Text-Room Adventure: Incorporating Pre-trained Language Models and Memory Units to Deep Q-Learning Framework

Fei Wang fwang598@usc.edu
Zhongheng He hezhongh@usc.edu
Kainuo Feng kainuofe@usc.edu
Yongcheng Wang ycwang039@usc.edu

Abstract

In interactive fiction, players interact with unknown game worlds described in natural language. Most of the interactions rely on commonsense knowledge. Previous works have tried to use knowledge graphs to represent commonsense knowledge explicitly. However, none of existing works utilized rich commonsense knowledge outside the game world. In this paper, we incorporate pre-trained language models into the deep Q-learning framework to bring in rich commonsense knowledge existing in external text corpus, and combine history actions into decision making process. Further, we build a 2D simulator that can visualize the game process to implement case study and assist detailed performance analysis. Experimental results on the Jericho[1] platform show that our method performs better than several strong baseline methods.

1 Introduction

Interactive fiction (IF) is a software simulating environment where players use text commands to control characters and influence the environment (Ziegfeld, 1989) with a history of more than 60 years. In the common game setting of IF, the player starts the game at an unknown game world described in natural language and needs to interact with all possible objects around to explore the world and finally win the game.

IF games rely on the player's commonsense knowledge as a prior for how to interact with the game world (Hausknecht et al., 2020). For example, when encountering locked doors, human players intuitively understand that they need some keys. However, such commonsense knowledge is hard to learn from scratch directly through interaction with the game environments.

[1]https://github.com/microsoft/jericho

Previous works have tried to use knowledge graphs (KG) to represent commonsense knowledge explicitly (Ammanabrolu and Riedl, 2019a,b; Ammanabrolu and Hausknecht, 2019). These works constructed KG through interaction with the game environments. Although Ammanabrolu and Riedl (2019b) tried to transfer the commonsense knowledge learned from different games, none of existing works utilized rich commonsense knowledge outside the game world.

In this paper, we incorporate pre-trained language models, such as BERT (Devlin et al., 2019), into the deep Q-learning framework (Mnih et al., 2013) to train AI agents for IF. Previous work shows that BERT contain rich commonsense knowledge from pre-training (Cui et al., 2020). Besides, we also combine history actions into the Q-learning computation since prior knowledge can be helpful in the continuous decision process, just like humans. Experimental results on the Jericho platform (Hausknecht et al., 2020) show that our method performs better than several strong baseline methods. To further verify the effectiveness of our method, we build a 2D simulator to visualize the game environments and actions issued by models. Detailed analysis shows that our predictions highly consist with optimal actions and are close to human performance.

In summary, our contribution is two-fold. First, we include both external commonsense knowledge and internal prior knowledge to the AI agents, by incorporating pre-trained language models and encoding history actions. Second, we build a 2D simulator that can visualize the game process to assist case study and detailed performance analysis.

2 Overview of the Text-Adventure Game

In this section, we briefly describe the definition and history of interactive fiction (Section 2.1), platforms for game development and research (Section 2.2), and the game scenario we use for experiments (Section 2.3).

2.1 Game History

The origin of interactive fiction can be traced to the 1960s and 1970s, when the simple natural language processing could be applied onto software programs. In the 1970s, Will Crowther, a programmer and an amateur caver, wrote the first text adventure game, ADVENT[2], which was further spread to the Internet and inspired many people to design and write their own text adventure games, like the Dog Star Adventure[3], the ZORK series[4], etc.

In the 1980s, IF became a standard product for many software companies, and the interactive fiction occurred outside the U.S. But since the 1990s, the market of IF has been declining due to the growth of multi-media games.

