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.

培训代理使用颠倒的强化学习

Rupesh Kumar srivastava [1*] pranav shyam [1†] filipe mutz [2‡] wo
jciechja kowski[1]

1nnaisense [2]瑞士A
I实验室IDSIA
NNAISENSE Technical Report

抽象的

传统的加强学习（RL）算法要么通过价值功能预测奖励，要么使用策略搜索最大化它们。我们研究一种替代方法：颠倒的增强学习（倒置RL或RL），该学习主要使用监督学习技术解决RL问题。同伴报告中概述了其许多主要原则[34]。在这里，我们介绍了RL的第一个具体实施，并证明了其在某些情节学习问题上的可行性。实验结果表明，它的性能与数十年来研究的传统基线算法的竞争能力令人惊讶，甚至超过了传统的基线算法。

1简介

虽然有丰富的技术历史将监督学习（SL）纳入强化学习（RL）算法，但人们认为不可能使用SL解决RL问题，因为来自环境的反馈在SL中提供了错误信号，但是评估信号是评估信号在RL [2，30]中。简而言之，代理会收到有关其行动有多么有用的反馈，而不是在任何情况下最好采取哪些行动。关于将RL问题变成SL问题的可能性，Barto和Dietterich [2]推测：“总的来说，没有办法这样做。”

在同伴技术报告中，Schmidhuber [34]提议通过向上加固学习（RL）弥合SL和RL之间的这一差距，其中环境反馈（例如奖励）是一个输入，而不是传统RL中的学习目标基于奖励预测的算法[37]。在这里，我们为情节任务开发了一种实用的RL算法，并表明确实可以在不使用基于价值的算法（例如Q-LEARNING [41]）或基于策略的基于策略的算法的情况下训练代理[1]中的代理作为政策梯度和进化算法[22，42]。取而代之的是，RL使用纯SL来训练代理商在过去的所有经验上，并避开了功能近似，自举和非政策训练的组合引起的问题[37]。我们首先描述其基本原理，然后在实验上证明其在三个稀疏和密集奖励结构的RL问题上的可行性。

[*]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.

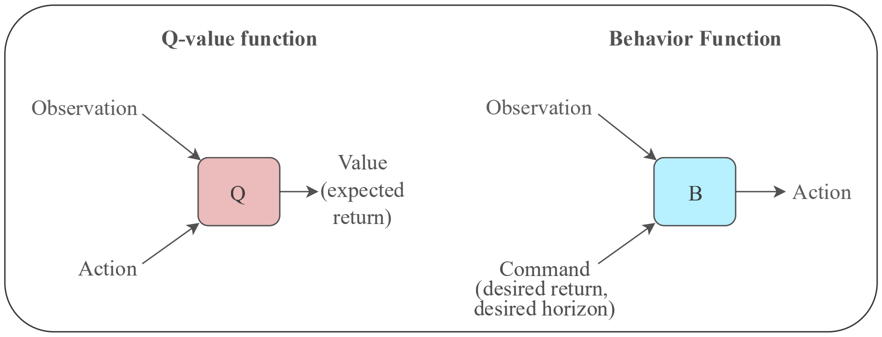


Figure 1: A key distinction between the action-value function (Q) in traditional RL (e.g. Q-learning) and the behavior function (B) in ˥ꓤ is that the roles of actions and returns are switched. In addition, B may have other command inputs such as desired states or the desired time horizon for achieving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

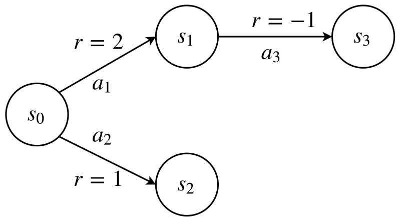


Figure 2: A toy environment with four discrete states.

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 ˥ꓤ 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 ˥ꓤ 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.

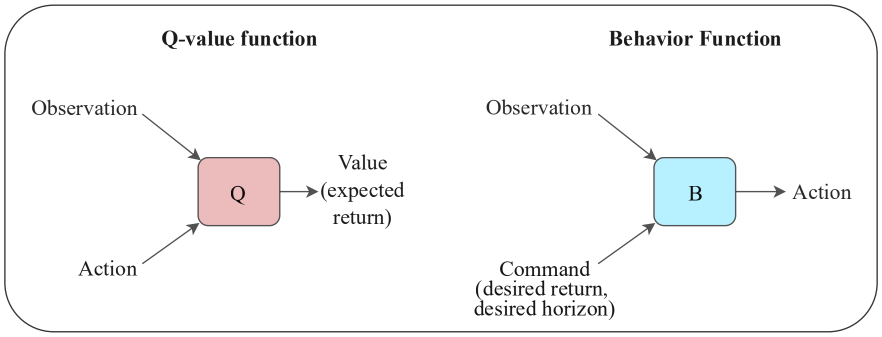


图1：传统RL（例如Q - 学习）中的动作值函数（Q）与RL行为函数（B）之间的关键区别是，动作和返回的作用是切换的。此外，B可能具有其他命令输入，例如所需状态，或获得所需的时间范围，以实现所需的返回。

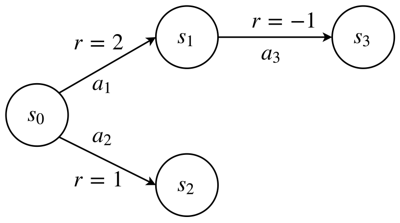


图2：具有四个离散状态的玩具环境。

表1：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颠倒的强化学习

2.1术语和符号

在接下来的内容中，s，a和r分别表示状态，行动和奖励。s和a ($\mathcal{S}$和$\mathcal{A}$)的值集取决于环境。正确的下标表示时间索引（例如$s_t, t \in \mathbb{N}^0$）。我们认为Markovian环境具有标量奖励（$r \in \mathbb{R}$），但RL的一般原理不限于这些设置。策略π：$\mathcal{S} \to \mathcal{A}$是一个在给定状态下选择操作的函数。策略可以是随机的，在这种情况下，它将状态映射到行动上的概率分布。每个情节都由代理与环境的互动以初始状态开始，并在遵循任何策略的同时以最终状态结束。轨迹τ是序列$\langle(s_t, a_t, r_t, s_{t+1})\rangle, t = 0, \ldots, T-1$包含描述长度$T$情节的数据。我们将轨迹的任何子序列称为段或行为，而对部分的累积奖励为回报。

2.2知识表示

传统的无模型RL算法可以广泛归类为基于价值或基于策略的基于价值。基于价值的算法的核心原则是奖励预测：对代理进行培训，以预测通常使用TD学习的任何州采取任何行动的预期折扣未来收益。相反，基于策略的算法是基于直接搜索最大化退货的策略。RL的基本原理与这两个类别不同：给定命令的特定定义，它定义了一个行为函数，该函数封装了有关到目前为止与已知命令兼容的行为的知识。行为函数的性质使用下面的两个示例来解释。

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 ⅂ꓤ, 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

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

Here $\text{len}(\tau)$ 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 ⅂ꓤ 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.

