Thinking While Moving: Deep Reinforcement Learning in Concurrent Environments

Ted Xiao
Google Brain
tedxiao@google.com

Eric Jang
Google Brain
ejang@google.com

Dmitry Kalashnikov
Google Brain
dkalashnikov@google.com

Sergey Levine
Google Brain, UC Berkeley
slevine@google.com

Julian Ibarz
Google Brain
julianibarz@google.com

Karol Hausman*
Google Brain
karolhausman@google.com

Alexander Herzog*
X
alexherzog@x.team

1 Introduction

In recent years, Deep Reinforcement Learning (DRL) methods have achieved tremendous success on a variety of diverse environments including video games [15], robotic grasping [12], and in-hand manipulation tasks [19]. While impressive, all of these examples use a *blocking* observe-think-act paradigm: the agent assumes that the environment will remain static while it thinks, so that its actions will be executed on the same states from which they were computed. This assumption breaks in the *concurrent* real world where the environment state evolves substantially as the agent processes observations and plans its next actions. In addition to solving dynamic tasks where blocking models would fail, thinking and acting in a concurrent manner can provide practical qualitative benefits such as smoother, more human-like motions and the ability to seamlessly plan for next actions while executing the current one.

In this paper, we aim to study and incorporate knowledge about concurrent environments in the context of DRL. In particular, we derive a modified Bellman Operator for concurrent MDPs, and present the minimal set of information that we must augment state observations with in order to recover blocking performance with Q-learning. We present experiments on different simulated environments that incorporate concurrent actions, ranging from common simple control domains to vision-based robotic grasping tasks.

2 Related Work

Although real-world robotics systems are inherently concurrent, it is possible to engineer them into approximately blocking systems. For example, low-latency hardware [19] can minimize time spent during state capture and policy inference, which are main sources of latency. Another option is to make the system dynamics blocking by design, where actions are executed to completion and the system velocity is decelerated to zero before a state is recorded [12]. However, this comes at the cost of jerkier robot motions, and does not generalize to tasks where it is not possible to wait for the system to come to rest between deciding new actions.

*Indicates equal contribution.

NeurIPS 2019 Workshop on Robot Learning: Control and Interaction in the Real World, Vancouver, Canada

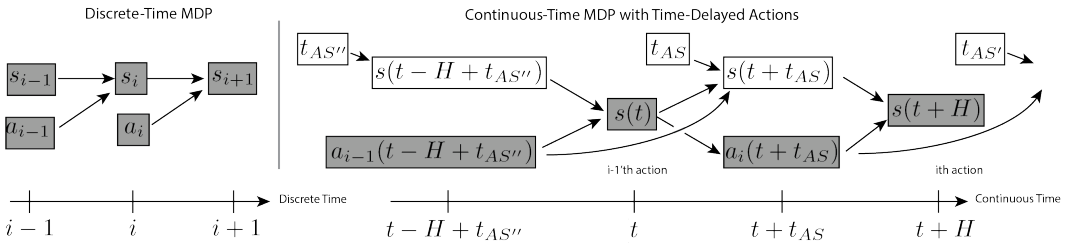


Figure 1: Shaded nodes represent observed variables and unshaded nodes represent unobserved random variables. **(a)**: in "blocking" MDPs, the environment state does not change while the agent records the current state and selects an action. **(b)**: in "concurrent" MDPs, state and action dynamics are continuous-time stochastic processes $s(t)$ and $a_i(t)$. At time t, the agent observes the state of the world $s(t)$, but by the time it selects an action $a_i(t + t_{AS})$, the last chosen action process $a_{i-1}(t - H + t_{AS''})$ has "rolled over" to an unobserved state $s(t + t_{AS})$. An agent that concurrently selects actions from old states while in motion may need to interrupt a previous action before it has finished executing its current trajectory.

Other works utilizing algorithmic changes as a more principled way to directly overcome the challenges of concurrent control can be grouped into three categories: 1) learning more general policies that are robust to latency [24], 2) including past history such as frame-stacking [18, 11], and 3) learning dynamics models to predict the future state at which the action will be executed [7, 2, 30]). These prior work are discussed in Appendix A.1.

Finally, we build many of the theoretical formulations and findings in continuous-time optimal control [13, 26] and reinforcement learning [16, 6, 4, 23], and show their applications to deep reinforcement learning methods on more complex, vision-based robotics tasks.

3 Value-based Reinforcement Learning in Concurrent Environments

The default blocking environment formulation is detailed in Figure 1a, and the effect of concurrent actions is illustrated in Figure 1b. Since state capture and policy inference occur sequentially, we consider the cumulative time for state capture, policy inference, and any communication latency to be one contiguous time duration, which we deem Action Selection (t_{AS}). t_{AS} encompasses the time duration from the instant state capture begins to when the next action is sent.

With the standard RL formulations described in Appendix A.3, we start by formalizing a continuous-time MDP with the differential equation [23]

$$ds(t) = F(s(t), a(t))dt + G(s(t), a(t))d\beta \tag{1}$$

where $\mathcal{S} = \mathbb{R}^d$ is a set of states, $\mathcal{A}$ is a set of actions, $F : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and $G : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describe the stochastic dynamics of the environment, and β is a Wiener process [20]. In the continuous-time setting, $ds(t)$ is analogous to the discrete-time p, defined in Section A.3. Continuous-time functions $s(t)$ and $a_i(t)$ specify the state and i-th action taken by the agent. The agent interacts with the environment through a state-dependent, deterministic policy function π and the return R of a trajectory $\tau = (s(t), a(t))$ is given by [6]:

$$R(\tau) = \int_{t=0}^{\infty} \gamma^t r(s(t), a(t))dt, \tag{2}$$

which leads to a continuous-time value function [23]:

$$\begin{aligned} V^\pi(s(t)) &= \mathbb{E}_{\tau\sim\pi}[R(\tau)|s(t)] \\ &= \mathbb{E}_{\tau\sim\pi}\left[\int_{t=0}^{\infty} \gamma^t r(s(t), a(t))dt\right], \end{aligned} \tag{3}$$

and similarly, a continuous Q-function:

$$Q^\pi(s(t), a, t, H) = \mathbb{E}_{s(\cdot)}\left[\int_{t'=t}^{t'=t+H} \gamma^{t'-t} r(s(t'), a(t'))dt' + \gamma^H V^\pi(s(t + H))\right], \tag{4}$$

where H is the constant sampling period between state captures (i.e. the duration of an action trajectory) and a refers to the continuous action function that is applied between t and $t+H$. The expectations are computed with respect to stochastic process p defined in Eq. 1.

