Exact (Then Approximate) Dynamic Programming for Deep Reinforcement Learning

Henrik Marklund[*1] **Suraj Nair**[*1] **Chelsea Finn**[1]

Abstract

Model-free reinforcement learning methods such as Q-learning and actor-critic methods have shown considerable success on a variety of problems. However, when combined with function approximation, these methods are notoriously brittle, and often face instability during training. At the heart of these methods is dynamic programming, where a value function is trained by bootstrapping values off itself with a temporal difference loss. As a result, these bootstrapped values may be incorrect, non-stationary, or even divergent, during a substantial portion of training. Such training conditions pose a unique challenge for deep networks, which have been shown to require more data and generalize poorly when trained with noisy labels. Motivated by these challenges, we propose a simple technique to stabilize deep Q learning by decoupling dynamic programming and function approximation. To do so, we quantize the agent's experience, run value iteration on the discrete graph, train a neural network policy via supervised learning on the tabular targets, and finally finetune the policy with deep Q learning. We observe that, across a range of challenging image based offline RL tasks, this approach consistently outperforms prior approaches such as double DQN and BCQ, which often diverge or fail completely, while generalizing more effectively than directly applying the tabular policy.

1 Introduction

Deep reinforcement learning is a powerful framework for automatically acquiring complex behavior, achieving strong performance in many games (Mnih et al., 2013b), and showing promising performance in domains such as robotics (Levine et al., 2016). One particularly effective and popular class of approaches for deep reinforcement learning are value-based methods, which have favorable properties such as sample efficiency and the ability to leverage off-policy data. These value-based methods most often utilize dynamic programming, where a value function is trained by bootstrapping values off itself with a temporal difference loss. Despite their promise, methods that combine dynamic programming with function approximation (i.e. approximate dynamic programming (ADP)) are known to be unstable and sensitive to hyper-parameters.

The primary contribution of this work is a deep reinforcement learning algorithm, exact then approximate dynamic programming (ETA-DP), that decouples bootstrapping and function approximation by sequentially applying discretization, value iteration, supervised learning, and then deep Q learning with function approximation. Unlike ADP methods, the first three steps of this approach are guaranteed to converge, though the quality of the converged solution depends on the quality of the quantization relative to the underlying MDP, as well as the loss landscape of the supervised optimization problem. We empirically find that ETA-DP exhibits strong performance on vision-based offline RL tasks, avoiding instabilities experienced by double DQN (Van Hasselt et al., 2016) and BCQ (Fujimoto et al., 2019b), while generalizing more effectively than directly applying the tabular solution.

2 Exact (Then Approximate) Dynamic Programming (ETA-DP)

To address the stability challenges associated with approximate dynamic programming, we propose exact (then approximate) dynamic programming (ETA-DP) as a technique for deep reinforcement learning. Our key insight is that by separating the bootstrapping and function approximation stages, we can maintain stability, while still leveraging the generalization power of deep neural networks.

Our method has four steps shown in Algorithm 1. **First**, we begin by quantizing the agents experience into discrete states, giving us a corresponding discrete state and next state for each transition in $\mathcal{D}$. **Second**, we run value iteration on these discretized transitions until convergence. **Third**, we use these tabular values to label every transition in the

[*]Equal contribution [1]Stanford University. Correspondence to: Henrik Marklund <marklund@stanford.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020.

original dataset $\mathcal{D}$ with an estimated Q value, which we then regress to directly using supervised learning with a function approximator. **Fourth**, we use a combination of supervised regression and DDQN to fine-tune the learned Q-function. Note that the first two steps corresponds exactly to prior approaches that use state quantization for reinforcement learning (Asada et al., 1996; 1998; Santamaria et al., 1997; Stone and Sutton, 2001; Corneil et al., 2018; Krose and Van Dam, 1995), while the latter two steps allow for further generalization. Further, the fourth step in particular can help overcome inaccuracies caused by the approximations made in the first two steps. We now go over each step in detail.

Algorithm 1 ETA-DP($\mathcal{D}, p_{min}$)

1: Initialize $N = |\mathcal{D}|$, Batch size B, Iterations I
2: /* Learn quantization function from data */
3: $\phi \leftarrow$ QUANTIZE($\mathcal{D}$)
4: **for** $n = 1, 2, ..., N$ **do**
5: $(s_t, a_t, s_{t+1}, r_t)_n = \mathcal{D}_n$
6: $x_t, x_{t+1} = \phi(s_t), \phi(s_{t+1})$
7: /* Store discrete states with transitions */
8: $\bar{\mathcal{D}}_n \leftarrow (s_t, a_t, s_{t+1}, r_t, x_t, x_{t+1})_n$
9: **end for**
10: /* Run value iteration on quantized dataset */
11: $\hat{Q}(x, a) \leftarrow$ VALUEITERATION($\bar{\mathcal{D}}$)
12: **for** (I iterations) & ($p : 1 \rightarrow p_{min}$) **do**
13: $(s_t, a_t, s_{t+1}, r_t, x_t, x_{t+1})_{1:B} \sim \bar{\mathcal{D}}$
14: **if** $\mathcal{U}(0, 1) < p$ **then**
15: /* Regress Tabular Values */
16: $L = (Q_\theta(s_t, a_t) - \hat{Q}(x_t, a_t))^2$
17: **else**
18: /* DDQN Update */
19: $y = r_t + \gamma Q_{\bar{\theta}}(s_{t+1}, \max_a Q_\theta(s_{t+1}, a))$
20: $L = (Q_\theta(s_t, a_t) - y)^2$
21: **end if**
22: Update θ using $\nabla_\theta L$
23: **end for**

