Crawling in Rogue's dungeons with deep reinforcement techniques

Andrea Asperti, Daniele Cortesi, Carlo De Pieri, Gianmaria Pedrini, Francesco Sovrano

***Abstract*—This article is a report of our extensive experimentation, during the last two years, of deep reinforcement techniques for training an agent to move in the dungeons of the famous Rogue video-game (neglecting, for the moment, other aspects of the game). The challenging nature of the problem is tightly related to what was the main innovative feature of the game, namely the procedural, random generation of new dungeon maps at each level: this peculiarity forbids any form of level-specific learning, and forces to address the navigation problem in its full generality. Other interesting aspects of the game from the point of view of automatic learning are the *partially observable* nature of the problem, since maps are initially not visible and get discovered during exploration, and the problem of *sparse rewards*, requiring the acquisition of complex, non-reactive behaviors involving memory and planning. In this article, we develop on previous works to make a more systematic comparison of different learning techniques, focusing in particular on Asynchronous Advantage Actor-Critic (A3C) and Actor-Critic with Experience Replay (ACER). In a game like Rogue, sparsity of rewards is somehow mitigated by the variability of the dungeon configurations (sometimes, by luck, exit is at hand); if this variability can be tamed - as ACER, better than other algorithms, seems able to do - the problem of sparse rewards can be overcome without any need of intrinsic motivations.**

***Index Terms*—Deep Reinforcement Learning, Rogue, Dungeon, Maze, Labyrinth, Asynchronous Advantage Actor-Critic, Experience Replay, A3C, ACER, Partially Observable Markov Decision Process, Sparsity of Rewards, Experience Replay, Attention.**

I. INTRODUCTION

GAME-like environments, providing realistic abstractions of real-life situations, have been at the core of many recent breakthroughs in AI such as [1], [2], [3], [4], [5]. In particular, dungeon based games, where the player is meant to explore new scenarios, collecting treasures, items and weapons helping him to survive, proved to be an especially stimulating and challenging subject [6]. These games are typically characterized by: (1) complex forms of exploration inside mazes and labyrinths [7]; (2) very *sparse* rewards; (3) the need to pay *attention* [8] to specific portions of the environment; (4) the need to exploit *memory* (as in the case of the famous T-maze); (5) the acquisition of complex, non-reactive behaviors involving some form of *planning*.

In the very last years there has been a real explosion of different techniques addressing the previous problems. In particular, sparsity of rewards is usually reputed to be one of the "hardest challenges in contemporary reinforcement learning“ [9], and many different approaches have been investigated to overcome this problem, based on different kinds of *intrinsic motivations* [10] such as empowerment [11], counting [12] or unsupervised auxiliary task [13], the deployment of hierarchical learning systems ([14], [15]) or a combination of them as in [9].

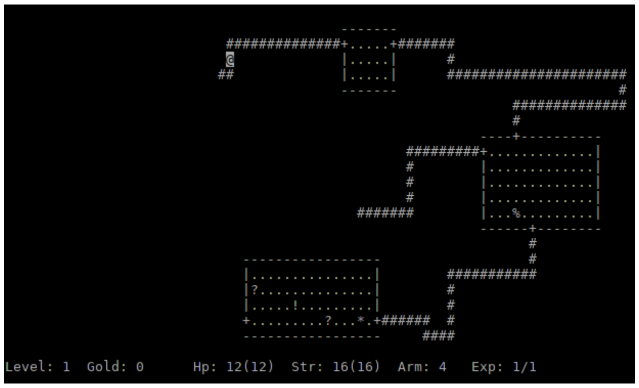
In many cases, results are tested on a limited number of games; when they are experimented over a large class of games, like in the case of the Atari-family, most techniques usually work well on a subset of them with relatively homogeneous features. All approaches are usually quite sensible to configuration hyper-parameters: minor variants may just prevent the network to learn. This substantial instability and the lack of meta-knowledge on the domain are a clear indication that we need a deeper and more extensive experimentation. Enlarging the set of games that can be used as a learning environment for reinforcement techniques, and improving our experience on the applicability and generality of these techniques is in fact a major research objective in deep reinforcement learning.

This article is focused on Rogue, the ancestor of the dungeon-crawling genre, and one of the most famous and widespread game in the history of video-games. While the game offers a lot of interesting challenges for reinforcement learning (see next section), we mostly focused for the moment on the navigation issue, essentially exploiting Rogue as a random generator of labyrinths of reasonable complexity.

In this article, we both introduce the new version of our "Rogueinabox" learning environment [16], explicitly designed to offer a simple interaction with Rogue (in the spirit of [17] for Atari games), and present the results of two years of experimentation on the game using different techniques, comprising DQN (enriched with several forms of intrinsic motivations and counting), Advantage Actor-Critic (A3C) (exploiting a decomposition in different game *situations*), and Actor-Critic with Experience Replay (ACER).

In a game like Rogue, sparsity of rewards is somehow mitigated by the randomness of the dungeon configurations: by chance, the way out of the labyrinth may sometimes be close to the rogue, offering a handy reward. Provided the learning algorithm is able to cope with the wild variability of dungeon maps, generalizing the behaviour to arbitrary scenarios, the problem of sparse rewards can in this case be overcome without any need of intrinsic motivations. In particular, a suitably configured version of ACER, seems to outperform different techniques, allowing to find and take the stairs of a dungeon level in about 98% of cases, by just rewarding the successful completion of the episode and anything else.

The authors are at the University of Bologna, Department of Informatics: Science and Engineering (DISI), Mura Anteo Zamboni 7, 40127, Bologna, Italy



```
Level: 1  Gold: 0      Hp: 12(12)  Str: 16(16)  Arm: 4   Exp: 1/1
```

Fig. 1: The bidimensional ASCII graphical interface of Rogue

A. Rogue

Rogue is a complex dungeon-crawling video-game of the eighties, the first of its genre. The player (the "rogue") is supposed to roam inside a dungeon structured over many different levels, with the final aim to retrieve the amulet of Yendor, coming back to the surface with it. If the player dies, the game restarts from scratch (permanent death); moreover, all levels are randomly generated and different from each other. This continuous generation of new and different levels is probably the most challenging aspect from the point of view of machine learning, since it prevents to learn ad hoc behaviors, forcing to discover action policies that generalize to arbitrary scenarios (in Atari games, like e.g. Montezuma's Revenge, each level is always identical to itself and may be replayed at will).

