An Exploration of Count-Based Exploration Methods in Text-Based Games: Final Report

G003 (s1837379, s1555620, s1554724)

Abstract

Teaching a reinforcement learning agent to solve text-based games is a challenging task. Actions in text-based games are free-form textual commands constructed out of a large dictionary of words, leading to an immense action space. In addition, the environment provides the agent only with sparse rewards in the form of game wins or losses, making it difficult to find winning policies. In this situation, a simple epsilon-greedy policy with epsilon annealing is not sufficient to encourage enough targeted exploration of the state and action spaces. Therefore, more effective exploration mechanisms are required to aid the discovery of optimal policies for the agent, which allow it to solve the game. We propose a reinforcement learning agent for the text-based game environment with an episodic counting-based exploration mechanism which helps it to discover action sequences that yield the final reward more quickly. We compare our mechanism with other counting-based exploration mechanisms, which, to the best of our knowledge, have not been applied to text-based game environments with comparably large action spaces. Because of the extensive compute time required to run our experiments, we are unable to make definite conclusions about the effectiveness of our proposed agent. However, initial results look promising.

1. Introduction

Humans and computers are increasingly collaborating through conversation-based mechanisms; in recent years, this trend can be observed in both consumer- and business-facing applications such as voice assistants and chatbots. These products must all understand a user's intent from their natural language prompt, and then decide on an action to take that can satisfy this intent. For example, a voice assistant may hear a prompt to "clean up the kitchen," based on which it must then understand that the user is attempting to invoke its home automation capabilities, and decide to take the correct action: to activate the robot vacuum cleaner and send it to the area labeled "kitchen."

One interesting scenario in which this mechanism of decision making based on natural language is required is Interactive Fiction (IF), more commonly known as text-based games (Zelinka, 2018). Text-based games are computer programs in which a player has to complete a quest purely through free-form textual descriptions and commands. The player is prompted with a description of a scenario (e.g. "You are in a dimly lit room. To your left, you see a switch on the wall.") and responds with an action ("Flip the switch."). The game then responds with a new description ("The room is now illuminated, and you see a doorway to your left."), and the process repeats. In more complex examples of IF, a player may need to explore the different areas and objects in the game first, before she can go on to reason about how to use this information to complete each step in the game's quest.

Creating agents to play text-based games has been an active area of artificial intelligence research in recent years, touching both the research areas of Natural Language Processing for understanding descriptions and potential commands, and Reinforcement Learning for deciding which action to take. Existing approaches such as (Côté et al., 2018) use Long Short-Term Memory (LSTM) networks (Sundermeyer et al., 2012) to process the natural-language game feedback, along with Deep-Q Networks (DQNs) (Mnih et al., 2013) to assign values to individual words, which they then use to form the next command to play.

However, in the environment of a complex text game with multiple steps required to complete a quest, these methods fall short. Related work on simple text-based games has proposed several techniques for encouraging exploration to tackle this problem, such as (Yuan et al., 2018a) which assigns bonuses to previously unexplored world locations. In this project, we explore both existing techniques and one novel technique to encourage such exploration in the setting of the complex TextWorld cooking IF game (Côté et al., 2018). Specifically, we compare the effects of a cumulative state counting bonus, (DQN-S+) and an episodic state counting bonus (DQN-S++) (Yuan et al., 2018a); model-based interval estimation-exploration bonus (DQN-MBIE-EB) and upper-confidence bound state-action counting (DQN-UCB-SA) (Strehl & Littman,

2008); and our novel approach, episodic k-means clustered counting bonus (DQN-KM++), inspired by (Abel et al., 2016). Although the exptensive compute time required to run experiments prevents us to make definitive conclusions about the effectiveness of our methods, our initial results are promising.

The rest of this report is structured as follows. In Section 2, we explain the text game reinforcement learning task and the TextWorld environment; in Section 3, we discuss previous work in encouraging exploration in text-based games; in Section 4, we discuss our specific implementation of each technique; in Section 5, we detail our experimental setup and results; and finally, in Section 6 we present our conclusions.