2.1 Quantizing States

Concretely, we begin with a dataset of N transitions $(s_t, a_t, s_{t+1}, r_t)_i \sim \mathcal{D}$ collected by some behavior policy. To quantize states, we learn a function $\phi : \mathcal{S} \rightarrow \mathcal{X}$, where $|\mathcal{X}| <= K$. Once learned, this quantizing function $x_t = \phi(s_t)$ can be used to map each state in the dataset of transitions $(s_t, a_t, s_{t+1}, r_t) \rightarrow (s_t, a_t, s_{t+1}, r_t, x_t, x_{t+1})$ to augment the original transitions with the discretized states (Algorithm 1, L3:8).

There are a number of choices for the function ϕ ranging from simple techniques like state thresholding and binning, to learning based approaches like K-means and tile coding, as well as deep parametric approaches like VQ-VAE (van den Oord et al., 2017) and Deep Cluster (Caron et al., 2018), and more sophisticated approaches that leverage dynamics (Corneil et al., 2018). In this work, we aim for simplicity in this step, and find K-means clustering in $\mathcal{S}$ to be effective, even for high-dimensional visual inputs. In Section 3.2, we also find the overall method to be robust to the choice of K, even across one order of magnitude.

We find that Batch K-means (Sculley, 2010) is able to scale to high-dimensional inputs and reasonably large datasets (50000+ images). Furthermore, more sophisticated clustering algorithms or generative models, such as DeepCluster or VQ-VAE, scale well and can be applied to larger datasets.

2.2 Value Iteration on Discrete Graph

Given the new discretized replay buffer $[(s_t, a_t, s_{t+1}, r_t, x_t, x_{t+1})_1, ...]$ we can now run value iteration on the discrete graph. Concretely, we iterate through transitions in the dataset, setting the Q value of a (state, action) pair to the average target value until converged, as described in Algorithm 2 (In supplement).

Note that since the dataset is fixed, we are effectively doing value iteration in the discrete graph with the transition probabilities estimated with the empirical transition probabilities. As a result, we can directly compute the average value over next states, and are guaranteed to converge (Chapter 4.4 of (Sutton and Barto, 2018)) without the need for a step size parameter as in tabular Q learning. Once value iteration has been run to convergence, we now have state action values $\hat{Q}(\phi(s_t), a)$ for all state-action pairs in the dataset, which we refer to as "tabular values". One benefit of using value iteration here is that it only considers actions seen in the dataset, and thus does not suffer from overestimation of out of distribution actions, a common problem in offline RL (Levine et al., 2020).

2.3 Supervised Regression of Tabular Values

Given the new replay buffer augmented with tabular values derived from value iteration on the discrete graph, we simply train our parametric Q function $Q_\theta(s_t, a_t)$ to match the corresponding $\hat{Q}(\phi(s_t), a_t)$ via supervised learning. Concretely, instead of optimizing the Bellman loss as in default DQN, we now minimize $\mathbb{E}_{(s_t, a_t, r_t, s_{t+1}, x_t, x_{t+1}) \sim \bar{\mathcal{D}}}[(Q_\theta(s_t, a_t) - \hat{Q}(x_t, a_t))^2]$, where $x_t = \phi(s_t)$ (Algorithm , L15) and Q_θ is parametrized as a deep neural network. Since SGD is guaranteed to converge to a stationary point[1] (Ghadimi and Lan, 2013), these first 3 steps of ETA-DP (quantization, value iteration, regression) are guaranteed to converge.

2.4 Approximate Dynamic Programming Finetuning

In Section 3.1, we will find that the first three steps already produce stable and performant learning on complex problems. However, to reduce our dependence on the accuracy of the initial quantization, we add a fourth and final step, which corresponds to fine-tuning with bootstrapped targets. In particular, we slowly incorporate bootstrapped target values (standard double DQN Bellman loss (Algorithm 1, L18:19)) into the training process, mixed with supervised