In concurrent settings (Figure 1b), an agent selects N action trajectories during an episode, $a_1, ..., a_N$, where each $a_i(t)$ is a continuous function generating controls as a function of time t. Let t_{AS} be the time duration of state capture, policy inference and any additional communication latencies. At time t, an agent begins computing the i-th trajectory $a_i(t)$ from state $s(t)$, while concurrently executing the previous selected trajectory $a_{i-1}(t)$ over the time interval $(t-H+t_{AS}, t+t_{AS})$. At time $t+t_{AS}$, where $t \leq t+t_{AS} \leq t+H$, the agent switches to executing actions from $a_i(t)$. The continuous-time Q-function for the concurrent case from Eq. 4 can be expressed as following:

$$Q^{\pi}(s(t), a_{i-1}, a_i, t, H) = \underbrace{\mathbb{E}_{s(\cdot)}\left[\int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{i-1}(t'))dt'\right]}_{\text{Executing action trajectory } a_{i-1}(t) \text{ until } t+t_{AS}}$$
$$+ \underbrace{\mathbb{E}_{s(\cdot)}\left[\int_{t'=t+t_{AS}}^{t'=t+H} \gamma^{t'-t} r(s(t'), a_i(t'))dt'\right]}_{\text{Executing action trajectory } a_i(t) \text{ until } t+H} + \underbrace{\mathbb{E}_{s(\cdot)}\left[\gamma^{H} V^{\pi}(s(t+H))\right]}_{\text{Value function at } t+H} \tag{5}$$

The first two terms correspond to expected discounted returns for executing the action trajectory $a_{i-1}(t)$ from time $(t, t+t_{AS})$ and the trajectory $a_i(t)$ from time $(t+t_{AS}, t+t_{AS}+H)$.

$$\mathcal{T}_c^* \hat{Q}(s(t), a_{i-1}, a_i, t, t_{AS}) = \int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{i-1}(t'))dt' +$$
$$\gamma^{t_{AS}} \max_{a_{i+1}} \mathbb{E}_p \hat{Q}^{\pi}(s(t+t_{AS}), a_i, a_{i+1}, t+t_{AS}, H-t_{AS}). \tag{6}$$

Analogously, we define the concurrent Q-function for the discrete-time case:

$$Q^{\pi}(s_t, a_{t-1}, a_t, t, t_{AS}, H) = r(s_t, a_{t-1}) + \gamma^{\frac{t_{AS}}{H}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} Q^{\pi}(s_{t+t_{AS}}, a_t, a_{t+1}, t+t_{AS}, t_{AS'}, H-t_{AS}) \tag{7}$$

Let $t_{AS'}$ be the "spillover duration" for action a_t beginning execution at time $t+t_{AS}$ (see Figure 1b). Then the concurrent Bellman Operator, specified by a subscript c, is:

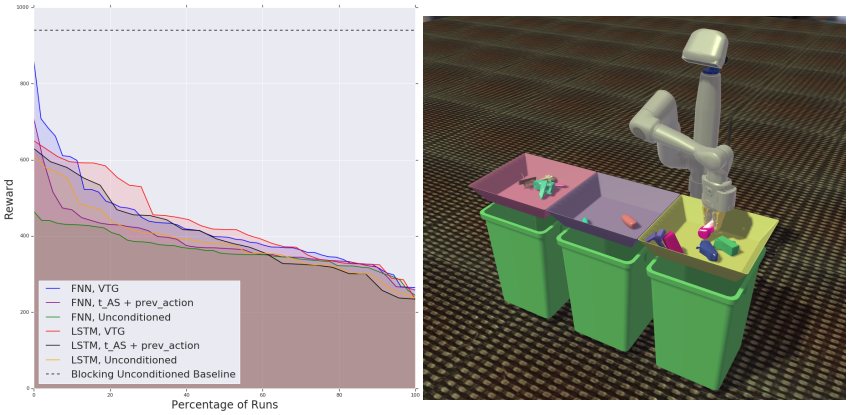
$$\mathcal{T}_c^* Q(s_t, a_{t-1}, a_t, t, t_{AS}, H) = r(s_t, a_{t-1}) + \gamma^{\frac{t_{AS}}{H}} \max_{a_{t+1}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} Q^{\pi}(s_{t+t_{AS}}, a_t, a_{t+1}, t+t_{AS}, t_{AS'}, H-t_{AS}). \tag{8}$$

See Appendix A.5 for derivation details and contraction proofs. By utilizing these concurrent Bellman operators with the standard Q-learning formulation, we can maintain Q-learning convergence guarantees [3]. We conclude that in an concurrent environment, knowledge of the previous action a_{i-1} and the action selection latency t_{AS} is sufficient for the Q-learning algorithm to converge. We describe various representations of this concurrent knowledge in A.6.

4 Experiments

We consider three additional features encapsulating *asynchronous knowledge* to condition the Q-function on: 1) Previous Action (a_{i-1}), 2) Action Selection time (t_{AS}), and 3) Vector-to-go (VTG), which we define as the remaining action to be executed at the instant state is captured.

First, we illustrate the effects of a concurrent control paradigm on value-based DRL methods through an ablation study on concurrent versions of the standard Cartpole and Pendulum environments.



