TempoRL: Learning When to Act

André Biedenkapp [1] **Raghu Rajan** [1] **Frank Hutter** [1 2] **Marius Lindauer** [3]

Abstract

Reinforcement learning is a powerful approach to learn behaviour through interactions with an environment. However, behaviours are usually learned in a purely reactive fashion, where an appropriate action is selected based on an observation. In this form, it is challenging to learn *when* it is necessary to execute new decisions. This makes learning inefficient, especially in environments that need various degrees of fine and coarse control. To address this, we propose a proactive setting in which the agent not only selects an action in a state but also for how long to commit to that action. Our TempoRL approach introduces skip connections between states and learns a skip-policy for repeating the same action along these skips. We demonstrate the effectiveness of TempoRL on a variety of traditional and deep RL environments, showing that our approach is capable of learning successful policies up to an order of magnitude faster than vanilla $\mathcal{Q}$-learning.

1. Introduction

Although reinforcement learning (RL) has celebrated many successes in the recent years (see e.g., Mnih et al., 2015; Lillicrap et al., 2016; Baker et al., 2020), in its classical form it is limited to learning policies in a mostly reactive fashion, i.e., observe a state and react to that state with an action. Guided by the reward signal, policies that are learned in such a way can decide which action is expected to yield maximal long-term rewards. However, these policies generally do not learn *when a new decision has to be made*. A more proactive way of learning, in which agents proactively commit to playing an action for multiple steps could further improve RL by (i) potentially providing better exploration compared to common one-step exploration; (ii) faster learning as proactive policies provide a form of temporal abstraction by requiring fewer decisions; (iii) explainability as learned agents can indicate when they expect new decisions are required.

Temporal abstractions are a common way to simplify learning of policies with potentially long action sequences. Typically, the temporal abstraction is learned on the highest level of a hierarchy and the required behaviour on a lower level (see e.g. Sutton et al., 1999; Eysenbach et al., 2019). For example, on the highest level a *goal policy* learns which states are necessarily visited and on the lower level the *behaviour* to reach goals is learned. Spacing goals far apart still requires to learn complex behaviour policies whereas a narrow goal spacing requires to learn complex goal policies. Another form of temporal abstraction is to use actions that work at different time-scales (Precup et al., 1998). Take for example an agent that is tasked with moving an object. On the highest level the agent would follow a policy with abstract actions, such as *pick-up object*, *move object*, *put-down object*, whereas on the lower level actions could directly control actuators to perform the abstract actions.

Such hierarchical approaches are still reactive, but instead of reacting to an observation on only one level, reactions are learned on multiple levels. Though these approaches might allow us to learn *which* states are necessarily traversed in the environment, they do not enable us to learn *when* a new decision has to be made on the behaviour level.

In this work, we propose an alternative approach: a proactive view on learning policies that allows us to jointly learn a behaviour and how long to carry out that behaviour. To this end, we re-examine the relationship between agent and environment, and the dependency on time. This allows us to introduce *skip connections* for an environment. These skip connections do not change the optimal policy or state-action-values but allow us to propagate information much faster. We demonstrate the effectiveness of our method, which we dub TEMPORL with tabular and deep function approximation on a variety of environments with discrete and continuous action spaces. Our contributions are:

1. We propose a proactive alternative to classical RL.
2. We introduce skip-connections for MDPs by playing an action for several consecutive states, which leads to faster propagation of information about future rewards.

[1]Department of Computer Science, University of Freiburg, Germany [2]BCAI, Renningen, Germany [3]Information Processing Institute (tnt), Leibniz University Hannover, Germany. Correspondence to: André Biedenkapp <biedenka@cs.uni-freiburg.de>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

3. We propose a mechanism based on a hierarchy for learning when to make new decisions through the use of skip-connections.
4. On classical and deep RL benchmarks, we show that TEMPORL outperforms plain DQN, DAR and FiGAR both in terms of learning speed and sometimes even by converging to better policies.

2. Related Work

A common framework for temporal abstraction in RL is the options framework (Precup et al., 1998; Sutton et al., 1999; Stolle & Precup, 2002; Bacon et al., 2017; Harutyunyan et al., 2018; Mankowitz et al., 2018; Khetarpal & Precup, 2019). Options are triples $\langle \mathcal{I}, \pi, \beta \rangle$ where $\mathcal{I}$ is the set of admissible states that defines in which states the option can be played; π is the policy the option follows when it is played; and β is a random variable that determines when an option is terminated. In contrast to our work, options require a lot of prior knowledge about the environment to determine the set of admissible states, as well as the option policies themselves. However, Chaganty et al. (2012) proposed to learn options based on observed connectedness of states. Similarly, SoRB (Eysenbach et al., 2019) uses data from the replay buffer to build a connectedness graph, which allows to query sub-goals on long trajectories. Further work on discovering options paid attention to the termination criterion, learning persistent options (Harb et al., 2018) and meaningful termination criteria (Vezhnevets et al., 2016; Harutyunyan et al., 2019).

Similarly, in AI planning macro actions provide temporal abstractions. However, macro actions are not always applicable as some actions can be locked. Chrpa & Vallati (2019) propose to learn when macro actions become available again, allowing them to identify non-trivial activities. For various problem domains of AI planning, varieties of useful macro actions are known and selecting which macro actions to consider is not trivial. Vallati et al. (2019) propose a macro action selection mechanism that selects which macro actions should be considered for new problems. Further, Nasiriany et al. (2019) show that goal-conditioned policies learned with RL can be incorporated into planning. With complex state observations goal states are difficult to define.

An important element in DQN's success in tackling various Atari games (Mnih et al., 2015) is due to the use of *frame skipping* (Bellemare et al., 2013). Thereby the agent skips over a few states, always playing the same action, before making a new decision. Without the use of frame skipping, the change between successive observations is small and would have required more observations to learn the same policy. Tuning the *skip-size* can additionally improve performance (Braylan et al., 2015; Khan et al., 2019). A similar line of research focuses on learning persistent policies which always act after a static, fixed time-step for one-dimensional (Metelli et al., 2020) and multi-dimensional actions (Lee et al., 2020). However, static skip-sizes might not be ideal. Dabney et al. (2020) demonstrated that temporally extended ϵ-greedy schedules improve exploration and thereby performance in sparse-reward environments while performing close to vanilla ϵ-greedy exploration on dense-reward environments.

Different techniques have been proposed to handle continuous time environments (Doya, 2000; Tiganj et al., 2017). Recently, Huang et al. (2019) proposed to use *Markov Jump Processes* (MJPs). MJPs are designed to study optimal control in MDPs where observations have an associated cost. The goal then is to balance the costs of observations and actions to act in an optimal manner with respect to total cost. Their analysis demonstrated that frequent observations are necessary in regions where an optimal action might change rapidly, while in areas of infrequent change, fewer observations are sufficient. In contrast to ours, this formalism strictly prohibits observations of the skipped transitions to save observation costs and thus losing a lot of information, which otherwise could be used to learn how to act while simultaneously learning when new decisions are required.

Schoknecht & Riedmiller (2002; 2003) demonstrated that learning with multi-step actions can significantly speed up RL. Relatedly, Lakshminarayanan et al. (2017) proposed *DAR*, a $\mathcal{Q}$-network with multiple output heads per action to handle different repetition lengths, drastically increasing the action space but improving learning. In contrast to that, Sharma et al. (2017) proposed *FiGAR*, a framework that jointly learns an action policy and a second repetition policy that decides how often to repeat an action. Crucially, its repetition policy is not conditioned on the chosen action resulting in independent repetition and behaviour actions. The polices are learned together through a joint loss. Thus, counter to our work, the repetition policy only learns which repetition length works well on average for all actions. Further, FiGAR requires modification to the training method of a base agent to accommodate the repetition policy. When evaluating our method in the context of DQN, we compare against DAR and in the context of DDPG against FiGAR as they were originally developed and evaluated on these agent types. The appendix, code and experiment results are available at `github.com/automl/TempoRL`.

3. TempoRL

We begin this section by introducing skip connections into MDPs, propagating information about expected future rewards faster. We then introduce a novel learning mechanism that makes use of a hierarchy to learn a policy that is capable of not only learning which action to take, but also *when* a new action has to be chosen.

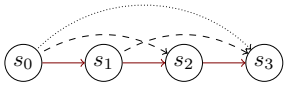


Figure 1. Example transitions with skip of length three (drawn with $\cdots$). At the same time we can also observe shorter skips of length two (- - -) and normal steps, i.e. skips of length one ($-$).

3.1. Temporal Abstraction through Skip MDPs

It is possible to make use of contextual information in MDPs (Hallak et al., 2015; Modi et al., 2018; Biedenkapp et al., 2020). To this end, we contextualize an existing MDP $\mathcal{M}$ to allow for skip connections as $\mathcal{M}_\mathcal{C} \coloneqq \{\mathcal{M}_c\}_{c\in\mathcal{C}}$ with $\mathcal{M}_c \coloneqq \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_c, \mathcal{R}_c \rangle$. Akin to options, a skip-connection c is a triple $\langle s, a, j \rangle$, where s is the starting state for a skip transition (and not a set of states as in the options framework); a is the action that is executed when skipping through the MDP; and j is the skip-length, where $a \in \mathcal{A}$, $s \in \mathcal{S}$ and $j \in \mathcal{J} = \{1, \ldots, J\}$. This context to the MDP induces different MDPs with shared state and action spaces ($\mathcal{S}$, $\mathcal{A}$), but different transitions $\mathcal{P}_c$ and reward functions $\mathcal{R}_c$ to account for the introduced skips.

