Potential-based reward shaping for learning to play text-based adventure games

Anonymous ACL submission

Abstract

Text-based games are an optimal testbed for language-based reinforcement learning (RL). In previous work, deep Q-learning is commonly used as the learning agent. Q-learning algorithms are challenging to apply to complex real-world domains due to, for example, their instability in training. Therefore, in this paper, we adapt the soft-actor-critic (SAC) algorithm to the text-based environment. To deal with sparse extrinsic rewards from the environment, we propose a potential-based reward shaping technique to provide more informative (dense) reward signals to the RL agent. We apply our method to play difficult text-based games. Our SAC method achieves higher scores than the Q-learning methods on many games with only half the number of training steps. This shows that it is well-suited for text-based games. Moreover, we show that the reward shaping technique helps the agent to learn the policy faster and achieve higher scores.

1 Introduction

Language-based interactions are an integral part of our everyday life. Reinforcement learning (RL) is a promising technique for developing agents that acting in real-life scenarios such as dialog systems. However, training these agents is difficult due to missing feedback or reward signals. Because of this, text-based adventure games are an ideal benchmark for developing language-based agents (Hausknecht et al., 2020). In games, the players receive automatic rewards from the game environment and we can use the final game score for comparing performances of different agents.

Figure 1 illustrates the problem setup for this paper. One main difference between text-based adventure games and other RL scenarios is the large and discrete action space. In contrast to other games (e.g., ATARI games), each action is characterized by a sentence or word (e.g., climb tree). Also, the action space is not fixed. For example, if the agent is in front of the house, the action "open door" is available, whereas if the agent is in the forest, other actions are possible, e.g. "climb tree", but not "open tree". Therefore, in addition to the action space, there is the space of valid actions in the current state (see Figure 1 for an example of gameplay in the game zork3). This space is much smaller than the space of all actions but can be significantly different in each step. In general, this space of valid actions is unknown to the agent, but a common simplification is to let the agent have the list of valid actions as input. A number of prior works in this domain focused on the above-mentioned challenges (Yao et al., 2020; Ammanabrolu and Hausknecht, 2020; Ammanabrolu et al., 2020; Guo et al., 2020; Xu et al., 2020). Most of those works used deep Q-learning as a learning agent.

Deep Q-learning has several drawbacks. As an off-policy algorithm, it suffers from high variance, and the performance can be unstable (Sutton and Barto, 2018). Other online, policy-based learning algorithms are also unsuitable for our scenario since the agent needs to reuse experiences from the training history. Therefore, in this paper, we develop a learning agent based on the soft actor critic (SAC) algorithm (Haarnoja et al., 2018), which combines both value-based and policy-based learning. Additionally, the maximum entropy technique encourages *stability* and *exploration*. SAC was originally designed for continuous action spaces; however, with slight modifications, it is applicable for discrete action spaces (Christodoulou, 2019). Nevertheless, it has never been applied to text-based adventure games before.

A problem that text-based adventure games have in common with many other RL problems is the sparseness of rewards. Especially at the beginning of training, the agent needs to perform many actions before receiving feedback. In text-based adventure games, this problem is even more severe due to the large and context-dependent action space.

Valid action spaces: ['turn off lamp', 'put down lamp', 'west', 'throw rock at lamp', 'east', 'northwest', 'southwest']

Info: This is a remarkable spot in the dungeon. Perhaps two hundred feet above you is a gaping hole in the earth's surface through which pours bright sunshine! A few seedlings from the world above, nurtured by the sunlight and occasional rains, have grown into giant trees, making this a virtual oasis in the desert of the Underground Empire. To the west is a sheer precipice, dropping nearly fifty feet to jagged rocks below. The way south is barred by a forbidding stone wall, crumbling from age. There is a jagged opening in the wall to the southwest, through which leaks a fine mist. The land to the east looks lifeless and barren. A rope is tied to one of the large trees here and is dangling over the side of the cliff, reaching down to the shelf below. It seems as if somebody has been here recently, as there is some fresh bread lying beneath one of the other trees.

Action(predicted by RL agent)**:** ***west***

reward: 0, score: 0

Valid action spaces: ['turn off lamp', 'take waybread', 'down', 'jump across cliff', 'put down lamp', 'east', 'throw waybread at lamp', 'throw lamp off cliff', 'southwest']

Info: Cliff Ledge. This is a rock-strewn ledge near the base of a tall cliff. The bottom of the cliff is another fifteen feet below. You have little hope of climbing up the cliff face, but you might be able to scramble down from here (though it's doubtful you could return). A long piece of rope is dangling down from the top of the cliff and is within your reach. A large chest, closed and locked, is lying among the boulders

Action(predicted by RL agent)**:** ***down***

Reward:1, **Score: 1**

Figure 1: This figure shows an example of gameplay for the game Zork3. The RL agent receives the valid action space, state information, reward, and score from the Jericho environment. The agent then needs to predict the action and move to the next state.

