Deep Reinforcement Learning Algorithms for Playing Text-based Games from Feedback

Vishal Vijay Kumar Jain

Master of Science

School of Computer Science
McGill University
Montreal, Quebec, Canada

December 2019

A thesis submitted to McGill University in partial
fulfillment of the requirements of the degree of
Master of Science

Abstract

Text-based games are an interesting domain at the intersection of natural language processing and reinforcement learning. From reinforcement learning perspective, they pose various challenges: combinatorial state and action space, sparse rewards and partial observability, i.e., the agent is informed of the consequences of its actions through textual feedback. Emphasising the later point, we design deep reinforcement learning algorithms which learn from feedback alone. We take advantage of the structural characteristics common to these games and propose two algorithmic improvements in applying Deep RL to play these games. We first propose a contextualisation mechanism, based on accumulated reward, which simplifies the learning problem and mitigates partial observability. We then study different methods that rely on the notion that most actions are ineffectual in any given situation, following Zahavy et al.'s idea of an admissible action. We evaluate these techniques in a series of text-based games of increasing difficulty based on the TextWorld framework, as well as the iconic game ZORK. Empirically, we find that these techniques improve the performance of a baseline deep reinforcement learning agent applied to text-based games.

Abrégé

Les jeux basés sur du texte sont un domaine intéressant à l'intersection du traitement du langage naturel et de l'apprentissage par renforcement. Du point de vue de l'apprentissage par renforcement, ils posent divers défis: état combinatoire et espace d'action, récompenses clairsemées et observabilité partielle, c'est-à-dire que l'agent est informé des conséquences de ses actions par un retour textuel. En mettant l'accent sur le dernier point, nous concevons des algorithmes d'apprentissage par renforcement profond qui apprennent uniquement à partir des commentaires. Nous profitons des caractéristiques structurelles communes à ces jeux et proposons deux améliorations algorithmiques dans l'application de Deep RL pour jouer à ces jeux. Nous proposons d'abord un mécanisme de contextualisation, basé sur la récompense accumulée, qui simplifie le problème d'apprentissage et atténue l'observabilité partielle. Nous étudions ensuite différentes méthodes qui reposent sur la notion que la plupart des actions sont inefficaces dans une situation donnée, en suivant l'idée de Zahavy et al. D'une action admissible. Nous évaluons ces techniques dans une série de jeux textuels de difficulté croissante basés sur le framework TextWorld, ainsi que le jeu iconique ZORK. Empiriquement, nous constatons que ces techniques améliorent les performances d'un agent d'apprentissage de renforcement profond de base appliqué aux jeux basés sur du texte.

Contribution of authors

- Chapter 1 originates from Introduction in our peer-reviewed conference paper [JFL+20]. It has been rewritten in this thesis to add new insights.
- Chapters 2 and 3 are based on new material written for this thesis except the sections on contextual setting and admissibility. These sections have appeared in peer-reviewed conference paper [JFL+20].
- In Chapter 4, the section discussing the algorithmic improvements i.e. "More Efficient Learning for IF Domains" and the "Related Work" section have appeared in peer-reviewed conference paper [JFL+20]. Rest of the chapter is new material.
- In Chapter 5, the section discussing the experiments and results i.e. "Empirical Analysis" has appeared in peer-reviewed conference paper [JFL+20]. This section has been lightly modified to include more figures and analysis. The section introducing Synthetic IF Benchmark originates from similar section in peer-reviewed conference paper [JFL+20]. It has been rewritten and extended to add new insights.

It should be noted that the work done for my Master's project has been accepted for oral presentation at AAAI'20 [JFL+20]. For this paper [JFL+20], I took part in the design of the algorithm, implemented it, performed the experiments, and wrote parts of the paper.

Acknowledgements

Thank you to my parents for their constant encouragement.

Research is a difficult endeavour. I am eternally grateful to my advisors Marc G. Bellemare and Doina Precup for guiding me through the highs and the lows when one strives to push boundaries of science.

Finally, thank you to the TextWorld Team at Microsoft Research, Montreal for many fruitful discussions and insights.

Contents

List of Figures

List of Tables

1

Introduction

In a text-based game, also called *interactive fiction* (IF), an agent interacts with its environment through a natural language interface. Actions consist of short textual commands, while observations are paragraphs describing the outcome of these actions. Playing a text-based is akin to having a dialog. In real life conversations of an agent with an end user, the agent needs to understand the end user's feedback as well as generate/choose a command which seeks to accomplish some goal (in this case, it might be to satisfy the customer). In such a situation, the agent needs to develop natural language understanding as well as take the right decisions. Although a text-based game is a more controlled environment, it embodies the challenges of a real life dialog, hence it has emerged as an important challenge for AI techniques in the sphere of natural language understanding and sequential decision making.

In this work, we view the task of creating an agent for playing these games from a reinforcement learning lens. While playing these games, the agent faces the challenges of partial observability, large state and action space and sparse rewards. Since these games are partially observable, the agent's choice of action at every timestep during the gameplay is dependent on history. To deal with this, Yuan et al. [YCS+18] provides agent with memory in form of LSTM. In fact, even before learning an optimal strategy, the agent needs to deal with huge action space since every sentence is a command. However, most of the actions are inadmissible, i.e., they have no effect in a given state. Using this notion of admissibility, AE-DQN [ZHM+18] dealt with large action space by eliminating actions. Similarly,

the state space is typically combinatorial in nature, due to the presence of objects and characters with which the player can interact. Moreover, these games are also Montezuma-Revengesque [BNVB13, Ope18], i.e., they exhibit reward sparsity. Usually the number of steps required to get a reward are anywhere between 10-20. So, the agent needs to employ better exploration strategy with these games.

Although the previous works make progress in dealing with some of the challenges, they simplify the task by creating a better state representation: using combination of LOOK and INVENTORY description. We call LOOK and INVENTORY as information gathering action since these actions help in revealing information about the hidden state. We forego using these information gathering actions by learning directly from the feedback. Learning from feedback alone means that the agent learns from the natural language observations provided by the game. It doesn't use any crutches in the form of walk-through, command templates or LOOK and INVENTORY description provided by the TextWorld [CKY+18]. This makes sense for the games outside the purview of TextWorld since executing information gathering actions costs a game step. In a life-death situation for the agent, incurring this cost may result in it loosing to an adversary.

On a different note, these games also have an interesting reward structure which makes further progress possible. Essentially, a text based game could be broken down into series of subtasks. Each subtask then corresponds to a different phase in the game. In this work, we use *score*, i.e., accumulated reward as a proxy for progress in the game. Once the agent receives this additional context, the issue of credit assignment becomes easier as the agent could effectively learn separate value function for different phases of the game. This ultimately results in faster game completion.

We now summarise three key contributions of this work:

- *score contextualisation* architecture which uses this special reward structure and embeds accumulated reward (score) as an additional context in the architecture to mitigate further the negative aspects of partial observability and ease credit assignment.
- extending Zahavy et al.'s [ZHM+18] work on action elimination, we introduce gating mechanisms such as *masking, dropout* and *consistent Q-learning* which are simpler in spirit and can be learned from feedback alone.

- a synthetic benchmark comprising of levels of increasing difficulty is introduced to test the effectiveness of the described algorithmic techniques.

Previous works ([ZHM+18, YCS+18]), have either tackled the issue of partial observability or large action space. With this work, we bring forth two algorithmic improvements in applying Deep Reinforcement Learning techniques to IF and tackle the challenges of partial observability and large action space simultaneously. Furthermore, the synthetic benchmark provides us with graded difficulty to check the effectiveness of the proposed techniques. It should be noted that this work has been accepted for oral presentation at AAAI'20 [JFL+20].

Finally, the thesis is organised as follows:

- chapter 2 describes the history, gameplay and challenges concerning the text-based games.
- chapter 3 delves into the background necessary to understand this work.
- chapter 4 discusses the algorithmic techniques, score contextualisation and gating mechanism and in the end an algorithm for learning in these domain is proposed.
- chapter 5 describes the various ablative studies to test the effectiveness of the proposed algorithmic techniques.
- chapter 6 concludes with future directions and discussion.

2

Text-based games

2.1 History and Now

Before the graphical displays became ubiquitous, text-based games were one of the first games that owe their existence to the computing revolution. The first text-based game "Colossal Cave" was written by Will Crowther in 1976. Soon after that, many titles such as the ZORK series swept the market and are still popular among the IF community. The classical games such as the Zork series and Adventureland focused on solving the game by collecting treasures that were spread across the game map. Modern games have focused on intricate storytelling and creating an enriching experience for the gamer while interacting with the game. These games are known as *interactive fiction* (IF), involve an agent interacting with its environment through a natural language interface; actions consist of short textual commands, while observations come in the form of paragraphs describing the outcome of these actions (Figure 2.1). With such immersive gameplay and the rich storylines created by an active IF community, text-based games have seen a resurgence.

From an AI perspective, IF has emerged as an important challenge [ABC+18], in great part because the genre combines natural language with sequential decision-making.

```
South of House                                    Score: 0        Moves: 4
You are standing in an open field west of a white house, with a boarded front
door.
There is a small mailbox here.

>open mailbox
Opening the small mailbox reveals a leaflet.

>take leaflet
Taken.

>read leaflet
"WELCOME TO ZORK!

ZORK is a game of adventure, danger, and low cunning. In it you will explore
some of the most amazing territory ever seen by mortals. No computer should be
without one!"


>s
South of House
You are facing the south side of a white house. There is no door here, and all
the windows are boarded.

>_
```

Figure 2.1: ZORK1 gameplay detailing interaction between the agent and the environment.

2.2 Gameplay

Text-based games can be classified on the basis of how player interacts with the game: in *parser-based* games, the player inputs the command by typing it character by character; in *choice-based* games, the player needs to select one of the commands from the given options; in *hypertext-based* games, the player interacts by clicking on one of the several links in the description. In this work, we deal with parser-based games.

(a) Parser-based (b) Choiced-based (c) Hypertext-based

Figure 2.2: Different genres.

In text-based games, when a player interacts with the game, it receives feedback in the form of text description and the next game score. This feedback is contingent on the

command that a player issues to the game engine and the state of the game at that time. For instance in Fig 2.1, `read leaflet` produces sensible feedback only when the player has leaflet in his possession. In this case, `take leaflet` should be issued before issuing `read leaflet` command. During the gameplay, the player gets a chance to explore the game *map* which consists of different rooms and objects. These rooms are discrete in-game locations. To reach one of these locations, player issues navigation commands `go` followed by `{north,east,...}` or orthogonal directions `{up,down}`. Once the player reaches a room, he receives the room description as the feedback. As in Fig 2.1, once the player moves `south`, it receives the description of the location `South of House`. Alternatively, if the player forgets his current location, these games provide an information gathering command called LOOK to get the current location description. Navigation-wise text-based games could be quirky: room exits and entrances do not always match (e.g. `go east` and then `go west` command may not result in the player coming back to the starting point).

Apart from the navigation commands, the player can issue commands such as `read leaflet`. These type of commands helps the player explore different objects he finds in the game. Text-based games involve extensive use of such interactive commands to explore the game state space further making these games AI-hard [Mue87]. In Fig 2.1 without issuing `open mailbox` command first, the player wouldn't know that there is a leaflet that could be read. However, not all commands issued are understood by the parser or are valid forms of interaction. For instance, if the player finds mailbox in the current location description, `open mailbox` is a valid command whereas `eat mailbox` might be invalid. These behaviors that are valid given a situation are called *affordances* [Gib77].

