Training Agents using Upside-Down Reinforcement Learning

Rupesh Kumar Srivastava[1*] Pranav Shyam[1†] Filipe Mutz[2‡]
Wojciech Jaśkowski[1] Jürgen Schmidhuber[12]

[1]NNAISENSE
[2]The Swiss AI Lab IDSIA

NNAISENSE Technical Report

Abstract

Traditional Reinforcement Learning (RL) algorithms either predict rewards with value functions or maximize them using policy search. We study an alternative: Upside-Down Reinforcement Learning (Upside-Down RL or ⅂ꓤ), that solves RL problems primarily using supervised learning techniques. Many of its main principles are outlined in a companion report [34]. Here we present the first concrete implementation of ⅂ꓤ and demonstrate its feasibility on certain episodic learning problems. Experimental results show that its performance can be surprisingly competitive with, and even exceed that of traditional baseline algorithms developed over decades of research.

1 Introduction

While there is a rich history of techniques that incorporate supervised learning (SL) into reinforcement learning (RL) algorithms, it is believed that fully solving RL problems using SL is not possible, because feedback from the environment provides *error signals* in SL but *evaluation signals* in RL [2, 30]. Put simply, an agent gets feedback about how useful its actions are, but not about which actions are the best to take in any situation. On the possibility of turning an RL problem into an SL problem, Barto and Dietterich [2] surmised: "In general, there is no way to do this."

In a companion technical report, Schmidhuber [34] proposes to bridge this gap between SL and RL through *Upside-Down Reinforcement Learning* (⅂ꓤ), where environmental feedback – such as the *reward* – is an input rather than the learning target as in traditional RL algorithms based on reward prediction [37]. Here we develop a practical ⅂ꓤ algorithm for episodic tasks and show that it is indeed possible to train agents in general model-free settings[1] without using value-based algorithms such as Q-learning [41], or policy-based ones such as policy gradients and evolutionary algorithms [22, 42]. Instead, ⅂ꓤ uses pure SL to train an agent on all past experiences, and sidesteps the issues arising from the combination of function approximation, bootstrapping and off-policy training [37]. We first describe its basic principles, then experimentally demonstrate its practical feasibility on three RL problems with both sparse and dense reward structure.

[*]Correspondence to: rupesh@nnaisense.com
[†]Now at OpenAI.
[‡]Now at IFES, Brazil.
[1]Stochastic environments with high-dimensional inputs, scalar and possibly sparse rewards, no expert demonstrations.

使用颠倒强化学习训练代理

Rupesh Kumar Srivastava[1*] Pranav Shyam[1†] Filipe Mutz[2‡] W
ojciech Ja kowski[1] Jürgen Schmidhuber[12]

1NNAISENSE [2]瑞
士AI实验室IDSIA

NNAISENSE Technical Report

摘要

传统的强化学习（RL）算法要么使用价值函数来预测奖励，要么使用策略搜索来最大化奖励。我们研究了另一种方法：倒置强化学习（Upside-Down RL 或 RL），它主要使用监督学习技术来解决RL问题。其许多主要原则在一份配套报告[34]中有所概述。在这里，我们提出了RL的第一个具体实现，并在某些 episodic 学习问题上展示了其可行性。实验结果表明，其性能可以出人意料地与数十年研究中开发的传统基线算法竞争，甚至超越它们。

1 引言

尽管存在将监督学习（SL）纳入强化学习（RL）算法中的丰富技术历史，但人们相信，完全使用SL解决RL问题是不可能的，因为环境反馈在SL中提供了错误信号，在RL中提供了评估信号[2, 30]。简单来说，智能体收到关于其行为有用性的反馈，但不会收到关于在任何情况下应采取哪些行为的反馈。关于将RL问题转换为SL问题的可能性，Barto和Dietterich[2]推测：“一般来说，这是不可能的。”

在一篇配套的技术报告中，Schmidhuber [34] 提出通过上下反转强化学习（RL）来弥合监督学习（SL）和强化学习（RL）之间的差距，其中环境反馈——例如奖励——是输入而不是传统基于奖励预测的RL算法中的学习目标[37]。在这里，我们开发了一种实用的RL算法来处理阶段性任务，并展示了确实可以在一般无模型设置中[1]训练代理而不使用基于价值的方法（如Q学习[41]），或基于策略的方法（如策略梯度和进化算法[22, 42]）。相反，RL 使用纯粹的监督学习来训练代理所有过往的经验，并绕过了函数逼近、自举和离策略训练带来的问题[37]。我们首先描述其基本原理，然后通过三个具有稀疏和密集奖励结构的RL问题的实验来证明其实用可行性。

*Correspondence to: rupesh@nnaisense.com

†Now at OpenAI.

‡Now at IFES, Brazil.

[1]Stochastic environments with high-dimensional inputs, scalar and possibly sparse rewards, no expert demonstrations.

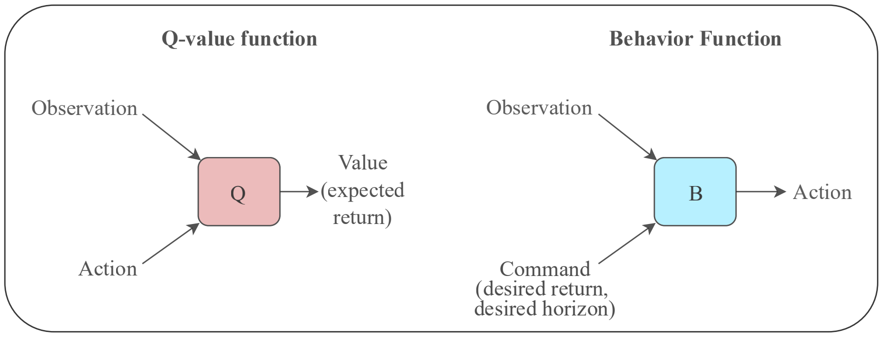


Figure 1: A key distinction between the action-value function (Q) in traditional RL (e.g. Q-learning) and the behavior function (B) in ⅂ᴚ is that the roles of actions and returns are switched. In addition, B may have other command inputs such as desired states or the desired time horizon for achieving a desired return.

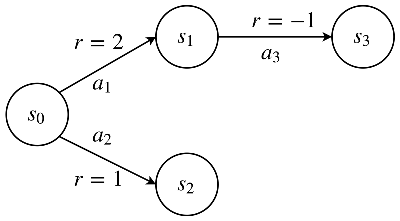


Figure 2: A toy environment with four discrete states.

Table 1: A behavior function for the toy environment.

State	Desired Return	Desired Horizon	Action
s_0	2	1	a_1
s_0	1	1	a_2
s_0	1	2	a_1
s_1	-1	1	a_3

2 Upside-Down Reinforcement Learning

2.1 Terminology & Notation

In what follows, s, a and r denote *state*, *action*, and *reward* respectively. The sets of values of s and a ($\mathcal{S}$ and $\mathcal{A}$) depend on the environment. Right subscripts denote time indices (e.g. $s_t, t \in \mathbb{N}^0$). We consider the Markovian environments with scalar rewards ($r \in \mathbb{R}$) as is typical, but the general principles of ⅂ᴚ are not limited to these settings. A *policy* $\pi : \mathcal{S} \rightarrow \mathcal{A}$ is a function that selects an action in a given state. A policy can be stochastic, in which case it maps a state to a probability distribution over actions. Each *episode* consists of an agent's interaction with the environment starting in an initial state and ending in a terminal state while following any policy. A *trajectory* τ is the sequence $\langle(s_t, a_t, r_t, s_{t+1})\rangle, t = 0, \ldots, T-1$ containing data describing an episode of length T. We refer to any subsequence of a trajectory as a *segment* or a *behavior*, and the cumulative reward over a segment as the *return*.

2.2 Knowledge Representation

Traditional model-free RL algorithms can be broadly classified as being *value-based* or *policy-based*. The core principle of value-based algorithms is reward prediction: agents are trained to predict the expected discounted future return for taking any action in any state, commonly using TD learning. Policy-based algorithms are instead based on directly searching for policies that maximize returns. The basic principle of ⅂ᴚ are different from both of these categories: given a particular definition of *commands*, it defines a *behavior function* that encapsulates knowledge about the behaviors observed so far compatible with known commands. The nature of the behavior function is explained using two examples below.

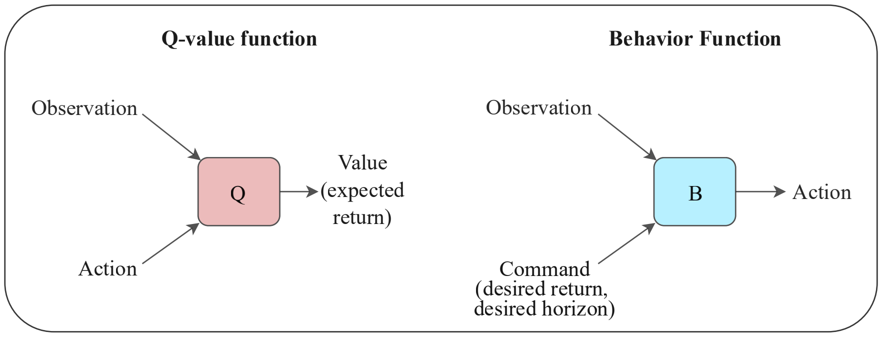


图1：传统RL（例如Q-学习）中的动作值函数(Q)与RL中的行为函数(B)之间的关键区别在于动作和回报的角色被交换了。此外，B可能还有其他命令输入，如期望状态或实现期望回报所需的时间 horizon。

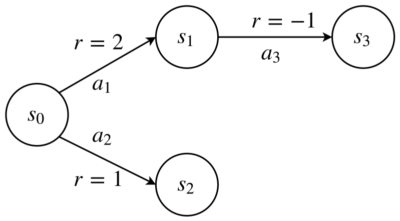


图 2: 一个具有四个离散状态的玩具环境。

表 1: 用于玩具环境的行为函数。

State	Desired Return	Desired Horizon	Action
s_0	2	1	a_1
s_0	1	1	a_2
s_0	1	2	a_1
s_1	-1	1	a_3

2 upside-down 强化学习

2.1 术语与符号notation

在以下内容中，s，a 和 r 分别表示状态、动作和奖励。s，a ($\mathcal{S}$ 和 $\mathcal{A}$) 的取值集取决于环境。右下标表示时间索引（例如 $s_t, t \in \mathbb{N}^0$）。我们考虑具有标量奖励的马尔可夫环境（$r \in \mathbb{R}$），这是典型的，但 RL 的一般原理不限于这些设置。策略 $\pi : \mathcal{S} \rightarrow \mathcal{A}$ 是一个函数，它在给定状态下选择一个动作。策略可以是随机的，在这种情况下，它将状态映射到动作的概率分布上。每个回合由代理与环境的交互组成，从初始状态开始，直到终止状态结束，遵循任何策略。轨迹 τ 是长度为 T 的序列 $\langle (s_t, a_t, r_t, s_{t+1}) \rangle, t = 0, \ldots, T-1$，描述一个回合的数据。我们将轨迹的任何子序列称为片段或行为，并将片段上的累积奖励称为回报。

2.2 知识表示

传统模型自由的强化学习算法可以大致分为基于值的方法和基于策略的方法。基于值的算法的核心原则是奖励预测：代理被训练来预测在任何状态下采取任何行动的期望折现未来回报，通常使用TD学习。基于策略的算法则直接搜索能够最大化回报的策略。强化学习的基本原则不同于这两类：给定特定的命令定义，它定义一个行为函数来封装到目前为止与已知命令兼容的行为知识。行为函数的性质将在下面两个例子中进行解释。

Illustrative Example 1. Consider a simple dart-throwing environment where each episode lasts a single step. An agent learning to throw darts receives a return inversely proportional to the hit distance from the center of the board. In each episode, the agent observes the initial state of the dart, and takes an action that determines the force and direction of the throw. Using value-based RL for this task would amount to training the agent to predict the expected return for various actions and initial states. This knowledge would then be used for action selection e.g. taking the action with the highest expected return.