In practice however, the transition and reward functions are unknown and do not allow to easily insert skips. Nevertheless, as we make use of action repetition, we can simulate a skip connection. A skip connects two states s and s' iff state s' is reachable from state s by repeating action a j-times. This gives us the following skip transition function:

$$\mathcal{P}_c(s, a, s') = \begin{cases} \prod_{k=0}^{j-1} \mathcal{P}^a_{s_k s_{k+1}} & \text{if reachable} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

with s_k and s_{k+1} the states traversed by playing action a for the kth time, and with $s_0 = s$ and $s_j = s'$. This change in the transition function is reflected accordingly in the reward:

$$\mathcal{R}_c(s, a, s') = \begin{cases} \sum_{k=0}^{j-1} \gamma^k \mathcal{R}^a_{s_k s_{k+1}} & \text{if reachable} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Thus, for skips of length 1 we recover the original transition function $\mathcal{P}_{\langle s,a,1\rangle}(s, a, s') = \mathcal{P}^a_{ss'}$ as well as the original reward function $\mathcal{R}_{\langle s,a,1\rangle}(s, a, s') = \mathcal{R}^a_{ss'}$. The goal with skip-MDPs is to find an optimal skip policy $\pi_J \colon \mathcal{S} \times \mathcal{A} \mapsto \mathcal{J}$, i.e., a policy that takes a state and a behaviour action as input and maps to a skip value that maximally reduces the total required number of decisions to reach the optimal reward. Thus, similar to skip-connections in neural networks, skip MDPs allow us to propagate information about future rewards much more quickly and enables us to determine *when* it becomes beneficial to switch actions.

3.2. Learning When to Make Decisions

In order to learn using skip connections we need a new mechanism that selects which skip connection to use. In order to facilitate this, we propose using a hierarchy in which a *behaviour* policy determines the action a to be played given the current state s, and a *skip* policy determines how long to commit to this behaviour.

To learn the behaviour, we can make use of classical $\mathcal{Q}$-learning, where the $\mathcal{Q}$-function gives a mapping of expected future rewards when playing action a in state s_t at time t and continuing to follow the behaviour policy π thereafter.

$$\mathcal{Q}^\pi(s, a) \coloneqq \mathbb{E}\left[r_t + \gamma \mathcal{Q}^\pi(s_{t+1}, a_{t+1}) | s = s_t, a\right] \tag{3}$$

To learn to skip, we first have to define a *skip-action space* that determines all possible lengths of skip-connections, e.g., $j \in \{1, 2, \ldots, J\}$. To learn the value of a skip we can make use of n-step $\mathcal{Q}$-learning with the condition that, at each step of the j steps, the action stays the same.

$$\mathcal{Q}^{\pi_J}(s, j|a) \coloneqq \mathbb{E}\left[\sum_{k=0}^{j-1} \gamma^k r_{t+k} + \gamma^j \mathcal{Q}^\pi(s_{t+j}, a_{t+j}) | s = s_t, a, j\right] \tag{4}$$

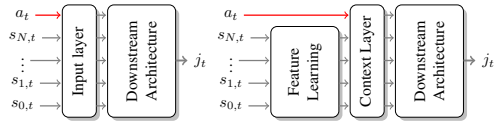
We call this a flat hierarchy since the behaviour and the skip policy have to always make decisions at the same time-step; however, the behaviour policy has to be queried before the skip policy. Once we have determined both the action a and the skip-length j we want to perform, we execute this action for j steps. We can then use standard temporal difference updates to update the behaviour and skip $\mathcal{Q}$-functions with all one-step observations and the overarching skip-observation. Note that the skip $\mathcal{Q}$-function can also be conditioned on continuous actions if the behaviour policy can handle continuous action-spaces.

One interesting observation regarding this learning scheme is that, when playing skip action j, we are able to also observe all smaller skip transitions for all intermediate steps. Figure 1 gives a visual representation. Specifically, we can directly see that, when executing a skip of length j, we can observe and learn from $\frac{j\cdot(j+1)}{2}$ skip-transitions in total. As we observe all intermediate steps, we can use this trajectory of transitions to build a local connectedness graph (similar to Figure 1) from which we can look up all skip-connections. This allows us to efficiently learn the values of multiple skips, i.e. the action value at different time-resolutions. For pseudo-code and more details we refer to Appendix B.

3.3. Learning When to Make Decisions in Deep RL

When using deep function approximation for TEMPORL we have to carefully consider how we parameterize the skip policy. Commonly, in deep RL we do not only deal with featurized states but also with image-based ones. Depending on the state modality we can consider different architectures:

Concatenation The simplest parametrization of our skip-policy assumes that the state of the environment we are



(a) Concat (b) Context

Figure 2. Schematic representations of considered architectures for learning when to make decisions, where a_t is the action coming from a separate behaviour policy.

learning from is featurized, i.e., a state is a vector of individual informative features. In this setting, the skip-policy network can take any architecture deemed appropriate for the environment, where the input is a concatenation of the original state s_t and the chosen behaviour action a_t, i.e., $s'_t = (s_t, a_t)$, see Figure 2a. This allows the skip-policy network to directly learn features that take into account the chosen behaviour action. However, note that this concatenation assumes that the state is already featurized.

Contextualization In deep RL, we often have to learn to act directly from images. In this case, concatenation is not trivially possible. Instead we propose to use the behaviour action as context information further down-stream in the network. Feature learning via convolutions can then progress as normal and the learned high-level features can be concatenated with the action a_t and be used to learn the final skip-value, see Figure 2b.

Shared Weights Concatenation and contextualization learn individual policy networks for the behaviour and skip policies and do not share information between the two. To achieve this we can instead share parts of the networks, e.g., the part of learning higher-level features from images (see Figure 3). This allows us to learn the two policy networks with potentially fewer weights than two completely independently learned networks. In the forward and backward passes, only the shared feature representation with the corresponding output layers are active. Similar to the contextualization, the output layers for the skip-values require the selected action, i.e. the argmax of the action outputs, as additional input.

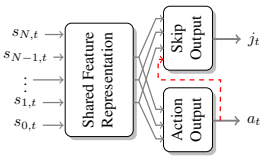


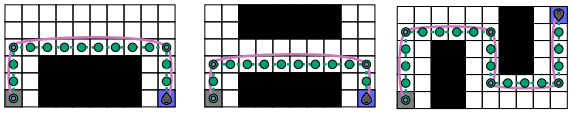
Figure 3. Architecture with shared feature representation for joint learning of when to make a decision and what action to take.

4. Experiments

We evaluated TEMPORL with tabular as well as deep $\mathcal{Q}$-functions. We first give results for the tabular case. All code, the appendix and experiment data including trained policies are available at `github.com/automl/TempoRL`. For details on the used hardware see Appendix C.

4.1. Tabular TempoRL

In this subsection, we describe experiments for a tabular $\mathcal{Q}$-learning implementation that we evaluated on various gridworlds with sparse rewards (see Figure 4). We first evaluate our approach on the *cliff* environment (see Figure 4a) before evaluating the influence of the exploration schedule on both vanilla and TEMPORL $\mathcal{Q}$-learning, which we refer to as $\mathcal{Q}$ and t-$\mathcal{Q}$, respectively.



(a) Cliff (b) Bridge (c) ZigZag

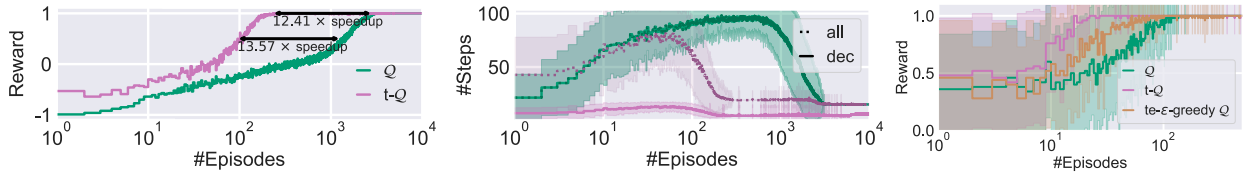
Figure 4. 6×10 Grid Worlds. Agents have to reach a fixed goal state from a fixed start state. Dots represent decision steps of vanilla and TEMPORL $\mathcal{Q}$-learning policies.

Gridworlds All considered environments (see Figure 4) are discrete, deterministic, have sparse rewards and have size 6×10. Falling off a cliff results in a negative reward (-1) and reaching a goal state results in a positive reward $(+1)$. For a more detailed description of the gridworld environments we refer to Appendix D.

