Estimating $Q(s, s')$ with Deep Deterministic Dynamics Gradients

Ashley D. Edwards [1] **Himanshu Sahni** [2] **Rosanne Liu** [1 3] **Jane Hung** [1] **Ankit Jain** [1] **Rui Wang** [1] **Adrien Ecoffet** [1]
Thomas Miconi [1] **Charles Isbell** [2 *] **Jason Yosinski** [1 3 *]

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 http://sites.google.com/view/qss-paper.

1. Introduction

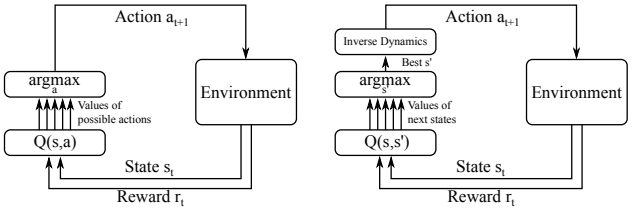
The goal of reinforcement learning is to learn how to act so as to maximize long-term reward. A solution is usually formulated as finding the optimal policy, *i.e.*, selecting the optimal action given a state. A popular approach for finding this policy is to learn a function that defines values though 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 reinforcement learning 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 s to a neighboring state $s' \in N(s)$ and then acting optimally thereafter:

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

[*]Co-senior authors [1]Uber AI Labs [2]Georgia Institute of Technology, Atlanta, GA, USA [3]ML Collective. Correspondence to: Ashley D. Edwards <ashedw88@gmail.com>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).



(a) Q or QSA-learning (b) QSS-learning

Figure 1. Formulation for **(a)** Q-learning, or QSA-learning vs. **(b)** QSS-learning. Instead of proposing an action, a QSS agent proposes a state, which is then fed into an inverse dynamics model that determines the action given the current state and next state proposal. The environment returns the next observation and reward as usual after following the action.

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 (see Figure 1). 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, as it only requires learning an inverse dynamics function upon transfer, rather than a full policy or value function. Finally, we show that QSS can learn policies purely from *observations* of (potentially sub-optimal) demonstrations with no access to demonstrator actions. Importantly, unlike other imitation from observation approaches, because it is off-policy, QSS can learn highly efficient policies even from sub-optimal or completely random demonstrations.

In order to realize the benefits of off-policy QSS, we must obtain value maximizing future state proposals without performing explicit maximization. There are two problems one would encounter in doing so. The first is that a set of neighbors of s are not assumed to be known a priori. This is unlike the set of actions in discrete QSA which are assumed to be provided by the MDP. Secondly, for continuous state and action spaces, the set of neighbors may be infinitely many, so maximizing over them explicitly is out of the question. To get around this difficulty, we draw inspiration from Deep Deterministic Policy Gradient (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 states that maximize $Q(s, \tau(s))$. Notably, this model is not conditioned on actions, and thus allows us to train QSS completely off-policy from observations alone.

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

2. The QSS formulation for RL

We are interested in solving problems specified through a Markov Decision Process, which consists of states $s \in S$, actions $a \in A$, rewards $r(s, s') \in R$, and a transition model $T(s, a, s')$ that indicates the probability of transitioning to a specific next state given a current state and action, $P(s'|s, a)$ (Sutton & Barto, 1998)[1]. For simplicity, we refer to all rewards $r(s, s')$ as r for the remainder of the paper. Importantly, we assume that the reward function does not depend on actions, which allows us to formulate QSS values without any dependency on actions.

Reinforcement learning aims to find a policy $\pi(a|s)$ that represents the probability of taking action a in state s. We are typically interested in policies that maximize the long-term discounted return $R = \sum_{k=t}^{H} \gamma^{k-t} r_k$, where γ is a discount factor that specifies the importance of long-term rewards and H is the terminal step.

Optimal QSA values express the expected return for taking action a in state s and acting optimally thereafter:

$$Q^*(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s', a')|s, a].$$

These values can be approximated using an approach known as Q-learning (Watkins & Dayan, 1992):

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)].$$

Finally, QSA learned policies can be formulated as:

$$\pi(s) = \arg\max_{a} Q(s, a).$$

We propose an alternative paradigm for defining optimal values, $Q^*(s, s')$, or the value of transitioning from state s to state s' and acting optimally thereafter. By analogy with the standard QSA formulation, we express this quantity as:

$$Q^*(s, s') = r + \gamma \max_{s'' \in N(s')} Q^*(s', s''). \quad (1)$$

[1]We use s and s' to denote states consecutive in time, which may alternately be denoted s_t and s_{t+1}.

