Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning

Prithviraj Ammanabrolu
Georgia Institute of Technology
Atlanta, GA 30332
raj.ammanabrolu@gatech.edu

Mark O. Riedl
Georgia Institute of Technology
Altanta, GA 30332
riedl@cc.gatech.edu

Abstract

Text-based adventure games provide a platform on which to explore reinforcement learning in the context of a combinatorial action space, such as natural language. We present a deep reinforcement learning architecture that represents the game state as a knowledge graph which is learned during exploration. This graph is used to prune the action space, enabling more efficient exploration. The question of which action to take can be reduced to a question-answering task, a form of transfer learning that pre-trains certain parts of our architecture. In experiments using the TextWorld framework, we show that our proposed technique can learn a control policy faster than baseline alternatives.

1 Introduction

Natural language communication can be used to affect change in the real world. Research at the intersection of reinforcement learning and natural language processing has shown that text-based games are a useful testbed for developing and testing reinforcement learning algorithms that process and respond to natural language inputs (Narasimhan et al., 2015; He et al., 2016). Formally, as defined by the TextWorld framework (Côté et al., 2018), a text-based game is a partially observable Markov decision process (POMDP), a 7-tuple of $\langle S, T, A, \Omega, O, R, \gamma \rangle$ representing the set of environment states, conditional transition probabilities between states, words used to compose text commands, observations, observation conditional probabilities, reward function, and the discount factor respectively. In text-based games, the agent never has access to the true underlying world state and has to reason about how to act in the world based only on the textual observations. Additionally, the agent's actions must be expressed through natural language commands, ensuring that the action space is combinatorially large. We thus see that text-based games pose a different set of challenges than traditional video games. Text-based games require a greater understanding of previous context to be able to explore the state-action space more effectively. Such games have historically proven to be difficult to play for AI agents, and the more complex variants such as *Zork* still remain firmly out of the reach of existing approaches.

We introduce two contributions to text-based game playing to deal with the combinatorially large state and action spaces. First, we show that a state representation in the form of a *knowledge graph* gives us the ability to effectively prune an action space. A knowledge graph captures the relationships between entities. The graph enables the agent to have a prior notion of what actions it should not take at a particular stage of the game. We show that having an action space pruned using this knowledge graph allows for faster convergence to an optimal control policy using an ϵ-greedy reinforcement learning algorithm—the agent doesn't waste trials on actions that will likely have low utility.

Our second contribution is a deep reinforcement learning architecture, Knowledge Graph DQN, that effectively uses this state representation to estimate the Q-value for a state-action pair. This architecture leverages recent advances in graph embedding and attention techniques (Guan et al.,

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

2018; Veličković et al., 2018) to learn which portions of the graph to pay attention to given an input state description in addition to having a mechanism that allows for natural language action inputs.

The framing of the POMDP as a question-answering (QA) problem is also explored. Previous work has shown that many NLP tasks can be framed as instances of question-answering and that we can transfer knowledge between these tasks (McCann et al., 2017). Similarly, we show how pre-training certain parts of the deep reinforcement learning architecture using existing QA methods improves performance. Although, our results are evaluated using games generated by the TextWorld framework, we do not make use of any information regarding TextWorld's state or quest generation functions during pre-training and hypothesize that our method can be extended to other text-based games. We present preliminary results on ablative experiments comparing approaches such as Bag-of-Words DQN, LSTM-DQN, and variations of our own architecture.

2 Related Work

A growing body of research has explored the challenges associated with text-based games (Narasimhan et al., 2015; Haroush et al., 2018; Côté et al., 2018). Narasimhan et al. (2015) attempt to solve parser-based text games by encoding the observations using an LSTM. This encoding vector is then used by an action scoring network that determines the scores for the action verb and each of the corresponding argument objects. The two scores are then averaged to determine Q-value for the state-action pair. He et al. (2016) present the Deep Reinforcement Relevance Network (DRRN) which uses two separate deep neural networks to encode the state and actions. The Q-value for a state-action pair is then computed by a pairwise interaction function between the two encoded representations. Both of these methods are not conditioned on previous observations and so are at a disadvantage when dealing with complex partially observable games. Additionally, neither of these approaches prune the action space and so end up wasting trials exploring state-action pairs that are likely to have low Q-values, likely leading to slower convergence times for combinatorially large action spaces.