In ⅂ᴚ, the agent's knowledge is represented not in terms of expected returns for various states and actions, but in terms of actions that are compatible with various states and desired returns i.e. the inputs and targets of the agent's learning procedure are switched. The dart throwing agent would be trained to directly produce the actions for hitting desired locations on the board, using a behavior function B[2] learned using its past experience. Figure 1 schematically illustrates this difference between B and the Q-value function commonly used in value-based RL. Since this environment consists of episodes with a single time step, both Q and B can be learned using SL. The next example illustrates a slightly more typical RL setting with longer time horizons.

Illustrative Example 2. Consider the simple deterministic Markovian environment in Figure 2 in which all trajectories start in s_0 or s_1 and end in s_2 or s_3. Additionally consider *commands* of the type: achieve a given desired return in a given desired horizon from the current state. A behavior function based on the set of all unique behaviors possible in this environment can then be expressed in a tabular form in Table 1. It maps states and commands to the action to be taken in that state compatible with executing the command. In other words, it answers the question: "if an agent is in a given state and desires a given return over a given horizon, which action should it take next?" By design, this function can now be used to execute any valid command in the environment without further knowledge.

Two properties of the behavior function are notable. First, the output of B can be stochastic even in a deterministic environment since there may be multiple valid behaviors compatible with the same command and state. For example, this would be the case if the transition $s_0 \to s_2$ had a reward of 2. So in general, B produces a probability distribution over actions. Second, B fundamentally depends on the set of trajectories used to construct it. Using a loss function L, we define the optimal behavior function $B^*_{\mathcal{T}}$ for a set of trajectories $\mathcal{T}$ as

$$\begin{aligned} B^*_{\mathcal{T}} = \arg\min_{B} \sum_{(t_1,t_2)} & L(B(s_{t_1}, d^r, d^h), a_{t_1}), \\ \text{where } & 0 < t_1 < t_2 < \mathrm{len}(\tau)\ \forall\, \tau \in \mathcal{T}, \\ d^r = & \sum_{t=t_1}^{t_2} r_t \text{ and } d^h = t_2 - t_1. \end{aligned} \tag{1}$$

Here $\mathrm{len}(\tau)$ is the length of any trajectory τ. For a suitably parameterized B, we use the cross-entropy between the observed and predicted distributions of actions as the loss function. Equivalently, we search for parameters that maximize the likelihood that the behavior function generates the available data, using the traditional tools of supervised learning. Similarly, we can define a behavior function *over a policy*. Instead of a set of trajectories, B^*_{π} minimizes the same loss over the distribution of trajectories generated when acting according to π.

2.3 An ⅂ᴚ Algorithm for Maximizing Episodic Returns

In principle, a behavior function can be learned for any policy that generates all possible trajectories in an environment given sufficient time (e.g. a random policy) and then used to select actions that lead to any desired return in a desired horizon achievable in the environment. But such a learning procedure is not practical since it relies on undirected exploration using a fixed policy. Moreover, in environments with scalar rewards, the goal is to learn to achieve high

[2]Denoted C by Schmidhuber [34]. We use B here for compatibility with the "behavior function" nomenclature.

示例 1. 考虑一个简单的飞镖投掷环境，其中每个episode持续一个步骤。一个学习投飞镖的智能体收到的回报与飞镖击中靶心的距离成反比。在每个episode中，智能体观察飞镖的初始状态，并采取一个行动来决定投掷的力和方向。使用基于值的强化学习解决这个问题意味着训练智能体预测各种行动和初始状态的期望回报。然后，这些知识将用于行动选择，例如选择期望回报最高的行动。

在RL中，代理的知识不是用各种状态和动作的预期回报来表示的，而是用与各种状态和期望回报兼容的动作来表示，即代理学习过程的输入和目标被交换了。飞镖投掷代理将被训练直接产生击中板上所需位置的动作，使用其过去经验学习的行为函数B[2]。图1以示意图的方式说明了B和在基于值的RL中常用的Q值函数之间的差异。由于该环境由单个时间步长的时期组成，因此Q和B都可以使用SL来学习。下一个示例说明了一个时间范围稍长的更典型的RL设置。

示例 2. 考虑图 2 中的简单确定性马尔可夫环境，其中所有轨迹从 s_0 或 s_1 开始并结束于 s_2 或 s_3。此外，考虑类型为：从当前状态实现给定的期望回报和给定的期望时间范围的命令。然后，基于此环境中所有可能的独特行为集，行为函数可以以表 1 的表格形式表示。它将状态和命令映射到在该状态下应采取的动作，以执行该命令。换句话说，它回答了这样的问题："如果一个代理处于给定状态并期望在给定的时间范围内获得回报，它应该采取什么后续行动？" 通过设计，此函数现在可以用于在没有进一步知识的情况下执行环境中的任何有效命令。

行为函数的两个性质值得注意。首先，即使在确定性环境中，B 的输出也可以是随机的，因为可能存在多种与同一命令和状态兼容的有效行为。例如，如果转换$s_0 \to s_2$ 的奖励为 2，这种情况就会发生。因此，一般来说，B 会产生一个动作的概率分布。其次，B 在很大程度上依赖于用于构建它的轨迹集。使用损失函数L，我们将一组轨迹$\mathcal{T}$的最优行为函数$B^*_{\mathcal{T}}$定义为

$$\begin{aligned} B^*_{\mathcal{T}} &= \arg\min_B \sum_{(t_1,t_2)} L(B(s_{t_1}, d^r, d^h), a_{t_1}), \\ &\text{where } 0 < t_1 < t_2 < \text{len}(\tau)\ \forall\, \tau \in \mathcal{T}, \\ &d^r = \sum_{t=t_1}^{t_2} r_t \text{ and } d^h = t_2 - t_1. \end{aligned} \tag{1}$$

这里 len(τ) 是任何轨迹 τ 的长度。对于一个适当参数化的 B，我们使用观察到的动作和预测的动作分布之间的交叉熵作为损失函数。等价地，我们使用监督学习的传统工具来寻找使行为函数生成可用数据的概率最大的参数。类似地，我们可以定义一个策略上的行为函数。而不是一组轨迹，B^*_π 在根据 π 行动生成的轨迹分布上最小化相同的损失。

2.3 一个用于最大化 episodic 返回的 RL 算法

原则上，可以为任何生成环境内所有可能轨迹的策略（例如，随机策略）学习一个行为函数，然后使用该函数选择能够达到任何所需回报的动作。但这种学习过程是不实际的，因为它依赖于使用固定策略的无向探索。此外，在具有标量奖励的环境中，目标是学会实现高

[2]Denoted C by Schmidhuber [34]. We use B here for compatibility with the "behavior function" nomenclature.

Algorithm 1 Upside-Down Reinforcement Learning: High-level Description.

```
1:  Initialize replay buffer with warm-up episodes using random actions        // Section 2.3.1
2:  Initialize a behavior function                                             // Section 2.3.2
3:  while stopping criteria is not reached do
4:      Improve the behavior function by training on replay buffer             // Exploit; Section 2.3.3
5:      Sample exploratory commands based on replay buffer                     // Section 2.3.4
6:      Generate episodes using Algorithm 2 and add to replay buffer           // Explore; Section 2.3.5
7:      if evaluation required then
8:          Evaluate current agent using Algorithm 2                           // Section 2.3.6
9:      end if
10: end while
```

returns and not to achieve any possible return over any horizon. Therefore, the concrete algorithm used in this paper trains a behavior function on the set of trajectories (or the agent's experience) so far and incorporates minimal additions that enable the continual collection of trajectories with higher returns.

High-level pseudo-code for the proposed algorithm is described in Algorithm 1. It starts by initializing an empty replay buffer to collect the agent's experiences during training, and filling it with a few episodes of random interactions. The behavior function of the agent is continually improved by supervised training on previous experiences recorded in the replay buffer. After each training phase, the behavior function is used to act in the environment to obtain new experiences that are added to the replay buffer. This procedure continues until a stopping criterion is met, such as reaching the allowed maximum number of interactions with the environment. The remainder of this section describes each step of the algorithm and introduces the hyperparameters. A concise list of hyperparameters is also provided in Appendix A.

2.3.1 Replay Buffer

⅂ꓤ does not explicitly maximize returns, but instead relies on exploration to continually discover higher return trajectories so that the behavior function can be trained on them. To drive learning progress, we found it helpful to use a replay buffer containing a fixed maximum number of trajectories with the highest returns seen so far, sorted in increasing order by return. The maximum buffer size is a hyperparameter. Since the agent starts learning with zero experience, an initial set of trajectories is generated by executing random actions in the environment. The trajectories are added to the replay buffer and used to start training the agent's behavior function.

2.3.2 Behavior Function

As described earlier, at any time t during an episode, the current behavior function B produces an action distribution in response to the current state s_t and command $c_t := (d^r_t, d^h_t)$, where $d^r_t \in \mathbb{R}$ is the *desired return* and $d^h_t \in \mathbb{N}$ is the *desired time horizon* at time t. The predicted action distribution $P(a_t|s_t, c_t) = B(s_t, c_t; \theta)$, where θ denotes a vector of trainable parameters, is expected to lead to successful execution of the command c_t interpreted as: "achieve a return d^r_t during the next d^h_t steps". For a given initial command input c_0, B can be used to generate a trajectory using Algorithm 2 by sampling actions predicted for the current command and updating the command according to the obtained rewards and elapsed time.

An important implementation detail is that d^h_t is always set to $\max(d^h_t, 1)$ such that it is a valid time horizon. Furthermore, d^r_t is clipped such that it is upper-bounded by the maximum return achievable in the environment. This only affects agent evaluations (not training) and avoids situations where negative rewards (r_t) can lead to desired returns that are not achievable from any state (see Algorithm 2; line 8).

Algorithm 1 Upside-Down Reinforcement Learning: High-level Description.

```
1: Initialize replay buffer with warm-up episodes using random actions        // Section 2.3.1
2: Initialize a behavior function                                              // Section 2.3.2
3: while stopping criteria is not reached do
4:     Improve the behavior function by training on replay buffer              // Exploit; Section 2.3.3
5:     Sample exploratory commands based on replay buffer                       // Section 2.3.4
6:     Generate episodes using Algorithm 2 and add to replay buffer            // Explore; Section 2.3.5
7:     if evaluation required then
8:         Evaluate current agent using Algorithm 2                             // Section 2.3.6
9:     end if
10: end while
```

返回值而不要在任何时间范围内实现任何可能的回报。因此，本文中使用的具体算法在迄今为止的轨迹集（或代理的经验）上训练一个行为函数，并引入最小的附加项以实现持续收集更高返回值的轨迹。

提出的算法的高级伪代码描述在算法1中。它首先通过初始化一个空的重放缓冲区来收集代理在训练期间的经验，并用一些随机交互的几个回合填充它。代理的行为函数通过在重放缓冲区中记录的先前经验上的监督训练不断得到改进。每次训练阶段结束后，使用行为函数在环境中采取行动以获得新的经验，这些经验被添加到重放缓冲区中。该过程将持续进行，直到满足停止标准，例如达到允许的最大交互次数。本节的其余部分描述了算法的每一步并介绍了超参数。超参数的简洁列表也在附录A中提供。

2.3.1 回放缓冲区

RL 不是显式地最大化回报，而是依靠探索不断发现更高的回报轨迹，从而使行为函数能够基于这些轨迹进行训练。为了推动学习进程，我们发现使用一个包含到目前为止看到的最高回报的固定最大数量轨迹的重放缓冲区是有帮助的，这些轨迹按回报的升序排序。最大缓冲区大小是一个超参数。由于智能体开始学习时没有任何经验，因此初始的一组轨迹是通过在环境中执行随机动作生成的。这些轨迹被添加到重放缓冲区，并用于开始训练智能体的行为函数。

2.3.2 行为函数

如前所述，在任何时间t期间的一次体验中，当前行为函数B会根据当前状态s_t和命令$c_t := (d_t^r, d_t^h)$生成一个动作分布，其中$d_t^r \in \mathbb{R}$是期望的回报，$d_t^h \in \mathbb{N}$是期望的时间范围在时间t。预测的动作分布$P(a_t|s_t, c_t) = B(s_t, c_t; \theta)$，其中$\theta$表示一组可训练参数，预期能够导致成功执行由命令$c_t$解释为“在接下来的$d_t^h$步中获得回报$d_t^r$”的命令。对于给定的初始命令输入$c_0$，可以通过采样为当前命令预测的动作并根据获得的奖励和经过的时间更新命令来使用B生成轨迹（使用算法2）。