2.2 Game Environments and platforms

From an interactive fiction to an interactive game, we need a platform to support the game environment and control the workflow. There are plenty of open source platforms able to create traditional parser-driven IF in which the user types text-based commands — for example, go east, go downstairs, read the newspaper — to interact with the game. From the 1990s to present, we have many successful IF platforms. And in the past decade, the two most popular platforms might be the Jericho(Hausknecht et al., 2020) and TextWorld(Côté et al., 2018) from Microsoft. And Facebook, Inc. also launched LIGHT(Jack Urbanek, 2019) as its text interactive platform for dialogue research.

In this project, we select Jericho as our platform for the text-adventure games. It is a lightweight python-based interface connecting learning agents with interactive fiction games. It runs on Linux-like systems and is easy to install or serve.

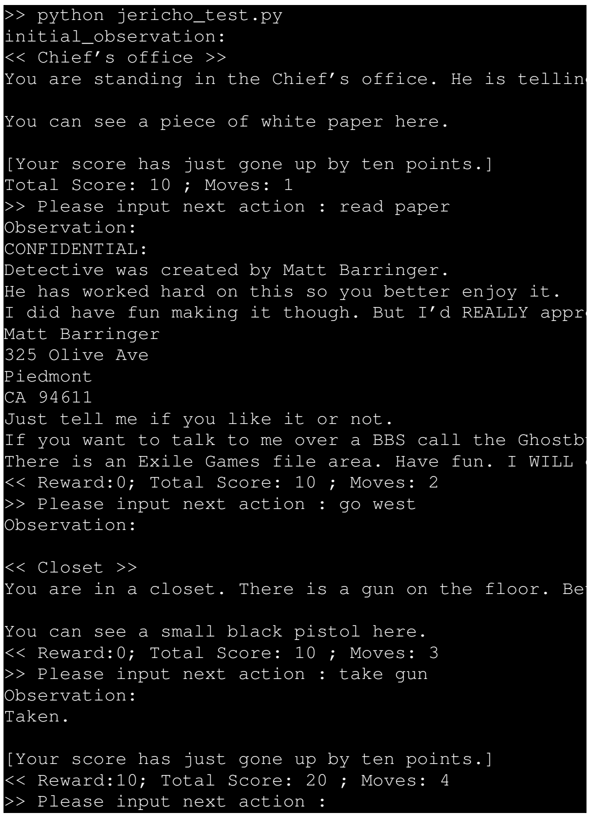
2.3 Game Scenario

We choose a single player text-adventure game whose name is "Detective" as an example to show you he basic scenario of a text-adventure game: the player character is a famous detective. At the beginning of the game, the player is standing in the police chief's office and is asked to investigate the mayor's murder case. The player should search the neighbor area for clues and evidence. Some events and clues are embedded in room descriptions. The Listing 1 is a demo of how we play the Detective game.

The entire game is based on text and we only need a terminal with python environment to run it. Look at the Figure 1, the game process is formed by three elements: observation, action and reward. Each turn, the player gets the observation, and then takes action according to this information. Then the game engine will tell the player how many scores he gets by taking this action. The Listing 1 is an example of our game. Through the observation, the player finds a gun on the floor. And he just typed "take gun" in the terminal as an action. Then he got 10 scores as a reward because this is a clue for the murder case. For some useless action, you can also see that the player gets zero reward.

The ultimate goal of the player is to get as much reward as possible in a limited number of actions.

Listing 1: The detective game demo.



```
>> python jericho_test.py
initial_observation:
<< Chief's office >>
You are standing in the Chief's office. He is tellin

You can see a piece of white paper here.

[Your score has just gone up by ten points.]
Total Score: 10 ; Moves: 1
>> Please input next action : read paper
Observation:
CONFIDENTIAL:
Detective was created by Matt Barringer.
He has worked hard on this so you better enjoy it.
I did have fun making it though. But I'd REALLY appr
Matt Barringer
325 Olive Ave
Piedmont
CA 94611
Just tell me if you like it or not.
If you want to talk to me over a BBS call the Ghostb
There is an Exile Games file area. Have fun. I WILL
<< Reward:0; Total Score: 10 ; Moves: 2
>> Please input next action : go west
Observation:

<< Closet >>
You are in a closet. There is a gun on the floor. Be

You can see a small black pistol here.
<< Reward:0; Total Score: 10 ; Moves: 3
>> Please input next action : take gun
Observation:
Taken.

[Your score has just gone up by ten points.]
<< Reward:10; Total Score: 20 ; Moves: 4
>> Please input next action :
```