2.3 Text-based Game as Reinforcement Learning Domain

With the gameplay described in the previous section, a text-based game lends itself to being treated as a sequential decision-making problem. Since the games are partially observable, the feedback received by the player doesn't describe the state of the environment. The environment state in a text-based game contains the information about each room, every treasure, player health, and non-player characters if present. The action that the player takes is a sentence. Once this action is executed by the game, the player receives the next feedback

and the score. The reward received can then be easily calculated as a score differential.

2.4 Challenges

From a reinforcement learning perspective, IF domains pose a number of challenges. In this section, we list all of them.

2.4.1 Partial Observability

Text-based games are partially observable. Only information pertaining to the current in-game location and the player's inventory is made available by the information gathering actions. For instance, the player might issue a command `press red button` and the feedback is `You press red button`. However, in some other location, there is a chest that is opened now; the player has no way to know whether the command was useful or not as the immediate feedback or the description of the current location does not contain information about the change in the other game location. Moreover, to play these games skillfully, the player can not just rely on the latest observation to issue the next command; it needs to take into account the previous feedbacks received and commands issued. For example, the commands `take apple` and `take red key` would give the same feedback `Taken` and relying on this immediate feedback alone wouldn't be a great strategy. To issue the next command, the player would also need to consider the previous command (in this case `take red key` or `take apple`).

2.4.2 Combinatorial State and Action Space

In classic text-based game ZORK, there are 85 rooms and 20 treasures. A room here is an in-game location. These treasures can easily be transported from one location to another. So, every room in ZORK could have more than one of these treasures. Thus, the size of the state space in ZORK $> 20^{85}$ since a treasure could be found in any one of the rooms in the game. The state space is therefore combinatorial in nature which makes applying tabular reinforcement learning for learning optimal policies an infeasible proposition. In a similar vein, every sentence can be issued as a valid command, which makes the action space combinatorial as well. However, not all actions are valid given the current situation.

The actions which actually result in a change in the underlying hidden state are called *admissible* actions [ZHM+18]. Usually, the number of such admissible actions is smaller than the combinatorial action set. The challenge of determining admissible actions at run-time is unique to text-based games as most of the reinforcement learning domains such as Arcade Learning Environment are *strongly operationalized*, i.e., the set of action which leads to state change is known in advance[1] [ABC+18]. Text-based games test the *epistemic feasibility* of the agent, i.e., can the agent efficiently derive that a particular action even exists in a current situation [McC77]? So, it is not just the combinatorial nature of the state and action space that an agent has to deal with, it has to develop reasoning to choose admissible actions and progress further in the game.

2.4.3 Reward Sparsity

The credit assignment problem concerns determining how the success of a system's overall performance is due to the various contributions of the system's components [Min61, SB98]. In an environment where the rewards are sparse, accurately determining the contributions of the various state-action pairs in getting that elusive reward is even more difficult. For most of the text-based games, their reward structure is usually sparse, with non-zero rewards only received when the agent accomplishes something meaningful, such as retrieving an important object or unlocking a new part of the domain. With sparse rewards, the agent indulges in random exploration until it receives the reward. This exercise is costly in terms of time and transitions sampled.

Recent works have used the concept of intrinsic motivation resulting in a more directed and sensible exploration strategy [BOG+16, PAED17]. Text-based games being partially observable exacerbate the issue of reward sparsity because even after receiving the sparse reward, the agent needs to take history into consideration to determine the right state to assign the credit. This exercise is inherently difficult since the action chains might contain irrelevant actions due to random exploration.

[1] ALE 0.6 provides a large action set and a minimal action set for each game. In practice, minimal action set is used since every action in the set leads to change in game state.

2.5 Contrasting with ALE

Bellemare et al. [BNVB13] introduced Arcade Learning Environment (ALE) as a Reinforcement learning testbed. Narsinham et al. [NKB15] introduced the first Deep Reinforcement Learning agent for a text-based game. We list points contrasting ALE with text-based games domain:

- Output by ALE games is visual in nature whereas, for text-based games, it is textual. This results in different architectural choices.
- ALE originally didn't have stochasticity; the recent version introduces stochasticity using sticky actions [MBT+18]. On the other hand, stochasticity in a text-based game comes naturally due to the presence of non-player characters (NPC). But, even NPCs could be deterministic implying that a text-based game could be deterministic as well.
- For text-based games, the effectual action set is not known in advance. In the case of ALE, the research community uses minimal action set which happens to be an effectual action set. Effectual action set here means that every action in the set results in state change when executed.

3

Background

In this work, we would like to design an agent that deals with the sequential decision making as well as the natural language understanding aspect involved in solving the text-based games. To this end, we firstly introduce concepts related to reinforcement learning which will lay the foundation for tackling the decision making aspect of the problem. In the later parts of the chapter, we present concepts necessary in creating good representation for natural language understanding. Parts of this chapter that introduce contextual setting and admissibility have appeared in peer-reviewed conference paper [JFL+20].

3.1 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning where the agent interacts with the environment to maximize its cumulative reward [SB98]. In a supervised learning paradigm, input-output pairs are provided which helps the agent to learn an accurate mapping from the input space to the output space, whereas in RL there is no training data provided beforehand; the agent needs to collect this data via exploration. This results in the exploration v/s exploitation trade-off. The exploration-exploitation trade-off means that the agent needs to balance exploring the uncharted parts of the environment to collect more information and exploitation of the acquired information thereof.

In RL setting, the agent interacts with the environment in discrete time steps. At a time step t, the agent receives the information from the environment s_t and the reward signal r_t.

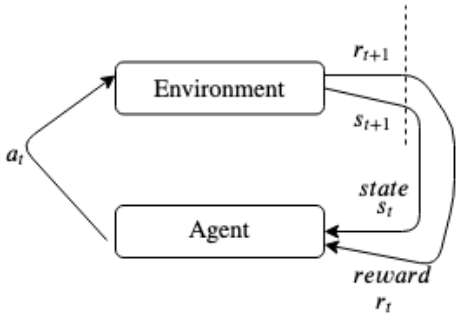

Figure 3.1: RL environment diagram

Based on this information, the agent executes action a_t and the environment responds with the next state s_{t+1} and the reward signal r_{t+1} (see Fig. 3.1). The material presented in this section and the subsequent ones concerning RL only address the parts which are relevant for this study. For a more thorough and rigorous treatment of RL, refer these textbooks [SB98, Sze10].

3.2 Markov Decision Process

To mathematically formalize the agent-environment interaction, Markov Decision Process provides us with an excellent toolkit [Bel56]. A Markov Decision Process (MDP) M is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where

- $\mathcal{S}$ is the set of states.
- $\mathcal{A}(s)$ is the set of actions available for state $s \in \mathcal{S}$.
- $\mathcal{P}(s, s', a)$ is the transition probability function from s to s' given action a is taken.
- $\mathcal{R}(s, s', a)$ is the immediate reward received when the agent transitions from s to s' when action a is taken.
- $\gamma \in [0, 1)$ is the discount factor determining the agent's preference for future or immediate rewards. A low γ implies that the agent values immediate rewards more.

Interestingly, the transition probability $\mathcal{P}$ follows the *Markov property*, i.e.,

$$\mathcal{P}(s_t, s_{t-1}, a_{t-1}) = Pr(s_t|s_{t-1}, a_{t-1}) = Pr(s_t|s_{t-1}, a_{t-1}, s_{t-2}, \ldots, s_0, a_0)$$

Essentially, in a Markovian environment, the agent's next state depends on its current state and the action regardless of past history. Although the Markovian assumption reduces the memory load of the agent, in a text-based game setting this assumption is invalid as the game is partially observable (see Section 2.4). In this case, we need a mathematical formalism that encapsulates the notion of partial observability (See Section 2.4.1). We will describe this formalism in the next section.

3.3 Text-based Games as POMDPs

As detailed in Section 2.4.1, partial observability is one of the key challenges in solving text-based games. We formalise text-based games as a partially observable Markov decision process (POMDP) $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{O}, \phi, \gamma)$ [KLC98], where

- $\mathcal{S}$ is a set of environment states. At time t, in a text-based game, the environment state s_t contains all the necessary information about all the rooms and the objects and the player health.
- $\mathcal{A}$ is a set of available actions. At time t, the player executes action a_t.
- $\mathcal{P}(s, s', a)$ is the transition probability from s to $s^{'}$ given action a is taken.
- $\mathcal{R}(s, s', a)$ is the immediate reward received when the agent transitions from s to $s^{'}$ when action a is taken.
- $\mathcal{O}$ is the set of observations available. In text-based games, an observation at time t, o_t is the feedback that is received by the agent.
- $\phi : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{O}$ describes the observation $o = \phi(s, a)$ when action a is taken in state s.
- $\gamma \in [0, 1)$ is the discount factor.

3.4 Policies and Value functions

In reinforcement learning, the agent needs to estimate how *useful* it is to be in the current state. We call this utility a state's *value*. In a similar vein, we define a state-action's *value* as the utility of taking an action in the current state. Moreover, we define $\pi : \mathcal{S} \rightarrow \mathcal{A}$ as a

mapping from states to the action. Generally, in RL, the policy is stochastic, i.e., the policy is formulated as conditional probability distribution $\pi(a|s)$ over the action set $\mathcal{A}$. Since π drives the agent's interaction with the environment and essentially determines the future reward the agent receives, the agent's state value (or state action value) is dependent on π.

When the agent follows the policy π, the state-value function V^π is defined as

$$V^\pi(s) := \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \big| s\big]$$

where r_t corresponds to $\mathcal{R}(s_t, s_{t+1}, a_t)$.

Similarly, the action value-function $Q^\pi(s, a)$ is defined as

$$Q^\pi(s, a) := \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \big| s, a\big]$$

The main aim of an agent is to find a policy that achieves a maximum value of V^π or Q^π. To do so, we need to define a partial ordering over the policies. We consider policy π to be dominated by π' when $V^{\pi'}(s) \geqslant V^\pi(s)$ for all the states s in the state space $\mathcal{S}$. We term policy π^* to be optimal when $\pi^* > \pi$ for all the policies $\pi \in \Pi$. Here Π is the set of stationary policies. For such a policy, we define the optimal value function V^* as

$$V^*(s) := \max_\pi V^\pi(s)$$

Similarly, the optimal action value function can be defined as

$$Q^*(s, a) := \max_\pi Q^\pi(s, a)$$

3.5 Q-Learning

To solve an MDP, the agent needs to either learn an optimal policy or an optimal value function. The agent could learn the concerned policy or value function either on-policy, i.e., the agent learns the same policy while using it to make decisions. Whereas in the case

of off-policy learning, the agent utilizes the experiences from behavior policy to improve the value function estimates for another policy.

Q-Learning [Wat89] was one of the earliest breakthroughs in off-policy learning and it is used to estimate the optimal value function. In the case of Q-Learning, the behavior policy is usually an ϵ-greedy policy and the policy being optimized is the greedy policy. An ϵ-greedy policy is a policy that chooses an action uniformly at random with probability ϵ and with $1-\epsilon$ probability chooses the action with the highest value function. When $\epsilon = 0$, we get the greedy policy.

Specifically, Q-Learning iteratively learns the estimates of state-action value (also called Q-values) by updating it towards the observed reward and the max Q-value over all actions in the resulting state

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t \tag{3.1}$$

where $\delta_t = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$ is the TD error [SB98] and $\alpha \in [0, 1)$.

