Reinforcement Learning and `Text Games`

Anonymous ACL submission

Abstract

We consider the task of training agents that play text-based games by learning and grounding language interactively. A competent agent will have good natural language understanding and an ability to learn optimal control policies. This is a sequential decision making problem where the world and actions are represented by text, hence, solving this problem requires a systematic integration between reinforcement learning and natural language processing. Progress in building AI that efficiently plays text-based games will have important impacts on this RL + NLP integration. Furthermore, the ability to learn optimal control policies in systems where action space is defined by sentences in natural language would allow many interesting real-world applications including (but not limited to) automatic optimization of dialogue systems (Chen et al., 2017).

1 Introduction

A defining component of human intelligence is the ability to interact with the world, as well as other humans, in an efficient way to learn and acquire knowledge. This interaction creates a rich spectrum of sensory stimuli that would otherwise be absent, which makes it a crucial skill for any machine learning algorithm to reach human level intelligence.

Traditionally, the dominant approach in NLP has been to gather large corpora of text and train deep models on them. These models pick up statistically prevalent patterns in language but they lack the core characteristic of language: *that it is used for interaction!*

We expect that in the future, the research community will slowly move beyond these static, statistical approaches of language modeling toward a more dynamic, interactive approach, in the form of sequential decision making problems.

The notion of *”Interactive perception”* (Bohg et al., 2017) has been receiving more and more attention not only in the field of robotics and natural language processing, but also in other, seemingly less relevant fields such as computer vision. In a recent survey paper, (Luketina et al., 2019) argue that:

> ”The time has come for natural language to become a first-class citizen of solutions to sequential decision making problems.”

Following this trend, we aim to build agents that learns to play text-games using deep reinforcement learning methods. These text games are synthetic environments and are significantly simpler than real-world scenarios; however, they still provide a promising platform to develop and test different approaches to the sequential decision making informed by natural language.

In text games the player has to complete a quest purely through textual descriptions and commands. At every stage of the game, the player is prompted with a text description of the quest and the environment they are in (e.g. `"You are hungry in a kitchen and there is a fridge in the corner."`) and responds with an action (`"Open fridge."`). The game then responds with a new description (`"You open fridge revealing an apple."`), and the process repeats.

More formally, text games are sequential decision making tasks for which states and actions are given in natural language. The goal of the player is to learn an optimal policy (i.e. a function that maps states to actions) such that it maximizes the rewards.

The paper is structured as follows: in section 2 we define the problem and the dataset used, section 3 has a survey of existing literature in this subject, in section 4 we go through the necessary background, in section 5 we describe our experiments and results, and finally in section 6 we conclude and discuss future work.

2 Problem Setup

Complex environments make training RL agents challenging for several reasons, including sparsity of reward, the credit assignment problem, exploration vs. exploitation and so on. For playing text games there are additional language challenges such as natural language understanding and reasoning, partially observable game states, and having access to only local information such as the current room description and the player's inventory. Another open problem is text generation when dealing with a combinatorially large action space. In this study we avoid dealing with this problem by using the information available about admissible commands.

2.1 The environment

For our experiments we chose to use TextWorld (Côté et al., 2018), an open source, extensible learning environment for reinforcement learning in text-based games. TextWorld framework allows us to generate games that are distinct but related with controlled vocabulary, and can be parser based[1] or choice-based[2] as desired. We will use three class of games to train and test our agents.

tw-simple These games take place in a typical house and consists in finding the right food item and cooking it. The reward setting can be set to one of {*dense, balanced, sparse*}, and the goal description can be {*detailed, brief, none*}. The following terminal command can be used to generate these games::

```
$ tw−make tw−simple \
−−goal detailed −−rewards dense \
−−output /games/simple−game.ulx −v
```

custom TextWorld can also be used to generate customized games. The options are:

- –world-size: controls the number of rooms in the world
- –nb-objects: controls the number of objects that can be interacted with (excluding doors)
- –quest-length: controls the minimum number of commands that is required to type in order to win the game

Here is an example commands to generate a custom game:

```
$ tw−make custom −−theme house \
−−world−size 2 −−nb−objects 5 \
−−quest−length 1 −−output \
/games/custom−game.ulx −v −f
```

tw-cooking These games are part of an ongoing competition. The goal is to cook food following a recipe. This involves finding food items and processing them accordingly. These games are much more difficult compared to tw-simple and custom games. This is because of sparsity of reward and brief description of objectives, but also because of additional complications in the quest. To win these games, the agent needs to: *go* to other rooms in the map, *open* containers and doors, *take* ingredients, *drop* items (due to limited inventory capacity), *cut* food items (needs to find knife), *cook* food items (grill, roast, fry), and eventually prepare and eat a meal. See Fig 1

2.2 Evaluation and Handicaps

The goal is to train agents that can play these games efficiently and win the game in as few steps as possible. Therefore, our evaluation metrics are (i) the total score accumulated by the agent and (ii) how fast it was able to achieve it. We will compare the performance of our agent to random baseline, as well as humans.