The full game offers many additional challenges, such as collecting objects, powering up the rogue, and fighting against monsters of increasing power; however we did not address these aspects for the moment, just focusing our attention on exploration. Actually, ignoring these aspects of the game makes the problem of *sparse rewards* even more severe; remarkably, we are able to train the agent to explore the dungeon *just rewarding the successful completion of the episode*!

In its standard configuration, the dungeon consists of 26 floors and each floor consists of up to 9 rooms of varying size and location, randomly connected through non-linear corridors, and small mazes.

As evident from Figure 1, an important feature that differentiates Rogue from other, more modern, 3D dungeons-based games such as VizDoom [18] or Labyrinth [3] is the graphical interface, that in the case of Rogue is ASCII-based (see Figure 1). All game items, such as walls, doors, monsters, objects, and so on, are not represented by icons or images, but by means of ASCII characters. This allows to bypass some detection problems typical of machine vision, that are currently pretty well understood, focusing the attention on more interesting and complex activities: the awareness of the surrounding environment, and the deployment of suitable, sensible behaviors.

The map gets disclosed during exploration, and it is only *partially observable* at the beginning of each round. This is another interesting aspect of Rogue, differentiating it from other games, like those of the Atari family: to reach a new floor the rogue must find and go down the stairs; the position of the stairs is different at each floor, likely located in a yet unexplored room, and hence hidden from sight at the beginning of exploration. In mathematical terms, the problem is described as a Partially Observable Markov Decision Process (POMDP).

Finding and taking the stairs are the main operations governing the agent movement: the difference between the first floors and the subsequent ones concerns the frequency of meeting monsters, dark rooms, mazes or hidden doors.

As a consequence, we organized the training process on the base of a single level, terminating the episode as soon as the rogue takes the stairs. In the rest of the work, unless stated otherwise, when we talk about the *performance* of an agent, we refer to the probability that it correctly completes a *single* level (measured as an average of 200 consecutive games), finding and taking the stairs within a maximum of 500 moves[1]. At the end of training, several agents reach performances around 97-98%.

B. Rogueinabox

In order to facilitate the interaction with Rogue we developed a suitable library, named Rogueinabox. The library was introduced in [16], and sensibly revisited and extended in [19], [20] along with its experimentation with different neural architectures. With the article, we release a new version [21] of Rogueinabox with several improvements on previous versions.

The tool allows users to easily specify custom rewards functions and game state representations. It also handles logging and the UI. The library is currently configured as an Application Programming Interface neatly decoupling the interface with the game from the developments of agents. Having removed from the library the concept of agent grants a large degree of freedom in the choice of machine learning algorithms, covering both value-function-based and actor-critic methods.

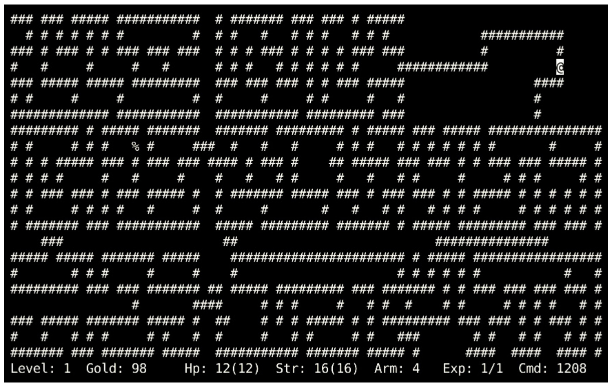
A considerable effort has been devoted to permit a simple configuration of many characteristics of the game, comprising the possibility to

1) enable/disable monsters;
2) enable/disable traps;
3) configure the life period before starvation;
4) configure the frequency of mazes;
5) set the depth of the amulet of Yendor;
6) set the random level generator seed
7) enable/disable secret passageways
8) enable/disable dark rooms

The game code itself had to be modified to allow this level of customization; the source is available at [22]. For instance, in Figure 2 we give an example of a rogue floor where all rooms have been turned into mazes.

Another improvement concerns the evaluation module, favoring a better comparison between training with different architectures. It currently allows keeping track of changes

[1]For a good agent, in average, little more than one hundred moves are typically enough.



```
Level: 1  Gold: 98    Hp: 12(12)  Str: 16(16)  Arm: 4   Exp: 1/1  Cmd: 1208
```

Fig. 2: A Rogue floor configured through Rogueinabox to just contain mazes.

in specific parameters during training, logging them in a dedicated file. It can also be easily extended not only to keep track of additional values but also to compute statistics at run time. This turned out to be very convenient to analyze details of different agents behavior.

C. *Structure of the article*

The article is structured in the following way. In Section II we discuss related bibliography and clarify the relation between this work and previous publications of the authors on the same subject. Section III contains a quick introduction to Reinforcement Learning (mostly meant to fix notation) and some of its basic techniques of particular interest for this article: Q-learning, Actor-Critic, and ACER. In Section IV we give experimental results relative to the problem of completing the exploration of the first level, discovering and taking the stairs. In the following section we briefly discuss a couple of more complex scenarios, mostly with the aim to provide a baseline for future works. The first experiment consists in trying to descend up to the 10th level (Section V-A); in the second one, we try to retrieve the amulet of Yendor (Section V-B) suitably located in the first or in the second level of the dungeon. Section VI is reserved to conclusions and future works.

II. Related work

Since the seminal work by Mnih et al. [1] exploiting a combination of Q-learning and neural networks (Deep Q-Networks, DQN) in application to Atari games [17], the field of application of deep reinforcement techniques for training agents to play video games has rapidly evolved, both in terms of improvements, such as Double Q-learning [2], or innovative architectures. DQN was the first technique we employed to address exploration in Rogue's dungeons [19]; even enriching the model with auxiliary rewards and counting mechanisms [23], results remained unsatisfactory. As observed in [24] for a similar grid-world problem, DQN techniques seem to suffer a lack of generalization when trained over highly heterogeneous maps, failing to understand the goal-directed nature of the behavior (and the maps in [24] were sensibly simpler, in size and complexity, than those of Rogue).

The class of RL algorithms that could be successfully combined with deep models was extended to on-policy techniques by the introduction of actor-critic methods, either asynchronous (A3C) [3] or synchronous (A2C) [25].

Our previous work [20] was precisely focused on the application of A3C to Rogue. A naive approach did not give good results, so we started testing several variants to better understand the source of the problems. Influenced by the recent literature [9] on similar topics, we conjectured that the main difficulty was the *sparsity of rewards*, which we attempted to reduce both acting on the rewarding mechanism (offering mid-term rewards for e.g. discovering new rooms), and splitting the problem in a small number of *situations* (room with no stairs, corridors, room with stairs), each one handled by a specific sub-net. We also cropped the view around the position of the rogue, in order to reduce the dimensionality of the problem, and to better adhere with similar works dealing with dungeons and labyrinths [18], [13].