3.6 Consistent Q-Learning

In the Q-Learning update rule (equation 3.1), the use of the max operator to determine the value of the next state can cause large overestimations of the action values. Hasselt et. al. [Has10] show that Q-learning can suffer a large performance penalty because of a positive bias that results from using the maximum value as an approximation for the maximum expected value. Consistent Q-Learning [BOG+16] is one way to deal with such overestimations and get rid of the positive bias. It learns a value function which is consistent with respect to a local form of policy stationarity. Defined for a Markov decision process, it replaces the term δ_t in equation 3.1 by

$$\delta_t^{\text{CQL}} := r_t + \begin{cases} \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t) & s_{t+1} \neq s_t \\ (\gamma - 1) Q(s_t, a_t) & s_{t+1} = s_t. \end{cases} \tag{3.2}$$

Consistent Q-learning can be shown to decrease the action-value of suboptimal actions while maintaining the action-value of the optimal action, leading to larger *action gaps* and a potentially easier value estimation problem.

3.7 Contextual Setting

At time step t, the agent selects an action according to a policy π which maps a *history* $h_t := o_1, a_1, \ldots, o_t$ to a distribution over actions, denoted $\pi(\cdot \mid h_t)$. This history is a sequence of observations and actions which, from the agent's perspective, replaces the unobserved environment state s_t. We denote by $B(s \mid h_t)$ the probability or *belief* of being in state s after observing h_t. Finally, we will find it convenient to rely on time indices to indicate the relationship between a history h_t and its successor, and denote by h_{t+1} the history resulting from taking action a_t in h_t and observing o_{t+1} as emitted by the hidden state s_{t+1}.

The action-value function Q^π describes the expected discounted sum of rewards when choosing action a after observing history h_t, and subsequently following policy π:

$$Q^\pi(h_t, a) = \mathbb{E}\big[\sum_{i \geqslant 0} \gamma^i r(s_{t+i}, a_{t+i}) \mid h_t, a\big],$$

where we assume that the action at time $t+j$ is drawn from $\pi(\cdot \mid h_{t+j})$; note that the reward depends on the sequence of hidden states $s_{t+1}, s_{t+2}, \ldots$ implied by the belief state $B(\cdot \mid h_t)$. The action-value function satisfies the Bellman equation over histories

$$Q^\pi(h_t, a) = \mathop{\mathbb{E}}_{s_t, s_{t+1}} \big[r(s_t, a) + \gamma \max_{a' \in \mathcal{A}} Q^\pi(h_{t+1}, a')\big].$$

When the state is observed at each step ($\mathcal{O} = \mathcal{S}$), this simplifies to the usual Bellman equation for Markov decision processes:

$$Q^\pi(s_t, a) = r(s_t, a) + \gamma \mathop{\mathbb{E}}_{s_{t+1} \sim P} \max_{a' \in \mathcal{A}} Q^\pi(s_{t+1}, a'). \tag{3.3}$$

In this fully observable case we will conflate s_t and h_t in the notation.

The Q-learning algorithm [Wat89] over histories maintains an approximate action-value function Q which is updated from samples h_t, a_t, r_t, o_{t+1} using a step-size parameter $\alpha \in$

$[0, 1)$:

$$Q(h_t, a_t) \leftarrow Q(h_t, a_t) + \alpha\delta_t$$
$$\delta_t = r_t + \gamma \max_{a \in \mathcal{A}} Q(h_{t+1}, a) - Q(h_t, a_t). \tag{3.4}$$

Q-learning is used to estimate the *optimal action-value function* attained by a policy which maximizes Q^π for all histories. In the context of our work, we will assume that this policy exists. Storing this action-value function in a lookup table is impractical, as there are in general an exponential number of histories to consider. Instead, we will make use of recurrent neural networks to maintain the approximation learned by Q-learning.

It should be noted that consistent Q-learning (see Section 3.6) is not immediately adaptable to the history-based formulation, since h_{t+1} and h_t are sequences of different lengths (and therefore not comparable). In the following sections, we derive a related algorithm suited to the history-based setting.

3.8 Admissibility

We will make use of the notion of an *admissible action*, following terminology by Zahavy et al. [ZHM+18].[1]

Definition 1 *An action a is* admissible in state s *if*

$$P(s \mid s, a) < 1.$$

That is, a is admissible in s if its application may *result in a change in the environment state. When $P(s \mid s, a) = 1$, we say that an action is* inadmissible.

We extend the notion of admissibility to histories as follows. We say that an action a is admissible given a history h if it is admissible in *some* state that is possible given h, or equivalently:

$$\sum_{s \in \mathcal{S}} B(s \mid h) P(s \mid s, a) < 1.$$

[1] Note that our definition technically differs from Zahavy et al. [ZHM+18]'s, who define an admissible action as one that is not ruled out by the learning algorithm.

We denote by $\xi(s) \subseteq \mathcal{A}$ the set of admissible actions in state s. With some abuse of notation, we define the *admissibility function*

$$\xi(s,a) := \mathbb{I}_{[a \in \xi(s)]}$$
$$\xi(h,a) := \Pr\{a \in \xi(S)\}, S \sim B(\cdot \mid h).$$

We write $\mathcal{A}_t$ for the set of admissible actions given history h_t, i.e., the actions whose admissibility in h_t is strictly greater than zero. In IF domains, inadmissible actions are usually dominated, and we will deprioritize or altogether rule them out based on our estimate of $\xi(h,a)$.

3.9 Function Approximation

For simpler MDPs with finite state space and action space, it is assumed that values are stored in table. However, once the state space becomes exponentially large wrt input features/dimensions, we encounter the curse of dimensionality [Bel56]. To deal with this phenomenon, we need to approximate the value functions by a function approximator. This function approximator could be a linear function, a neural network or any other function.

When using a function approximator, $Q(s_t, a_t) \approx Q(s_t, a_t; \theta_t)$ where θ_t are the parameters corresponding to the function approximator.

3.10 Neural Network

In this work, we use a Neural network as the function approximator. A Neural Network consists of a multiple layers of interconnected nodes. The initial layer receives the input and the output layer has output signals to which input might map. The middle layers are called hidden layers. These layers are responsible for capturing the inherent pattern in the data so that error is minimal. Each layer also applies a non-linear activation function ψ to the output value produced by the nodes. For a node with a weight vector $\mathbf{w}$ and a bias b, the output $f(x)$ is defined as

$$f(x) := \psi(\mathbf{w} \cdot \mathbf{x} + b)$$

In theory, it has been shown a Neural Network with even one hidden layer is capable of classifying input which is not linearly separable. Linear separability here means that the classes with the input of dimension n can be separated by a hyperplane of dimension $n-1$. Additionally, a neural network with only one hidden layer has been shown to be a universal function approximator [Hor91].

3.11 Gradient Descent

Learning a neural network means that there is a loss function that needs to be minimized. To do so, an input vector $\mathbf{x}$ and a desirable output vector $\mathbf{y}$ is provided. The neural network then uses an optimization algorithm to minimize the loss between the output of the neural network and $\mathbf{y}$.

One of these optimization algorithms is Backpropagation [Hec89, RHW+88]. Backpropagation, as the name suggests, computes the error in the output layer and propagates it back through the rest of the network. Specifically, it calculates the gradient of the cost function with respect to the outer layer. The gradients of the inner layer are computed via the chain rule from the gradient values of the outer layer.

To update the weights using backpropagated errors, we can then use stochastic gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\Delta$$

where Δ is the computed gradient and α is the learning rate.
Alternatively, we could use Adam [KB15] or RMSprop [TH12] which vary learning rates depending on changing gradients and work better in practice.

3.12 Recurrent Neural Network

A class of neural network that has been successfully used for language tasks is the class of Recurrent Neural Network (RNN). In RNN, as the name suggests, there are additional recurrent connections in the node apart from the "normal" input. With these recurrent connections, the network is able to exhibit temporal behavior as it uses its hidden state (memory)

to retain past behavior as well as capture long-term dependencies.

Training RNN requires a different type of backpropagation called Backpropagation Through Time (BPTT) [Wer90]. BPTT essentially unfolds the network and propagates the error to the earlier timesteps. In practice though, it is observed that an RNN fails to retain important information for long enough, thus it fails to learn important dependencies when there are too many timesteps in between [Hoc91, BSF94].

To tackle the issue of not capturing long-term dependencies, LSTM was introduced [HS97]. The authors proposed the concept of gates in the cell to regulate the flow of information. There are three gates: input gate, output gate and forget gate; together these gates with their thresholding mechanism control the information flow thus easing the issue of not retaining long-term dependencies. For a more detailed introduction, see Chris Olah's blog [Ola15].

3.13 Word Embedding

To effectively use high dimensional complex data such as text as the input in the neural network, the text data needs to be converted into a real vector. A Word embedding takes a word or phrase from the vocabulary and maps it to a real vector. Several methods have been proposed to generate this mapping such as using neural networks [MSC+13], dimensionality reduction of window-based co-occurrence matrix [LC14], probabilistic models [GCPT07] to name the few.

In this work, we use a separate neural network layer that learns these embeddings. With a huge vocabulary, we desire to learn a mapping in such a way that words having similar connotations, their real vectors are close enough. For this work, we learn the embeddings from scratch. Learning from scratch means that we don't use pre-trained word embeddings such as Word2Vec [MSC+13], GloVe [PSM14] or others. Our neural net architecture which includes the embedding layer is initialized randomly. The weights of the embedding layer along with the rest of the architecture are then learned end to end from the losses.

4

Algorithmic Contributions

In this chapter, we would like to design an agent that learns to play a text-based game from feedback alone. Fortunately, there are three structural aspects of this domain that makes progress possible:

- **Rewards from Subtasks.** The optimal behaviour completes a series of subtasks towards the eventual game end;
- **Transition Structure.** Most actions have no effect in a given state;
- **Memory as State.** Remembering key past events is often sufficient to deal with partial observability.

Although previous works [NKB15, ZHM+18] have remarked on these properties, we forego previously made assumptions and provide fresh tools to more tractably solve IF domains. More generally, we believe these tools to be useful in partially observable domains with a similar structure. In the following section, we propose two algorithmic improvements in applying Deep Reinforcement Learning to these games. The section has appeared in a peer-reviewed conference paper [JFL+20]. We then propose an algorithm that applies these techniques while training the agent. We conclude this chapter by discussing the related works and their contributions in progressing the state of art in IF.