Although this equation may be applied to any environment, for it to be a useful formulation, the *environment must be deterministic*. To see why, note that in QSA-learning, the max is over actions, which the agent has perfect control over, and any uncertainty in the environment is integrated out by the expectation. In QSS-learning the max is over next states, which in stochastic environments are not perfectly predictable. In such environments the above equation does faithfully track a certain value, but it may be considered the "best possible scenario value" — the value of a current and subsequent state assuming that any stochasticity the agent experiences turns out as well as possible for the agent. Concretely, this means we assume that the agent can transition reliably (with probability 1) to any state s' that it is possible (with probability > 0) to reach from state s.

Of course, this will not hold for stochastic domains in general, in which case QSS-learning does not track an actionable value. While this limitation may seem severe, we will demonstrate that the QSS formulation affords us a powerful tool for use in deterministic environments, which we develop in the remainder of this article. Henceforth we assume that the transition function is deterministic, and the empirical results that follow show our approach to succeed over a wide range of tasks.

2.1. Bellman update for QSS

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

We define the Bellman update for QSS-learning as:

$$Q(s, s') \leftarrow Q(s, s') + \alpha[r + \gamma \max_{s'' \in N(s)} Q(s', s'') - Q(s, s')]. \quad (2)$$

Note $Q(s, s')$ is undefined when s and s' are not neighbors. In order to obtain a policy, we define $\tau(s)$ as a function that selects a neighboring state from s that maximizes QSS:

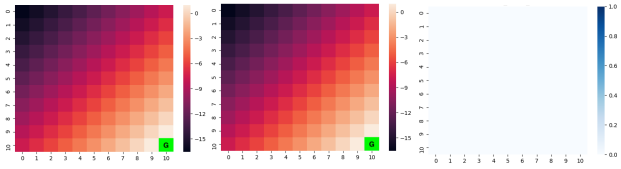
$$\tau(s) = \arg\max_{s' \in N(s)} Q(s, s'). \quad (3)$$

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:

$$\pi(s) = I(s, \tau(s)). \quad (4)$$

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 can rewrite Equation 2 as:

$$Q(s, s') = Q(s, s') + \alpha[r + \gamma Q(s', \tau(s')) - Q(s, s')]. \quad (5)$$



(a) $\max_a Q(s,a)$ (b) $\max_{s'} Q(s,s')$ (c) $\frac{QSS-QSA}{|QSS|}$

Figure 2. 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.

2.2. Equivalence of $Q(s,a)$ and $Q(s,s')$

Let us now investigate the relation between values learned using QSA and QSS.

Theorem 2.2.1. *QSA and QSS learn equivalent values in the deterministic setting.*

Proof. Consider an MDP with a deterministic state transition function and inverse dynamics function $I(s,s')$. QSS can be thought of as equivalent to using QSA to solve the sub-MDP containing only the set of actions returned by $I(s,s')$ for every state s:

$$Q(s,s') = Q(s, I(s,s'))$$

Because the MDP solved by QSS is a sub-MDP of that solved by QSA, there must always be at least one action a for which $Q(s,a) \geq \max_{s'} Q(s,s')$.

The original MDP may contain additional actions not returned by $I(s,s')$, but following our assumptions, their return must be less than or equal to that by the action $I(s,s')$. Since this is also true in every state following s, we have:

$$Q(s,a) \leq \max_{s'} Q(s, I(s,s')) \quad \text{for all } a$$

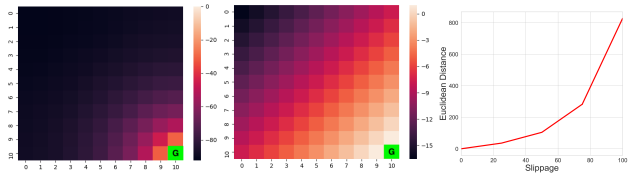
Thus we obtain the following equivalence between QSA and QSS for deterministic environments:

$$\max_{s'} Q(s,s') = \max_a Q(s,a)$$

This equivalence will allow us to learn accurate action-values without dependence on the action space. □

3. QSS in tabular settings

In simple settings where the state space is discrete, $Q(s,s')$ can be represented by a table. We use this setting to highlight some of the properties of QSS. In each experiment, we evaluate within a simple 11x11 gridworld where an agent, initialized at $\langle 0,0 \rangle$, navigates in each cardinal direction and receives a reward of -1 until it reaches the goal.



(a) $\max_a Q(s,a)$ (b) $\max_{s'} Q(s,s')$ (c) Value distance

Figure 3. Learned values for tabular Q-learning in an 11x11 gridworld with stochastic transitions. The first two figures show a heatmap of Q-values for QSA and QSS in a gridworld with 100% slippage. The final figure represents the euclidean distance between the learned values in QSA and QSS as the transitions become more stochastic (averaged over 10 seeds with 95% confidence intervals).

