Policy-based Reinforcement Learning for Generalisation in Interactive Text-based Environments

Edan Toledo[*1,4] **Jan Buys**[2] **Jonathan Shock**[3,5,6]
[1] Department of Computer Science and Technology, University of Cambridge
[2] Department of Computer Science, University of Cape Town
[3] Department of Mathematics and Applied Mathematics, University of Cape Town
[4] InstaDeep [5] INRS, Montreal, Canada [6] NiTHeCS, South Africa
et498@cam.ac.uk, jbuys@cs.uct.ac.za,
jonathan.shock@uct.ac.za

Abstract

Text-based environments enable RL agents to learn to converse and perform interactive tasks through natural language. However, previous RL approaches applied to text-based environments show poor performance when evaluated on unseen games. This paper investigates the improvement of generalisation performance through the simple switch from a value-based update method to a policy-based one, within text-based environments. We show that by replacing commonly used value-based methods with REINFORCE with baseline, a far more general agent is produced. The policy-based agent is evaluated on Coin Collector and Question Answering with interactive text (QAit), two text-based environments designed to test zero-shot performance. We see substantial improvements on a variety of zero-shot evaluation experiments, including tripling accuracy on various QAit benchmark configurations. The results indicate that policy-based RL has significantly better generalisation capabilities than value-based methods within such text-based environments, suggesting that RL agents could be applied to more complex natural language environments.

1 Introduction

General domain language comprehension is an important component of the goal to create Artificial Intelligence agents that can perform real-world activities (Gil and Selman, 2019). In the pursuit of such systems, classic text-based games are often utilised as a bridge to the generality of the real world (Yuan et al., 2018; Ammanabrolu and Riedl, 2019a).

Text-based games are a useful test-bed for exploring interactive dialogue agents in the context of partially observable Markov decision processes (POMDPs) that have large combinatorial action spaces. Additionally, they provide environments that depend on the comprehension and use of natural language. In recent years, there has been a shift in focus from simply performing well in a single text-based environment (Haroush et al., 2018; Narasimhan et al., 2015b) to generalising in unseen environments (Xu et al., 2021; Adhikari et al., 2020; Yuan et al., 2018, 2019). This shift in focus has often relied on complex methods and bespoke solutions (Ammanabrolu and Riedl, 2019b,a; Yin et al., 2020; Adolphs and Hofmann, 2019). A great deal of this research has utilised value-based reinforcement learning (RL) methods which often show high training performance but poor zero-shot generalisation. These methods have been shown to overfit and memorise the training environments, thereby not learning a generalisable policy (Farebrother et al., 2018a).

Previous work has shown that policy-based methods can work well in non text-based settings where generalization is required (Cobbe et al., 2020). However, to the best of our knowledge no controlled comparison has been made between the generalisability of value-based and policy-based RL methods in text-based environments. This leads us to investigate policy-based RL in text-based environments. We hypothesise that learning an accurate and general value function in a partially observable environment with a large action space (such as natural language) is too difficult for most value-based methods. We aim to show that policy-based approaches solve this generalisation issue by directly learning a stochastic and general policy.

In this paper, we use two text-based environments, Coin Collector (Yuan et al., 2018) and QAit (Yuan et al., 2019). Both environments utilise TextWorld (Côté et al., 2018) to generate environments on the fly. Agents must acquire natural language skills in order to navigate successfully within the worlds and achieve some specified goal. We compare a policy-based method, trained using the

[*] Work done whilst at InstaDeep and the University of Cape Town

Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, pages 1230–1242

REINFORCE with baseline algorithm (Williams, 1992) against Coin Collector and QAit's value-based approaches on the environments' respective test sets. For comparison, we make use of both Coin Collector and QAit's open-sourced codebases with a simple replacement of the value-based update and action selection method.

Our results show that the policy-based method leads to substantially improved performance over previous value-based methods on a variety of zero-shot evaluations. In the Coin Collector task, we see greater average performance and stability on the easy and medium test sets (after minimal hyper-parameter tuning). On the QAit benchmark, large accuracy improvements are obtained for location questions (28% to 98% for fixed maps) and existence questions (69.2% to 94.8% for fixed maps). The results indicate that policy-based RL has significantly better generalisation capabilities than value-based methods within these contexts, and are more suited to text-based environments.

This paper's main contributions are as follows:

1. We show that policy-based methods have better and more stable zero-shot performance on two text-based environments.

2. We show that large improvements can be made through a simple replacement of the update method without significant change to any deep learning architectures.

3. We provide new results for the QAit and Coin Collector environments against which further research can be compared.

2 Background

2.1 Policies and Value Functions

In text-based environments, agents are presented with a descriptive paragraph describing their current state (Figure 1). Agents interact with the environment using natural language commands. These commands could be long descriptive sentences such as *"go to the kitchen and pick up the copper key"* or a short templated sequence such as *"go kitchen"* and *"pickup copper key"*. The environment interprets these commands and responds with a new paragraph or sentence describing the changed state or event.