[1] Under the necessary smoothness/noise/step-size assumptions

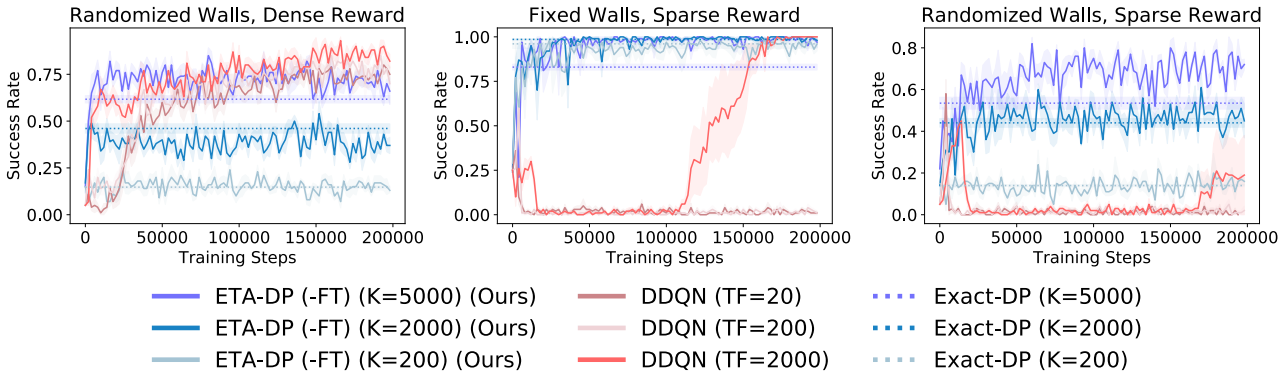

Figure 1: **Experiment 1 (maze navigation from images).** Compares the success rate of ETA-DP (-FT), DDQN, and Exact-DP on the visual navigation task over the course of offline training. In sparse reward settings (**middle/right**) DDQN is highly unstable, while ETA-DP (-FT) consistently achieves higher success rates. In the hardest setting (**right**), by using an expressive neural network ETA-DP (-FT) is able to outperform Exact-DP. Results are averaged over 5 seeds, each using a randomly generated dataset of 1000 episodes.

targets defined previously. In practice, the targets are the tabular values with probability p and are the DDQN targets with probability $1-p$, where we anneal p from 1 to $p_{min} = 0.1$ over the course of training.

3 Experiments

In our experiments we investigate two central questions. **(1)** By simply running supervised learning on the tabular values derived from the discrete graph, does ETA-DP provide increased stability and efficiency compared to deep Q learning? **(2)** By combining ETA-DP with deep Q learning finetuning, are we able to get "the best of both worlds", that is, good asymptotic performance while maintaining the stability and efficiency of discrete value iteration, even in settings where state quantization may be difficult? To answer these questions, we compare ETA-DP (with and without DDQN finetuning) to existing ADP algorithms in the offline setting, across both low dimensional and image based tasks.

Experimental Domains. Our experiments study problems with continuous states and discrete actions. Specifically, we test on the CartPole environment from (Brockman et al., 2016), a top-down visual maze navigation task with randomized walls from (Nair and Finn, 2020), and visual, egocentric navigation to a target object from (Chevalier-Boisvert, 2018). All experiments are done in the offline RL setting. That is, all methods are trained using a batch of offline data collected from a behavior policy π_b (which may be partially-learned or random), and are evaluated in the environment without collecting any new data for learning. Environment details (including visuals and qualitative results) for all environments and the corresponding behavior policies can be found in the supplement.

Comparisons. Our experiments compare the following methods. **ETA-DP:** Our method as presented in Section 2, which quantizes the agent's experience, runs value iteration, does supervised regression of the tabular targets, and finetunes with DDQN. **ETA-DP (-FT):** Identical to ETA-DP, but does not perform DDQN finetuning, only performing supervised regression on the tabular target values. **Exact-DP:** Identical to ETA-DP, but without either the supervised learning or finetuning steps. Given a new state, we map it directly to its discrete state, and use the tabular Q-function for action selection, effectively performing a "nearest neighbors" lookup. Note that this approach is not learning a parametrized Q-function, and as a result cannot generalize beyond the (discrete state, action) pairs it has seen. In the rare event that the nearest neighbor has no action values, the agent takes a random action. **DDQN:** We compare to double deep Q learning (Van Hasselt et al. (2016)) (DDQN). DDQN uses the max over the current Q function when computing target network values, reducing overestimation. **BCQ:** We also compare to batch constrained deep Q Learning (BCQ) (Fujimoto et al., 2019b), an offline RL algorithm which constrains the learned policy to be similar to what it saw in the data. All hyper-parameters and architecture details for all methods including our own can be found in the supplement. All experiments show task performance (and standard error shading) every 2000 training iterations, run over either 3/5 seeds, where each seed affects both the dataset and training.

3.1 Experiment 1: Does ETA-DP (Without Finetuning) Outperform Deep Q Learning?

In this experiment, we explore if supervised learning on tabular values achieves improved stability and performance compared to deep Q learning. To do so we compare ETA-DP without finetuning (ETA-DP (-FT)) to DDQN with varying target frequencies (TF) (how frequently the target weights are updated), as well as the ablation (Exact-DP) which directly applies the learned discrete Q function for control. DDQN with large TF also closely resembles Neural Fitted Q Iteration (Riedmiller, 2005). **Cartpole.** Results on the Carpole env can be found in the supplement.

