Learning Routines for Effective Off-Policy Reinforcement Learning

Edoardo Cetin [1]   **Oya Celiktutan** [1]

Abstract

The performance of reinforcement learning depends upon designing an appropriate action space, where the effect of each action is measurable, yet, granular enough to permit flexible behavior. So far, this process involved non-trivial user choices in terms of the available actions and their execution frequency. We propose a novel framework for reinforcement learning that effectively lifts such constraints. Within our framework, agents learn effective behavior over a *routine space*: a new, higher-level action space, where each *routine* represents a set of 'equivalent' sequences of granular actions with arbitrary length. Our routine space is learned end-to-end to facilitate the accomplishment of underlying off-policy reinforcement learning objectives. We apply our framework to two state-of-the-art off-policy algorithms and show that the resulting agents obtain relevant performance improvements while requiring fewer interactions with the environment per episode, improving computational efficiency.

1. Introduction

The applicability of machine learning has seen tremendous advancements with the advent of deep, expressive models (LeCun et al., 2015). These models enabled to tackle complex problems given large quantities of data through end-to-end training, removing bottlenecks related to hand-designed features. Such trend has seen success also in reinforcement learning, where deep learning enabled the achievement of significant milestones (Mnih et al., 2013; Silver et al., 2017) and the specification of practical algorithms (Lillicrap et al., 2015; Fujimoto et al., 2018; Haarnoja et al., 2018a).

However, within the reinforcement learning framework, there are still many fixed components, related to the agent's interface with the environment, that are the result of expert-engineering and are often quite influential on the performance (Mahmood et al., 2018). Algorithms that learn also these additional components end-to-end would alleviate some of the deployment burdens and likely improve final performance given enough data. One of such components is the agent's action space. The performance of current reinforcement learning algorithms is dependent on reasoning with an expressive set of actions that have a tangible effect on the current state. Specifying this, generally includes setting an environment-specific fixed frequency with which the agent alternates reasoning and executing behavior.

On the other hand, humans are capable of effectively performing actions to the lowest level of granularity while only directly reasoning with higher-level abstractions of what they *intend* to do. This can be attributed to their observable ability of acquiring, through repetition of a particular task, temporally-extended motor-skills that can be re-enacted without conscious thought (Sweatt, 2009). Doing so allows to abstract away the details regarding the composition of each motor-skill, thus, minimizing the required reasoning to solve the task. We will refer to these motor-skills, acquired with the objective of solving a particular task more efficiently as *routines*.

From these observations, we design a new class of reinforcement learning algorithms that learn behavior together with a new 'action space', which we name the *routine space*. In this space, each *learned routine* represents higher-level concepts of what the agent intends to achieve with respect to the task goal and can be mapped to a whole set of 'equivalent' variable-length sequences of action primitives. Within our framework, this space is learned end-to-end to facilitate the underlying objectives of any base reinforcement learning algorithm.

Our experimental results demonstrate that utilizing our proposed *routine framework* improves the performance of two different off-policy reinforcement learning algorithms tested on the environments from the DeepMind Control Suite (Tassa et al., 2018). Moreover, using our framework, agents need to reason only after experiencing the outcome of each routine rather than each action. Therefore, they are able to query their policy much more infrequently by learning to perform longer routines from states that do not require a fine level of control. Practically, this enables for compu-

[1]Centre for Robotics Research, Department of Engineering, King's College London. Correspondence to: Edoardo Cetin <edoardo.cetin@kcl.ac.uk>.

tationally efficient deployment, faster data-collection, and easier real-time inference (Dulac-Arnold et al., 2019). For access to our open-source implementations, please visit `sites.google.com/view/routines-rl/`.

In summary, the contribution of this work is three-fold:

- We introduce the concept of routines in reinforcement learning and discuss its inherent advantages.

- We propose effective methods to integrate the routine framework with off-policy reinforcement learning and describe two new algorithms, based on the *Twin Delayed DDPG* (Fujimoto et al., 2018) and *Soft Actor-Critic* (Haarnoja et al., 2018b) algorithms.

- We provide an extensive evaluation on the DeepMind Control Suite and show that the routine framework boosts performance and computational efficiency, with agents requiring substantially fewer policy queries to successfully act during an episode.

2. Related Work

Learning higher-level abstractions of an agent's actuation interface has been a long-studied problem in reinforcement learning under several similar conceptual frameworks. *Options* (Sutton et al., 1999; Precup, 2001) represent one of such frameworks in which an agent tries to learn a set of sub-policies, where execution is alternated according to relative termination conditions and a higher-level controller. Learning these sub-policies has been attempted by specifying subgoals (Dayan & Hinton, 1993; Dietterich, 2000) and, more recently, end-to-end optimization (Bacon et al., 2017) with varying degrees of success. Yet, it still has remained an open problem. The routine framework could be viewed as an unconventional instance of the options framework, where the routine space could represent a whole space of task-dependent 'sub-policies' corresponding to fixed distributions over action sequences. Similarly, several works in hierarchical reinforcement learning have attempted to partition an agent's policy into low-level and high-level controllers. One of the key challenges, though, has remained recovering effective low-level behavior. To tackle this challenge, some of these works considered feeding extrinsic rewards for achieving heuristically useful behavior (Florensa et al., 2017) or solving a diverse range of subtasks (Tessler et al., 2017; Hausman et al., 2018). Alternatively, other works studied intrinsic objectives related to exploration measures (Kulkarni et al., 2016; Gregor et al., 2016). Recently, Bahl et al. (2020) also explored hierarchical structures, utilizing a dynamical system to achieve some higher-level reparameterization of the action space, incorporating effective inductive bias into the agent's model.