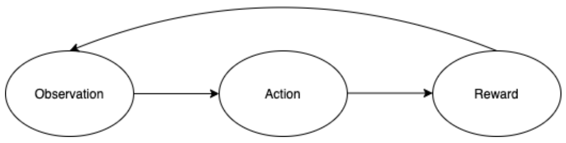


Figure 1: Agent work process.

[2]http://hdl.handle.net/2142/16406
[3]https://www.mobygames.com/game/dog-star-adventure
[4]https://adventuregamers.com/gameseries/view/1582

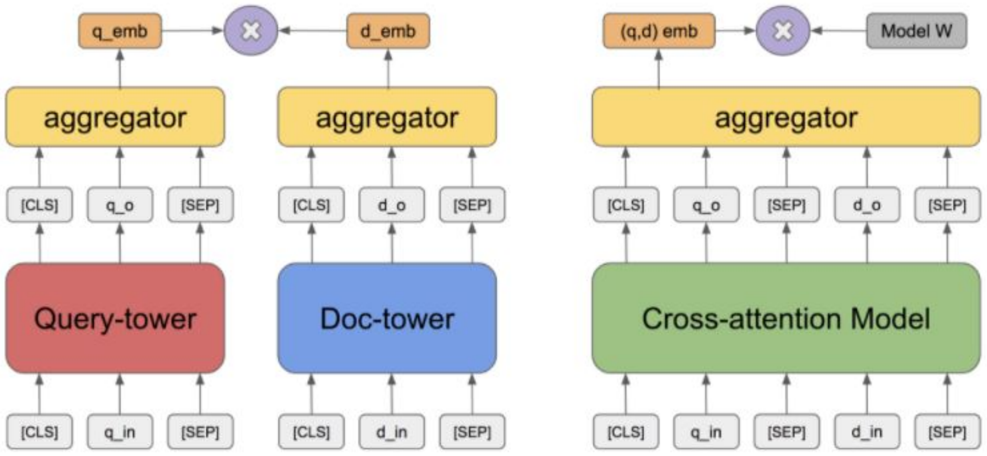


Figure 2: Two-tower model (left) and cross-attention model (right). Figure is copied from Chang et al. (2019).

3 Related Work

Our work is connected to two research topics. Each has a large body of work which we can only provide as a highly selected summary.

3.1 Learning Agents for IF

IF Game Agent has been well explored and applied on IF game agents, focusing on text understanding and interaction between state information and action. Narasimhan et al. proposed LSTM-DQN (Narasimhan et al., 2015), which utilized LSTM as a representation generator to jointly learn state representations and action policies using game rewards as feedback. Hausknecht et al. extend LSTM-DQN as Template-DQN (Hausknecht et al., 2020) for template-based action generation, by using the output of LSTM-DQN to predict the placeholds in the template. Deep Reinforcement Relevance Network (DRRN) (He et al., 2015) was introduced for choice-based action selection. It represented action and state spaces with separate embedding, which are then combined with an interaction function to approximate the Q-function in reinforcement learning.