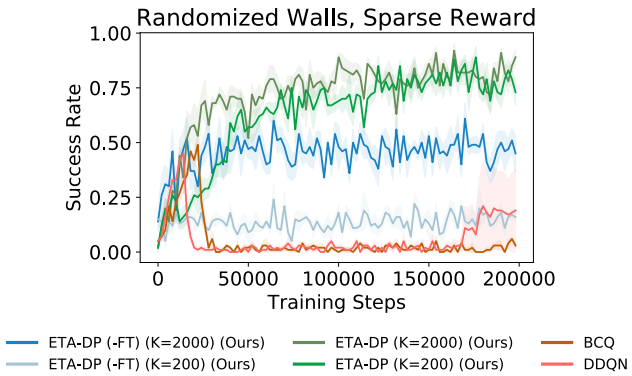

Figure 2: **Experiment 2 (maze navigation from images).** Compares ETA-DP to ETA-DP (-FT) as well as DDQN and BCQ in the visual maze navigation task.

Image Based Maze Navigation. In Figure 1 we present a comparison between ETA-DP (-FT), Double DQN (DDQN), and Exact-DP on a visual maze navigation task with either dense/sparse reward or fixed/randomized walls. We run varying numbers of clusters for ETA-DP (-FT) and Exact-DP, and varying target update frequencies for DDQN.

First, we observe that in the easier dense reward setting (**left**) DDQN is stable, and as a result matches the performance of ETA-DP (-FT). However in the challenging sparse reward setting (**middle/right**) DDQN often fails completely, while ETA-DP (-FT) and Exact-DP solve the task. Second, we observe that for ETA-DP (-FT) that large numbers of clusters tend to perform better, as too few clusters leads to over-discretizing the state space, i.e. grouping different images into the same cluster, which naturally degrades performance. Finally, we see that for fewer clusters [200, 2000], using Exact-DP generally performs the same as ETA-DP (-FT), as most state action tuples will have directly been seen in the dataset. However Exact-DP's performance improvement degrades with more clusters, while ETA-DP (-FT) is able to generalize well to unseen states, and as a result achieves the best performance on the most challenging setting (**right**).

These experiments provide evidence that supervised learning on tabular targets inferred by value iteration on the discretized graph can produce much more stable learning, and in some cases, achieve better performance, despite minimal hyper parameter tuning. We observe empirically that the source of DDQN instability is overestimation (plot in supplement), which is not an issue for ETA-DP.

3.2 Experiment 2: Does Combining Regression of Tabular Values and Deep Q Learning Finetuning Provide Both Asymptotic Performance and Stability?

In this experiment we investigate question **(2)**, using our full ETA-DP method (with all 4 steps). To do so we compare our full ETA-DP method with DDQN (target freqency

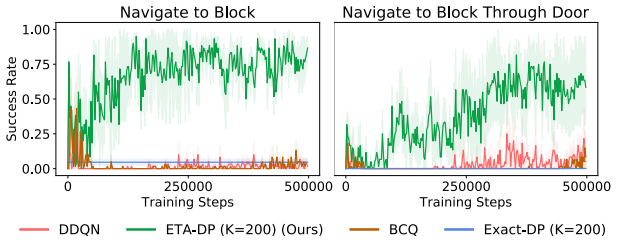

Figure 3: **Experiment 2 (egocentric visual navigation).** Success rates on the challenging egocentric navigation tasks. ETA-DP achieves strong performance, while Exact-DP, DDQN and BCQ fail to learn at all.

2000) and BCQ as points of comparison, and in some cases include ETA-DP (-FT) or Exact-DP as an ablation. In all experiments, ETA-DP uses identical hyper-parameters to the best performing DDQN.

Image Based Maze Navigation. In Figure 2 we compare ETA-DP, ETA-DP (-FT), DDQN, and BCQ on the most challenging of the top-down maze navigation tasks, namely with sparse rewards and randomized wall configurations. We make two key observations. First, we find that by finetuning with DDQN, ETA-DP is able to significantly exceed the performance of both ETA-DP (-FT) as well as DDQN and BCQ. Second, we observe that although the performance of ETA-DP (-FT) is quite different depending on the choice of K, this gap is closed when doing DDQN finetuning. In other words, even with a suboptimal K and thus a suboptimal quantization, combining tabular values with DDQN can yield substantial performance benefits.

Egocentric Visual Navigation. Lastly, we study the performance of ETA-DP on two challenging egocentric navigation tasks, built on the MiniWorld (Chevalier-Boisvert, 2018) environment (Figure 3). This environment is especially challenging for quantization, as the observations are first-person images. In the first task, the agent must navigate from a random initialization on one side of the room to a blue block on the other side of the room. The second task is identical to the first, except there is also a wall in the middle of the room with a doorway through which the agent must navigate to reach the target block. All agents are trained over 3 datasets of 300 episodes each, all collected by a random policy.