To speed up the convergence, it is therefore desirable to have a denser reward function. A popular way to achieve this is through reward shaping.

Finding a good reward function is difficult and requires significant manual effort, background information, or expert knowledge. A well-known reward shaping technique, circumventing the need for external knowledge, is potential-based reward shaping (Ng et al., 1999) which has strong theoretical guarantees. This enables faster convergence at the beginning of training which we show for several of the difficult games.

To sum up, our contributions are as follows:

1. We propose to use SAC as an alternative to deep Q-learning for text-based adventure games.

2. We propose a variant of potential-based reward shaping for discrete action spaces that is effective for text-based adventure games.

3. We compare our method on a range of difficult games and show that we can achieve better scores than deep Q-learning with fewer training episodes on many of the games.

4. Additionally, we show that convergence is faster with reward shaping.

2 Related work

Text-based adventure games In general, for text-based adventure games, there are choice-based agents and parser-based agents (Hausknecht et al., 2020). Other related work focuses not on the RL agent but on action generation (Ammanabrolu and Hausknecht, 2020; Yao et al., 2020; Ammanabrolu et al., 2020; Xu et al., 2020; Guo et al., 2020). Parser-based agents (Narasimhan et al., 2015) generate actions using verb-object combinations, whereas choice-based agents choose an action from a pre-generated list of actions. In this work, we follow the line of choice-based agents which is a simplification that allows us to concentrate on the RL part of our method.

Deep reinforcement relevance network (DRRN) (He et al., 2016) is one main choice-based method. The basic idea behind DRRN is encoding the actions and states into embedding vectors separately, and then the state and its corresponding actions embed vectors as inputs into a neural network to approximate the Q values of all possible actions $Q(s_t, a_t^i)$. The action at each time step is selected by $a_t = argmax_{a_t^i}(Q(s_t, a_t^i))$.

Hausknecht et al. (2020) built the Jericho Interactive Fiction environment which includes 57 different games that are categorized into possible, difficult, and extreme games. In this work, we focus on the difficult games that were compared by Hausknecht et al. (2020) because they tend to have sparser rewards than the possible games. The difficult games still include several games where no method has been able to achieve a score higher than a random agent to date.

NAIL (Hausknecht et al., 2019) is an agent, which is not choice-based, that is trained to play any unseen text-based game without training or repeated interaction and without receiving a list of valid actions. We compare both DRRN (and variants) and NAIL in our experiments, but only DRRN has the exact same experimental setup and handicaps as our agent. NAIL serves as a baseline of scores possible without any simplifications of gameplay.

Yao et al. (2021) investigate whether the RL agent can make a decision without any semantic understanding. They evaluate three variants based on DRRN: a) only location information is available as observation b) observations and actions are hashed instead of using the pure text c) inverse dynamic loss based vector representations are used. Their results show that the RL agent can achieve high scores in some cases, even without language semantics. We compare to the best results of the three variants.

Soft-actor-critic (Haarnoja et al., 2018) combines both advantages of value-based and policy-based learning. The drawback of value-based learning like deep Q learning is the instability of the performance because the policy can have high variance (Sutton and Barto, 2018). The SAC algorithm includes three elements. The first is separate predict and critic neural networks, the second is that offline learning can reuse the past collections via replay buffer, which is the same as deep Q learning, and the third is that the entropy of the policy is maximized to encourage exploration. The optimal policy aims to find the highest expected rewards and maximize the entropy:

$$\pi^{\star} = \arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[\gamma^t(r(s_t,a_t)+ \alpha\mathcal{H}(\pi(.|s_t)))] \tag{177}$$

The original SAC is evaluated on several continuous control benchmarks. Since we are dealing with discrete text data, we base our method on the framework for discrete action spaces by Christodoulou (2019). The key difference between continuous and discrete action spaces is the computation of the action distribution. For discrete action spaces, it is necessary to compute the probability of each action in the action space. The actor policy is changed from $\pi_\phi(a_t|s_t)$, a distribution over the continuous action space, to $\pi_\phi(s_t)$, a discrete distribution over the discrete action space. In other words, the actor policy is changed from a Gaussian policy to a categorical policy.