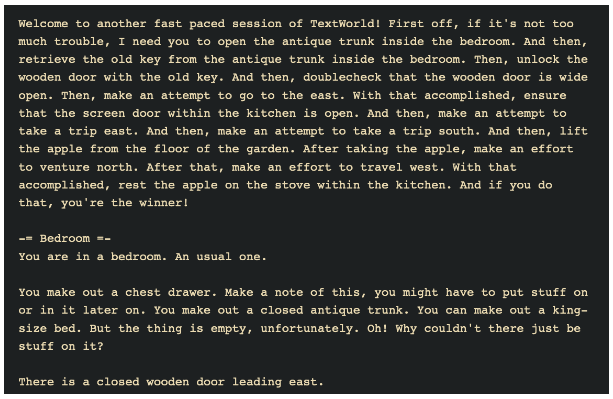
To make training more tractable, agents can request additional information from the game engine. In our experiments we consistently provide the agent with the list of admissible commands at every step of the game. This circumvents the difficulties of text generation, and in effect, turns the games from parser-based to choice based.

3 Related Work

LSTM-DQN (Narasimhan et al., 2015) proposed an agent with an LSTM representation generator that encodes sentence descriptions of the game, and a neural network action scorer that learns two separate Q values (Watkins and Dayan, 1992) for action-object pairs given the current state representation. The final Q value is calculated by averaging these two values. Information about all possible

[1]Parser-based: game simulator accepts any text input

[2]Choice-based: the player selects a text input from given set of choices



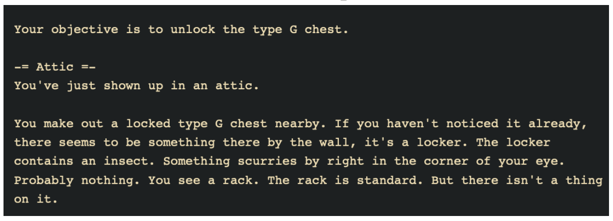
```
Welcome to another fast paced session of TextWorld! First off, if it's not too
much trouble, I need you to open the antique trunk inside the bedroom. And then,
retrieve the old key from the antique trunk inside the bedroom. Then, unlock the
wooden door with the old key. And then, doublecheck that the wooden door is wide
open. Then, make an attempt to go to the east. With that accomplished, ensure
that the screen door within the kitchen is open. And then, make an attempt to
take a trip east. And then, make an attempt to take a trip south. And then, lift
the apple from the floor of the garden. After taking the apple, make an effort
to venture north. After that, make an effort to travel west. With that
accomplished, rest the apple on the stove within the kitchen. And if you do
that, you're the winner!

-= Bedroom =-
You are in a bedroom. An usual one.

You make out a chest drawer. Make a note of this, you might have to put stuff on
or in it later on. You make out a closed antique trunk. You can make out a king-
size bed. But the thing is empty, unfortunately. Oh! Why couldn't there just be
stuff on it?

There is a closed wooden door leading east.
```

(a) tw-simple

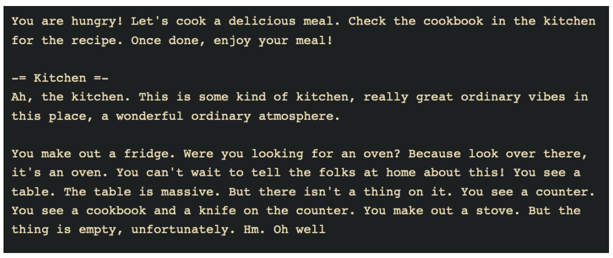


```
Your objective is to unlock the type G chest.

-= Attic =-
You've just shown up in an attic.

You make out a locked type G chest nearby. If you haven't noticed it already,
there seems to be something there by the wall, it's a locker. The locker
contains an insect. Something scurries by right in the corner of your eye.
Probably nothing. You see a rack. The rack is standard. But there isn't a thing
on it.
```