Again we observe that ETA-DP significantly outperforms the comparisons, succeeding most of the time, while Exact-DP fails, and DDQN and BCQ struggle to learn at all. These settings are especially challenging for DDQN and BCQ, since the offline dataset collected is by a random behavior policy. Random data is known to cause challenges for offline RL (Fu et al., 2020), such as overestimation on unseen actions. This overestimation issue is mitigated by ETA-DP by training on targets found from value iteration, since value iteration only considers actions seen in the data.

References

D. Abel, D. E. Hershkowitz, and M. L. Littman. Near optimal behavior via approximate state abstraction. In *ICML*, 2016.

J. Achiam, E. Knight, and P. Abbeel. Towards characterizing divergence in deep q-learning. *ArXiv*, abs/1903.08894, 2019.

M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning 23, 279–303*, 1996.

M. Asada, S. Noda, and K. Hosoda. Action-based sensor space segmentation for soccer robot learning. *Applied Artificial Intelligence*, 12:149–164, 03 1998. doi: 10.1080/088395198117802.

K. Asadi, D. J. Abel, and M. L. Littman. Learning state abstractions for transfer in continuous control. *ArXiv*, abs/2002.05518, 2020.

L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

O. Biza, R. W. Platt, J.-W. van de Meent, and L. L. Wong. Learning discrete state abstractions with deep variational inference. *ArXiv*, abs/2003.04300, 2020.

M. Brittain, J. Betram, X. Yang, and P. Wei. Prioritized sequence experience replay. https://arxiv.org/pdf/1905.12726.pdf,, 2020.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

M. Chevalier-Boisvert. gym-miniworld environment for openai gym. https://github.com/maximecb/gym-miniworld, 2018.

D. S. Corneil, W. Gerstner, and J. Brea. Efficient model-based deep reinforcement learning with variational state tabulation. In *ICML*, 2018.

J. del R. Millán, D. Posenato, and E. Dedieu. Continuous-action q-learning, year = 2002, journal = Machine Learning,.

S.-H. Dong, B. V. Roy, and Z. Zhou. Provably efficient reinforcement learning with aggregated states. *ArXiv*, abs/1912.06366, 2019.

J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *ICML*, 2019.

J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *ArXiv*, abs/2004.07219, 2020.

S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau. Benchmarking batch deep reinforcement learning algorithms. *ArXiv*, abs/1910.01708, 2019a.

S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019b.

S. Ghadimi and G. Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *ArXiv*, abs/1309.5549, 2013.

D. Ghosh, A. Gupta, J. Fu, A. Reddy, C. Devin, B. Eysenbach, and S. Levine. Learning to reach goals without reinforcement learning. *ArXiv*, abs/1912.06088, 2019.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

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

M. Jaderberg, V. Mnih, W. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *ArXiv*, abs/1611.05397, 2017.

L. ji Lin. Reinforcement learning for robots using neural networks. 1992.

I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *ArXiv*, abs/2004.13649, 2020.

B. Krose and J. Van Dam. Adaptive state space quantisation for reinforcement learning of collision-free navigation. 01 1995.

A. Kumar, X. B. Peng, and S. Levine. Reward-conditioned policies. *ArXiv*, abs/1912.13465, 2019.

H. Lau, K. Mak, and I. Lee. Adaptive vector quantization for reinforcement learning. *IFAC Proceedings Volumes*, 35(1):493–498, 2002.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17:39:1–39:40, 2016.

S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020.

L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

K. Lolos, I. Konstantinou, V. Kantere, and N. Koziris. Adaptive state space partitioning of markov decision processes for elastic resource management. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 191–194, 2017.

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

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013b.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

S. Nair and C. Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1gzR2VKDH.

J. Oh, Y. Guo, S. Singh, and H. Lee. Self-imitation learning. In *ICML*, 2018.

X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *ArXiv*, abs/1910.00177, 2019.

J. Peters and S. Schaal. Using reward-weighted regression for reinforcement learning of task space control. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 262–267, 2007.

D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291, 2018.

M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, editors, *Machine Learning: ECML 2005*, pages 317–328, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31692-3.

J. C. Santamaria, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2):163–217, 1997. doi: 10.1177/105971239700600201. URL https://doi.org/10.1177/105971239700600201.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

J. Schmidhuber. Reinforcement learning upside down: Don't predict rewards - just map them to actions. *ArXiv*, abs/1912.02875, 2019.

D. Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 1177–1178, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772862. URL https://doi.org/10.1145/1772690.1772862.

E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *ArXiv*, abs/1612.07307, 2017.

A. A. Sherstov and P. Stone. Function approximation via tile coding: Automating parameter choice. In J.-D. Zucker and L. Saitta, editors, *Abstraction, Reformulation and Approximation*, pages 194–205, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31882-8.

