A. Detailed Baseline Description

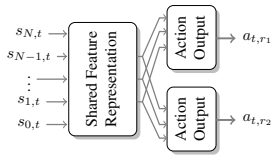


Figure A1. Schematic DAR Architecture with duplicate output heads to learn at two time-scales r_1 and r_2.

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.

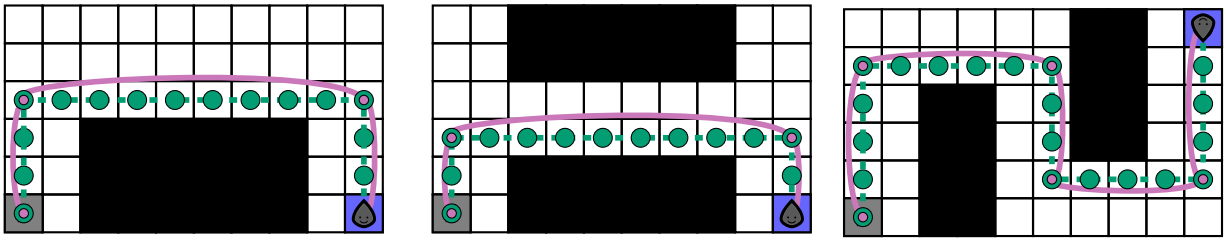
Deep RL Experiments on Atari Environments These experiments were run on a compute cluster with nodes equipped with two Intel Xeon E5-2630v4 and 128GB memory running CentOS 7. For training, the agents were allocated 10 CPUs and required at most 48 hours to complete training.

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.

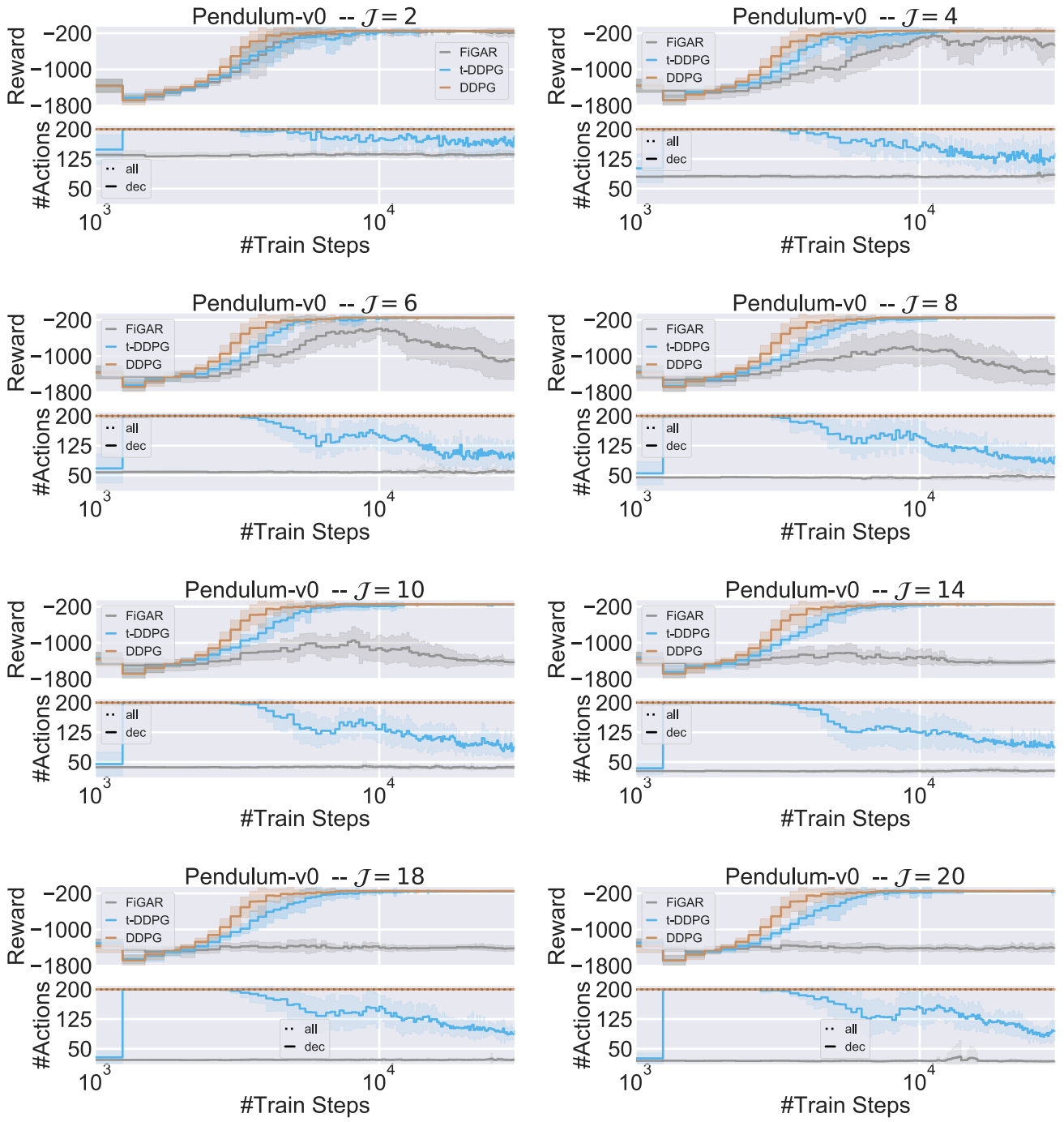


Figure F1. Learning curves of different DDPG agents on Pendulum-v0. J indicates the maximal skip-length used when training t-DDPG and FiGAR. Solid lines give the mean and the shaded area the standard deviation over 15 seeds. Top-row images show the reward achieved and bottom-row images the required steps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

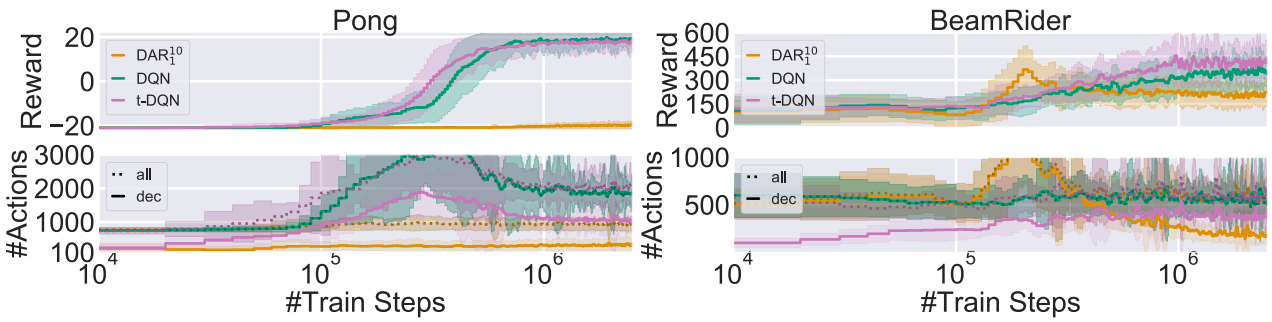
Figure F1 depicts the learning curve for all DDPG agents with increasing maximal skip-value. As described in the main paper, both FiGAR and t-DDPG slightly lag behind vanilla DDPG when only allowing for skips of length 2. However, with increasing max-skip value FiGAR quickly begins to struggle and in 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.