An agent in a text-based environment uses a policy to predict a command string as a sequence of words at every time step t. A policy π is defined

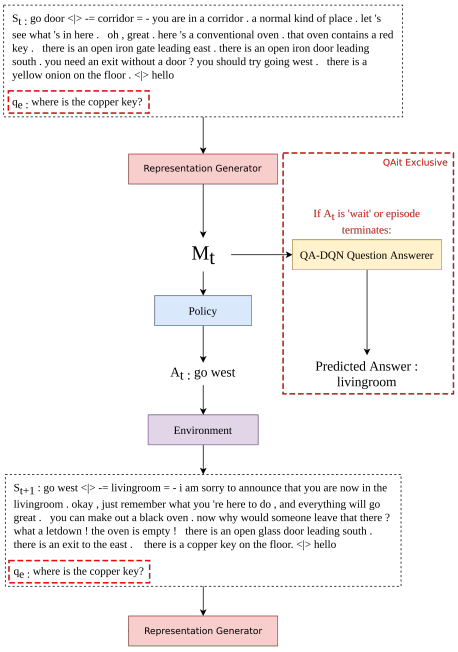


Figure 1: Overview of agent-environment interaction. The agent receives a textual observation from the environment, encoded in a latent representation, which is used to compute the policy and value functions. In QAit this representation is also used for the QA module.

as a mapping of environment states or observations $\mathcal{S}$ to the probabilities of selecting each possible word in the command string $a \in \mathcal{A}$ (Sutton and Barto, 2018). If an agent is following a policy π at time step t, the policy function $\pi(a|s)$ is the probability of performing the specific action $A_t = a$ given that the agent is in state $S_t = s$. The agent's high-level policy for command strings represents the joint policy of each sub-policy used for each word.

Widely-used RL methods such as Q-learning (Watkins and Dayan, 1992) and Actor-Critic algorithms (Konda and Tsitsiklis, 1999) estimate the state-action value (or Q-value) function within an environment. This function maps a state-action pair to the expected cumulative future reward under a given policy. The expected cumulative reward G_t is defined as

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \qquad (1)$$

where T is the terminal time step, R_k is the reward at time step k, and γ is the discount factor (the weight of importance given to future rewards). Formally, the Q-value function for any given policy is defined for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$ as

$$Q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a], \qquad (2)$$

while the state value function $V_\pi(s)$ is $\max_a Q_\pi(s, a)$.

2.2 Value-Based Methods

In value-based methods the agent tries to learn $V(s)$ (or $Q(s, a)$), which tells the agent what states it should be in (and which actions it should take) in order to maximise G_t. The agent can use $Q(s, a)$ to select the action which will take it to the most valuable next state by choosing $\text{argmax}_a\ Q_\pi(s, a)$. This is called the greedy policy and is commonly used to implement control in value-based methods (Sutton and Barto, 2018).

2.3 Policy-based Methods

In policy-based methods the agent tries to learn the policy directly. In contrast to value-based methods, this allows the agent to learn stochastic policies (Sutton and Barto, 2018).

REINFORCE (Williams, 1992) is a Monte Carlo method that updates the policy function's parameters (neural network weights) directly by moving them in the direction which will increase the expected future returns (the network's loss function). However, REINFORCE suffers from high variance with noisy gradient estimates and no clear credit assignment to positive or negative actions throughout the episode (Sutton and Barto, 2018). A simple improvement is to reduce the variance of the empirical returns G_t by subtracting a baseline function $b(s)$ in the policy gradient. The baseline is regarded as a proxy for the true expected return. A popular option for the baseline function is the state value function $V(S_t)$. This requires the agent to learn the value function alongside the policy, which can introduce a bias at the cost of lowering variance. REINFORCE uses Monte Carlo episodic sample returns G_t (refer to §2.1) to update the baseline and calculate the advantage factor with which to update the policy.

2.4 Generalisation in Text-based Games

Much research has gone into applying reinforcement learning to text-based games (see Osborne et al. (2022) for a survey), predominantly using value-based update methods. These value-based agents have been shown to often overfit on the training environment and have poor out-of-domain performance (Yuan et al., 2018, 2019). This is supported by evidence showing that DQN suffers from severe overfitting on training environments for Atari games (Farebrother et al., 2018b).

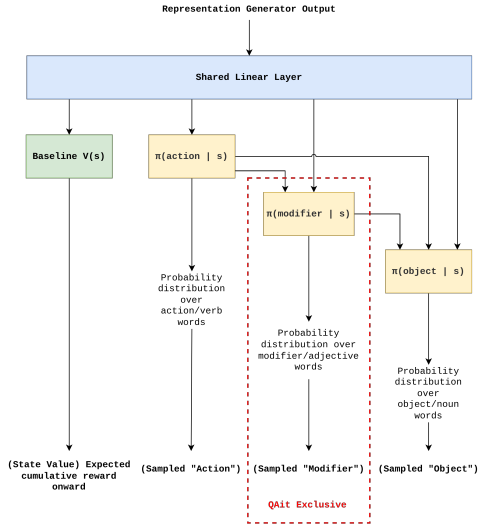


Figure 2: Overview of REINFORCE with baseline *action scorer* architecture replacement. The policy and value functions all share a linear layer that takes in the encoded text observation as input. Both Coin Collector and QAit experiments make use of this architecture for interaction, with the exception that Coin Collector's policy outputs two words whereas QAit's policy outputs three words.

Research on improving agents' generalisation performance in text-based games has mainly focused on adding components such as knowledge graphs, as well as improving value-based learning algorithms (Adhikari et al., 2020; Yuan et al., 2018; Ammanabrolu and Riedl, 2019b,a; Yin et al., 2020; Yuan et al., 2019).

Adolphs and Hofmann (2019) showed that employing an actor-critic learning method, A2C (Mnih et al., 2016), significantly increased generalisation performance compared to value-based baselines such as LSTM-DQN (Narasimhan et al., 2015a) and DRRN (He et al., 2015). However, their results do not provide a direct comparison and their agent employs an additional helper model for command generation. This helper model is domain-specific and does not allow one to fully infer the performance difference from a simple switch to a policy-based learning method. There has thus been no research directly showing that policy-based methods improve generalisation over value-based ones when applied to text-based tasks.