Haroush et al. (2018) introduce the Action Eliminating Network (AEN) that attempts to restrict the actions in each state to the top-k most likely ones, using the emulator's feedback. The network learns which actions should not be taken given a particular state. Their work shows that reducing the size of the action space allows for more effective exploration, leading to better performance. Their network is also not conditioned on previous observations.

Knowledge graphs have been demonstrated to improve natural language understanding in other domains outside of text adventure games. For example, Guan et al. (2018) use commonsense knowledge graphs such as *ConceptNet* (Speer and Havasi, 2012) to significantly improve the ability of neural networks to predict the end of a story. They represent the graph in terms of a knowledge context vector using features from *ConceptNet* and graph attention (Veličković et al., 2018). The state representation that we have chosen as well as our method of action pruning builds on the strengths of existing approaches while simultaneously avoiding the shortcomings of ineffective exploration and lack of long-term context.

3 Knowledge Graph DQN

3.1 State Representation and Action Pruning

In our approach, our agent learns a knowledge graph, stored as a set of RDF triples, i.e. 3-tuples of $\langle subject, relation, object \rangle$. These triples are extracted from the observations using Stanford's Open Information Extraction (OpenIE) (Angeli et al., 2015) with some additional rules specific to text adventural games to account for the fact that OpenIE is not optimized to the regularities of text adventure descriptions. This gives the agent what essentially amounts to a mental map of the game.

The knowledge graph is updated after every agent action (see Figure 1). The update rules are defined such that there are portions of the graph offering short and long-term context. The "you" node represents the agent and relations out of this node are updated after every action, with the exception of certain types of relations relating to inventory. Other relations persist after each action. We intend for the update rules to be applied to text-based games in different domains and so only hand-craft a minimal set of rules that we believe apply generally, found in the Appendix A.

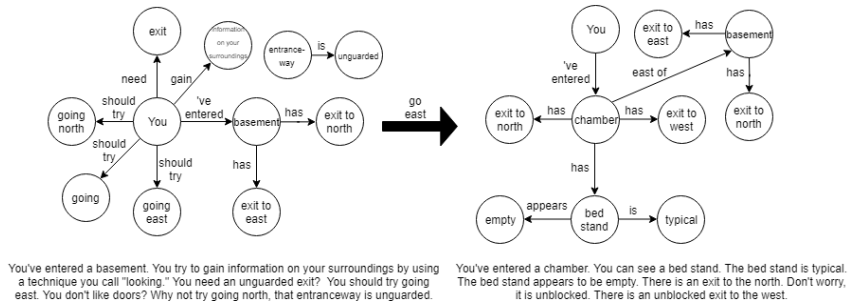

Figure 1: Graph state update example given two observations

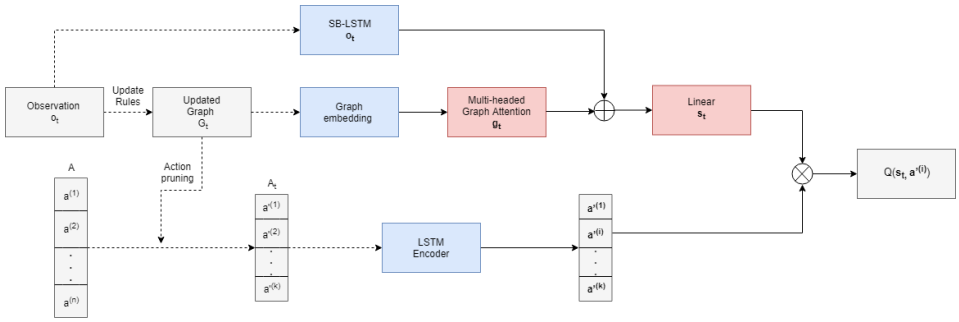

Figure 2: KG-DQN architecture, blue shading indicates components that can be pre-trained and red indicates no pre-training. The solid lines indicate gradient flow for learnable components.

The knowledge graph is used to prune the combinatorially large space of possible actions available to the agent. Given the current state representation s_t, the action space is pruned by ranking the full set of actions and selecting the top-k. Actions are scored by summing the following: +1 for each object in the action that is present in the graph; and +1 if there exists a valid directed path between the two objects in the graph. We indirectly make the assumption that each action has at most two objects, inserting a key in a lock for example.