3.1. Example of equivalence of QSA and QSS

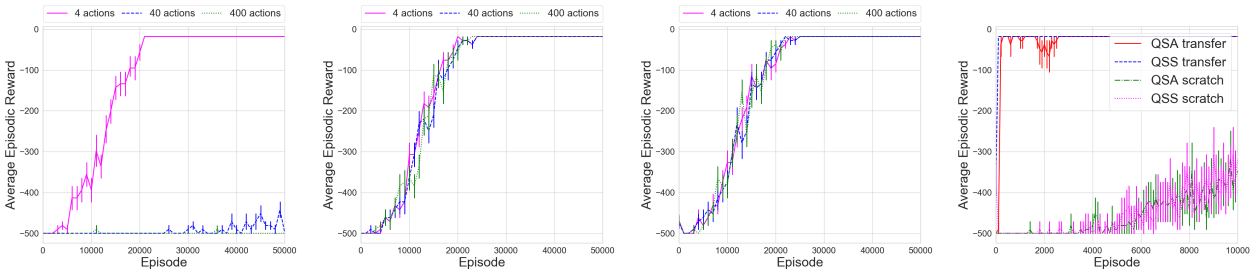
We first examine the values learned by QSS (Figure 2). The output of QSS increases as the agent gets closer to the goal, which indicates that QSS learns meaningful values for this task. Additionally, the difference in value between $\max_a Q(s,a)$ and $\max_{s'} Q(s,s')$ approaches zero as the values of QSS and QSA converge. Hence, QSS learns similar values as QSA in this deterministic setting.

3.2. Example of QSS in a stochastic setting

The next experiment measures the impact of stochastic transitions on learned QSS values. To investigate this property, we add a probability of slipping to each transition, where the agent takes a random action (i.e. slips into an unintended next state) some percentage of time. First, we notice that the values learned by QSS when transitions have 100% slippage (completely random actions) are quite different from those learned by QSA (Figure 3a-b). In fact, the values learned by QSS are similar to the previous experiment when there was no stochasticity in the environment (Figure 2b). As the transitions become more stochastic, the distance between values learned by QSA and QSS vastly increases (Figure 3c). This provides evidence that the formulation of QSS assumes the best possible transition will occur, thus causing the values to be overestimated in stochastic settings. We include further experiments in the appendix that measure how stochastic transitions affect the average episodic return.

3.3. QSS handles redundant actions

One benefit of training QSS is that the transitions from one action can be used to learn values for another action. Consider the setting where two actions in a given state transition to the same next state. QSA would need to make updates for both actions in order to learn their values. But QSS 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



(a) QSA (b) QSS (c) QSS + inverse dynamics (d) Transfer of permuted actions

Figure 4. Tabular experiments in an 11x11 gridworld. The first three experiments demonstrate the effect of redundant actions in QSA, QSS, and QSS with learned inverse dynamics. The final experiment represents how well QSS and QSA transfer to a gridworld with permuted actions. All experiments shown were averaged over 50 random seeds with 95% confidence intervals.

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 (Figure 4a-b).

We also evaluate how QSS is impacted when the inverse dynamics model I is learned rather than given (Figure 4c). We instantiate $I(s, s')$ as a set that is updated when an action a is reached. We sample from this set anytime I is called, and return a random sampling over all redundant actions if $I(s, s') = \emptyset$. Even in this setting, QSS is able to perform well because it only needs to learn about a single action that transitions from s to s'.

3.4. QSS enables value function transfer of permuted actions

The final experiment in the tabular setting considers 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, which would need to be retrained from scratch. We trained QSA and QSS in an environment where the actions were labeled as 0, 1, 2, and 3, then transferred the learned values to an environment where the labels were shuffled. We found that QSS was able to learn much more quickly in the transferred environment than QSA (Figure 4d). Hence, we were able to retrain the inverse dynamics model more quickly than the values for QSA. Interestingly, QSA also learns quickly with the transferred values. This is likely because the Q-table is initialized to values that are closer to the true values than a uniformly initialized value. We include an additional experiment in the appendix where taking the incorrect action has a larger impact on the return.

4. Extending to the continuous domain with D3G

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. One such approach is Deep Q-learning, which uses a deep neural network to approximate QSA (Mnih et al., 2013; Mnih et al., 2015). The loss is formulated as: $\mathcal{L}_\theta = \|y - Q_\theta(s, a)\|$, where $y = r + \gamma \max_{a'} Q_{\theta'}(s', a')$.

Here, θ' is a target network that stabilizes training. Training is further improved by sampling experience from a replay buffer $s, a, r, s' \sim D$ to decorrelate the sequential data observed in an episode.