The resulting model, described in [20], is an example of non-generic Hierarchical Reinforcement Learning (HRL) model based on a simple variation of the options framework [26], [27], [28]. It is composed by an ad hoc high-level manager able to partition the state space into n different subspaces (called situations) for respectively n different A3C low-level workers, forming in such way a hierarchy of policies.

This model is able to achieve remarkable performances [20], allowing to train an agent able to discover and take the stairs in about 97% of cases; a video showing the agents' behavior is available on YouTube[2]. However, its architectural design, relying too much on the meta knowledge of the specific problem, was not satisfactory.

A different approach to compensate (among other things) sparsity of rewards consists in exploiting *experience replay* to improve *sample efficiency*, as described in [5]. This is the key approach deployed in this article, allowing us to obtain the same performances of [20] but with a sensibly more generic architecture.

III. Reinforcement Learning Background

This section contains a short introduction to Reinforcement Learning Techniques, mostly with the aim to fix notation. The content is quite standard, and we largely borrowed it from our previous work [20].

A Reinforcement Learning problem is typically formalized as a Markov Decision Process (MDP). In this setting, an agent interacts at discrete time steps with an external environment. At each time step t, the agent observes a state s_t and chooses an action a_t according to some policy π, that is a mapping (a probability distribution) from states to actions. As a result of its action, the agent obtains a reward r_t (see Fig. 3), and the environment passes to a new state $s' = s_{t+1}$. The process is then iterated until a terminal state is reached.

The future cumulative reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total accumulated reward from time starting at t. $\gamma \in [0, 1]$ is the so called *discount factor*: it represents the difference in importance between present and future rewards.

[2]https://www.youtube.com/watch?v=1j6_165Q46w

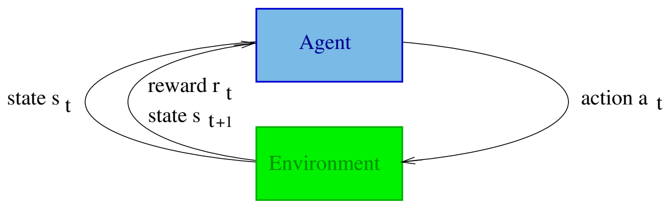


Fig. 3: Basic operations of a Markov Decision Process

The goal of the agent is to maximize the expected cumulative return starting from an initial state $s = s_t$.

The *action value* $Q^\pi(s,a) = \mathbb{E}^\pi[R_t|s = s_t, a = a_t]$ is the expected return for selecting action a in state s_t and prosecuting with strategy π.

Given a state s and an action a, the optimal action value function $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ is the best possible action value achievable by any policy.

Similarly, the *value* of state s given a policy π is $V^\pi(s) = \mathbb{E}^\pi[R_t|s = s_t]$ and the optimal value function is $V^*(s) = \max_\pi V^\pi(s)$.

A. Q-learning and DQN

The Q-function and the V-function can be represented by suitable function approximators, e.g. neural networks. We shall use the notation $Q(s,a;\theta)$ to denote an approximate action-value function with parameters θ.

In (one-step) Q-learning, we try to approximate the optimal action value function: $Q(s,a) \approx Q(s,a;\theta)$ by learning the parameters via back propagation according to a sequence of loss function functions defined as follows:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)} \left[(r + \gamma\, max_{a'}\, Q(s',a',\theta_{i-1}) - Q(s,a,\theta_i))^2\right]$$

where s' is the new state reached from s taking action a and $U(D)$ is the uniform distribution on stored transitions for experience replay.

The previous loss function is motivated by the well-known Bellman equation, that must be satisfied by the optimal Q^* function:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma max_{a'} Q^*(s',a')]$$

Indeed, if we know the optimal state-action values $Q^*(s',a')$ for next states, the optimal strategy is to take the action that maximizes $r + \gamma max_{a'} Q^*(s',a')$.

Q-learning is an *off-policy* reinforcement learning algorithm: we can learn from actions taken according to a different policy. The main drawback of this method (in the one-step case) is that a reward only directly affects the value of the state action pair s, a that led to the reward. The values of other state action pairs are affected only indirectly through the updated value $Q(s,a)$; back propagation to relevant previous states and actions may require several updates, slowing down the learning process.

B. Actor-Critic, A3C and A2C

In contrast to value-based methods, policy-based methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameters θ by gradient ascent on $\mathbb{E}[R_t]$.

The standard REINFORCE [7] algorithm updates the policy parameters θ in the direction $\nabla_\theta\, \mathbb{E}[log\pi(a_t|s_t;\theta)R_t]$, which is an unbiased estimate of $\nabla_\theta\, \mathbb{E}[R_t]$.

It is possible to reduce the variance of this estimate while keeping it unbiased by subtracting a learned function of the state $b_t(s_t)$ known as a baseline. The gradient is then $\nabla_\theta\, \mathbb{E}[log\pi(a_t|s_t;\theta)(R_t - b_t)]$.

A learned estimate of the value function is commonly used as the baseline $b_t(s_t) \approx V^\pi(s_t)$. In this case, the quantity $R_t - b_t$ can be seen as an estimate of the *advantage* of action a_t in state s_t for policy π, defined as $A^\pi(a_t|s_t) = Q^\pi(s_t,a_t) - V^\pi(s_t)$, just because R_t is an estimate of $Q^\pi(s_t,a_t)$ and b_t is an estimate of $V^\pi(s_t)$.

This approach can be viewed as an actor-critic architecture where the policy π is the actor and the baseline b_t is the critic.

A3C [3] is a particular implementation of this technique based on the asynchronous interaction of several parallel couples of Actor and Critic. It was originally believed that asynchronicity, leading to more independence, was an important aspect of A3C, stabilizing learning without the need for experience replay as in DQN; however, A2C [25] showed that this was merely due to having larger amounts of data, as in A2C's case the policies gathering data are identical.

C. ACER

The Actor-Critic with Experience Replay (ACER) algorithm [5] combines the A3C framework with experience replay, making it *off-policy*, and a more efficient version of Trust Region Policy Optimization (TRPO) [29], improving on the sample efficiency of A3C.