3 Policy-based Text-based Agents

We propose a policy-based agent that makes only minor changes to previous value-based approaches. For both environments, the neural network architec-

ture is kept largely unmodified, in order to focus on comparing the generalisation performance of the value-based baselines and the policy-based method (REINFORCE with baseline). The only changes are the use of a policy-based update method, an additional MLP to predict state values (to use as $b(s)$ - the baseline function), and the conditioning of future command words on previous command words (shown in Figure 2). For most value-based methods, the Q-values of actions are learnt. This means that the MLP used to predict Q-values already has the necessary number of output nodes for a policy-based method.

Each output layer represents the policy for each word in the command tuple, e.g. (Action, Modifier, Object) for QAit and (Verb, Noun) for Coin Collector. Each output gives a probability distribution over the vocabulary. Each word in the command tuple is then sampled from these probability distributions to form the command at each game step. This more closely emulates how a human would speak compared to value-based methods, which traditionally act using greedy action selection at evaluation time (which is deterministic) and epsilon-greedy action selection at training time (which introduces some stochasticity) (Sutton and Barto, 2018).

The baseline function approximates the state-value function to aid in the training of the policy. The baseline network also uses the policy's shared linear layer as input to produce the state value $V(s)$. This is done to regularise the shared linear layer to a common representation to aid the policy and value function in generalisation (Silver et al., 2017). The REINFORCE with baseline algorithm is used to update model weights. Each policy's entropy (for each word in the command tuple) is subtracted from the total loss. This incentivises more stochastic policies to be learnt whilst maximising reward, in order to promote exploration and generalisation. The loss functions used for the policies $\mathcal{L}_a$ and baseline $\mathcal{L}_c$ are:

$$\mathcal{L}_a = \frac{1}{T}\sum_{t=0}^{T}(G_t - V(s_t)) * \log \pi(a_t) \quad (3)$$

$$\mathcal{L}_c = \frac{1}{2T}\sum_{t=0}^{T}(G_t - V(s_t))^2 \quad (4)$$

The command word conditioning is done as follows: Each consecutive word in the command tuple receives the vector of raw outputs from the previous word, concatenated with the shared linear output. A small ablation of the effects of command conditioning indicated only a slight performance increase. Conditioning on consecutive words did not lead to any performance gains on the value-based baselines.

4 Coin Collector

Coin Collector (Yuan et al., 2018) is a text-based deterministic version of the chain experiment (Osband et al., 2016; Plappert et al., 2017) that introduces off-chain nodes (distractor rooms) to distract and confuse the agent.

4.1 Environment

An agent needs to learn how to navigate a through a path of rooms (nodes) to the final goal location whereby it must collect a "coin". In the optimal case, an agent never revisits distractor rooms. Every game starts off with the agent at one end of the *chain* and the *coin* at the other end. Coin Collector games have a finite horizon, thereby requiring the agent to collect the coin with a limited number of mistakes deviating from the optimal trajectory.

Coin Collector has two parameters that control environment difficulty. There are 3 possible modes (*easy*, *medium*, *hard*) for constructing a game, with zero, one, and two *distractors* along the optimal path, respectively. The agent needs to learn that if it makes a mistake and enters a distractor room, it needs to go back the way it came and continue in a different direction. Each game also has a difficulty **level** which indicates the length of the optimal path.

The Coin Collector environment has two reward types. An agent is given an exploration reward each time it enters a previously unseen *physical location*. Additionally, it receives a terminal reward of 1 if it collects the coin.

4.2 Interaction

Interaction in the environment requires a very limited vocabulary. Dividing action selection into verb and noun, the action space is: {*go*, *take*} × {*north*, *south*, *east*, *west*, *coin*}. The only valid commands are: *go north*, *go east*, *go south*, *go west* and *take coin*.

4.3 Evaluation

Each mode (easy, medium, hard) has 5 distinct test sets of increasing difficulty levels: 5, 10, 15, 20, 30. Each test set contains 10 unseen games. We evaluate agents on all test sets during training to

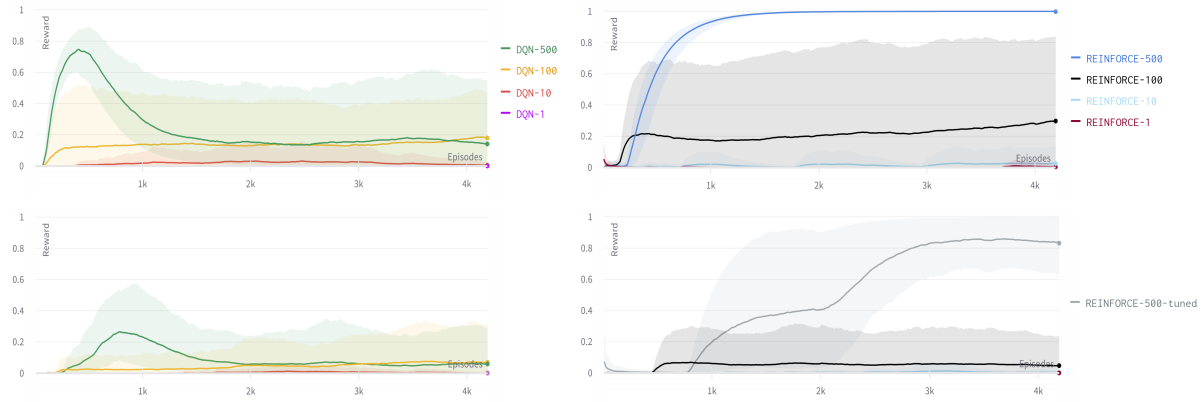


Figure 3: Coin Collector easy (top row) and medium (bottom row) test set results, averaged over all training runs. Shaded regions indicates max and min performance achieved.