说明性示例1。考虑一个简单的投掷飞镖环境，每个情节都持续一个步骤。一个学会投掷飞镖的经纪人收到的回报与距董事会中心的命中距离成反比。在每个情节中，代理都观察到飞镖的初始状态，并采取了决定投掷力和方向的动作。将基于价值的RL执行此任务将等于训练代理，以预测各种动作和初始状态的预期收益。然后，这些知识将用于行动选择，例如采取预期回报最高的行动。

在RL中，代理商的知识不是根据各种状态和行动的预期收益来表示的，而是与各种状态兼容的行动和所需的回报，即代理人学习过程的输入和目标。将使用行为函数B[2]学习的经验，将对飞镖投掷代理人直接制作撞击板上所需位置的动作。图1在示意性地说明了B和Q - 值函数之间通常使用的基于值的RL的功能之间的差异。由于此环境由一个时间步的情节组成，因此Q和B都可以使用SL学习。下一个示例说明了更典型的RL设置，并显示了更长的时间范围。

说明性示例2。考虑图2中所有轨迹在s_0或s_1中启动的简单确定性马克维亚环境，并以s_2或s_3结束。另外，请考虑类型的命令：在当前状态的给定期望的视野中获得给定的所需返回。然后，基于此环境中所有唯一行为的集合的行为函数可以在表1中以表格形式表示。它将其映射到与执行命令兼容的该状态中要采取的操作。换句话说，它回答了以下问题：“如果代理处于给定状态并希望在给定的视野上给定的回报，那么下一步应该采取哪种操作？”通过设计，此功能现在可以在没有更多知识的情况下在环境中执行任何有效的命令。

行为函数的两个属性是显着的。首先，即使在确定性的环境中，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，如果过渡$s_0 \to s_2$的奖励为2。因此，通常，B会产生对动作的概率分布。其次，B从根本上取决于用于构建它的轨迹集。使用损失函数L，我们为一组轨迹$\mathcal{T}$的最佳行为函数$B^*_{\mathcal{T}}$定义为

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

这里的len（τ）是任何轨迹τ的长度。对于适当的参数化B，我们将观察到的动作分布和预测分布之间的跨渗透性作为损失函数。同等地，我们搜索使用传统的监督学习工具，可以最大程度地提高行为函数生成可用数据的可能性。同样，我们可以通过策略定义行为函数。 B^*_{π}而不是一组轨迹，而是将相同的损失最小化，而在根据π进行作用时生成的轨迹的分布。

2.3一种用于最大化情节回报的RL算法

原则上，可以为在有足够时间（例如随机策略）中生成所有可能的环境中生成所有可能的轨迹的任何策略，然后选择导致在环境中可以实现的期望的地平线上的任何所需返回的动作中学习行为函数。但是，这样的学习程序是不切实际的，因为它依赖于使用固定政策的无方向探索。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 $c_t \coloneqq (d^r_t, d^h_t)$, 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，而不是在任何视野上实现任何可能的回报。因此，本文中使用的混凝土算法在迄今为止的一组轨迹（或代理的经验）上训练行为函数，并结合了最小的添加，这些添加的添加功能可以连续收集具有更高回报的轨迹。

算法1中描述了所提出的算法的高级伪代码。它首先要初始化一个空的重播缓冲液以在训练过程中收集代理的经历，并在训练过程中收集几集随机相互作用。通过对重播缓冲区中记录的先前经验的监督培训，可以不断提高代理的行为函数。在每个训练阶段之后，行为函数被用来在环境中起作用，以获取添加到重型缓冲区中的新体验。此过程一直持续到达到停止标准为止，例如达到允许与环境的最大数量相互作用。本节的其余部分描述了算法的每个步骤，并介绍了超参数。附录A中还提供了超参数的简明列表。

2.3.1重播缓冲区

RL不会明确地最大化回报，而是依靠探索来不断发现较高的回报轨迹，以便可以对其进行训练。为了推动学习进度，我们发现使用一个重播缓冲区，该缓冲区包含固定的最大轨迹数量，到目前为止，回报率最高，并通过返回按顺序排序。最大缓冲尺寸为超参数。由于代理人开始以零体验学习，因此通过在环境中执行随机操作而生成一组初始轨迹。轨迹被添加到重播缓冲区中，并用于开始训练代理的行为功能。

2.3.2行为函数

如前所述，在情节中的任何时间t时，当前的行为函数B对响应当前状态s_t和命令c_t的响应产生一个动作分布：$=(d_t^r, d_t^h)$，其中$d_t^r \in \mathbb{R}$是所需的返回，$d_t^h \in \mathbb{N}$是时间t的所需时间范围。预测的动作分布
$P(a_t|s_t, c_t) = B(s_t, c_t; \theta)$，其中$\theta$表示可训练参数的向量，预计将导致成功执行命令$c_t$被解释为：“在下一个$d_t^h$步骤中获得返回$d_t^r$”。对于给定的初始命令输入$c_0$，$B$可用于使用算法2来生成轨迹，通过对当前命令进行预测并根据获得的奖励和经过的时间更新命令。

一个重要的实现细节是d_t^h始终设置为最大（$d_t^h, 1$），因此它是有效的时间范围。此外，d_t^r被夹住，以至于它在环境中可实现的最大回报率上限。这仅影响代理评估（而不是训练），并避免了负奖励（r_t）可以导致所需的回报的情况，而这些回报是从任何状态中无法实现的（请参见算法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训练行为功能

如第2.2节所述，B接受了过去一集中大量输入目标示例的监督培训。训练的目的是使行为功能根据等式1的重播缓冲区中所有先前记录的轨迹一致。

为了从重播缓冲区中的随机情节中绘制训练示例，随机选择了时间步长索引t_1和t_2，以使$0 \leq t_1 < t_2 \leq T$，其中T是所选情节的长度。然后，培训B的输入为（$s_{t_1}, (d^r, d^h)$），其中$d^r = \sum_{t=t_1}^{t_2} r_t$和$d^h = t_2 - t_1$，目标是$a_{t_1}$，在$t_1$处采取的动作。总而言之，培训示例是通过在过去选择时间范围，动作，观察和奖励而生成的，并生成与它们一致的输入目标对。

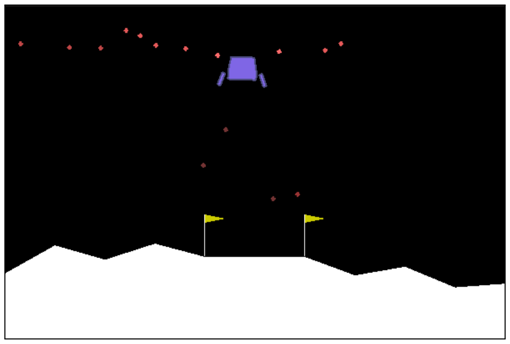
几种启发式方法可用于选择并将培训示例组合为基于梯度的SL的迷你批次。对于本文中的所有实验，从每个情节中进行了“尾随段”，即，我们设置$t_2 = T - 1$，其中T是任何情节的长度。这会放弃大量潜在的培训示例，但非常适合情节任务，其目标是优化总奖励，直到每一集结束。它还使培训变得更容易，因为行为函数只需要学习执行可能的命令的子集即可。为了使设置保持简单，在每个训练步骤中为所有实验执行了使用ADAM [13]的固定训练迭代。