For this experiment, we limit our TEMPORL agent to a maximum skip length of $J = 7$; thus, a learned optimal policy requires 4 decision points instead of 3. For evaluations using larger skips we refer to Appendix E. Note that increasing the skip-length improves TEMPORL up to some point, at which it has too many irrelevant skip-actions at its disposal which slightly decreases the performance. We compare the learning speed, in terms of training policies, of our approach to a vanilla $\mathcal{Q}$-learning agent. Both methods are trained for 10 000 episodes using the same ϵ-greedy strategy, where ϵ is linearly decayed from 1.0 to 0.0 over all episodes.

Figure 5a depicts the evaluation performance of both methods. TEMPORL is $13.6\times$ faster than its vanilla counterpart to reach a reward of 0.5, and $12.4\times$ faster to reach a reward of 1.0 (i.e., always reach the goal). Figure 5b shows the number of required steps in the environment, as well as the number of decision steps. TEMPORL is capable of finding a policy that reaches the goal much faster than vanilla $\mathcal{Q}$-learning while requiring far fewer decision steps. Furthermore, TEMPORL recovers the optimal policy quicker than vanilla $\mathcal{Q}$-learning. Lastly we can observe that after having trained for $\approx 6\,000$ episodes, TEMPORL starts to increase the number of decision points. This can be attributed to skip values of an action having converged to the same value and our implementation selecting a random skip as tie-breaker.

Table 1 summarizes the results on all environments in terms of normalized area under the reward curve and number of decisions for three different ϵ-greedy schedules. A reward AUC value closer to 1.0 indicates that the agent was capable



(a) Cliff – Reward per Episode (b) Cliff – Steps per Episode (c) Temporal Exploration

Figure 5. Evaluation performance of tabular $\mathcal{Q}$-learning agents over 100 random seeds. (a) & (b): The agents were trained with a linearly-decaying ϵ-greedy policy on the cliff environment. (a) Achieved reward. (b) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies. (c) Comparison to temporally extended ϵ-greedy exploration (te-ϵ-greedy $\mathcal{Q}$ in plot) on a 23×23 Gridworld (Dabney et al., 2020). t-$\mathcal{Q}$ is our proposed TEMPORL agent. The lines/shaded area represent the mean/standard deviation.

Table 1. Normalized AUC for reward and average number of decision steps. Both agents are trained with the same ϵ schedule.

(a) linearly decaying ϵ-schedule

	Cliff		Bridge		ZigZag	
	$\mathcal{Q}$	t-$\mathcal{Q}$	$\mathcal{Q}$	t-$\mathcal{Q}$	$\mathcal{Q}$	t-$\mathcal{Q}$
$\text{Reward}_{\text{AUC}}$	0.92	**0.99**	0.75	**0.97**	0.57	**0.92**
Decisions	27.9	**5.2**	49.5	**5.0**	83.6	**7.9**

(b) logarithmically decaying ϵ-schedule

$\text{Reward}_{\text{AUC}}$	0.96	**0.99**	0.94	**0.98**	0.90	**0.96**
Decisions	21.7	**4.9**	21.4	**5.3**	35.6	**6.9**

(c) constant $\epsilon = 0.1$

$\text{Reward}_{\text{AUC}}$	**0.99**	**0.99**	0.98	**0.99**	0.95	**0.99**
Decisions	17.1	**5.1**	14.7	**5.2**	27.6	**7.1**

of learning to reach the goal quickly. A lower number of decisions is better as fewer decisions were required to reach the goal, making a policy easier to learn. In view of both metrics, TEMPORL readily outperforms the vanilla agent, learning much faster and requiring far fewer decisions.

Sensitivity to Exploration As the used exploration mechanism can have a dramatic impact on agent performance we evaluated the agents for three commonly used ϵ-greedy exploration schedules. In the cases of linearly and logarithmically decaying schedules, we decay ϵ over all 10 000 training episodes, starting from 1.0 and decaying it to 0 or 10^{-5}, respectively. In the constant case, we set $\epsilon = 0.1$.

As shown in Table 1, maybe not surprisingly, too much (linear) and too little (log) exploration are both detrimental to the agent's performance. However, TEMPORL performs quite robustly even using suboptimal exploration strategies. TEMPORL outperforms its vanilla counterpart in all cases, showing the effectiveness of our proposed method.

Guiding Exploration To demonstrate TEMPORL not only benefits through better exploration but also learning *when* to act, we use the 23×23 Gridworld and agent hyperparameters as introduced by Dabney et al. (2020).

An agent starts in the top center and has to find a goal further down and to the left only getting a reward for reaching the goal within 1000 steps. Temporally-extended exploration (te-ϵ-greedy $\mathcal{Q}$-learning; Dabney et al., 2020) is able to cover a space much better than 1-step exploration. However, it falls short in guiding the agent back to high reward areas. TEMPORL enables an agent to quickly find a successful policy that reach a goal while exploring around such a policy. Figure 5c shows TEMPORL reliably reaches the goal after ≈ 30 episodes. An agent using temporally-extended epsilon greedy exploration does not reliably reach the goal in this time-frame and on average requires twice as many steps.

4.2. Deep TempoRL

In this section, we describe experiments for agents using deep function approximation implemented with PyTorch (Paszke et al., 2019) in version 1.4.0. We begin with experiments on featurized environments before evaluating on environments with image states. We evaluate TEMPORL for DQN with different architectures for the skip $\mathcal{Q}$-function. We compare against *dynamic action repetition* (DAR; Lakshminarayanan et al., 2017) for the DQN experiments and against *Fine grained action repetition* (FiGAR Sharma et al., 2017) for experiments with DDPG.[1]

4.2.1. ADVERSARIAL ENVIRONMENT – DDPG

Setup We chose to first evaluate on OpenAI gyms (Brockman et al., 2016) Pendulum-v0 as it is an adversarial setting where high action repetition is nearly guaranteed to overshoot the balancing point. Thus, agents using action repetition that make mistakes during training will have to spend additional time learning *when* it is necessary to be reactive; a challenge vanilla agents are not faced with. We trained all DDPG agents (Lillicrap et al., 2016) for a total of 3×10^4 training steps and evaluated the agents every 250 training steps. The first 10^3 steps follow a uniform random policy to generate the initial experience. We used Adam (Kingma & Ba, 2015) with PyTorchs default settings.

[1]Neither DAR, nor FiGAR are publicly available and thus we used our own reimplementation available at `github.com/automl/TempoRL`.

Table 2. Average normalized reward AUC for DDPG agents on Pendulum-v0. t-DDPG and FiGAR are evaluated over different maximal skip-lengths for 15 seeds. Corresponding learning curves are given in Appendix F

	t-DDPG							FiGAR						
DDPG	2	4	6	8	10	14	20	2	4	6	8	10	14	20
0.92	0.89	0.89	*0.90*	0.89	0.89	0.89	0.88	*0.76*	0.57	*0.39*	0.31	0.28	0.25	0.24

Agents All actor and critic networks of all DDPG agents consist of two hidden layers with 400 and 300 hidden units respectively. Following (Sharma et al., 2017), FiGAR introduces a second actor network that shares the input layer with the original actor network. The output layer is a softmax layer with J outputs, representing the probability of repeating the action for $j \in \{1, \dots, J\}$ time-steps. Both actor outputs are jointly input to the critic and gradients are directly propagated from the critic through both actors. TEMPORL DDPG (which we refer to as t-DDPG in the following) uses the concatenation architecture which takes the state with the action output of the DDPG actor as input and makes use of the critic's $\mathcal{Q}$-function when learning the skip $\mathcal{Q}$-function. We evaluate t-DDPG and FiGAR on a grid of maximal skip lengths of $\{2, 4, 6, \dots, 20\}$. See Appendix F for implementation details and all used hyperparameters.

Pendulum Table 2 confirms that agents using action repetition indeed are slower in learning successful policies, as reflected by the normalized reward AUC. As FiGAR does not directly inform the skip policy about the chosen repetition value or vica versa, the agent tends to struggle quite a lot in this environment already with only two possible skip-values and is not capable of handling larger maximal skip values. In contrast to that, t-DDPG only slightly lags behind vanilla DDPG and readily adapts to larger skip lengths, by quickly learning to ignore irrelevant skip-values. Further, due to making use of n-step learning, t-DDPG starts out very conservative as large skip values appear to lead to larger negative rewards in the beginning. With more experience however, t-DDPG learns *when* switching between actions becomes advantageous, thereby approximately halving the required decisions (see Appendix F).

4.2.2. FEATURZIED ENVIRONMENTS – DQN

Setup We trained all agents for a total of 10^6 training steps using a constant ϵ-greedy exploration schedule with ϵ set to 0.1. We evaluated all agents every 200 training steps. We used the Adam with a learning rate of 10^{-3} and default parameters as given in PyTorch v1.4.0. For increased learning stability, we implemented all agents using double deep $\mathcal{Q}$ networks (van Hasselt et al., 2016). All agents used a replay buffer with size 10^6 and a discount factor γ of 0.99. The TEMPORL agents used an additional replay buffer of size 10^6 to store observed skip-transitions. We used the MountainCar-v0 and LunarLander-v2 environments. See Appendix G for a detailed description of the environments.