In ACER a neural network parameterize π and Q (respectively by θ and θ_v), instead of V as in A3C. The baseline V can simply be computed, given π and Q, as $V(s) = \sum_a \pi(a|s)Q(s,a)$. Without TRPO, the ACER gradient is defined as follows

$$g_t = \bar{\rho}_t \nabla_\theta log\ \pi(a_t|s_t;\theta)[Q^{ret}(s_t,a_t) - V(s_t;\theta_v)] + \mathop{\mathbb{E}}_{a\sim\pi} \left(\left[\frac{\rho_t(a) - c}{\rho_t(a)}\right]_+ \nabla_\theta log\ \pi(a|s_t;\theta)[Q(s_t,a;\theta_v) - V(s_t;\theta_v)] \right)$$

where Q^{ret} is the *Retrace* target [30], $\rho_t = \rho_t(a_t) = \frac{\pi(a_t|s_t;\theta)}{\mu(a|s_t)}$ is referred to as *marginal importance weight* (μ is the policy used to take the action which, during experience replay, may differ from π), $\bar{\rho}_t = min\{c, \rho_t\}$ is its truncation and $[x]_+ = max\{0, x\}$. These last two components are the main elements of a technique the authors call *truncation with bias correction trick*, the first meant to ensure the variance of the update is bounded and the second to keep the estimate unbiased.

With TRPO, the update is corrected so that the resulting policy does not deviate too far, in terms of KL divergence, from an *average policy network* representing a mean of past policies. Its parameters are denoted by θ_a and updated *softly* at the same time of the policy by the rule $\theta_a \leftarrow \alpha\theta_a + (1-\alpha)\theta$.

IV. Exploring the first level

This section is focused on exploration of the first level of the dungeon; in the next section we shall briefly discuss more complex scenarios.

A. From Partitioned A3C to ACER

A3C alone does not seem to suffice to properly address the exploration of Rogue's dungeons. In [20], we circumvented the problem splitting states in a small number of predefined situations, each one delegated to a specific A3C sub-net, whose structure is described in Figure 4.

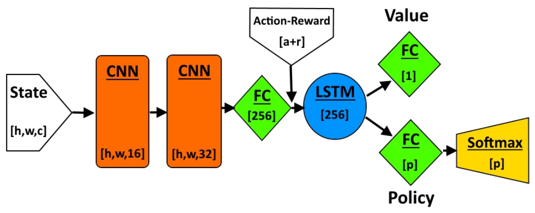


Fig. 4: The A3C architecture used in [20]

We tested several different situations (see [20]); the simplest configuration just consists of two basic situations: the rogue is either in the room with the stairs or not. This essentially decomposes the exploration task in two simpler objectives: (1) find the room with the stairs; (2) take them.

We also properly tuned the rewarding mechanism to mitigate sparsity of rewards, offering mid-term recompenses for discovering new rooms or traversing new doors.

Finally, we cropped the view around the position of the rogue, obtaining a sort of first person view of the game. On one side, this makes the game intrinsically partially observable, but on the other it removes the necessity to understand the relevance of the rogue, and to focus the attention on it.

This architecture obtains excellent performances: at the end of training, the agent is able to discover and take the stairs in about 97% of cases.

However, many architectural choices rely on meta-knowledge specific to the given problem, both in the individuation of relevant game situations, as well as in the rewarding mechanism, that seems to be too informative, ad hoc, and it is likely the source of strange biases in the learned policy.

We tried to work out these issues while achieving a comparable success rate. The main ingredients of the approach described in the next sections are the following:

1) we added experience replay to A3C, passing to a suitably configured version of ACER; notably, we work with unusually long temporal sequences of states (60);
2) we adopt a *particularly sparse* system of rewards, that only recompenses descending the stairs, i.e. the successful completion of the episode; according to our experiments, most techniques are in trouble with such rewards;
3) a state representation encompassing the *entire map*, and not centered on the rogue, minimizing pre-processing of input states;
4) a *three towers* neural network architecture, processing the input at multiple scales, and abstracting from position via global maxpooling: a light and effective architecture.

Experimentally we discovered that working with a momentum based optimizer, and especially one exploiting Nesterov's momentum (RMSProp)[3] was particularly effective, confirming the good properties of this optimizer in conjunction with recurrent neural networks. We based our ACER implementation on the OpenAI's baselines [31], that we altered only as far as to enable the interaction with Rogue via the Rogueinabox library, and suitably configured to our needs (the hyper-parameter setting is discussed in Section IV-E). Our implementation is open access, and can be found at [32].

B. State representation

The state representation we use encompasses the entire map. It is implemented as a matrix with shape 22×80 i.e. the dimensions of the screen in ASCII characters minus the top and bottom rows (that provide dialog messages).

The most successful representation we devised is composed of five layers (thereby setting the dimension of a single state at $22 \times 80 \times 5$), each displaying, respectively, the positions of:

1) walls; 2) doors and corridors;
3) floor tiles; 4) stairs;
5) the rogue

We set these coordinates with value 1 and all other cells to zero. This is similar to what was used in [16], [19], albeit with a slightly lower level of abstraction.

In [16] we also used some additional layers providing handcrafted forms of memory of past rogue positions. This was superseded by the use of a Long-Short Term Memory (LSTM) layer in [20], similarly to what we do in this article.

C. Reward function

We use the simplest possible reward function: a positive reward (+10) for descending the stairs and zero for everything else. This choice was mainly dictated by two considerations:

1) compare the performance of different reinforcement techniques in a regime of particularly sparse rewards;
2) avoid to introduce any bias in the agent behavior.

The reward is much sparser than what was employed in [20] and is the main reason that induced us to switch to the ACER algorithm: ACER was explicitly designed to be more sample efficient and our study provides an additional empirical confirmation of this claim.

D. Neural Network

To avoid relying on a sub map cropped around the rogue we recovered our original architecture described in [19] and summarized in Fig. 5. It exploits three Towers implementing a *pyramidal approach*, where each tower is processing the input at a different level of detail; their results are then concatenated together and subsequently elaborated via LSTM and dense layers.

All towers share a similar mechanism: they start performing convolutions with small (3×3) kernels on the input and then

[3] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

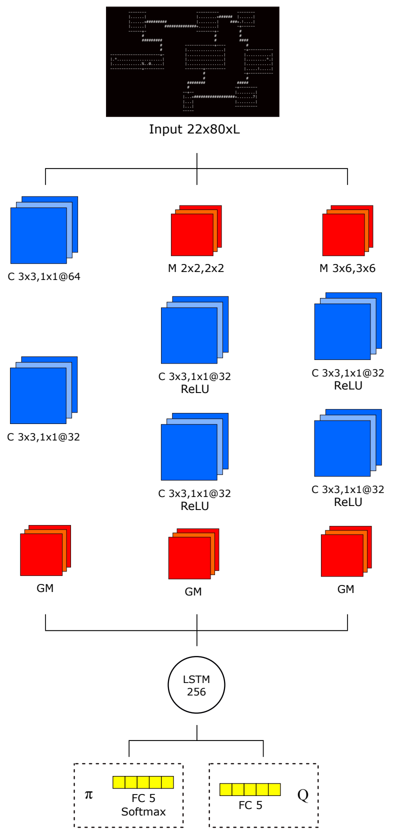


Fig. 5: Three towers neural network architecture

apply a *global maxpooling* to focus on the presence/absence of the given feature. Let us remark that the global maxpooling essentially throws away all spatial information (while drastically reducing the size of the network).