2. Data set and task

The text-based game is a problem of a sequential decision making where the input and output of the problem is presented in natural language. These problems can be thought of as discrete-time Partially Observable Markov Decision Processes (POMDPs) (Kaelbling et al., 1998). That is because the descriptive text does not present complete information about the environment or may do it ambiguously (Côté et al., 2018).

We adopt the formal definition of text-based games from (Zelinka, 2018), reproduced in Definition 1.

Definition 1 *Definition 1 (Text-based game). Let us define a text game as a tuple* $G = \langle H, H_t, S, A, D, T, R \rangle$, *where*

- H *is a set of game states,* H^* *is a set of terminating game states,* $H^* \subseteq H$,
- O *is a set of possible state descriptions,*
- A *is a set of possible action descriptions,*
- D *is a function generating text descriptions,* $D : H \rightarrow (S \times 2^A)$,
- T *is a transition function,* $T : (H \times A) \rightarrow H$,
- R *is a reward function,* $R : (S_t, A_t, S_{t+1}) \rightarrow \mathbb{R}$.

To solve such reinforcement learning problems we aim to learn an optimal policy: a mapping from states and to best next actions that would allow us to maximise the reward. When selecting actions, the action-value function or Q-function $Q : (O \times A) \rightarrow \mathbb{R}$ is used to estimate the total (discounted) reward indicated by $Q(o, a)$ that the agent will receive by taking the action a from the (current) state s and following his policy. Similarly, the value function $V : O \rightarrow \mathbb{R}$ approximates the total expected (discounted) reward - $V(o)$ that the agent will receive by following his policy from the observation o.

We chose to use TextWord (Côté et al., 2018) - Microsoft's Open Source project, as our learning environment for training and evaluating our Reinforcement Learning agents on the text-based games.

Environment States: The current environment state $h_t \in H$ at time t holds a complete information about that state including its description o_t which is deterministic in TextWorld implementation, however different underlying states may have the same state description. Transition function $T : (H \times A) \rightarrow H$ is fully deterministic in TextWorld implementation and always gets applied with a probability of 1.

Actions: At each discrete timestamp t, the agent issues a command a_t. Only a small subset of all vocabulary permutations are in A and subsequently change the world state H. The resulting action space is immense and unmanageable for existing RL algorithms.

Observations: The text output received by the agent after taking an action a from state h is an observation o. Note that multiple different observations can be returned from the same state depending on a recent inadmissible action one took.

Reward function: Certain set of actions will an lead agent to some $h_t^* \in H^*$ where agent will receive a reward of 1. TextWorld has implemented functionality which can assign intermediary rewards during training, however we did not use this feature.

3. Related work

RL in Text-based Games: The Long-Short-Term-Memory Deep-Q-Network (LSTM-DQN) model has been introduced by (Narasimhan et al., 2015) and been tested on two text-based environments. (He et al., 2015) introduced an alternative Deep Reinforcement Relevance Network which focused on choice-based games and show that the model is extracting meaning rather than simply memorizing strings of text. The drawback of choice based games is that model must know addmisable actions at each state which makes the problem easier.

Counting-based Exploration: There have been multiple counting-based approaches introduced to encourage exploration. One of the earlier attempts include (Strehl & Littman, 2008) which propose counting n(s, a) state-action pairs and adding a bonus to reward in the form of $\frac{\beta}{\sqrt{n(s,a)}}$ which was later showed by (Kolter & Ng, 2009) that inverse-square-root-dependence is optimal. (Yuan et al., 2018a), on the other hand, argued that due to the fact that most actions does not change underlying environment's state such counting is counter-productive.