Agents The basic DQN architecture consists of 3 layers, each with 50 hidden units and *ReLu* activation functions. The DAR baseline used the same architecture as the DQN agent but duplicated the output heads twice, each of which is associated with specific repetition values allowing for fine and coarse control. We evaluated possible coarse control values on the grid $\{2, 4, 6, 8, 10\}$, keeping the fine-control value fixed to 1 to allow for actions at every-time step.

For TEMPORL agents not using weight sharing we used the same DQN architecture for both $\mathcal{Q}$-functions. The concatenation architecture used an additional input unit whereas the context architecture added the behaviour action as context at the third layer after using 10 additional hidden units to process the behaviour action. An agent using a weight-sharing architecture shared the first two layers of the DQN architecture and used the third layer of the DQN architecture to compute the behaviour $\mathcal{Q}$-values. The skip-output used 10 hidden units to process the behaviour action and processed this output together with the hidden state of the 2nd layer in a 3rd layer with 60 hidden units. We refer to a DQN using TEMPORL as t-DQN in the following.

Influence of the Skip-Architecture We begin by evaluating the influence of architecture choice on our t-DQN on both environments, before giving a more in-depth analysis on the learning behaviour in the individual environments. To this end, we report the normalized reward AUC for all three proposed architectures and different maximal skip-lengths, see Table 3. Both the concat and context architectures behave similarly on both environments, which is to be expected as they differ very little in setup. Both architectures have an increase in AUC before reaching the best maximal skip-length for the respective environment. The shared architecture, mostly conceptualized for image-based environments, however shows more drastic reactions to choice of J, leading to the best result in the first and to the worst result in the other environment.

MountainCar Tables 3a & 4a depict the performance of the agents for different maximal skip lengths and Figure 6a shows the learning curves of the best TEMPORL architecture as well as the best found DAR agent. On MountainCar the DQN baseline struggles in learning a successful policy, resulting in a small AUC of 0.50 compared to the best result of t-DQN of 0.64. Furthermore, a well tuned DAR baseline,

Table 3. Average normalized reward AUC for different TEMPORL architectures and maximal skip-lengths over 50 seeds. All agents are trained with the same ϵ schedule. Bold faced values give the overall best AUC and cursive values the best per architecture.

(a) MountainCar-v0

Max Skip	2	4	6	8	10
concat	0.469	0.523	0.602	0.626	*0.630*
context	0.429	0.540	0.601	0.608	*0.620*
shared	0.440	0.464	0.592	0.561	***0.644***

(b) LunarLander-v2

Max Skip	2	4	6	8	10
concat	0.855	***0.878***	0.868	0.862	0.830
context	0.858	*0.876*	0.871	0.859	0.837
shared	*0.851*	0.837	0.803	0.769	0.696

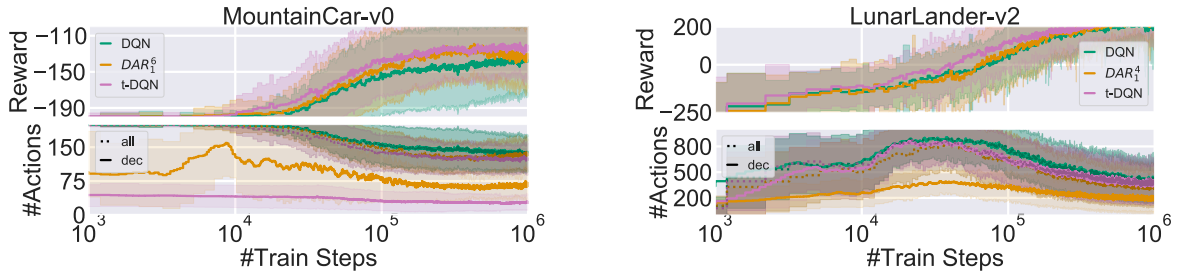


Figure 6. Evaluation performance of deep $\mathcal{Q}$-learning agents on MountainCar-v0 and LunarLander-v2. Solid lines give the mean and the shaded area the standard deviation over 50 random seeds. The sub- and superscripts of DAR give the best found fine and coarse repetition values respectively. t-DQN is our proposed method using the best architecture as reported in Table 3. (top) Achieved rewards. (bottom) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies.

Table 4. Average normalized reward AUC for maximal skip-length of 10 for MountainCar-v0 and 4 for LunarLander-v2 over 50 seeds. All agents are trained with the same ϵ schedule. We show varying $\frac{max}{min}$ repetitions for DAR and the best t-DQN architecture (see Table 3). Bold faced values give the overall best AUC and cursive values the best per method which are plotted in Figure 6.

(a) MountainCar-v0

		DAR				
DQN	t-DQN	$\frac{2}{1}$	$\frac{4}{1}$	$\frac{6}{1}$	$\frac{8}{1}$	$\frac{10}{1}$
0.50	***0.64***	0.43	0.45	*0.60*	0.56	0.56

(b) LunarLander-v2

0.83	***0.88***	0.84	*0.85*	0.81	0.72	0.60

carefully trading off fine control and coarse control results in an AUC of 0.593. Figure 6a shows that DAR learns to trade off both coarse and fine-control. However, as DAR does not know that two output heads correspond to the same action, with different repetition values, DARs reward begins to drop in the end as it learns to overly rely on coarse control. During the whole training procedure the best t-DQN agent and the best DAR agent result in policies that require far fewer decisions, with t-DQN requiring only ≈ 50 decisions per episode reducing the number of decisions by a factor of ≈ 3 compared to vanilla DQN.

LunarLander For such a dense reward there is only a small improvement for t-DQN and a properly tuned DAR agent. Again the t-DQN agent performs best, achieving a slightly higher AUC of 0.88 than the best tuned DAR agent (0.85), see Tables 3b & 4b. Further, Figure 6b shows that, in this setting, t-DQN agents quickly learn to be very reactive, acting nearly at every time-step. Again, DAR can not learn that some output heads apply the same behaviour action for multiple time-steps, preferring coarse over fine control.

4.2.3. ATARI ENVIRONMENTS

Setup We trained all agents for a total of 2.5×10^6 training steps (i.e. only 10 million frames) using a linearly ϵ-greedy exploration schedule over the first 200 000 time-steps with a final fixed ϵ set to 0.01. We evaluated all agents every 10 000 training steps and evaluated for 3 episodes. For increased learning stability we implemented all agents using double deep $\mathcal{Q}$ networks. For DQN we used the architecture of Mnih et al. (2015) which also serves as basis for our shared t-DQN and the DAR architecture. As maximal skip-value we chose 10. A detailed list of hyperparameters is given in Appendix H. Following (Bellemare et al., 2013), we used a frame-skip of 4 to allow for a fair comparison to the base DQN. We used *OpenAI Gym*'s We trained all agents on the games BEAMRIDER, FREEWAY, MSPACMAN, PONG and QBERT.

Learning When to Act in Atari Figure 7 depicts the learning curves as well as the number of steps and decisions. The training behaviour from our TEMPORL agents falls into one of three categories on all evaluated Atari games.

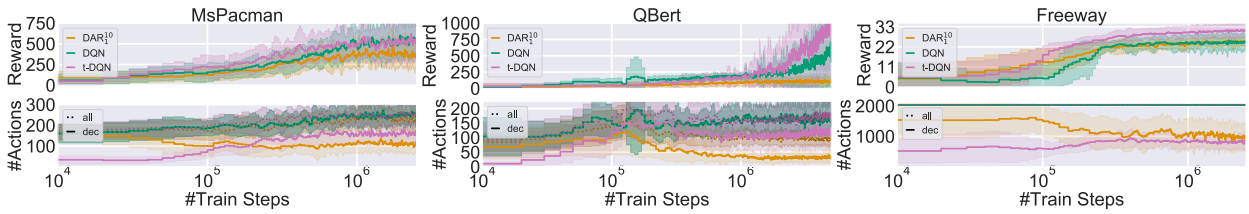


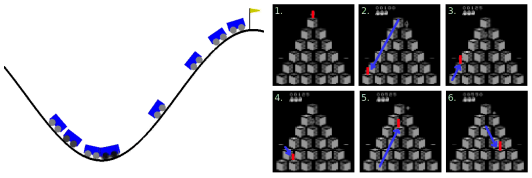
Figure 7. Evaluation performance on Atari environments. Solid lines give the mean and the shaded area the standard deviation over 15 random seeds. (top) Achieved rewards. (bottom) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies. To make trends more visible, we smooth over a window of width 7.

(i) Our learned t-DQN exhibits a slight improvement in learning speed, on MSPACMAN and PONG[2] before being caught up by DQN, with both methods converging to the same final reward (see Figures 7a & H1a). Nevertheless, TEMPORL learns to make use of different degrees of fine and coarse control to achieve the same performance. For example, a trained proactive TEMPORL policy requires roughly 33% fewer decisions. DAR on the other hand, learns to overly rely on the coarse control, leading to far fewer decisions but also worse final performance.

(ii) On QBERT the learning performance of our t-DQN lags behind that of vanilla DQN over the first 10^6 steps. Figure 7b (bottom) shows that in the first $\approx 0.5 \times 10^6$ steps, TEMPORL first has to learn which skip values are appropriate for Qbert. In the next $\approx 0.5 \times 10^6$ steps, our t-DQN begins to catch up in reward, while using its learned fine and coarse control, before starting to overtake its vanilla counterpart. As it was not immediately clear if this trend would continue after 2.5×10^6 training steps, we continued the experiment for twice as many steps. TEMPORL continues to outperform its vanilla counterpart, having learned to trade off different levels of coarse and fine control. The effect of over-reliance of DAR on the coarse control is further amplified on QBERT resulting in far worse policies than either vanilla DQN and TEMPORL.