一个重要的实现细节是，d_t^h 总是被设置为 $\max(d_t^h, 1)$ 以确保它是一个有效的时间范围。此外，d_t^r 被裁剪，使其不超过在环境中可实现的最大回报。这仅影响智能体评估（而不是训练），并避免了由于负奖励 (r_t) 导致无法从任何状态实现的期望回报的情况（参见算法 2；第 8 行）。

Algorithm 2 Generates an Episode using the Behavior Function.

```
Input: Initial command $c_0 = (d^r_0, d^h_0)$, Initial state $s_0$, Behavior function $B(;\theta)$
Output: Episode data $E$
1:  $E \leftarrow \varnothing$
2:  $t \leftarrow 0$
3:  while episode is not over do
4:      Compute $P(a_t|s_t, c_t) = B(s_t, c_t; \theta)$
5:      Execute $a_t \sim P(a_t|s_t, c_t)$ to obtain reward $r_t$ and next state $s_{t+1}$ from the environment
6:      Append $(s_t, a_t, r_t)$ to $E$
7:      $s_t \leftarrow s_{t+1}$                                  // Update state
8:      $d^r_t \leftarrow d^r_t - r_t$                          // Update desired reward
9:      $d^h_t \leftarrow d^h_t - 1$                            // Update desired horizon
10:     $c_t \leftarrow (d^r_t, d^h_t)$
11:     $t \leftarrow t + 1$
12: end while
```

2.3.3 Training the Behavior Function

As discussed in Section 2.2, B admits supervised training on a large amount of input-target examples from any past episode. The goal of training is to make the behavior function produce outputs consistent with all previously recorded trajectories in the replay buffer according to Equation 1.

To draw a training example from a random episode in the replay buffer, time step indices t_1 and t_2 are selected randomly such that $0 \leq t_1 < t_2 \leq T$, where T is the length of the selected episode. Then the input for training B is $(s_{t_1}, (d^r, d^h))$, where $d^r = \sum_{t=t_1}^{t_2} r_t$ and $d^h = t_2 - t_1$, and the target is a_{t_1}, the action taken at t_1. To summarize, the training examples are generated by selecting the time horizons, actions, observations and rewards in the past, and generating input-target pairs consistent with them.

Several heuristics may be used to select and combine training examples into mini-batches for gradient-based SL. For all experiments in this paper, only "trailing segments" were sampled from each episode, i.e., we set $t_2 = T - 1$ where T is the length of any episode. This discards a large amount of potential training examples but is a good fit for episodic tasks where the goal is to optimize the total reward until the end of each episode. It also makes training easier, since the behavior function only needs to learn to execute a subset of possible commands. To keep the setup simple, a fixed number of training iterations using Adam [13] were performed in each training step for all experiments.

2.3.4 Sampling Exploratory Commands

After each training phase, the agent can attempt to generate new, previously infeasible behavior, potentially achieving higher returns. To profit from such exploration through generalization, one must first create a set of new initial commands c_0 to be used in Algorithm 2. We use the following procedure to sample commands:

1. A number of episodes from the end of the replay buffer (i.e., with the highest returns) are selected. This number is a hyperparameter and remains fixed during training.
2. The exploratory desired horizon d^h_0 is set to the mean of the lengths of the selected episodes.
3. The exploratory desired returns d^r_0 are sampled from the uniform distribution $\mathcal{U}[M, M + S]$ where M is the mean and S is the standard deviation of the selected episodic returns.

This procedure was chosen due to its simplicity and ability to adjust the strategy using a single hyperparameter. Intuitively, it tries to generate new behavior (aided by environmental stochasticity) that achieves returns at the edge of the best known behaviors in the replay. While Schmidhuber [34] notes that a variety of heuristics may be used here,

Algorithm 2 Generates an Episode using the Behavior Function.

```
Input: Initial command $c_0 = (d_0^r, d_0^h)$, Initial state $s_0$, Behavior function $B(;\theta)$
Output: Episode data $E$
1:  $E \leftarrow \varnothing$
2:  $t \leftarrow 0$
3:  while episode is not over do
4:      Compute $P(a_t|s_t, c_t) = B(s_t, c_t; \theta)$
5:      Execute $a_t \sim P(a_t|s_t, c_t)$ to obtain reward $r_t$ and next state $s_{t+1}$ from the environment
6:      Append $(s_t, a_t, r_t)$ to $E$
7:      $s_t \leftarrow s_{t+1}$                                   // Update state
8:      $d_t^r \leftarrow d_t^r - r_t$                             // Update desired reward
9:      $d_t^h \leftarrow d_t^h - 1$                               // Update desired horizon
10:     $c_t \leftarrow (d_t^r, d_t^h)$
11:     $t \leftarrow t + 1$
12: end while
```

2.3.3 训练行为函数

如第2.2节所述，B可以在任何过去episode的大规模输入-目标示例上进行监督训练。训练的目标是使行为函数产生与回放缓冲区中所有先前记录的轨迹一致的输出，根据方程1。

为了从回放缓冲区中随机选择一个训练示例，随机选择了时间步长索引 t_1 和 t_2，使得 $0 \le t_1 < t_2 \le T$，其中 T 是所选episode的长度。然后用于训练的输入 B 是 $(s_{t_1}, (d^r, d^h))$，其中 $d^r = \sum_{t=t_1}^{t_2} r_t$ 和 $d^h = t_2 - t_1$，目标是 a_{t_1}，即在 t_1 采取的动作。总结来说，训练示例是通过选择过去的时序、动作、观察和奖励生成输入-目标对。

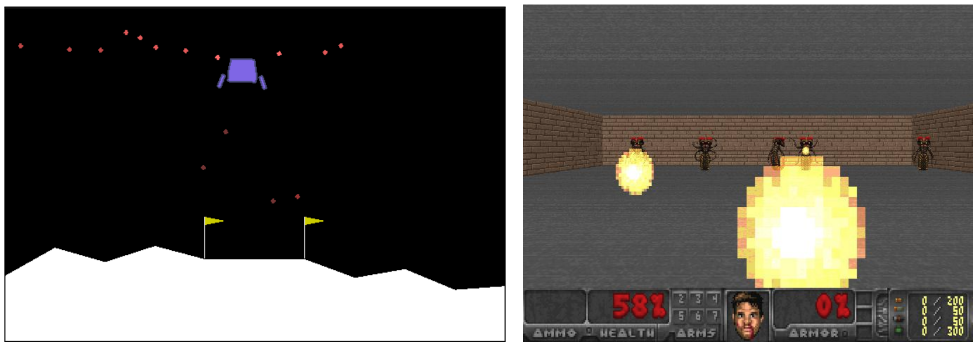
几种启发式方法可以用于选择和组合训练示例以形成用于梯度基学习策略的mini-batch。在本文的所有实验中，仅从每个episode中采样“尾随段落”，即我们设$t_2 = T - 1$，其中T是任何episode的长度。这舍弃了大量的潜在训练示例，但对于目标是优化每个episode结束前总奖励的 episodic 任务来说是一个很好的匹配。这也使得训练更加容易，因为行为函数只需要学会执行可能命令的子集。为了保持设置的简单性，在每次训练步骤中，所有实验都使用 Adam [13] 进行了固定次数的训练迭代。

2.3.4 抽样探索命令

在每个训练阶段之后，智能体可以尝试生成新的、之前不可行的行为，有可能获得更高的回报。为了通过泛化从中获利，首先必须创建一个新的初始命令集 c_0 用于算法 2。我们使用以下步骤来采样命令：

1. 从重播缓冲区的末尾选择了一些回放片段（即，具有最高回报的片段）。这个数量是一个超参数，在训练过程中保持固定。2. 探索性期望时滞 d_0^h 设置为所选片段长度的均值。3. 探索性期望回报 d_0^r 从均匀分布 $\mathcal{U}[M, M + S]$ 中采样，其中 M 是所选片段回报的均值，S 是所选片段回报的标准差。

该过程被选择是因为它的简单性以及能够通过单个超参数调整策略的能力。直观上，它尝试通过环境中的随机性生成新的行为，以在回放中实现最佳已知行为边缘的回报。虽然 Schmidhuber [34] 指出，这里可以使用各种启发式方法，



(a) LunarLander-v2 (b) TakeCover-v0

Figure 3: Test environments. In LunarLander-v2, the agent does not observe the visual representation, but an 8-dimensional state vector instead. In TakeCover-v0, the agent observes a down-sampled gray-scale visual inputs.

in practice it is very important to select exploratory commands that lead to behavior that is meaningfully different from existing experience so that it drives learning progress. An inappropriate exploration strategy can lead to very slow or stalled learning.

2.3.5 Generating Experience

Once the exploratory commands are sampled, it is straightforward to generate new exploratory episodes of interaction by using Algorithm 2, which works by repeatedly sampling from the action distribution predicted by the behavior function and updating its inputs for the next step. A fixed number of episodes are generated in each iteration of learning, and added to the replay buffer.

2.3.6 Evaluation

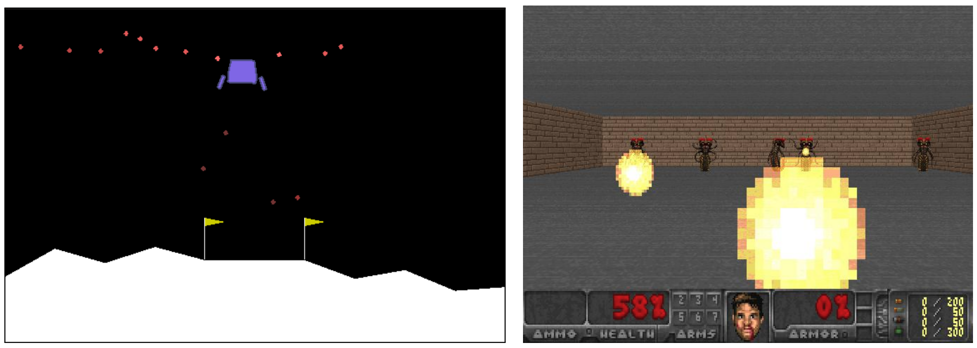
Algorithm 2 is also used to evaluate the agent at any time using evaluation commands derived from the most recent exploratory commands. The initial desired return d_0^r is set to the lower bound of the desired returns from the most recent exploratory command, and the initial desired horizon d_0^h from the most recent exploratory command is reused. In certain conditions, *greedy* actions – using the mode of the action distribution – can also be used, but we omit this option here for simplicity.

3 Experiments

The goal of our experiments was to determine the practical feasibility of ⅂ꓤ and put its performance in context of two well-known traditional RL algorithms: Deep Q-Networks (DQN; 20) and Advantage Actor-Critic (A2C; synchronous version of the algorithm proposed by Mnih et al. [21]).