We proved experimentally that, especially on the first tower, the agent learns very rapidly to synthesize features comprising the rogue *in conjunction* with other entities of interest (walls, doors, stairs, etc), hence implicitly *focusing its attention* on the rogue, with no need to understand his position on the map.

The distinction between the three towers is essentially in the different level of detail at which the input is processed. To this aim, we simply apply a progressive, initial maxpooling to the input, with stride 2×2 for the second tower and stride 3×6 for the third one. Other, more sophisticated techniques can of course be envisaged, e.g. progressively augmenting the size and stride of convolutions. The observation of the map at different scales is required to compensate the destructive spatial behavior of global maxpooling.

With the intent of better explaining the network's behavior we produced a video [33] where we show the output of the three towers and how it relates with the input state, during a typical play. For each tower, the receptive fields of activated neurons are summed up to create a sort of *heatmap* of the level. A snapshot of the video is shown in Figure 6. It is important to stress that, in order to create the video, we exploit an information *not available* to the agent, namely the location of the receptive field in the map (the global maxpooling layer precisely throws away this information). The agent does not

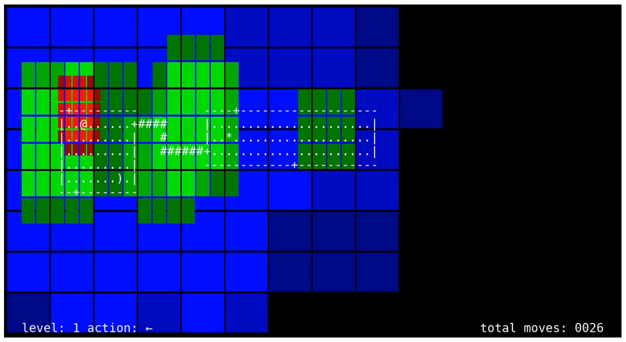


Fig. 6: Receptive fields for high-activation neurons of the three towers (respectively, in red, green and blue) before maxpooling. The maxpooling layer looses the spatial location of the receptive field in the map, whose global structure can only be reconstructed through memory. The fact that receptive fields follow the rogue means that the network correctly understands the relevance of the rogue, synthesizing filters responding to the joint presence of the rogue with other relevant game items (walls, doors, stairs, corridors, etc.)

know where the rogue and the other elements of the game are located: it only knows their mutual position, as captured by the learned convolutional filters. A global understanding of the map may only derive from past memories, that is one of the reasons why long temporal sequences for the lstm-model are required.

The current network architecture is relatively small: it has a dimension of 4 hundred thousand parameters to be compared with the around 3 million parameters of the network in [20].

E. Hyper-Parameters

In table I we summarize the hyper-parameter values we use.

Parameter	Value
parallel actors	16
episode max length	500
entropy β	0.01
discount factor γ	0.99
batch size (t_{max})	**60**
ACER memory buffer size	50000
ACER replay ratio r	4
ACER importance weight clipping c	10
ACER average policy α	0.99
ACER KL divergence constraint δ	1
initial learning rate η	0.0007
rms decay	0.99
rms momentum	0
rms epsilon	$1e^{-5}$
rms clip norm	10

TABLE I: ACER Hyper-parameters

As already remarked, we work with an unusually large input sequence for the LSTM network: in our case $t_{max} = 60$. We recall that in A3C is $t_{max} = 5$ [3], while the default for ACER [5], [31] is $t_{max} = 20$. In the case of Rogue, $t_{max} = 20$ is not working properly: the success rate is below 0.94, in comparison with the score close to .98 that can be achieved with $T_{max} = 60$.

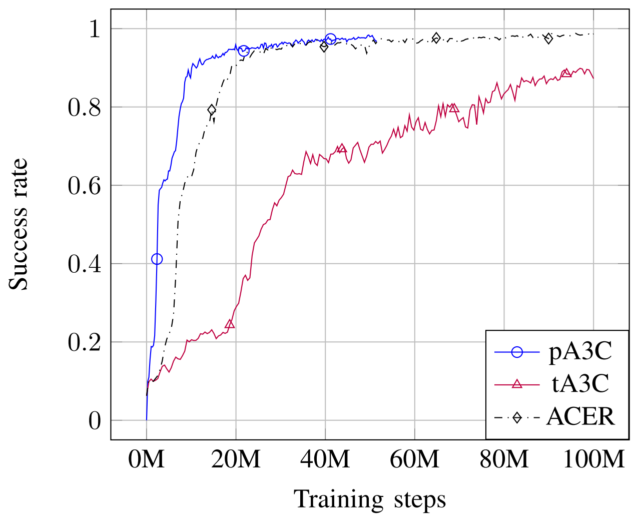


Fig. 7: Results of partitioned A3C (pA3C) and ACER; We also show the performance of A3C (tA3C) with the same ACER configuration (i.e. model, state representation, reward)

Agent	pA3C	tA3C	ACER
Success rate	98%	87.25%	98.69%
Avg number of seen tiles	386	354	385
Avg number of steps to succeed	111	113	119

TABLE II: Final results of partitioned A3C (pA3C), ACER and A3C with the same ACER configuration (tA3C)

F. Evaluation criteria

When developing several methods to solve a task it is important to establish a well-defined set of metrics under which each different effort becomes comparable. We use the same criteria established in [20] and base our evaluation on the following statistics:

1) The average number of episodes in which the agent is able to descend the stairs; when this happens, we declare the episode won and reset the game;
2) The average number of steps taken when climbing down;
3) The average return;
4) The average number of seen tiles;

Our emphasis is on the first two points, but we also keep an eye on the others. In all of our experiments, we average these values over the most recent 200 games played at a given training step. Since we always employ several parallel actors, in the statistics we display in the various plots the values are the average *over the averages* of each actor. In these and all other evaluation metrics, when victory conditions are not met within the maximum amount of actions established, then the episode is considered lost and the game reset.

G. Results

We show the results in figure 7 and summarize the final scores in table II. We were able to match what we achieved with partitioned A3C and cropped view, but solving all architectural aspects we were unhappy with.