(iii) In games such as FREEWAY and BEAMRIDER (Figures 7c & H1b), we see an immediate benefit to jointly learning *when* and *how* to act through TEMPORL. For these games, our t-DQNs begin to learn faster and achieve a better final reward than vanilla DQNs. An extreme example for this is FREEWAY, where the agents have to control a chicken to cross a busy multi-lane highway as often as possible within a fixed number of frames. To this end one action has to be played predominantly, whereas the other two possible actions are only needed to sometimes avoid an oncoming car. The vanilla DQN learns to nearly constantly play the predominant action, but does not learn proper avoidance strategies, leading to a reward of $\approx$ 25 (i.e successfully crossing the road 25 times). t-DQN on the other hand not only learns faster to repeatedly play the predominant action, but also learns proper avoidance strategies by learning to anticipate *when* a new decision has to be made, resulting in an average reward close to the best possible reward of 34. Here, DAR profits from the use of a coarse control, learning faster than vanilla DQN. However, similarly to vanilla DQN, DAR learns a policy that can only achieve a reward of 25, not learning to properly avoid cars.

[2]Results for PONG and BEAMRIDER are given in Appendix H.



(a) MountainCar (b) QBert

Figure 8. Example States in which TEMPORL makes new decisions. The agents are trained with a maximal skip-length of 10. (a) Example states in which TEMPORL learned when to make new decisions in MountainCar, starting slightly to the right of the valley. (b) Example states of Qbert. To make it easier to see where QBert is in the images we highlight him as a red square and indicate the taken trajectory with a blue arrow.

5. Analysis of TempoRL Policies

To analyze TEMPORL policies and the decisions *when* to act we selected trained agents and evaluated their policies on the environments they were trained on. Videos for all the behaviours we describe here are part of the supplementary[3]. In the tabular case we plot the key-states for an agent (see Figure 4) that can skip at most 7 steps ahead. On the Cliff environment TEMPORL learns to make a decision in the starting state, once it has cleared the cliff, once before reaching the other side (since it can not skip more than 7 states) and once to go down into the goal. Similar observations can be made for all grid-worlds. This shows that in this setting our TEMPORL agents are capable of skipping over unimportant states and learned *when* they are required to perform novel actions.

[3]Available at `github.com/automl/TempoRL`

Key-states in which TEMPORL decides to take new actions in the featurized MountainCar environment are shown in Figure 8a. Starting slightly to the right of the valley, the agent learns to gain momentum by making use of skips, repeating the left action[4]. As soon as TEMPORL considers the run-up to be sufficient to clear the hill on the right, it switches the action direction. From this point on TEMPORL sticks with this action and always selects the largest available skip-length (i.e. 10). Still, TEMPORL has to make many intermediate decisions, as the agent is limited by the maximal skip-length.

Finally, we evaluated TEMPORL's skipping behaviour on Qbert. An example of key states in which TEMPORL decides to make new decisions are given in Figure 8b. Our TEMPORL agent learns to use large skip-values to reach the bottom of the left column, lighting up all platforms in between. After that the agent makes use of large skips to light up the second diagonal of platforms. Having lit up a large portion of the platform, TEMPORL starts to make fewer uses of skips. This behaviour is best observed in the video provided in the supplementary. Also, note that we included all trained networks in our supplementary such that readers can load the networks to study their behaviour.

This analysis confirms that TEMPORL is capable of not only reacting to states but also learning to anticipate *when* a switch to a different action becomes necessary. Thus, besides the benefit of improved training speed through better guided exploration, TEMPORL improves the interpretability of learned policies.

6. Conclusion

We introduced skip-connections into the existing MDP formulation to propagate information about future rewards much faster by repeating the same action several times. Based on skip-MDPs, we presented a learning mechanism that makes use of existing and well understood learning methods. We demonstrated that our new method, TEMPORL is capable of learning not only how to act in a state, but also *when* a new action has to be taken, without the need for prior knowledge. We empirically evaluated our method using tabular and deep function approximation and empirically evaluated the learning behaviour in an adversarial setting. We demonstrated that the improved learning speed not only comes from the ability of repeating actions but that the ability to learn which repetitions are helpful provided the basis of learning *when* to act. For both tabular and deep RL we demonstrated the high effectiveness of our approach and showed that even in environments requiring mostly fine-control our approach performs well. Further, we evaluated the influence of exploration strategies, architectural choices and maximum skip-values of our method and showed it to be robust.

As pointed out by Huang et al. (2019), observations might be costly. In such cases, we could make use of TEMPORL to learn how to behave and when new actions need to be taken; when using the learned policies, we could use the learned skip behaviour to only observe after having executed the longest skips possible. All in all, we believe that TEMPORL opens up new avenues for RL methods to be more sample efficient and to learn complex behaviours. As future work, we plan to study distributional TEMPORL as well as how to employ different exploration policies when learning the skip policies and behaviour policies.

[4]Note, in the particular example given in Figure 8a the agent first performs the left action twice, each for one time-step before it recognizes that it is gaining momentum and it can make use of large skips.

Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC, André Biedenkapp, Raghu Rajan and Frank Hutter by the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG as well as funding by the Robert Bosch GmbH, and Marius Lindauer by the DFG through LI 2801/4-1. The authors would like to thank Will Dabney for providing valuable initial feedback as well as Fabio Ferreira and Steven Adriaensen for feedback on the first draft of the paper.

References

Bacon, P., Harb, J., and Precup, D. The option-critic architecture. In S.Singh and Markovitch, S. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'17)*. AAAI Press, 2017.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: iclr.cc.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

Biedenkapp, A., Bozkurt, H. F., Eimer, T., Hutter, F., and Lindauer, M. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In Lang, J., Giacomo, G. D., Dilkina, B., and Milano, M. (eds.), *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, pp. 427–434, June 2020.

Braylan, A., Hollenbeck, M., Meyerson, E., and Miikkulainen, R. Frame skip is a powerful parameter for learn-

ing to play atari. In *Proceedings of the Workshops at Twenty-ninth National Conference on Artificial Intelligence (AAAI'15)*, 2015.

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

Chaganty, A., Gaur, P., and Ravindran, B. Learning in a small world. In van der Hoek, W., Padgham, L., Conitzer, V., and Winikoff, M. (eds.), *International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2012*, pp. 391–397. IFAAMAS, 2012.

Chrpa, L. and Vallati, M. Improving domain-independent planning via critical section macro-operators. In Hentenryck, P. V. and Zhou, Z. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'19)*, pp. 7546–7553. AAAI Press, 2019.

Dabney, W., Ostrovski, G., and Barreto, A. Temporally-extended ϵ-greedy exploration. *arXiv:2006.01782 [cs.LG]*, 2020.

Doya, K. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

Eysenbach, B., Salakhutdinov, R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 15220–15231, 2019.

Hallak, A., Castro, D. D., and Mannor, S. Contextual markov decision processes. *arXiv:1502.02259 [stat.ML]*, 2015.

Harb, J., Bacon, P., Klissarov, M., and Precup, D. When waiting is not an option: Learning options with a deliberation cost. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pp. 3165–3172. AAAI Press, 2018.

Harutyunyan, A., Vrancx, P., Bacon, P., Precup, D., and Nowé, A. Learning with options that terminate off-policy. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pp. 3173–3182. AAAI Press, 2018.

Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., and Precup, D. The termination critic. In Chaudhuri, K. and Sugiyama, M. (eds.), *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 89 of *Proceedings of Machine Learning Research*, pp. 2231–2240. PMLR, 2019.

Huang, Y., Kavitha, V., and Zhu, Q. Continuous-time markov decision processes with controlled observations. In *Proceedings of the 57th Annual Allerton Conference on Communication, Control, and Computing*, pp. 32–39. IEEE, 2019.

Khan, A., Feng, J., Liu, S., and Asghar, M. Z. Optimal skipping rates: training agents with fine-grained control using deep reinforcement learning. *Journal of Robotics*, 2019, 2019.

Khetarpal, K. and Precup, D. Learning options with interest functions. In Hentenryck, P. V. and Zhou, Z. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'19)*, pp. 9955–9956. AAAI Press, 2019.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, 2015. Published online: iclr.cc.

Lakshminarayanan, A. S., Sharma, S., and Ravindran, B. Dynamic action repetition for deep reinforcement learning. In S.Singh and Markovitch, S. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'17)*, pp. 2133–2139. AAAI Press, 2017.

Lee, J., Lee, B., and Kim, K. Reinforcement learning for control with multiple frequencies. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, volume 33, 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. In *Proceedings of the International Conference on Learning Representations (ICLR'16)*, 2016. Published online: iclr.cc.

Mankowitz, D. J., Mann, T. A., Bacon, P., Precup, D., and Mannor, S. Learning robust options. In McIlraith, S. and Weinberger, K. (eds.), *Proceedings of the Conference on Artificial Intelligence (AAAI'18)*, pp. 6409–6416. AAAI Press, 2018.