Potential-based reward shaping Introduced in the seminal work of Ng et al. (1999), potential-based reward shaping (PBRS) is one of the most well-studied reward design techniques. The shaped reward function is obtained by modifying the reward using a state-dependent potential function. The technique preserves a strong invariance property: a policy π is optimal under shaped reward *iff* it is optimal under extrinsic reward. Furthermore, when using the optimal value function V^* under the original reward function as the potential function, the shaped rewards achieve the maximum possible informativeness. In a large number of prior studies interested in PBRS, Wiewiora et al. (2003) proposes the *state-action potential advice* methods, which not only can estimate a good or bad state, but also can advise action. Grześ and Kudenko (2010) evaluates the ideas of using the online learned value function as a potential function. Moreover, Harutyunyan et al. (2015) introduced an arbitrary reward function by learning a secondary Q-function. They consider the difference between sampled next state-action value and the expected next state-action value as dynamic advice. Based on Harutyunyan et al. (2015) reward shaping technique, Brys et al. (2015) developed the policy transfer to learn the policy from a source task. Devidze et al. (2021) proposed a reward design framework, EXPRD, which interprets two key criteria of a reward function: *informativeness* and *sparseness*.

Reward in NLP based RL agent One of the challenges of using RL to solve natural language processing (NLP) tasks is the difficulty of designing reward functions. There could be more than one factor that affects the rewards, such as semantic understanding and grammatical correction. Li et al. (2016) define reward considering three factors: *ease of answering*, *information flow*, and *semantic coherence* for dialogue generation tasks. Reward shaping techniques have also been used in other NLP-based RL tasks, for example, Lin et al. (2018)

used knowledge-based reward shaping for a multi-hop knowledge graph reasoning task. The agent receives a reward of 1 if the prediction is the correct answer. Otherwise, the agent receives a score computed by the pre-trained knowledge graph embedding $f(e_s.r_q, e_T)$. The core difference to our model is that we do not pre-define any function or knowledge as a reward signal, instead shaping the rewards automatically.

3 Problem setting and background

The experiment agent An environment is defined as a Markov Decision Process (MDP) $M := (\mathcal{S}, \mathcal{A}, T, \gamma, R)$, where the set of states and actions are denoted by $\mathcal{S}$ and $\mathcal{A}$ respectively. $T : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ captures the state transition dynamics, i.e., $T(s' \mid s, a)$ denotes the probability of landing in state s'. the reward R and terminal signal d from the game environment, and γ is the discount factor. The stochastic policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ as a mapping from a state to a probability distribution over actions, i.e., $\sum_a \pi(a|s) = 1$ by a neural network.

Notice that the valid action space size is changeable after each time step. And following Hausknecht et al. (2020), we differentiate between game state s and observation o, where the observation refers only to the text that is output by the game whereas the state corresponds to the locations of players, items, monsters, etc. Our agent has knowledge of the observations and not of the complete game state.

3.1 SAC for discrete action spaces

The SAC algorithm has a separate predictor (actor) and critic. In the following, we first describe the two crucial equations for updating the critic and then the actor policy update.

In the critic part, following the original SAC definition (Haarnoja et al., 2018) and adaptation to the discrete setting by Christodoulou (2019), the targets for the Q-functions are computed by:

$$y(r, s', d) = r + \gamma(1 - d) \left(\left(\min_{i=1,2} \left(Q_{\hat{\theta}_i}(s') \right) - \alpha \log \left(\pi(s'_t) \right) \right) \right) \quad (1)$$

where in our scenario the target Q-values and the policy distribution range over the set of valid actions $A_{valid}(s')$ (Hausknecht et al., 2020). As was proposed by Haarnoja et al. (2018), we use two Q-functions and two Q target functions, and i is the index of the Q-function. γ is a discount factor and $d \in \{0, 1\}$ is 1 if the terminal state has been reached.

The critic optimization is the same as in the original SAC algorithm, learning to minimize the distance between the target soft Q-function and the Q approximation with stochastic gradients:

$$\nabla J_Q(\theta) = \nabla \mathbb{E}_{a \sim \pi(s), s \sim D} \left(Q_{\phi_i}(s) - y(r, s', d) \right)^2, \quad (2)$$

where D is the replay buffer and $i \in \{1, 2\}$. If using double Q-functions, the agent should learn the loss functions of both Q-neural networks.

As proposed by Christodoulou (2019) the update of the actor policy is given by:

$$\nabla J_\pi(\phi) = \nabla \mathbb{E}_{s \sim D} \left[\pi_t(s)^T [\alpha \log \pi_\phi(s) - Q_\theta(s)] \right]. \quad (3)$$

where $Q_\theta(s)$ denotes the actor value by the Q-function (critic policy), $\log \pi_\phi(s)$ and $\pi_t(s)$ are the expected entropy and probability estimate by the actor policy.