3.1 Environments

LunarLander-v2 (Figure 3a) is a simple Markovian environment available in the Gym RL library [4] where the objective is to land a spacecraft on a landing pad by controlling its main and side engines. During the episode the agent receives negative reward at each time step that decreases in magnitude the closer it gets to the optimal landing



(a) LunarLander-v2 (b) TakeCover-v0

图3：测试环境。在LunarLander-v2中，代理不观察视觉表示，而是观察一个8维状态向量。在TakeCover-v0中，代理观察下采样的灰度视觉输入。

在实践中，选择能够导致与现有经验有意义不同的行为的探索命令是非常重要的，这样可以推动学习进步。不恰当的探索策略可能导致学习非常缓慢甚至停滞。

2.3.5 生成经验

一旦探索性命令被采样，通过使用算法2生成新的探索性交互 episode 是直截了当的，该算法通过反复从行为函数预测的动作分布中采样，并更新其输入以进行下一步来工作。在学习的每一迭代中生成固定数量的 episode，并将其添加到重放缓冲区中。

2.3.6 评估

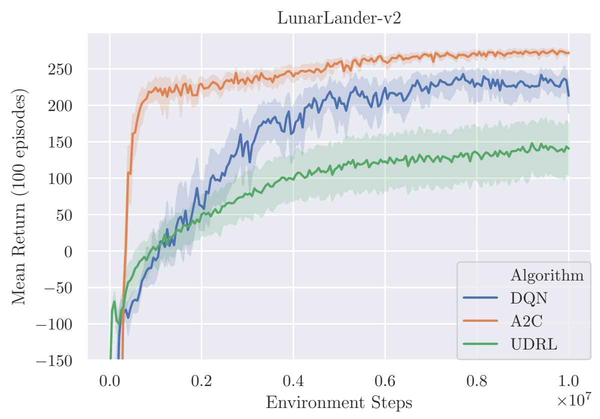
算法 2 也被用于在任何时间点评估代理，使用从最近的探索命令派生的评估命令。初始期望回报 d_0^r 被设置为最近探索命令的期望回报的下界，而初始期望时间 horizon d_0^h 则重用来自最近探索命令的值。在某些条件下，贪婪动作——使用动作分布的模式——也可以被使用，但为了简化，我们在这里省略了这一选项。

3 实验

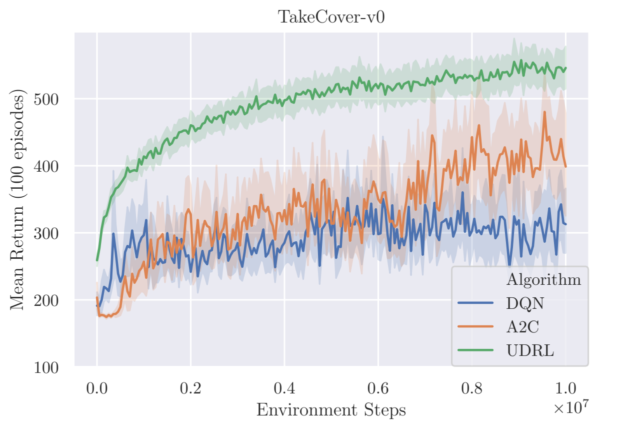
我们的实验目标是确定RL的实际可行性，并将其性能放在两种著名的传统RL算法——深度Q网络（DQN；20）和优势actor-critic（A2C；Mnih等人提出的算法的同步版本，[21]）的背景下进行比较。

3.1 环境

LunarLander-v2 (图3a) 是一个简单的马尔可夫环境，可在 Gym RL 库 [4] 中找到，目标是通过控制主引擎和侧引擎在着陆平台上着陆。在每个时间步骤中，当代理越接近最优着陆点时，其收到的负奖励的幅度会减小。



(a) On LunarLander-v2, ꓕꓤ is able to train agents that land the spacecraft, but is beaten by traditional RL algorithms.



(b) On TakeCover-v0, ꓕꓤ is able to consistently yield high-performing agents, while outperforming DQN and A2C.

Figure 4: Evaluation results for LunarLander-v2 and TakeCover-v0. Solid lines represent the mean of evaluation scores over 20 runs using tuned hyperparameters and experiment seeds 1–20. Shaded regions represent 95% confidence intervals using 1000 bootstrap samples. Each evaluation score is a mean of 100 episode returns.

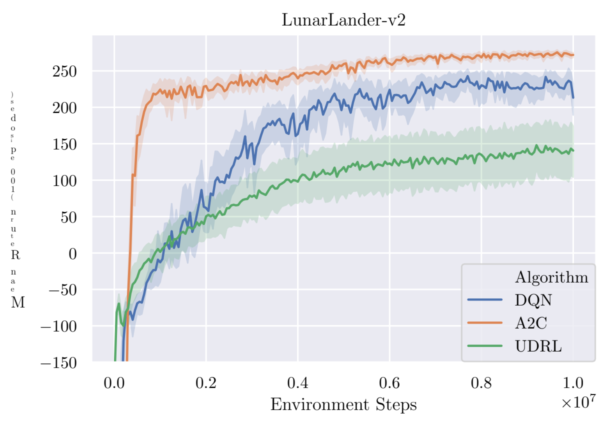
position in terms of both location and orientation. The reward at the end of the episode is -100 for crashing and +100 for successful landing. The agent receives eight-dimensional observations and can take one out of four actions.

TakeCover-v0 (Figure 3b) environment is part of the VizDoom library for visual RL research [12]. The agent is spawned next to the center of a wall in a rectangular room, facing the opposite wall where monsters randomly appear and shoot fireballs at the agent. It must learn to avoid fireballs by moving left or right to survive as long as possible. The reward is +1 for every time step that the agent survives, so for ꓕꓤ agents we always set the desired horizon to be the same as the desired reward, and convert any fractional values to integers. Technically, the agent has a non-Markovian interface to the environment, since it cannot see the entire opposite wall at all times. To reduce the degree of partial observability, the eight most recent visual frames are stacked together to produce the agent observations. The frames are also converted to gray scale, and downsampled from an original resolution of 160$\times$120 to 32$\times$32.

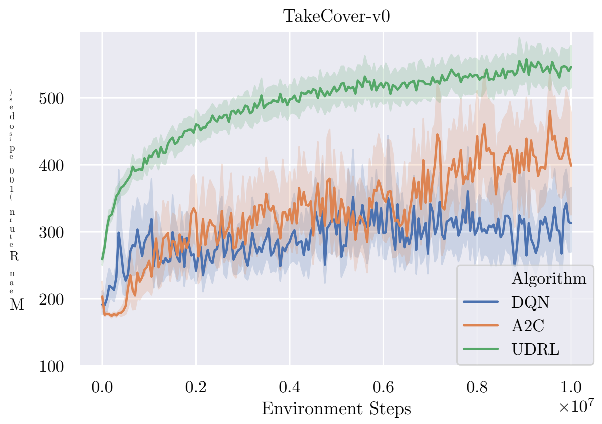
3.2 Setup

All agents were implemented using artificial neural networks. The behavior function for UDRL agents was implemented using fully-connected feed-forward networks for LunarLander-v2, and convolutional neural networks (CNNs; 16) for TakeCover-v0. The command inputs were scaled by a fixed scaling factor, transformed by a fully-connected sigmoidal layer, and then multiplied element-wise with an embedding of the observed inputs (after the first layer for fully-connected networks; after all convolutional layers for CNNs). Apart from this small modification regarding UDRL command inputs, the network architectures were identical for all algorithms.

All experiments were run for 10M environmental steps, with the agent being evaluated for 100 episodes at 50K step intervals. For each environment, random sampling was first used to find good hyperparameters for each algorithm and model based on final performance. With this configuration, final experiments were executed with 20 seeds (from 1 to 20) for each environment and algorithm. Random seeds for resetting the environments were sampled from [1M, 10M) for training, [0.5M, 1M) for evaluation during hyperparameter tuning, and [1, 0.5M) for final evaluation with the best hyperparameters. Details of the hyperparameter tuning procedure are provided in Appendix B.



(a) 在LunarLander-v2中，RL能够训练出能够着陆航天器的代理，但会被传统的RL算法击败。



(b) 在 TakeCover-v0 中，RL 能够持续生成高性能的智能体，同时在性能上超越 DQN 和 A2C。

图4：LunarLander-v2和TakeCover-v0的评估结果。实线表示使用调优后的超参数和实验种子1–20进行的20次运行的评估得分均值。阴影区域表示使用1000个重抽样样本的95%置信区间。每个评估得分是100个episode回报的均值。

位置既包括位置也包括姿态。在episode结束时，撞毁的奖励是-100，成功着陆的奖励是+100。智能体接收到八维的观测，并可以采取四种动作之一。

TakeCover-v0（图3b）环境是VizDoom库中用于视觉强化学习研究的一部分[12]。智能体在矩形房间的中心墙壁旁边生成，并面向对面的墙壁，而怪物会在对面的墙壁上随机出现并对智能体发射火球。智能体必须学会通过左右移动来躲避火球，以尽可能长时间地生存。奖励为每一步智能体生存时获得+1，因此对于RL智能体，我们总是将期望的时域设置为与期望的奖励相同，并将任何小数值转换为整数。从技术上讲，智能体与环境之间的接口是非马尔可夫的，因为智能体不能在所有时间都看到对面的整个墙壁。为了减少部分可观测性的程度，最近的八帧视觉图像被堆叠在一起以生成智能体的观察结果。这些帧也被转换为灰度，并从原始分辨率160×120下采样到32×32。

3.2 配置

所有代理都是使用人工神经网络实现的。UDRL代理的行为函数在LunarLander-v2中使用全连接前馈网络实现，而在TakeCover-v0中使用卷积神经网络（CNNs；16）实现。命令输入通过一个固定的缩放因子进行缩放，然后通过一个全连接的sigmoid层进行转换，并与观察到的输入的嵌入（对于全连接网络，在第一层之后；对于CNN，在所有卷积层之后）进行元素-wise相乘。除了对UDRL命令输入的这一小处修改外，所有算法的网络架构都是相同的。

所有实验都运行了10M环境步长，代理在每50K步长间隔进行100个episode的评估。对于每个环境，首先使用随机采样来找到每个算法和模型的最佳超参数，基于最终性能。在这种配置下，最终实验在每个环境和算法中使用20个种子（从1到20）执行。用于重置环境的随机种子从训练时的[1M, 10M)中采样，调优超参数时的评估从[0.5M, 1M)中采样，使用最佳超参数的最终评估从[1, 0.5M)中采样。超参数调优过程的详细信息参见附录B。

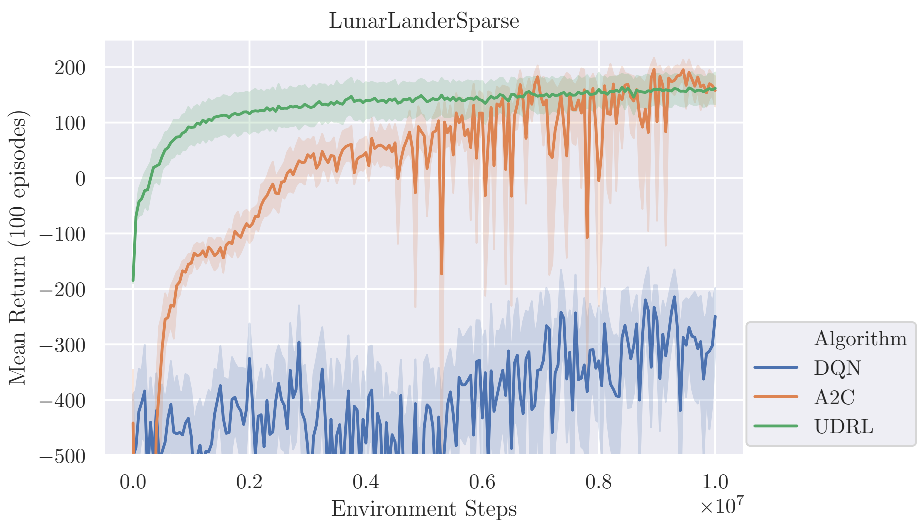


Figure 5: Results for LunarLanderSparse, a sparse reward version of LunarLander-v2 where the cumulative reward is delayed until the end of each episode. ⅂ꓤ learns both much faster and more consistently than both DQN and A2C. Plot semantics are the same as Figure 4.

3.3 Results

The results of the final 20 runs are plotted in Figure 4, with dark lines showing the mean evaluation return and shaded regions indicating 95% confidence intervals with 1000 bootstrap samples.

For LunarLander-v2, a return of 100–140 indicates successful landing and returns above 200 are reached by close-to-optimal policies. ⅂ꓤ lags behind DQN and A2C on this task. Inspection of the individual agents (for different seeds) showed that it is able to consistently train agents that land successfully, but while some agents learn quickly and achieve returns similar to A2C/DQN, some others plateau at lower returns.[3] We conjecture that this environment is rather suitable for TD learning by design due to its dense reward structure and large reward signals at the end.

For TakeCover-v0, the maximum possible return is 2100 due to episodic time limits. However, due to the difficulty of the environment (the number of monsters increases with time) and partial observability, the task is considered solved if the average reward over 100 episodes is greater than 750. On this task, ⅂ꓤ comfortably outperforms both DQN and A2C, demonstrating its applicability to high-dimensional control problems.