Metelli, A., Mazzolini, F., Bisi, L., Sabbioni, L., and Restelli, M. Control frequency adaptation via action persistence in batch reinforcement learning. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. Proceedings of Machine Learning Research, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control

through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Modi, A., Jiang, N., Singh, S. P., and Tewari, A. Markov decision processes with continuous side information. In *Algorithmic Learning Theory (ALT'18)*, volume 83 of *Proceedings of Machine Learning Research*, pp. 597–618. PMLR, 2018.

Moore, A. W. *Efficient memory-based learning for robot control*. PhD thesis, Trinity Hall, University of Cambridge, Cambridge, 1990.

Nasiriany, S., Pong, V., Lin, S., and Levine, S. Planning with goal-conditioned policies. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 14814–14825. 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Proceedings of the 32nd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 8024–8035, 2019.

Precup, D., Sutton, R. S., and Singh, S. P. Theoretical results on reinforcement learning with temporally abstract options. In *Proceedings of the 10th European Conference on Machine Learning (ECML)'98*, pp. 382–393, 1998.

Schoknecht, R. and Riedmiller, M. A. Speeding-up reinforcement learning with multi-step actions. In Dorronsoro, J. R. (ed.), *Proceedings of the International Conference on Artificial Neural Networks (ICANN'02)*, volume 2415 of *Lecture Notes in Computer Science*, pp. 813–818. Springer, 2002.

Schoknecht, R. and Riedmiller, M. A. Reinforcement learning on explicitly specified time scales. *Neural Computing and 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. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: iclr.cc.

Stolle, M. and Precup, D. Learning options in reinforcement learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation SARA'02*, volume 2371 of *Lecture Notes in Computer Science*, pp. 212–223. Springer, 2002.

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.

Tiganj, Z., Shankar, K. H., and Howard, M. W. Scale invariant value computation for reinforcement learning in continuous time. In *Proceedings of the AAAI Spring Symposia'17*, 2017.

Vallati, M., Chrpa, L., and Serina, I. MEvo: a framework for effective macro sets evolution. *Journal of Experimental & Theoretical Artificial Intelligence*, 0(0):1–19, 2019.

van Hasselt, H. Double q-learning. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A. (eds.), *Proceedings of the 23rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'10)*, pp. 2613–2621, 2010.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In Schuurmans, D. and Wellman, M. (eds.), *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*, pp. 2094–2100. AAAI Press, 2016.

Vezhnevets, A., Mnih, V., Osindero, S., Graves, A., Vinyals, O., and Agapiou, J. Strategic attentive writer for learning macro-actions. In Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16)*, pp. 3486–3494, 2016.

A. Detailed Baseline Description

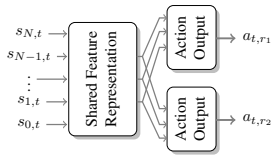


Figure A1. Schematic DAR Architecture with duplicate output heads to learn at two time-scales r_1 and r_2.

Dynamic Action Repetition (DAR; Lakshminarayanan et al., 2017) is a framework for discrete-action space deep RL algorithms. For a discrete-action space $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ DAR duplicates this space such that an agent can choose from $2 \times |\mathcal{A}|$ actions. Further, DAR introduces two hyperparameters r_1 and r_2, each of which are associated with one half of the new action space. These hyperparameters determine the number of time-steps an action will be played for, with both actions a_k and a_{2k} ($1 \leq k \leq |\mathcal{A}|$) performing the same behaviour but a_k is repeated for r_1 time-steps and a_{2k} for r_2 time-steps. When training an agent, there are no modifications to the training procedure, other than an agent now having to select from a larger action space. Figure A1 schematically depicts a DAR DQN agents $\mathcal{Q}$-network architecture.

This gives an agent two levels of control to decide on how long to apply an action. A drawback of this framework is that the output heads are independent from each other and are not aware that certain action outputs have the same influence on the environment for $\min(r_1, r_2)$ time-steps. Further, both r_1 and r_2 have to be defined beforehand, requiring good prior knowledge about the potential levels of fine and coarse control in an environment.

Fine Grained Action Repetition (FiGAR; Sharma et al., 2017) is a framework for both discrete and continuous action spaces. Instead of learning a single policy that has to learn both which action to play and how long to follow it (as in DAR), FiGAR decouples the behaviour and repetition learning by using two separate policies $\pi_a : \mathcal{S} \to \mathcal{A}$ and $\pi_r : \mathcal{S} \to \{1, 2, \ldots, \text{max repetition}\}$. When training an agent, based on a state s, π_a decides which action to play and simultaneously π_r decides how long to repeat a selected action starting from s. At the time of selecting their respective actions, neither π_a nor π_r are aware of the other policies decision. Thus, the action and the respective repetition value are selected independently from each other.

To couple the learning of both policies Sharma et al. (2017) use a joint loss to update the network weights and further suggest to use weight-sharing of the input-layers of the two policy networks. Although this aligns the policies when performing a training step, at decision time the policies remain uninformed about each others behaviour. Counter to DAR, FiGAR allows for much more fine-grained control over the action repetition. However, FiGAR requires more modification of a base algorithm to allow for learning of control at different time-steps. With TEMPORL we propose a method that allows for the same fine-grained level of control while requiring no modifications to the base agent architecture.

Algorithm 1 TEMPORL $\mathcal{Q}$-learning

1: **Input:** environment env with states $\mathcal{S}$ and actions $\mathcal{A}$, skip-Actions $\mathcal{J}$,
behaviour and skip $\mathcal{Q}$-functions $\mathcal{Q}(\cdot,\cdot)$, $\mathcal{Q}(\cdot,\cdot|\cdot)$, training episodes E
2: Initialize $\mathcal{Q}(s,a)$, $\mathcal{Q}(s,j|a) \forall s \in \mathcal{S}, a \in \mathcal{A}, j \in \mathcal{J}$
3: **for** episode$\in \{1,\dots,E\}$ **do**
4: $s \leftarrow env.reset()$
5: **repeat**
6: $a \leftarrow \pi(s)$ # e.g. ϵ-greedy $\arg\max_{a' \in \mathcal{A}} \mathcal{Q}(s,a')$
7: $j \leftarrow \pi_j(s,a)$ # e.g. ϵ-greedy $\arg\max_{j' \in \mathcal{J}} \mathcal{Q}(s,j'|a)$
8: trajectory $\leftarrow [s]$ # Tracks the skip trajectory
9: **repeat**
10: $r, s' \leftarrow env.step(a)$
11: append s' to trajectory # Records the state transitions
12: $\mathcal{Q}(s,a) \leftarrow td\_update(\mathcal{Q}(s,a), r, s')$ # See Equation 5
13: $s \leftarrow s'$
14: **until** all skips $1,\dots,j$ performed or episode ends
15: $\mathcal{G} \leftarrow build\_connectedness\_graph$(trajectory) # Build a local connectedness graph from the observed trajectory
16: **for all** connections $c \in \mathcal{G}$ **do**
17: get $s_{start}, s_{end}, j', r'$ from c
18: $\mathcal{Q}(s_{start}, j'|a) \leftarrow td\_update\_skip(\mathcal{Q}(s_{start}, j'|a), r', s_{end})$ # See Equation 6
19: **end for**
20: **until** episode finished
21: **end for**

B. Implementation Details

Algorithm 1 details how to train a TEMPORL $\mathcal{Q}$-learning agent. All elements that are new to TEMPORL are shown in black whereas vanilla $\mathcal{Q}$-learning code is greyed out. The functions td_update (Line 12) and td_update_skip (Line 18) are formally stated in Equations 5 and 6 respectively and give the temporal difference updates required during learning.

$$\mathcal{Q}(s_t,a_t) = \mathcal{Q}(s_t,a_t) + \alpha \underbrace{\left(\underbrace{\left(r_t + \gamma \max \mathcal{Q}(s_{t+1},\cdot) \right)}_{\text{TD-Target}} - \mathcal{Q}(s_t,a_t) \right)}_{\text{TD-Delta}} \tag{5}$$

$$\mathcal{Q}(s_t,j_t|a_t) = \mathcal{Q}(s_t,j_t|a_t) + \alpha \underbrace{\left(\underbrace{\left(\sum_{k=0}^{j-1} \gamma^k r_{t+k} + \gamma^j \max \mathcal{Q}(s_{t+j},\cdot) \right)}_{\text{TD-Target}} - \mathcal{Q}(s_t,j_t|a_t) \right)}_{\text{TD-Delta}} \tag{6}$$

Where α is the learning rate and γ the discount factor. Note that the TD-Target in Equation 6 (as well as the skip $\mathcal{Q}$-function in Equation 4) is using the behaviour $\mathcal{Q}$-function and not the skip $\mathcal{Q}$-function. Thus, the skip $\mathcal{Q}$-function estimates the expected future rewards, assuming that the current skip will be the only skip in the MDP. This allows us to avoid overestimating $\mathcal{Q}$-values through multiple skips and focuses on learning of the value of the executed skip similar to double $\mathcal{Q}$-learning (van Hasselt, 2010). Further, learning of the skip-values does not interfere with learning of the behaviour $\mathcal{Q}$-function.