3.2 Model Architecture and Training

Narasimhan et al. (2015) define a fixed set of actions for parser based text-games. Similarly, we also reduce the TextWorld parser-based game to a fixed action-set DQN by defining a set of actions A that contain every action that can be taken in the game. When playing the game, the agent receives the observation o_t from the simulator, which is a textual description of current game state. The state graph G_t is updated according to the given observation, and the current pruned action set A_t is computed as described in Section 3.1. We use the Q-Learning technique (Watkins and Dayan, 1992) to learn a control policy $\pi(a_t|s_t)$, $a_t \in A$, which gives us the probability of taking action a_t given the current state s_t. The policy is determined by the Q-value of a particular state-action pair, which is updated using the Bellman equation (Sutton and Barto, 2018). The policy is thus to take the action that maximizes the Q-value in a particular state, which will correspond to the action that maximizes the reward expectation given that the agent has taken action a_t at the current state s_t and followed the policy $\pi(a|s)$ after.

The architecture in Figure 2 is responsible for computing the representations for both the state s_t and the actions $a^{(i)} \in A$ and coming to an estimation of the Q-value for a particular state and action. Actions are pruned by scoring each $a \in A$ according to the mechanism previously described using G_t. We then embed and encode all of these action strings using an LSTM encoder (Sutskever et al., 2014). During the forward activation, the agent first uses the observation to update the graph G_t using the rules outlined in Section 3.1. The graph is then embedded into a single vector $\mathbf{g_t}$ using Graph Attention Veličković et al. (2018) where the node features consist of the averaged word vectors for that node. Simultaneously, an encoded representation of the observation $\mathbf{o_t}$ is computed using a

Algorithm 1 ϵ_1, ϵ_2-greedy learning algorithm for KG-DQN

for episode=1 to M **do**
  Initialize action dictionary A and graph G_0
  Reset the game simulator
  Read initial observation o_1
  $G_1 \leftarrow updateGraph(G_0, o_1)$; $A_1 \leftarrow pruneActions(A, G_0)$ ▷ Section 3.1
  for step t=1 to T **do**
    if $random() < \epsilon_1$ **then**
      if $random() < \epsilon_2$ **then**
        Select random action $a_t \in A$
      else
        Select random action $a_t \in A_t$
    else
      Compute $Q(\mathbf{s_t}, \mathbf{a^{(i)}}; \theta)$ for $a^{(i)} \in A$ for network parameters θ ▷ Section 3.2, Eq. 1
      Select a_t based on $\pi(a|s_t)$
    Execute action a_t in the simulator and observe reward r_t
    Receive next observation $ot + 1$
    $G_{t+1} \leftarrow updateGraph(G_t, o_{t+1})$; $A_{t+1} \leftarrow pruneActions(A, G_{t+1})$ ▷ Section 3.1
    Compute $\mathbf{s_{t+1}}$ and $\mathbf{A_{t+1}} = \{\mathbf{a'^{(i)}}$ for all $a'^{(i)} \in A\}$ ▷ Section 3.2
    Set priority $p_t = 1$ if $r_t > 0$, else $p_t = 0$
    Store transition $(\mathbf{s_t}, \mathbf{a_t}, r_t, \mathbf{s_{t+1}}, \mathbf{A_{t+1}}, p_t)$ in replay buffer D
    Sample mini-batch of transitions $(\mathbf{s_k}, \mathbf{a_k}, r_k, \mathbf{s_{k+1}}, \mathbf{A_{k+1}}, p_k)$ from D, with fraction ρ having $p_k = 1$
    Set $y_k = r_k + \gamma \max_{\mathbf{a} \in \mathbf{A_{k+1}}} Q(\mathbf{s_t}, \mathbf{a}; \theta)$, or $y_k = r_k$ if s_{k+1} is terminal
    Perform gradient descent step on loss function $L(\theta) = (y_k - Q(\mathbf{s_t}, \mathbf{a_t}; \theta))^2$

Sliding Bidirectional LSTM (SB-LSTM). The final Q-value for a state-action pair is given as:

$$Q(\mathbf{s_t}, \mathbf{a_t}) = f(W_l(\mathbf{g_t} \oplus \mathbf{o_t}) + b_l) \cdot \mathbf{a_t} \tag{1}$$

where W_l, b_l represent the final linear layer's weights and biases. Note that this method of separately computing the representations for the state and action is similar to the approach taken in the DRRN (He et al., 2016).