2.3.4采样探索命令

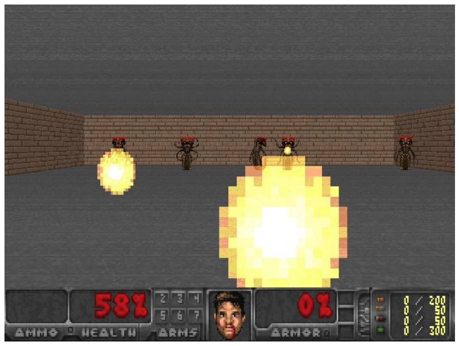
在每个训练阶段之后，代理可以尝试产生新的，以前不可行的行为，从而有可能获得更高的回报。要通过概括从这种探索中获利，必须首先创建一组新的初始命令c_0，以在算法2中使用。我们使用以下过程来示例命令：

1。从重播缓冲区末尾（即，回报率最高）的许多发作。该数字是一个超参数，在训练期间保持固定。 2。探索性期望的地平线d_0^h设置为所选情节长度的平均值。 3。从统一分布$\mathcal{U}[M, M + S]$中取样探索性所需的回报d_0^r，其中M是平均值，S是所选情节返回的标准偏差。

由于其简单性和使用单个超参数调整策略的能力，因此选择了此过程。凭直觉，它试图产生新的行为（在环境随机性的帮助下），该行为在重播中最著名的行为的边缘获得回报。 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

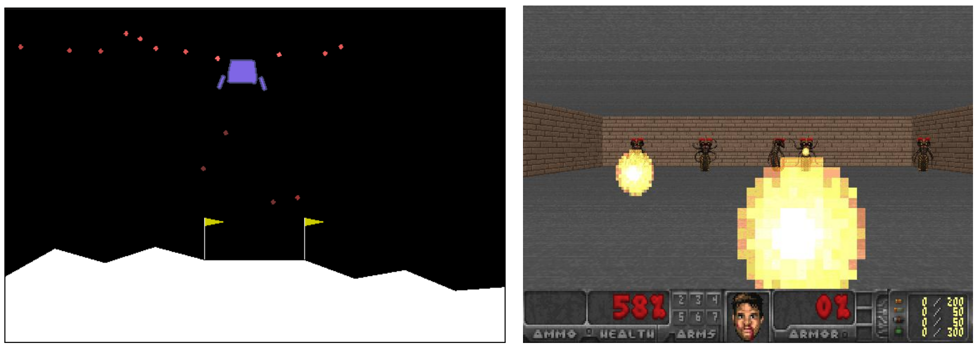
Algorithm 2 is also used to evaluate the agent at any time using evaluation commands derived from the most recent exploratory commands. The initial desired return d^r_0 is set to the lower bound of the desired returns from the most recent exploratory command, and the initial desired horizon d^h_0 from 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，代理不观察到视觉表示，而是8维状态向量。在Bakecover-V0中，代理观察了一个下采样的灰度视觉输入。

在实践中，选择导致行为与现有经验有意义的行为，从而推动学习进步有意义的行为非常重要。不当的探索策略会导致学习缓慢或失速。

2.3.5产生经验

一旦采样了探索性命令，就可以简单地使用算法2生成新的探索性交互作用，该算法2通过从行为函数预测的动作分布中反复采样，并为下一步更新其输入。在每次学习的迭代中都会生成固定数量的情节，并添加到重播缓冲区中。

2.3.6评估

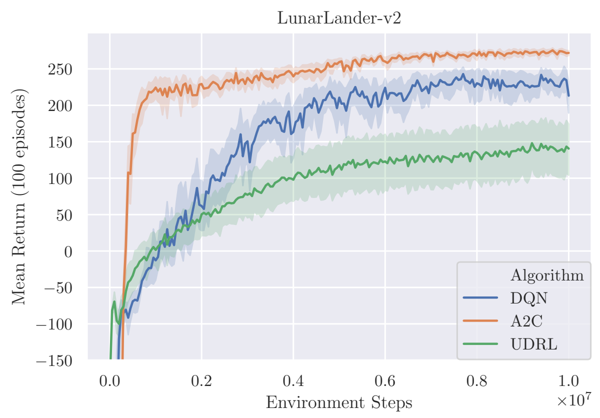
算法2还用于使用从最新探索性命令得出的评估命令随时评估代理。最初所需的返回d_0^r设置为最新探索性命令所需返回的下限，并且重复使用了最新的探索性命令中的初始所需的范围d_0^h。在某些情况下，也可以使用贪婪的动作（使用动作分布的方式），但我们在这里省略此选项为简单起见。

3个实验

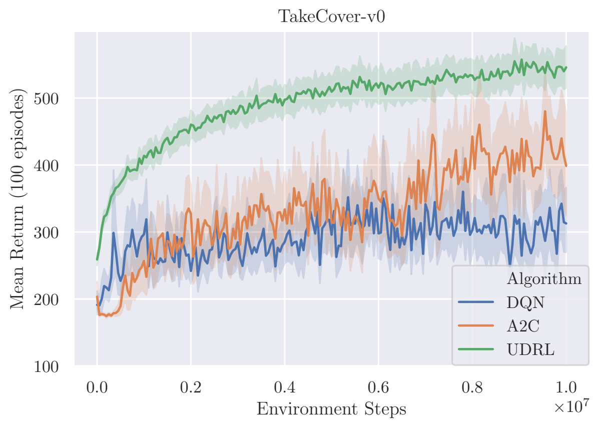
我们实验的目的是确定RL的实际可行性，并将其性能置于两种众所周知的传统RL算法的背景下：深Q-NET WORKS（DQN; 20）和Advantage Actor-Critic（A2C；同步版本的算法版本） Mnih等人提出的[21]。

3.1环境

Lunarlander-V2（图3A）是一个简单的马尔可夫环境，可在体育馆RL图书馆[4]中提供，其目的是通过控制其主发动机和侧面发动机将航天器降落在降落垫上。在情节中，代理在每个时间步骤中都会获得负奖励，该步骤降低了幅度的距离，它越接近最佳着陆点



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



(b) On TakeCover-v0, ⅂ꓤ 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.