As shown in Algorithm 1 in lines 10 and 11, equations 2 and 3 constitute the basic SAC algorithm without reward shaping, where critic and actor are updated in turn. In the next section, we will explain the reward shaping in lines 2–9 of the algorithm.

4 Method

The whole algorithm is given by Algorithm 1. We start by reward shaping in line 2. The shaping reward function $F : S \times A \times S \rightarrow \mathbb{R}$ (Ng et al., 1999) is given by

$$F(s, a, s') = \gamma \Phi(s') - \Phi(s), \quad (4)$$

where s' is the target state and s refers to the source state. As defined in Section 2, when using the optimal value-function V^* under original reward as the potential function, i.e., $\Phi(s) = V^*(s)$, the shaped rewards achieve the maximum possible informativeness.

Dynamic reward shaping

Since we do not have access to the optimal value function V^*, we use the idea of dynamic reward shaping. In particular, Grześ and Kudenko (2010) generalized the form in Equation 4 to dynamic potentials, and empirically showed an advantage in

Algorithm 1 SAC with potential-based reward shaping

Require: policy π; Q-functions $\theta_1, \theta_2, \hat{\theta_1}, \hat{\theta_2}$; replay buffer D; roll-out N
1: **for** step $= 1 \dots$ max step **do**
▷ Update the *critic*:
2: **if** Reward Shaping is True **then**
3: $V_{step}(s) \leftarrow \pi(s)^T \Big[(Q_{\hat{\theta}_i}(s) - \alpha \log(\pi(s))\Big)]$ (Equation 7) ▷ Compute soft state value
4: **for** $i = 1 \dots N$ **do**:
5: $V_{step}(s) \leftarrow (1-\alpha)V_{step}(s) + \alpha(r + \gamma' V_{step}(s'))$ (Equation 8) ▷ Update value function
6: **end for**
7: $F_{step}(s, a, s') \leftarrow \gamma V_{step}(s') - V_{step}(s)$ (Equation 5) ▷ Compute shaping function
8: $\hat{R}(s, a) \leftarrow R(s, a) + F_{step}(s, a, s')$ (Equation 6) ▷ Compute reshaped reward
9: **end if**
10: Update Q-function (Equation 2)
▷ Update the *actor*:
11: Update policy (Equation 3)
12: **end for**

helping the agent. The idea is that the RL agent uses the current approximation of the value function as a potential function. More precisely, the shaped function F_l at learning time step l can be represented as follows (Algorithm 1, line 7):

$$F_l(s, a, s') = \gamma V_l(s') - V_l(s), \tag{5}$$

where $\Phi(s)$ from Equation 5 is given by $V_l(s)$ and superscript l denotes the learning time step. Hence, the new shaped reward $\hat{R} : A \times S \rightarrow \mathbb{R}$ at learning time step l is defined as

$$\hat{R}(s, a) := R(s, a) + F_l(s, a, s'), \tag{6}$$

where $R(s, a)$ is the original extrinsic reward from the environment (Algorithm 1, line 8).

To shape reward signals, we use the soft state value function instead of the plain value function. This allows us to use reward shaping without a separate neural network for the reward function. Experimentally, we found this also to perform similar to using a plain value function approximated using a neural network (see Section 5.3.2). Haarnoja et al. (2018) also mention that it is in principle not necessary to add a separate approximator for the state value although they find it to stabilize results in practice. More precisely, we directly utilize the original form of the soft value function as given in the SAC algorithm for discrete action spaces (Christodoulou, 2019):

$$V(s) = \pi(s)^T \Big[(Q_{\hat{\theta}_i}(s) - \alpha \log(\pi(s))\Big)], \tag{7}$$

where Q denotes the target Q-functions. The soft value has two terms, the expected Q value at the given state and the entropy regularized probability of all possible actions. The Q function aims to update the policy to maximize the expected reward. The maximum entropy policy brings the agent into the states with less knowledge while still satisfying the side information (Ziebart et al., 2010).

Using Equation 7, the value function $V(s)$ is updated inspired by the batch RL idea (Sutton and Barto, 2018; Lange et al., 2012) and the N-steps Q iteration algorithm (Ernst et al., 2005). Instead of using the sample once to learn the TD, we can repeat the sample N times to estimate the TD value (see Algorithm 1, lines 4–6).

$$V(s) = (1-\alpha)V(s) + \alpha(r + \gamma' V(s')) \tag{8}$$

Now, we can rewrite the target Equation 1 by incorporating Equation 5:

$$\begin{aligned} &y(r, s', d) = \\ &[r + (\gamma V(s') - V(s))] + \gamma(1-d)V(s') \end{aligned} \tag{9}$$

This concludes the description of our reward shaping algorithm which relies on the soft value function and utilizes an N-step update.