Macro-actions (Precup et al., 1997) represent another con-cept related to routines and options, proposing to use particular fixed sequences of actions to build a higher-level abstraction of the action space. A simple way of incorporating macro-actions into the learning framework is to specify policies reasoning with action repetitions. Several works showed that such practice can speed up learning and facilitate exploration (Schoknecht & Riedmiller, 2002; 2003; Neunert et al., 2020). Particularly, Lakshminarayanan et al. (2017) proposed a manually augmented discrete action space where each action would be present at two different temporal resolutions, while Sharma et al. (2017) and Biedenkapp et al. (2020) factorized the policy to output both an action and the corresponding number of repetitions. Closer to our work, the framework from Vezhnevets et al. (2016) utilized more expressive macro-actions by keeping a running plan of future behavior to execute, adaptively updated with attentive reading and writing operations.

In line with our conceptualization, several works have considered building explicit embedding spaces of behavior to directly facilitate learning. Within this area, some works considered learning representations of individual actions based on the environment dynamics, in order to aid exploration in large discrete action spaces (Dulac-Arnold et al., 2015; Chandak et al., 2019). Other works also considered building embeddings of temporally-extended behavior by exploiting demonstrations (Tennenholtz & Mannor, 2019) or representations of target 'goal-states' (Vezhnevets et al., 2017; Nachum et al., 2018). Sharing some commonalities with our method, *SeCTAR* (Co-Reyes et al., 2018) used a model to 'auto-encode' fixed-length sequences of states, and learned a low-level policy to follow the reconstructed state trajectory conditioned on the lower-dimensional embeddings. Additionally, Whitney et al. (2019) proposed to learn representations of fixed-length action sequences to be maximally useful for predicting state transitions. While many of these representations demonstrated meaningful results improving sample-efficiency and performance, unlike routines, they still required specifying explicit heuristic objectives to obtain an effective encoding.

3. Preliminaries

3.1. Markov Decision Process

In reinforcement learning, the agent's attempted task can be described as a Markov Decision Process (MDP), defined as $(S, A, r, P, p_0, \gamma)$. At each time-step, the agent experiences a state in its state space, $s \in S$, and performs an action in its action space, $a \in A$. $P : S \times A \times S \to \mathbb{R}$ and $p_0 : S \to \mathbb{R}$ represent the transition dynamics and the initial state distribution, determining the likelihoods of experiencing particular transitions. $r : S \times A \to \mathbb{R}$ is the reward function, outputting numerical measures representing the usefulness of observed behavior towards achieving the task's objective.

The reinforcement learning goal is to obtain a policy π to maximize the expected overall performance quantified by the sum of discounted experienced rewards:

$$\arg\max_{\pi} \mathbb{E}_{p_{\pi}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \qquad (1)$$

where $p_{\pi}(\tau)$ represents the distribution of trajectories τ encountered by the agent from its interaction with the environment.

3.2. Off-Policy Learning

At any point during the agent's trajectory, we can calculate its expected future performance after taking a particular action, quantified by the expected discounted sum of future rewards. This quantity is called the Q-function:

$$Q^{\pi}(s, a) = \mathbb{E}_{p_{\pi}(\tau|s_0=s, a_0=a)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \qquad (2)$$

The Q-function can be represented by a parameterized model Q_{ϕ} and learned for any policy π by iteratively minimizing a squared temporal difference (TD-) loss of the form $\arg\min_{\phi} \mathbb{E}_{(s,a,s',r)\sim B} \left[(Q_{\phi}^{\pi}(s,a) - y)^2 \right]$, where the TD-targets y are obtained by computing the Bellman backups $y = r + \gamma Q_{\phi'}^{\pi}(s', \pi(s'))$. This objective is often optimized by sampling batches of uncorrelated transitions from a replay buffer B and using a slowly updated *target* network $Q_{\phi'}^{\pi}$ to compute the TD-targets y.

The agent's policy can also be represented by a parameterized model π_{θ} and iteratively improved by maximizing its relative Q-function, following the policy gradient theorem (Sutton et al., 2000; Silver et al., 2014):

$$\arg\max_{\theta} \mathbb{E}_{s\sim B, a\sim\pi_{\theta}(\cdot|s)} \left[Q_{\phi}^{\pi_{\theta}}(s,a) \right]. \qquad (3)$$

3.3. Advanced Policy Gradient Algorithms

Current state-of-the-art off-policy algorithms build upon the concepts expressed in Section 3.2 and propose further complementary ideas. In particular, we will be considering the *Twin-delayed DDPG* (*TD3*) algorithm (Fujimoto et al., 2018), which proposes several practical innovations to aid the optimization of the policy gradient objective such as learning independently two Q-networks, $Q_{\phi_1}^{\pi}, Q_{\phi_2}^{\pi}$, and updating the policy less frequently for stability. We will also be considering the *Soft Actor-Critic* (*SAC*) algorithm (Haarnoja et al., 2018a;b), which proposes to optimize an augmented maximum-entropy reinforcement learning objective (Ziebart, 2010):

$$\arg\max_{\pi} \mathbb{E}_{p_{\pi}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + \alpha H(\pi(a_t|s_t)) \right]. \qquad (4)$$

4. The Routine Framework

4.1. Routines Formalization

For continuous control problems, each action can be represented by a vector in $\mathbb{R}^a$ corresponding to some actuation inputs, e.g, the torque to be applied to each of an agent's joints. Generally, the execution length of each action is an environment-specific fixed hyper-parameter, determining with which frequency the agent reasons and interacts with the environment. As such, this value can often influence the resulting agent's performance.

We represent each routine, $n \in N$, as a vector in $\mathbb{R}^n$, corresponding to the *routine space*. Each routine is tied with a subset of S^A:

$$S^A = \{(a_1, a_2, ..., a_l) : l \leq L\}, \qquad (5)$$

corresponding to the set of all action sequences up to some maximum length L, built from the original action space A. Such maximum length limitation ensures that each routine has a finite result when actuated. Conceptually, all the action sequences tied with a given routine should accomplish related results with respect to the task's objective, giving rise to a many-to-one mapping. Unlike actions the meaning of each routine should not depend on a specific agent's actuators and can even change during optimization.