We train the network using experience replay (Lin, 1993) with prioritized sampling (cf., Moore and Atkeson (1993)) and a modified version of the ϵ-greedy algorithm (Sutton and Barto, 2018) that we call the ϵ_1, ϵ_2-greedy learning algorithm. The experience replay strategy finds paths in the game, which are then stored as transition tuples in a experience replay buffer D. The ϵ_1, ϵ_2-greedy algorithm explores by choosing actions randomly from A with probability ϵ_1 and from A_t with a probability ϵ_2. This is to account for situations in which an action for which the agent has no prior on based on G_t must be taken to advance the game. We then sample a mini-batch of transition tuples consisting of $(\mathbf{s_k}, \mathbf{a_k}, r_k, \mathbf{s_{k+1}}, \mathbf{A_{k+1}}, p_k)$ from D and compute the temporal difference loss as: $L(\theta) = r_k + \gamma \max_{\mathbf{a} \in \mathbf{A_{k+1}}} Q(\mathbf{s_t}, \mathbf{a}; \theta) - Q(\mathbf{s_t}, \mathbf{a_t}; \theta)$. Replay sampling from D is done by sampling a fraction ρ from transition tuples with a positive reward and $1 - \rho$ from the rest. The exact training mechanism is described in Algorithm 1.

Portions of our deep Q-network can be pre-trained using a question-answering technique. That is, we can model playing text-based games using the question-answering paradigm that learns to answer the question of what action to take given an observation. The process of pre-training the network involves generating traces of actions and states from similar games using an oracle—an agent that is capable of finishing the game perfectly in the least number of steps possible—to provide a label for the best action to take. We then use the DrQA (Chen et al., 2017) technique to compute the weights for the SB-LSTM in Figure 2. The observation embeddings and action embeddings learned during pre-training are also used to initialize the embedding layers for the graph attention and in the action LSTM encoder, respectively (blue boxes in Figure 2). The QA network is not trained on the same games that the rest of the architecture is trained on, thus representing a form of transfer learning.

4 Experiments

We conduted experiments in the TextWorld framework (Côté et al., 2018) using their "home" theme. The games were generated with different random seeds; all games have 10 rooms, 20 total objects, and have an objective that requires 5 actions to be performed, though other actions, such as movement, may be necessary precursors for those actions. The reward function is as follows: +1 for each action taken that moves the agent closer to finishing the quest; -1 for each action taken that extends the minimum number of steps needed to finish the quest from the current stage; 0 for all other situations. The maximum achievable reward in our test game is 5.

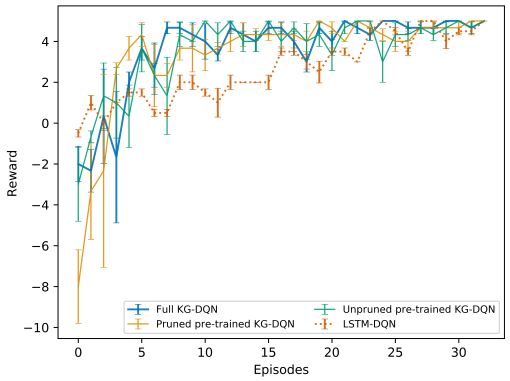

Figure 3: Reward learning curve for select experiments

Model	Steps
Random Command	319.8
BOW-DQN	83.1 (8.0)
LSTM-DQN	72.4 (4.6)
Unpruned KG-DQN	131.7 (7.7)
Pruned KG-DQN	97.3 (9.0)
Full KG-DQN	73.7 (8.5)

Table 1: Average number of steps taken to complete the game

Pre-training of the SB-LSTM within the QA architecture is conducted by generating 200 games from the same theme. The QA system was then trained on data from walkthroughs of a subset of 160 of these generated games and evaluated on the held-out set of 40 games. A random game was chosen from the test-set of games and used to perform the rest of the experiments. The vocabulary size for the final testing game is 746 and it has a branching factor (size of action set A) of 143.

We compare our technique to three baselines: random command (sampling from the list of admissible actions returned by the TextWorld simulator at each step), a Bag-of-Words DQN, and LSTM-DQN (Narasimhan et al., 2015). To achieve the most competitive baselines, we used a randomized grid search to choose the best hyperparameters for the BOW-DQN and LSTM-DQN baselines (e.g., hidden state size, γ, ρ, etc.). We tested three versions of our KG-DQN: (1) un-pruned with pre-training, (2) pruned without pre-training, and (3) pruned with pre-training (full). Our models use 50 dimensional word embeddings, 2 heads on the graph attention layers, mini-batch size of 16, and perform a gradient descent update every 5 steps taken by the agent. The models are evaluated by observing the time to reward convergence (Figure 3) and the average number of steps required to finish the game with $\epsilon = 0.1$ over 5 episodes after training has completed (Table 1) . All results are averaged across 5 separate trials with the standard deviations shown.

