observation in the right way.

There have been two previous bodies of work that tested for generalization specifically in cooking games from the First TextWorld Problems dataset. The first is the competition winner [2] and the sec-

[2]https://www.microsoft.com/en-us/research/blog/first-textworld-problems-the-competition-using-text-based-games-to-advance-capabilities-of-ai-agents/

ond is a text version of Go-Explore (Madotto et al., 2020). The competition winner used BERT for entity extraction and template filling, with a simple Upper Confidence Bound (Auer, 2002) control policy. Go-Explore uses a unique exploration strategy to collect high reward trajectories, and uses this as training data for a sequence model that predicts the next action given a trajectory.

Neither of these works use a 'standard' reinforcement learning framework, and both perform well on zero-shot generalization on cooking games. Reinforcement learning algorithms are designed to learn through trial and error. As explained in this work, an episode ends early when items are not processed or cooked correctly as the item disappears from the game and the recipe is unable to be completed. While ending the game is the only logical thing to do, it inhibits the agents from collecting enough experiences of what should and should not be done, which results in agents easily getting stuck in a local optima. Adding a negative reward has been explored when the game ends early, but this only prevents agents from ever wanting to use a knife since it often leads to negative rewards. Increasing the entropy coefficient has helped in training for Level 3, but the requirement for commonsense reasoning and inability to explore unseen states is an issue that entropy was not able to solve in Level 4.

The issues with lost information in the state, exploring with the use of the reinforcement learning framework, along with the fact that the previous two works do not use reinforcement learning, all suggest a fundamental change is needed in the approach of this work to achieve better training and evaluation performance.

6.1 Future Work

There is a need for a stronger connection between states and actions, as well as a need for agents that don't require as much exploration to learn control policies. In future work, an attention based model may produce better connections between states and actions at the word level. Each admissible action could be concatenated with the state, and a self-attention/transformer architecture can be used to encode the text and produce state-action pairs. With this approach, an action representation can be thought of as a weighted sum of all words in the state and action, inhibiting the agent from ignoring all state text. To take this one step further, an ensemble of transformer models could be used for the description, recipe, feedback, and previous action text to ensure each component of the observation is being used effectively. For a control policy, the competition winner has proven a simple policy generalizes well, so UCB will also be explored with the attention based model.

6.2 Full Model

Section 3 introduces a command generation model that was developed concurrently with the control policy. The idea was to simply replace Line 6 of Algorithm 1 to generate admissible actions from this model opposed to accessing this information from the simulator. This model has been trained to 0.99 F score on the train/test set from the First TextWorld problems dataset, and has also been used in zero-shot and adaption evaluation for Level 1 and 2, but it was decided not to pursue this model further until the issues in the previous section are addressed.

7 Conclusion

The goal of this project was to improve generalization in reinforcement learning agents for text-based games. Previous works have focused on training on a single game at a time, and it was shown that these models to not directly translate to a different environment. There has been minimal previous works in this domain that focus on generalization with a reinforcement learning control policy. A new model, motivated by the previous works of single game models, was proposed to do an effective evaluation of generalization across various levels of difficulties. It was shown that this model still lacks commonsense reasoning and is unable to perform well in a zero-shot evaluation setting. A key finding is that models developed to play single games cannot generalize by simply training on more instances of similar games. An attention based model is discussed as future work with justification as to why this model may lead to better training and generalization results.

8 Additional Comments

This section highlights additional work conducted during the MS project that did not directly fit into the narrative of this report. The purpose is to provide additional evidence that enough work has been conducted throughout the project to grant 6 graduate credits.

Knowledge Graph Tuple Classifier The fine tuned BERT command generation model was originally a model to extract knowledge graph tuples from the given textual description and inventory as an observation O. The idea was to replace the OpenIE and hand crafted KG extraction from the KG-A2C model. The data was collected from the First TextWorld Problems dataset in the same way, but ground truth knowledge graph tuples were extracted from the simulator opposed to admissible commands. The probability of a KG tuple (e_1, r, e_2) was defined as $P(E_1 = e_1, E_2 = e_2, R = r|O = o)$, where E_1 is the first entity, E_2 is the second entity, R is the relation between two entities, and O is a BERT-encoded text observation. The model could be broken into three probabilities $P_1 = P(E_1 = e_1|O = o)$, $P_2 = P(E_2 = e_2|O = o)$, and $P_3 = P(r|E_1 = e_1, E_2 = e_2, O = o)$. P_1 and P_2 are a binary classification task where the output of each is a binary vector indicating which entities are present from a set of known entities. P_3 is a multi label classification task where the output is the probability of each relation from a known set of relations for every pair of entities. Once it was determined that the sequential action decoding model may not be the best model for testing evaluation, this model was adapted to the command generation model.

Backplay A Backplay (Resnick et al., 2018) training strategy was investigated for the single game TA2C, single game GT-A2C , and multi game GT-A2C . This strategy utilizes the walkthrough of each game, where agents begin closest to the goal state, train for a set amount of steps until convergence, then move one step backwards. This is repeated until the agent has trained starting from each point in the walkthrough. This was found to help with exploration as agents are forced to train on all parts of the environment, but was found to not help with zero-shot generalization. In addition, the use of the walkthrough would have required comparisons to meta-learning algorithms as baselines, and the results were not significant enough to pursue this training strategy further.

State Encodings When first analyzing the zero-shot evaluation results from Section 5.3, it was clear the agents were not using the recipe which is given every step. The recipe directly provides text on how to complete each game, which should be all used by agents to easily solve most games. A more direct one-hot encoding of the recipe was instead given to the agent when producing the state vector s_t, as well as given directly to the Actor in Figure 1 during action scoring, but there was no significant change in zero-shot scores. The pretrained BERT model for generating commands was also used for encoding the observations and actions, replacing all GRUs. Graph encodes such as GAT (Veličković et al., 2017) and R-GCNs (Schlichtkrull et al., 2018) were explored to encode the ground truth knowledge graph representation of each state, all still without a significant increase in performance. Once it was determined that removing the state completely still gives similar results, it was evident that only changing state encoding was not going to solve the issues faced.

Acknowledgments

I would like to give my gratitude towards Dr. Marc-Alexandre Côté of Microsoft Research, Montréal and Trapit Bansal of UMass Amherst for their consistent guidance throughout the project. You provided lessons that I will utilize for the rest of my career.

Additionally, thank you for Dr. Marc-Alexandre Côté and Dr. Andrew McCallum for being official readers for my MS Project.

References

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L Hamilton. Learning dynamic knowledge graphs to generalize on text-based games. *arXiv preprint arXiv:2002.09127*, 2020.

Prithviraj Ammanabrolu and Matthew Hausknecht. Graph constrained reinforcement learning for natural language action spaces. *arXiv preprint arXiv:2001.08837*, 2020.

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 344–354, 2015.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Workshop on Computer Games*, pp. 41–75. Springer, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Matthew Hausknecht, Prithviraj Ammanabrolu, Côté Marc-Alexandre, and Yuan Xingdi. Interactive fiction games: A colossal adventure. *CoRR*, abs/1909.05398, 2019. URL `http://arxiv.org/abs/1909.05398`.

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. *arXiv preprint arXiv:2001.08868*, 2020.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

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

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*, pp. 593–607. Springer, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

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.

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*, pp. 3562–3573, 2018.