(b) custom



```
You are hungry! Let's cook a delicious meal. Check the cookbook in the kitchen
for the recipe. Once done, enjoy your meal!

-= Kitchen =-
Ah, the kitchen. This is some kind of kitchen, really great ordinary vibes in
this place, a wonderful ordinary atmosphere.

You make out a fridge. Were you looking for an oven? Because look over there,
it's an oven. You can't wait to tell the folks at home about this! You see a
table. The table is massive. But there isn't a thing on it. You see a counter.
You see a cookbook and a knife on the counter. You make out a stove. But the
thing is empty, unfortunately. Hm. Oh well
```

(c) tw-cooking

Figure 1: Sample games.

verb-object pairs is available to the model at all times. It is shown that LSTM-DQN consistently outperforms random agents, as well as BOW-DQN and BOW-LIN models. These experiments demonstrate the importance of learning good representations of text in order to play these games well.

being-in-the-world Constraining the action space of the text game to verb-object pairs and having separate Q function for each of them provides enough structure to make learning more tractable. Building agents with independent modules that communicate to achieve a shared goal is not new. (DePristo and Zubek, 2001) argued that employing such a hybrid architecture helps their agent perform better in a Multi-User Dungeon (MUD) world.

DRRN (He et al., 2015) introduced Deep Reinforcement Relevance Network (DRRN) for playing *hypertext-based*[3] games using a simple BOW representation of the input text. DRRN uses separate word embeddings for action and state pairs and its learning algorithm is a variant of DQN (Mnih et al., 2013). The final Q value is obtained by calculating the inner product of Q value vectors for state-action pairs.

KG-DQN A popular structure for NLP is the *Knowledge Graph (KG)* (Gruber, 1993). (Ammanabrolu and Riedl, 2018) recently proposed a deep RL framework that represents the game state as a KG which is learned during exploration. The KG provides a persistent memory of the environment which in turn helps their agent learn a control policy faster than baseline alternatives. As another KG based approach, (Barzdins et al., 2019) used the RDF* graph database as a world model to win a previous iteration of the TextWorld competition.

4 Background

4.1 Text Game Definition

Text games can be seen as partially observable Markov decision processes (POMDP) where the environment state is never observed directed. Formally, a text game is a discrete-time POMDP defined by $(S, T, A, \Omega, O, R, \gamma)$ where S is a set of environment states, T is conditional transition probabilities between states, A is the set of text commands, Ω is the set of observation, O is a set of conditional observation probabilities, R is the reward function which maps state and action pairs to a real number and $\gamma \in [0, 1]$ is the discount factor.

4.2 Learning Algorithm

Reinforcement Learning (Sutton and Barto, 2018) is a commonly used framework for learning control policies in game environments. Here we use Advantage Actor Critic (A2C), a policy gradient-based algorithm to tackle the learning problem. Below we provide a short introduction to reinforcement learning algorithms and briefly discuss how are they used to learn optimal control policies.

Policy Gradient Policy gradient methods are ubiquitous in model free reinforcement learning. It is also the ”actor” part of Actor-Critic methods. The objective function for policy gradients is de-

[3]Hypertext-based: actions are represented as clickable links inside the state description

fined as

$$J_\pi(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T-1} r_{t+1}\Big], \tag{1}$$

where r_t is the reward received at time t and π is the policy. The goal is to maximize the total amount of rewards accumulated over time. This is a maximization problem so we can optimize the policy by using gradient ascent

$$\theta \leftarrow \theta + \frac{\partial}{\partial\theta} J_\pi(\theta). \tag{2}$$

According to policy gradient theorem, it can be shown that

$$\nabla_\theta J_\pi(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log\, \pi_\theta(a_t|s_t) G_t, \tag{3}$$

where $G_t = \sum_{t'=t+1}^{T-1} \gamma^{t'-t-1} r_{t'}$ is the return, or discounted future reward. This is the Monte-Carlo policy gradient algorithm also known as REINFORCE (Williams, 1992).

Q-learning Not all reinforcement learning algorithms are on-policy. In Q-learning (Watkins and Dayan, 1992; Mnih et al., 2013), a model free off-policy algorithm, the agent takes an action a in state s by consulting a state-action value function $Q(s, a)$, which is a measure of the action's expected long term reward. That is

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi}\Big[G_t \big| S_t = s, A_t = a\Big]. \tag{4}$$

Starting from a random Q-function, the agent continuously updates its Q-values by playing the game and obtaining rewards. The iterative updates are derived from

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a'), \tag{5}$$

which is known as the Bellman equation (Bellman, 1954).