V. More complex scenarios

In this section we discuss some simple generalizations of the exploration problem relative to the first level. The purpose of this section is to explain that there are still a lot of interesting issues to be solved, and to set some baselines for future research, possibly by other teams.

A. Reaching the 10th level

The good results we achieved pushed us to test how far our agent could crawl in the dungeons. We thus set up a new experiment in which we allowed the agent to descend as far as the 10th level.

Here an important simplification comes into play: from the second level onward some passages may be hidden, requiring the agent to explicitly search for them, repeatedly typing for a stochastic number of times the “s” key in proximity of the *presumed* (sic!) position of the hidden door. The sections beyond these hidden passages might easily enclose the stairs, requiring the player to learn to habitually search (with the dedicated action) what appears to be a dead-end in a corridor or a room without visible doors. We deemed this to be too hard to learn for our model and thereby disabled secret passages.

Other interesting aspects of the game typical of deep levels are *dark rooms* and *labyrinths* (figure 8). The former are rooms where only the cells *immediately surrounding* the rogue are displayed on the screen. The walls of such rooms are not shown when it is first entered, however they remain visible once discovered for the first time. The same holds for stairs and items, but not for floor tiles or monsters. The labyrinths are exactly what the name suggests: dense webs of corridors with numerous branches and dead-ends. The frequency of both labyrinths and dark rooms increases with the depth of the level.

We essentially employed the same configuration (state representation, reward mechanism, neural network and hyper-parameters) and evaluation criteria used in Section IV. We just slightly adapted the state representation in order to take into account the fact that tiles in dark rooms disappears from view when the rogue moves away, while, once uncovered, their presence was persistently stored in the state. For an evaluation of the impact of this modification on the agent performance see [34].

Results are shown in Table III, where we compare our ACER agent with a random agent. We show statistics for increasing values of maximum steps per episode: 1200, 2000, 3000 and 10000 steps. The latter bound is unreasonably high: it only serves as a sort of asymptotic analysis of the behaviors. Results are shown in Table III.

The scores are slightly worse of what be could expected: given the 98% success rate on the first level, one could hope to reach the 10th level, thereby descending 9 levels, in about $0.98^9 \sim 83\%$ of cases.

The difficulty is clearly posed by dark rooms and labyrinths. It is not difficult to prove that the average number of steps spent in levels presenting those challenges is sensibly higher than usual (see [34] for details).

Let we also observe that, after 100 Million iterations, training was still in an ascending trend. It is possible that

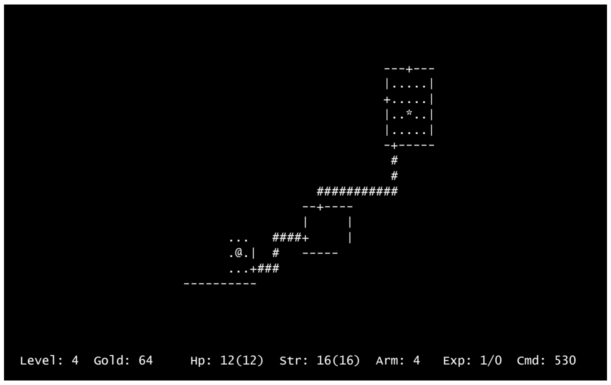
Success Rate										
Max Steps	Agent	levels								
		1st	2nd	3rd	4th	5th	6th	7th	8th	9th
1200	ACER	97.72	93.41	86.19	76.56	63.62	47.03	29.91	17.34	8.78
	Rand.	15.50	3.00	–	–	–	–	–	–	–
2000	ACER	98.00	94.00	88.00	78.00	67.50	58.50	46.50	36.00	26.50
	Rand.	19.50	1.40	–	–	–	–	–	–	–
3000	ACER	99.00	96.50	91.00	86.00	81.50	75.00	64.50	56.00	49.00
	Rand.	20.50	1.50	0.50	–	–	–	–	–	–
10000	ACER	99.00	97.00	93.50	88.00	82.00	75.50	67.00	59.00	50.00
	Rand.	37.50	13.00	3.50	1.50	1.00	–	–	–	–

(A)

Average levels descended		
Max steps	Agent	Value
1200	ACER	5.2
	Random	.19
2000	ACER	5.9
	Random	.22
3000	ACER	7.0
	Random	.25
10000	ACER	7.1
	random	.57

(B)

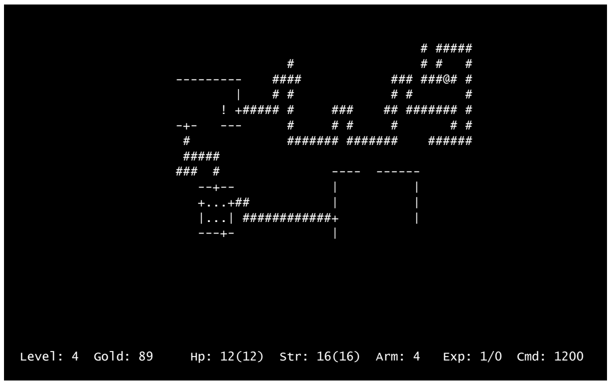
TABLE III: Results for descending until the 10th level. (A) Success rate for ACER and Random agent at different depths, varying the maximum steps per episode. (B) Average number of levels descended. All rates have been computed over 1000 episodes



```
                           ---+---
                           |.....|
                           +.....|
                           |..*..|
                           |.....|
                           -+-----
                            #
                            #
                  ###########
                --+----
                |      |
      ...   ####+      |
      .@.|  #   -----
      ...+###
----------

Level: 4  Gold: 64     Hp: 12(12)  Str: 16(16)  Arm: 4   Exp: 1/0  Cmd: 530
```

(a)



```
                                   # #####
                #                  # #   #
---------     ####             ### ###@# #
        |     # #              # #       #
      ! +##### #     ###     ## ####### #
-+-   ---       #     # #      #       # #
 #              ####### #######     ######
 #####
###  #                ----  ------
   --+--              |          |
   +...+##            |          |
   |...| #############+          |
   ---+-              |

Level: 4  Gold: 89     Hp: 12(12)  Str: 16(16)  Arm: 4   Exp: 1/0  Cmd: 1200
```

(b)

Fig. 8: (a) Dark rooms (b) A labyrinth (surrounded by dark rooms)

given more training time (or computational power) our model could have reached better results.