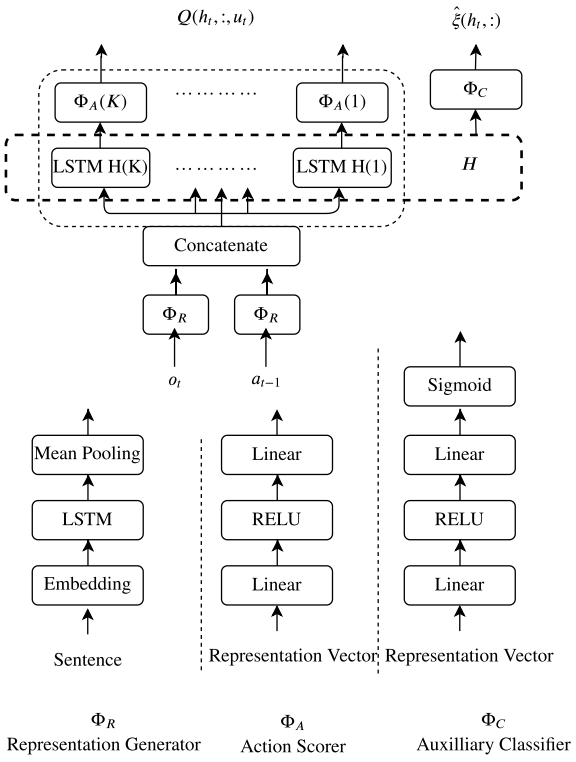

Figure 4.1: Our IF architecture consists of three modules: a representation generator Φ_R that learns an embedding for a sentence, an action scorer Φ_A that chooses a network head i (a feed-forward network) conditional on score u_t, learns its Q-values and outputs $Q(h_t, :, u_t)$ and finally, an auxilliary classifier Φ_C that learns an approximate admissibility function $\hat{\xi}(h_t, :)$. The architecture is trained end-to-end.

4.1 More Efficient Learning for IF Domains

We are interested in learning an action-value function that is close to optimal and from which can be derived a near-optimal policy. We would also like learning to proceed in a sample-efficient manner. In the context of IF domains, this is hindered by both the partially observable nature of the environment and the size of the action space. In this section, we propose two complementary ideas that alleviate some of the issues caused by partial observability and large action sets. The first idea contextualizes the action-value function on a surrogate notion of progress based on total reward so far, while the second seeks to eliminate inadmissible actions from the exploration and learning process.

Although our ideas are broadly applicable, for concreteness we describe their implementation in a deep reinforcement learning framework. Our agent architecture (Figure 4.1) is derived from the LSTM-DRQN agent [YCS+18] and the work of Narsimhan et al. [NKB15].

4.1.1 Score Contextualisation

When games are used as a test-bed for applying reinforcement learning algorithms, it is now customary to translate the player's score differential into rewards [BNVB13, Ope18]. Our setting is similar to Arcade Learning Environment in the sense that the environment provides the score. In IF, points are awarded when the player acquires an important object or completes a relevant subtask in the game. Usually, these awards are a good enough proxy for player's progression in the game and occur in a linear or almost linear structure. For text-based games, rewards are relatively sparse. We emphasize that this is in contrast to the more general reinforcement learning setting, which may provide a reward for surviving or achieving something at a certain rate. In the video game SPACE INVADERS, for example, the notion of "finishing the game" is ill-defined: the player's objective is to keep increasing their score until they run out of lives.

We make use of the IF reward structure as follows. We call *score* the agent's total (undiscounted) reward since the beginning of an episode, remarking that the term extends beyond game-like domains. At time step t, the score u_t is

$$u_t := \sum_{i=0}^{t-1} r_i.$$

Since the score denotes the agent's progression in a text-based game, using it as a state variable is a sensible choice. We now propose an approach called *score contextualisation*. In this approach, a separate action-value function for each possible score is maintained. This action-value function is denoted $Q(h_t, a_t, u_t)$. The use of additional context variables has by now been demonstrated in a number of settings ([RZQ+19]; [IKVM18]; [GSR+18]). First, credit assignment becomes easier since the score provides clues as to the hidden state. Second, in settings with function approximation we expect optimization to be simpler since

for each u, the function $Q(\cdot, \cdot, u)$ needs only be trained on a subset of the data and hence can focus on features relevant to this part of the environment.

In a deep network, we implement score contextualisation using K network heads and a map $\mathcal{J} : \mathbb{N} \to \{1, \ldots, K\}$ such that the $\mathcal{J}(u_t)^{th}$ head is used when the agent has received a score of u_t at time t. This provides the flexibility to either map each score to a separate network head, or multiple scores to one head. Taking $K = 1$ uses one monolithic network for all subtasks, and fully relies on this network to identify state from feedback. In our experiments, we assign scores to network heads using a round-robin scheme with a fixed K. Using Narshimhan et al.'s [NKB15] terminology, our architecture consists of a shared *representation generator* Φ_R with K independent LSTM heads, followed by a feed-forward *action scorer* $\Phi_A(i)$ which outputs the action-values (Figure 4.1).

4.1.2 Action Gating Based on Admissibility

Using the notion of admissibility introduced in Section, we now seek to eliminate or more generally *gate* actions. Consider an action a which is inadmissible in state s. By definition, taking this action does not affect the state. We further assume that inadmissible actions produce a constant level of reward, which we take to be 0 without loss of generality:

$$a \text{ inadmissible in } s \implies r(s, a) = 0.$$

This assumption is reasonable in IF domains, and more generally holds true in domains that exhibit subtask structure, such as the video game MONTEZUMA'S REVENGE [BSO+16]. We can combine knowledge of P and r for inadmissible actions with Bellman's equation to deduce that

$$a \text{ inadmissible in } s \implies Q(s, a) = 0. \tag{4.1}$$

If we know that a is inadmissible, then we also know its action-value without needing to learn it.

We propose learning a classifier whose purpose is to predict the admissibility function. Given a history h, this classifier outputs, for each action a, the probability $\hat{\xi}(h, a)$ that this action is admissible. Because of state aliasing, this probability is in general strictly between

0 and 1; furthermore, it may be inaccurate due to approximation error. We, therefore, consider action gating schemes that are sensitive to intermediate values of $\hat{\xi}(h, a)$. The first two schemes produce an approximately admissible set $\hat{\mathcal{A}}_t$ which varies from time step to time step; the third directly uses the definition of admissibility in a history-based implementation of the consistent Bellman operator.

Dropout. The *dropout* method randomly adds each action a to $\hat{\mathcal{A}}_t$ with probability $\hat{\xi}(h_t, a)$.

Masking. The *masking* method uses an elimination threshold $c \in [0, 1)$. The set $\hat{\mathcal{A}}_t$ contains all actions a whose estimated admissibility is at least c:

$$\hat{\mathcal{A}}_t := \{a : \hat{\xi}(h_t, a) \geqslant c\}.$$

The masking method is a simplified version of Zahavy et.al's [ZHM+18] action elimination algorithm, whose threshold is adaptively determined from a confidence interval, itself derived from assuming a value function and admissibility functions that can be expressed linearly in terms of some feature vector.

In both the dropout and masking methods, we use the action set $\hat{\mathcal{A}}_t$ in lieu of the full action set $\mathcal{A}$ when selecting exploratory actions.

Consistent Q-learning for Histories (CQLH). The third method leaves the action set unchanged but instead drives the action-values of purportedly inadmissible actions to 0. This is done by adapting the consistent Bellman operator (3.2) to the history-based setting. First, we replace the indicator $\mathbb{I}_{[s_{t+1} \neq s_t]}$ by the probability $\hat{\xi}_t := \hat{\xi}(h_t, a_t)$. Second, we drive $Q(s_t, a_t)$ to 0 in the case when we believe the state is unchanged, following the argumentation of (4.1). This yields a version of consistent Q-learning which is adapted to histories, and makes use of the predicted admissibility:

$$\begin{aligned}\delta_t^{\text{CQLH}} := r_t + \gamma \max_{a \in \mathcal{A}} Q(h_{t+1}, a)\hat{\xi}_t \\ + \gamma Q(h_t, a_t)(1 - \hat{\xi}_t) - Q(h_t, a_t).\end{aligned}$$

One may ask whether this method is equivalent to a belief-state average of consistent Q-learning when $\hat{\xi}(h_t, a_t)$ is accurate,, i.e., equals $\xi(h_t, a_t)$. In general, this is not the case: the admissibility of an action depends on the hidden state, which in turn influences the action-value at the next step. As a result, the above method may underestimate action-values when there is state aliasing (e.g., $\hat{\xi}(h_t, a_t) \approx 0.5$), and yields smaller action gaps than the state-based version when $\hat{\xi}(h_t, a_t) = 1$. However, when a_t is known to be inadmissible ($\hat{\xi}(h_t, a_t) = 0$), the methods do coincide, justifying its use as an action gating scheme.

We implement these ideas using an *auxiliary classifier* Φ_C. For each action a, this classifier outputs the estimated probability $\hat{\xi}(h_t, a)$, parametrized as a sigmoid function. These probabilities are learned from bandit feedback: after choosing a from history h_t, the agent receives a binary signal e_t as to whether a was admissible or not. In our setting, learning this classifier is particularly challenging because the agent must predict admissibility solely based on the history h_t. As a point of comparison, using the information-gathering commands LOOK and INVENTORY to establish the state, as proposed by Zahavy et al. [ZHM+18], leads to a simpler learning problem, but one which does not consider the full history. The need to learn $\hat{\xi}(h_t, a)$ from bandit feedback also encourages methods that generalize across histories and textual descriptions.

4.2 Algorithm

4.2.1 Minibatch Sampling

For sampling mini-batch, we use prioritised sampling [MA93] over episodes, i.e., we sample τ_p fraction of episodes that had atleast one positive reward, τ_n fraction with atleast one negative reward and $1-\tau_p-\tau_n$ from whole episodic memory $\mathcal{D}$. For each sampled episode, we further sample transition sequence starting from random step j in the episode [HS15].

4.2.2 Notations

Following are the notations important to understand the algorithm:

- o_t, r_t, e_t : observation (,i.e., feedback), reward and admissibility signal received at

time t.

- a_t : command executed in game-play at time t.
- u_t : cumulative rewared/score at time t.
- Φ_R : representation generator.
- Φ_C : auxiliary classifier.
- K : number of network heads in score contextualisation architecture.
- $\mathcal{J}$: dictionary mapping cumulative rewards to network heads.
- $H(k)$: LSTM corresponding to network head k.
- $\Phi_A(k)$: Action scorer corresponding to network head k.
- h_t : agent's context/history state at time t.
- T: maximum steps for an episode.
- p_i : boolean that determines whether +ve reward was received in episode i.
- q_i : boolean that determines whether -ve reward was received in episode i.
- τ_p : fraction of episodes where $\exists t < T : r_t > 0$
- τ_n : fraction of episodes where $\exists t < T : r_t < 0$
- l : sequence length.
- n : minimum history size for a state to be updated.
- $\mathcal{A}$: action set.
- $\hat{\mathcal{A}}_t$: admissible set generated at time t.
- I_{target} : update interval for target network
- ϵ : parameter for $\epsilon-$greedy exploration strategy.
- ϵ_1 : softness parameter, i.e., ϵ_1 fraction of times $\hat{\mathcal{A}}_t = \mathcal{A}$.
- c : threshold parameter for action elimination strategy Masking.
- G_{max} : maximum steps till which training is performed.

Full training procedure is listed in Algorithm 1.

