ESTIMATING $Q(s, s')$ WITH DEEP DETERMINISTIC DYNAMICS GRADIENTS

Ashley D. Edwards [*1], Himanshu Sahni[2], Rosanne Liu[1], Jane Hung[1], Ankit Jain[1], Rui Wang[1], Adrien Ecoffet[1], Thomas Miconi[1], Charles Isbell[2], and Jason Yosinski[1]

[1]Uber AI
[2]Georgia Institute of Technology

ABSTRACT

In this paper, we introduce a novel form of value function, $Q(s, s')$, that expresses the utility of transitioning from a state s to a neighboring state s' and then acting optimally thereafter. In order to derive an optimal policy, we develop a forward dynamics model that learns to make next-state predictions that maximize this value. This formulation decouples actions from values while still learning off-policy. We highlight the benefits of this approach in terms of value function transfer, learning within redundant action spaces, and learning off-policy from state observations generated by sub-optimal or completely random policies. Code and videos are available at `sites.google.com/view/qss-paper`.

1 INTRODUCTION

The goal of Reinforcement Learning (RL) is to learn how to act so as to maximize long term reward. A solution is usually formulated as finding the optimal policy, *i.e.*, selecting the optimal action given a state. A popular approach for finding the optimal policy is to learn a function that defines values though actions, $Q(s, a)$, where $\max_a Q(s, a)$ is a state's value and $\arg\max_a Q(s, a)$ is the optimal action (Sutton & Barto, 1998). We will refer to this approach as QSA.

Here, we propose an alternative formulation for off-policy RL that defines values solely through states, rather than actions. In particular, we introduce $Q(s, s')$, or simply QSS, which represents the value of transitioning from one state to a neighboring state and then acting optimally thereafter:

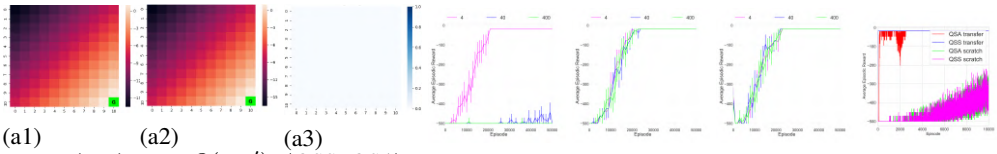
$$Q(s, s') = r(s, s') + \gamma \max_{s'' \in N(s')} Q(s', s'').$$

In this formulation, instead of proposing an action, the agent proposes a desired next state, which is fed into an inverse dynamics model that outputs the appropriate action to reach it. We demonstrate that this formulation has several advantages. First, redundant actions that lead to the same transition are simply folded into one value estimate. Further, by removing actions, QSS becomes easier to transfer than a traditional Q function in certain scenarios. Finally, QSS can learn policies purely from *observations* of (potentially sub-optimal) demonstrations with no access to demonstrator actions.

In order to realize the benefits of off-policy QSS, we must obtain value maximizing future state proposals without performing explicit maximization. To get around this difficulty, we draw inspiration from Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), which learns a policy $\pi(s) \rightarrow a$ over continuous action spaces that maximizes $Q(s, \pi(s))$. We develop the analogous Deep Deterministic Dynamics Gradient (D3G), which trains a forward dynamics model $\tau(s) \rightarrow s'$ to predict next state transitions that maximize $Q(s, \tau(s))$.

We begin the next section by formulating QSS, then describe its properties within tabular settings. We then outline the case of using QSS in continuous settings, where we use D3G to train $\tau(s)$. We evaluate in both tabular problems and MuJoCo tasks (Todorov et al., 2012).

[*]Correspondence to: Ashley Edwards ⟨ ashley.edwards@uber.com ⟩



(a1) $\max_a Q(s,a)$ (a2) $\max_{s'} Q(s,s')$ (a3) $\frac{|QSS-QSA|}{|QSS|}$ (b1) QSA (b2) QSS (b3) ID (b4) Transfer

Figure 1: (a) Learned values for tabular Q-learning in an 11x11 gridworld. The first two figures show a heatmap of Q-values for QSA and QSS. The final figure represents the fractional difference between the learned values in QSA and QSS. (b) Tabular experiments for QSS in an 11x11 gridworld. The first three experiments demonstrate the effect of redundant actions in QSA, QSS, and QSS with learned Inverse Dynamics (ID). The final experiment represents transfer to a gridworld with permuted actions. All experiments shown were averaged over 50 random seeds with 95% confidence intervals.