B. Recovering the amulet

Our second experiment consists in trying to retrieve the amulet of Yendor, provided it is placed in an early level, specifically the first or the second. After recovering the relic, in order to win the game, the agent has to find and ascend the stairs in as many level as it descended (and they will be different from the corresponding level during descent), coming back to surface.

Retrieving the amulet does not require a special action. Similarly to any other object in Rogue, the player automatically takes it by just stepping on it.

We adopt a slight simplification with respect to the standard Rogue game; instead of having two separate moves for going down and going up the stairs, we use a single action for taking the stairs: before reaching the amulet level, this is understood as going down, and after that, as going up. In this way, the total number of action is the same as for the previous experiments.

The main point of interest of placing the amulet in the very first floor, is mostly to evaluate the navigation skills of our agent once it has recovered the amulet and already discovered the location of the stairs. With respect to these issues, we obtained interesting and partially unexpected results. Here we set the maximum amount of steps per episode back to 500.

On the other hand, the purpose of placing the amulet at the second floor is to show the limit of our learning configuration and set the bar for future work. We expected terrible results and were not surprised in this regard. Here we set the maximum amount of steps per episode to 1000, that should be more than enough.

For these new endeavors we persist with our ACER configuration, slightly tweaking it by:

1) adding a layer in our state representation, showing the position of the amulet with a 1 and 0 in all other positions;
2) positively rewarding stepping on the amulet and winning the game; anything else is rewarded with 0.

The previous rewarding strategy is possibly a cause of the bad behaviors we get when the amulet is at deeper levels. In fact, we give no reward at all for descending, and the agent is supposed to learn very complex and long sequences of actions without any intermediate incentive.

In this case, in addition to the usual metrics, we also compute the average number of times the amulet is recovered.

We show the results of this first experiment in Figure 9 and Table IV. As expected, this task is harder than simply descending the first level: the agent is able to accomplish its goal in about $\sim 87\%$ of times, (although the learning curves were still in a slowly ascending trend when we interrupted the training). This behavior is probably due to a new skill required for this endeavor: pathfinding to a known location. We see from Table IV that the agent discovers the stairs position

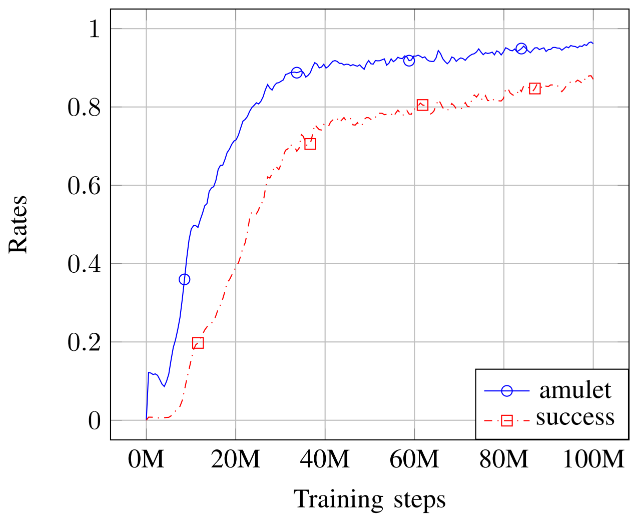


Fig. 9: Results of ACER recovering the amulet from the first level

Statistic	ACER	random
Success rate	87.06%	2.00%
Amulet recovered rate	96.16%	14.50%
Pathfinding to stairs rate	90.54%	13.79%
Pathfinding to undiscovered stairs rate	94.75%	6.21%
Pathfinding to previously found stairs rate	86.95%	44.00%
Stairs found before amulet rate	52.10%	17.24%
Avg number of steps to succeed	205	274
Avg number of seen tiles	486	106

TABLE IV: Final results of ACER recovering the amulet from the first level

before the amulet in about half of the cases. In this kind of scenario it is able to succeed $\sim 87\%$ of the times after recovering the amulet, while in the opposite case - when it has yet to uncover the stairs - it is able to win almost 95% of the games. Sometimes the agents shows some difficulty in stepping toward the right direction and on some occasions it is not able to reach the stairs in time. For instance, when the path to the previously discovered stairs requires a significant detour, like in figure 10, the agent frequently attempts to cross walls many times before taking the correct path. This imperfect pathfinding is an indication of where to focus the attention in future works and is a symptom of something defective, either in the algorithms or in the neural network architecture.

Moving on to the second experiment, if the amulet is placed in the second floor the agent is not able to learn any more, essentially behaving as a random agent (see Table V).

Statistic	ACER	random
Success rate	0.06%	0%
Amulet recovered rate	2.06%	1.00%
First level descent rate	13.34%	12.50%
Second level ascent rate	0.06%	0%
Avg number of steps to succeed	59	–
Avg number of seen tiles	143	137

TABLE V: Final results of ACER recovering the amulet from the second level

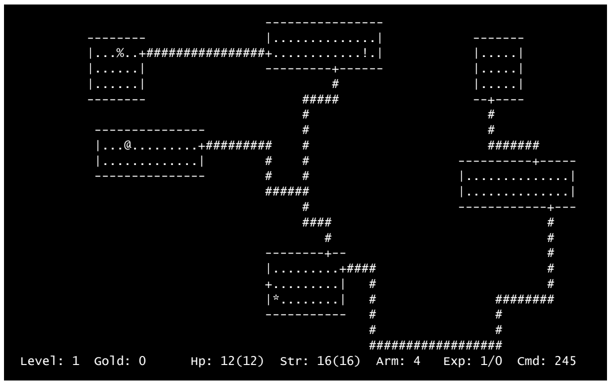

```
                                  ----------------
         --------                 |..............|            -------
         |...%..+#################+............!.|            |.....|
         |......|                 ---------+------            |.....|
         |......|                          #                  |.....|
         --------                      #####                  --+----
                                       #                        #
          ---------------              #                        #
          |...@.........+#########     #                        #######
          |.............|         #    #                    ----------+-----
          ---------------         #    #                    |..............|
                                  ######                    |..............|
                                       #                    ------------+---
                                       ####                             #
                                          #                             #
                                  --------+--                           #
                                  |.........+####                       #
                                  +.........|   #                       #
                                  |*........|   #                ########
                                  -----------   #                #
                                                #                #
                                                ##################
Level: 1  Gold: 0      Hp: 12(12)  Str: 16(16)  Arm: 4    Exp: 1/0  Cmd: 245
```

Fig. 10: ACER agent attempting to cross walls to reach the stairs after having recovered the amulet

The problem is that the probability of quasi-randomly recovering the amulet from the second level is not enough to teach the agent anything useful, therefore it keeps moving around haphazardly even after 100 million training steps. This issue may be related to the exceedingly sparse rewards, but may also be affected by the learning algorithm and the n-step length of 60, which may be too low in this scenario. Maybe, in this case too, an improved form of experience replay could suffice to solve the issue.