3.4 Importance of Reward Structure: Sparse Lunar Lander

Is the better performance of DQN and A2C on LunarLander-v2 primarily because its reward function is more suitable for traditional algorithms? To answer this question, the setup was modified to accumulate all rewards until the end of each episode and provide them to the agent only at the last time step. The reward at all other time steps was zero, so that the total episode return remained the same. The evaluation study was repeated, including a hyperparameter search for all three algorithms, for the new *LunarLanderSparse* environment. The results for the final 20 runs are plotted in Figure 5.

⅂ꓤ agents (green) learned faster and more reliably with this reward function and outperformed DQN and A2C, both of which suffered from severe difficulties in dealing with delayed and sparse rewards. A2C could achieve high rewards on this task due to a large hyperparameter search, but its performance was both very sensitive to hyperparameter settings

[3]This highlights the importance of evaluating with a sufficiently large number of random seeds, without which ⅂ꓤ can appear on par with DQN.

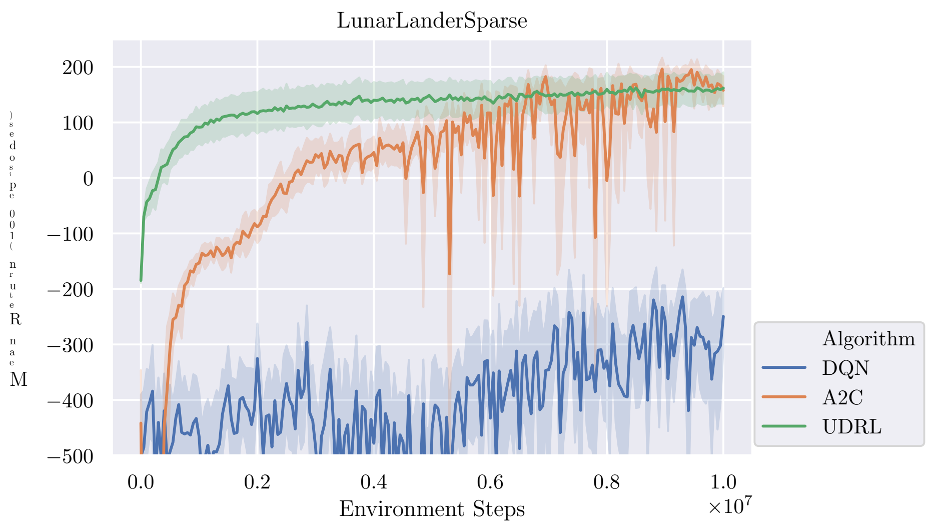


图 5：LunarLanderSparse 的结果，这是延迟到每个episode结束才累积奖励的 LunarLander-v2 的稀疏奖励版本。RL 不仅学习得更快，而且更一致，比 DQN 和 A2C 都要好。图的含义与图 4 相同。

3.3 结果

最终20次运行的结果绘制在图4中，实线表示平均评估回报，阴影区域表示使用1000个自助样本的95%置信区间。

对于LunarLander-v2，返回值在100–140之间表示成功着陆，而超过200的返回值则由接近最优的策略达到。在这一任务上，RL在性能上落后于DQN和A2C。对不同种子的个体代理进行检查后发现，它能够一致地训练出能够成功着陆的代理，但是一些代理能够快速学习并达到与A2C/DQN相似的返回值，而另一些则在较低的返回值处停滞不前。[3] 我们推测，由于其密集的奖励结构和最终的大奖励信号，这种环境从设计上更适于TD学习。

对于TakeCover-v0，由于 episodic 时间限制，最大可能回报是 2100。然而，由于环境的难度（随时间增加的怪物数量）和部分可观测性，如果平均回报在 100 个episode上的平均值大于 750，该任务就被认为是解决了。在这个任务上，RL 舒心地超越了 DQN 和 A2C，展示了其在高维控制问题上的适用性。

3.4 奖励结构的重要性：稀疏月球着陆器

是由于其奖励函数更适合传统算法，从而使DQN和A2C在LunarLander-v2上的表现更好吗？为了回答这个问题，将设置修改为在每个episode结束时累积所有奖励，并仅在最后一个时间步将这些奖励提供给代理。其他所有时间步的奖励均为零，以保持整个episode的回报不变。然后对新的LunarLanderSparse环境进行了评估研究，包括对所有三种算法进行超参数搜索。最终20次运行的结果如图5中所示。

RL智能体（绿色）使用这个奖励函数学习得更快更可靠，并且在处理延迟和稀疏奖励方面优于DQN和A2C，后者在处理延迟和稀疏奖励时遇到了严重的困难。尽管A2C通过大量的超参数搜索能够在该任务上达到高奖励，但其性能对超参数设置非常敏感。

[3]This highlights the importance of evaluating with a sufficiently large number of random seeds, without which $_{LR}$ can appear on par with DQN.

(compared to ⅂ꓤ) and rather unstable during training. Overall, we find that ⅂ꓤ is capable of training agents with both sparse and dense rewards (LunarLanderSparse and TakeCover-v0), but in some environments sparse rewards may work better than dense rewards. This counterintuitive property is an important avenue for future research.

The results above also lead us to a broader observation about RL benchmarks. The "actual" task in the LunarLander-v2 and LunarLanderSparse environments is exactly the same. Even the total episode return is the same for the same sequence of actions in the two environments by construction. Yet we obtain very different results from algorithms based on different principles simply based on the choice of the reward function. However, reward functions for benchmark problems are often developed side-by-side with existing algorithms and this can inadvertently favor certain learning paradigms over others. A potential way out of this bias is to evaluate each algorithm using a variety of reward functions for the same underlying RL task in order to understand its applicability to new RL problems.

3.5 Sensitivity of Trained Agents to Desired Returns

The ⅂ꓤ objective trains the agent to achieve all known returns over known horizons, but the complete learning algorithm used in our experiments is designed to achieve higher returns as training progresses. This is done by keeping the highest return episodes in the replay buffer, and also biasing exploration towards higher returns. Nevertheless, at the end of training, the agents were found to able to exhibit large changes in behavior based on the level of initial desired episode return (d^r_0).

We evaluated agents at the end of training on all three environments by setting various values of d^r_0 and plotting the obtained mean episode return over 100 episodes. The results are shown in Figure 6. Figures 6a and 6b show a strong correlation between obtained and desired returns for randomly selected agents on LunarLander-v2 and LunarLanderSparse. Note that in the later stages of training, the agents are only trained on episodes with returns close to the maximum.

Not all agents achieve such strong correspondence between desired and obtained returns. Figure 6c shows another agent that solves the task for most values of desired returns, and only achieves lower returns for very low values of desired returns. This indicates that stochasticity during training can affect how trained agents generalize to different commands, and suggests another direction for future investigation.

In the TakeCover-v0 environment, it is rather difficult to achieve precise values of desired returns. Stochasticity in the environment (the monsters appear randomly and shoot in random directions) and increasing difficulty over the episode imply that it is not possible to achieve lower returns than 200 and it gets considerably hard to achieve higher mean returns. The results in Figure 6d reflect these constraints, but still show that the agent is sensitive to the command inputs to some extent.

4 Related Work

Improving RL through SL has a long and rich history beyond the scope of this paper [33, 37]. For example, RL systems based on experience replay [18] attempt to leverage SL on past (off-policy) experience to improve value function approximation. Off-policy training can be augmented with goal-conditional value functions [11] (see also Pong et al. [27], Schaul et al. [31]) such that value functions for goals not being pursued by the current policy can also be updated based on the same interaction history. This idea was recently combined with experience replay by Andrychowicz et al. [1] and extended to policy gradients by Rauber et al. [28]. Oh et al. [23] proposed learning to imitate past good actions to augment actor-critic algorithms. Despite some differences, all of the above algorithms still rely on reward prediction using learned value functions, whereas ⅂ꓤ uses neither.

There is also substantial prior work on learning reward or goal-conditional policies that directly produce actions [5, 6, 15, 35], but these rely either on a pre-trained world model, a dataset of goal and optimal policy parameters, or policy gradients for learning. While the behavior function in ⅂ꓤ does bear a high-level similarity to goal-conditioned

(与RL相比)并且在训练过程中较为不稳定。总体而言，我们发现RL能够训练出在稀疏奖励和密集奖励环境（LunarLanderSparse和TakeCover-v0）下的代理，但在某些环境中，稀疏奖励可能比密集奖励效果更好。这一反直觉的特性是未来研究的重要方向。

上述结果还使我们对 RL 基准测试有一个更广泛的观察。在 LunarLander-v2 和 LunarLanderSparse 环境中，"实际"任务完全相同。即使在两个环境中，对于相同的动作序列，总集回溯也是一样的。然而，基于不同原则的算法仅仅根据奖励函数的选择就得到了非常不同的结果。但是，基准问题的奖励函数往往是在现有算法的同时开发的，这可能会无意中倾向于某些学习范式而忽视其他范式。克服这种偏见的一种潜在方法是使用多种奖励函数来评估每个算法在相同的底层 RL 任务上的适用性，以便理解其对新 RL 问题的应用。

3.5 训练代理对期望回报的灵敏度

RL目标训练代理实现已知范围内的所有已知回报，但在我们的实验中使用的完整学习算法设计目的是随着训练的进行实现更高的回报。这是通过在重播缓冲区中保留回报最高的episode，并且还偏向于探索更高回报来实现的。然而，在训练结束时，发现代理能够根据初始期望episode回报水平(d_0^r)表现出很大的行为变化。

我们在训练结束时在所有三个环境中评估了智能体，通过设置 d_0^r 的不同值并绘制在 100 个episode中获得的平均回合回报，展示了评估结果，如图6所示。图6a和6b显示了在LunarLander-v2和LunarLanderSparse中随机选择的智能体的实际回报与期望回报之间存在很强的相关性。请注意，在训练的后期阶段，智能体仅在接近最大回报的episode中进行训练。

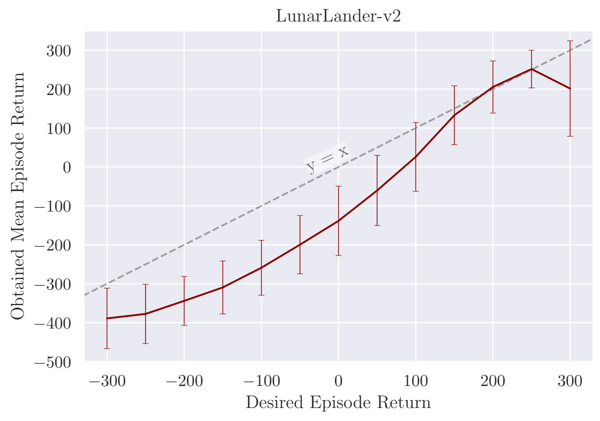
并非所有代理都能在期望回报和获得回报之间实现如此强烈的对应关系。图6c显示另一个代理在大多数期望回报值下都能完成任务，只有在期望回报值非常低时才能获得较低的回报。这表明训练过程中的随机性可能会影响训练后代理对不同命令的泛化能力，并暗示了未来研究的另一个方向。

在TakeCover-v0环境中，实现期望回报的精确值相当困难。环境中的随机性（怪物随机出现并随机射击）和每回合难度的增加意味着无法实现低于200的回报，并且实现更高的平均回报变得相当困难。图6d中的结果反映了这些限制，但仍显示代理在一定程度上对命令输入敏感。