Within our framework, the policy π represents a conditional distribution over routines rather than actions. Additionally, we define a routine Q-function, defined on the set of all states and routines that represents the expected future performance when starting a trajectory by actuating a particular routine:

$$Q^{\pi}(s, n) = \mathbb{E}_{p_{\pi}(\tau|s_0=s, n_0=n)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \qquad (6)$$

4.2. Learning Routines

Routine decoder. In order to execute any given routine in the environment, we define a model that specifies a mapping from routines to action sequences in S^A. We call this model the *routine decoder*, $D_{\omega_1} : N \mapsto (\mathbb{R}^{a\times L}, \mathbb{R}^{L-1})$. In its simplest form, we represent this model with a neural network D_{ω_1}, taking as input a routine n, and outputting an action sequence of length L, $a_{1:L}^n$, together with a list of early termination probabilities $e_{1:L-1}^n$. The early termination probabilities can then be used to sample an action sequence in S^A, by representing the likelihood of 'cutting-off' the outputted action sequence $a_{1:L}^n$ into a subsequence $a_{1:l}^n$ for different lengths l, as described by the following
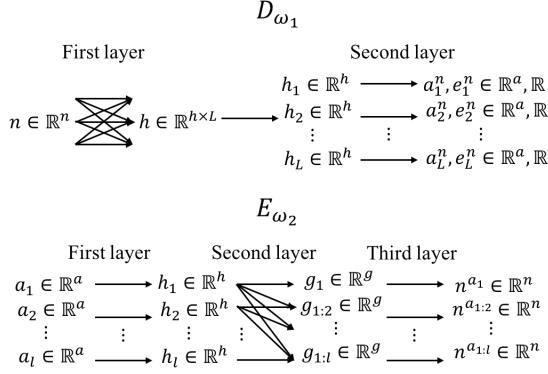
Figure 1. Simplified network structures of the routine decoder D_{ω_1} (Top) and encoder E_{ω_2} (Bottom).

equations:

$$p(a_{1:l}^n|n) = e_l^n \prod_{j=1}^{l-1}(1 - e_j^n) \text{ for } 1 \leq l < L,$$

$$p(a_{1:L}^n|n) = \prod_{j=1}^{L-1}(1 - e_j^n). \tag{7}$$

Therefore, to query the agent's behavior from state s, we evaluate the agent's policy π to obtain a routine n. Then, we feed n to the routine decoder and utilize its outputs to sample the routine length l with its corresponding action sequence $a_{1:l}^n$ to execute.

Routine encoder. Since the meaning of each routine can evolve during training, recording the representations of performed routines within the replay buffer would not be useful. Instead, we record the resulting executed action sequences, which have an invaried effect on the environment. Thus, to use replay buffer data for learning the routine Q-function, we require a model to translate back action sequences from the replay buffer to their corresponding routines. We call this model the *routine encoder*, $E_{\omega_2} : S^A \mapsto N$. We represent this model as an additional neural network E_{ω_2} taking as input action sequences $a_{1:l}$ and outputting the corresponding routine $n^{a_{1:l}}$.

Models parameterization. The routine decoder is parameterized as a 2-layer neural network, taking as input a routine n. Particularly, the first layer of the decoder produces a hidden representation that is further split into L separate sub-representations. These sub-representations are then processed independently by the second layer to obtain L corresponding actions $a_{1:L}$ and early termination probabilities $e_{1:L}$. The routine encoder is parameterized as a 3-layer neural network, taking as input sequences of actions $a_{1:l}$. The first layer starts by embedding independently the actions into l hidden representations. Subsequently, the second layer further embeds the l produced representations through

a set of L 'index-specific' weights and ultimately sums them over, obtaining a unique aggregated representation for the whole sequence. Note that by keeping a running sum of the embeddings we are also able to obtain the aggregated representations for all action sub-sequences $\{a_{1:j} : j < l\}$ as byproducts of this computation. The final layer, then, processes these aggregated representations outputting the corresponding routines $n^{a_1}, n^{a_1,a_2}, ..., n^{a_{1:l}}$. We provide a visual representation of these models in Figure 1.

Consistency objectives. The routine decoder and encoder should represent a many-to-one mapping where different action sequences with equivalent effect on the environment are mapped to individual routines. To achieve this, we propose to impose two consistency objectives to tie the routine decoder D_{ω_1} and encoder E_{ω_2} models. The first objective is to enforce the desired many-to-one consistency, ensuring that reconstructed action sequences are mapped back to the same routine that generated them. Practically, we optimize the routine decoder to decode any particular routine n into an action sequence $a_{1:l}^n$ that would be mapped back to the same routine by the routine encoder $n^{a_{1:l}^n}$. We achieve this by specifying an L2 objective for reconstructing the starting routine representation $J_{mto}(\omega_1)$:

$$\arg\min_{\omega_1} \mathbb{E}_{B,\pi} \left[||E_{\omega_2}(D_{\omega_1}(n))) - n||^2 \right]. \tag{8}$$

The second objective serves to learn the early termination probabilities $e_{1:L-1}$ and is motivated by the prior understanding that, in most tasks, only action sequences with the same length l should be equivalent towards achieving a target goal. Particularly, this is enforced by minimizing a binary cross-entropy loss for the early termination probability in a reconstructed action-sequence of length l, pushing e_l to 1 and e_j to 0 for all $j < l$:

$$L_{lc}(e_{1:L-1}) = \sum_{j=1}^{l-1}(log(1 - e_j)) - log(e_l). \tag{9}$$

This loss is minimized with respect to action sequences sampled from the replay buffer and the resulting objective, $J_{lc}(\omega)$, is optimized for both the routine encoder and routine decoder parameter sets $\omega = \{\omega_1 \cup \omega_2\}$:

$$\arg\min_{\omega} \mathbb{E}_B \left[L_{lc}(D_{\omega_2}(E_{\omega_1}(a_{1:l}))) \right]. \tag{10}$$