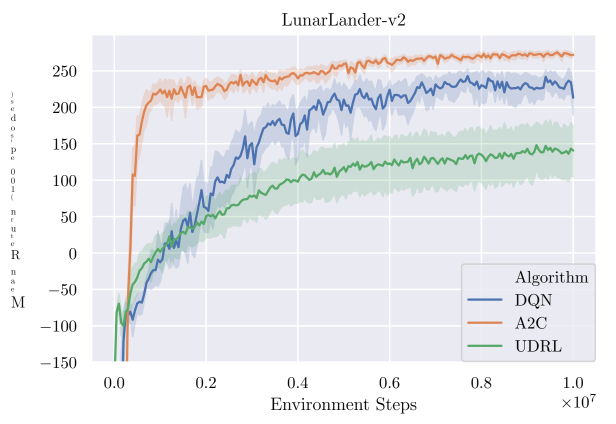
position in terms of both location and orientation. The reward at the end of the episode is -100 for crashing and +100 for successful landing. The agent receives eight-dimensional observations and can take one out of four actions.

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 ⅂ꓤ 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 160×120 to 32×32.

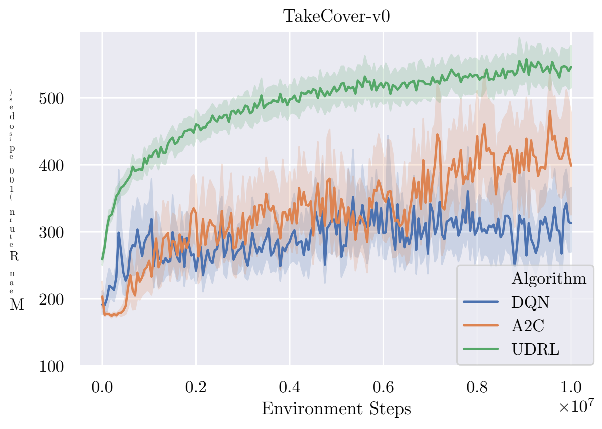
3.2 Setup

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上，RL能够训练降落航天器的代理，但被传统的RL算法击败。



（b）在Bakecover-V0上，RL能够始终如一地产生高性能的代理，而表现优于DQN和A2C。

图4：Lunarlander-V2和Takecover-V0的评估结果。实线代表使用调谐的超参数和实验种子1-20的评估得分的平均值。阴影区域代表使用1000个自举样品的95%置信区间。每个评估得分是100集回报的平均值。

位置和定向方面的位置。情节结束时的奖励是-100，对于成功着陆的撞车和+ 100。代理人收到八维观察结果，可以采取四个动作中的一个。

Takecover-V0（图3B）环境是Visual RL研究的Vizdoom库的一部分[12]。代理被催生在矩形房间的墙壁中心旁边，面对对面的墙壁，怪物随机出现并向经纪人射击火球。它必须学会通过向左或向右移动以尽可能长的时间避免火球。奖励为+ 1对于代理人生存的每个时间步骤，因此对于RL代理，我们始终将所需的视野设置为与所需的奖励相同，并将任何分数值转换为整数。从技术上讲，代理具有与环境的非马克维亚界面，因为它始终无法看到整个相对的墙。为了降低部分可观察性的程度，将最近的八个视觉框架堆叠在一起以产生代理观察。框架也将转换为灰度，并从$160 \times 120 \times 32$的原始分辨率下采样。

3.2设置

所有代理使用人工神经网络实施。使用针对Lunarlander-V2的完全连接的前馈网络和卷积神经网络（CNNS; 16），用于TakeCover-V0，使用了UDRL药物的行为函数。命令输入通过固定缩放系数缩放，由完全连接的sigmoidal层转换，然后将元素与观察到的输入的嵌入乘以（在完全连接的网络的第一层之后；在所有CNNS的所有卷积层之后））。除了对UDRL命令输入的小小修改之外，网络架构对于所有算法都是相同的。

所有实验均以10m的环境步骤进行，并以50k步长评估100个发作。对于每个环境，首先使用随机抽样来根据最终性能找到每种算法和模型的良好超参数。通过这种配置，对于每个环境和算法，用20种种子（从1到20）执行最终实验。从[1m，10m）中取样用于重置环境的随机种子以进行训练，[0.5m，1m）在高参数调整期间进行评估，[1，0.5m）以最终评估使用最佳的超参数。附录B中提供了高参数调整程序的详细信息。

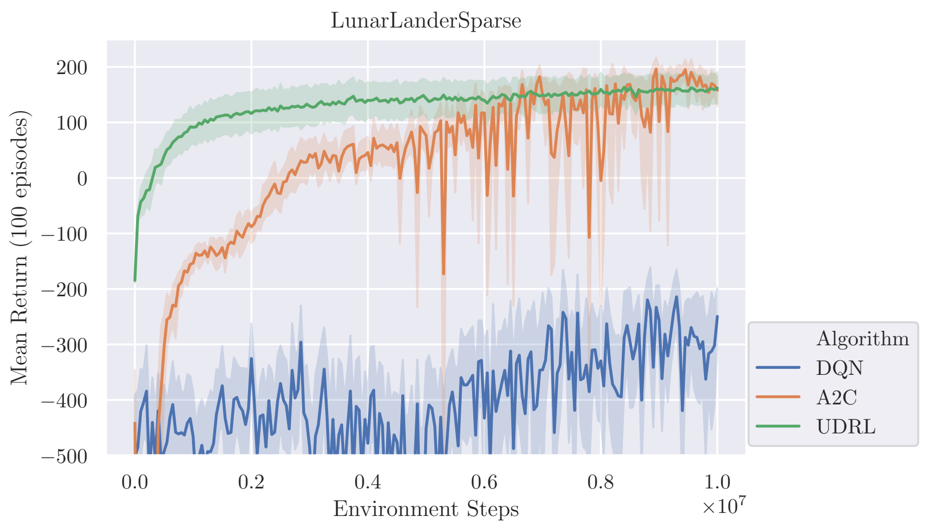


Figure 5: Results for LunarLanderSparse, a sparse reward version of LunarLander-v2 where the cumulative reward is delayed until the end of each episode. ⅂ꓤ learns both much faster and more consistently than both DQN and A2C. Plot semantics are the same as Figure 4.

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.

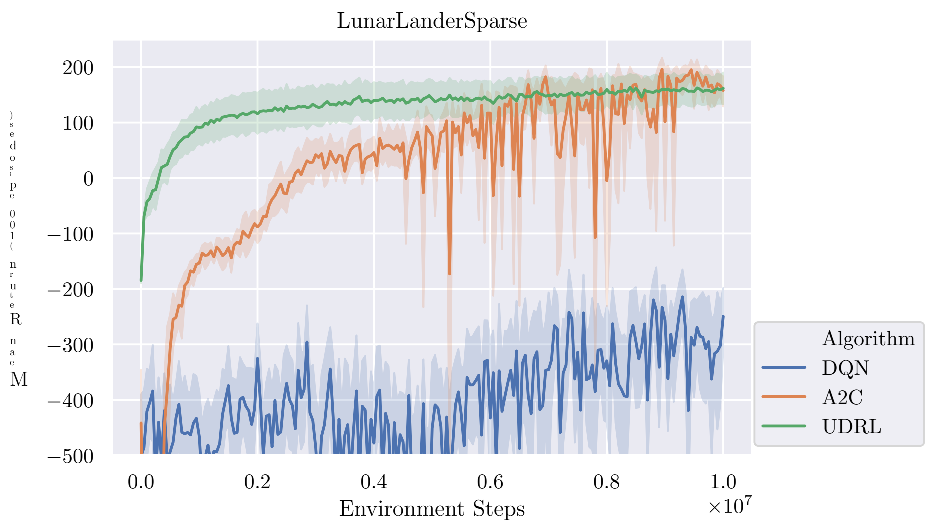


图5：Lunarlandersparse的结果，Lunarlandersparse是Lunarlander-V2的稀疏奖励版本，其中累积奖励被推迟到每集结束。 RL比DQN和A2C更快，更一致地学习。情节语义与图4相同。