4.2 Algorithm

Algorithm 1 General training procedure

function ACT($o_t, a_{t-1}, u_t, h_{t-1}, \mathcal{J}, \epsilon, \epsilon_1, c, \theta$)
  Get network head $k = \mathcal{J}(u_t)$.
  $h_t \leftarrow$ LSTM $H(k)[w_t, h_{t-1}]$.
  $Q(h_t, :, u_t; \theta) \leftarrow \Phi_A(k)(h_t); \hat{\xi}(h_t, a; \theta) \leftarrow \Phi_C(h_t)$.
  Generate $\hat{\mathcal{A}}_t$ (see Section 4.1.2).
  With probability ϵ, $a_t \leftarrow$ Uniform($\hat{\mathcal{A}}_t$), else $a_t \leftarrow \text{argmax}_{a \in \hat{\mathcal{A}}_t} Q(h_t, a, u_t; \theta)$
  return a_t, h_t
end function
function TARGETS(f, γ, θ^-)
  $(a_0, o_1, a_1, r_2, u_2, e_2, o_2, \ldots, o_l, a_l, r_{l+1}, e_{l+1}, u_{l+1}) \leftarrow f; h_{b,0} \leftarrow 0$
  Pass transition sequence through H to get $h_{b,1}, h_{b,2}, \ldots, h_{b,l}$
  $E_{b,i} \leftarrow \xi(h_{b,i}, a_i; \theta^-)$
  $y_{b,i} \leftarrow \max_{a \in \mathcal{A}} Q(h_{b,i+1}, a, u_{b,i+1}; \theta^-)$
  $y_{b,i} \leftarrow E_{b,i} y_{b,i} + (1 - E_{b,i})\, Q(h_{b,i+1}, a_i, u_{b,i+1}; \theta^-)$ **if** using CQLH.
  $y_{b,i} \leftarrow r_{i+1}$ **if** o_i is terminal **else** $y_{b,i} \leftarrow r_{i+1} + \gamma y_{b,i}$
  return $y_{b,:}, E_{b,:}$
end function

Input: $G_{\text{max}}, I_{\text{look}}, I_{\text{update}}, \gamma, \epsilon_1, \epsilon, c, K, \mathbb{I}_{[\text{using}\Phi_C]}, n$
Initialize episodic replay memory $\mathcal{D}$, global step counter $G \leftarrow 0$, dictionary $\mathcal{J} = \{\}$.
Initialize parameters θ of the network, target network parameter $\theta^- \leftarrow \theta$.
while $G < G_{\text{max}}$ **do**
  Initialize score $u_1 = 0$, hidden State of H, $h_0 = 0$ and get start textual description o_1 and initial command $a_0 =$ 'look'. Set $p_k \leftarrow 0, q_k \leftarrow 0$.
  for $t \leftarrow 1$ to T **do**
    $a_t, h_t \leftarrow$ ACT($o_t, a_{t-1}, u_t, h_{t-1}, \mathcal{J}, \epsilon, \epsilon_1, c, \theta$)
    $a_t \leftarrow$ 'look' **if** $t \bmod 20 == 0$
    Execute action a_t, observe $\{r_{t+1}, o_{t+1}, e_{t+1}\}$.
    $p_k \leftarrow 1$ **if** $r_t > 0$; $q_k \leftarrow 1$ **if** $r_t < 0$; $u_{t+1} \leftarrow u_t + r_t$
    Sample minibatch of transition sequences f (See Section 4.2.1)
    $y_{b,:}, E_{b,:} \leftarrow$ TARGETS(f, γ, θ^-)
    Perform gradient descent on $\mathcal{L}(\theta) = \sum_{i=j+n-1}^{j+l} [y_{b,i} - Q(h_{b,i}, a_i, u_{b,i}; \theta)^2 + \mathbb{I}_{[using\Phi_C]} \text{BCE}(e_i, E_{b,i})]$
    $\theta^- \leftarrow \theta$ **if** $t \bmod I_{update} == 0$
    $G \leftarrow G + 1$
    End episode **if** o_{t+1} is terminal.
  end for
  Store episode in $\mathcal{D}$.
end while

4.3 Related Work

RL applied to Text-based Games: LSTM-DQN [NKB15] deals with parser-based text adventure games and uses an LSTM to generate feedback representation. The representation is then used by an action scorer to generate scores for the action verb and objects. The two scores are then averaged to determine Q-value for the state-action pair. In the realm of choice-based games, He et al. [HCH+16] uses two separate deep neural nets to generate representations for feedback and action respectively. Q-values are calculated by dot-product of these representations. None of the above approaches deals with partial observability in text-based games.

Admissible Action Set Learning: Tao et al. [TCYA18] approach the issue of learning admissible set given context as a supervised learning one. They train their model on (input, label) pairs where input is context (concatenation of feedbacks by LOOK and INVENTORY) and the label is the list of admissible commands given this input. AE-DQN [ZHM+18] employs an additional neural network to prune in-admissible actions from the action set given a state. Although the paper doesn't deal with partial observability in text adventure games, authors show that having a tractable admissible action set led to faster convergence. Fulda et al.'s agent [FRMW17] work on bounding the action set through affordances. Their agent is trained through tabular Q-Learning.

Partial Observability: Yuan et al. [YCS+18] replace shared MLP in [NKB15] with LSTM cell to calculate context representation. However, they use the concatenation of feedbacks by LOOK and INVENTORY as the given state to make the game more observable. Their work also doesn't focus on pruning in-admissible actions given a context. Finally, Ammanabrolu and Riedl [AR19] deal with partial observability by representing the state as a knowledge graph and continuously updating it after every game step. But, the graph updation is hand-crafted, i.e., after every command executed and feedback received, there are hand-crafted update rules that prune and add edges in the graph. It would have been interesting if the agent could have learned these update rules during the gameplay.

5

Experiments and Discussion

In chapter 4, we discussed algorithmic improvements to apply Deep RL for IF. In this chapter, we will introduce a synthetic IF benchmark. This benchmark provides a graded measure to test the agent on challenges presented by text-based games namely large action space, combinatorial state space, partial observability, and reward sparsity. Finally, we will discuss various ablation studies that investigate the effectiveness of the score contextualisation and action gating in various settings. The section on ablation studies, i.e., "Empirical Analysis" has appeared in a peer-reviewed conference [JFL+20]. It has been lightly modified to include more figures and analysis.

5.1 A Synthetic IF Benchmark

Both score contextualisation and action gating are tailored to domains that exhibit the structure typical of IF. To assess how useful these methods are, we will make use of a synthetic benchmark based on the TextWorld framework [CKY+18]. TextWorld provides a reinforcement learning interface to text-based games along with an environment specification language for designing new environments. Environments provide a set of locations, or *rooms*, objects that can be picked up and carried between locations, and a reward function based on interacting with these objects. Following the genre, special *key* objects are used to access parts of the environment.

Our benchmark provides seven environments of increasing complexity, which we call

Table 5.1: Main characteristics of each level in our synthetic benchmark.

LEVEL	# ROOMS	# OBJECTS	# SUB-TASKS	$\lvert\mathcal{A}\rvert$
1	4	2	2	8
2	7	4	3	15
3	7	4	3	15
4	9	8	4	50
5	11	15	5	141
6	12	20	6	283
7	12	20	7	295

levels. We control complexity by adding new rooms and/or objects to each successive level. Each level also requires the agent to complete a number of subtasks (Table 5.1), most of which involve carrying one or more items to a particular location. The reward is provided only when the agent completes one of these subtasks. Thematically, each level involves collecting food items to make a salad, inspired by the first TextWorld competition. Example objects include an apple and a head of lettuce, while example actions include `get apple` and `slice lettuce with knife`. Accordingly, we call our benchmark SaladWorld. For possible scores and subtasks information for each level of the suite, see Table 5.2.

5.1.1 Why Introduce a Benchmark?

Rather than introducing a difficult game, we introduce a benchmark since it gives us an opportunity to test whether our ideas of score contextualisation and action gating hold across the varying levels of complexity in the suite. It should be noted, in this work, we don't intend to solve all the levels. In the following subsections, we describe the challenges that the suite presents.

Large Action Space

In IF, there are multiple ways to interact with an object given a context. One of these interactions could be directly using the portable object (e.g. `eat lettuce`) and another could be using it in conjunction with another object (in most cases stationary) e.g. `put`

`lettuce on counter`. As seen in Table 5.1, this results in many possible potential commands for the more complex levels in the suite when the number of objects (both portable and stationary) increases, making gameplay difficult. In fact, the action space becomes combinatorial as the number of object increases. To simplify the settings here a bit, we constrain the action set by limiting the possible interactions. Still, there is a noticeable increase in the action set as the level increases.

Combinatorial State Space

In a text-based game, the agent could transport a portable object from one room to another, changing the environment state. This agent behavior results in it encountering combinatorial possibilities of room-object combinations when the number of objects and rooms increases in the later levels of the suite, hence the issue of combinatorial state space. Not only this, but the state of an environment could also change even when the agent uses a portable object in conjunction with another object (e.g. putting the lettuce on the counter in kitchen changes the state of the kitchen, hence changing the environment state).

Partial Observability and Sparse Rewards

Completing each subtask requires memory of what has previously been accomplished, along with where different objects are. Together with this, each level in the SaladWorld involves some amount of history-dependent admissibility i.e the admissibility of the action depends on the history rather than the state. For example, `put lettuce on counter` can only be accomplished once `take lettuce` (in a different room) has happened. Regarding reward sparsity in text-based games, each subtask usually involves 10-20 steps in the case when the agent acts optimally. Moreover, with each successive level, the reward to complete additional subtask becomes sparser.

Finally, with successive levels, as demonstrated the action set size increases and state-space becomes combinatorially large. This implies that the same quest in Level i is tougher to solve than in Level j where $i > j$ due to the increased difficulty in exploring state space as well as action space. Moreover, as the number of sub-tasks and their complexity increases, the higher levels become more difficult to solve. This requires better exploration strategies both in action space as well as state space.

Table 5.2: Subtasks information and scores possible for each level of the suite.

Level	Subtasks	Possible scores
1	Following subtasks with reward and fulfilling condition: • 10 points when the agent first enters the vegetable market. • 5 points when the agent gets lettuce from the vegetable market and puts lettuce on the counter.	10, 15
2	All subtasks from previous level plus this subtask: • 5 points when the agent takes the blue key from open space, opens the blue door, gets tomato from the supermarket and puts it on the counter in the kitchen.	5, 10, 15, 20
3	All subtasks from level 1 plus this subtask: • 5 points when the agent takes the blue key from open space, goes to the garden, opens the blue door with the blue key, gets tomato from the supermarket and puts it on the counter in the kitchen. **Remark:** Level 3 game differs from Level 2 game in terms of number of steps required to complete the additional sub-task (which is greater in case of Level 3)	5, 10, 15, 20
4	All subtasks from previous level plus this subtask: • 5 points when the agent takes parsley from the backyard and knife from the cutlery shop to the kitchen, puts parsley into fridge and knife on the counter.	5, 10, 15, 20, 25

Continued on next page

Table 5.2 – *Continued from previous page*

Level	Subtasks	Possible scores
5	All subtasks from previous level plus this subtask: • 5 points when the agent goes to fruit shop, takes chest key, opens container with chest key, takes the banana from the chest and puts it into the fridge in the kitchen.	5, 10, 15, 20, 25, 30
6	All subtasks from previous level plus this subtask: • 5 points when the agent takes the red key from the supermarket, goes to the playroom, opens the red door with the red key, gets the apple from cookhouse and puts it into the fridge in the kitchen.	5, 10, 15, 20, 25, 30, 35
7	All subtasks from previous level plus this subtask: • 5 points when the agent prepares the meal.	5, 10, 15, 20, 25, 30, 35, 40

5.2 Empirical Analysis

In the first set of experiments, we use SaladWorld to establish that both score contextualisation and action gating provide positive benefits in the context of IF domain. We then validate these findings on the celebrated text-based game ZORK used in prior works [FRMW17, ZHM+18]. The second set of experiments investigates the effect of prioritization and infrequent LOOK during the training. In our work, training is done using a balanced form of prioritized replay (see Section) which we found improves baseline performance on ZORK appreciably. We also find that, for simplifying the exploration problem, our agent that takes a forced LOOK action every 20 steps performs better than the one that doesn't.