Actor Critic Actor-Critic algorithms are policy-based but they also utilize value functions to have the best of the two worlds.

In REINFORCE algorithm, we update the policy parameter through Monte-Carlo updates (i.e. taking random samples). This introduces an inherent high variability in log probabilities and cumulative reward values, because each trajectories during

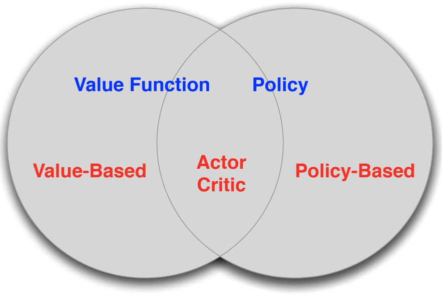


Figure 2: From David Silver's RL course slides

training can deviate from each other at great degrees. Consequently, the high variability in log probabilities and cumulative reward values will make noisy gradients, and cause unstable learning and/or the policy distribution skewing to a non-optimal direction. Policy gradient methods can be improved by introducing a baseline

$$\nabla_\theta J_\pi(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log\, \pi_\theta(a_t|s_t)(G_t - b(s_t)). \tag{6}$$

The return G_t is by definition always positive, so intuitively, including this baseline will help distinguish between "good" actions that lead to high return (higher than the baseline) and "bad" actions that lead to low returns (lower than the baseline). Adding the baseline reduces the variance and stabilizes learning. Moreover, it is theoretically justified as long as the baseline doesn't depend on a:

$$\begin{aligned} &\mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\, \pi_\theta(s, a)\, b(s)\Big] \\ &\quad= \sum_{s\in S} d^{\pi_\theta}(s) \sum_a \nabla_\theta\, \pi_\theta(s, a) b(s) \\ &\quad= \sum_{s\in S} d^{\pi_\theta}(s) b(s) \nabla_\theta \sum_a \pi_\theta(s, a) \\ &\quad= 0. \end{aligned} \tag{7}$$

The last equality hods true because $\sum_a \pi_\theta(s, a) = 1$. There are a variety of options for the baseline function $b(s)$ that lead to slightly different actor-critic algorithms. Here we use the state value function $V^\pi(s)$ as our baseline function

$$\nabla_\theta J_\pi(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\, \pi_\theta(a_t|s_t) A^\pi(s, a)\Big], \tag{8}$$

where we have defined the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s, a)$, hence the name Advantage Actor Critic.

5 Experiments

Model Architecture Our proposed model is fairly simple and consists of GRUs and feedforward neural networks. At playing time, the observations and admissible commands are first embedded then encoded using two separate GRUs. The observation hidden vector is then used to update the current state, encoded by another GRU. Then the state representation vector and hidden vector of admissible commands are pushed through a command scorer neural network that calculates the $Q(s, a)$ values. The output from state GRU is processed by another neural network, the critic, to calculate the value function $V(s)$. Finally, we use a softmax policy that samples actions from

$$\pi(a|s) = \frac{\exp\Big(Q(s,a)\Big)}{\sum_{a'} \exp\Big(Q(s,a')\Big)}. \tag{9}$$

The loss function is standard A2C function. We also experimented with entropy regularization inspired from a recent proposal called Soft Actor Critic (Haarnoja et al., 2018) and observed that it doesn't help our models. See Fig 4a. This is probably because SAC is more suitable for problems with continuous action spaces.

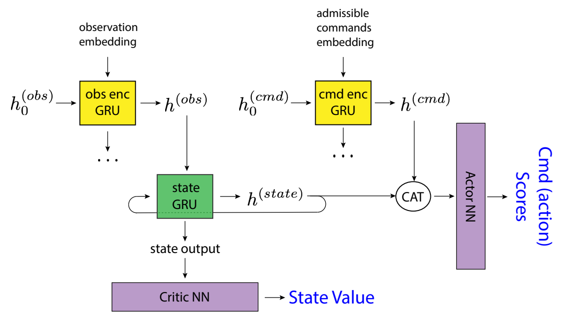


Figure 3: Agent architecture.