(Yuan et al., 2018b) has introduced the idea of *cumulative bonus* $\beta \times n(o_t)^{-1/3}$ where $n(o_t)$ is the number of times agent has observed o_t from the beginning and *episodic discovery bonus* where bonus is set to β when $n(o_t) = 0$ and 0 otherwise. All the counts are reset at the beginning of each period. These approaches were shown to generalise well on unseen games, however it must be noted that they limited their action space to

10 actions and their experiments did not require interacting with objects of the level to achieve sub-goals; such as opening a locked door with a key card that the agent has picked up, which is otherwise, a standard feature of text-based games.

To further foster development in the promising direction we propose new discovery bonus based on state representation unsupervised clustering.

4. Methodology

4.1. Model Architecture

In this paper, we adopt LSTM-DQN (Narasimhan et al., 2015) architectures for our baseline. This architecture is commonly chosen as the baseline for text-based environments (Zelinka, 2018).

The LSTM part of the LSTM-DQN is often called observation encoder or observation representation generator denoted Φ_R. Mean pooling is used to get the observation representing vector. On top of the LSTM network sits a fully connected MLP which computes the Q values—hence called DQN, often referred to as action scorer, denoted Φ_A. Therefore, the full network structure can be expressed as $\Phi_A(\Phi_R(o))$ where o is an observation from the environment.

The output of LSTM-DQN are Q values which span the full vocabulary of the environment, in our case 20200 words. To construct an action command string 2 nouns, 2 adjectives and a verb are chosen having highest Q values in each set and concatenated in a form

action ← *"[verb] [adjective_1] [noun_1] [preposition] [adjective_2] [noun_2]"*

Prepositions for specific verbs are hard coded, for example- take: from, pick: from, chop: on, etc. This allows to construct commands such as *"take red potato from blue table"*. There is also special symbols ϵ, which signifies and empty word, and κ, which signifies the end of sentence which are added to the dictionary and are included in the the sets of nouns, verbs and adjectives. This allows to generate *"Go east"* when *adjective_1* is set to ϵ and *adjective_2* is set to κ.

The baseline algorithm which we use for our experiments is described in algorithm 1, we augment this algorithm with methods described below to encourage exploration.

4.2. Promoting Exploration

To promote exploration we use an intrinsic reward by counting state visits and approximating the confidence of our estimated Q values. We compare five approaches which have not been applied on IF environment with unconstrained action spaces. Intrinsic counting state and state action reward methods can be can be split into two categories. First is count-based exploration via counting observations, where predetermined simple hashing function are applied on the observation which is then compared to a list of the ones already seen by the agent to determine the reward. Second is count-based exploration via a learned hashing, where the hashing function is learned, which propose as method that to our best knowledge has not been applied in this setting.

Algorithm 1 LSTM-DQN

Initialize prioritized replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} := \theta$
for episode 1 to M **do**
  Preprocessed sequence $\Phi_1 = \Phi_R(o_1)$
  for t = 1 to T **do**
    With probability ε select a random action a_t
    Otherwise construct action my selecting verb, 2 nouns and 2 adjectives according to $a_t = \arg\max_w(Q(\Phi_R(o_t), w; \theta)$, where for each domain (verb, noun, adjective) only words depending to that domain are considered.
    Execute action a_t in environment and observe reward r_t and observation o_{t+1}
    Preprocess $\phi_{t+1} = \Phi_R(o_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

$$\text{Set } y_j = \begin{cases} r_j \text{ if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) \text{ otherwise} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  end for
end for

4.2.1. Upper Confidence Bound

One exploration promotion method is using UCB (Upper Confidence Bound), which gives state of the art results on for multi arm bandit problem in reinforcement learning (Tang et al., 2017). Intuitively, we are adding an uncertainty term to the Q value which models expected (discounted) reward under current policy.

$$\arg\max_w(Q(o,w) + \frac{\beta}{\sqrt{n(o,w)+1}}) \quad (1)$$

This gives us some notion of an upper bound for actions consisting of words which weren't picked often with this particular observation o. Here β is a hyperparameter which represents how uncertain about his expected reward the agent is, clearly as $n(o,w) \to \infty$ then

$(Q(o,w) + \frac{\beta}{\sqrt{n(o,w)+1}}) \rightarrow Q(o,w)$, the more certain the agent is about his expected reward.