3.3结果

最后20次运行的结果在图4中绘制，深线显示了平均评估回报，并带有1000个自举样品的阴影区域，指示95%的置信区间。

对于Lunarlander-V2，100–140的回报表明，近距离最佳政策达到了成功的降落和200以上的回报。在此任务上，RL落后于DQN和A2C。对单个代理商（对于不同种子）的检查表明，它能够始终如一地训练能够成功着陆的代理，但是尽管有些代理商很快学习并获得类似于A2C/DQN的回报，但其他一些高原在较低的回报处。[3]，我们猜想了由于其密集的奖励结构和巨大的奖励信号，这种环境非常适合通过设计学习。

对于Takecover-V0，由于情节时间限制，最大可能的收益为2100。但是，由于环境的难度（怪物的数量随时间增加）和部分可观察性，如果100集中的平均奖励大于750。 ，证明其适用于高维控制问题。

3.4奖励结构的重要性：稀疏的月球着陆器

DQN和A2C在Lunarlander-V2上的表现更好，主要是因为其奖励功能更适合传统算法？为了回答这个问题，对设置进行了修改，以积累所有奖励，直到每集结束，并仅在最后一个时间步骤提供给代理。其他所有时间步骤的奖励为零，因此总的情节返回保持不变。重复评估研究，包括针对新的Lunarlandersparse环境的所有三种算法的高参数搜索。最后20次运行的结果在图5中绘制。

RL代理（绿色）通过这种奖励功能和表现优于DQN和A2C的速度更快，更可靠地学习，两者在处理延迟和稀疏的奖励方面都遇到了严重的困难。由于大量的超参数搜索，A2C可以在此任务上获得高奖励，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 $_{\text{LR}}$ 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），在培训期间相当不稳定。总体而言，我们发现RL能够培训具有稀疏和密集奖励（Lunarlandersparse和Takecover-v0）的训练代理商，但是在某些环境中，稀疏的奖励可能比密集的奖励更好。这种违反直觉的财产是未来研究的重要途径。

上面的结果还使我们对RL基准有了更广泛的观察。 Lunarlander-V2和Lunarlandersparse环境中的“实际”任务完全相同。即使在两个环境中，构造中相同的动作序列也相同。然而，我们基于奖励函数的选择，基于不同的原理获得了算法的截然不同的结果。但是，基准问题的奖励功能通常与现有算法并排开发，这可能会无意中偏爱某些学习范式而不是其他学习范式。摆脱这种偏见的一种潜在方法是使用各种基础RL任务使用各种奖励功能评估每种算法，以了解其对新RL问题的适用性。

3.5训练有素对所需回报的敏感性

RL目标训练代理，以在已知的视野上获得所有已知的回报，但是我们实验中使用的完整学习算法旨在随着训练的进展而获得更高的回报。这是通过在重播缓冲区中保留最高回报发作的方法，并偏向于更高回报的探索来完成。然而，在训练结束时，发现代理可以根据初始所需的发作返回（d_0^r）的水平表现出巨大的行为变化。

我们通过设置d_0^r的各种值并绘制所获得的平均发作返回100集，从而在所有三个环境的训练结束时评估了代理。结果如图6所示。图6a和6b显示了在Lunarlander-V2和Lunarlandersparse上随机选择的代理的获得的和所需回报之间的密切相关性。请注意，在培训的后期阶段，代理只对剧集进行了培训，其回报接近最大值。

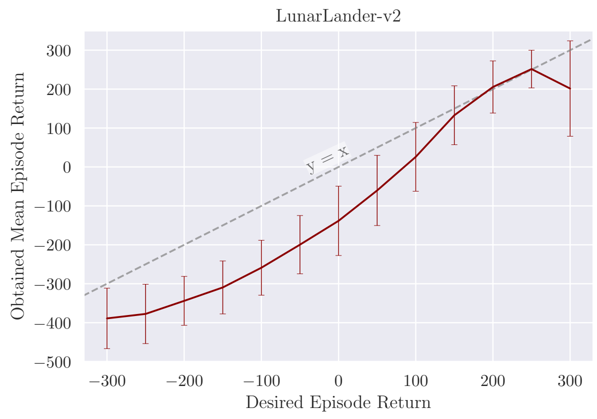
并非所有代理都达到了所需和获得的回报之间的强烈对应关系。图6c显示了另一个为大多数所需返回值求解任务的代理，并且仅在非常低的所需返回值的值中获得较低的回报。这表明训练期间的随机性会影响受过训练的代理对不同命令的推广方式，并提出了另一个方向进行未来的研究。

在Bakecover-V0环境中，很难获得所需回报的精确值。环境中的随机性（怪物随机出现并在随机方向上射击），并且在发作中增加了难度，这意味着无法获得比200较低的回报，并且很难获得更高的平均收益。图6D中的结果反映了这些约束，但仍表明该代理在某种程度上对命令输入敏感。

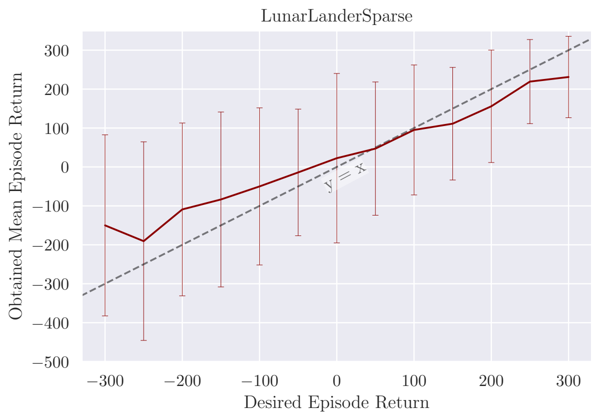
4项相关工作

通过SL改善RL，除了本文范围之外，还具有悠久而丰富的历史[33，37]。例如，基于经验重播的RL系统[18]试图利用过去（非政策）体验的SL来提高价值函数近似。可以通过目标条件价值功能来增强政体培训[11]（另请参见Pong等人[27]，Schaul等人[31]），因此目前政策未实现的目标的价值功能也可以根据相同的交互历史记录进行更新。这个想法最近与Andrychowicz等人的经验重播相结合。 [1]并扩展到Rauber等人的政策梯度。 [28]。哦，等。 [23]拟议的学习模仿过去的良好行动来增强参与者批评算法。尽管存在一些差异，但上述所有算法仍然使用学习价值函数依赖奖励预测，而RL都不使用。