VI. CONCLUSIONS

In this article we compared several recent Deep Reinforcement techniques applied to the famous Rogue video game of the 80s, restricting our goals to exploration only. This ruled out many interesting aspects of the game, (dealing with enemies, inventory management, discovering secret passages, some textual components, etc.); unfortunately, the mere task of exploration already proved to be *much more difficult* than expected (and it is not entirely solved yet). It looks important to separately address the different aspects of the game, in order to appreciate the benefits of the different techniques: until exploration is not properly solved there is no point in adding additional complexity. On the other hand, by ruling out the need of collecting objects and the associated rewards we are (intentionally) exacerbating the problem of sparse rewards, posing an interesting challenge for most of the known deep reinforcement techniques.

In games like Rogue, sparsity of rewards is somehow mitigated by the heterogeneity of the dungeon maps: while the dungeon exit is *in average* quite far (80-85 steps), it may happen that in some lucky cases it is just a few steps away from the rogue, providing a handy reward that can be profitably exploited for learning.

The problem is hence to manage generalization in a regime of relatively sparse rewards. According to our experiments, actor-critic techniques with a good form of experience replay to improve sample efficiency, and exploiting sufficiently long temporal sequences of states via long-short term memory models, seem to be sufficient to solve the issue in non-static environments (like Rogue, unlike the Atari game Montezuma's Revenge), with no need of additional intrinsic rewards. On the task of exploring the map and taking the stairs, we have been able to achieve an average success rate of about 98%, *just rewarding the completion of the episode*. Specifically, we used a variant of ACER [5] with temporal sequences of length 60, that outperformed DQN [1] (even enriched with counting and other forms of intrinsic or extrinsic rewards) and A3C [3]. In a previous work [20], we had been able to obtain similar performances with A3C, but at the price of a much higher domain knowledge, expressed in the form of procedural dispatching of states relative to different game situations to specific sub-networks.

The exploration problem in not yet entirely solved; in particular, the problem of secret passages seems particularly hard to tackle. Similarly, other aspects of the game should be integrated and dealt with in the future, likely requiring different techniques.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[2] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[5] Z. Wang, "Sample efficient actor-critic with experience replay," 2016.

[6] V. Cerny and F. Dechterenko, "Rogue-like games as a playground for artificial intelligence–evolutionary approach," in *International Conference on Entertainment Computing*. Springer, 2015, pp. 261–271.

[7] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[8] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2017–2025. [Online]. Available: http://papers.nips.cc/paper/5854-spatial-transformer-networks

[9] N. Dilokthanakul, C. Kaplanis, N. Pawlowski, and M. Shanahan, "Feature control as intrinsic motivation for hierarchical reinforcement learning," *CoRR*, vol. abs/1705.06769, 2017. [Online]. Available: http://arxiv.org/abs/1705.06769

[10] S. P. Singh, A. G. Barto, and N. Chentanez, "Intrinsically motivated reinforcement learning," in *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, 2004, pp. 1281–1288. [Online]. Available: http://papers.nips.cc/paper/2552-intrinsically-motivated-reinforcement-learning

[11] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Empowerment: a universal agent-centric measure of control," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*, 2005, pp. 128–135. [Online]. Available: https://doi.org/10.1109/CEC.2005.1554676

[12] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.

[13] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *CoRR*, vol. abs/1611.05397, 2016. [Online]. Available: http://arxiv.org/abs/1611.05397

[14] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *CoRR*, vol. abs/1604.06057, 2016. [Online]. Available: http://arxiv.org/abs/1604.06057

[15] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," *CoRR*, vol. abs/1703.01161, 2017. [Online]. Available: http://arxiv.org/abs/1703.01161

[16] A. Asperti, C. De Pieri, and G. Pedrini, "Rogueinabox: an environment for roguelike learning," *International Journal of Computers*, vol. 2, pp. 146–154, 2017. [Online]. Available: http://www.iaras.org/iaras/filedownloads/ijc/2017/006-0022(2017).pdf

[17] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res. (JAIR)*, vol. 47, pp. 253–279, 2013. [Online]. Available: http://dx.doi.org/10.1613/jair.3912

[18] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "Vizdoom: A doom-based AI research platform for visual reinforcement learning," *CoRR*, vol. abs/1605.02097, 2016. [Online]. Available: http://arxiv.org/abs/1605.02097

[19] A. Asperti, C. De Pieri, M. Maldini, G. Pedrini, and F. Sovrano, "A modular deep-learning environment for rogue," *WSEAS Transactions on Systems and Control*, vol. 12, 2017. [Online]. Available: http://www.wseas.org/multimedia/journals/control/2017/a785903-070.php

[20] A. Asperti, D. Cortesi, and F. Sovrano, "Crawling in rogue's dungeons with (partitioned) A3C," in *The Fourth International Conference on Machine Learning, Optimization, and Data Science. Volterra, Tuscany, Italy*, vol. to appear. Springer, 2018.

[21] A. Asperti, D. Cortesi, C. De Pieri, G. Pedrini, and F. Sovrano, "Rogueinabox: a rogue environment for ai learning," https://github.com/rogueinabox/rogueinabox_lib.

[22] A. Asperti, D. Cortesi, C. De Pieri, and G. Pedrini, "Rogue custom build for rogueinabox lib," https://github.com/rogueinabox/rogue.

[23] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 1471–1479. [Online]. Available: http://papers.nips.cc/paper/6383-unifying-count-based-exploration-and-intrinsic-motivation

[24] L. P. Kaelbling. Value iteration. [Online]. Available: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/node19.html

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[26] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[27] M. Stolle and D. Precup, "Learning options in reinforcement learning," in *International Symposium on abstraction, reformulation, and approximation*. Springer, 2002, pp. 212–223.

[28] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture." in *AAAI*, 2017, pp. 1726–1734.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[30] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 1054–1062.

[31] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

[32] A. Asperti, D. Cortesi, C. De Pieri, and G. Pedrini, "Openai acer for rogueinabox," https://github.com/rogueinabox/openai_acer.

[33] A. Asperti, D. Cortesi, C. De Pieri, G. Pedrini, and F. Sovrano. Playing rogue with acer – receptive fields visualization. Youtube. [Online]. Available: https://www.youtube.com/watch?v=XF9-arwFn3c

[34] D. Cortesi, "Reinforcement learning in rogue," Master Thesis, University of Bologna, 2018.