We call model with this method for exploration DQN-UCB-SA. Model is constructed by replacing line 8 in the algorithm 1 with equation (1). This allows us to pick 5 words and construct and action string by the schema described in Section 4.1.

4.2.2. Count-Based Exploration via Counting States

(Yuan et al., 2018a) propose state counting functions:

$$R^{+} = \beta \cdot n(o)^{\frac{-1}{3}}$$

$$R^{++} = \begin{cases} \beta & \text{if } n(o) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where $n(\cdot)$ is reset to zero each episode

We used a simple hashing algorithm which simply maps every observation, which is just a short paragraph to a numerical value, where exactly the same peace of text is mapped to the same value. Models with these methods are called DQN-S+ and DQN-S++. It is important to note, however, that even though in their paper (Yuan et al., 2018a) they are evaluating on a text-base game environment, they limit the vocabulary to just 6 nouns and 2 verbs compared to a dictionary of a size 20200 for a standard game, and use simplified descriptions. Therefore we think it is important to see how these methods perform when the action space is many times larger. We have also implemented a closely related exploration bonus which counts state-action, in our case observation-word pairs and is of the form $R_{sa} = \beta \cdot n(o,w)^{\frac{-1}{2}}$. For construct all of these architectures we change the line 11 of algorithm 1 to store transition $(\phi_t, a_t, r_t + R', \phi_{t+1})$ in D where $R' \in \{R+, R++, R_{sa}\}$.

4.2.3. Count-Based Exploration via Learned Hashing

In this section we propose a novel observation counting method which borrows from ideas of learned hashing (Strehl & Littman, 2008) where we take an output of the LSTM part of the network $\Phi(o)$ and treat it as a observation encoding. However, simply rewarding agent for different observation encoding (lets say by measuring euclidean distance between a new one an the ones agent has already seen) might lead to unintended consequences when the LSTM simply learns to encode states very sparsely. Instead we build on the idea from (Abel et al., 2016) where they used supervised clustering to classify observation representations. We proposed using unsupervised clustering with K-Means to map observation representations to specific clusters, and rewards agent when he sees a state that has not been classified belonging to a cluster yet this episode. Our algorithm might be summed up in the following.

$$R_{KM} = \begin{cases} \beta & \text{if } n(K\_Means.predict(\phi(o)) = 1 \\ 0 & \text{otherwise} \end{cases}$$

where $n(\cdot)$ is reset to zero each episode

Model with these method are called DQN-KM++. To construct this model we change the line 11 of algorithm 1 to store transition $(\phi_t, a_t, r_t + R_{KM}, \phi_{t+1})$ in D where $R' \in \{R+, R++, R_{sa}\}$.To not have the KMeans cluster centers be too dynamic, KMeans is only retrained every epoch with all of the new observation encodings seen in the previous epoch.

5. Experiments

In our experiments, we aim to find out whether our different counting models could encourage exploration in the large state and action spaces in the TextWorld cooking game. We hypothesize that models with better exploration capabilities are able to do better at longer, more complex games. We generate games of various lengths to test whether our counting models outperform the baseline on longer games. Our source code is on GitHub[1].

5.1. Game Generation

The TextWorld learning environment (Côté et al., 2018) can be used to generate cooking games in which the player (or, in our case, agent) has to collect items (tools and ingredients) across different rooms and use them to cook a meal. The difficulty of these games can be adjusted along the following axes:

- **World size**: Across how many rooms the objects can be located.
- **Number of objects**: How many tools and ingredients are in the game.
- **Quest length**: The number of steps in the recipe that the player must follow to win the game.