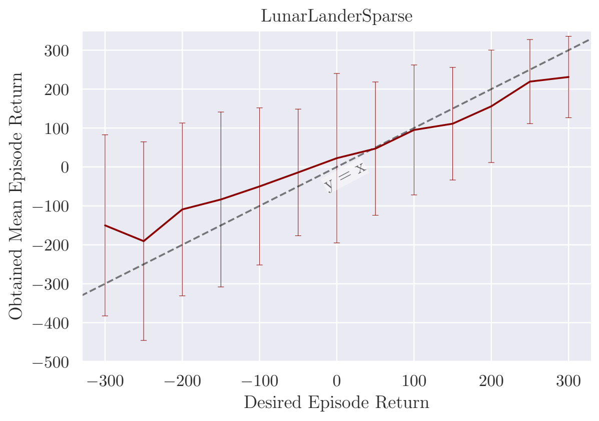
4 相关工作

通过SL改进RL有着悠久而丰富的历史，超出了本文的范围 [33, 37]。例如，基于经验回放的RL系统 [18] 试图利用SL来利用过去的（离策略）经验以改进价值函数逼近。离策略训练可以通过目标条件价值函数 [11]（参见Pong等人 [27] 和Schaul等人 [31]）来增强，使得当前策略未追求的目标的价值函数也可以基于相同的交互历史进行更新。Andrychowicz等人 [1] 最近将这一想法与经验回放结合，并由Rauber等人 [28] 扩展到了策略梯度中。Oh等人 [23] 提出了学习模仿过去的好行为来增强演员-评论家算法。尽管存在一些差异，上述所有算法仍然依赖于使用学习到的价值函数来进行奖励预测，而RL则不使用。

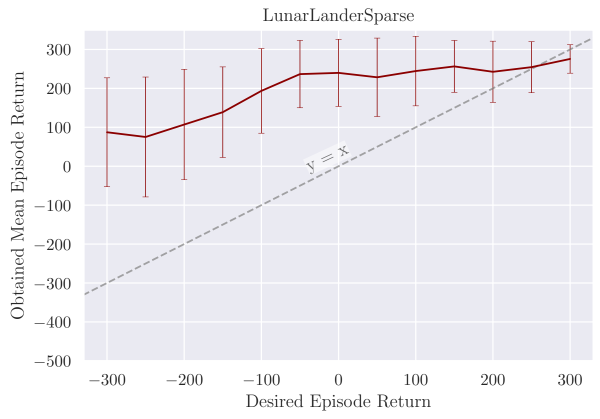
也有大量的前期工作专注于学习奖励或目标条件策略，这些策略可以直接产生动作 [5, 6, 15, 35]，但这些方法要么依赖于预训练的世界模型，要么依赖于目标和最优策略参数的数据集，要么依赖于策略梯度来学习。虽然在RL中的行为函数与目标条件的行为在高层次上具有相似性，但



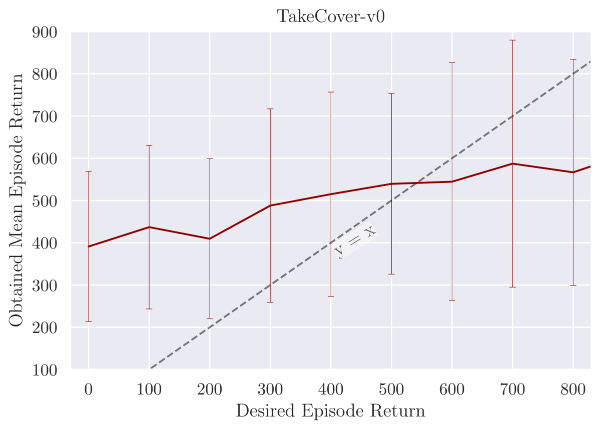
(a)



(b)



(c)



(d)

Figure 6: Obtained vs. desired episode returns for UDRL agents at the end of training. Each evaluation consists of 100 episodes. Error bars indicate standard deviation from the mean.

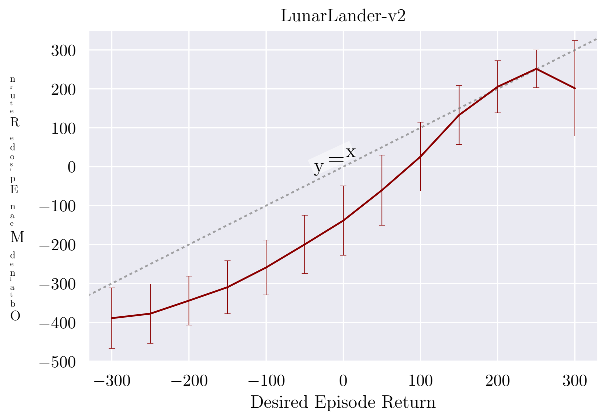
policies, the differences motivating its name are: (a) it takes time-varying desired returns and time horizons as inputs (Algorithm 2), as opposed to fixed goal states, (b) it does not predict rewards at all, (c) it is trained using SL (not policy gradients) on all past behavior, eliminating distinctions between on-policy and off-policy training.

PowerPlay [32, 36] uses extra task inputs to directly produce actions and is also continually trained to make sure it does not forget previously learned skills. However, its task inputs are not selected systematically based on previously achieved tasks/rewards.

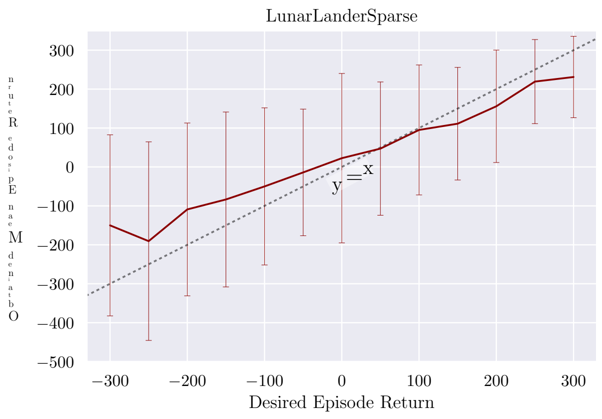
A control approach proposed by Dosovitskiy and Koltun [7] uses SL to predict future values of measurements (possibly rewards) given actions, which also sidesteps traditional RL algorithms. A characteristic property of ⅂ꓤ, however, is its very simple shortcut: it learns the mapping from rewards to actions directly from (possibly accidental) experience, and does not predict rewards at all.

5 Discussion

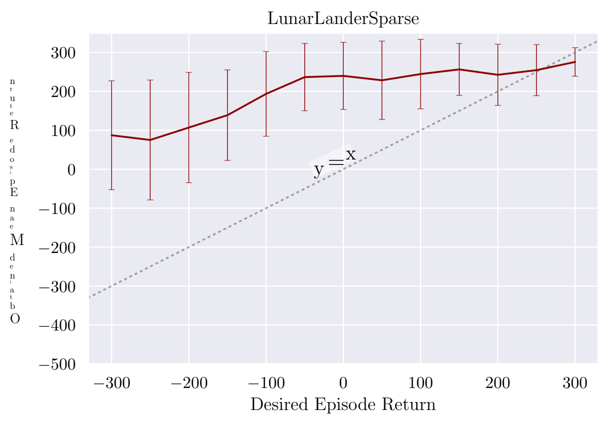
We introduced basic ideas of ⅂ꓤ and showed that it can solve certain challenging RL problems. Since RL benchmarks often tend to get developed alongside RL algorithms, a departure from traditional paradigms may motivate new



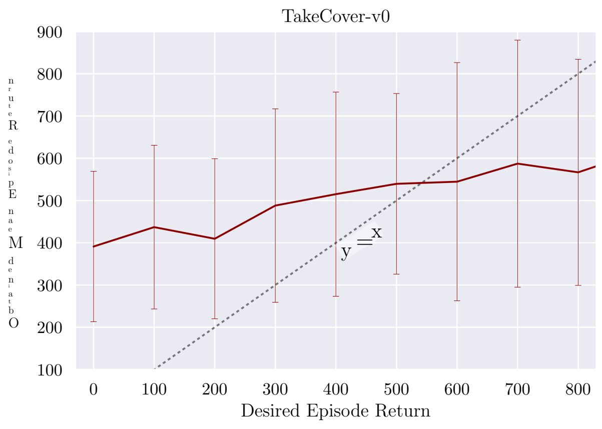
(a)



(b)



(c)



(d)

图6：UDRL智能体训练结束时获得的回报与期望的回报之间的对比。每次评估包括100个episode。误差棒表示均值的标准偏差。

政策的不同之处，使其名称得以体现：(a) 它以时间变化的目标回报和时间范围作为输入（算法2），而不是固定的目标状态，(b) 它根本不预测奖励，(c) 它是使用SL在所有过去行为上进行训练（而不是使用策略梯度），从而消除了在线策略和离线策略训练之间的区别。

PowerPlay [32, 36] 使用额外的任务输入直接生成动作，并且持续训练以确保它不会忘记之前学到的技能。然而，它的任务输入并不是根据之前完成的任务/奖励系统性地选择的。

Dosovitskiy 和 Koltun [7] 提出的一种控制方法使用 SL 预测给定动作时未来测量值（可能是奖励）的变化，从而绕过了传统的 RL 算法。然而，RL 的一个特征性质是它有一个非常简单的捷径：它直接从（可能是偶然的）经验中学习奖励到动作的映射，并根本不预测奖励。

5 讨论

我们介绍了RL的基本思想，并展示了它能够解决某些具有挑战性的RL问题。由于RL基准往往与RL算法共同发展，偏离传统范式可能会激励新的

problem domains that fit ⅂ᴚ better than traditional RL.

Many TD-based RL algorithms use discount factors that distort true environmental returns. TD learning is also very sensitive to the frequency of taking actions, which can limit its applicability to robot control [26]. In contrast, ⅂ᴚ explicitly takes into account observed rewards and time horizons in a precise and natural way, does not assume infinite horizons, and does not suffer from distortions of the basic RL problem. Note that other algorithms such as evolutionary RL [22] can also avoid these issues in other ways.

Well-known issues arise when off-policy TD learning is combined with high-dimensional function approximation. These issues — referred to by Sutton and Barto [37] as the *deadly triad* — can severely destabilize learning and are usually addressed by adding a variety of ingredients, though complete remedies remain elusive [38].

⅂ᴚ, on the other hand, works fundamentally in concert with high-capacity function approximators (since tabular behavior functions can not generalize), does not require learning from non-stationary targets and does not distinguish between on-policy and off-policy training. Instead, it brings fundamental questions related to catastrophic forgetting [19], continual learning [29], and generalization from past experience to the forefront.

6 Future Research Directions

There are several directions along which the ideas presented in this paper may be extended. On the agent architecture side, using recurrent instead of feedforward neural networks as behavior functions will be necessary for general partially observable environments, and useful for fully observable but noisy environments. New formats of command inputs and/or architectural modifications tailored to them are likely to substantially improve the inductive bias of ⅂ᴚ agents. In general, a wide variety of well-known SL techniques for model design, regularization and training can be employed to improve ⅂ᴚ's learning stability and efficiency.

Many aspects of the training algorithm were kept deliberately simple for this initial study. Future work should utilize other semantics for command inputs such as "reach a given goal state in at most T time steps", and strategies for sampling history segments other than just trailing segments. Similarly, it is probably unnecessary to generate a constant number of exploratory episodes per iteration, which decreases sample efficiency. We also expect that hyperparameters such as the number of optimization updates per iteration can be automatically adjusted during training.

Our current version of ⅂ᴚ utilizes a very simple form of exploration enabled by its design: it simply attempts to achieve high returns by generalizing from known states and returns. This was sufficient to drive learning in the tested environments with small number of available actions and high stochasticity that helps discover new behaviors to learn from. In other environments, additional forms of undirected (random) and directed exploration are likely to be fruitful or even necessary.

Finally, there is a vast open space of possible combinations of ⅂ᴚ and algorithms based on learning environmental models, Temporal Differences, optimal control and policy search. Such combinations may lead to more general learning agents that are useful in a variety of environments.

Contributions

All authors contributed to discussions. JS proposed the basic principles of ⅂ᴚ [34]. WJ developed the code setup for the baselines. PS and FM developed early implementations and conducted initial experiments on fully deterministic environments. RKS developed the final algorithm, supervised PS and FM, conducted experiments and wrote the paper.