(a) Cartpole

(b) Non-Blocking QT-Opt

Figure 2: (a) Environment rewards achieved by DQN with different asynchronous knowledge features on the concurrent Cartpole task for every hyperparameter in a sweep, sorted in decreasing order. The results for the concurrent Pendulum task as well as a larger version of this figure are provided in A.8. (b) An overview of the simulated robotic grasping task. A static manipulator arm attempts to grasp procedurally generated objects placed in bins front of it.

Table 1: Large-Scale Robotic Grasping Results

Blocking Actions	VTG	Previous Action	Grasp Success	Episode Duration	Action Completion
Yes	No	No	$91.53\% \pm 1.04\%$	120.81s $\pm$9.13s	$\mathbf{89.53}\% \pm 2.267\%$
No	No	No	$83.77\% \pm 9.27\%$	97.16s $\pm$6.28s	$34.69\% \pm 16.80\%$
No	Yes	No	$92.55\% \pm 4.39\%$	$\mathbf{82.98s \pm 5.74s}$	$47.28\% \pm 14.25\%$
No	No	Yes	$92.70\% \pm 1.42\%$	87.15s $\pm$4.80s	$50.09\% \pm 14.25\%$
No	Yes	Yes	$\mathbf{93.49}\% \pm \mathbf{1.04}\%$	90.75s $\pm$4.15s	$49.19\% \pm 14.98\%$

We find that utilizing concurrent knowledge representations are important across many different hyperparameter combinations. Further analysis and implementation details are described in Appendix A.7.1.

Next, we evaluate scalability of our approach to a practical robotic grasping task in simulation and the real world. The details of the setup are shown in Figure 2b and explained in Appendix A.7.2 Table 1 summarizes the performance for blocking and concurrent modes comparing unconditioned models against the asynchronous knowledge models in simulation, and Table 2 shows a similar comparison in the real world. Our results show that the asynchronous knowledge models acting in concurrent mode are able achieve comparable baseline task performance of the blocking execution unconditioned baseline in simulation, while acting much faster and smoother. The qualitative benefits of faster, smoother trajectories are drastically apparent when viewing video playback of learned policies[2]. We discuss these results further in Appendix A.7.2.

[2] `https://youtu.be/Gr2sZVwrX5w`

Table 2: Real-World Robotic Grasping Results.

Blocking Actions	VTG	Grasp Success	Policy Duration
Yes	No	$\mathbf{81.43}\%$	22.60s $\pm$12.99s
No	Yes	68.60%	$\mathbf{11.52s \pm 7.272s}$

5 Discussion and Future Work

We presented a theoretical framework to analyze concurrent systems by considering the action execution and action selection portions of the environment. Viewing this formulation through the lens of continuous-time value-based reinforcement learning, we showed that by considering asynchronous knowledge (t_{AS}, previous action, or VTG), the concurrent continuous-time and discrete-time Bellman Operators remain contractions and thus maintain standard Q-Learning convergence guarantees. Our theoretical findings were supported by experimental results on Q-learning models acting in concurrent simple control tasks as well as a complex concurrent large-scale robotic grasping task. In addition to learning successful concurrent grasping policies, the asynchronous knowledge models were able to act faster and more fluidly.

While our work focused on Value-based RL methods for both our theoretical framework and our experimental setups, the concurrent action execution paradigm is an important and understudied problem for DRL as a whole. A natural extension of this work is to evaluate different types of DRL methods, such as on-policy learning methods and policy gradient methods. In addition, the true test of concurrent methods is to attempt them in real-world settings, where robots must truly think and act at the same time.

References

[1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, editors, *NIPS*, pages 1–8. MIT Press, 2006.

[2] Artemij Amiranashvili, Alexey Dosovitskiy, Vladlen Koltun, and Thomas Brox. Motion perception in reinforcement learning with dynamic objects. In *CoRL*, volume 87 of *Proceedings of Machine Learning Research*, pages 156–168. PMLR, 2018.

[3] S. W. Carden. Convergence of a q-learning variant for continuous states and actions. *Journal of Artificial Intelligence Research*, 49:705–731, 2014.

[4] Rémi Coulom. *Reinforcement learning using neural networks, with applications to motor control*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2002.

[5] Nicolás Cruz, Kenzo Lobos-Tsunekawa, and Javier Ruiz del Solar. Using convolutional neural networks in robots with limited computational resources: Detecting nao robots while playing soccer. *CoRR*, abs/1706.06702, 2017.

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

[7] Vlad Firoiu, Tina Ju, and Joshua Tenenbaum. At Human Speed: Deep Reinforcement Learning with Action Delay. *arXiv e-prints*, October 2018.

[8] Nicolas Frémaux, Henning Sprekeler, and Wulfram Gerstner. Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS computational biology*, 9:e1003024, 04 2013.

[9] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Chris Harris, Vincent Vanhoucke, et al. Tf-agents: A library for reinforcement learning in tensorflow, 2018.

[10] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *CoRR*, abs/1806.10293, 2018.

[13] H J Kappen. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11):P11011–P11011, nov 2005.

[14] Shuang Li, Shuai Xiao, Shixiang Zhu, Nan Du, Yao Xie, and Le Song. Learning temporal point processes via reinforcement learning. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 10804–10814, USA, 2018. Curran Associates Inc.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[16] Rémi Munos and Paul Bourgine. Reinforcement learning for continuous stochastic control problems. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 1029–1035. MIT Press, 1998.

[17] Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9816–9826. Curran Associates, Inc., 2018.

[18] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[19] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[20] Sheldon M Ross, John J Kelly, Roger J Sullivan, William James Perry, Donald Mercer, Ruth M Davis, Thomas Dell Washburn, Earl V Sager, Joseph B Boyce, and Vincent L Bristow. *Stochastic processes*, volume 2. Wiley New York, 1996.

[21] Erik Schuitema, Lucian Busoniu, Robert Babuka, and Pieter P. Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3226–3231, 2010.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

[23] Correntin Tallec, Leonard Blier, and Yann Ollivier. Making Deep Q-learning Methods Robust to Time Discretization. *arXiv e-prints*, January 2019.

[24] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. *arXiv e-prints*, April 2018.

