Github: https://github.com/eminalparslan/textworld-llm/tree/main
Youtube video: https://youtu.be/CjRIHDYZW_Q

Fine-Tuning Language Models for Text-Based Games

Emin Alp Arslan
eminalp@utexas.edu
College of Natural Sciences
University of Texas

Abstract

The introduction of reinforcement learning environments like TextWorld have increased interest in environments with natural language state and action spaces. Text-based games (TBGs) in particular present unique challenges for reinforcement learning agents. Recently, the advent of large language models (LLMs) have enhanced the capacity for machine comprehension of language. This paper explores the application of LLMs as reinforcement learning policies in such text-based games through the use of policy gradient fine-tuning techniques. I find that fine-tuned LLMs enable an agent not only to adapt to its observations and a proper action space, but also result in a more generalizable policy compared to conventional approaches. I also discuss the tradeoffs of this approach.

1 Introduction

TextWorld (Côté et al., 2018) is a program for generating text-based games with a focus on training reinforcement learning agents. These text-based games (TBGs) are sequential games where the game's state and player's actions are conveyed through text in an interactive loop. See Table 1 for an example of a TextWorld game. TextWorld has the ability to generate games of varying difficulty across different domains and is an environment of interest to many researchers because of its unique challenges. First, the environment is text-based, meaning the agent must have some baseline understanding of language in order to both interpret the observations from the environment, and to perform actions based on those observations. Second, the environment is partially observable; the agent has only the current—or last few—observations at each decision point, making it difficult for the agent to know its own progress and stay focused on the goal. These two challenges make TextWorld an interesting environment for contemporary RL research, especially in the era of large language models (LLMs) that have advanced the machine comprehension of language.

TextWorld environments are sometimes formulated as Markov decision processes (MDPs) due to their sequential nature. However, more often they are formulated as partially observable Markov decision processes (POMDPs) since the agent would have to keep track of all past observations to act optimally (Côté et al., 2018). The environment state consists of information often hidden from the agent like the position of various entities, the inventory of the agent, and subgoals. The observations of the agent are often much more constrained, typically responding to the agent's last action. The actions are text commands, and the agent often doesn't know whether the commands are valid. The rewards come from the scoring mechanism of the environment, where there are typically subgoals that earn the agent points as it gets closer to the goal.

Role	Observation/Action
Game	You are hungry! Let's cook a delicious meal. Check the cookbook in the kitchen for the recipe. Once done, enjoy your meal!
Agent	> **read cookbook**
Game	You open the copy of "Cooking: A Modern Approach (3rd Ed.)" and start reading: Ingredients: red hot pepper Directions: slice the red hot pepper, fry the red hot pepper, prepare meal
Agent	> **take red hot pepper**
Game	You take the red hot pepper from the counter. Your score has just gone up by one point.
Agent	> **cook the red hot pepper**
Game	What do you want to cook the red hot pepper with?
Agent	> **cook the red hot pepper with the oven**
Game	You burned the red hot pepper! *** You lost! ***

Table 1: Example of a short TextWorld game (edited for brevity).

2 Motivation

Large language models, particularly foundation models, provide a great opportunity for reinforcement learning environments with linguistic state/action spaces. They can imbue agents with a base comprehension of language to solve such environments. Fine-tuning these models has become more prevalent, especially with the advent of more efficient training techniques like QLoRA (Dettmers et al., 2023). Perhaps the most prevalent application of RL for these large language models is reinforcement learning from human feedback (RLHF), introduced by Ouyang et al. (2022) from OpenAI to align their GPT models to human preferences. In their paper, they use PPO (Schulman et al., 2017), a policy optimization algorithm, to directly tune the weights of the model, acting as their policy, to maximize its reward signal. The aim of this paper is to extend this framework of RL fine-tuning to sequential text-based environments like TextWorld.

3 Related Works

One of the primary motivations for TextWorld was to explore environments where the state and the action space is language and the partially observable nature of these games (Côté et al., 2018). The paper introduced the environment in the reinforcement learning setting and noted that language understanding was one of its primary challenges. The state-of-the-art at the time was a two-staged model called LSTM-DQN (Narasimhan et al., 2015). A recurrent LSTM module (called the Representation Generator ϕ_R) was used to encode the observations from the environment and traditional DQN networks (collectively called the Action Scorer ϕ_A) were used to predict the Q-values for verbs and objects separately. These are combined to approximate the Q-values: $Q(s, a) \approx \phi_A(\phi_R(s))[a]$. The other state-of-the-art model was the Deep Reinforcement Relevance Network (DRRN) (He et al., 2016). This model represents the action and state spaces with separate embedding vectors which are compared with a relevance function (like an inner product). The output of this function is the Q-value. Both of these architectures were used in other text-based environments with very promising results, but face many challenges such as the combinatorial nature of having language as the state and action space and how they lack the ability to consider previous actions and observations to deal with partial observability.