see the performance and stability of generalisation as an agent learns how to solve its training games. By evaluating on higher-level games than the agent has trained on, we can see to what extent it is truly able to generalise and extrapolate its policy beyond what it has seen. The evaluation is done on agents trained on 1, 10, 100, and 500 unique level 10 games in their respective modes. Each agent is trained for 20 000 episodes.

4.4 Architecture

The original Coin Collector architecture (Yuan et al., 2018) has two modules: The representation generator and the action scorer. The representation generator is used to encode textual observations and produce an input for the action scorer. It consists of an embedding layer, an LSTM (Hochreiter and Schmidhuber, 1997), and a mean pooling layer. The action scorer is used to produce Q-values for each possible action in the command sequence (§4.2). It consists of two MLPs that share a lower layer. This model does not condition on previous actions or observations, which can present an issue due to the partially observable nature of the environment. To mitigate this issue, the previous command is concatenated to the observation. Yuan et al. (2018) implemented two value-based RL methods, LSTM-DQN (Narasimhan et al., 2015a) and LSTM-DRQN (Hausknecht and Stone, 2015; Lample and Chaplot, 2017), with this architecture. The only difference between the two is that LSTM-DRQN uses a recurrent action scorer module which takes in the hidden state output of the previous timestep. Our policy-based agent makes use of this architecture with changes illustrated in section 3.

4.5 Results

Figure 3 shows the results of our Coin Collector experiments, comparing the LSTM-DQN to our policy-based agent trained with REINFORCE. The curves indicate the average performance over all test sets for each agent when trained on 1, 10, 100, and 500 games. Both LSTM-DQN and REINFORCE failed to perform adequately when trained on 1 or 10 games.

Yuan et al. (2018) showed that LSTM-DQN performs better on the medium setting and is comparable to LSTM-DRQN on the easy setting, so in this paper we only compare to LSTM-DQN, which also simplifies the update method replacement. Experiments in the hard mode setting were not run as we were unable to reproduce the reported results using the original code base.

For the test sets in the easy setting, our policy-based agent matches or outperforms DQN on every level. We see that here the REINFORCE agent manages to achieve a similar maximum performance when training on only 100 games compared to LSTM-DQN training on 500 games. Additionally, the policy-based method is much more stable and does not suffer from catastrophic losses in performance due to overfitting the training data. This suggests that the policies learn not to simply memorise the training set as seen in the LSTM-DQN agent, which loses generalisation ability as training continues.

In the medium setting, the performance improvement is less apparent than the easy setting. The LSTM-DQN architecture performs better overall than the REINFORCE agent, largely due to the policy-based method failing to learn anything in the 500 games setting. Due to the policy-based

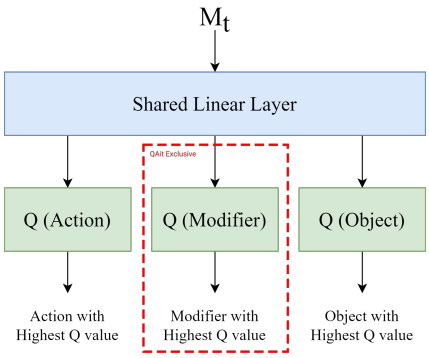


Figure 4: Overview of DQN network (action scorer) architecture.

method simply being a substitute for the value-based update, no hyperparameters were changed. This was done intentionally to show that all that is needed to generalise better is to replace the update method. By changing the discount factor from 0.5 (as is specified by Yuan et al. (2018)) to 0.9 the REINFORCE agent trained on 500 games achieves significantly higher performance than LSTM-DQN on all the medium test sets.

5 QAit

The Question Answering with interactive text (QAit) (Yuan et al., 2019) task is a text-based question answering problem in which an agent must interact with a partially observable text-based environment to gather the declarative knowledge required to answer questions. QAit aims to test an agent's language comprehension abilities by posing questions about the location, existence and attributes of objects distributed throughout the environment.

5.1 Environment

We use the implementation of QAit built on top of TextWorld to create text environments and associated questions dynamically. All environments are generated procedurally according to two environment categories (fixed map and random map). The fixed maps always have 6 unique rooms, while for random map the number of rooms is drawn from a uniform distribution between 2 and 12. For both maps types the number of entities is sampled from a uniform distribution between 3 and 6 times the number of rooms in the map. Room connectivity and structure is also changed from map to map.

Questions based on each environment are created on the fly as an agent plays a game. Yuan et al. (2019) trained agents on datasets consisting of 1, 2, 10, 100, and 500 created environments, as well as an unlimited setting where different environments are sampled for each question. In the unlimited setting more than 10^{40} different games can be created, making it unlikely that the agent will see the same environment multiple times.

The QAit environment has two reward types. An agent is given an exploration reward each time it enters a previously unseen *physical location* and/or new *inventory status*. In addition, a sufficient information reward is given at the end of the episode. These rewards are explained in §5.3.