5 Results and Conclusions

KG-DQN converges at least 40% faster than baselines (we don't show BOW-DQN in Figure 3 because it is inferior to LSTM-DQN). The pre-trained and non-pre-trained versions of KG-DQN converge at about the same rate. The full version of our agent completes quests with roughly the same number of steps as LSTM-DQN. TextWorld's reward function allows for a lot of exploration of the environment without penalty so it is possible for a model that has converged on the maximum total reward to complete quests in as few as five steps or in many hundreds of steps. That KG-DQN achieves faster reward convergence and uses a relatively low number of steps to complete quests suggests that it not only optimizes to the reward function but learns additional knowledge along the way that guides it to make good choices. In LSTM-DQN this knowledge is captured through a hidden state that is transferred from step to step. In KG-DQN, this knowledge is made explicit—and interpretable—in the knowledge graph, which can grow arbitrarily large without forgetting.

Our experimental results suggest that incorporating knowledge graphs into DQN for playing text adventure games is a promising approach. Furthermore, modeling the agent's action selection as a question-answering problem provides the opportunity for transfer learning. Future work includes testing on more complex games, improving the use of knowledge graph features, and developing an end-to-end architecture that learns graph update and action pruning rules. Our results suggest that knowledge graphs and question-answering frameworks provide a promising path forward toward agents that use natural language to affect change on partial observable worlds.

References

G. Angeli, J. Premkumar, M. Jose, and C. 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)*, 2015.

D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*, 2017.

M.-A. Côté, Á. Kádár, X. Yuan, B. Kybartas, E. Fine, J. Moore, M. Hausknecht, L. E. Asri, M. Adada, W. Tay, and A. Trischler. TextWorld : A Learning Environment for Text-based Games. In *Proceedings of the ICML/IJCAI 2018 Workshop on Computer Games*, page 29, 2018.

J. Guan, Y. Wang, and M. Huang. Story Ending Generation with Incremental Encoding and Commonsense Knowledge. *arXiv:1808.10113v1*, 2018. URL `https://arxiv.org/pdf/1808.10113.pdf`.

M. Haroush, T. Zahavy, D. J. Mankowitz, and S. Mannor. Learning How Not to Act in Text-Based Games. In *Workshop Track at ICLR 2018*, pages 1–4, 2018.

J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf. Deep Reinforcement Learning with a Natural Language Action Space. In *Association for Computational Linguistics (ACL)*, 2016.

L.-J. Lin. *Reinforcement learning for robots using neural networks*. PhD thesis, Carnegie Mellon University, 1993.

B. McCann, N. S. Keskar, C. Xiong, and R. Socher. The Natural Language Decathlon : Multitask Learning as Question Answering. *arXiv:1806.08730*, 2017. URL `https://arxiv.org/pdf/1806.08730.pdf`.

A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, Oct 1993.

K. Narasimhan, T. Kulkarni, and R. Barzilay. Language Understanding for Text-based Games Using Deep Reinforcement Learning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

R. Speer and C. Havasi. Representing General Relational Knowledge in ConceptNet 5. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, 2012.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *International Conference on Learning Representations (ICLR)*, 2018.

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

A Appendix

Knowledge Graph Update Rules

Expanding on the summary of the update rules given in Figure 1 and Section 3.1, we add a minimal set of rules to help Stanford's OpenIE deal with the regularities of the TextWorld environment. They are:

1. Linking the current room type (e.g. "basement", "chamber") to the items found in the room with the relation "has"
   (a) E.g.: <chamber, has, bed stand>
2. Extracting information regarding entrances and exits and linking them to the current room.
   (a) E.g.: <basement, has, exit to north>
3. Removing all relations relating to the "you" node with the exception of inventory every action.
   (a) E.g.: <you, have, cubical key>
4. Linking rooms with directions based on the action taken to move between the rooms.
   (a) E.g.: <chamber, east of, basement> after the action "go east" was taken to go from the basement to the chamber

All other RDF triples generated are taken from OpenIE.