To overcome the challenge of partial observability, Yuan et al. (2018) use an LSTM-DRQN (not to be confused with LSTM-DQN mentioned previously). The LSTM-DQN has the limitation that it cannot memorize previous actions or observations since the LSTM module produces a vector representation of the observation, which is used in the DQN without recurrence. By adding a

recurrent layer to the DQN, it gains a memory of previous observations, overcoming the obstacle of partial observability (Hausknecht & Stone, 2017).

Zhao et al. (2021) look at meta-reinforcement learning techniques where the agent first explores the environment to gain information about the task before exploiting that knowledge and solving the task. The goal of this approach is to allow for generalization across other text-based environments. The learning-to-explore approach they use splits an episode into exploration and execution stages. This approach doesn't face the sample inefficiency problem other meta-reinforcement learning approaches face. LeDeepChef, on the other hand, is another agent with a design goal of generalizing to text-based games of the same theme (Adolphs & Hofmann, 2020). It was second in Microsoft Research's *First TextWorld Problems: A Language and Reinforcement Learning Challenge* which had a cooking theme with highly variable environment parameters. It used an actor-critic model and solved the combinatorial action space by combining multiple actions to a single high-level command. Ammanabrolu & Riedl (2019) took advantage of recent advances in graph embedding and attention to have their KG-DQN (Knowledge Graph DQN) architecture learn which portion of the graph to pay attention to with natural language actions, effectively pruning large swathes of the action space. The graph contains information about the relationship between objects and allows the agent to have knowledge about the world and the effect of actions.

A common theme among all of these approaches is an attempt to cull the action space into a reasonable size. Tao et al. (2018) acknowledge this and train a hierarchical recurrent network with an encoder-decoder architecture that generates a set of valid commands from the current state. This could be used in conjunction with the LSTM-DQN or DRRN frameworks, or any other architecture that has issues with large action spaces.

The advent of more sophisticated language models provides opportunities for new methods for dealing with textual state and actions spaces without relearning linguistic priors. CALM (Contextual Action Language Model) generates action candidates for each state and is trained on human gameplay and then fine-tuned with the DRRN architecture (Yao et al., 2020). Yin et al. (2020) also incorporates the general commonsense knowledge these language models contain by having a teacher DQN teach a more powerful language model-based DQN (in this case 12 layers of BERT) student model. This approach essentially enhances all of the RL-based methods we have mentioned so far.

The approach in this paper differs from these in that it fine-tunes the language model directly as a policy, getting the benefit of its commonsense knowledge and linguistic comprehension.

4 Methodology

My work builds off of the framework starter code for Microsoft's 2019 *First TextWorld Problems* challenge. This code also includes an implementation of the LSTM-DQN approach described previously and is one of the baselines I will compare my approach with. The other baseline I use is the base LLM without any fine-tuning. This allows me to see the effect of fine-tuning on performance.

The scoring of each agent is done with a script that sums the total points the agent received across all its games in the test set. These points are returned from the TextWorld environment every time the agent reaches a subgoal. The score is then scaled according to what handicaps the agent had in its observations. For example, if the agent had the ability to check its inventory, its score would receive a small penalty. If the agent had access to the list of admissible commands, then it would receive a very large penalty. All of my experiments had the same handicap that the agent couldn't access the list of admissible commands.

The baseline LLM agent used the Llama2-7B model (Touvron et al., 2023) directly without fine-tuning with the following system prompt: `"You are playing a text-based game with a cooking theme. You will receive observations about the current state of the game and respond with commands. Here are some example commands: 'examine counter', 'inventory', 'go north', 'pick up the knife', 'fry the apple on the stove', 'open`

door', 'look', and 'goal' to remind yourself of the goal. These commands might not work and there are many commands not listed here. When responding, first reason about the game state to decide the best action to reach the goal and then say 'command: <your command>'. Only respond with the command and don't say anything else, even when you are told your commands aren't recognized." Optimizing the system prompt was not a focus of this paper, as it is not relevant to the goal of evaluating the effect of fine-tuning in this environment.