The function $build\_connectedness\_graph$ (Line 15) builds takes an observed trajectory and builds connectedness graph of states that are reachable by repeatedly playing the same action (see Figure 1 in the main paper). Each connection contains information about start and end states, the length of the skip and the discounted reward for that skip.

C. Used Compute Resources

Tabular & Deep RL Experiments on Featurized Environments For the tabular as well as the deep experiments on featurized environments, we evaluated all agents on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache and and 188GB (shared) RAM running Ubuntu 18.04 LTS 64 bit. In all cases, the agents were allocated one CPU. The tabular agents required at most 20 minutes to complete training, whereas the deep agents required at most 15 hours.

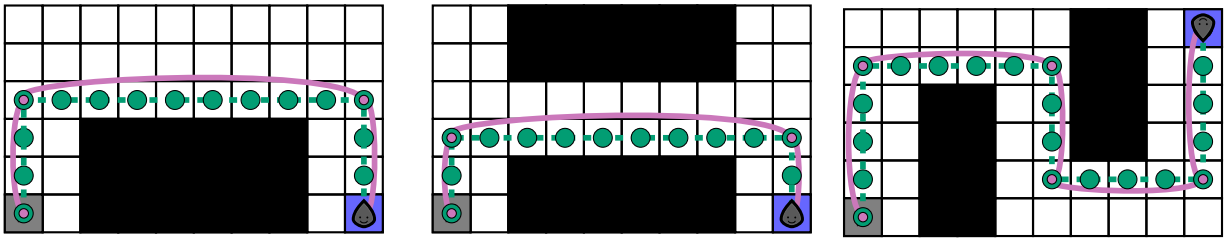
Deep RL Experiments on Atari Environments These experiments were run on a compute cluster with nodes equipped with two Intel Xeon E5-2630v4 and 128GB memory running CentOS 7. For training, the agents were allocated 10 CPUs and required at most 48 hours to complete training.

D. Gridworld Details

All considered environments (see Figure D1) are discrete, deterministic, have sparse rewards and have size 6×10. Falling off a cliff results in a negative reward (-1) and reaching a goal state results in a positive reward ($+1$). Both cliff and goal states terminate an episode. All other states result in no reward. An agent can only execute the actions up, $down$, $left$, $right$ with diagonal moves not possible. If the agent does not reach a goal/cliff in 100 steps, an episode terminates without a reward.

For the Cliff environment, a shortest path through the environment requires 16 steps. However, to reach the goal, decisions about unique actions are only required at 3 time points. The first is in the starting state and determines that action up should be repeated 3-times, the next is repeating action $right$ 10-times and the final one is repeating action $down$ 3-times. Thereby, an optimal proactive policy that is capable of joint decision of action and skip length requires $\sim 80\%$ fewer decisions than an optimal reactive policy that has to make decisions in each state. As the Bridge environment is very similar, but has a smaller cliff area below, an optimal proactive policy also requires roughly $\sim 80\%$ fewer decisions.

On the more complex ZigZag environment, an optimal policy requires 20 steps in total to reach the goal. In this environment however, an agent has to switch direction more often. Leading to a total of 5 required decisions. Thus in this environment an optimal proactive policy requires roughly 75% fewer decisions.



(a) Cliff (b) Bridge (c) ZigZag

Figure D1. Copy of Figure 4 form the main paper. 6×10 Grid Worlds. Agents have to reach a fixed goal state from a fixed start state. Large/small dots represent decision steps of vanilla and TEMPORL $\mathcal{Q}$-learning policies.

Table E1. Normalized AUC for reward and average number of decision steps for varying maximal skip-lengths J. All agents are trained with the same ϵ schedule. $\mathcal{R}$ denotes normalized area under the reward curve and D the average number of decision steps. Values are results of running 10 random seeds. Columns 1 and 7 are equivalent to columns 5 & 6 in Table 1.

(a) linear decaying ϵ-schedule

	$\mathcal{Q}$	t-$\mathcal{Q}$														
J	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathcal{R}$	0.57	0.63	0.76	0.87	0.93	0.93	0.92	0.91	0.90	0.91	0.88	0.87	0.86	0.87	0.85	0.84
D	83.6	36.5	20.6	13.2	10.1	8.3	7.7	7.8	7.5	7.4	7.6	7.4	7.6	7.6	7.8	7.4

(b) logarithmic decaying ϵ-schedule

$\mathcal{R}$	0.90	0.91	0.93	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.95	0.96	0.95	0.95
D	35.6	21.7	14.9	11.6	9.5	8.6	6.4	6.3	6.5	5.9	6.1	6.2	7.0	6.8	7.0	6.0

(c) constant $\epsilon = 0.1$

$\mathcal{R}$	0.95	0.98	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98
D	27.6	15.8	12.0	9.1	8.2	7.8	6.8	6.9	6.7	7.1	6.6	7.2	6.2	6.5	7.0	6.9

E. Influence of the Maximum Skip-Length

The maximum skip length J is a crucial hyperparameter of TEMPORL. A too large value might lead to many irrelevant choices which the agent has to learn to ignore; whereas a too small value might not reduce the complexity of the environment sufficiently enough, leading to barely an improvement over the vanilla counterpart. To evaluate the influence of the hyperparameter on our method we trained various TEMPORL agents with varying maximal skip-lengths, starting from 2 up to 16. Larger skips than 10 will never be beneficial for the agent as the agent is guaranteed to run into a wall for some steps. Depending on where in the environment the agent is located, smaller skip-values might allow it to quickly traverse through the environment.

Table E1 shows the influence of J on the ZigZag environment (see Figure 4c). In this environment, the largest skip value that is possible without running into a wall is 6. Thus, small skip values up to 5 quickly improve the performance over the vanilla counterpart, not only in terms of anytime performance but also in terms of required decisions. In the case of a suboptimal exploration policy, in the form of linearly decaying ϵ-greedy schedule (see Table E1a), larger skip-values quickly lead to a decrease in anytime performance, as the agent has to learn to never choose many non-improving skip actions.

For a more suiting exploration policy, too large skip-actions do not as quickly degrade the anytime performance of our TEMPORL agents. In the case of a logarithmically decaying ϵ schedule (Table E1b), we can see that skip sizes larger or equal than 12 start to negatively influence the anytime performance, whereas with a constant ϵ schedule only a skip-size of 16, nearly 3 times as large as the largest sensible choice, has a negative effect.

Similar observations can be made for deep TEMPORL on both Pedulum, MountainCar and LunarLander, see Tables 2 - 4 in the main paper. We can see that choosing larger maximal skip-values is beneficial, up to a point, at which many irrelevant, and potentially useless choices are in the action space. For these, TEMPORL first has to learn on which part of the skip-action-space to focus before really learning when new decisions need to be taken.

It is worth noting that, in the tabular case, all evaluated skip-sizes J result in better anytime-performance and a lower number of required decision points compared to vanilla $\mathcal{Q}$-learning, for all considered exploration strategies. In future work, we will study how to allow TEMPORL to select large skip-actions without needing to learn to distinguish between many irrelevant choices. One possible way of doing this could be by putting the skip-size on a log scale. For example using $\log_2$ could result in only 10 actions where a TEMPORL agent could skip up to 1024 steps ahead but would still be able to exert fine control with the smaller actions.