[25] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018.

[26] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. pages 2397 – 2403, 06 2010.

[27] Utkarsh Upadhyay, Abir De, and Manuel Gomez-Rodrizuez. Deep reinforcement learning of marked temporal point processes. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 3172–3182, USA, 2018. Curran Associates Inc.

[28] Eleni Vasilaki, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner. Spike-based reinforcement learning in continuous state and action space: When policy gradient methods fail. *PLoS computational biology*, 5:e1000586, 12 2009.

[29] Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. Planning and learning in environments with delayed feedback. In Joost N. Kok, Jacek Koronacki, Ramón López de Mántaras, Stan Matwin, Dunja Mladenic, and Andrzej Skowron, editors, *ECML*, volume 4701 of *Lecture Notes in Computer Science*, pages 442–453. Springer, 2007.

[30] Wenhao Yu, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *CoRR*, abs/1702.02453, 2017.

A Appendix

A.1 Related Work

Minimizing Concurrent Effects Although real-world robotics systems are inherently concurrent, it is sometimes possible to engineer them into approximately blocking systems. For example, using low-latency hardware [1] and low-footprint controllers [5] minimizes the time spent during state capture and policy inference. Another option is to design actions to be executed to completion via closed-loop feedback controllers and the system velocity is decelerated to zero before a state is recorded [12]. In contrast to these works, we tackle the concurrent action execution directly in the learning algorithm. Our approach can be applied to tasks where it is not possible to wait for the system to come to rest between deciding new actions.

Algorithmic Approaches Other works utilize algorithmic modifications to directly overcome the challenges of concurrent control. Previous work in this area can be grouped into five approaches: (1) learning policies that are robust to variable latencies [24], (2) including past history such as frame-stacking [10], (3) learning dynamics models to predict the future state at which the action will be executed [7, 2], (4) using a time-delayed MDP framework [29, 7, 21], and (5) temporally-aware architectures such as Spiking Neural Networks [28, 8], point processes [27, 14], and adaptive skip intervals [17]. In contrast to these works, our approach is able to (1) optimize for a specific latency regime as opposed to being robust to all of them, (2) consider the properties of the source of latency as opposed to force the network to learn them from high-dimensional inputs, (3) avoid learning explicit forward dynamics models in high-dimensional spaces, which can be costly and challenging, (4) consider environments where actions are interrupted as opposed to discrete-time time-delayed environments where multiple actions are queued and each action is executed until completion. The approaches in (5) show promise in enabling asynchronous agents, but are still active areas of research that have not yet been extended to high-dimensional, image-based robotic tasks.

A.2 Concurrent Action Environments

In *blocking* environments (Figure 3a in the Appendix), actions are executed in a sequential blocking fashion that assumes the environment state does not change between when state is observed and when actions are executed. This can be understood as state capture and policy inference being viewed as instantaneous from the perspective of the agent. In contrast, *concurrent* environments (Figure 3b in the Appendix) do not assume a fixed environment during state capture and policy inference, but instead allow the environment to evolve during these time segments.

A.3 Discrete-Time Reinforcement Learning Preliminaries

We use standard reinforcement learning formulations in both discrete-time and continuous-time settings [22]. In the discrete-time case, at each time step i, the agent receives state s_i from a set of possible states $\mathcal{S}$ and selects an action a_i from some set of possible actions $\mathcal{A}$ according to its policy π, where π is a mapping from $\mathcal{S}$ to $\mathcal{A}$. The environment returns the next state s_{i+1} sampled from a transition distribution $p(s_{i+1}|s_i, a_i)$ and a reward $r(s_i, a_i)$. The return for a given trajectory of states and actions is the total discounted return from time step i with discount factor $\gamma \in (0, 1]$: $R_i = \sum_{k=0}^{\infty} \gamma^k r(s_{i+k}, a_{i+k})$. The goal of the agent is to maximize the expected return from each state s_i. The Q-function for a given stationary policy π gives the expected return when selecting action $\mathbf{a}$ at state $\mathbf{s}$: $Q^\pi(s, a) = \mathbb{E}[R_i|s_i = s, a_i = a]$. Similarly, the value function gives expected return from state s: $V^\pi(s) = \mathbb{E}[R_i|s_i = s]$.

The default blocking environment formulation is detailed in Figure 1a.

A.4 Defining Blocking Bellman operators

As introduced in Section 3, we define a continuous-time Q-function estimator with concurrent actions.

$$\hat{Q}(s(t), a_{i-1}, a_i, t, H) = \int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{i-1}(t'))dt' + \tag{9}$$

$$\int_{t''=t+t_{AS}}^{t''=t+H} \gamma^{t''-t} r(s(t''), a_i(t''))dt'' + \gamma^H V(s(t+H)) \tag{10}$$

$$= \int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{i-1}(t'))dt' + \tag{11}$$

$$\gamma^{t_{AS}} \int_{t''=t+t_{AS}}^{t''=t+H} \gamma^{t''-t-t_{AS}} r(s(t''), a_i(t''))dt'' + \gamma^H V(s(t+H)) \tag{12}$$

$$= \int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{i-1}(t'))dt' + \tag{13}$$

$$\gamma^{t_{AS}} [\int_{t''=t+t_{AS}}^{t''=t+H} \gamma^{t''-t-t_{AS}} r(s(t''), a_i(t''))dt'' + \gamma^{H-t_{AS}} V(s(t+H))] \tag{14}$$

We observe that the second part of this equation (after $\gamma^{t_{AS}}$) is itself a Q-function at time $t + t_{AS}$. Since the future state, action, and reward values at $t + t_{AS}$ are not known at time t, we take the following expectation:

$$Q(s(t), a_{i-1}, a_i, t, H) = \int_{t'=t}^{t'=t+t_{AS}} \gamma^{t'-t} r(s(t'), a_{i-1}(t'))dt' + \tag{15}$$

$$\gamma^{t_{AS}} \mathbb{E}_s \hat{Q}(s(t), a_i, a_{i+1}, t + t_{AS}, H - t_{AS}) \tag{16}$$

which indicates that the Q-function in this setting is not just the expected sum of discounted future rewards, but it corresponds to an expected future Q-function.