4.3. End-to-End Optimization for Off-Policy Learning

An *effective* routine space must strive to facilitate the underlying reinforcement learning task. Hence, the *driving* learning signal used to obtain the routine space comes directly from optimizing the objectives of the underlying reinforcement learning algorithms. In this section, we first formulate
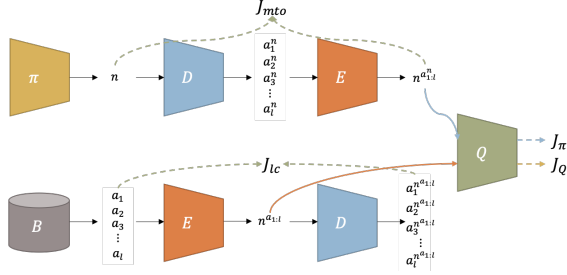
Figure 2. Simplified model optimizations in the routine framework. The top path represents the joint optimization of the policy gradient J_π and many-to-one consistency J_{mto} objectives. The bottom path represents the joint optimization of the temporal difference J_Q and length consistency J_{lc} objectives.

these objectives in the context of applying the routine framework to off-policy reinforcement learning and, subsequently, detail how to perform the end-to-end optimization.

Learning the routine Q-function. To integrate the considered off-policy algorithms with the routine framework, we need to specify a learning procedure to recover an accurate Q-function for every state and routine. We take a *natural* approach to perform this, and optimize a modified squared TD-loss, similarly to how this optimization would be carried out for actions. Particularly, for each sampled state s we collect the sequence of L subsequent actions $a_{1:L}$, next states $s'_{1:L}$ and rewards $r_{1:L}$ from the replay buffer to perform L temporal difference updates in parallel. We encode each 'sub-sequence' of consecutive actions, starting from a_1, into their corresponding routines $n^{a_{1:l}} = E_{\omega_1}(a_{1:l})$. This can be performed efficiently with our routine encoder architecture, feeding only the original full-length sequence $a_{1:L}$. The Q-function objective is then to minimize J_Q:

$$\arg\min_\phi \mathbb{E}_{B,E_{\omega_2}} \left[\sum_{l=1}^{L} (Q_\phi^\pi(s, n^{a_{1:l}}) - y_l)^2 \right]. \quad (11)$$

The TD-targets, y, of this optimization are computed as the discounted sum of rewards obtained from executing actions $a_{1:l}$ together with the target Q value at next state s'_l:

$$y_l = \sum_{j=1}^{l} \left(\gamma^{(j-1)} r_j \right) + \gamma^l Q_{\phi'}^\pi(s'_l, E_{\omega_1}(D_{\omega_2}(\pi(s'_l)))).$$

$$(12)$$

Please note that in Equation 12 the routine used to evaluate the target Q-network $Q_{\phi'}^\pi$ at state s'_l is not simply the routine selected by the policy $\pi(s'_l)$, but rather its 'auto-encoded' version $E_{\omega_1}(D_{\omega_2}(\pi(s'_l)))$. The necessity of this comes from the fact that the Q-function is always updated with routines obtained by encoding action sequences sampled from the replay buffer distribution $E_{\omega_1}(a_{1:l})$, which, in practice, can differ from the distribution of routines outputted by the actor.

Thus, the additional 'auto-encoding' ensures that the inputs of the target Q-function also come from encoded action sequences, empirically providing further stabilization.

This update can be performed efficiently with respect to all routines $n^{a_{1:L}}$ in parallel, making use of matrix multiplication. We provide pseudocode in Section A of the Appendix.

Learning a policy with routines. Given an informative Q-function over our routine space, we can optimize a parameterized policy π_θ to maximize its expected output. Thus, we specify an objective similar to Equation 3, optimizing θ over the Q value estimates from on-policy routines. Namely, we maximize the routine policy gradient objective J_π:

$$\arg\max_\theta \mathbb{E}_{s\sim B, n\sim \pi_\theta(\cdot|s)} \left[Q_\phi^{\pi_\theta}(s, E_{\omega_1}(D_{\omega_2}(n))) \right]. \quad (13)$$

As in Equation 12, the output of the policy is 'auto-encoded' before being evaluated by $Q_\phi^{\pi_\theta}$, bringing the distribution of evaluated routines closer to the one used to learn $Q_\phi^{\pi_\theta}$. The processing performed to compute all four optimization objectives is summarized in Figure 2.

Building a maximally useful routine space. In order to build a routine space that facilitates solving the target task, we utilize the training signal from the Q-function and policy optimizations to learn E_{ω_2} and D_{ω_1}. Particularly, the routine encoder, E_{ω_2}, is trained together with the Q-function to minimize the temporal difference objective J_Q from Equation 11. During this step, we also decode the routine representations computed from the action sequences in the replay buffer, $E_{\omega_1}(a_{1:l})$, to optimize the length consistency objective J_{lc} from Equation 10.

$$\arg\min_\omega J_Q + J_{lc}. \quad (14)$$

This optimization has the effect of building a routine space that facilitates the Q-function $Q_\phi^{\pi_\theta}$ to accurately represent the true Q values of different sets of action sequences. In particular, the routine encoder E_{ω_1} will be pushed to map action sequences with different Q values (thus, different effects towards achieving the task's objective) to different routine representations, facilitating $Q_\phi^{\pi_\theta}$ to minimize the TD-loss. Moreover, E_{ω_1} will be pushed to map the routine representation of unseen action sequences closer to the routine representation of other seen action sequences accomplishing the same task objective, for which $Q_\phi^{\pi_\theta}$ already minimizes the TD-loss.