More recently, novel combinations of text game and other tasks (i.e., machine reading comprehension) has been proposed, which further improves the performance of the agents. Guo, Xiaoxiao, et al. used a reading comprehension model with Bidirectional Attention Flow (Seo et al., 2016) to encode state and action information, by treating the observation as a context and action as a query like a question and answering task. Moreover, they retrieve past observations in previous steps to determine long-term effects of action, aiming to tackle the partial observability of text games (Guo et al., 2020b). Yao, Shunyu, et al. used a pre-train language model (i.e., GPT-2) to generate candidate actions, showing the potential of direct action-generation (Yao et al., 2020). Ammanabrolu et al. proposed a graph-based deep reinforcement learning (Ammanabrolu and Riedl, 2018). They represented the game state as a knowledge graph and updated it during the exploration. And when the agent needs to select the actions, we turn the information of the knowledge graph to a single vector and use it in a neural network. This work was further extended to KG-A2C (Ammanabrolu and Hausknecht, 2020)) by exploring and generating actions using a template-based action space and utilizing Advantage Actor Critic training method.

3.2 Pre-trained Language Models

Recently, language models pre-trained on a large scale of corpus (Devlin et al., 2019; Peters et al., 2018; Yang et al., 2020; Zhang et al., 2019) demonstrate surprising power in encoding context and semantic information and bring natural language processing to a new era, among which BERT obtained the state-of-the-art results on eleven popular natural language processing tasks when it was proposed in 2018. Nowadays, the BERT has become the backbone and foundation of many other works related to natural language processing (Liu et al., 2019; Lan et al., 2020; Joshi et al., 2020).

In detail, BERT is a transformer-based (Vaswani et al., 2017) architecture pre trained with two tasks: Masked Language Model(MLM) and Next Sentence Prediction(NSP). In MLM, the model needs to predict the masked tokens in the original sentence, and in this way, it learns to capture the bidirectional contextual information. In NSP, the model needs to predict whether the given two sentences are consecutive, aiming to capture the relationship between different sentences. After the pre-training, BERT can be fine-tuned in many downstream tasks, i.e., Question Answering.

In our work, we use BERT to encode the textual description and explore the relationship between states and actions.

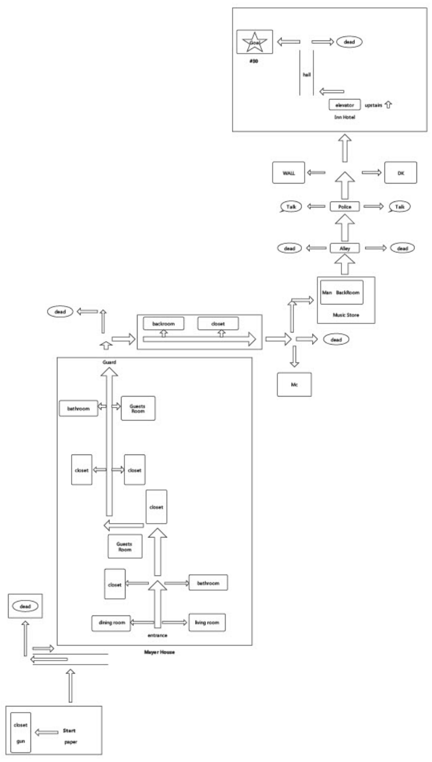


Figure 3: Abstract 2D map of the game

4 Method

In this section, we first introduce the deep reinforcement learning framework for text adventure games, and then describe two types of state-action matching models.

4.1 Overview of the Learning Framework

Following previous works (He et al., 2016; Hausknecht et al., 2020), we solve the sequential decision making problem for text adventure games with deep Q-learning (Mnih et al., 2013).

The game starts with description of the game background s_0. At each time step t, the agent issues a textual action $a_t \in A_t$ based on the environment description s_t. Then the agent receives a reward r_t according to the game score earned by a_t. The environment description is updated to s_{t+1}. The game stops when the agent reach some special states or the maximum steps.

We define the Q-function as the expected reward of taking the action a_t under the state s_t

$$Q(s_t, a_t) = E[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) | s_t, a_t],$$

where γ is a discount factor. For inference, we choose the action with the maximum Q-value

$$\pi(s_t) = \arg\max_{a_t \in A_t} Q(s_t, a_t).$$

In our framework, the Q-function is fitted by deep text matching networks taking textual descriptions of states and actions as input features.