In order to show the discrete-time version of the problem, we parameterize the discrete-time concurrent Q-function as:

$$\hat{Q}(s_t, a_{t-1}, a_t, t, t_{AS}, H) = r(s_t, a_{t-1}) + \gamma^{\frac{t_{AS}}{H}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} r(s_{t+t_{AS}}, a_t) + \tag{17}$$

$$\gamma^{\frac{H}{H}} \mathbb{E}_{p(s_{t+H}|s_{t+t_{AS}}, a_t)} V(s_{t+H}) \tag{18}$$

which with $t_{AS} = 0$, corresponds to a synchronous environment.

Using this parameterization, we can rewrite the discrete-time Q-function with concurrent actions as:

$$\hat{Q}(s_t, a_{t-1}, a_t, t, t_{AS}, H) = r(s_t, a_{t-1}) + \gamma^{\frac{t_{AS}}{H}} [\mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} r(s_{t+t_{AS}}, a_t) + \tag{19}$$

$$\gamma^{\frac{H-t_{AS}}{H}} \mathbb{E}_{p(s_{t+H}|st+t_{AS}, a_t)} V(s_{t+H})] \tag{20}$$

$$= r(s_t, a_{t-1}) + \gamma^{\frac{t_{AS}}{H}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} \hat{Q}(s_t, a_t, a_{t+1}, t + t_{AS}, t_{as'}, H - t_{AS}) \tag{21}$$

A.5 Contraction Proofs for the Blocking Bellman operators

Proof of the Discrete-time Blocking Bellman Update

Lemma A.1. *The traditional Bellman operator is a contraction, i.e.:*

$$||\mathcal{T}^* \mathcal{Q}_\infty(s, a) - \mathcal{T}^* \mathcal{Q}_\in(s, a)|| \leq c||Q_1(s, a) - Q_2(s, a)||, \tag{22}$$

where $\mathcal{T}^* \mathcal{Q}(s, a) = r(s, a) + \gamma \max_{a'} \mathbb{E}_p Q(s', a')$ *and* $0 \leq c \leq 1$.

Proof. In the original formulation, we can show that this is the case as following:

$$\mathcal{T}^* \mathcal{Q}_1(s, a) - \mathcal{T}^* \mathcal{Q}_2(s, a) \tag{23}$$

$$= r(s, a) + \gamma \max_{a'} \mathbb{E}_p[Q_1(s', a')] - r(s, a) - \gamma \max_{a'} \mathbb{E}_p[Q_2(s', a')] \tag{24}$$

$$= \gamma \max_{a'} \mathbb{E}_p[Q_1(s', a') - Q_2(s', a')] \tag{25}$$

$$\leq \gamma \sup_{s', a'} [Q_1(s', a') - Q_2(s', a')], \tag{26}$$

with $0 \leq \gamma \leq 1$ and $||f||_\infty = \sup_x [f(x)]$. □

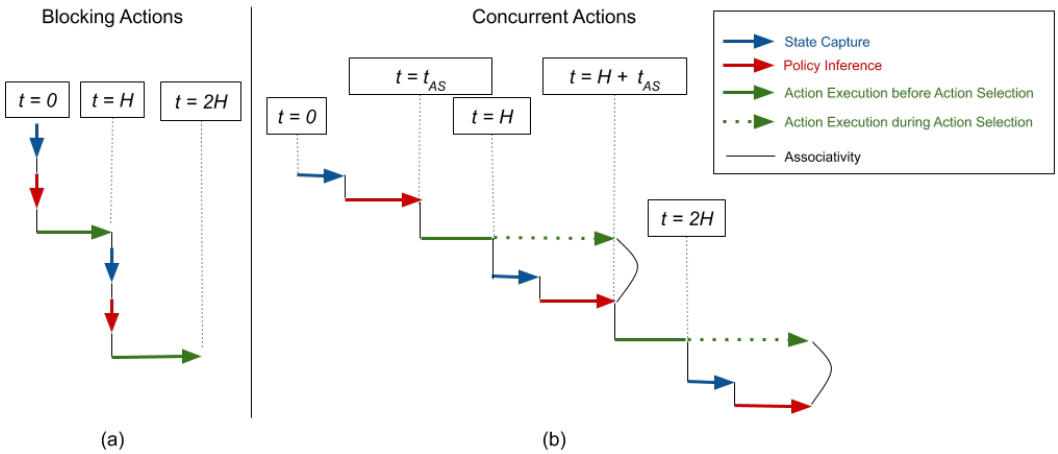


Figure 3: The execution order of different stages are shown relative to the sampling period H as well as the latency t_{AS}. **(a)**: In "blocking" environments, state capture and policy inference are assumed to be instantaneous. **(b)**: In "concurrent" environments, state capture and policy inference are assumed to proceed concurrently to action execution.

Similarly, we can show that the updated Bellman operators introduced in 3 are contractions as well.

Lemma A.2. *The concurrent discrete-time Bellman operator is a contraction.*

Proof of Lemma A.2

Proof.