Our baseline agent is the LSTM-DRQN agent [YCS+18] but with a different action rep-

resentation. We augment this baseline with either or both score contextualisation and action gating and observe the resulting effect on agent performance in SaladWorld. We measure this performance as the fraction of subtasks completed during an episode, averaged over time. In all cases, our results are generated from 5 independent trials of each condition. To smooth the results, we use the moving average with a window of 20,000 training steps. The graphs and the histograms report average $\pm$ std. deviation across the trials.

5.2.1 Hyper-parameters

Training hyper-parameters: For all the experiments unless specified, $\gamma = 0.9$. Weights for the learning agents are updated every 4 steps. Score contextualisation uses $K = 5$ network heads; the baseline corresponds to $K = 1$. Parameters of score contextualisation architecture are learned end to end with Adam optimiser [KB15] with learning rate $\alpha = 0.001$. To prevent imprecise updates for the initial states in the transition sequence due to in-sufficient history, we use the updating mechanism proposed by Lample and Chaplot [LC17]. In this mechanism, considering the transition sequence of length $l, o_1, o_2, \ldots, o_l$, errors from $o_1, o_2, \ldots, o_n$ aren't back-propagated through the network. In our case, the sequence length $l = 15$ and minimum history size for a state to be updated $n = 6$ for all experiments. Score contextualisation heads are trained to minimize the Q-learning loss over the whole transition sequence. On the other hand, Φ_C minimizes the BCE (binary cross-entropy) loss over the predicted admissibility probability and the actual admissibility signal for every transition in the transition sequence. The behavior policy during training is $\epsilon-$greedy over the admissible set $\hat{\mathcal{A}}_t$. Each episode lasts for a maximum of T steps. For Level 1 game, we anneal $\epsilon = 1$ to 0.1 over 1000000 steps and $T = 100$. For rest of the games in the suite, we anneal $\epsilon = 1$ to 0.1 over 1000000 steps and $T = 200$. To simplify exploration, our agent further takes a forced LOOK action every 20 steps.

Architecture hyper-parameters: In representation generator Φ_R, word embedding size is 20 and the number of hidden units in encoder LSTM is 64. For a network head k, the number of hidden units in context LSTM is 512; action scorer $\Phi_A(k)$ is a two-layer MLP: sizes of the first and second layer are 128 and $|\mathcal{A}|$ respectively. Auxilliary classifier Φ_C has the same configuration as $\Phi_A(k)$.

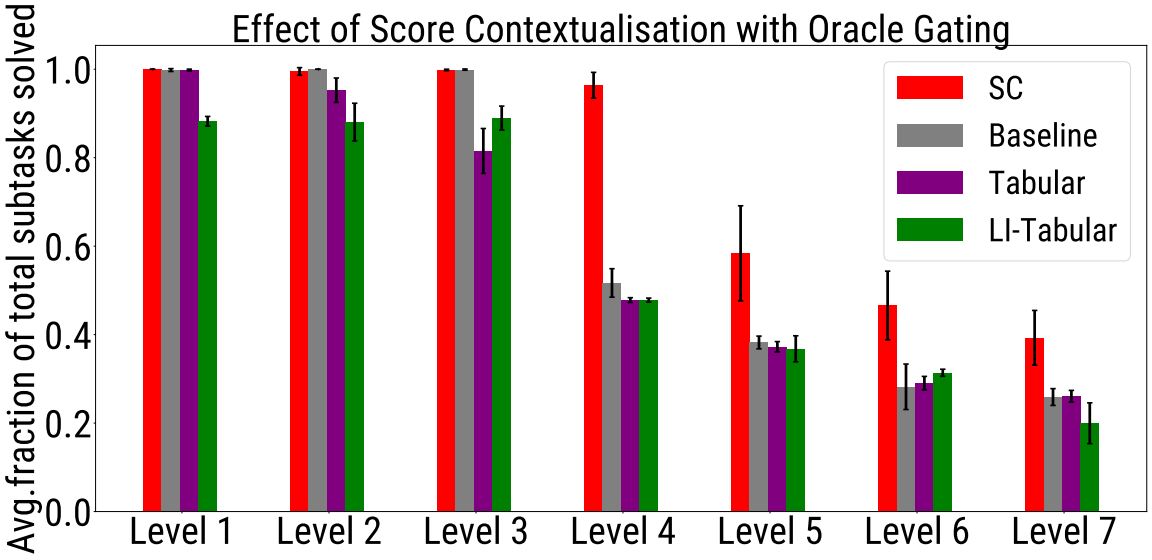

Figure 5.1: Fraction of tasks solved by each method at the end of training for 1.3 million steps. The tabular agents, which do not take history into account, perform quite poorly. LI stands for "look, inventory" (see text for details).

5.2.2 Score Contextualisation

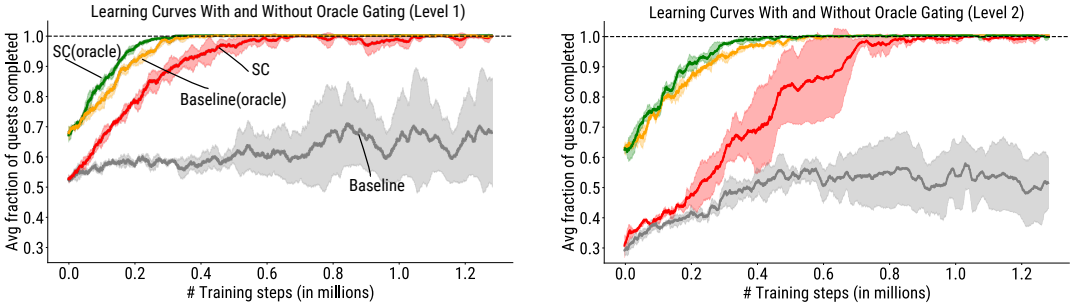

Figure 5.2: Comparing whether score contextualisation as an architecture provides a useful representation for learning to act optimally. Column 1 and 2 correspond to Level 1 and 2 respectively.

We first consider the effect of score contextualisation on our agents' ability to complete tasks in SaladWorld. We ask,

Does score contextualisation mitigate the negative effects of partial observ-

ability?

We begin in a simplified setting where the agent knows the admissible set $\mathcal{A}_t$. We call this setting *oracle gating*. This setting lets us focus on the impact of contextualisation alone. We compare our score contextualisation (SC) to the baseline and also to two "tabular" agents. The first tabular agent treats the most recent feedback as a state, and hashes each unique description-action pair to a Q-value. This results in a memoryless scheme that ignores partial observability. The second tabular agent performs the information-gathering actions LOOK and INVENTORY to construct its state description, and also hashes these to unique Q-values. Accordingly, we call this the "LI-tabular" agent. This latter scheme has proved to be a successful heuristic in the design of IF agents [FRMW17], but can be problematic in domains where taking information-gathering actions can have negative consequences (as is the case in ZORK).

Figure 5.1 shows the performance of the four methods across SaladWorld levels, after 1.3 million training steps. We observe that the tabular agents' performance suffers as soon as there are multiple subtasks, as expected. The baseline agent performs well up to the third level, but then shows significantly reduced performance. We hypothesize that this occurs because the baseline agent must estimate the hidden state from longer history sequences and effectively learn an implicit contextualisation. Beyond the fourth level, the performance of all agents suffers, suggesting the need for a better exploration strategy, for example using expert data [TZC+19].

We find that score contextualisation performs better than the baseline when the admissible set is unknown. Figure 5.2 compares learning curves of the SC and baseline agents with oracle gating and using the full action set, respectively, in the simplest of levels (Level 1 and 2). We find that score contextualisation can learn to solve these levels even without access to $\mathcal{A}_t$, whereas the baseline cannot. Our results also show that oracle gating simplifies the problem, and illustrate the value in handling inadmissible actions differently.

We hypothesize that score contextualisation results in a simpler learning problem in which the agent can more easily learn to distinguish which actions are relevant to the task and hence facilitate credit assignment. Our result indicates that it might be unreasonable to expect contextualisation to arise naturally (or easily) in partially observable domains with

large action sets. We conclude that score contextualisation mitigates the negative effects of partial observability.

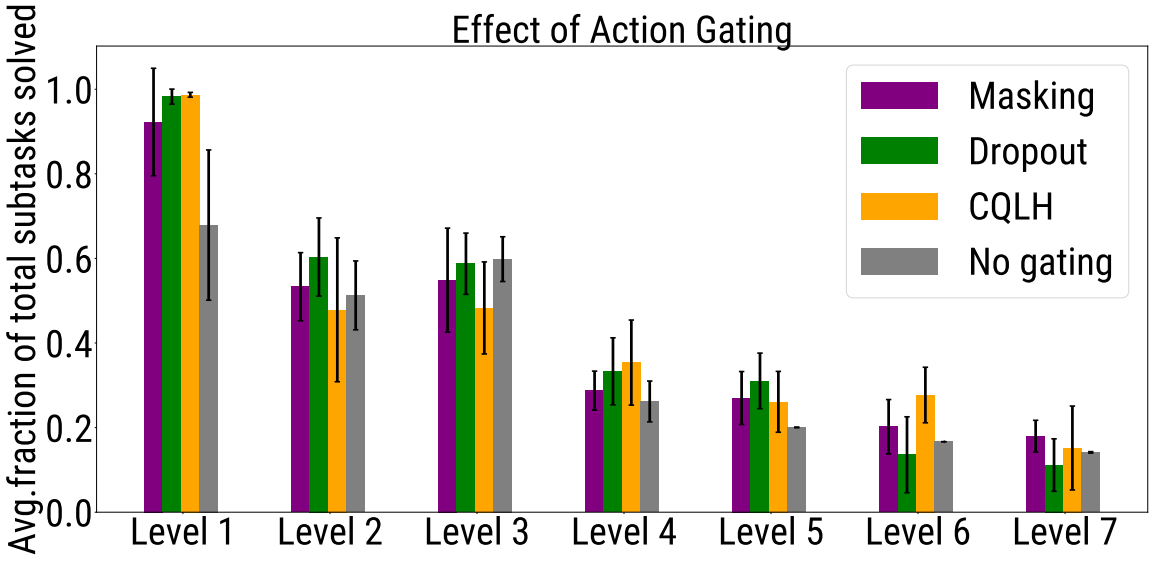

Figure 5.3: Fraction of tasks solved by each method at the end of training for 1.3 million steps. Except in Level 1, action gating by itself does not improve end performance.

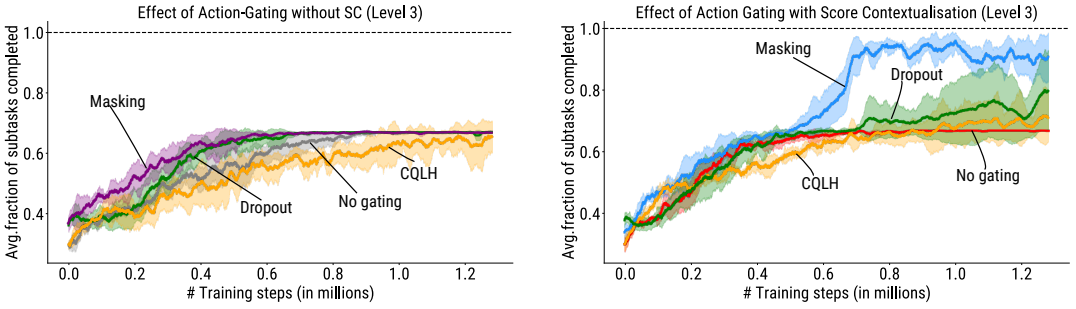

Figure 5.4: Effectiveness of action gating with score contextualisation in Level 3. Of the three methods, masking performs best.