Similarly, the routine decoder is trained together with the policy to maximize the policy gradient objective J_π from Equation 13. We also utilize the samples from $n \sim \pi_\theta(\cdot|s)$ to jointly optimize the many-to-one consistency objective J_{mto} with $E_{\omega_1}(D_{\omega_2}(n))$ at minimal extra cost:

$$\arg\max_{\omega_2} J_\pi - J_{mto}. \quad (15)$$

This optimization has the effect of building a routine space that facilitates the policy π_θ to output routines for which the decoded actions maximize our estimate of $Q_\phi^{\pi_\theta}$. Note that we do not optimize the routine encoder E_{ω_1} with respect to this objective, since doing so would encourage the 'auto-encoded' routine $E_{\omega_1}(D_{\omega_2}(n))$ to diverge from the original routine n, in order to maximize Eq. 13, without such change being necessarily reflected in the decoded action sequence from $D_{\omega_2}(n)$. Therefore, we avoid the mappings between the routine and action spaces changing only to adversarially *fool* the Q-function $Q_\phi^{\pi_\theta}$ towards outputting high values.

4.4. Advantages of Utilizing Routines

There are multiple advantages in learning an agent that reasons with routines rather than actions. Our framework increases the agent's expressivity, allowing it to output behavior executed at different timescales. As also empirically observed in prior works (Neunert et al., 2020; Dabney et al., 2020), the introduction of temporally-extended structured behavior appears to yield non-trivial exploration benefits in different tasks. Practically, this freedom also enables to lower the amount of reasoning the agent needs to perform to complete a particular episode since its policy can learn to select routines spanning multiple time-steps when a very fine level of control is not needed. This has non-trivial implications for speeding up experience collection, facilitating applications for time-sensitive real-world applications, and better scaling to expensive policy models such as neural network ensembles.

Another kind of advantage comes from the optimization of the reinforcement learning objectives in the routine framework. Reinforcement learning algorithms appear to be sensible to the granularity of their action spaces. For example, different action repeats have concrete effects on the agent's performance in different environments (Braylan et al., 2015), even motivating some recent works to directly learn this parameter (Metelli et al., 2020; Lee et al., 2020). Additionally, techniques such as learning Q-functions with multi-step targets often lead to faster learning by speeding up reward propagation (Sutton & Barto, 2018) and have been widely adopted in many state-of-the-art algorithms (Mnih et al., 2016; Hessel et al., 2018; Barth-Maron et al., 2018). By enabling agents to reason and learn at different frequencies concurrently, the routine framework provides a way to exploit faster reward propagation, without necessarily increasing the variance in the Q-function TD-targets. We perform experiments to better understand and decouple the exploration and learning benefits provided by our framework in Section F of the Appendix.

Hence, there are several characteristics of the routine framework differentiating it from prior works. Our framework does not rely on external heuristics such as state represen-

tations (Vezhnevets et al., 2017; Nachum et al., 2018) or transitions information (Co-Reyes et al., 2018; Whitney et al., 2019) to recover higher-level abstractions of behavior. Instead, our end-to-end approach should recover a routine space tailored to maximize the performance of each individual task, with the potential of mutating to facilitate different learning stages. Moreover, routines can represent arbitrarily complex distributions of action sequences, thus providing increased flexibility as compared to algorithms reasoning with action repetitions (Lakshminarayanan et al., 2017; Sharma et al., 2017; Biedenkapp et al., 2020). Additionally, as described in this section, the computation of routines is specifically well suited for off-policy reinforcement learning, enabling to naturally harness the benefits of faster reward propagation without its major downsides,

5. Practical Algorithms

We integrate the routine framework and optimization procedures specified in Section 4 with additional practices from two state-of-the-art off-policy reinforcement learning algorithms, described in Section 3.3.

5.1. Integration with TD3

We first describe making use of the routine framework and additional models with *TD3* (Fujimoto et al., 2018). Since *TD3* aims to learn a deterministic policy, incorporating routines simply amounts to replacing the TD-loss and policy gradient loss with their routine counterparts, augmented by the many-to-one and length consistency objectives as specified in Section 4.3.

Utilizing a routine decoder also allows injecting Gaussian exploration noise on two levels: the action level and the routine level. Injecting noise at the routine level provides *TD3* means of using additional state-dependent exploration. In our experiments, we use a combination of these two types of noise which appears to yield the best results. We investigate the effects of each distinct type of noise on performance in Section F of the Appendix.

5.2. Integration with SAC

We further describe integrating the routine framework with *SAC* (Haarnoja et al., 2018b). Unlike *TD3*, *SAC* proposes to optimize an augmented maximum entropy objective with a stochastic policy outputting the parameters of an independent Gaussian distribution over actions. When reasoning with routines, we optimize this augmented objective by making use of a 'Gaussian' routine decoder and a deterministic policy. Particularly, the routine decoder now outputs L vectors for both the means $\mu_{1:L}^n$ and standard deviations $\sigma_{1:L}^n$ of independent multivariate Gaussian distributions, together with early termination probabilities $e_{1:L-1}^n$:

$D_{\omega_1} : N \mapsto (\mathbb{R}^{2 \times a \times L}, \mathbb{R}^{L-1})$. To sample an action sequence from a routine n, we first sample the terminating index l, as in Equation 7, to obtain the relevant vector means and standard deviations, $\mu_{1:l}^n$ and $\sigma_{1:l}^n$:

$$p(\mu_{1:l}^n, \sigma_{1:l}^n | n) = e_l^n \prod_{j=1}^{l-1} (1 - e_j^n) \text{ for } 1 \le l < L,$$

$$p(\mu_{1:L}^n, \sigma_{1:L}^n | n) = \prod_{j=1}^{L-1} (1 - e_j^n). \quad (16)$$