$$\mathcal{T}_c^* \mathcal{Q}_1(s_t, a_{i-1}, a_i, t, t_{AS}, H) - \mathcal{T}_c^* \mathcal{Q}_2(s_t, a_{i-1}, a_i, t, t_{AS}, H) \tag{27}$$

$$= r(s_t, a_{i-1}) + \gamma^{\frac{t_{AS}}{H}} \max_{a_{i+1}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} Q_1(s_t, a_i, a_{i+1}, t + t_{AS}, t_{AS'}, H - t_{AS}) \tag{28}$$

$$- r(s_t, a_{i-1}) - \gamma^{\frac{t_{AS}}{H}} \max_{a_{i+1}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{t-1})} Q_2(s_t, a_i, a_{i+1}, t + t_{AS}, t_{AS'}, H - t_{AS}) \tag{29}$$

$$= \gamma^{\frac{t_{AS}}{H}} \max_{a_{i+1}} \mathbb{E}_{p(s_{t+t_{AS}}|s_t, a_{i-1})} [Q_1(s_t, a_i, a_{i+1}, t + t_{AS}, t_{AS'}, H - t_{AS}) - Q_2(s_t, a_i, a_{i+1}, t + t_{AS}, t_{AS'}, H - t_{AS})] \tag{30}$$

$$\leq \gamma^{\frac{t_{AS}}{H}} \sup_{s_t, a_i, a_{i+1}, t+t_{AS}, t_{AS'}, H-t_{AS}} [Q_1(s_t, a_i, a_{i+1}, t + t_{AS}, t_{AS'}, H - t_{AS}) - Q_2(s_t, a_i, a_{i+1}, t + t_{AS}, t_{AS'}, H - t_{AS})] \tag{31}$$

□

Lemma A.3. *The concurrent continuous-time Bellman operator is a contraction.*

Proof of Lemma A.3

Proof. To prove that this the continuous-time Bellman operator is a contraction, we can follow the discrete-time proof, from which it follows:

$$\mathcal{T}_c^* \mathcal{Q}_1(s(t), a_{i-1}, a_i, t, t_{AS}) - \mathcal{T}_c^* \mathcal{Q}_2(s(t), a_{i-1}, a_i, t, t_{AS}) \tag{32}$$

$$= \gamma^{t_{AS}} \max_{a_{i+1}} \mathbb{E}_p [Q_1(s(t), a_i, a_{i+1}, t + t_{AS}, H - t_{AS}) - Q_2(s(t), a_i, a_{i+1}, t + t_{AS}, H - t_{AS})] \tag{33}$$

$$\leq \gamma^{t_{AS}} \sup_{s(t), a_i, a_{i+1}, t+t_{AS}, H-t_{AS}} [Q_1(s(t), a_i, a_{i+1}, t + t_{AS}, H - t_{AS}) - Q_2(s(t), a_i, a_{i+1}, t + t_{AS}, H - t_{AS})] \tag{34}$$

□

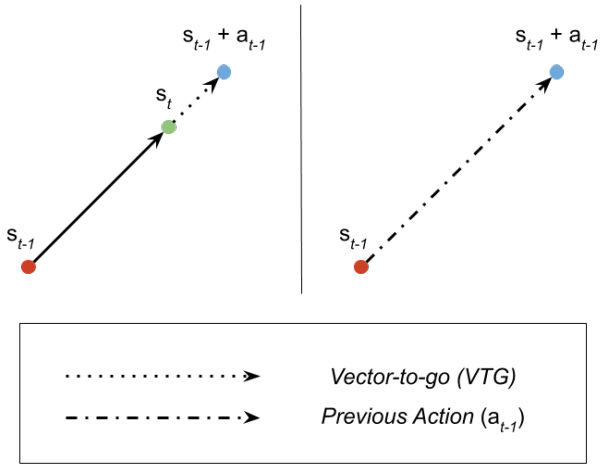


Figure 4: Concurrent knowledge representations can be visualized through an example of a 2-D pointmass discrete-time toy task. Vector-to-go represents the remaining action that may be executed when the current state s_t is observed. Previous action represents the full commanded action from the previous timestep.

A.6 Concurrent Knowledge Representation

While we have shown that knowledge of the concurrent system properties (t_{AS} and a_{t-1}, as defined previously for the discrete-time case) is theoretically sufficient, it is often hard to accurately predict t_{AS} during inference on a complex robotics system. In order to allow practical implementation of our algorithm on a wide range of RL agents, we consider three additional features encapsulating *concurrent knowledge* used to condition the Q-function: (1) Previous action (a_{t-1}), (2) Action selection time (t_{AS}), and (3) Vector-to-go (VTG), which we define as the remaining action to be executed at the instant the state is measured. We limit our analysis to environments where a_{t-1}, t_{AS}, and VTG are all obtainable and H is held constant. Previous action a_{t-1} is the action that the agent executed at the previous timestep. Action selection time t_{AS} is a measure of how long action selection takes, which can be represented as either a categorical or continuous variable; in our experiments, which take advantage of a bounded latency regime, we normalize action selection time using these known bounds. Vector-to-go *VTG* is a feature that combines a_{t-1} and s_t by encoding the remaining amount of a_{t-1} left to execute. See Figure 4 for a visual comparison.

We note that a_{t-1} is available across the vast majority of environments and it is easy to obtain. Using t_{AS}, which encompasses state capture, communication latency, and policy inference, relies on having some knowledge of the concurrent properties of the system. Calculating VTG requires having access to some measure of action completion at the exact moment when state is observed. When utilizing a first-order control action space, such as joint angle or desired pose, VTG is easily computable if proprioceptive state is measured and synchronized with state observation. In these cases, VTG is an alternate representation of the same information encapsulated by a_{t-1} and the current state.

A.7 Experiment Results and Implementation Details

A.7.1 Cartpole and Pendulum Ablation Studies

Here, we describe the results implementation details of the toy task Cartpole and Pendulum experiments in Section 4.

To estimate the relative importance of different asynchronous knowledge representations, we conduct an analysis of the sensitivity of each type of asynchronous knowledge representations to combinations of the other hyperparameter values, shown in Figure 2a. While all combinations of asynchronous knowledge representations increase learning performance over baselines that do not leverage this information, the clearest difference stems from including VTG. In Figure 6, we conduct a similar analysis but on a Pendulum environment where t_{AS} is fixed every environment; thus, we do not focus on t_{AS} for this analysis but instead compare the importance of VTG with frame-stacking previous actions and observations. While frame-stacking helps nominally, the majority of the performance increase results from utilizing information from VTG.

For the environments, we use the 3D MuJoCo implementations of the `Cartpole-Swingup` and `Pendulum-Swingup` tasks in DeepMind Control Suite [25]. We use discretized action spaces for first-order control of joint position actuators. For the observation space of both tasks, we use the default state space of ground truth positions and velocities.