5.2.3 Score Contextualisation with Learned Action Gating

The previous experiment (in particular, Figure 5.2) shows the value of restricting action selection to admissible actions. With the goal in mind of designing an agent that can operate from feedback alone, we now ask:

Can an agent learn more efficiently when given bandit feedback about the admissibility of its chosen actions?

We address this question by comparing our three action gating mechanisms. As discussed in Section 4.1.2, the output of the auxiliary classifier describes our estimate of an action's admissibility for a given history.

As an initial point of comparison, we tested the performance of the baseline agent when using the auxiliary classifier's output to gate actions. For the masking method, we selected $c = 0.001$ from a larger initial parameter sweep. The results are summarized in Figure 5.3. While action gating alone provides some benefits in the first level, performance is equivalent for the rest of the levels.

However, when combined with score contextualisation (see Fig 5.4, 5.5), we observe some performance gains. In Level 3 in particular, we almost recover the performance of the SC agent with oracle gating. From our results, we conclude that masking with the right threshold works best, but leave as an open question whether the other action gating schemes can be improved.

Figure 5.6 shows the final comparison between the baseline LSTM-DRQN and our new agent architecture which incorporates action gating and score contextualisation (see Figure 5.7). Our results show that the augmented method significantly outperforms the baseline, and is able to handle more complex IF domains. From level 4 onwards, the learning curves in the appendix show that combining score contextualisation with masking results in faster learning, even though final performance is unchanged. We posit that better exploration schemes are required for further progress in SaladWorld.

5.2.4 ZORK

As a final experiment, we evaluate our agent architecture on the IF ZORK I, the first installment of the popular trilogy. ZORK provides an interesting point of comparison for our methods, as it is designed by and for humans – following the ontology of Bellemare et al. [BNVB13], it is a domain which is both *interesting* and *independent*. Our main objective is to compare the different methods studied with Zahavy et al.'s AE-DQN agent [ZHM+18].

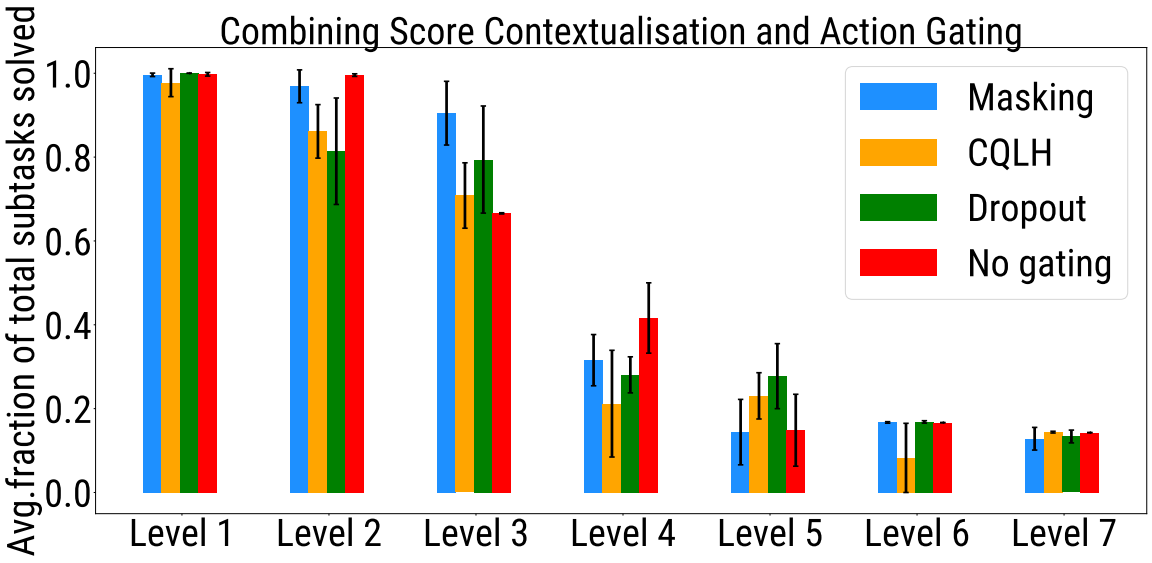

Figure 5.5: Fraction of tasks solved by each method at the end of training for 1.3 million steps. For first 3 levels, SC + Masking is better or equivalent to SC. For levels 4 and beyond, better exploration strategies are required.

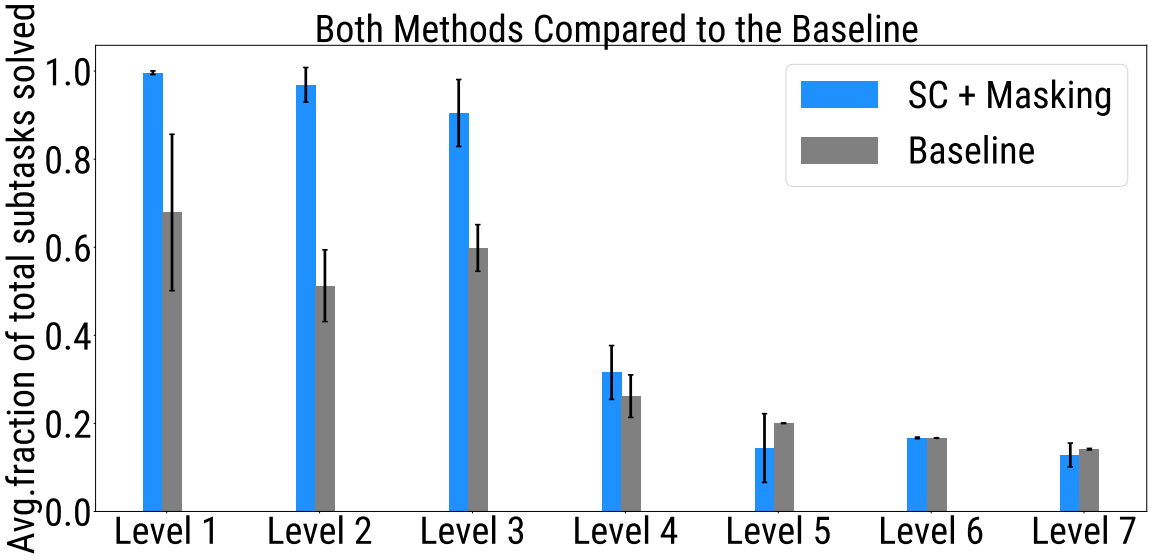

Figure 5.6: Score contextualisation and masking compared to the baseline agent. We show the fraction of tasks solved by each method at the end of training for 1.3 million steps.

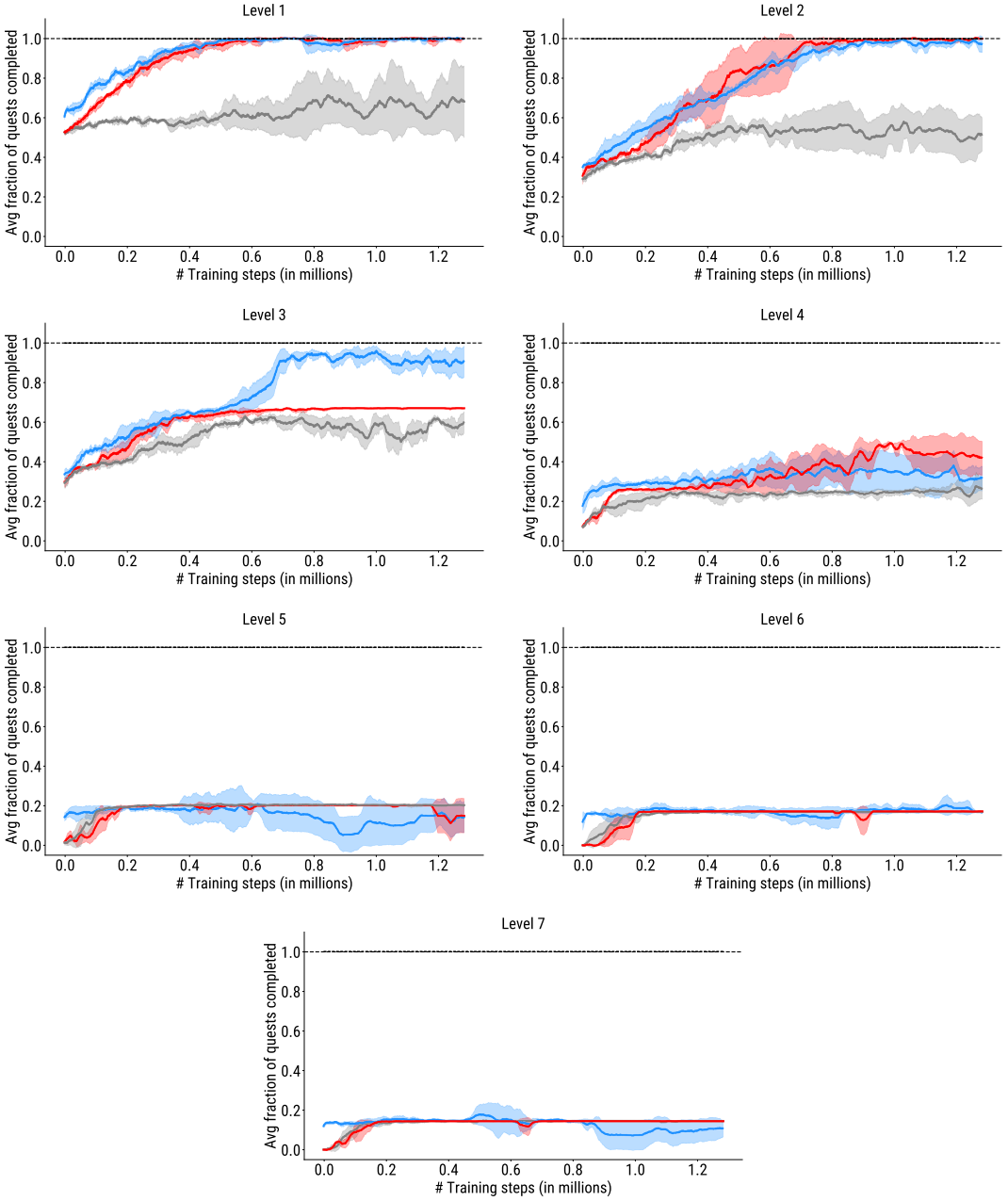

Figure 5.7: Learning curves for Score contextualisation (SC : **red**), Score contextualisation + Masking (SC + Masking : **blue**) and Baseline (**grey**) for all the levels of the SaladWorld. For the simpler levels, i.e., level 1 and 2, SC and SC + Masking perform better than Baseline. With difficult level 3, only SC + Masking solves the game. For levels 4 and beyond, we posit that better exploration strategies are required.