4.2 Two-tower State-action Matching

Two-tower matching models have been used in various of text matching tasks (Huang et al., 2013; He et al., 2016; Chang et al., 2019). The query and candidate embeddings are from two independent text encoders and then aggregated to calculate the matching score.

We apply BERT (Devlin et al., 2019) as text encoders for both actions and states. The textual descriptions of candidate actions and states are encoded by different BERT models. The encoded representations are concatenated and then fed to a multi-layer perceptron (MLP).

4.3 Cross-attention State-action Matching

Cross-attention matching models is another type of matching models. Previous work (Guo et al., 2020a) have shown that cross-attention models usually produce better results but cost longer computation time in comparison with two-tower models.

We apply BERT as the joint encoder of actions and states. Following Devlin et al. (2019), we combine the text sequence of action and state and distinguish them by adding different segment tokens. The final representation of the first token `[CLS]` is then sent to a MLP to calculate the score.

4.4 Encode History Actions

To be updated.

4.5 Break Circular States

To be updated

Table 1: Scores for different games

Game	best steps	RAND	NAIL	TDQN	DRRN	DRRN_BERT	DRRN_BERT_MEMO	MaxScore
detective	50	113.7	136.9	169	197.8	200	**290**	360
deephome	350	1	13.3	1	1	14	**27**	300

Algorithm 1 Breaking Circular States

```
1: Initialize a dict $T$ to record the number of times
   each state is visited;
2: while game is not finished do
3:     Observe the state $s$;
4:     Update $T$;
5:     Sort valid actions $A$;
6:     Issue an action $A[T[s]]$;
7: end while
```

5 Visualization

While our game is purely based on natural language, sometimes it's hard to catch the decision process of our model throughout the whole game. Therefore, we build a 2D map on Unity and use it as a simulator to better demonstrate the continuous actions of our model. This can help us analyze our model and improve it at a macro perspective.

Figure 3 is an abstract 2D map of our game. There are tens of rooms and buildings. In some rooms, there are also lots of properties and evidence which are crucial to victory. The player needs to explore the map and collect them. Figure 4 is the structure of the whole game world in unity 2D. We can walk on the street and enter a random building, which can lead to a different result. The objects in the game may be a reward-able clue, may be useless, or may even cause the player's death. Figure 5 is the inside of a building. When we are at the front door, depending on the action the player chooses, we may go to different rooms and meet different events.

The scenes are properly ordered in unity2D and the main character has been put into every scene. Our player is able to move flexibly in the scene and cross the scenes with the camera following all the time. Our ultimate goal of the visualization is that creating an engine which can read the player actions from the model output and automatically simulate the movements and player-environment interaction in our map.

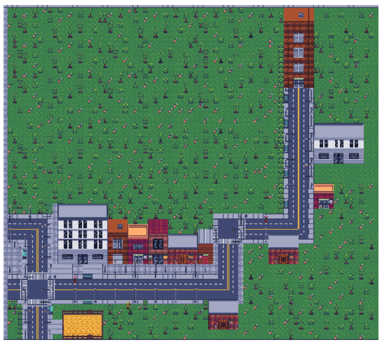


Figure 4: Unity 2D map of the game world.

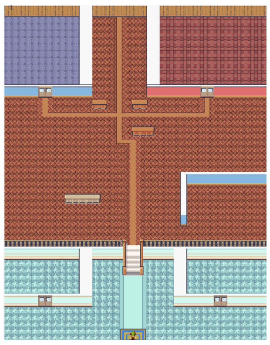


Figure 5: Unity 2D map inside a building.

6 Experiments

In this section, we conduct experiments on Jericho platform (Section 6.2). We also provide visualized case studies (Section 6.3), and detailed analysis of limitations (Section 6.4) of our models.