There are three types of questions that are posed to the agent: **Location** questions ask the whereabouts of objects situated within the world, e.g. "Where is the can of soda?", where a suitable answer would be "fridge". **Existence** questions ask about the presence of objects situated within the world, e.g. "is there a raw egg in the world?", where the answer would simply be yes or no. **Attribute** questions ask whether or not an object has a certain associated attribute, e.g. "is apple edible", where the answer is also yes or no. These questions are the most challenging, as the agent has to both find and interact with the object, e.g. find the apple, pick it up, and try to eat it.

5.2 Interaction

All text commands are triplets of the form action, modifier, object (e.g., "open metallic gate"). When there is no ambiguity, the environment understands commands without modifiers, e.g. "pick key" will result in picking up the "copper key" provided it is the only key in the room. At each game step, there are separate lexicons for actions, modifiers and objects. An episode of experience terminates when a maximum number of steps is reached or the *wait* command is issued by the agent, indicating that it wants to answer the question. For our experiments, we use a maximum of 80 steps. There are, on average, 17 actions, 18 modifiers, and 27 objects per game, and 93.1 and 89.7 observed tokens in fixed and random maps games, respectively.[1]

5.3 Evaluation

The QAit test set provides 500 hold-out games for each of the question types and for both map types. This test set is used to benchmark the generalisation abilities of agents in all experimental

[1] Statistics calculated over 10,000 sampled games.

configurations. This allows for models to be assessed in a reproducible manner and is analogous to supervised learning test sets. The evaluation metrics used are accuracy and sufficient information. Each agent is trained for 200 000 episodes. **Accuracy** refers to the proportion of correctly answered questions. The distribution of answers in the QAit evaluation set is presented in the Appendix (Tables 6, 7 and 8).

Sufficient Information is used to evaluate the amount of information gathered by the agent and whether or not the information was sufficient to answer the question (Yuan et al., 2019). It is also used as part of the reward function. It measures the performance of the navigation and interaction required by the agent to answer a given question. For each question type, the sufficient information score is calculated as follows:

- **Location**: A score of 1 is given if the entity mentioned in the question is present in the final observation when the agent decides to stop the interaction, indicating that the agent has observed the information it needs to answer the question successfully. Otherwise, a score of 0 is given.
- **Existence**: If the answer to the question is ***yes***, a score of 1 is given if the entity mentioned in the question is present in the final observation. If the answer is ***no***, a score between 0 and 1 is given representing the proportion of the environment that the agent has explored, accounting for the fact that an agent can only be sure that an entity does not exist if it has explored the whole environment.
- **Attribute**: Attribute questions have a set of heuristics defined to verify each attribute and assign a score of sufficient information. Each attribute has specific commands that need to be executed or certain states the agent needs to be in for sufficient information to be gathered.

5.4 Architecture

As in the Coin Collector architecture, the original QAit architecture (Yuan et al., 2019) consists of a representation generator and an action scorer. An additional module is used to answer the question (the component exclusive to QAit in figure 1). The representation generator is a transformer encoder (Vaswani et al., 2017) consisting of an embedding layer, two stacks of transformer blocks (one for encoding and the other for aggregation), and a final attention layer. The action scorer has a shared linear layer followed by MLPs for each word in the command sequence (Figure 4). The question answerer appends an additional stack of aggregation transformer blocks to compute the answer from the encoder output. At each game step, the question representation is merged with the representation of the current game observation to produce the final state representation, so that the agent cannot forget the goal. The QAit task (Yuan et al., 2019) provides three value-based RL methods using this architecture as baselines: DQN (Mnih et al., 2013), DDQN (Van Hasselt et al., 2016) and Rainbow (Hessel et al., 2018). As with Coin Collector, our policy-based agent uses this architecture with the changes presented in §3.

5.5 Results

Table 1 gives the QAit test set results for all experiments as well as the baseline models' performance as reported by Yuan et al. (2019). Tables 3 and 4 in the Appendix give the full results including training performance. The policy-based method outperforms all the value-based methods on location and existence questions in each of the number of games settings on both map types, in many instances by a large margin. In many settings the increase in sufficient information score is even larger than the increase in question answering accuracy.

Most notable is the large increase in test accuracy of the policy-based method in larger numbers of training games. The value-based methods' performances increase only slightly with the number of games, compared to the large jumps in the policy-based method, indicating much better generalisation ability. In the unlimited games setting the policy-based agent reaches 98% and 90.2% accuracy on location and existence questions respectively on the fixed map setting. This is compared to the best value-based agent's accuracies of 28% and 69.2%. On existence questions, the increase in sufficient information score from 0.246 to 0.781 is even larger, suggesting a dramatic improvement in navigation and interaction. On random maps the improvement is also large, albeit not as stark as in fixed maps. We believe that this is due to the more difficult nature of random map-type games having potentially double the number of rooms, thereby making exploration more difficult and giving rise to more possible entities.