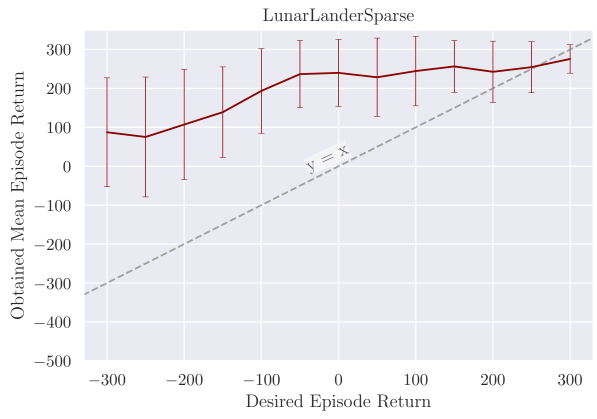
在学习奖励或目标条件政策方面，还进行了大量的工作，直接产生行动[5,6,15,35]，但这些依赖于预训练的世界模型，目标和最佳策略参数的数据集，或学习的政策梯度。虽然RL中的行为函数确实与目标条件具有高级相似性



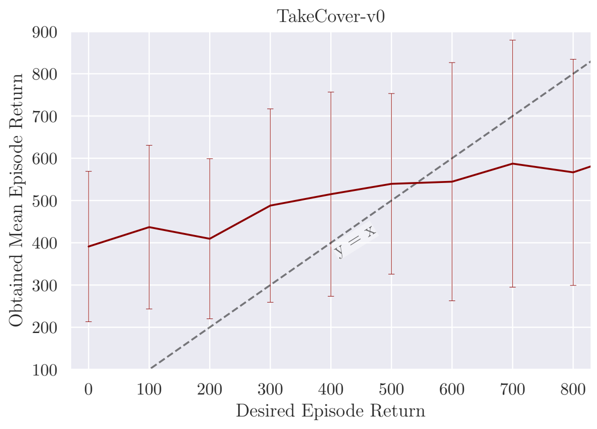
(a)



(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.

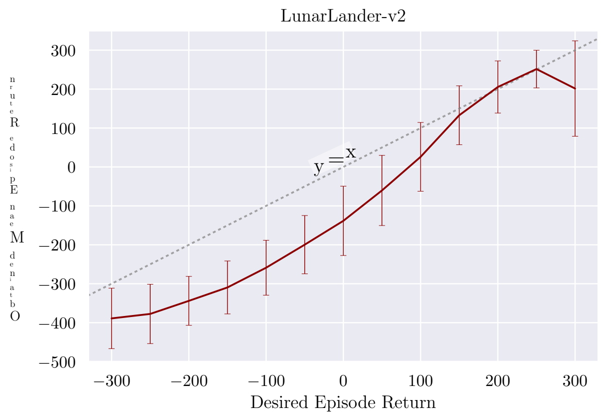
policies, the differences motivating its name are: (a) it takes time-varying desired returns and time horizons as inputs (Algorithm 2), as opposed to fixed goal states, (b) it does not predict rewards at all, (c) it is trained using SL (not policy gradients) on all past behavior, eliminating distinctions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

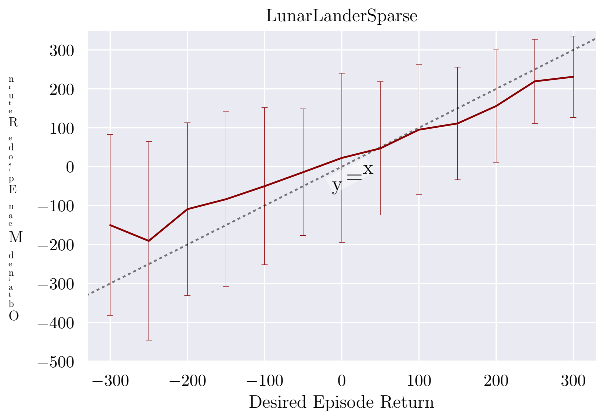
A control approach proposed by Dosovitskiy and Koltun [7] uses SL to predict future values of measurements (possibly rewards) given actions, which also sidesteps traditional RL algorithms. A characteristic property of ⅂ꓤ, however, is its very simple shortcut: it learns the mapping from rewards to actions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

5 Discussion

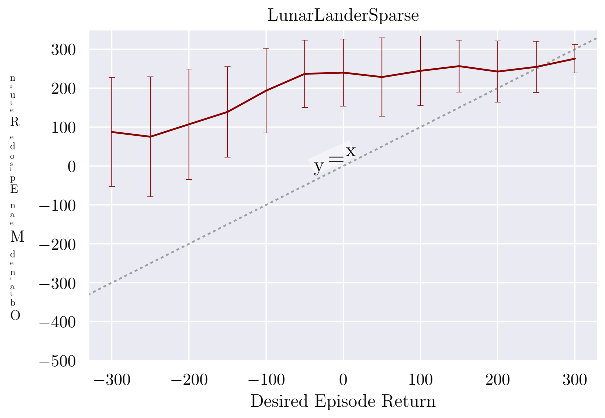
We introduced basic ideas of ⅂ꓤ and showed that it can solve certain challenging RL problems. Since RL benchmarks often tend to get developed alongside RL algorithms, a departure from traditional paradigms may motivate new



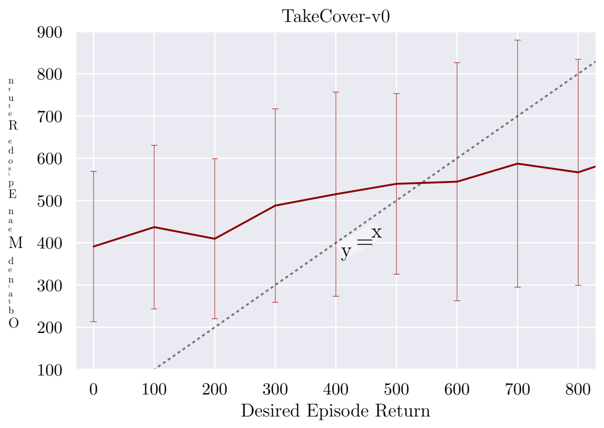
(a)



(b)



(c)



(d)

图6：在训练结束时，UDRL代理人获得了与所需的情节返回。每个评估包括100集。误差条表示与平均值的标准偏差。

策略，促使其名称的差异是：（a）与固定目标状态相比，它需要时间变化的期望回报和时间范围作为输入（算法2），（b）它根本不预测奖励，（c）它是对所有过去行为的SL（不是政策梯度）训练的，从而消除了政策和外部培训之间的区别。

PowerPlay [32，36]使用额外的任务输入来直接制作动作，并且也经过培训，以确保它不会忘记以前学习的技能。但是，其任务输入未根据先前实现的任务/奖励系统地选择。

Dosovitskiy和Koltun [7]提出的一种控制方法使用SL来预测给定动作的未来测量值（可能是奖励），这也避开了传统的RL算法。但是，RL的特性属性是它非常简单的快捷方式：它直接从（可能是偶然的）体验中学习了从奖励到动作的映射，并且根本无法预测奖励。

5讨论

我们介绍了RL的基本思想，并表明它可以解决某些具有挑战性的RL问题。由于RL基准通常倾向于与RL算法一起开发，因此偏离传统范式可能会激发新的范围

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.