6.1 Implementation Details

We use the BERT-base model released by Google[5]. For the two-tower model, we set the max sequence length of actions as 16 and that of states as 256. For the cross-attention model, we set the max sequence length as 512. The model is trained asynchronously on 8 parallel instances of the game environment for 20k steps with a batch size of 64. Following previous work (Hausknecht et al., 2020), episodes are terminated after 100 valid steps or game over/victory. The learning rate is set to 1e-5 as suggested by Devlin et al. (2019). γ is set to 0.9. We implement our method based on Pytorch (Paszke et al., 2019) and Transformers (Wolf et al., 2019).

[5]https://github.com/google-research/bert

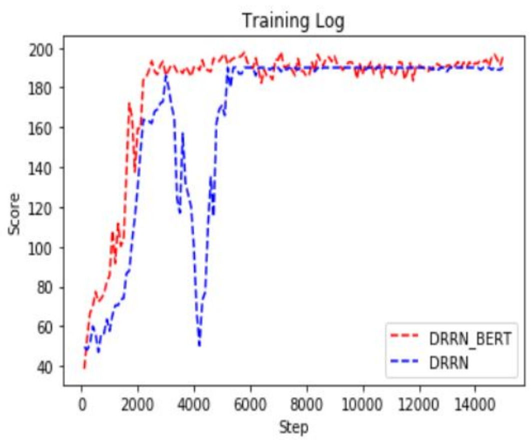


Figure 6: Training process of different methods.

6.2 Main Results

Table 1 shows the performance of our model and baselines on detective game. DRRN performs the best among all baselines. Our two-tower model DRRN_BERT is better than DRRN. The improvement shows that text adventure games can benefit from pre-trained language models.

Figure 6 shows the achieved scores during training. It is obvious that DRRN_BERT converges faster and the learning process is more stable. We suppose that it is because we start from the pre-trained language models instead of learning from scratch.

DRRN_BERT_MEMO

6.3 Case Study

6.3.1 Pre-midterm: the Detective Game

Figure 7 shows the difference between the result of our work and the standard human-played walk-through offered by the game developer. There are two walk-through paths in the figure, the green arrows represent the model output while the red arrows represent the standard human strategy.

Our model manages to avoid all the dangerous events and successfully arrives at the exit point. The two routes overlap to a large extent. In this specific game, Detective, our model is able to behave and take actions like a human. It utilizes the previous information and take actions under the reward policy of this game.

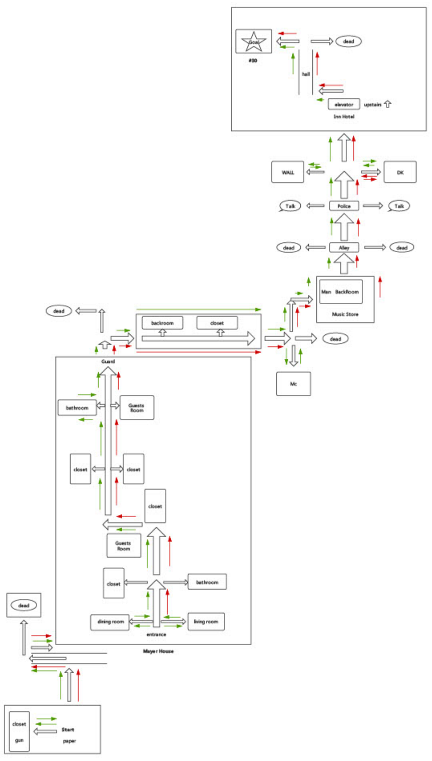


Figure 7: The midterm Detective game partial routes of our model output (green) and the standard human player walk-through (red).

6.3.2 Post-midterm: the Deephome Game

In the pre-midterm session, the games we selected in the prior-midterm session is simple and the total number of actions we need to win a game is less than one hundred. And the reward policy of the games we selected is quite stable. For example, in the Detective game, each valid action normally gets 10 to 20 scores of reward. Therefore, we plan to further launch experiments on large games.