Subsequently, we use the relevant vector means and standard deviations to sample actions with probabilities $p(a_{1:l} | \mu_{1:l}^n, \sigma_{1:l}^n)$ given by an independent multivariate Gaussian distribution:

$$a_{1:l}^n \sim N(\mu_{1:l}^n, (\sigma_{1:l}^n)^2). \quad (17)$$

The TD-targets in *SAC* are computed by augmenting the traditional backups with a discounted term evaluating the policy's entropy at the next state: $r + \gamma E_{a' \sim \pi(\cdot | s')}[Q_{\phi'}^\pi(s', a') - \pi(a'|s')]$. Similarly, we also augment the routine Q-function TD-targets from Equation 12 with a discounted term, evaluating the routine decoder's entropy for the policy routine at the relevant subsequent state, $n_l' = \pi(s_l')$. Particularly, the TD-targets for a routine recovered from an action sequence of length l in the replay buffer are:

$$y_l^{SAC} = \sum_{j=1}^l \left(\gamma^{(j-1)} r_j \right) + \gamma^l \, \mathbb{E}_{a_{1:l'}'} [Q_{\phi'}^\pi(s_l', E_{\omega_1}(a_{1:l'}')) +$$
$$- log(p(a_{1:l'}' | n_l'))]. \quad (18)$$

Policy optimization in *SAC* is also similarly performed by augmenting the Q-function output with a term incentivizing the policy's entropy: $\mathbb{E}_{s \sim B, a \sim \pi_\theta(\cdot | s)} \left[Q_\phi^{\pi_\theta}(s, a) - \pi_\theta(a|s) \right]$. Hence, we again convert this term to incentivize the routine decoder's entropy in decoding the policy routine at sampled states $n = \pi(s)$. This leads to a new policy optimization objective:

$$J_\pi^{SAC} = \mathbb{E}_{B, a_{1:l}} \left[Q_\phi^{\pi_\theta}(s, E_{\omega_1}(a_{1:l})) - log(p(a_{1:l} | n))) \right]. \quad (19)$$

Unlike our *TD3* integration, we do not explore adding stochasticity at the routine level from the policy's output. Further details regarding our integration with *TD3* and *SAC* are provided in Section B of the Appendix.

6. Experimental Results

In this section, we provide an evaluation of the proposed routine framework utilizing the DeepMind Control Suite (Tassa
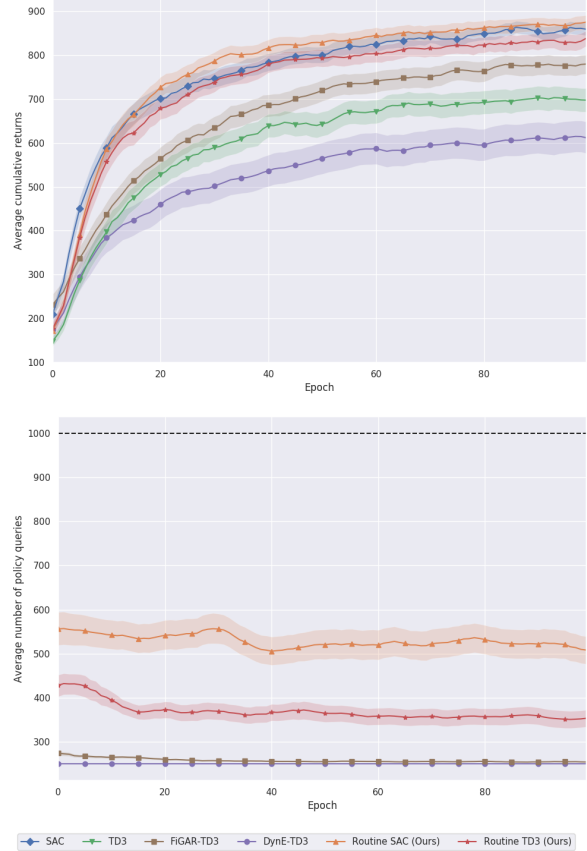

Figure 3. Average cumulative returns (Top) and number of policy queries (Bottom) across all fourteen evaluated environments, as a function of the epoch number. We repeat each experiment ten times and show both mean and standard deviations across all runs.

et al., 2018). We compare the performance of different algorithms as a function of the total number of observations experienced, with each epoch corresponding to experiencing $10,000$ observations. We repeat each experiment ten times for the main *Performance Analysis* and five times for the other ablations, providing both the mean and standard deviation across the runs. We record five evaluation runs at the end of each epoch and collect both the cumulative returns and the total number of times that routine-based agents need to query their policy to complete an episode. Having to query the policy a lower number of times provides concrete improvements to the agent's computational efficiency, having implications in speeding up the process of experience collection and facilitating actuation.

6.1. Performance Analysis

Firstly, to understand the effectiveness of the routine framework, we evaluate the novel routine off-policy algorithms specified in Section 5 in fourteen distinct tasks. For these experiments, we fix the maximum routine length to $L = 4$. We
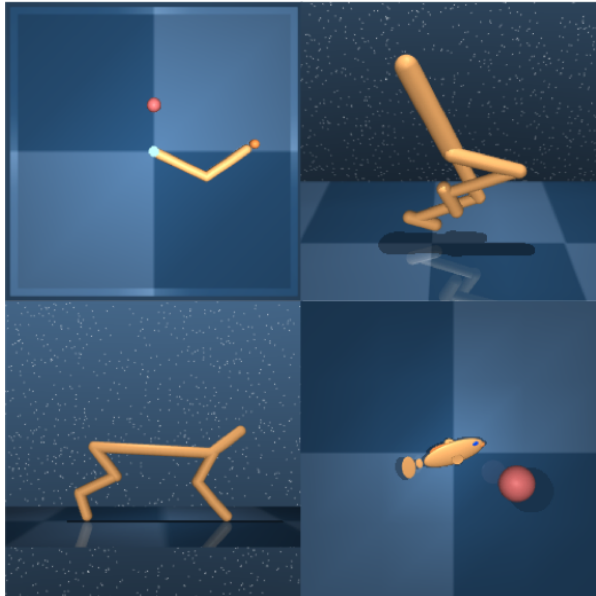
Figure 4. Visualization of the environments used in the expressivity analysis, namely, *Reacher*, *Walker*, *Cheetah* and *Fish*.