For the baseline learning algorithms, we use the TensorFlow Agents [9] implementations of a Deep Q-Network agent, which utilizes a Feed-forward Neural Network (FNN), and a Deep Q-Recurrent Neutral Network agent, which utilizes a Long Short-Term Memory (LSTM) network. Learning parameters such as `learning_rate`, `lstm_size`, and `fc_layer_size` were selected through hyperparameter sweeps.

To approximate different difficulty levels of latency in concurrent environments, we utilize different parameter combinations for action execution steps and action selection steps (t_{AS}). The number of action execution steps is selected from {0ms, 5ms, 25ms, or 50ms} once at environment initialization. t_{AS} is selected from {0ms, 5ms, 10ms, 25ms, or 50ms} either once at environment initialization or repeatedly at every episode reset. The selected t_{AS} is implemented in the environment as additional physics steps that update the system during simulated action selection.

Frame-stacking parameters affect the observation space by saving previous observations and actions. The number of previous actions to store as well as the number of previous observations to store are independently selected from the range $[0, 4]$. Concurrent knowledge parameters, as described in Section 4, include whether to use VTG and whether to use t_{AS}. Including the previous action is already a feature implemented in the frame-stacking feature of including previous actions. Finally, the number of actions to discretize the continuous space to is selected from the range $[3, 8]$.

A.7.2 Large Scale Robotic Grasping

Simulated Environment We simulate a 7 DoF arm with an over-the-shoulder camera (see Figure 2ba). A bin in front of the robot is filled with procedurally generated objects to be picked up by the robot and a sparse binary reward is assigned if an object is lifted off a bin at the end of an episode. We train a policy with QT-Opt [12], a deep Q-Learning method that utilizes the cross-entropy method (CEM) to support continuous actions. In blocking mode, a displacement action is executed until completion: the robot uses a closed-loop controller to fully execute an action, decelerating and coming to rest before observing the next state. In concurrent mode, an action is triggered and executed without waiting, which means that the next state is observed while the robot remains in motion. States are represented in form of RGB images and actions are continuous Cartesian displacements of the gripper 3D positions and yaw. In addition, the policy commands discrete gripper open and close actions and may terminate an episode. In blocking mode, a displacement action is executed until completion: the robot uses a closed loop controller to fully execute an action, decelerating and coming to rest before observing the next state. In concurrent mode, an action is triggered and executed without waiting, which means that the next state is observed while the robot remains in motion. It should be noted that in blocking mode, action completion is close to 100% unless the gripper moves are blocked by contact with the environment or objects; this causes average blocking mode action completion to be lower than 100%, as seen in Table 1.

Table 1 summarizes the performance for blocking and concurrent modes comparing unconditioned models against the concurrent knowledge models described in Section A.6. Our results indicate that the VTG model acting in concurrent mode is able to recover baseline task performance of the blocking execution unconditioned baseline, while the unconditioned baseline acting in concurrent model suffers some performance loss. In addition to the success rate of the grasping policy, we also evaluate the speed and smoothness of the learned policy behavior. Concurrent knowledge models are able to learn faster trajectories: *episode duration*, which measures the total amount of wall-time used for an episode, is reduced by 31.3% when comparing concurrent knowledge models with blocking unconditioned models, even those that utilize a shaped *timestep penalty* that reward faster policies.

When switching from blocking execution mode to concurrent execution mode, we see a significantly lower *action completion*, measured as the ratio from executed gripper displacement to commanded displacement, which expectedly indicates a switch to a concurrent environment. The concurrent knowledge models have higher action completions than the unconditioned model in the concurrent environment, which suggests that the concurrent knowledge models are able to utilize more efficient motions, resulting in smoother trajectories. The qualitative benefits of faster, smoother trajectories are drastically apparent when viewing video playback of learned policies 2.

Real robot results In addition, we evaluate qualitative policy behaviors of concurrent models compared to blocking models on a real-world robot grasping task, which is shown in Figure 2bb. As seen in Table 2, the models achieve comparable grasp success, but the concurrent model is 49% faster than the blocking model in terms of *policy duration*, which measures the total execution time of the policy (this excludes the infrastructure setup and teardown times accounted for in episode duration, which can not be optimized with concurrent actions). In addition, the concurrent VTG model is able to execute smoother and faster trajectories than the blocking unconditioned baseline, which is clear in video playback[2].

Algorithm We train a policy with QT-Opt [12], a Deep Q-Learning method that utilizes the Cross-Entropy Method (CEM) to support continuous actions. A Convolutional Neural Network (CNN) is trained to learn the Q-function conditioned on an image input along with a CEM-sampled continuous control action. At policy inference time, the agent sends an image of the environment and batches of CEM-sampled actions to the CNN Q-network. The highest-scoring action is then used as the policy's selected action. Compared to the formulation in **(author?)** [12], we also add a *concurrent knowledge* feature of VTG and/or previous action a_{t-1} as additional input to the Q-network. Algorithm 1 shows the modified QT-Opt procedure.

Algorithm 1: QT-Opt with Concurrent Knowledge

Initialize replay buffer D;
Initialize random start state and receive image o_0;
Initialize concurrent knowledge features $c_0 = [VTG_0 = 0, a_{t-1} = 0, t_{AS} = 0]$;
Initialize environment state $s_t = [o_0, c_0]$;
Initialize action-value function $Q(s, a)$ with random weights θ;
Initialize target action-value function $\hat{Q}(s, a)$ with weights $\hat{\theta} = \theta$;
while *training* **do**
  for *t = 1, T* **do**
    Select random action a_t with probability ϵ, else $a_t = \mathbf{CEM}(Q, s_t; \theta)$;
    Execute action in environment, receive o_{t+1}, c_t, r_t;
    Process necessary concurrent knowledge features c_t, such as VTG_t, a_{t-1}, or t_{AS};
    Set $s_{t+1} = [o_{t+1}, c_t]$;
    Store transition (s_t, a_t, s_{t+1}, r_t) in D;
    if *episode terminates* **then**
      Reset s_{t+1} to a random reset initialization state;
      Reset c_{t+1} to 0;
    end
    Sample batch of transitions from D;
    for *each transition* (s_i, a_i, s_{i+1}, r_i) *in batch* **do**
      if *terminal transition* **then**
        $y_i = r_i$;
      else
        Select $\hat{a}_{i+1} = \mathbf{CEM}(\hat{Q}, s_i; \hat{\theta})$;
        $y_i = r_i + \gamma \hat{Q}(s_{i+1}, \hat{a}_{i+1})$;
      end
      Perform **SGD** on $(y_i - Q(s_i, a_i; \theta)^2$ with respect to θ;
    end
    Update target parameters $\hat{Q}$ with Q and θ periodically;
  end
end