Model	Fixed			Random		
	Location	**Existence**	**Attribute**	**Location**	**Existence**	**Attribute**
Random	0.027	0.497	0.496	0.034	0.5	0.499
10 games						
DQN	0.180 (0.188)	0.568 (0.156)	0.518 (0.030)	0.156 (0.160)	0.566 (0.142)	0.518 (0.036)
DDQN	0.188 (0.208)	0.566 (0.128)	0.516 (0.036)	0.142 (0.154)	0.606 (0.153)	0.500 (0.033)
Rainbow	0.156 (0.170)	0.590 (0.131)	**0.520 (0.023)**	0.144 (0.170)	0.586 (0.128)	0.530 (0.018)
REINFORCE	**0.230 (0.244)**	**0.654 (0.357)**	0.498 (0.049)	**0.266 (0.282)**	**0.656 (0.317)**	**0.534 (0.038)**
100 games						
DQN	0.194 (0.206)	0.614 (0.160)	0.498 (0.014)	0.184 (0.204)	0.668 (0.181)	0.524 (0.017)
DDQN	0.168 (0.196)	0.650 (0.216)	0.528 (0.017)	0.188 (0.204)	0.662 (0.205)	0.544 (0.019)
Rainbow	0.156 (0.160)	0.602 (0.207)	0.524 (0.022)	0.174 (0.184)	0.654 (0.190)	0.504 (0.032)
REINFORCE	**0.786 (0.802)**	**0.898 (0.768)**	**0.530 (0.043)**	**0.492 (0.508)**	**0.822 (0.471)**	**0.546 (0.055)**
500 games						
DQN	0.224 (0.244)	0.674 (0.279)	**0.534 (0.014)**	0.204 (0.216)	0.678 (0.214)	0.530 (0.017)
DDQN	0.218 (0.228)	0.626 (0.213)	0.508 (0.026)	0.222 (0.246)	0.656 (0.188)	0.486 (0.023)
Rainbow	0.190 (0.196)	0.656 (0.207)	0.496 (0.029)	0.172 (0.178)	0.678 (0.191)	0.494 (0.017)
REINFORCE	**0.948 (0.958)**	**0.948 (0.892)**	0.466 (0.045)	**0.570 (0.588)**	**0.836 (0.560)**	**0.534 (0.044)**
unlimited games						
DQN	0.216 (0.216)	0.662 (0.246)	**0.514 (0.016)**	0.188 (0.188)	0.668 (0.218)	**0.506 (0.018)**
DDQN	0.258 (0.258)	0.628 (0.134)	0.480 (0.024)	0.206 (0.206)	0.694 (0.196)	0.482 (0.017)
Rainbow	0.280 (0.280)	0.692 (0.157)	0.514 (0.014)	0.258 (0.258)	0.686 (0.193)	0.470 (0.017)
REINFORCE	**0.980 (0.980)**	**0.902 (0.781)**	0.462 (0.032)	**0.738 (0.738)**	**0.858 (0.584)**	0.502 (0.042)

Table 1: QAit test set results for fixed and random map configurations. QA accuracy is reported with sufficient information scores in brackets.

Model	Fixed		
	Location	**Existence**	**Attribute**
500 games			
DQN	0.430 (0.430)	0.742 (0.136)	0.700 (0.015)
DDQN	0.406 (0.406)	0.734 (0.173)	0.714 (0.021)
Rainbow	0.358 (0.358)	0.768 (0.187)	0.736 (0.032)
Policy-based	**0.990 (0.990)**	**0.964 (0.916)**	**0.748 (0.048)**

Table 2: QAit training performance for agents trained on 500 fixed games. QA accuracy is reported with sufficient information scores in brackets.

For attribute-type questions, neither the policy-based model nor the value-based methods achieve results significantly higher than a random agent in terms of QA accuracy in any of the settings. By looking at the sufficient information score, we can see that the models rarely end up in the states they should be in order to answer the question. Therefore these results are most likely due to chance.

The policy-based method achieves higher training QA accuracy and sufficient information than the value-based methods when using the same number of training episodes (Table 2). This is notable since off-policy methods (DQN, DDQN, Rainbow) are generally more sample efficient than on-policy methods (REINFORCE). The REINFORCE agent only updates weights once per episode, whereas the value-based methods update their weights every 20 steps (2 to 4 times per episode). This result suggests that learning a policy directly in the QAit environment is an easier task than approximating the complicated Q-function.

Analysis of the value-based methods' interaction in the test set shows that the methods often get the agent stuck during exploration. This is largely due to the deterministic nature of the algorithms for interaction. We hypothesise that the stochastic nature of the policy-based method affords it enough flexibility to learn more generalised policies.

6 Conclusion

This work demonstrates the advantages of using policy-based methods in textual environments. More specifically, we investigated the differences in generalisation performance of the REINFORCE with baseline algorithm compared to value-based RL baselines on two sets of text-based tasks. The results strongly suggest that policy-based RL methods are not only more suited for textual domains due to their training performance, but also possess generalisation capabilities beyond their value-based counterparts.

Limitations

As this paper is a comparison of policy-based and value-based deep RL methods and their generalisation capabilities, a large limitation is the scarcity of text-based environments available to evaluate on. To confirm the stark differences in generalisation performance, a large suite of text-based environments would be needed for training and evaluation. This is clearly hampered by the lack of such suitable environments. Additionally, more policy-based methods and value-based baselines would need to be evaluated to confirm that the performance differences are not environment- or algorithm-specific.

Acknowledgements

This work is based on research supported in part by the National Research Foundation of South Africa (Grant Number: 129850) and the South African Centre for High Performance Computing. Computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team: hpc.uct.ac.za.

References

Ashutosh Adhikari, Xingdi (Eric) Yuan, Marc-Alexandre Côté, Mikulas Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and William L. Hamilton . 2020. Learning dynamic belief graphs to generalize on text-based games. In *NeurIPS 2020*. ACM.

Leonard Adolphs and Thomas Hofmann. 2019. Ledeepchef: Deep reinforcement learning agent for families of text-based games. *CoRR*, abs/1909.01646.

Prithviraj Ammanabrolu and Mark Riedl. 2019a. Playing text-adventure games with graph-based deep reinforcement learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3557–3565, Minneapolis, Minnesota. Association for Computational Linguistics.