2 THE QSS FORMULATION FOR RL

We first consider the simple setting where we have access to an inverse dynamics model $I(s, s') \to a$ that returns an action a that would take the agent from state s to s'. Next, we assume access to a function $N(s)$ that outputs the neighbors of s. We use this as an illustrative example and will formulate the problem without these assumptions in the next section.

We define the Bellman update for QSS-learning as:

$$Q(s, s') = Q(s, s') + \alpha[r + \gamma \max_{s'' \in N(s)} Q(s', s'') - Q(s, s')]. \tag{1}$$

Note $Q(s, s')$ is undefined when s and s' are not neighbors. Although this equation may be applied to any environment, for it to be a useful formulation, the environment must be deterministic.

In order to obtain a policy, we define $\tau(s)$ as a function that selects a neighboring state from s that maximizes QSS: $\tau(s) = \arg\max_{s' \in N(s)} Q(s, s')$. In words, $\tau(s)$ selects states that have large value, and acts similar to a policy over states. In order to obtain the policy over actions, we use the inverse dynamics model: $\pi(s) = I(s, \tau(s))$. This approach first finds the state s' that maximizes $Q(s, s')$, and then uses $I(s, s')$ to determine the action that will take the agent there. We rewrite equation 1 as:

$$Q(s, s') = Q(s, s') + \alpha[r + \gamma Q(s', \tau(s')) - Q(s, s')]. \tag{2}$$

3 QSS IN TABULAR SETTINGS

In simple problems where the state space is discrete, $Q(s, s')$ can be represented by a table. We use this setting in an 11x11 gridworld to highlight some of the properties of QSS. All results are shown in Figure 1.

Example of equivalence of QSA and QSS. We first examine the values learned by QSS. QSS learns meaningful values for this task, as the output increases as the agent get closer to the goal. Additionally, the difference in value between $\max_a Q(s, a)$ and $\max_{s'} Q(s, s')$ approaches zero as the values of QSS and QSA converge. Hence, QSS learns similar values as QSA.

QSS handles redundant actions. One benefit of training QSS is that it only updates the transitions, thus ignoring any redundancy in the action space. We further investigate this property in a gridworld with redundant actions. Suppose an agent has four underlying actions, up, down, left, and right, but these actions are duplicated a number of times. As the number of redundant actions increases, the performance of QSA deteriorates, whereas QSS remains unaffected. We also evaluate how QSS is impacted when the inverse dynamics model I is learned rather than given. QSS is able to perform well as it only needs to learn about a single action that transitions from s to s'.

QSS enables value function transfer of permuted actions. We consider the scenario of transferring to an environment where the meaning of actions has changed. We imagine this could be useful in environments where the physics are similar but the actions have been labeled differently. In this case, QSS values should directly transfer, but not the inverse dynamics. QSS is able to learn much more quickly in the transferred environment than QSA.

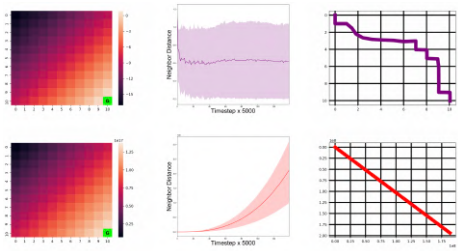


Figure 2: Gridworld experiments for D3G (top) and D3G⁻ (bottom). The left column represents the value function $Q(s, \tau(s))$. The middle column is the average nearest neighbor predicted by τ when s was initialized to $\langle 0, 0 \rangle$. These results were averaged over 5 seeds with 95% confidence intervals. The final column is a trajectory predicted by $\tau(s)$ when starting from the top left corner.

4 QSS IN CONTINUOUS SETTINGS

In contrast to domains where the state space is discrete and both QSA and QSS can represent relevant functions with a table, in continuous settings or environments with large state spaces, we must approximate values with function approximation. A clear difficulty with training QSS in continuous settings is it is not possible to iterate over an infinite state space to find a maximizing neighboring state. Instead, we propose training a model to directly output the state that maximizes QSS. We introduce an analogous approach to TD3 (Fujimoto et al., 2018) for training QSS, Deep Deterministic Dynamics Gradients (D3G). Like the deterministic policy gradient formulation $Q(s, \pi_\psi(s))$, D3G learns a model $\tau_\psi(s) \rightarrow s'$ that makes predictions that maximize $Q(s, \tau_\psi(s))$. To train the critic Q_θ, we specify the loss as: $L_\theta = \sum_i \|y - Q_{\theta'_i}(s, s')\|$. Here, the target y is specified as: $y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tau'_\psi(s'))]$. Similar to TD3, we utilize two critics to stabilize training and a target network for Q. We additionally use a target network for τ, which is updated slowly as $\psi' \leftarrow \eta\psi + (1-\eta)\psi$. We train τ to maximize the expected return, J, starting from any state s:

$$\begin{aligned} \nabla_\psi J &= \mathbb{E}[\nabla_\psi Q(s, s')_{s' \sim \tau_\psi(s)}] \\ &= \mathbb{E}[\nabla_{s'} Q(s, s') \nabla_\psi \tau_\psi(s)] \qquad \text{[using chain rule]} \end{aligned} \tag{3}$$