The agents that needed training (the LSTM-DQN and the fine-tuned Llama2-7B model), used the sample games that came with the starter code as the training set. I used the default hyperparameters for the LSTM-DQN agent and trained for approximately 2 hours on an Nvidia RTX 3090.

The fine-tuning of the LLM was done with HuggingFace's TRL library in conjunction with their PEFT library for efficient fine-tuning with QLoRA (Dettmers et al., 2023) at 4-bit precision. I left most of the hyperparameters such as the learning rate and the Lora configuration to their default values. Fine-tuning these models takes a lot of time, so it is difficult to experiment with different hyperparameters in a short time. The model was fine-tuned for 4-5 hours on 2 Nvidia RTX 3090s.

The agent used the last five observations and actions along with the same system prompt above to choose the next command at each decision step. This number was chosen as it maximized the context length of the model. If the model didn't output a well-formed command (or any command at all), the default action of `wait` was used, as it was unlikely to change the state of the game.

The true reward function is the points the agent earned at each step with its action. During training, I shaped the reward function to apply a large penalty (-5 points) when the game was lost and a small penalty (-0.5 points) when the default action `wait` was chosen. There wasn't a methodology behind the values for the penalties; it was based more on intuition.

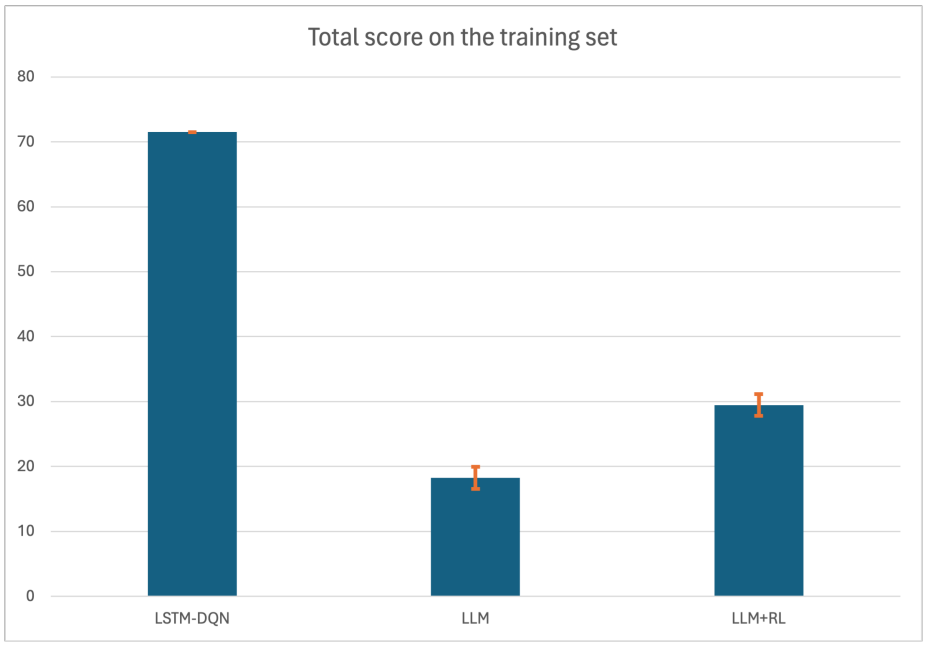
5 Results

I evaluated each agent on the training set and a test set. The training set contained the 10 games that were used to train the agent while the test set included another 70 environments (from the competition's test set) it hadn't encountered before. This was done to evaluate how well the agent generalized to similar, but different environments. All the environments followed the same theme of 'cooking', but had different goals and subgoals. Figure 1 shows the total scores for each agent for the training and test sets respectively.