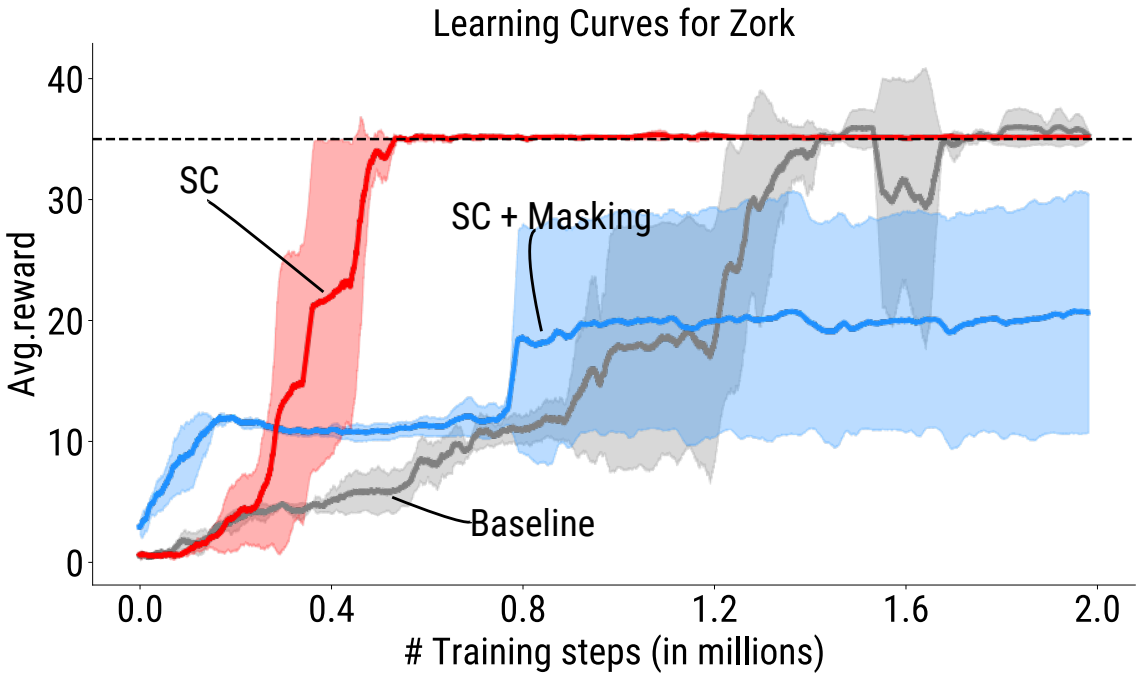

Figure 5.8: Learning curves for different agents in ZORK.

Following their experimental setup, we take $\gamma = 0.8$ and train for 2 million steps. All agents use the smaller action set (131 actions). Unlike AE-DQN, however, our agent does not use information-gathering actions (LOOK and INVENTORY) to establish the state.

Figure 5.8 shows the corresponding learning curves. Despite operating in a harder regime than AE-DQN, the score contextualizing agent reaches a score comparable to AE-DQN, in about half of the training steps. All agents eventually fail to pass the 35-point benchmark, which corresponds to a particularly difficult in-game task (the "troll quest") which involves a timing element, and we hypothesize requires a more intelligent exploration strategy.

5.2.5 Prioritised Sampling and Infrequent LOOK

Our algorithm uses prioritised sampling and executes a LOOK action every $I_{look} = 20$ steps. The baseline agent LSTM-DRQN follows this algorithm. We now ask,

Does prioritised sampling and an infrequent LOOK *play a significant role in*

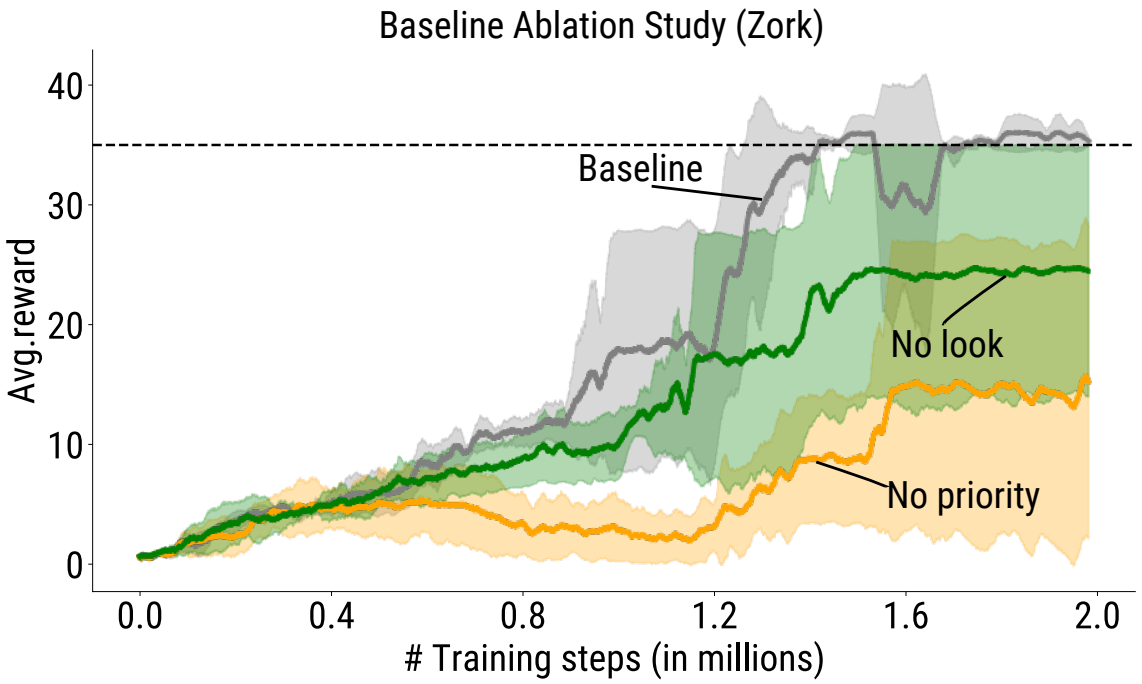

Figure 5.9: Learning curves for baseline ablation study.

the baseline's performance?

For this experiment, we compare the Baseline to two agents. The first agent is the Baseline without prioritized sampling and the second is the one without an infrequent look. Accordingly, we call them "No-priority (NP)" and "No-LOOK (NL)" respectively. We use ZORK as the testing domain.

From Fig 5.9, we observe that the Baseline performs better than the NP agent. This is because prioritized sampling helps the baseline agent to choose the episodes in which rewards are received in, thus assigning credit to the relevant states faster and overall better learning. In the same figure, the Baseline performs slightly better than the NL agent. We hypothesize that even though LOOK command is executed infrequently, it helps the agent in exploration and do credit assignment better.

6

Final Conclusion and Future Work

In this work, we introduced two algorithmic improvements for deep reinforcement learning applied to interactive fiction (IF). While naturally rooted in IF, we believe our ideas extend more generally to partially observable domains and large discrete action spaces. Our results on the synthetic domain SaladWorld and the IF domain ZORK shows the usefulness of these improvements.

Future research should focus on developing better exploration strategies that can help progress further in more difficult levels of the SaladWorld. It would be interesting to see how the existing exploration methods fare in the IF domains. One could start with adapting easier exploration methods such as pseudo-counts [BSO+16] to these domains. Since IF domains exhibit partial observability, the newer exploration methods developed might be transferable to other partially observable domains.

In future, using state of the art language models such as transformers [VSP+17] should help with the issue of representation learning. Going forward, we believe better contextualisation mechanisms should yield further gains. In ZORK, in particular, we hypothesize that going beyond the 35-point limit will require more tightly coupling exploration with representation learning – in our case, score contextualisation. An interesting avenue for future research could be developing a language model particularly suited to IF. This makes sense since the language pertinent to IF is a small subset of the English language. Furthermore, this language model could be fine-tuned when the agent interacts with the IF domains.

Another interesting direction could be using hierarchical reinforcement learning. Text-based games fit as a natural domain for testing hierarchical reinforcement learning approaches since a text-based game could be broken down into subtasks; each subtask requiring usage of different skill. In games that have a similar theme for instance cooking, learning a skill such as taking an object or cutting an object while training becomes useful because during the test time, agent is able to solve an unseen game faster. This means that learning these skills is a key to achieve generalization in text-based games.

List of Publications

Published:

- **Algorithmic Improvements for Deep Reinforcement Learning applied to Interactive Fiction.** Vishal Jain, William Fedus, Hugo Larochelle, Doina Precup, Marc G. Bellemare. *To appear in Proceedings of the Thirty-fourth AAAI Conference on Artificial Intelligence (AAAI 20).* [**Accepted for Oral Presentation**]

Bibliography

[ABC+18] Timothy Atkinson, Hendrik Baier, Tara Copplestone, Sam Devlin, and Jerry Swan. The text-based adventure AI competition. *IEEE Transactions on Games*, 2018.

[AR19] Prithviraj Ammanabrolu and Mark Riedl. 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*, 2019.

[Bel56] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 1956.

[BNVB13] Marc G. Bellemare, Yasser Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2013.

[BOG+16] Marc G. Bellemare, Georg Ostrovski, Arthur Guez, Philip S. Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

[BSO+16] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*. 2016.

BIBLIOGRAPHY

[CKY+18] 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. *CoRR*, 2018.

[FRMW17] Nancy Fulda, Daniel Ricks, Ben Murdoch, and David Wingate. What can you do with a rock? affordance extraction via word embeddings. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[GCPT07] Amir Globerson, Gal Chechik, Fernando Pereira, and Naftali Tishby. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, 2007.

[Gib77] James J Gibson. The theory of affordances. *Hilldale, USA*, 1977.

[GSR+18] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *International Conference on Learning Representations*, 2018.

[Has10] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*. 2010.

[HCH+16] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[Hec89] Hecht-Nielsen. Theory of the backpropagation neural network. In *International Joint Conference on Neural Networks*, 1989.

[Hoc91] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 1991.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 1991.

BIBLIOGRAPHY

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[HS15] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.

[IKVM18] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

[JFL+20] Vishal Jain, William Fedus, Hugo Larochelle, Doina Precup, and Marc G. Bellemare. Algorithmic improvements for deep reinforcement learning applied to interactive fiction. In *Proceedings of Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[KB15] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[KLC98] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 1998.

[LC14] Rémi Lebret and Ronan Collobert. Word embeddings through hellinger PCA. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, 2014.

[LC17] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[MA93] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. 1993.

BIBLIOGRAPHY

[MBT+18] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 2018.

[McC77] John McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, 1977.

[Min61] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 1961.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013.

[Mue87] E. T. Mueller. *Daydreaming and Computation: A Computer Model of Everyday Creativity, Learning and Emotions in the Human Stream of Thought*. PhD thesis, 1987.

[NKB15] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[Ola15] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[Ope18] OpenAI. Gym retro. https://github.com/openai/retro, 2018.

[PAED17] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014.

[RHW+88] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 1988.

[RZQ+19] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *International Conference on Machine Learning*, 2019.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[Sze10] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[TCYA18] Ruo Yu Tao, Marc-Alexandre Côté, Xingdi Yuan, and Layla El Asri. Towards solving text-based games by producing adaptive action spaces, 2018.

[TH12] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[TZC+19] Chen Tessler, Tom Zahavy, Deborah Cohen, Daniel J. Mankowitz, and Shie Mannor. Sparse imitation learning for text based games with combinatorial action spaces. *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, 2017.

[Wat89] Christopher J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

BIBLIOGRAPHY

[Wer90] Paul Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

[YCS+18] 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, 2018.

[ZHM+18] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.