We vary the quest length from 1 to 4, and keep the number of objects constant at 4 and the world size constant at 2. Initial experiments showed that these parameters result in games of reasonable difficulty for the baseline, with room for improvement for our models. We generate multiple games for each quest length, using different seeds, , in order to create confidence bounds around our results. We used seeds 1234, 3345 and 6516 for all quest lengths. Additionally, because only three seeds yielded high variance for higher quest lengths, we generated additional experiments with seed 9632 for quest lengths 3 and 4, and an additional game with seed 8888 for quest length 4.

[1] https://github.com/MariusUrbonas/long-short-term-memes

Model	QL 1	QL 2	QL 3	QL 4
DQN (Baseline)	0.99 ± 0.03	0.96 ± 0.06	0.49 ± 0.45	0.27 ± 0.38
DQN-UCB-SA	0.99 ± 0.03	0.95 ± 0.07	0.46 ± 0.43	0.09 ± 0.18
DQN-MBIE-EB	0.99 ± 0.04	0.94 ± 0.08	0.49 ± 0.46	0.09 ± 0.18
DQN-S+	0.93 ± 0.10	0.63 ± 0.47	0.00 ± 0.02	0.00 ± 0.00
DQN-S++	0.98 ± 0.04	0.90 ± 0.13	0.21 ± 0.33	0.00 ± 0.00
DQN-KM++	0.98 ± 0.03	0.96 ± 0.05	0.46 ± 0.48	0.42 ± 0.45

Table 1. Mean scores with error bounds for our models trained on games with quest lengths 1 through 4.

.

The following TextWorld command[2] can be used to generate one such game:

```
tw-make custom --world-size 2 --nb-
    ↪ objects 4 --quest-length 1 --seed
    ↪ 1234 --output game_name.ulx
```

We created a script to automate the process of generating these games and keep all the configuration files, generated games, model checkpoints, run scripts, and training statistics in one place. To use it, call the following from our repository's root directory, passing the folder containing the code for the agent and a name for the experiment:

```
python generate_experiments.py
    ↪ model_folder exp_name
```

This creates a folder `experiments/exp_name` with the following sub-folders:

- `config`: a config file for each quest length / seed combination, with appropriate paths and files names for saving model checkpoints and statistics.
- `games`: the files generated by `tw-make`.
- `models`: empty; will be used during training to save model checkpoints.
- `scripts`: a bash script for each quest length / seed combination that submits a slurm[3] job to run `train.py` for the model in `model_folder`.
- `stats` empty; will be used after training to save statistics.

Finally, the script `run_scripts.py` can be called with an experiment's script folder as argument to run all the scripts in that folder and submit the appropriate slurm jobs.

5.2. Training

We trained all our models in the same way: training for 400 epochs with batches of 16 parallel games running for a maximum of 100 steps per game; we determined these parameters using initial small-scale experiments on the baseline, as before.

For the rest of the LSTM-DQN parameters, we kept the default settings from the implementation by (Côté et al., 2018). The LSTM has an embedding size of 64; the encoder has an RNN hidden unit size of 192; the action scorer has a hidden dimension of 128; and the dropout is 0. The optimizer used is Adam (Kingma & Ba, 2014) optimizer, with learnig rate 0.001 and gradient clipping norm 5. For the epsilon-greedy policy, we anneal ϵ from 1.0 down to 0.2 over 300 episodes. The random seed for the algorithm is 42. These and other (less interesting) parameters can be found in our `base_config.yaml` file.

Our algorithms all use a parameter β to encourage exploration, as explained in Section 4. Across all experiments, we set β to 0.001. Because all our games run for a maximum of 100 steps, the agent would be able to gain a reward of up to $100 \times 0.001 = 0.1$[4], compared to a reward of 1 for winning a game. This way, we hope to discourage the agent from focusing on exploring once it learns how to solve the game.

5.3. Results

In the following subsections we report and interpret the results of running our different models on the games described above. These results are summarized in Table 1 and Figures 1 and 2.