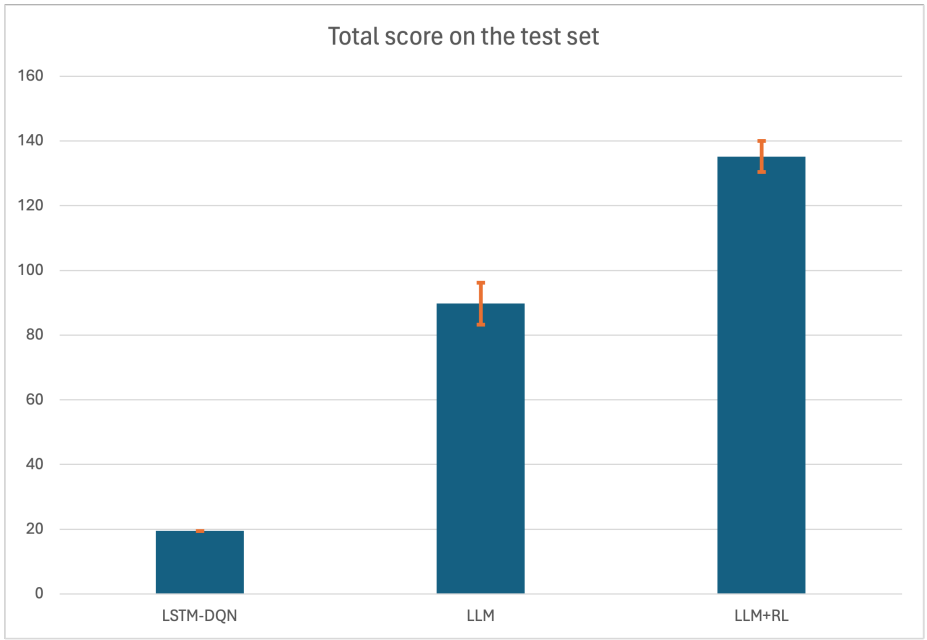
6 Conclusions

The LSTM-DQN agent outperforms both the baseline LLM agent and the fine-tuned agent when scored on the games it was trained on, and by a large margin. This seems to show that the conventional reinforcement learning approaches learn the environment and the linguistic capabilities to solve the environment more effectively than LLMs. We can think about the LSTM-DQN agent as learning only what is needed, being forced to learn only enough language to accomplish its task. On the other hand, the LLM already has a general knowledge of language, but needs to narrow its focus to a specific application—in this instance, generating commands in a text-based game. One can imagine these two approaches as opposites, one bottom-up and the other top-down.

We can see the tradeoff in these approaches when we look at the results of the test set. The LSTM-DQN agent fails to transfer its knowledge to the new environments (games), but the LLM, and more importantly the fine-tuned LLM, are a different story. Even the baseline LLM vastly outperforms the LSTM-DQN model on the test set. The advantage of the fine-tuned model over the baseline LLM is carried over to the test set as well, showing the efficacy of fine-tuning a policy LLM over the conventional LSTM-DQN network, especially when generalization to similar environments is desirable.



(a) Results for the training set.



(b) Results for the test set.

Figure 1: Total scores for each agent on the training and test sets, averaged over 10 runs. The scores were adjusted based off the information they had available during training, which included everything except for the set of admissible commands. The orange bars show the standard deviation over the 10 runs. The score scales are different between graphs as there are a different number of games in training and test sets.

There are other disadvantages of fine-tuning LLMs for such environments, however. Namely, the difficulty and inefficiency of the training process. Both the training and the actual inference-time speed of the model is much slower than the LSTM-DQN. The training is also fairly unstable, with performance varying wildly over the course of its training. It is possible there is a configuration of hyperparameters that makes the training process more stable, but I did not have enough time to verify this hypothesis.

References

Leonard Adolphs and Thomas Hofmann. Ledeepchef deep reinforcement learning agent for families of text-based games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7342–7349, 2020.

Prithviraj Ammanabrolu and Mark O. Riedl. Playing text-adventure games with graph-based deep reinforcement learning, 2019.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. In *Computer Games Workshop at ICML/IJCAI 2018*, pp. 1–29, June 2018. URL https://www.microsoft.com/en-us/research/publication/textworld-a-learning-environment-for-text-based-games/.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space, 2016.

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

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Ruo Yu Tao, Marc-Alexandre Côté, Xingdi Yuan, and Layla El Asri. Towards solving text-based games by producing adaptive action spaces. *arXiv preprint arXiv:1812.00855*, 2018.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep calm and explore: Language models for action generation in text-based games. *arXiv preprint arXiv:2010.02903*, 2020.

Xusen Yin, Ralph Weischedel, and Jonathan May. Learning to generalize for sequential decision making, 2020.

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.

Z Zhao, M Sun, and X Ma. Meta-reinforcement learning for mastering multiple skills and generalizing across environments in text-based games. Association for Computational Linguistics, 2021.