S. R. Sinclair, S. Banerjee, and C. L. Yu. Adaptive discretization for episodic reinforcement learning in metric spaces. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3:1 – 44, 2019.

R. K. Srivastava, P. Shyam, F. W. Mutz, W. Jaśkowski, and J. Schmidhuber. Training agents using upside-down reinforcement learning. *ArXiv*, abs/1912.02877, 2019.

P. Stone and R. Sutton. Scaling reinforcement learning toward robocup soccer. pages 537–544, 01 2001.

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

R. S. Sutton, A. R. Mahmood, and M. White. An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, 17 (1):2603–2631, 2016.

H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel. exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS*, 2017.

J. N. Tsitsiklis and B. van Roy. Feature-based methods for large scale dynamic programming. *Mach. Learn.*, 22 (1–3):59–94, Jan. 1996. ISSN 0885-6125. doi: 10.1007/BF00114724. URL https://doi.org/10.1007/BF00114724.

J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

E. Uchibe, M. Asada, and K. Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*, volume 3, pages 1329–1336 vol.3, 1996.

A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *NIPS*, 2017.

H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. Deep reinforcement learning and the deadly triad. *ArXiv*, abs/1812.02648, 2018.

Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

S. Whiteson. Adaptive tile coding for value function approximation. 2007.

S. Zhang, W. Böhmer, and S. Whiteson. Deep residual reinforcement learning. *ArXiv*, abs/1905.01072, 2019.

A Supplementary Materials

A.1 Environment Details

In this section we provide further details about the environments used in our experiments. As noted in the main text, we use the following environments: CartPole, four versions of a Maze environment, and two Miniworld environments.

A.1.1 MAZE

We run our algorithm on 4 versions of a maze environment where the agent is trying to get to the goal where it receives a reward of 1. The goal is always in the same location at the far right in the middle (see figure 5). The environments has 4 actions: up, down, right, left. The observations are 64x64 images with 3 color channels. There are 4 walls in the maze whose length are either fixed or randomized. The underlying state can be described with x-y coordinates ranging from (-.27, -.27) to (.27, .27.) If the agent is not by a wall, the agent moves either in x or y, with .035 with each action.

Dense vs sparse reward. We test our algorithm with and without reward shaping. When there is dense reward shaping, the reward is the negative L2 distance from the goal.

Randomized walls. In the "Randomized Wall" setting, the walls have the x position as in the "Fixed Walls" settings, but the length of the walls are randomly sampled. Specifically, the position of the gap in the wall is randomized. In the fixed wall setting the gaps in the walls are in the middle. See figure 5 for examples.

A.1.2 MINIWORLD

We run our algorithm on two miniworld environments where the task is to navigate to a fixed goal. In the first environment, the goal of the agent is to navigate from a random position on one side of the room to a fixed blue block on the other side of the room. And in the second environment, the agent must first enter a second room, and then navigate to the blue block. There are 3 actions: left, right, and forward. The observations are RGB images with 3 color channels and size 60 x 80. Each wall has a different color in order to make sure the decision process less partially observed.

A.2 Experiment details

In this section we detail the hyperparameters and architectures used in our experiments

Data collection
For the CartPole experiment we collect data using standard DQN. In the Maze and Miniworld experiments, we use a uniform random policy to collect the data.

Architecture
For the CartPole experiment we use a multi-layer perception with three hidden layers of size 64. For both the maze and miniworld experiment we use a convolutional neural network as described in (Mnih et al., 2015). The network has 3 convolutional layers followed by two linear layers with relu activations. The number of channels are 32, 64 and 64. The filter sizes are 8, 4 and 3 and the strides are 4, 3 and 1. The penultimate linear layer has 512 neurons.

Hyperparameters.
In all experiments , we minimize the Huber loss using the Adam optimizer with a batch size of 32 and learning rate of 1e-5. Pytorch default values are used for the other parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and ϵ =1e-8. We use double DQN and update the target net with a period of 20, 200 or 2000 as described in the main text.

During the tabular phase, we stop the value iteration algorithm, when the max error between previous and current matrix, is below $\theta = 0.01$. In the ETA-DP (-FT) experiments, where there is no finetuning, we use a learning rate of $1e-3$ during the supervised learning phase.

BCQ.
We implement the discrete version of BCQ introduced in Fujimoto et al. (2019a). We use their implementation as provided in `https://github.com/sfujim/BCQ/tree/master/discrete_BCQ`. We use an action threshold of 0.3. This implementation is described in appendix A.2 in their paper.

Concretely, we adapt the CNN described previously by adding another branch after the convolutional layers. In addition to having two linear layers after the convolutional layers, we add a second branch also containing two linear layers. The first branch outputs the estimated Q value, and the second branch outputs the estimated probability of that action, under the behavior policy. The loss is the sum of three terms: 1) The standard Huber loss for the predicted value and the target value as in DQN, 2) The cross entropy loss and 3) a regularization term where we square the logit for the estimated action probability. The last term forces the estimated action probabilities to not be too centered around one value.