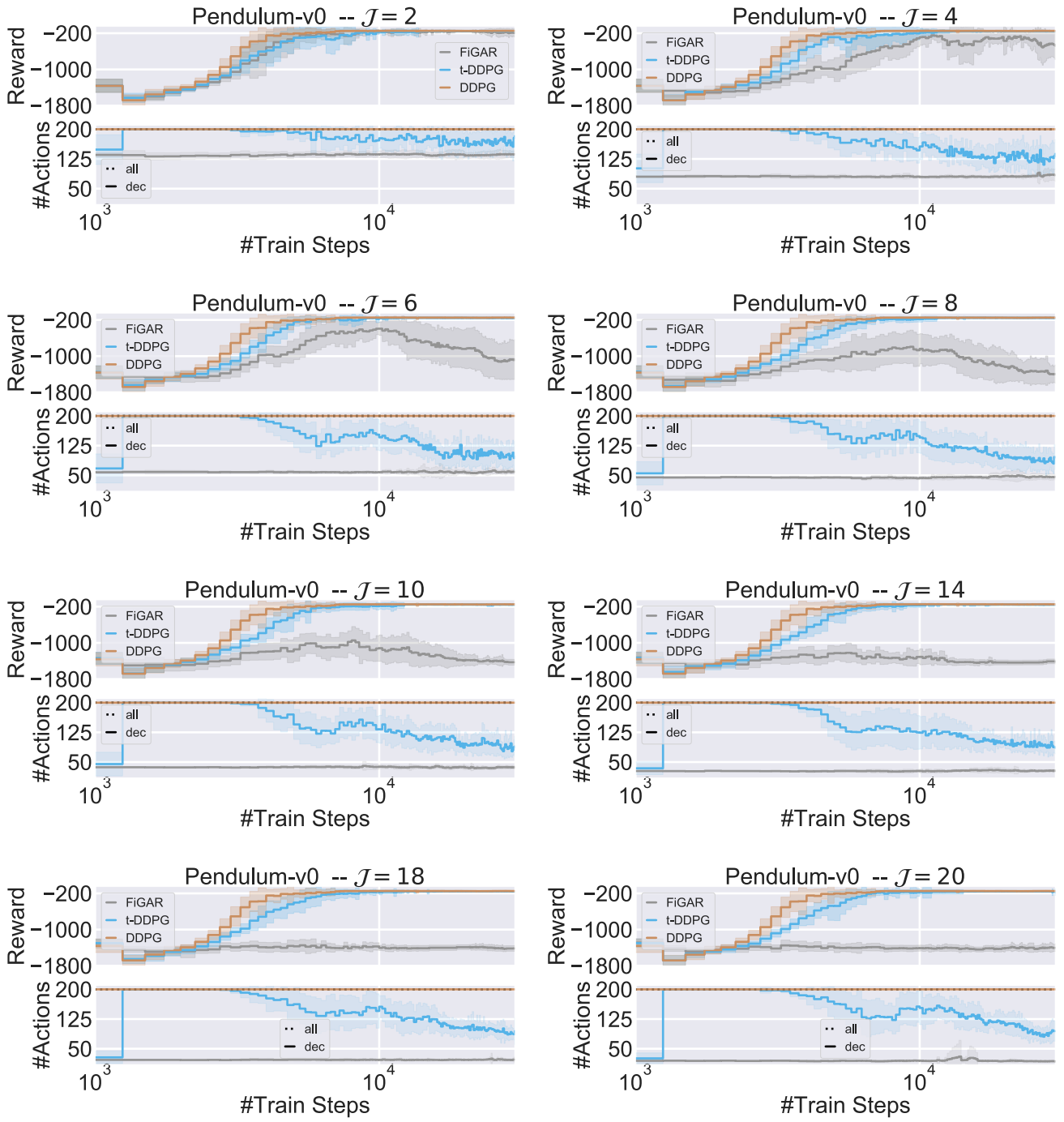


Figure F1. Learning curves of different DDPG agents on Pendulum-v0. J indicates the maximal skip-length used when training t-DDPG and FiGAR. Solid lines give the mean and the shaded area the standard deviation over 15 seeds. Top-row images show the reward achieved and bottom-row images the required steps and decisions per evaluation rollout.

F. DDPG Implementation Details and Additional Results

As base implementation for DDPG, we used publicly available code[5] and used the default hyperparameters, except we replaced the number of maximal training steps and initial random steps as described in the main paper. When implementing FiGAR, we followed the description by Sharma et al. (2017). Thus, the repetition policy uses a constant epsilon-greedy exploration. Likewise, we use a constant epsilon-greedy exploration to learn our t-DDPG.

For our t-DDPG implementation we could use the same algorithm as described in Algorithm 1. Only the greyed out parts of normal $\mathcal{Q}$-learning have to replaced by DDPG training specific elements. For example, for DDPG, the exploration policy for the actor is given by adding exploration noise rather than following an epsilon-greedy policy. Further, we again can make use of the base agents $\mathcal{Q}$-function as shown in Equation 6.

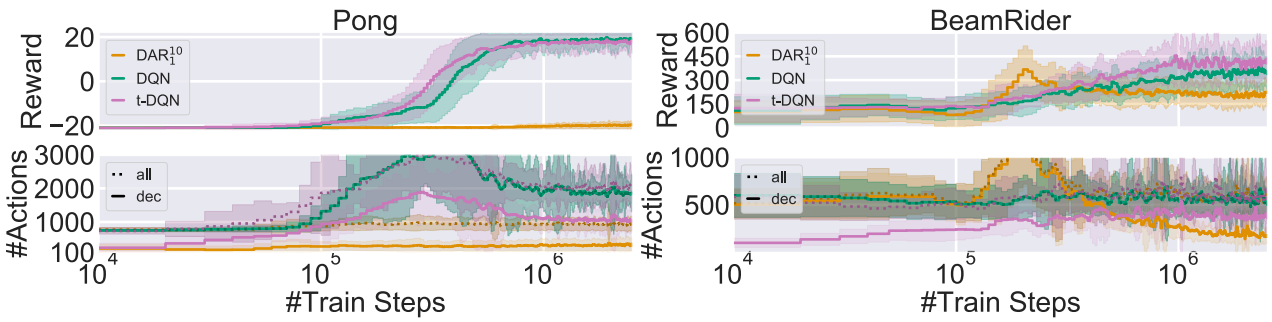
Figure F1 depicts the learning curve for all DDPG agents with increasing maximal skip-value. As described in the main paper, both FiGAR and t-DDPG slightly lag behind vanilla DDPG when only allowing for skips of length 2. However, with increasing max-skip value FiGAR quickly begins to struggle and in the end even converges to worse policies, always preferring large skip-values. Our t-DDPG using TEMPORL performs much more stable and is hardly affected by increasing the maximal skip length. Further, t-DDPG over time learns *when* it is necessary to switch to new actions, roughly halving the required decisions.

G. Featurized Environments Description

MountainCar is a challenging exploration task and requires an agent to control an under powered car to drive up a steep hill on one side (Moore, 1990). To reach the goal, an agent has to build up momentum. The agent always receives a reward of -1 until it has crossed the goal position and a reward of 0 afterwards. The observation consists of the car position and velocity and the agent can either accelerate to the left or right or do nothing. To build up momentum an agent potentially has to repeat the same action multiple times. Thus, we evaluate both t-DQN and DAR on the grid $\{2, 4, 6, 8, 10\}$ for the maximal (while keeping the minimal skip value fixed to 1) skip-value over 50 random seeds (see Tables 3a & 4a).

LunarLander The task for an agent is to land a space-ship on a lunar surface. To this end, the agent can choose to fire the main engine, steer left or right or do nothing. Firing of the engines incurs a small cost of -0.3, whereas crashing or successfully landing results in a large cost or reward of -100 and 100 respectively. We expect that an environment with such a dense reward, where actions directly influence the achieved reward does not benefit from leveraging skips.

[5] https://github.com/sfujim/TD3



(a) Pong (b) BeamRider

Figure H1. Evaluation performance on Atari environments. Solid lines give the mean and the shaded area the standard deviation over 15 random seeds. (top) Achieved rewards. (bottom) Length of executed policy ($\cdots$) and number of decisions (—) made by the policies.

H. Atari

Architectures *DQN:* As architecture for DQN we used that of Mnih et al. (2015) and used this as basis for our shared architecture. This architecture has three layers of convolutions to handle the 84×84 input images. The first convolution layer has 84 input channels, 32 output channels, a kernel size of 8 and a stride of 4. The second has 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2. The second has 64 input channels, 64 output channels, a kernel size of 3 and a stride of 1. This is followed by two hidden layers with 512 units each.

TEMPORL: The shared architecture used by our TEMPORL agent uses the same architecture as just described but has an additional output stream for the skip-outputs. The skip output stream combines a hidden layer with 10 units together with the output of the last convolutional layer. It then processes these features again in two fully connected hidden layers with 512 units each.

DAR: Similarly, the DAR agent builds on the DQN architecture of Mnih et al. (2015). However, the final output layer is duplicated and the duplicate outputs act at a different time-resolution. To give DAR the same coarse control as would be possible with our TEMPORL agent we fix the fine and coarse control levels to 1 and 10 respectively.

Table H1. Hyperparameters used for the Atari Experiments

Hyperparameter	Value
Batch Size	32
γ	0.99
Gradient Clip	40.0
Target update frequency	500
Learning starts	10 000
Initial ϵ	1.0
Final ϵ	0.01
ϵ time-steps	200 000
Train frequency	4
Loss Function	Huber Loss
Optimizer	Adam
Learning rate	10^-4
β_1	0.9
β_2	0.999
Replay-Buffer Size	5×10^4
Skip Replay-Buffer Size	5×10^4
J	10

Additional Results on PONG: Our learned t-DQN exhibits a slight improvement in learning speed, PONG before being caught up by DQN (similar to the results on MsPacman in the main paper, see Figure 7a), with both methods converging to the same final reward. Nevertheless, TEMPORL learns to make use of different degrees of fine and coarse control to achieve the same performance, requiring roughly 1 000 fewer decisions.

The DAR agent really struggles to learn a meaningful policy on this game, never learning to properly avoid getting scored on or scoring itself. A likely reason for the poor performance is the choice of hyperparameters. Potentially choosing smaller skip-value for the coarse control could allow to learn better behaviour with DAR.

Additional Results on BEAMRIDER: Figure H1b shows an immediate benefit to jointly learning *when* and *how* to act through TEMPORL. Our t-DQN begins to learn faster and achieve a better final reward than vanilla DQN.

Interestingly, the DAR agent, starting out with choosing to mostly apply fine control starts to learn much faster than vanilla DQN and our TEMPORL agent, nearly reaching the final performance of vanilla DQN already $\approx 900\,000$ time-steps earlier. However, the performance starts to drop when DAR starts to increase usage of the coarse control. Once the DAR agents have learned this over-reliance on the coarse control, they do not recover, resulting in the worst final performance.