For Table 1 and Figure 2, we calculate error bounds as follows. For each episode, we take the standard deviation of the score for different seeds for each quest length and different runs within each batch, and average this over all 400 or 50 episodes, respectively. These error bounds are quite large; we explain why this occurs in the caption of Figure 1, along with why the learning curves for quest length 3 and 4 exhibit a step pattern.

We hoped to run more seeds to decrease the size of these bounds and the significance of this step pattern, but this was not possible due to the extensive time it takes to train each game. Across a laptop with a 2018 i9 Intel CPU, the MLP cluster, and DICE, models take between 6 and 10 hours to train. Given that we have 6 models $\times 4$ quest lengths $= 24$ experiments, running an extra seed for each of them takes somewhere between 144 and 200 compute hours.

[2] For installation instructions, see: https://github.com/Microsoft/TextWorld.

[3] https://slurm.schedmd.com/

[4] Or even less for some of our algorithms.

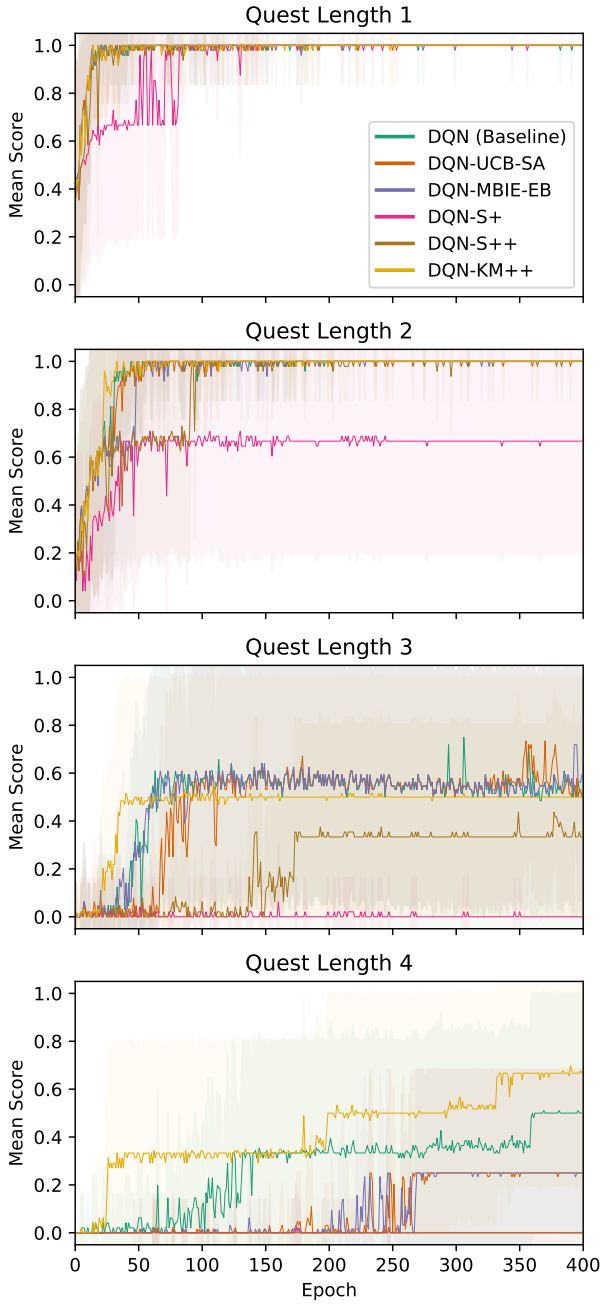

Figure 1. Learning curves for our models trained on games of quest lengths 1 through 4. Error bounds at each epoch represent the standard deviation of scores across different game generation seeds and different parallel game runs within a training batch. The large magnitude of these bars can be explained by batches failing to learn to solve the game for some generation seeds. For instance, DQN-S+ on quest length 2 failed to learn for one of three seeds (resulting in batch mean scores of approx 1, 1 and 0), so its mean score ended up around $\frac{1+1+0}{3} \approx 0.67$ and its standard deviation was around $\frac{|1-0.67|+|1-0.67|+|0-0.67|}{3} \approx 0.44$. This same effect also happens for most models trained on quest lengths 3 and 4. The step pattern in this graph is caused by batches of game runs starting to "figure out" the game; once they do this, they score 1 on most epochs and push the overall average up to a new plateau.