Table 1. Average cumulative returns and number of policy queries (in brackets) over the last 10 epochs of training.

	Reacher hard	Walker walk	Cheetah run	Fish swim
TD3	962 ± 10 (1000)	870 ± 95 (1000)	862 ± 17 (1000)	142 ± 81 (1000)
Routine TD3 ($L = 2$)	778 ± 379 (994)	927 ± 82 (855)	817 ± 134 (619)	444 ± 52 (621)
Routine TD3 ($L = 4$)	$\mathbf{975 \pm 4}$ (471)	$\mathbf{975 \pm 2}$ (704)	$\mathbf{880 \pm 23}$ (338)	745 ± 38 (315)
Routine TD3 ($L = 8$)	967 ± 14 (336)	969 ± 4 (194)	856 ± 30 (243)	797 ± 30 (189)
Routine TD3 ($L = 16$)	963 ± 14 (357)	924 ± 80 (203)	725 ± 78 (195)	$\mathbf{807 \pm 9}$ (177)
SAC	956 ± 13 (1000)	968 ± 4 (1000)	844 ± 31 (1000)	530 ± 33 (1000)
Routine SAC ($L = 2$)	972 ± 6 (809)	927 ± 82 (994)	829 ± 27 (756)	281 ± 65 (779)
Routine SAC ($L = 4$)	969 ± 10 (315)	$\mathbf{969 \pm 1}$ (495)	846 ± 15 (529)	580 ± 140 (765)
Routine SAC ($L = 8$)	$\mathbf{973 \pm 3}$ (322)	969 ± 4 (296)	$\mathbf{866 \pm 14}$ (443)	$\mathbf{614 \pm 185}$ (307)
Routine SAC ($L = 16$)	954 ± 30 (235)	892 ± 73 (231)	793 ± 49 (390)	527 ± 282 (167)

provide all other hyper-parameters used by our algorithms in Section C of the Appendix. We compare their performance against the state-of-the-art baseline off-policy algorithms from which they are derived, and two algorithms from recent works that extend *TD3* through temporal abstractions:

- *FiGAR TD3* - Modified version of the *FiGAR DDPG* algorithm described by Sharma et al. (2017), tuned to obtain improved performance in the DeepMind Control Suite environments. This algorithm parameterizes a policy outputting an action and a probability distribution over action repetitions. We provide details of our implementation in Section D of the Appendix.

- *DynE TD3* - Algorithm proposed by Whitney et al. (2019) that builds action-sequence embeddings using state information. We adapted the authors' original implementation.

In Figure 3 we provide visualizations for both the average returns and the average number of policy evaluations per episode across all fourteen tasks as a function of the number of epochs. Note that for each task in the DeepMind Control Suite the obtainable cumulative returns are always in the range $[0, 1000]$. From the obtained results, we show that integrating the routine framework improves both examined state-of-the-art algorithms. Particularly, from the integration of routines with *TD3* we observe substantial gains both in terms of final performance and convergence speed. The gains in final performance appear to be more significant in the harder exploration environments, indicating that the routine framework and the two levels noise injection described in Section 5.2 concretely help exploration. While *SAC* already achieves high final performance on the Deep-Mind Control Suite, incorporating routines still provides improvements in terms of convergence speed and stability, showing the effectiveness of our framework.

Our *Routine TD3* algorithm also convincingly outperforms both *FiGAR TD3* and *DynE TD3*, especially in some of the more complex locomotion tasks in the *Cheetah* and *Walker* environments. We believe one of the reasons for this gap is that *FiGAR* and *DynE* introduce temporal abstractions with representations that are either fixed or learned from some particular heuristic. Hence, while these representations might work well for particular tasks, they struggle to be generally effective for a large suite of diverse tasks. In contrast, routines build a representation space optimized end-to-end to facilitate off-policy learning for any specific underlying task, which we believe is one of the main reasons for their effectiveness on diverse problems. Additionally, both *FiGAR* and *DynE* cannot model high-level behavior over a full set of arbitrary action sequences, lacking the expressiveness of our framework.

Both algorithms that make use of the routine framework required around half the number of policy evaluations per episode, considerably improving efficiency from their baselines. Interestingly, we observe the routine version of *TD3* to be marginally more efficient. Perhaps, this could be motivated by the lack of routine level noise applied to the routine version of *SAC*, causing it to under-explore its routine space. Both *FiGAR TD3* and *DynE TD3* recover policies acting as infrequently as possible. However, this efficiency is partially symptomatic of their inability to recognize when to act with greater precision, being one of the reason for their lower performance in the harder environments. We provide the full set of results obtained per environment, including the relative performance curves in Section E of the Appendix.

Having no incentives to output 'long' routines, we would expect our optimizations to eventually converge to a policy that would exploit the finest level of control available to maximize performance. One way to counteract this would

be to add small reasoning penalties to the reward each time the agent's policy is queried, making the agent aware of computational costs. While we do not make use of any such penalties, interestingly, our results show that the recovered routine policies do not converge to selecting exclusively 'short' routines. Instead, they appear to converge to selecting mostly longer routines, with some shorter routines being performed in the harder environments in scenarios where the agent is physically unstable. We hypothesize this occurs because in the majority of states there are very small performance gains in reasoning at every step. At the same time, faster reward propagation and target policy smoothing can already provide some implicit stimulus to the Q values of 'longer' routines in the stochastic optimization context of TD-learning.