5 Experimental results

5.1 Datasets

The experiments are run on the Jericho environment (Hausknecht et al., 2020)[1], which categorizes the games into three groups: possible games, difficult games, and extreme games. In the following

[1] https://github.com/microsoft/jericho

Game	Max	RAND	DRRN	NAIL	(Yao et al., 2021)	SAC	SAC+RS
		(Hausknecht et al., 2020)				Ours	
advent	350	36	36	36	-	36.00±0.00	36.00±0.00
balances	51	10	10	10	10	10.00±0.00	9.98±0.01
deephome	300	1	1	13.3	-	**28.95 ±0.25**	22.09 ±0.23
gold	0	0	4.1	3	-	5.98±1.16	**7.74 ± 0.79**
jewel	90	0	1.6	1.6	-	5.89 ±1.64	**7.70±1.99**
karn	170	0	**2.1**	1.2	-	0.01±0.01	0.01±0.01
ludicorp	150	13.2	13.8	8.4	14.8	14.89±0.40	**15.73 ±0.09**
yomomma	35	0	**0.4**	0	-	0.16 ±0.02	0.13 ±0.06
zenon	20	0	0	0	-	0.00±0.00	0.00±0.00
zork1	350	0	32.6	10.3	**43.1**	30.74 ±5.57	32.72 ±7.33
zork3	7	0.2	0.5	1.8	0.4	2.69±0.05	**2.72±0.04**

Table 1: The average score of the **last** 100 episodes is shown for three repetitions of each game with standard deviation. The maximum number of training steps is 50,000. RAND, DRRN, and NAIL results are by Hausknecht et al. (2020). This table only shows the best scores of the four variants in Yao et al. (2021)'s paper.

experiments, we focused on the difficult games, which have sparser rewards and require a higher level of long-term decision-making strategies than the possible games.

5.2 Experimental settings

We built a choice-based agent. The agent predicts one of the possible actions from the action space distribution based on the observation of the current time step and the previous action from the last time step. The agent receives the valid action space using the same handicaps as the DRRN method from the Jericho game environments identified by the world-change detection. As shown in Table 1, we ran the main experiments in two variants. In Figure 3 we compare two additional variants: a) SAC: This is the basic RL agent using the SAC algorithm. b) SAC+RS: Here we use the reward shaping technique in combination with SAC. This is our main algorithm as given in Algorithm 1. c) SAC+1S_RS: This variant is the same as SAC+RS except that $N = 1$ instead of $N = 32$. This means reward shaping is done without the N-step repetition of the TD update. d) SAC+NN_RS: In this variant we replace line 3 of Algorithm 1 with a neural network that estimates the plain value function. In appendix A, we show the details of the architectures and parameters for the neural networks and the RL agent.