5.4. Interpretation

As mentioned in section Section 5.3, high variance in our results does not allow to give definite conclusions about the performance of each model, since most of the error bounds largely overlap. That being said, we give an interpretation to the trends that our current data shows.

Overall, the trend is that for longer quest length (quest length 4) DQN-S+, DQN-S++, DQN-MBIE-EB and DQN-UCB-SA take longer to learn a policy which gives allows to complete the game than the baseline. This, however, is expected as the action space is large, and different observations don't correspond to different underlying states of the environment as specified in Section 2. Therefore, in a case of DQN-S+, DQN-S++, DQN-UCB-SA initially inadmissible actions are highly rewarded as there are many ways that the environment can say that the actions is inadmissible. This results in agent prioritizing sub-optimal policies which is reflected in slower learning time.

The DQN-MBIE-EB agent suffers from a similar pitfall: the action and observation spaces are large therefore lowering the uncertainty bound takes a long time.

In comparison, the DQN-KM++ agent seems to find the winning state faster compared to the baseline which is why it begins to get positive rewards after around 30 epochs - as shown in graphs for quest lengths 3 and 4 (see Figure 1). After further inspection into the values, we observed that the LSTM learns to encode observations, specifically the ones which imply that the action is not admissible, close in space, therefore KMeans algorithm assigns it to the same cluster, which results in agent not getting additional rewards for inadmissible actions even though the observations are slightly different.

All agents unfortunately but unsurprisingly did not generalise to unseen games (we omit this graph as it does not convey anything meaningful) and a much larger scale of experiments would be needed to safely conclude if it is capable of doing so. We attribute it mostly to the lack of relevant words and formulations between the games.

6. Conclusions and Future Work

We presented a minimalist text-game playing agent capable of learning to play short length text-based games. The agent uses unsupervised clustering to decide whether the observations are a lot different to the ones it has already seen and prefers actions which take it to such unseen observations. This method compared to similar state and action counting methods seems to show promising results, but we are unable to give definite conclusions whether it is better because of time constraints that prevented us from running additional experiments.