6.2. Expressivity Analysis

Increasing the maximum routine length parameter L should provide the agents with a greater potential for deployment efficiency and should make reward propagation occur at an even faster rate. On the other hand, we hypothesize that it would make the Q-function harder to optimize since it would need to learn accurate Q values for a routine space that represents a larger number of action sequences in S^A. Hence, we empirically examine the effects of increasing the maximum routine length parameter L on both performance and deployment efficiency. We evaluate both novel routine algorithms with a maximum routine length parameter L of 2, 4, 8, and 16 on the four environments shown in Figure 4. Thus, we compare their performance and efficiency in contrast to the relative state-of-the-art baselines.

We summarize the results in Table 1, where we report the final cumulative returns and number of policy queries in the last ten epochs of training. We show the full performance curves in Section E of the Appendix. Particularly, the highest final cumulative returns in all but one environment are obtained for the maximum routine lengths of 4 and 8. Moreover, the efficiency of the recovered agents appears to visibly improve by increasing L, requiring less than a fifth of policy queries in some experiments with $L = 8$ and $L = 16$. However, by increasing the maximum routine length to 16 we start observing a tradeoff where we obtain further efficiency benefits but with some performance costs. Interestingly, we also observe that for some environments, setting $L = 2$ leads to slightly worse performance both with respect to the other routine algorithms but also with respect to the non-routine baselines. We believe this is because, in such environments, the expressivity added by a routine framework that can represent only action sequences of length 2 is too limited to outweigh the downsides that the increased model complexity introduces.

7. Discussion and Future Work

We proposed the routine framework, a novel approach to reinforcement learning that strives to learn an effective higher-dimensional 'action space' named the routine space, achieving improved performance and deployment efficiency. Each routine is trained end-to-end to represent a set of length-agnostic action sequences with equivalent effects for tackling a target task. Our framework is fully compatible with off-policy reinforcement learning and we demonstrate its effectiveness by successfully applying it to two state-of-the-art algorithms for continuous control. Empirically, we show that the novel resulting algorithms consistently achieve better final cumulative returns and convergence speed than their relative baselines and two recent alternative frameworks for reasoning with temporal abstractions. Moreover, the recovered agents can avoid reasoning at every step, requiring only a fraction of the policy queries to complete an episode, yielding non-trivial benefits for computational efficiency and real-world deployment. One natural extension for future works would be to apply the routine framework for algorithms working in discrete environments, to test the ubiquity of its applicability and effectiveness. Additionally, it would be interesting to analyze the structure and transferability of the routine spaces recovered in different tasks as a way to assess similarity and perform few-shot learning.

Acknowledgements

References

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Bahl, S., Mukadam, M., Gupta, A., and Pathak, D. Neural dynamic policies for end-to-end sensorimotor learning. *arXiv preprint arXiv:2012.02788*, 2020.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Tb, D., Muldal, A., Heess, N., and Lillicrap, T. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Biedenkapp, A., Rajan, R., Hutter, F., and Lindauer, M. Towards temporl: Learning when to act. In *Workshop on Inductive Biases, Invariances and Generalization in Reinforcement Learning (BIG@ ICML'20)*, 2020.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. S. Learning action representations for reinforcement learning. *arXiv preprint arXiv:1902.00183*, 2019.

Co-Reyes, J. D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*, 2018.

Dabney, W., Ostrovski, G., and Barreto, A. Temporally-extended {\epsilon}-greedy exploration. *arXiv preprint arXiv:2006.01782*, 2020.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. In Hanson, S., Cowan, J., and Giles, C. (eds.), *Advances in Neural Information Processing Systems*, volume 5, pp. 271–278. Morgan-Kaufmann, 1993. URL https://proceedings.neurips.cc/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Dulac-Arnold, G., Mankowitz, D., and Hester, T. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29: 3675–3683, 2016.

Lakshminarayanan, A., Sharma, S., and Ravindran, B. Dynamic action repetition for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.

Lee, J., Lee, B.-J., and Kim, K.-E. Reinforcement learning for control with multiple frequencies. In *Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS 2020)*. Neural information processing systems foundation, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. Benchmarking reinforcement learning algorithms on real-world robots. *arXiv preprint arXiv:1809.07731*, 2018.

Metelli, A. M., Mazzolini, F., Bisi, L., Sabbioni, L., and Restelli, M. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning*, pp. 6862–6873. PMLR, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in neural information processing systems*, pp. 3303–3313, 2018.

Neunert, M., Abdolmaleki, A., Wulfmeier, M., Lampe, T., Springenberg, T., Hafner, R., Romano, F., Buchli, J., Heess, N., and Riedmiller, M. Continuous-discrete reinforcement learning for hybrid control in robotics. In *Conference on Robot Learning*, pp. 735–751. PMLR, 2020.

Precup, D. Temporal abstraction in reinforcement learning. 2001.

Precup, D., Sutton, R. S., and Singh, S. P. Planning with closed-loop macro actions. In *Working notes of the 1997 AAAI Fall Symposium on Model-directed Autonomous Systems*, pp. 70–76, 1997.

Schoknecht, R. and Riedmiller, M. Speeding-up reinforcement learning with multi-step actions. In *International Conference on Artificial Neural Networks*, pp. 813–818. Springer, 2002.

Schoknecht, R. and Riedmiller, M. Reinforcement learning on explicitly specified time scales. *Neural Computing & Applications*, 12(2):61–80, 2003.

Sharma, S., Lakshminarayanan, A. S., and Ravindran, B. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. 2014.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Sweatt, J. D. *Mechanisms of memory*. Academic Press, 2009.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tennenholtz, G. and Mannor, S. The natural language of actions. *arXiv preprint arXiv:1902.01119*, 2019.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., Agapiou, J., et al. Strategic attentive writer for learning macro-actions. *Advances in neural information processing systems*, 29:3486–3494, 2016.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Whitney, W., Agarwal, R., Cho, K., and Gupta, A. Dynamics-aware embeddings. *arXiv preprint arXiv:1908.09357*, 2019.

Ziebart, B. D. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.