We discuss in the next section how this formulation alone may be problematic. As in the tabular case, $\tau_\psi(s)$ acts as a policy over states that aims to maximize Q, except now it is being trained to do so. To obtain the necessary action, we apply an inverse dynamics model I as before: $\pi(s) = I_\omega(s, \tau_\psi(s))$. Now, I is trained using a neural network with the loss specified as: $L_\omega = \|I_\omega(s, s') - a\|$.

With the current formulation of the D3G loss, $\tau(s)$ can suggest incorrect states that the critic has overestimated the value for. We propose regularizing τ by ensuring the proposed states are reachable in a single step. We use this regularizer as a substitute for training interactions with τ. Given a state s, we use the dynamics policy $\tau(s)$ to predict the next state s'_τ. We use the inverse dynamics model $I(s, s'_\tau)$ to determine the action a that would yield this transition. We then plug that action into a forward dynamics model $f_\phi(s, a)$ to obtain the final next state, s'_f. In other words, we regularize the model to make predictions that are consistent with the inverse and forward dynamics model learned from environment data. To train the forward dynamics model, we compute: $L_\phi = \|f_\phi(s, a) - s'\|$. The final loss for τ is:

$$L_\psi = -Q_\theta(s, s'_f) + \beta \|\tau_\psi(s) - s'_f)\|. \tag{4}$$

We have the second regularization term to further encourage next state predictions. The final target for training Q becomes: $y = r + \gamma Q_\theta(s', f_\phi(s', I_\omega(s', t'_\psi(s'))))$.

Example of D3G in a gridworld. We first evaluate D3G within a simple 11x11 gridworld with discrete states and actions (Figure 2). We visualize the values learned by D3G and D3G without cycle loss (D3G⁻). Here, D3G and D3G⁻ both learn meaningful values that increase as the agents gets closer to the goal. However, D3G⁻ vastly overestimates the values. Next, we evaluate if $\tau(s)$ learns to predict neighboring states from $\langle 0, 0 \rangle$. D3G is able to predict states that are not more than one step away from the nearest neighbor of thus state. D3G⁻ makes predictions that are significantly outside of the range of the grid. We see this further when visualizing a trajectory of state predictions made by τ. D3G⁻ makes predictions along the diagonal until it extends beyond the grid range. However, QSS learns to predict grid-like steps to the goal, as is required by the environment. This suggests that the cycle loss ensures predictions made by $\tau(s)$ are reachable by the agent.

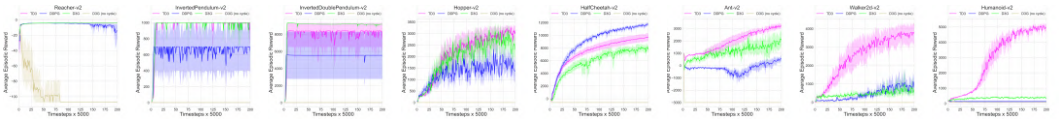


Figure 3: Experiments for training TD3, DDPG, D3G$^-$ and D3G in MuJoCo tasks. The experiments were averaged over 10 seeds with 95% confidence intervals.

	Reacher-v2			InvertedPendulum-v2		
%	π_o	BCO	D3G	π_o	BCO	D3G
0	-4.1 $\pm$ 0.7	**-4.2 $\pm$ 0.6**	-14.7 $\pm$ 30.5	1000 $\pm$ 0	**1000 $\pm$ 0**	3.0 $\pm$ 0.9
25	-12.5 $\pm$ 1.0	-4.3 $\pm$ 0.6	**-4.2 $\pm$ 0.6**	52.3 $\pm$ 3.7	**1000 $\pm$ 0**	602.1 $\pm$ 487.4
50	-22.6 $\pm$ 0.9	-4.9 $\pm$ 0.7	**-4.2 $\pm$ 0.6**	18.0 $\pm$ 2.4	12.1 $\pm$ 8.3	**900.2 $\pm$ 299.2**
75	-32.6 $\pm$ 0.4	-6.6 $\pm$ 1.3	**-4.6 $\pm$ 0.6**	11.4 $\pm$ 1.3	12.1 $\pm$ 8.3	**1000 $\pm$ 0**
100	-40.6 $\pm$ 0.5	-9.7 $\pm$ 0.8	**-6.4 $\pm$ 0.7**	8.6 $\pm$ 0.3	31.0 $\pm$ 4.7	**1000 $\pm$ 0**

Table 1: Learning from observation results. We take the average of the maximum average score after 100000 training iterations for 10 seeds.