For simplicity, the algorithm is described as if run synchronously on a single machine. In practice, episode generation, Bellman updates and Q-fitting are distributed across many machines and done asynchronously; refer to [12] for more details. Standard DRL hyperparameters such as random exploration probability (ϵ), reward discount (γ), and learning rate are tuned through a hyperparameter sweep. For the time-penalized baselines in Table 1, we manually tune a *timestep penalty* that returns a fixed negative reward at every timestep. Empirically we find that a timestep penalty of -0.01, relative to a binary sparse reward of 1.0, encourages faster policies. For the non-penalized baselines, we set a timestep penalty of -0.0.

A.8 Figures

See Figure 5 and Figure 6.

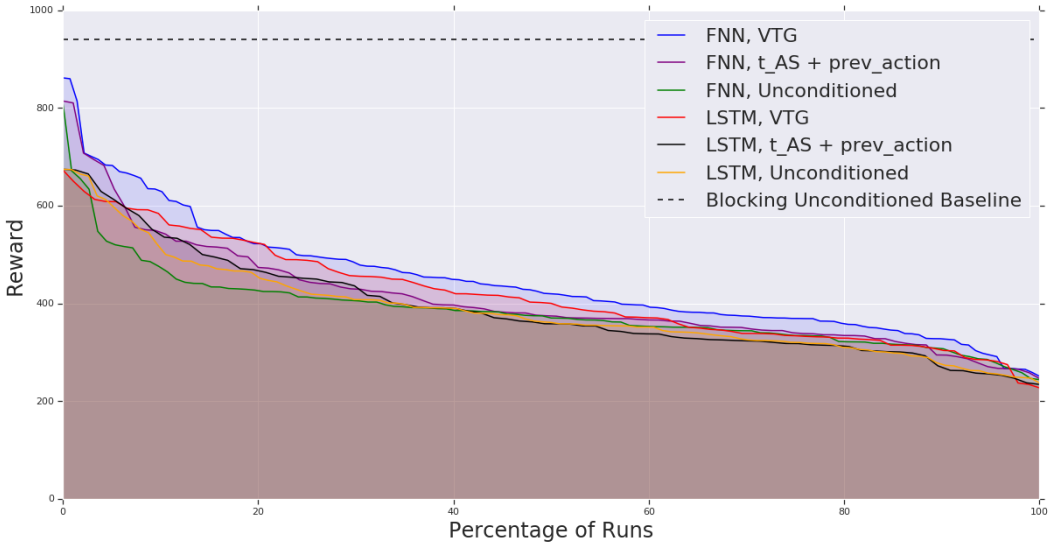


Figure 5: Environment rewards achieved by DQN with different network architectures [either a feedforward network (FNN) or a Long Short-Term Memory (LSTM) network] and different concurrent knowledge features [Unconditioned, vector-to-go (VTG), or previous action and t_{AS}] on the concurrent Cartpole task for every hyperparameter in a sweep, sorted in decreasing order. Providing the critic with VTG information leads to more robust performance across all hyperparameters. This figure is a larger version of 2a.

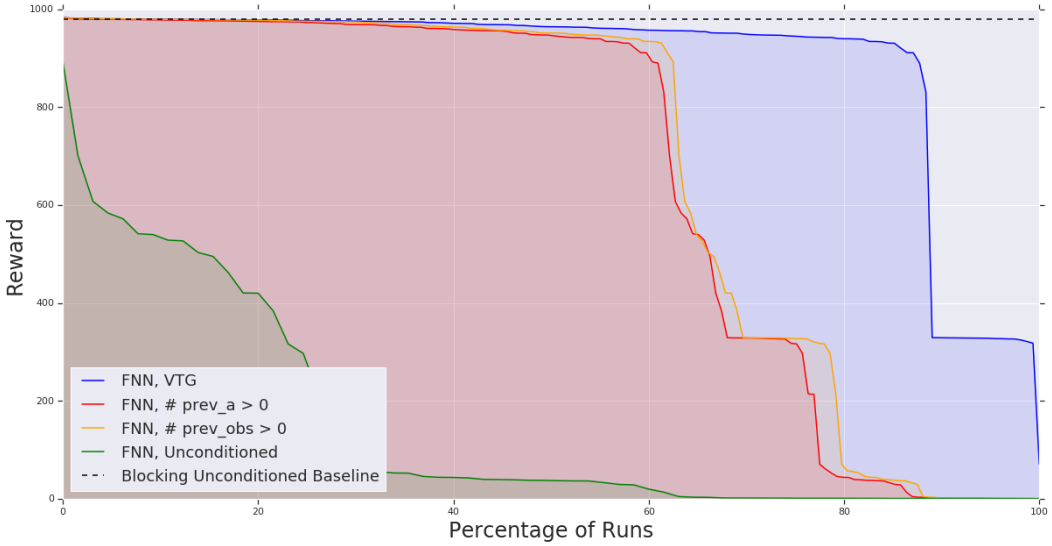


Figure 6: Environment rewards achieved by DQN with a FNN and different frame-stacking and concurrent knowledge parameters on the concurrent Pendulum task for every hyperparameter in a sweep, sorted in decreasing order.