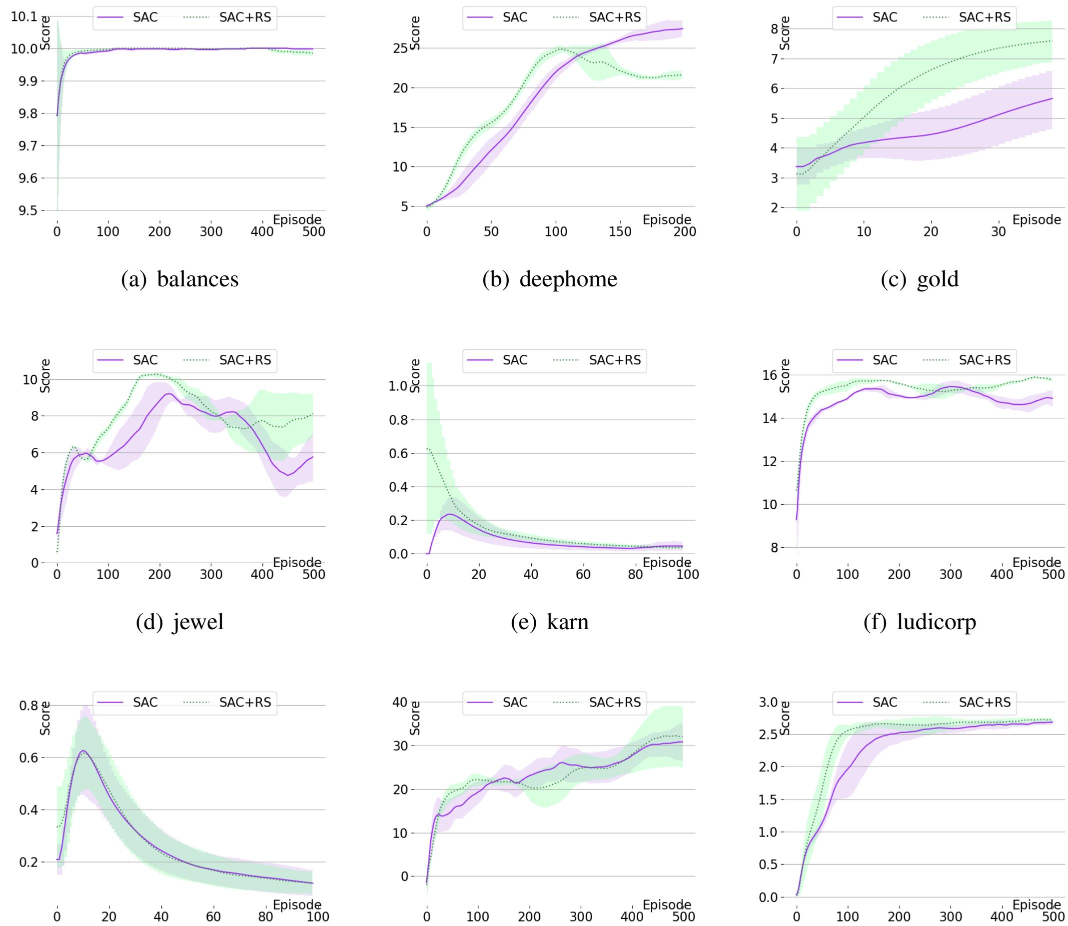
Input representation Following Hausknecht et al. (2020), the state s includes three elements: (observation, inventory, look) at the current time step. The representation of the elements in the state and the action are tokenized by a SentencePiece (Kudo and Richardson, 2018) model and then used seperately GRUs to learn the embeddings. The embedding size is 128. During training, the agent randomly samples the data from the replay buffer.

5.3 Results

We compare our results with the previous choice-based agents using deep Q-learning in Section 5.3.1. The effect of reward shaping and variants thereof is discussed in Section 5.3.2.

5.3.1 Comparison to Q-learning methods

Table 1 shows the game score of the SAC-based learning agent and SAC with reward shaping (SAC+RS). In comparison with DRRN and Yao et al. (2021), which are deep-Q learning-based RL agents, five of the SAC agent-based games can achieve notably higher scores. Three games got the same scores, and zork1 achieves similar results to DRRN (which is the closest baseline) but only uses half of the training steps. Only the scores of Yomomma and Karn are lower than those using the Deep-Q-learning agent. Same as for the baselines, we compute the average of the last 100 episodes for each run of the game. Each game is run three times and the mean and standard deviation are shown. For each run of one game, eight environments are run in parallel and the average score is computed. The results of the baselines are taken directly from the respective papers. The training progress is shown in Figure 2 where the game score is plotted over training episodes. We can see that the method converges well except for two games, karn and yomomma, where the agent