传统RL更适合于解决的问题领域。

许多TD基于的RL算法使用折扣因子来扭曲真实的环境回报。TD学习对采取行动的频率也非常敏感，这可以限制其在机器人控制中的适用性[26]。相比之下，RL明确地考虑了观察到的奖励和时间范围，以一种精确和自然的方式，不假设无限的时间范围，并且不会遭受基本RL问题的扭曲。请注意，其他算法如进化RL[22]也可以通过其他方式避免这些问题。

当离策学习与高维函数逼近结合时，会出现一些众所周知的问题。这些问题——如辛顿和巴托[37]所称的致命三角——可能会严重 destabilize 学习，并且通常通过添加各种成分来解决，尽管完全的解决方案仍然难以捉摸[38]。

另一方面，RL 基本上与高容量函数逼近器协同工作（因为表格式行为函数无法泛化），不需要从非平稳目标中学习，并且不区分有策略训练和无策略训练。相反，它将与灾难性遗忘[19]、持续学习[29]以及从过去经验中泛化相关的基本问题置于 forefront。

6 未来的研究方向

本文中提出的想法可以沿几个方向扩展。在代理架构方面，对于一般部分可观测环境，将需要使用递归而非前馈神经网络作为行为函数；对于完全可观测但噪声较大的环境，这将是很有用的。新的命令输入格式和/或针对它们的架构修改很可能会显著提高强化学习代理的归纳偏置。总体而言，可以广泛采用众所周知的监督学习（SL）技术来设计模型、正则化和训练，以提高强化学习的学习稳定性和效率。

许多训练算法的方面在初始研究中被故意保持简单。未来的工作应该利用其他语义作为命令输入，例如“在最多 T 步骤内达到给定的目标状态”，以及除了仅采样尾部段落之外的其他历史段落采样策略。同样，每迭代生成固定数量的探索性episode可能是不必要的，这会降低样本效率。我们还预计，如每迭代的优化更新次数这样的超参数可以在训练过程中自动调整。

我们当前的 RL 版本通过其设计启用了一种非常简单的探索形式：它仅仅通过从已知状态和回报中进行泛化来尝试获得高回报。这在可用动作数量较少且高随机性有助于发现新行为以学习的测试环境中足以驱动学习。在其他环境中，额外的无定向（随机）和有定向的探索形式可能是有益的，甚至可能是必要的。

最后，基于学习环境模型、时差方法、最优控制和策略搜索的 RL 和算法的组合有着巨大的可能性空间。这样的组合可能会导致更加通用的学习代理，这些代理在各种环境中都很有用。

贡献

所有作者都参与了讨论。JS 提出了RL的基本原则[34]。WJ 开发了基线的代码设置。PS 和 FM 开发了早期的实现并在完全确定性的环境中进行了初步实验。RKS 开发了最终的算法，监督了PS和FM，进行了实验并撰写了论文。

Acknowledgments

We thank Paulo E. Rauber for discussions during the early phase of this project. We are also grateful to Faustino Gomez for several suggestions that improved an earlier draft of this report, and Nihat Engin Toklu for discussions that led to the LunarLanderSparse environment.

References

[1] Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba
2017. Hindsight Experience Replay. *arXiv:1707.01495 [cs].*

[2] Barto, A. G. and T. G. Dietterich
2004. Reinforcement learning and its relationship to supervised learning. In *Handbook of Learning and Approximate Dynamic Programming*, P. 672.

[3] Bradski, G.
2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*

[4] Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba
2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540.*

[5] Da Silva, B., G. Konidaris, and A. Barto
2012. Learning parameterized skills. *arXiv preprint arXiv:1206.6398.*

[6] Deisenroth, M. P., P. Englert, J. Peters, and D. Fox
2014. Multi-task policy search for robotics. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Pp. 3876–3881. IEEE.

[7] Dosovitskiy, A. and V. Koltun
2016. Learning to Act by Predicting the Future. *arXiv:1611.01779 [cs].*

[8] Hakenes, S.
2018. Vizdoomgym. `https://github.com/shakenes/vizdoomgym`.

[9] Hill, A., A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu
2018. Stable Baselines. *GitHub repository.*

[10] Hunter, J. D.
2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[11] Kaelbling, L. P.
1993. Learning to Achieve Goals. In *IJCAI*, Pp. 1094–1099. Citeseer.

[12] Kempka, M., M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski
2016. Vizdoom: A doom-based AI research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Pp. 1–8. IEEE.

[13] Kingma, D. and J. Ba
2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[14] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, and Jürgen Schmidhuber
2017. The Sacred Infrastructure for Computational Research. In *Proceedings of the 16th Python in Science Conference*, Katy Huff, David Lippa, Dillon Niederhut, and M. Pacer, eds., Pp. 49 – 56.

致谢

我们感谢保罗·E拉伯在该项目早期阶段的讨论。我们也要感谢法斯图诺·戈麦斯提供的若干建议，这些建议改进了这份报告的早期草稿，以及尼哈特·恩吉·托卢的讨论，这些讨论导致了LunarLanderSparse环境的形成。

参考文献

[1] Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, 和 W. Zaremba 2017. 后见之明的经验回放。arXiv:1707.01495 [cs]

[2] Barto, A. G. 和 T. G. Dietterich 2004. 增强学习及其与监督学习的关系. 在《学习和近似动态规划手册》中，第672页。[3] Bradski, G. 2000. OpenCV库. Dr. Dobb’s Journal of Software Tools。[4] Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, 和 W. Zaremba 2016. OpenAI Gym. arXiv 预印本 arXiv:1606.01540。[5] Da Silva, B., G. Konidaris, 和 A. Barto 2012. 学习参数化的技能. arXiv 预印本 arXiv:1206.6398。

[6] Deisenroth, M. P., P. Englert, J. Peters, 和 D. Fox 2014. 机器人领域的多任务策略搜索. 在 2014 年 IEEE 国际机器人与自动化会议 (ICRA) 上，第 3876–3881 页. IEEE.

[7] Dosovitskiy, A. 和 V. Koltun 2016. 通过预测未来来学习行动。arXiv:1611.01779 [cs]。

[8] Hakenes, S. 2018. Vizdoomgym. `https://github.com/shakenes/vizdoomgym`.

[9] 希尔, A., A. 拉芬, M. 埃尔内斯图, A. 格莱夫, R. 特奥雷, P. 达哈瓦, C. 海塞, O. 克利莫夫, A. 尼科尔, M. 帕勒普特, A. 拉德福德, J. 施卢曼, S. 西多尔, 和 Y. 吴 2018. 稳定基线. GitHub 仓库。

[10] Hunter, J. D. 2007. Matplotlib: 一个二维图形环境. 计算机科学与工程, 9(3):90–95.

[11] Kaelbling, L. P. 1993. 学习实现目标. 在 IJCAI，第 1094–1099 页. Citeseer.

[12] Kempka, M., M. Wydmuch, G. Runc, J. Toczek, 和 W. Ja kowski 2016. Vizdoom：一种基于 doom 的视觉强化学习 AI 研究平台。In 2016 IEEE Conference on Computational Intelligence and Games (CIG)，Pp. 1–8。IEEE。

[13] Kingma, D.和J. Ba 2014. Adam: 一种随机优化方法. arXiv预印本 arXiv:1412.6980.

[14] Klaus Greff, Aaron Klein, Martin Chovanec, Frank Hutter, 和 Jürgen Schmidhuber 2017. 计算机科学研究的神圣基础设施。在第16届Python科学会议论文集，Katy Huff, David Lippa, Dillon Niederhut, 和 M. Pacer, 编，第49 – 56页。

[15] Kupcsik, A. G., M. P. Deisenroth, J. Peters, and G. Neumann
2013. Data-efficient generalization of robot skills with contextual policy search. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.

[16] LeCun, Y., B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel
1990. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, Pp. 396–404.

[17] Liaw, R., E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica
2018. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

[18] Lin, L.-J.
1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321.

[19] McCloskey, M. and N. J. Cohen
1989. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:109–164.

[20] Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu
2016. Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783 [cs]*.

[21] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al.
2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[22] Moriarty, D. E., A. C. Schultz, and J. J. Grefenstette
1999. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276.

[23] Oh, J., Y. Guo, S. Singh, and H. Lee
2018. Self-Imitation Learning. *arXiv:1806.05635 [cs, stat]*.

[24] Oliphant, T. E.
2015. *Guide to NumPy*, 2nd edition. USA: CreateSpace Independent Publishing Platform.

[25] Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer
2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[26] Plappert, M., M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba
2018. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research.

[27] Pong, V., S. Gu, M. Dalal, and S. Levine
2018. Temporal Difference Models: Model-Free Deep RL for Model-Based Control. *arXiv:1802.09081 [cs]*.

[28] Rauber, P., A. Ummadisingu, F. Mutz, and J. Schmidhuber
2017. Hindsight policy gradients. *arXiv:1711.06006 [cs]*.

[29] Ring, M. B.
1994. *Continual Learning in Reinforcement Environments*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712.

[30] Rosenstein, M. T. and A. G. Barto
2004. Supervised Actor-Critic Reinforcement Learning. In *Handbook of Learning and Approximate Dynamic Programming*, P. 672.

[31] Schaul, T., D. Horgan, K. Gregor, and D. Silver
2015. Universal Value Function Approximators. In *International Conference on Machine Learning*, Pp. 1312–1320.

[15] Kupcsik, A. G., M. P. Deisenroth, J. Peters, and G. Neumann 2013. 使用上下文策略搜索实现机器人技能的数据高效泛化。在第二十七届人工智能AAAI会议。[16] LeCun, Y., B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel 1990. 使用反向传播网络的手写数字识别。在神经信息处理系统进展，第396–404页。[17] Liaw, R., E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica 2018. Tune：一个分布式模型选择和训练的研究平台。arXiv 预印本 arXiv:1807.05118。[18] Lin, L.-J. 1992. 基于强化学习、规划和教学的自我改进反应式代理。机器学习，8(3-4):293–321。[19] McCloskey, M. 和 N. J. Cohen 1989. 连接主义网络中的灾难性干扰：序列学习问题。学习和动机的心理学，24:109–164。

[20] Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, 和 K. Kavukcuog
lu 2016. 异步深度强化学习方法。arXiv:1602.01783 [cs].

[21] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. 2015. 通过深度强化学习达到人类水平的控制。自然, 518(7540):529–533.

[22] Moriarty, D. E., A. C. Schultz, 和 J. J. Grefenstette 1999. 进化算法在强化学习中的应用。人工智能研究杂志，11:241–276。

[23] 周 Oh，郭宇，辛沙，和 李河 2018. 自我模仿学习
。arXiv:1806.05635 [cs, stat].

[24] Oliphant, T. E. 2015. NumPy 使用指南，第2版。美国: CreateSpace 独立出版平台。

[25] Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Ler
er 2017. PyTorch中的自动微分。在 NIPS-W.

[26] Plappert, M., M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Ch ociej,穿着衣服, P. Welinder, V. Kumar,和 W. Zaremba 2018. 多目标强化学习：具有挑战性的机器人环境和研究请求
。

[27] 蓝格, V., 顾思, M. 达拉尔, 和 斯提芬·莱文 2018. 时间差分模型：基于模型的控制的模型自由深度
强化学习。arXiv:1802.09081 [cs]。

[28] Rauber, P., A. Ummadisingu, F. Mutz, 和 J. Schmidhuber
2017. 回溯策略梯度。arXiv:1711.06006 [cs]。

[29] Ring, M. B. 1994. 强化环境中的持续学习. 博士学位论文, 德克萨斯大学奥斯汀分校计算机科学系, 德克萨斯州奥斯汀市, 邮政编码78712.

[30] 肯尼森, M. T. 和 A. G. 巴尔托 2004. 监督式行为-批评强化学习. 在《学习与近似动态规划手册》, 第 672 页.

[31] Schaul, T., D. Horgan, K. Gregor, 和 D. Silver 2015. 具有普遍性的价值函数逼近器。在 国际机器学习大会，第 1312-1320 页。

[32] Schmidhuber, J.
2013. PowerPlay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313.

[33] Schmidhuber, J.
2015. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[34] Schmidhuber, J.
2019. Reinforcement learning upside down: Don't predict rewards – just map them to actions. *NNAISENSE/IDSIA Technical Report.*

[35] Schmidhuber, J. and R. Huber
1991. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2(1 & 2):125–134.

[36] Srivastava, R. K., B. R. Steunebrink, and J. Schmidhuber
2013. First Experiments with PowerPlay. *Neural Networks*, 41:130–136.

[37] Sutton, R. S. and A. G. Barto
2018. *Reinforcement Learning: An Introduction.* MIT press.