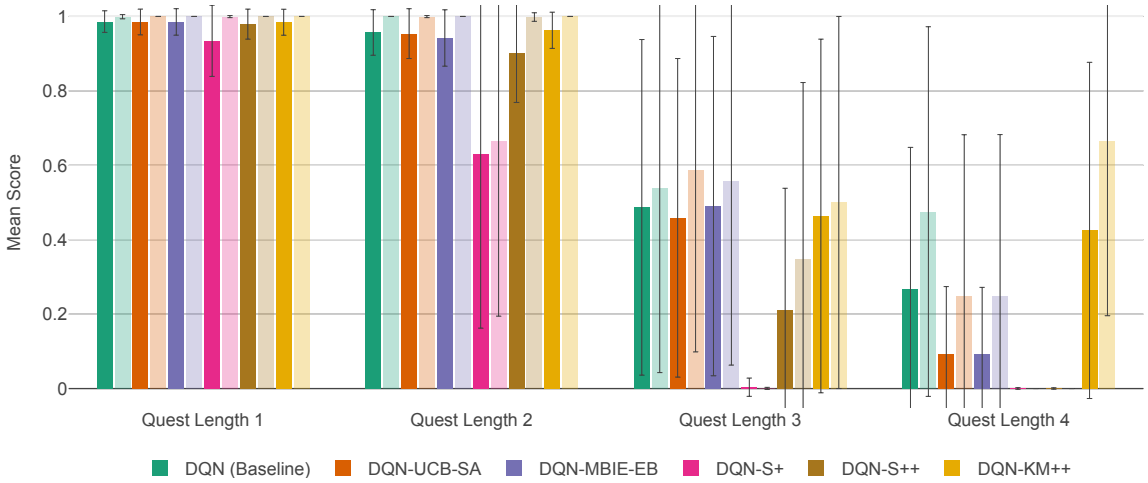

Figure 2. Mean scores with error bounds for our different models trained on games with quest lengths 1 through 4. Darker-shade bars represent means over the full 400 episodes; lighter-shade bars represent means over the last 50 episodes.

Future work should mainly focus on expanding the text-based game domain and on conducting experiments at a much larger scale. We hypothesise that even agents as simple as the DQN-KM++ agent should show some generalisation capabilities given enough data. It would also be beneficial to explore how these exploration mechanisms effect more complicated models such as LSTM-DRQN (Long Short Term Memory - Deep Recurrent Q-Network) (Yuan et al., 2018a).

In the end, the results also imply that simple algorithms without additional preprocessing are not sufficient for solving full-scale text-based games such as Zorg, as the sequences of actions to complete these kind of games are many times larger. We hypothesise that to make our discussed methods feasible for more complex games, pretraining the LSTM network which encodes a set given of observations might be useful, similarly additional methods should be applied to lower the action space, such as training a separate network to recognise valid action structures to avoid inadmissible commands such as "Eat west". These tests can serve as additional future work.

References

Abel, David, Agarwal, Alekh, Diaz, Fernando, Krishnamurthy, Akshay, and Schapire, Robert E. Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv preprint arXiv:1603.04119*, 2016.

Côté, Marc-Alexandre, Kádár, Ákos, Yuan, Xingdi, Kybartas, Ben, Barnes, Tavian, Fine, Emery, Moore, James, Hausknecht, Matthew, Asri, Layla El, Adada, Mahmoud, Tay, Wendy, and Trischler, Adam. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018.

He, Ji, Chen, Jianshu, He, Xiaodong, Gao, Jianfeng, Li, Lihong, Deng, Li, and Ostendorf, Mari. Deep reinforcement learning with an unbounded action space. *CoRR*, abs/1511.04636, 2015. URL http://arxiv.org/abs/1511.04636.

Kaelbling, Leslie Pack, Littman, Michael L, and Cassandra, Anthony R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kolter, J Zico and Ng, Andrew Y. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 513–520. ACM, 2009.

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

Narasimhan, Karthik, Kulkarni, Tejas D., and Barzilay, Regina. Language understanding for text-based games using deep reinforcement learning. *CoRR*, abs/1506.08941, 2015. URL http://arxiv.org/abs/1506.08941.

Strehl, Alexander L and Littman, Michael L. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Sundermeyer, Martin, Schlüter, Ralf, and Ney, Hermann. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

Tang, Haoran, Houthooft, Rein, Foote, Davis, Stooke, Adam, Chen, OpenAI Xi, Duan, Yan, Schulman,

John, DeTurck, Filip, and Abbeel, Pieter. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pp. 2753–2762, 2017.

Yuan, Xingdi, Côté, Marc-Alexandre, Sordoni, Alessandro, Laroche, Romain, Combes, Remi Tachet des, Hausknecht, Matthew, and Trischler, Adam. Counting to explore and generalize in text-based games. *arXiv preprint arXiv:1806.11525*, 2018a.

Yuan, Xingdi, Côté, Marc-Alexandre, Sordoni, Alessandro, Laroche, Romain, des Combes, Remi Tachet, Hausknecht, Matthew J., and Trischler, Adam. Counting to explore and generalize in text-based games. *CoRR*, abs/1806.11525, 2018b. URL http://arxiv.org/abs/1806.11525.

Zelinka, Mikuláš. Baselines for reinforcement learning in text games. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 320–327. IEEE, 2018.