比传统RL更好的问题域。

许多基于TD的RL算法都使用折扣因子扭曲了真实的环境回报。 TD学习对采取动作的频率也非常敏感，这可能会限制其对机器人控制的适用性[26]。相比之下，RL明确地考虑了观察到的奖励和时间范围，以自然的方式不假定无限的视野，并且不会遭受基本RL问题的扭曲。请注意，其他算法（例如进化RL [22]）也可以以其他方式避免这些问题。

当将非政策性TD学习与高维函数近似结合使用时，就会出现众所周知的问题。这些问题（由Sutton和Barto [37]称为致命的三合会）可能会严重破坏学习的稳定，并且通常通过添加各种成分来解决，尽管完全的补救措施仍然难以捉摸[38]。

另一方面，RL从根本上与高容量函数近似函数共同起作用（由于表格行为函数无法概括），因此不需要从非平稳目标学习，也不需要在政策和政体和政体之间进行区分训练。取而代之的是，它带来了与灾难性遗忘[19]，持续学习[29]以及从过去的经验到最前沿的概括有关的基本问题。

6未来的研究指示

本文中提出的想法可能会扩展到几个方向。在代理体系结构方面，对于一般可观察的环境，使用反复进行的，而不是前馈神经网络作为行为函数，对于完全可观察但嘈杂的环境有用。为其量身定制的新型命令输入和/或架构修改格式可能会大大改善RL代理的电感偏差。通常，可以采用各种用于模型设计的众所周知的SL技术，正规化和培训来提高RL的学习稳定性和效率。

对于这项最初的研究，培训算法的许多方面都非常简单。未来的工作应利用其他语义来用于命令输入，例如“最多达到*T*时间步骤中的给定目标状态”，以及采样历史段以外的策略，而不是落后段。同样，可能不需要生成恒定的数字探索性发作是通过迭代的，这会降低样本效率。

我们当前的RL版本采用了一种非常简单的探索形式，它可以通过其设计来实现：它只是试图通过从已知状态和退货中概括来实现高回报。这足以驱动在经过少量可用动作和高随机性的经过测试的环境中学习，这有助于发现新的行为可以学习。在其他环境中，其他形式的无向（随机）和有向探索可能是富有成效的甚至是必要的。

最后，基于学习环境模型，时间差异，最佳控制和策略搜索，RL和算法的可能组合有了广阔的开放空间。这样的组合可能会导致在各种环境中有用的更一般的学习代理。

贡献

所有作者都为讨论做出了贡献。 JS提出了RL的基本原理[34]。 WJ为基线开发了代码设置。 PS和FM开发了早期实施，并在完全确定的环境上进行了初步实验。 RKS开发了最终的算法，负责的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Paulo E. Rauber在该项目的早期阶段进行的讨论。我们还感谢福斯蒂诺·戈麦斯（Faustino Gomez）提出了一些建议，这些建议改善了本报告的较早草案，而Nihat Engin Toklu进行了讨论，导致了Lunarlanders parse环境。

参考

[1] Andrychowicz，M.，F。Wolski，A。Ray，J。Schneider，R。Fong，P。Welinder，B。McGrew，J。Tobin，J。Tobin，P。Abbeel和W. Zaremba 2017。 ARXIV：1707.01495 [CS]。

[2] Barto，A。G.和T. G. Dietterich，2004年。强化学习及其与监督学习的关系。在学习和近似动态编程手册中，第672页。[3] Bradski，G.2000。OpenCV库。 DOBB博士的软件工具杂志。 [4] Brockman，G.，V。Cheung，L。Pettersson，J。Schneider，J。Schulman，J。Tang和W. Zaremba，2016年。Openai Gym。 ARXIV预印型ARXIV：1606.01540。 [5] Da Silva，B.，G。Konidaris和A. Barto 2012。学习参数化技能。 ARXIV预印型ARXIV：1206.6398。

[6] Deisenroth，M。P.，P。Englert，J。Peters和D. Fox 2014。多任务策略搜索机器人技术。 2014年IEEE国际机器人与自动化会议（ICRA），pp。 3876–3881。 IEEE。

[7] Dosovitskiy，A。和V. Koltun 2016。学会通过预测未来来采取行动。 ARXIV：1611.01779 [CS]。

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

[9]希尔，A. Schulman，S。Sidor和Y. Wu 2018。稳定的基线。 GitHub存储库。

[10] Hunter，J。D.2007。Matplotlib：2D图形环境。科学与工程学计算，9（3）：90–95。

[11] Kaelbling，L。P.，1993年。学会实现目标。在ijcai，pp。 109
4–1099。 Citeseer。

[12] Kempka，M.，M。Wydmuch，G。Runc，J。Toczek和W.Ja kowski2016。Vizdoom：基于末日的AI AI研究平台，用于视觉增强学习。在2016年IEEE计算情报与游戏会议（CIG），pp。 1–8。 IEEE。

[13] Kingma，D。和J. Ba2014。Adam：一种随机优化的方法。 ARXIV预印型A
RXIV：1412.6980。

[14] Klaus Greff，Aaron Klein，Martin Chovanec，Frank Hutter和JürgenSchmidhuber2017。计算研究的神圣基础设施。在第16届Python的科学会议论文集中，凯蒂·霍夫（Katy Huff），戴维·利帕（David Lippa），狄龙·尼德胡特（Dillon Niederhut）和M. Pacer编辑，pp。 49 - 56。

[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和G. Neumann 2013。通过上下文策略搜索对机器人技能的数据有效概括。在第27届AAAI人工智能会议上。 [16] Lecun，Y.，B。E. Boser，J。S. Denker，D。Henderson，R。E. Howard，W。E. Hubbard和L. D. Jackel，1990年。带有后端网络的手写数字识别。在神经信息处理系统的进步中，pp。 396–404。 [17] Liaw，R。，E。Liang，R。Nishihara，P。Moritz，J。E. Gonzalez和I. Stoica2018。Tune：Tune：分布式模型选择和培训的研究平台。 ARXIV预印型ARXIV：1807.05118。 [18] Lin，L.-J。 1992年。基于强化学习，计划和教学的自我改善反应剂。机器学习，8（3-4）：293–321。 [19] McCloskey，M。和N. J. Cohen 1989。连接派网络中的灾难性干扰：顺序学习问题。学习与动机的心理学，24：109–164。