D3G can be used to solve control tasks. We next evaluate our approach in more complicated MuJoCo tasks from OpenAI Gym (Brockman et al., 2016). We compare against TD3 and DDPG. In several tasks, D3G is able to perform as well as TD3 and significantly outperforms DDPG (Figure 3). D3G$^-$ is not able to accomplish any of the tasks. D3G does perform poorly in more difficult environments–Humanoid and Walker. Interestingly, DDPG also performs poorly in these tasks. Nevertheless, we have demonstrated that D3G can be used to solve difficult control tasks.

D3G enables learning from observations. Because $Q(s, s')$ does not include actions, we can use it to learn from observations, rather than imitate, in an off-policy manner. We assume we are given a dataset of state observations, rewards, and termination conditions obtained by some policy π_o. We then train D3G to learn QSS values and a model $\tau(s)$ offline without interacting with the environment. We cannot use the previous cycle loss described, as this data does not consist of actions. Instead, we need another function that allows us to cycle from $\tau(s)$ to a predicted next state. We propose using $Q(s, s')$ as a replacement of the action. Namely, we now train the forward dynamics model with the following loss: $L_\phi = \|f_\phi(s, Q_{\theta'}(s, s')) - s'\|$. We can then use the same losses as before for training QSS and τ. Once we learn this model, we can use it to determine how to act in an environment. Given a state s, we use $\tau(s)$ to propose the best next state to reach. To determine what action to take, we train an inverse dynamics model $I(s, s')$ from a few steps taken in the environment, and use it to predict the action a that the agent should take. We compare this to Behavioral Cloning from Observation (BCO) (Torabi et al., 2018) (Table 1). As π_o becomes more random, D3G significantly outperforms BCO, and achieves high rewards in environments collected from random policies. Interestingly, D3G performs poorly when the data has 0% randomness. This is likely because off-policy learning requires that every state has some probability of being visited.

5 Discussion and Conclusion

The concept of generating states is reminiscent of hierarchical RL, where the policy is implemented as a hierarchy of sub-policies (Barto & Mahadevan, 2003; Dayan & Hinton, 1993; Vezhnevets et al., 2017; Kulkarni et al., 2016; Nachum et al., 2018). This work is also related to goal generation approaches in RL, where a *goal* is a set of desired states, and a policy is learned to act optimally toward reaching the goal (Schaul et al., 2015; Florensa et al., 2018; Forestier et al., 2017; Florensa et al., 2017; Nair et al., 2018; Mandlekar et al., 2019; Sahni et al., 2019; Andrychowicz et al., 2017). One could think of the $Q(s, s')$ function in QSS as operating like a manager by suggesting a target state, and of the $I(s, s')$ function as operating like a worker by providing an action that reaches that state. Unlike with hierarchical RL, however, both operate at the same time scale. Several works have considered predicting next states from observations, such as videos, which can be useful for planning or video prediction (Finn & Levine, 2017; Kurutach et al., 2018; Rybkin et al., 2018; Schmeckpeper et al., 2019). In our work, τ is trained automatically to make predictions that maximize the return.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems 30*, pp. 5048–5058. Curran Associates, Inc., 2017.

Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793. IEEE, 2017.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 482–495, 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pp. 1514–1523, 2018.

Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pp. 3675–3683, 2016.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pp. 8733–8744, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ajay Mandlekar, Fabio Ramos, Byron Boots, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *arXiv preprint arXiv:1911.05321*, 2019.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pp. 9191–9200, 2018.

Oleh Rybkin, Karl Pertsch, Konstantinos G Derpanis, Kostas Daniilidis, and Andrew Jaegle. Learning what you can do before doing anything. *arXiv preprint arXiv:1806.09655*, 2018.

Himanshu Sahni, Toby Buckley, Pieter Abbeel, and Ilya Kuzovkin. Addressing sample complexity in visual tasks using her and hallucinatory gans. In *Advances in Neural Information Processing Systems 32*, pp. 5823–5833. Curran Associates, Inc., 2019.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Karl Schmeckpeper, Annie Xie, Oleh Rybkin, Stephen Tian, Kostas Daniilidis, Sergey Levine, and Chelsea Finn. Learning predictive models from observation and interaction. *arXiv preprint arXiv:1912.12773*, 2019.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3540–3549. JMLR. org, 2017.