(a) balances (b) deephome (c) gold

(d) jewel (e) karn (f) ludicorp

(g) yomomma (h) zork1 (i) zork3

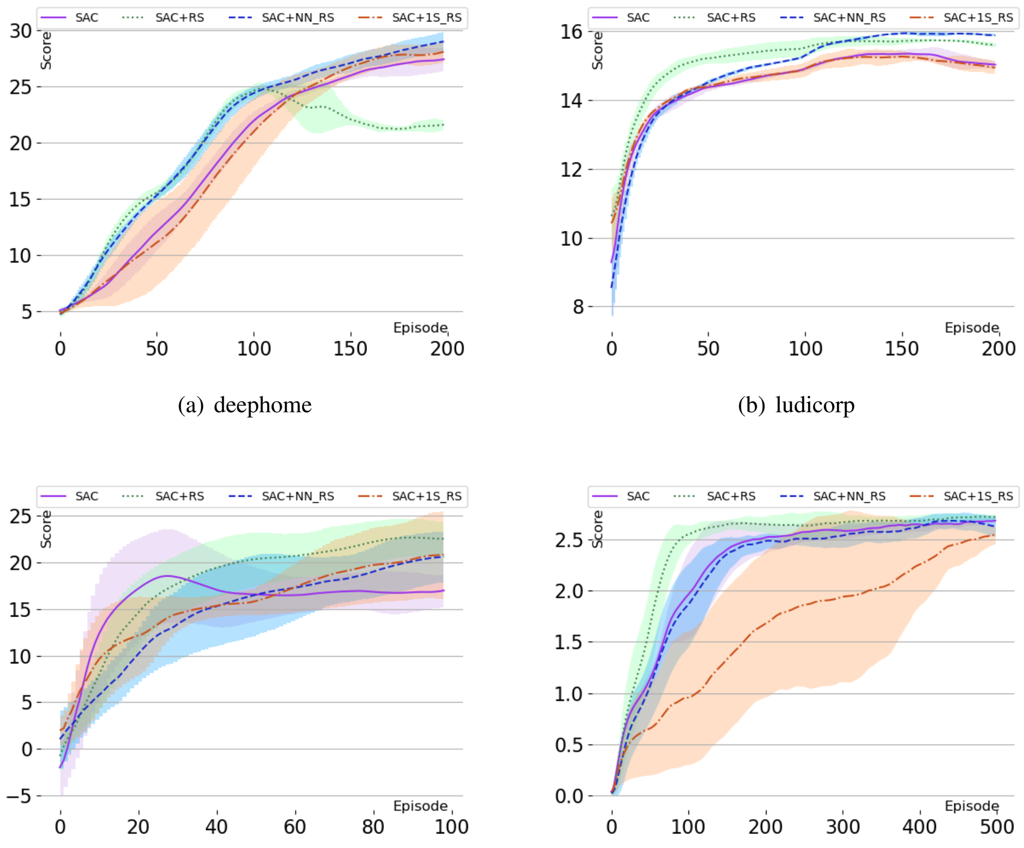
Figure 2: This figure shows the development of the game scores over training episodes where shaded areas correspond to standard deviations. Compared is the SAC agent with and without reward shaping. You can see that reward shaping leads to faster convergence at the beginning of training for b) deephome, d) jewel, f) ludicorp and i) zork3. The end score is higher with reward shaping for five of the nine games. Shown are only the games where the agents learn something (advent and zenon are excluded).

is not able to learn. Overall, the results indicate that SAC is a well-suited learning agent to solve text-based games.

5.3.2 Reward shaping

Overall, the final score of SAC with reward shaping is higher or the same for eight of the eleven games as shown in Table 1. Only for one game, deephome, does reward shaping reduces the score. We leave the investigation of this issue to future work. Another observation is that in many cases the standard deviation is lower when reward shaping is used than when reward shaping is not used.

Figure 2 shows the game score over training episodes. We can see that shaping the original rewards (SAC+RS) leads to faster convergence than without reward shaping (SAC). As mentioned in Section 4, the soft state value can achieve a similar performance as the state value while using fewer parameters. To experimentally prove this point, we run an additional variate of our method following Grześ and Kudenko (2010) to reshape the reward using the state value. The state values are approximated by a multi-layer neural network. The input of the neural network is the state. The target value is estimated by $G_t = r_t + \gamma V(S_{t+1})$, and the neural network updates by minimizing the MSE loss function of TD error at each time step: $L = MSE(G_t - V(S_t))$. We show the results in Figure 3. As we expected, the neural network-based value approximation (SAC+NN_RS) can get similar performance as directly using the soft state value from the critic policy. It is necessary to run



(a) deephome (b) ludicorp

(c) zork1 (d) zork3

Figure 3: This figure compares the SAC agent with and without reward shaping (RS), N-step repetition (1S_RS), and state-value-based RS (NN_RS). We can see that NN_RS can perform similarly as directly using soft-value as reward signals, and 1S_RS results in higher variances.The shaded areas correspond to standard deviations.

more experiments for the neural network-based function to find appropriate parameters. We sometimes even get better performance using the soft value function.

We also empirically investigate the effect of the N-step update described in Section 4 and Algorithm 1, lines 4–6. In Figure 3 we compare the update with $N = 32$ steps (SAC+RS) to the update with only one step (SAC+1S_RS). As the figure shows, the method converges to a similar final score, but exhibits much higher variance. In the case of zork3, the convergence is also slower. Therefore, we can conclude that the N-step update is beneficial for stabilizing training.

6 Conclusion and limitations

We propose a SAC-based RL agent to play text-based adventure games. The results show that the SAC-based agent can get significantly higher scores than deep-Q learning for some difficult games while using only half the number of training steps. Furthermore, we use a reward-shaping technique to deal with sparse rewards. This allows us to learn intermediate rewards, which speeds up learning at the beginning of training for some games and leads to higher scores than without reward shaping for many games. We compare this method to several state-of-the-art baselines based on deep Q-learning and show that we achieve higher scores with fewer training steps in many cases.