Deephome is a super complex text-adventure game, which costs nearly 300 steps to walk through. Figure 8 shows the partial walk-through routes of our model(green) and the human player(red). Exploring the Ore Mine and City Generator has zero reward. While human player explores every place without reward evaluation, our model ignores these two places and only explore the Coal Mine which has reward. This means our model is able to get evaluate each possible action, then give up the useless action and only choose the reward-able action. This behavior saves lots of time and steps.

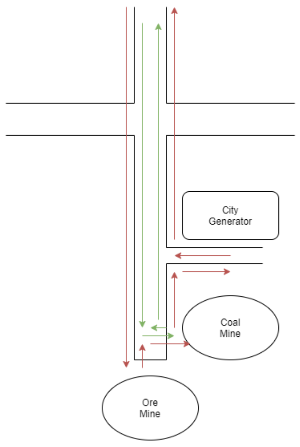


Figure 8: The final Deephome game partial routes of our model output (green) and the standard human player walk-through (red).

6.4 Limitations

Although we arrived at a milestone of several games, we still meet with several limitations. Our memory unit becomes ineffective when the game is super complex and the total number of actions we need to win a game is large. Second, sometimes our model might stick to a local optimal action and ignore the global optimal direction. This may further create a local endless loop. We must add additional constraints to avoid the endless loop.

7 Conclusions and Future Work

In this project, we incorporate pre-trained language models into the deep Q-learning framework to bring in rich commonsense knowledge existing in large text corpus. And we also combine history actions as internal prior knowledge to support the decision process. As a result, our DRRN_BERT_MEMO model and beats the previous state-of-the-art method. It performs well in several popular text-adventure games and achieves scores close to human performance.

In the future, we plan to implement knowledge graph building, because our current history memory sections might become ineffective when the game turns to be larger and larger. Linear history encoders can not remember extreme-long history, while the knowledge graph can. Besides, we can upgrade our visualization part and let the environment creation part and the user movement part become automatic. This may save us lots of time at the case study part.

References

Prithviraj Ammanabrolu and Matthew Hausknecht. 2019. Graph constrained reinforcement learning for natural language action spaces. In *International Conference on Learning Representations*.

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

Prithviraj Ammanabrolu and Mark Riedl. 2019a. 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.

Prithviraj Ammanabrolu and Mark O Riedl. 2018. Playing text-adventure games with graph-based deep reinforcement learning. *arXiv preprint arXiv:1812.01628*.

Prithviraj Ammanabrolu and Mark O Riedl. 2019b. Transfer in deep reinforcement learning using knowledge graphs. *EMNLP-IJCNLP 2019*, page 1.

Wei-Cheng Chang, X Yu Felix, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. 2019. Pre-training tasks for embedding-based large-scale retrieval. In *International Conference on Learning Representations*.

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. 2018. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Leyang Cui, Sijie Cheng, Yu Wu, and Yue Zhang. 2020. Does bert solve commonsense task via commonsense knowledge? *arXiv preprint arXiv:2008.03945*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Weiwei Guo, Xiaowei Liu, Sida Wang, Huiji Gao, Ananth Sankar, Zimeng Yang, Qi Guo, Liang Zhang, Bo Long, Bee-Chung Chen, et al. 2020a. Detext: A deep text ranking framework with bert. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2509–2516.

Xiaoxiao Guo, Mo Yu, Yupeng Gao, Chuang Gan, Murray Campbell, and Shiyu Chang. 2020b. Interactive fiction game playing as multi-paragraph reading comprehension with reinforcement learning. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7903–7910.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2015. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2016. 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.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338.

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

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. Spanbert: Improving pre-training by representing and predicting spans.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. Xlnet: Generalized autoregressive pretraining for language understanding.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. 2020. Keep calm and explore: Language models for action generation in text-based games. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. Ernie: Enhanced language representation with informative entities.

Richard Ziegfeld. 1989. Interactive fiction: A new literary genre? *New Literary History*, 20(2):341–372.