[20） ARXIV：1602.01783 [CS]。

[22] 2015年。通过深厚的增强学习来控制人类水平。自然，518（7540）：529–533。

[22] Moriarty，D。E.，A。C. Schultz和J. J. Grefenstette1999。强化学习的进化算法。人工智能研究杂志，11：241–276。

[23]哦，J.，Y。Guo，S。Singh和H. Lee 2018。自我模
拟学习。 ARXIV：1806.05635 [CS，STAT]。

[24] Oliphant，T。E. 2015。《 Numpy指南》，第二版。美国：CreateSpace独立出版
平台。

[25]帕斯克（A. 。在Nips-W中。

[26] Plappert，M.，M。Andrychowicz，A。Ray，B。McGrew，B。Baker，G。Powell，J。Schneider，J。Tobin，M。Ch
P. Welinder，V。Kumar和W. Zaremba 2018。多目标增强学习：挑战机器人环境和研究要求。

[27] Pong，V.，S。Gu，M。Dalal和S. Levine 2018。时间差异模型：用于基于模型的控制的无模型
深度RL。 ARXIV：1802.09081 [CS]。

[28] Rauber，P。，A。Ummadisingu，F。Mutz和J. Schmid
huber 2017。事后政策梯度。 ARXIV：1711.06006 [CS]。

[29] Ring，M。B.，1994年。在增强环境中的持续学习。德克萨斯州奥斯汀大学奥斯汀分校计算机科学系博士学位论文，得克萨斯州奥斯汀78712。

[30] Rosenstein，M。T.和A. G. Barto 2004。监督参与者 - 批判性的增强学习。在学习和近似动态编程手册中，第672页。

[31] Schaul，T.，D。Horgan，K。Gregor和D. Silver 2015。通用值函数近似值。在国际机器学习会议上，pp。 1312–1320。

[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。强力游戏：通过不断寻找最简单的仍然无法解决的问题来培训越来越普遍的问题解决者。心理学领域，4：313。 [33] Schmidhuber，J。2015。神经网络中的深度学习：概述。神经网络，61：85−117。 [34] Schmidhuber，J。2019。强化学习颠倒：不要预测奖励 - 只是将其映射到行动中。 Nnaisense/IDSIA技术报告。

[35] Schmidhuber，J。和R. Huber 1991。学会生成人工中央凹轨迹进行目标检测。国际神经系统杂志，2（1＆2）：125−134。

[36] Srivastava，R。K.，B。R. Steunebrink和J. Schmidhuber 2013。使用PowerPlay进行了首次实验。神经网络，41：130−136。 [37] Sutton，R。S.和A. G. Barto 2018。强化学习：简介。麻省理工学院出版社。 [38] Van Hasselt，H.，Y。Doron，F。Strub，M。Hessel，N。Sonnerat和J. Modayil，2018年。深钢筋学习和致命的三合会。 ARXIV：1812.02648 [CS]。 [39] Walt，S。v。d。，S。C。Colbert和G. Varoquaux2011。Numpy阵列：有效数值计算的结构。科学与工程学计算，13（2）：22−30。 [40] Waskom，M.，O。Botvinnik，D。O'Kane，P。Hobson，J。Ostblom，S。Lukauskas，D。C。C. Gemperline，T。Augspurger，Y。Halchenko，J. B.Cole，J.B.Cole，J.Warmenhoven，J.J De Ruiter，C。Pye，S。Hoyer，J。Vanderplas，S。Villalba，G。Kunter，E。Quintero，P。Bachant，M。Martin，K。Meyer，A。Miles，A。Miles，Y。Ram，T。Brunner，T。Yarkoni，M。L。Williams，C。Evans，C。Fitzgerald，Brian和A. Qalieh2018。Mwaskom/seaborn：v0.9.0（2018年7月）。 [41] Watkins，C。J. C. H. 1989。从延迟的奖励中学习。牛津国王学院的博士学位论文。 [42] Williams，R。J.，1992年。用于连接剂增强学习的简单统计梯度遵循算法。机器学习，8（3-4）：229−256。

A Upside-Down RL Hyperparameters

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

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]

颠倒的RL超参数

表2总结了RL的所有超参数的名称和角色。

表2：UDRL超参数的摘要

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高参数调整

通过随机采样Lunarlander-V2和Lunarlandersparse的256个配置来调整超参数，并为Takecover-V0进行72个配置。对于Lunarlander-V2，用3个随机种子评估了超参数的每个随机构型，并将结果平均。对于其他任务，使用单个种子评估每个配置。

根据最后20个评估的评估得分的平均值选择了最佳的超参数配置，从而在训练结束时产生了具有最佳平均性能的配置。

在以下小节中，我们定义了为每个环境和算法调整的每个超参数的值列表。对于DQN和A2C，任何其他超参数都以其稳定生物线的默认值留下，但是我们确实启用了原始论文中找不到的其他技巧，例如多步自举或双Q学习。

B.1 Lunarlander-V2和Lunarlandersparse

建筑超参数

- 网络体系结构（指示每层单位数量）：[[[32]，[32，32]，[32，64]，[32，64，64]，[32，64，64，64，64]，[64]，[64]，[64]，[64]，[64]， [64，64]，[64，128]，[64，128，128]，[64，128，128，128]]

DQN超参数

- 激活功能：[tanh , relu]
- 批量大小：[16，32，64，128]
- 缓冲尺寸：[10,000、50,000、100,000、500,000、1,000,000]
- 折扣因子：[0.98，0.99，0.995，0.999]
- 勘探分数：[0.1，0.2，0.4]

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

A2C Hyperparameters

- Activation function: $[\tanh, \text{relu}]$
- Discount factor: [0.98, 0.99, 0.995, 0.999]
- Entropy coefficient: [0, 0.01, 0.02, 0.05, 0.1]
- Learning rate: $\text{numpy.logspace}(-4, -2, \text{num} = 101)$
- 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`: $\text{numpy.logspace}(-4, -2, \text{num} = 101)$
- `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

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 ꓶꓤ the following choices were different:

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

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

A2C超参数

- 激活功能：[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 , num = 101）
- 价值函数损失系数：[0.1，0.2，0.5，1.0]
- RMSprop的衰减参数：[0.98，0.99，0.995]
- 每个更新的步骤数：[1，2，5，10，20]

颠倒的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 , 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个卷积层，每个网络都有3个× 3滤波器，1个Pixel输入填充板在各个方向和2个像素的步幅。

从[[32，48，96，128]，[32，64，128，256]，[48，96，192，384]中取样了卷积层的结构（指示每层卷积通道数）]。

从[[64，128]，[64，128，128]，[128，256]，[128，256，256]，[128，256]，[128，128]，[256，[256 , 256]]。

DQN和A2C的过度选择与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 软件实施

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]
- 雷音调0.6.6 [17]
- 神圣[14]
- Seaborn [40]
- 稳定的生物线2.5.0 [9]
- Vizdoom 1.1.6 [12]
- Vizdoomgym [8]