A.3 Additional results

Experiment 1 Additional Results

In Figure 4 we compare our method to baselines on the Cartpole environment, plotting the task reward (and standard error) vs. training step (where 1 training step is one gradient update). All runs are averaged over 3 datasets generated by DQN agents, each of 1000 episodes.

First, we observe that ETA-DP (-FT) is stable and reliably solves the task, reaching cumulative reward above 150. Note that while there is some performance difference based on the number of clusters K, ETA-DP (-FT) still stably achieves good performance. Second, we observe that with a carefully chosen target update frequency (2000) DDQN is on par with ETA-DP (-FT), however with too frequent target updates DDQN can diverge completely. Lastly, we see that in this

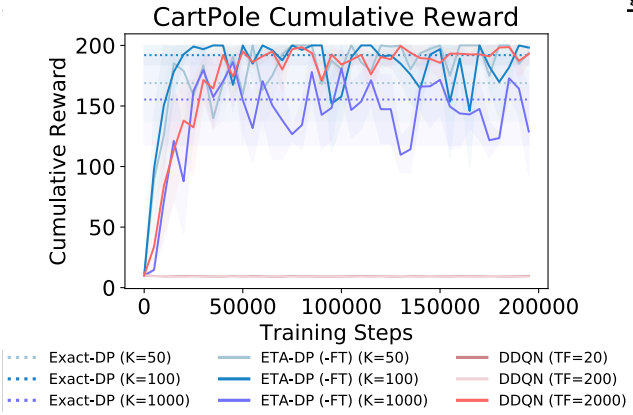

Figure 4: **Experiment 1 (CartPole).** ETA-DP (-FT) and Exact-DP achieve the best performance, while DDQN exhibits sensitivity to hyperparameters.

domain, directly applying Exact-DP also provides strong performance, comparable to ETA-DP.

State quantization

In Figure 5 and Figure 6 we provide qualitative examples of the result of state quantization in the maze and miniworld environments when using 200 clusters.

Overestimation plots

In Figure 7, we plot how the maximum Q value prediction evolves during training. We can see that in both the Maze and Miniworld experiments, Double DQN as well as BCQ starts overestimating Q values. Moreover, we see that for a low target frequency, namely TF=200, the overestimation becomes even larger.

A.4 Method Details

Description of value iteration on fixed dataset shown in Algorithm 2.

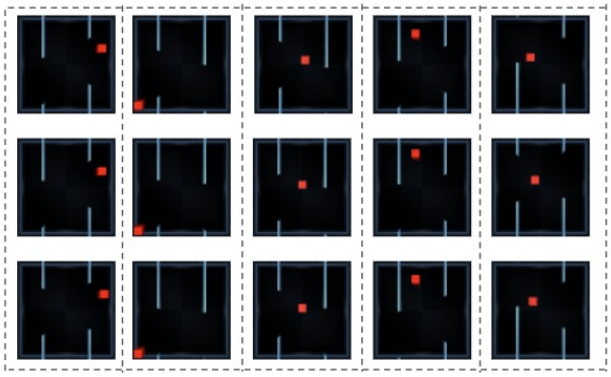

Figure 5: **State Quantization in Maze**. Each column shows examples corresponding to one distinct cluster.

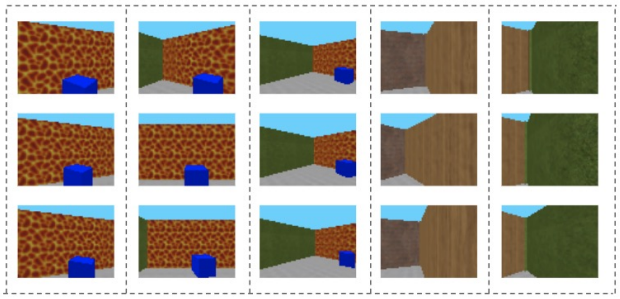

Figure 6: **State Quantization in Miniworld**. Each column shows observations corresponding to one distinct cluster.

Algorithm 2 VALUEITERATION($\bar{\mathcal{D}}$)

Initialize $\hat{Q} = 0$
while not converged **do**
  Initialize list of target values $L = []$
  for $n = 1, 2, ..., N$ **do**
    $(s_t, a_t, s_{t+1}, r_t, x_t, x_{t+1})_n = \bar{\mathcal{D}}_n$
    $L(x_t, a_t) \leftarrow \text{APPEND}(r_t + \gamma \max_a \hat{Q}(x_{t+1}, a))$
  end for
  for $(x_t, a_t) \in \bar{\mathcal{D}}$ **do**
    $\hat{Q}(x_t, a_t) \leftarrow \text{MEAN}(L(x_t, a_t))$
  end for
end while
return $\hat{Q}$

B Related Work

While value based RL has seen remarkable success in recent years, when used in conjunction with off policy learning and function approximation, it remains prone to instability and hyper-parameter sensitivity (Quillen et al., 2018; Henderson et al., 2018). Baird (1995) and Tsitsiklis and Van Roy (1997) show that off-policy TD learning with function approximation may diverge, and with large neural network function approximators tackling challenging tasks, these issues of the "deadly triad" (Sutton and Barto, 2018) remain or are exacerbated (Achiam et al., 2019; van Hasselt et al., 2018). As a result, there have been numerous works in recent years trying to study and tackle the stability challenges of deep value based RL from various angles.