For our main experiments, we train the word embeddings alongside the other parameters in the model.

5.1 tw-simple games

Setup We generated 100 training and 20 validation tw-simple games with *dense* rewards and *detailed* goal descriptions to make training as easy as possible. We then trained and tested our A2C agent using these games. To investigate the benefits of transfer learning, we used BERT (Devlin et al., 2019) to get contextual word embeddings of the observations and commands to provide the agent with some weak common sense about the world. This approach is expected to improve model performance compared to the case where embeddings were learned on the fly at training time. Additionally, we added a fully-connected layer between outputs from BERT and the rest of our model to reduce dimensionality.

Results Our A2C agent with trained embeddings consistently outperforms the random agent by a large margin, but is inferior to humans. Surprisingly, not only did including BERT not help at all, it even hurt the model performance. This could be due to several reasons. Our first instinct was dimensionality mismatch. The smallest possible size of word embeddings produced by BERT is 768, as opposed to the embedding dimensionality of 128 we used for the other model. However, adding a fully-connected layer with 128 units to reduce dimensionality didn't help either. Therefore, there must be another explanation. We kept track of the number of unique words that occur in these games and realized that TextWorld is a small world: it has a vocabulary of less than a thousand words. This is not large enough to utilize the full power of BERT, a model with millions of parameters trained on corpora with orders of magnitude larger vocabulary.

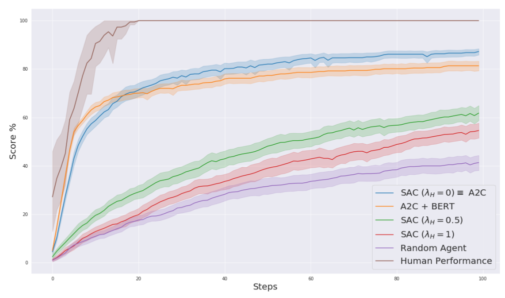
5.2 custom games

setup To find out how fast our agent was able to win the games, we studied its behavior for games generated with given quest lengths $\in \{1, 2, 3, 4\}$. Again, the train/valid split was 100/20 games. Other defining parameters for these games were uniformly sampled: world size $\in \{1, 2, 3\}$ and number of objects $\in \{5, 10\}$.

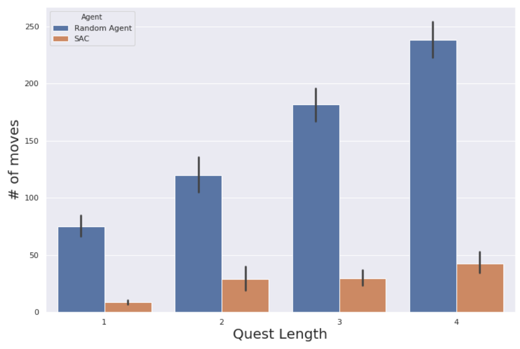
Results Once more, our agent completely outperforms the random baseline but can't quite match the human skyline.

5.3 tw-cooking games

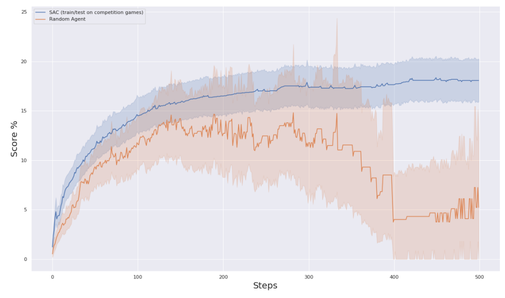
setup We trained an identical agent as before on 4440 training games from the competition and validated on 222 validation games. These games are longer and they come with sparse or balanced rewards. The goal is only partially described and the rest of the quest objective is hidden somewhere in the game, usually a cookbook that contains recipes. Playing these games requires skills as sophisticated as dropping unnecessary items after using them due to limited inventory capacity.



(a) Trained/validated on 100/20 tw-simple games.



(b) Tained/validated on 100/20 custom games.



(c) Trained/validated on 4440/222 tw-cooking games.

Figure 4: Results

Results The agent still outperforms random baseline, but not in a significant way. This is understandable because the architecture we used too simple to learn complicated games such as the ones present in the competition. See Fig 4.