While we focused on the RL algorithm in this work, the limitations are, e.g., the knowledge representation and learning of the valid action space. In future work, we plan to adapt our method to play without the valid action handicap. We will apply the SAC agent and potential-based reward shaping technique to the action space generation task.

References

Prithviraj Ammanabrolu and Matthew Hausknecht. 2020. Graph constrained reinforcement learning for natural language action spaces. In *International Conference on Learning Representations*.

Prithviraj Ammanabrolu, Ethan Tien, Matthew Hausknecht, and Mark O Riedl. 2020. How to avoid being eaten by a grue: Structured exploration strategies for textual worlds. *arXiv preprint arXiv:2006.07409*.

Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. 2015. Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188.

Petros Christodoulou. 2019. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*.

Rati Devidze, Goran Radanovic, Parameswaran Kamalaruban, and Adish Singla. 2021. Explicable reward design for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34:20118–20131.

Damien Ernst, Mevludin Glavic, Pierre Geurts, and Louis Wehenkel. 2005. Approximate value iteration in the reinforcement learning context. application to electrical power system control. *International Journal of Emerging Electric Power Systems*, 3(1).

Marek Grześ and Daniel Kudenko. 2010. Online learning of shaping rewards in reinforcement learning. *Neural networks*, 23(4):541–550.

Xiaoxiao Guo, Mo Yu, Yupeng Gao, Chuang Gan, Murray Campbell, and Shiyu Chang. 2020. Interactive fiction game playing as multi-paragraph reading comprehension with reinforcement learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 7755–7765, Online. Association for Computational Linguistics.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

Anna Harutyunyan, Sam Devlin, Peter Vrancx, and Ann Nowé. 2015. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. 2020. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7903–7910.

Matthew Hausknecht, Ricky Loynd, Greg Yang, Adith Swaminathan, and Jason D Williams. 2019. Nail: A general interactive fiction agent. *arXiv preprint arXiv:1902.04259*.

Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. 2016. Deep reinforcement learning with a natural language action space. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1621–1630, Berlin, Germany. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Austin, Texas. Association for Computational Linguistics.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2018. Multi-hop knowledge graph reasoning with reward shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3243–3253.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. 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.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 278–287. Morgan Kaufmann.

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

Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. 2003. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning,(ICML-03)*, pages 792–799.

Yunqiu Xu, Meng Fang, Ling Chen, Yali Du, Joey Tianyi Zhou, and Chengqi Zhang. 2020. Deep reinforcement learning with stacked hierarchical attention for text-based games. *Advances in Neural Information Processing Systems*, 33:16495–16507.

Shunyu Yao, Karthik Narasimhan, and Matthew Hausknecht. 2021. Reading and acting while blindfolded: The need for semantics in text game agents. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3097–3102, Online. Association for Computational Linguistics.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. 2020. Keep CALM and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 8736–8754, Online. Association for Computational Linguistics.

Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey. 2010. Modeling interaction via the principle of maximum causal entropy. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010), June 21-24, 2010, Haifa, Israel*, pages 1255–1262. Omnipress.

A Appendix

Experimental settings:

Neural networks and parameters The policy neural network includes three linear layers with two hidden dimensions $D_1 = 512$ and $D_2 = 128$, each hidden layer connects with the ReLU activation function, and the categorical distribution is on top to ensure that the sum of action probabilities is one. The Q-function neural network has also three linear layers with ReLU activation functions. Both policy and Q-function update at each step, and the target Q functions update the weights from the Q-function every two steps.

The RL agent parameters were set as follows: the batch size is 32, and the learning rate of both policy and Q-function neural networks is 0.0003. Epsilon-Greedy action selection and a fixed entropy regularization coefficient were used in all of the experiments. For each game, we ran 8 environments in parallel to get the average score of the last 100 episodes, and each model ran three times to compute the average scores. The maximum number of training steps per episode is 100.

Since the RL agent interacts with the game environments, the training time depends on the game implementation in the Jericho framework. For example, zork1, and zork3 are comparably fast to train, whereas Gold takes an extremely long time compared to the rest of the games. Because of this, we only trained gold for 4,000 steps, yomomma for 10,000 steps, and karn for 10,000 steps. Our comparison methods also use varying step sizes for these games (but they use more training steps than we do). Most of the previous work trained the agent in a maximum of 100,000 steps, whereas the maximum number of training steps for our method is only 50,000 in all experiments.

Computing infrastructure We ran the experiments on Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz and the Nvidia GPUs (can be one of GeForce RTX 2080 or Tesla V100).

B supplementary material

Our experiments are based on the publicly accessible Jericho environment (Hausknecht et al., 2020) that provides the environment for playing all games in our experiments. Our code is attached as a supplement.