Stabilizing deep Q learning. Early work on neural fitted

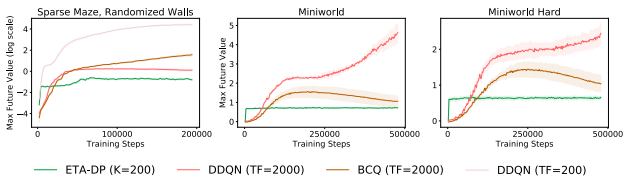

Figure 7: **Overestimation** Shows the average target value predicted by different methods. We observe that DDQN struggles with overestimation, as does BCQ (to a lesser extent).

Q iteration (NFQ) (Riedmiller, 2005) explored training to convergence on a set of data/targets before collecting new data. Similarly, Mnih et al. (2013a) proposed to integrate experience replay (ji Lin, 1992) and target networks. There have since been many new approaches for tackling stability, which aim to reduce overestimation bias (Van Hasselt et al., 2016; Wang et al., 2015; Fujimoto et al., 2018), reduce volatility in the target network (Lillicrap et al., 2015; Zhang et al., 2019; Kostrikov et al., 2020), and stabilize gradients with auxiliary losses (Jaderberg et al., 2017; Shelhamer et al., 2017). Distributional RL (Bellemare et al., 2017) has additionally been shown to improve stability. Many of these techniques have been combined to achieve state of the art performance (Hessel et al., 2018). Like this work, these works effectively change the targets, or the gradients which influence the targets, in order to make learning more stable. However unlike prior work, this work explicitly focuses on decoupling target generation and function approximation.

Reweighting transitions for stability and speed. Another approach to stabilizing deep Q learning is to re-weight transitions during TD learning. Many of these methods draw inspiration from prioritized sweeping (Sutton and Barto, 2018) in the tabular setting, where states with high Bellman error are prioritized in updates, as well as the states that lead into them. Prioritized experience replay (PER) (Schaul et al., 2015) and prioritized sequence experience replay (PECR) (Brittain et al., 2020) apply this idea in the deep Q learning setting. Numerous other works have tried similar sampling schemes, based on adversarial objectives (Fu et al., 2019), likelihood under the policy (Sutton et al., 2016), and estimated target error (Kumar et al., 2019). While these works aim to stabilize Q learning by modifying the data distribution, they still jointly perform bootstrapping and function approximation, unlike our method.

Framing reinforcement learning as supervised learning. Similar to this work, there has been significant prior work that aims to turn reinforcement learning into a supervised learning problem. Early work (Peters and Schaal, 2007) posed reinforcement learning as reward weighted regression, and recent work has incorporated advantage weighting (Peng et al., 2019). Another approach is to frame reinforcement learning as self-imitation (Oh et al., 2018), where the agent clones its past actions or those of a learned policy, and has also been explored recently in the goal-conditioned setting (Ghosh et al., 2019). Another way of transforming reinforcement learning into a supervised learning problem is by conditioning on desired reward or advantage, a technique explored in recent work (Schmidhuber, 2019; Srivastava et al., 2019; Kumar et al., 2019). Unlike most of these works, our technique still relies on bootstrapping, as it enables reasoning over long horizons. However it does so in the absence of function approximation, and introduces function approximation only for supervised learning.

Discretizing states for tabular reinforcement learning. State quantization combined with tabular reinforcement learning is a technique that has been applied to control problems for decades. In robotics, early approaches used manually-designed quantizations of the state space in conjunction with Q learning for control (Asada et al., 1996; Uchibe et al., 1996). Learning based techniques have also been applied to learn discretizations, such learned segmentation (Asada et al., 1998; del R. Millán et al.), tile coding (Santamaria et al., 1997; Stone and Sutton, 2001; Sherstov and Stone, 2005), or more recently deep networks (Corneil et al., 2018; Biza et al., 2020), and applied them to exploration (Tang et al., 2017) or hierarchical RL (Asadi et al., 2020). Prior work has also learned these discetizations adaptively (Krose and Van Dam, 1995; Lolos et al., 2017; Whiteson, 2007; Sinclair et al., 2019; Lau et al., 2002). Similarly, a large body of work has studied state abstraction (Tsitsiklis and van Roy, 1996; Li et al., 2006; Abel et al., 2016; Dong et al., 2019) in relation to Q learning. All of these works are related to ours in that they quantize the state space (or learn a state abstraction) and use it with Q learning. However our key contribution is in additionally leveraging supervised learning and deep Q learning on top of the learned tabular solution, enabling generalization, which we find is critical for good performance on challenging problems.