Prithviraj Ammanabrolu and Mark Riedl. 2019b. Transfer in deep reinforcement learning using knowledge graphs. In *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, pages 1–10, Hong Kong. Association for Computational Linguistics.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. 2020. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR.

Marc-Alexandre Côté, Ákos Kádár, Xingdi (Eric) Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. 2018. Textworld: A learning environment for text-based games. In *Computer Games Workshop at ICML/IJCAI 2018*, pages 1–29.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. 2018a. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*.

Jesse Farebrother, Marlos C. Machado, and Michael Bowling. 2018b. Generalization and regularization in dqn. In *NeurIPS'18 Deep Reinforcement Learning Workshop*.

Yolanda Gil and Bart Selman. 2019. A 20-year community roadmap for artificial intelligence research in the us. *arXiv preprint arXiv:1908.02624*.

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

Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

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

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

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

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015a. Language understanding for text-based games using deep reinforcement learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal. Association for Computational Linguistics.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. 2015b. Language understanding for text-based games using deep reinforcement learning. In *EMNLP*, pages 1–11.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29.

Philip Osborne, Heido Nõmm, and André Freitas. 2022. A survey of text games for reinforcement learning informed by natural language. *Transactions of the Association for Computational Linguistics*, 10:873–887.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning*, 8(3-4):279–292.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

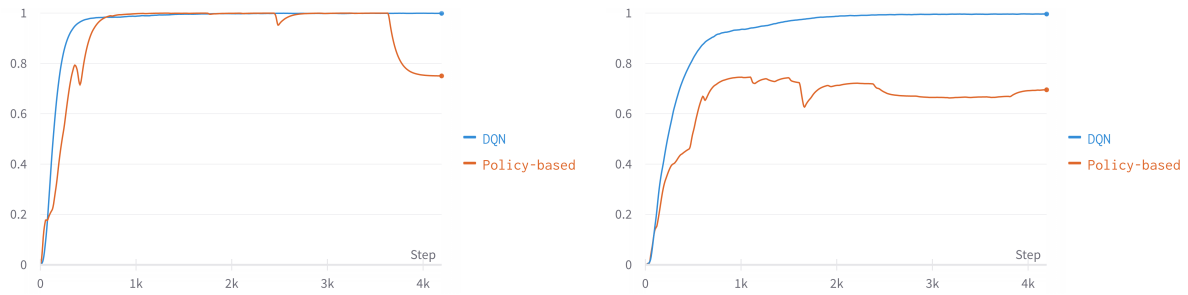
Yunqiu Xu, Meng Fang, Ling Chen, Yali Du, and Chengqi Zhang. 2021. Generalization in text-based games via hierarchical reinforcement learning. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1343–1353, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xusen Yin, Ralph Weischedel, and Jonathan May. 2020. Learning to generalize for sequential decision making. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3046–3063, Online. Association for Computational Linguistics.

Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Chris Pal, Yoshua Bengio, and Adam Trischler. 2019. Interactive language learning by question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2796–2813, Hong Kong, China. Association for Computational Linguistics.

Xingdi (Eric) Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. 2018. Counting to explore and generalize in text-based games. In *European Workshop on Reinforcement Learning (EWRL)*.

A Appendix



(a) Easy Setting

(b) Medium Setting

Figure 5: Coin Collector Average training reward over easy and medium training runs.

Model	Location		Existence		Attribute	
	Train	Test	Train	Test	Train	Test
Fixed						
Random	-	0.027	-	0.497	-	0.496
1 game						
DQN	0.972 (0.972)	0.122 (0.160)	1.000 (0.881)	**0.628 (0.124)**	1.000 (0.049)	0.500 (0.035)
DDQN	0.960 (0.960)	0.156 (0.178)	1.000 (0.647)	0.624 (0.148)	1.000 (0.023)	0.498 (0.033)
Rainbow	0.562 (0.562)	0.164 (0.178)	1.000 (0.187)	0.616 (0.083)	1.000 (0.049)	**0.516 (0.039)**
REINFORCE	1.000 (1.000)	**0.168 (0.172)**	1.000 (0.933)	0.584 (0.217)	1.000 (0.216)	0.514 (0.060)
10 games						
DQN	0.654 (0.654)	0.180 (0.188)	0.822 (0.390)	0.568 (0.156)	1.000 (0.055)	0.518 (0.030)
DDQN	0.608 (0.608)	0.188 (0.208)	0.842 (0.479)	0.566 (0.128)	1.000 (0.064)	0.516 (0.036)
Rainbow	0.616 (0.616)	0.156 (0.170)	0.768 (0.266)	0.590 (0.131)	0.998 (0.059)	**0.520 (0.023)**
REINFORCE	1.000 (1.000)	**0.230 (0.244)**	0.976 (0.820)	**0.654 (0.357)**	0.996 (0.068)	0.498 (0.049)
100 games						
DQN	0.498 (0.498)	0.194 (0.206)	0.756 (0.139)	0.614 (0.160)	0.838 (0.019)	0.498 (0.014)
DDQN	0.456 (0.458)	0.168 (0.196)	0.768 (0.134)	0.650 (0.216)	0.878 (0.020)	0.528 (0.017)
Rainbow	0.340 (0.340)	0.156 (0.160)	0.762 (0.129)	0.602 (0.207)	0.924 (0.044)	0.524 (0.022)
REINFORCE	0.988 (0.988)	**0.786 (0.802)**	0.940 (0.830)	**0.898 (0.768)**	0.958 (0.048)	**0.530 (0.043)**
500 games						
DQN	0.430 (0.430)	0.224 (0.244)	0.742 (0.136)	0.674 (0.279)	0.700 (0.015)	**0.534 (0.014)**
DDQN	0.406 (0.406)	0.218 (0.228)	0.734 (0.173)	0.626 (0.213)	0.714 (0.021)	0.508 (0.026)
Rainbow	0.358 (0.358)	0.190 (0.196)	0.768 (0.187)	0.656 (0.207)	0.736 (0.032)	0.496 (0.029)
REINFORCE	0.990 (0.990)	**0.948 (0.958)**	0.964 (0.916)	**0.948 (0.892)**	0.748 (0.048)	0.466 (0.045)
Unlimited games						
DQN	0.300 (0.300)	0.216 (0.216)	0.752 (0.119)	0.662 (0.246)	0.562 (0.034)	**0.514 (0.016)**
DDQN	0.318 (0.318)	0.258 (0.258)	0.744 (0.168)	0.628 (0.134)	0.572 (0.027)	0.480 (0.024)
Rainbow	0.316 (0.330)	0.280 (0.280)	0.734 (0.157)	0.692 (0.157)	0.566 (0.017)	0.514 (0.014)
REINFORCE	0.986 (0.986)	**0.980 (0.980)**	0.932 (0.828)	**0.902 (0.781)**	0.552 (0.034)	0.462 (0.032)