6 Discussion and Future Work

In this paper we explored the applications of reinforcement learning to text-based games. Our proposed agent has a simple architecture and uses A2C algorithm to learn while training. We compared it's performance to a random agent that samples actions randomly from the list of admissible commands, as well as humans. Our agent has a much better performance compared to the random baseline on simple games with dense rewards and detailed goal. For tw-cooking games it's performance isn't different from the random agent in any meaningful way. Here are some possible ways to further enhance our model in the future:

Transfer Learning We observed that the most obvious (and easiest) way to transfer knowledge from pretrained models to our agent didn't really work. Using BERT to get slightly better contextual word embeddings for a sentence such as `"eat banana"` isn't very clever. These kinds of sentences occur frequently in admissible commands or short observations at different stages of the game. BERT will be more useful whenever there is more context. For instance, consider ”I went to the bank to withdraw money, so that I can buy sandwich with it and eat it at the river bank”. It would be useful to have two separate embeddings for the two different word senses of *bank* in this example, which will lead to improved performance on downstream tasks, but TextWorld is simply too small for these effects to be important. Hence, using BERT as a replacement for word embedding is overkill.

Affordance Extraction Another path of exploration could be to use BERT for more appropriate purposes such as ”extracting affordances” (Fulda et al., 2017; McGrenere, 2000). If a sentence says there is a locked door and a key in the drawer, a natural command would be to take the key and unlock the door with it. The affordance here is the ability to unlock doors with keys. We anticipate BERT will prove much more useful in this approach, as it's main training objective was to predict masked words in documents (i.e.. denoising loss). After extracting these affordances, they can then be used for action pruning (Zahavy et al., 2018; Haroush et al., 2018).

Imitation Learning A central challenge in reinforcement learning is dealing with sparsity of rewards. TextWorld competition games are no exception. Often the reward signal is too weak to provide sufficient information when training agents with deep neural networks. Having some sort of supervision is always helpful. This could be in the form of pretraining some parts of the model on a task with labeled data (e.g. question answering), or to learn from an expert as in imitation learning (Ross et al., 2010).

Better Architecture and Memory Specialized agent memory is crucial for success in text-based games. In our current implementation all the information about history, trajectory and current observation is encoded into a d-dimensional vector which understandably makes backtracking extremely difficult for our agent. Our current model can be further enhanced by adding attention mechanisms (Vaswani et al., 2017) or other architectural innovations.

Meta-Learning Meta-learning is a line of research concerned with designing models that can learn new skills or adapt to new environments rapidly with a few training examples (Finn et al., 2017; Nichol and Schulman, 2018). It helps learn better initializations for the model whenever there is a distribution of tasks, and we would like to train agents that learn fast and performs well on previously unseen tasks. TextWorld games follow the same theme but the agent needs slightly different skills to succeed in each game. This makes it a perfect testing ground for meta-learning.

Acknowledgments

We would like to thank Hal and Amr for their help and guidance. We are also grateful for all the blog posts and code implementations we found online that helped save us some time.

References

Prithviraj Ammanabrolu and Mark O. Riedl. 2018. Playing text-adventure games with graph-based deep reinforcement learning. In *NAACL-HLT*.

Guntis Barzdins, Didzis Gosko, Paulis F. Barzdins, Uldis Lavrinovics, Gints Bernans, and Edgars Celms. 2019. Rdf* graph database as interlingua for the textworld challenge. *2019 IEEE Conference on Games (CoG)*, pages 1–2.

Richard Bellman. 1954. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, 60(6):503–515.

Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S. Sukhatme. 2017. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33:1273–1291.

Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A survey on dialogue systems: Recent advances and new frontiers. *ArXiv*, abs/1711.01731.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew J. Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. In *CGW@IJCAI*.

Mark A. DePristo and Robert Zubek. 2001. being-in-the-world. *AAAI*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Nancy Fulda, Daniel Ricks, Ben Murdoch, and David Wingate. 2017. What can you do with a rock? affordance extraction via word embeddings. In *IJCAI*.

Thomas R. Gruber. 1993. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43:907–928.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290.

Matan Haroush, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. 2018. Learning how not to act in text-based games. In *ICLR*.

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. In *ACL*.

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

Joanna McGrenere. 2000. Affordances: Clarifying and evolving a concept.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. *ArXiv*, abs/1506.08941.

Alex Nichol and John Schulman. 2018. Reptile: a scalable metalearning algorithm.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2010. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, second edition. The MIT Press.

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

Christopher J.C.H. Watkins and Peter Dayan. 1992. Technical note: Q-learning. *Machine Learning*, 8:279–292.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *NeurIPS*.