[38] van Hasselt, H., Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil
2018. Deep Reinforcement Learning and the Deadly Triad. *arXiv:1812.02648 [cs].*

[39] Walt, S. v. d., S. C. Colbert, and G. Varoquaux
2011. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

[40] Waskom, M., O. Botvinnik, D. O'Kane, P. Hobson, J. Ostblom, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Brunner, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, and A. Qalieh
2018. mwaskom/seaborn: v0.9.0 (july 2018).

[41] Watkins, C. J. C. H.
1989. *Learning from Delayed Rewards.* PhD thesis, King's College, Oxford.

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

[32] Schmidhuber, J. 2013. PowerPlay: 通过不断搜索最简单的未解决难题来训练越来越通用的问题解决者。心理学前沿，4:313。[33] Schmidhuber, J. 2015. 神经网络中的深度学习：综述。神经网络，61:85−117。[34] Schmidhuber, J. 2019. 反转强化学习：不要预测奖励——直接将它们映射到动作。NNAISENSE/IDSIA技术报告。

[35] Schmidhuber, J. 和 R. Huber 1991. 学习生成人工焦斑轨迹以检测目标。国际神经网络系统杂志，2(1 & 2): 125−134。

[36] Srivastava, R. K., B. R. Steunebrink, 和 J. Schmidhuber 2013. First Experiments with PowerPlay. Neural Networks, 41:130−136. [37] Sutton, R. S. 和 A. G. Barto 2018. Reinforcement Learning: An Introduction. MIT press. [38] van Hasselt, H., Y. Doron, F. Strub, M. Hessel, N. Sonnerat, 和 J. Modayil 2018. Deep Reinforcement Learning and the Deadly Triad. arXiv:1812.02648 [cs]. [39] Walt, S. v. d., S. C. Colbert, 和 G. Varoquaux 2011. The numpy array: A structure for efficient numerical computation. Computing in Science & Engineering, 13(2):22−30. [40] Waskom, M., O. Botvinnik, D. O’Kane, P. Hobson, J. Ostblom, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Brunner, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, 和 A. Qalieh 2018. mwaskom/seaborn: v0.9.0 (july 2018). [41] Watkins, C. J. C. H. 1989. Learning from Delayed Rewards. 博士论文, 剑桥大学, 奥克兰. [42] Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229−256.

A Upside-Down RL Hyperparameters

Table 2 summarizes the name and role of all the hyperparameters for ˥ꓤ.

Table 2: A summary of UDRL hyperparameters

Name	Description
`batch_size`	Number of (input, target) pairs per batch used for training the behavior function
`horizon_scale`	Scaling factor for desired horizon input
`last_few`	Number of episodes from the end of the replay buffer used for sampling exploratory commands
`learning_rate`	Learning rate for the ADAM optimizer
`n_episodes_per_iter`	Number of exploratory episodes generated per step of UDRL training
`n_updates_per_iter`	Number of gradient-based updates of the behavior function per step of UDRL training
`n_warm_up_episodes`	Number of warm up episodes at the beginning of training
`replay_size`	Maximum size of the replay buffer (in episodes)
`return_scale`	Scaling factor for desired horizon input

B Hyperparameter Tuning

Hyperparameters were tuned by randomly sampling 256 configurations for LunarLander-v2 and LunarLanderSparse, and 72 configurations for TakeCover-v0. For LunarLander-v2, each random configuration of hyperparameters was evaluated with 3 random seeds and the results were averaged. For other tasks, each configuration was evaluated with a single seed.

The best hyperparameter configuration was selected based on the mean of evaluation scores for last 20 evaluations, yielding the configurations with the best average performance towards the end of training.

In the following subsections, we define the lists of values for each of the hyperparameters that were tuned for each environment and algorithm. For DQN and A2C, any other hyperparameters were left at their default values for Stable-Baselines, but we did enable additional tricks not found in the original papers such as multi-step bootstrapping or double Q-learning.

B.1 LunarLander-v2 & LunarLanderSparse

Architecture Hyperparameters

- Network architecture (indicating number of units per layer): [[32], [32, 32], [32, 64], [32, 64, 64], [32, 64, 64, 64], [64], [64, 64], [64, 128], [64, 128, 128], [64, 128, 128, 128]]

DQN Hyperparameters

- Activation function: $[\mathrm{tanh}, \mathrm{relu}]$
- Batch Size: [16, 32, 64, 128]
- Buffer Size: [10,000, 50,000, 100,000, 500,000, 1,000,000]
- Discount factor: [0.98, 0.99, 0.995, 0.999]
- Exploration Fraction: [0.1, 0.2, 0.4]

A upside-down RL 超参数

表2总结了所有RL超参数的名称和角色。

表 2: UDRL 超参数总结

Name	Description
`batch_size`	Number of (input, target) pairs per batch used for training the behavior function
`horizon_scale`	Scaling factor for desired horizon input
`last_few`	Number of episodes from the end of the replay buffer used for sampling exploratory commands
`learning_rate`	Learning rate for the ADAM optimizer
`n_episodes_per_iter`	Number of exploratory episodes generated per step of UDRL training
`n_updates_per_iter`	Number of gradient-based updates of the behavior function per step of UDRL training
`n_warm_up_episodes`	Number of warm up episodes at the beginning of training
`replay_size`	Maximum size of the replay buffer (in episodes)
`return_scale`	Scaling factor for desired horizon input

B 超参数调优

超参数通过随机采样分别对 LunarLander-v2 和 LunarLanderSparse 采样了 256 种配置，对 TakeCover-v0 采样了 72 种配置。对于 LunarLander-v2，每个随机的超参数配置使用 3 个随机种子进行评估并将结果平均。对于其他任务，每个配置使用单个种子进行评估。

最佳超参数配置是根据过去20次评估的评价得分的平均值选择的，这产生了在训练后期具有最佳平均性能的配置。

在以下子节中，我们定义了每个环境和算法中调优的每个超参数的值列表。对于DQN和A2C，任何其他超参数都保持了Stable-Baselines的默认值，但我们启用了原始论文中未找到的额外技巧，如多步-bootstrap或双Q学习。

B.1 LunarLander-v2 & LunarLanderSparse

架构超参数

- 网络架构（每层单元数量）: [[32], [32, 32], [32, 64], [32, 64, 64], [32, 64, 64, 64], [64], [64, 64], [64, 128], [64, 128, 128], [64, 128, 128, 128]]

DQN 超参数

- 激活函数: [tanh, relu]
- Batch Size: [16, 32, 64, 128]
- 缓冲区大小: [10000, 50000, 100000, 500000, 1000000]
- 折扣因子: [0.98, 0.99, 0.995, 0.999]
- 探索分数: [0.1, 0.2, 0.4]

- Exploration Final Eps: [0.0, 0.01, 0.05, 0.1]
- Learning rate: $\mathrm{numpy.logspace}(-4, -2, \mathrm{num} = 101)$
- Training Frequency: [1, 2, 4]
- Target network update frequency: [100, 500, 1000]

A2C Hyperparameters

- Activation function: $[\mathrm{tanh}, \mathrm{relu}]$
- Discount factor: [0.98, 0.99, 0.995, 0.999]
- Entropy coefficient: [0, 0.01, 0.02, 0.05, 0.1]
- Learning rate: $\mathrm{numpy.logspace}(-4, -2, \mathrm{num} = 101)$
- Value function loss coefficient: [0.1, 0.2, 0.5, 1.0]
- Decay parameter for RMSProp: [0.98, 0.99, 0.995]
- Number of steps per update: [1, 2, 5, 10, 20]

Upside-Down RL Hyperparameters

- `batch_size`: [512, 768, 1024, 1536, 2048]
- `horizon_scale`: [0.01, 0.015, 0.02, 0.025, 0.03]
- `last_few`: [25, 50, 75, 100]
- `learning_rate`: $\mathrm{numpy.logspace}(-4, -2, \mathrm{num} = 101)$
- `n_episodes_per_iter`: [10, 20, 30, 40]
- `n_updates_per_iter`: [100, 150, 200, 250, 300]
- `n_warm_up_episodes`: [10, 30, 50]
- `replay_size`: [300, 400, 500, 600, 700]
- `return_scale`: [0.01, 0.015, 0.02, 0.025, 0.03]

B.2 TakeCover-v0

Architecture Hyperparameters All networks had four convolutional layers, each with 3×3 filters, 1 pixel input padding in all directions and stride of 2 pixels.

The architecture of convolutional layers (indicating number of convolutional channels per layer) was sampled from [[32, 48, 96, 128], [32, 64, 128, 256], [48, 96, 192, 384]].

The architecture of fully connected layers following the convolutional layers was sampled from [[64, 128], [64, 128, 128], [128, 256], [128, 256, 256], [128, 128], [256, 256]].

Hyperpameter choices for DQN and A2C were the same as those for LunarLander-v2. For ˥ᴚ the following choices were different:

- `n_updates_per_iter`: [200, 300, 400, 500]
- `replay_size`: [200, 300, 400, 500]

- 探索最终ε值: [0.0, 0.01, 0.05, 0.1]
- 学习率: numpy. logspace$(-4, -2, \text{num} = 101)$
- 训练频率: [1, 2, 4]
- 目标网络更新频率: [100, 500, 1000]

A2C 超参数

- 激活函数: [tanh, relu]
- 折扣因子: [0.98, 0.99, 0.995, 0.999]
- 熵系数: [0, 0.01, 0.02, 0.05, 0.1]
- 学习率: numpy. logspace$(-4, -2, \text{num} = 101)$
- 值函数损失系数: [0.1, 0.2, 0.5, 1.0]
- RMSProp的衰减参数: [0.98, 0.99, 0.995]
- 每步更新的步数:[1, 2, 5, 10, 20]

倒置的RL超参数

- `batch_size`[512, 768, 1024, 1536, 2048]
- `horizon_scale`: [0.01, 0.015, 0.02, 0.025, 0.03]
- `last_few`[25, 50, 75, 100]
- `learning_rate`:numpy. logspace$(-4, -2, \text{num} = 101)$
- `n_episodes_per_iter`: [10, 20, 30, 40]
- `n_updates_per_iter`[100, 150, 200, 250, 300]
- `n_warm_up_episodes`[10, 30, 50]
- `replay_size`[300, 400, 500, 600, 700]
- `return_scale`: [0.01, 0.015, 0.02, 0.025, 0.03]

B.2 TakeCover-v0

架构超参数 所有网络都有四层卷积层，每层有3×3过滤器，在所有方向上有1像素输入填充，并且步长为2像素。

卷积层的架构（表示每层的卷积通道数）是从[[32, 48, 96, 128], [32, 64, 128, 256], [48, 96, 192, 384]]中采样的。

全连接层的架构在卷积层之后是从[[64, 128], [64, 128, 128], [128, 256], [128, 256, 256], [128, 128], [256, 256]]中采样的。

Hyper参数选择对于DQN和A2C与LunarLander-v2相同。对于RL，以下选择不同:

- `n_updates_per_iter`[200, 300, 400, 500]
- `replay_size`[200, 300, 400, 500]

- `return_scale`: [0.1, 0.15, 0.2, 0.25, 0.3]

C Software Implementation

Our setup directly relied upon the following open source software:

- Gym 0.11.0 [4]
- Matplotlib [10]
- Numpy 1.15.1 [24, 39]
- OpenCV [3]
- Pytorch 1.1.0 [25]
- Ray Tune 0.6.6 [17]
- Sacred [14]
- Seaborn [40]
- Stable-Baselines 2.5.0 [9]
- Vizdoom 1.1.6 [12]
- Vizdoomgym [8]

- `return_scale`: [0.1, 0.15, 0.2, 0.25, 0.3]

C 软件实现

我们的设置直接依赖以下开源软件:

- 健身房 0.11.0 [4]
- Matplotlib [10]
- Numpy 1.15.1 [24, 39]
- OpenCV [3]
- Pytorch 1.1.0 [25]
- Ray Tune 0.6.6 [17]
- 圣洁 [14]
- Seaborn [40]
- Stable-Baselines 2.5.0 [9]
- Vizdoom 1.1.6 [12]
- Vizdoomgym [8]