Table 3: Results of the fixed map QAit experiments. QA accuracy is shown first and sufficient information scores are shown in brackets.

Random						
Model	**Location**		**Existence**		**Attribute**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
Random	-	0.034	-	0.5	-	0.499
10 games						
DQN	0.818 (0.818)	0.156 (0.160)	0.898 (0.607)	0.566 (0.142)	1.000 (0.056)	0.518 (0.036)
DDQN	0.794 (0.794)	0.142 (0.154)	0.868 (0.575)	0.606 (0.153)	1.000 (0.037)	0.500 (0.033)
Rainbow	0.670 (0.670)	0.144 (0.170)	0.828 (0.468)	0.586 (0.128)	1.000 (0.071)	0.530 (0.018)
REINFORCE	0.924 (0.924)	**0.266 (0.282)**	0.942 (0.788)	**0.656 (0.317)**	0.958 (0.091)	**0.534 (0.038)**
100 games						
DQN	0.550 (0.550)	0.184 (0.204)	0.758 (0.230)	0.668 (0.181)	0.878 (0.021)	0.524 (0.017)
DDQN	0.524 (0.524)	0.188 (0.204)	0.754 (0.365)	0.662 (0.205)	0.890 (0.025)	0.544 (0.019)
Rainbow	0.442 (0.442)	0.174 (0.184)	0.754 (0.285)	0.654 (0.190)	0.878 (0.044)	0.504 (0.032)
REINFORCE	0.862 (0.866)	**0.492 (0.508)**	0.846 (0.613)	**0.822 (0.471)**	0.952 (0.061)	**0.546 (0.055)**
500 games						
DQN	0.430 (0.430)	0.204 (0.216)	0.752 (0.162)	0.678 (0.214)	0.678 (0.019)	0.530 (0.017)
DDQN	0.458 (0.458)	0.222 (0.246)	0.754 (0.158)	0.656 (0.188)	0.716 (0.024)	0.486 (0.023)
Rainbow	0.370 (0.370)	0.172 (0.178)	0.748 (0.275)	0.678 (0.191)	0.636 (0.020)	0.494 (0.017)
REINFORCE	0.818 (0.818)	**0.570 (0.588)**	0.866 (0.628)	**0.836 (0.560)**	0.754 (0.045)	**0.534 (0.044)**
Unlimited games						
DQN	0.316 (0.316)	0.188 (0.188)	0.728 (0.213)	0.668 (0.218)	0.812 (0.055)	**0.506 (0.018)**
DDQN	0.326 (0.326)	0.206 (0.206)	0.740 (0.246)	0.694 (0.196)	0.580 (0.023)	0.482 (0.017)
Rainbow	0.340 (0.340)	0.258 (0.258)	0.728 (0.210)	0.686 (0.193)	0.564 (0.018)	0.470 (0.017)
REINFORCE	0.792 (0.794)	**0.738 (0.738)**	0.860 (0.624)	**0.858 (0.584)**	0.550 (0.043)	0.502 (0.042)

Table 4: Results of the random map QAit experiments. QA accuracy is shown first and sufficient information scores are shown in brackets.

	Fixed Map	Random Map
Actions / Game	17	17
Modifiers / Game	18.5	17.7
Objects / Game	26.7	27.5
# Obs. Tokens	93.1	89.7

Table 5: Statistics of the QAit dataset. Numbers are averaged over 10,000 randomly sampled games. (Yuan et al., 2019)

Map Type:	**Fixed**	**Random**
pantry	68	39
livingroom	87	34
shed	89	44
inventory	27	32
corridor	79	41
bedroom	75	32
driveway	75	38
street	-	38
bathroom	-	46
supermarket	-	29
garden	-	38
backyard	-	51
driveway	-	38

Table 6: QAit Answer distribution for location type questions

Map Type:	**Fixed**	**Random**
yes	252	237
no	248	263

Table 7: QAit Answer distribution for existence type questions.

Map Type:	**Fixed**	**Random**
yes	242	236
no	258	264

Table 8: QAit Answer distribution for attribute type questions.