4.1. Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) applies Deep Q-learning to problems with continuous actions. Instead of computing a max over actions for the target y, it uses the output of a policy that is trained to maximize a critic Q: $y = r + \gamma Q_{\theta'}(s, \pi_{\psi'}(s))$. Here, $\pi_\psi(s)$ is known as an actor and trained using the following loss:

$$\mathcal{L}_\psi = -Q_\theta(s, \pi_\psi(s)).$$

This approach uses a target network θ' that is moved slowly towards θ by updating the parameters as $\theta' \leftarrow \eta\theta + (1 - \eta)\theta'$, where η determines how smoothly the parameters are updated. A target policy network ψ' is also used when training Q, and is updated similarly to θ'.

4.2. Twin Delayed DDPG

Twin Delayed DDPG (TD3) is a more stable variant of DDPG (Fujimoto et al., 2018). One improvement is to delay the updates of the target networks and actor to be slower than the critic updates by a delay parameter d. Additionally, TD3 utilizes Double Q-learning (Hasselt, 2010) to reduce overestimation bias in the critic updates. Instead of training a single critic, this approach trains two and uses the one that

Algorithm 1 D3G algorithm

Inputs: Demonstrations or replay buffer D
Randomly initialize $Q_{\theta_1}, Q_{\theta_2}, \tau_\psi, I_\omega, f_\phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \psi' \leftarrow \psi$
for $t \in T$ **do**
  if imitation **then**
    Sample from demonstration buffer $s, r, s' \sim D$
  else
    Take action $a \sim I(s, \tau(s)) + \epsilon$
    Observe reward and next state
    Store experience in D
    Sample from replay buffer $s, a, r, s' \sim D$
  end if

  Compute $y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', C(s', \tau_{\psi'}(s')))$
  // Update critic parameters:
  Minimize $\mathcal{L}_\theta = \sum_i \|y - Q_{\theta_i}(s, s')\|$

  if t mod d **then**
    // Update model parameters:
    Compute $s'_f = C(s, \tau_\psi(s))$
    Minimize $\mathcal{L}_\psi = -Q_{\theta_1}(s, s'_f) + \beta\|\tau_\psi(s) - s'_f)\|$
    // Update target networks:
    $\theta' \leftarrow \eta\theta + (1-\eta)\theta'$
    $\psi' \leftarrow \eta\psi + (1-\eta)\psi'$
  end if

  if imitation **then**
    // Update forward dynamics parameters:
    Minimize $\mathcal{L}_\phi = \|f_\phi(s, Q_{\theta'_1}(s, s')) - s'\|$
  else
    // Update forward dynamics parameters:
    Minimize $\mathcal{L}_\phi = \|f_\phi(s, a) - s'\|$
    // Update inverse dynamics parameters:
    Minimize $\mathcal{L}_\omega = \|I_\omega(s, s') - a\|$
  end if
end for

minimizes the output of y:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\psi'}(s')).$$

The loss for the critics becomes:

$$\mathcal{L}_\theta = \sum_i \|y - Q_{\theta_i}(s, a)\|.$$

Finally, Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.1)$ is added to the policy when sampling actions. We use each of these techniques in our own approach.

4.3. Deep Deterministic Dynamics Gradients (D3G)

A clear difficulty with training QSS in continuous settings is that 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 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, we specify the loss as:

$$\mathcal{L}_\theta = \sum_i \|y - Q_{\theta_i}(s, s')\|. \tag{6}$$

Here, the target y is specified as:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tau_{\psi'}(s'))]. \tag{7}$$

Similar to TD3, we utilize two critics to stabilize training and a target network for Q.

We train τ to maximize the expected return, J, starting from any state s:

$$\begin{aligned} \nabla_\psi J &= \mathbb{E}[\nabla_\psi Q(s, s')_{s' \sim \tau_\psi(s)}] \\ &= \mathbb{E}[\nabla_{s'} Q(s, s') \nabla_\psi \tau_\psi(s)] \quad \text{[using chain rule]} \end{aligned} \tag{8}$$

This can be accomplished by minimizing the following loss:

$$\mathcal{L}_\psi = -Q_\theta(s, \tau_\psi(s)).$$

We discuss in the next section how this formulation alone may be problematic. We additionally use a target network for τ, which is updated as $\psi' \leftarrow \eta\psi + (1-\eta)\psi$ for stability. 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 using gradient descent. To obtain the necessary action, we apply an inverse dynamics model I as before:

$$\pi(s) = I_\omega(s, \tau_\psi(s)). \tag{9}$$

Now, I is trained using a neural network with data $\langle s, a, s' \rangle \sim D$. The loss is:

$$\mathcal{L}_\omega = \|I_\omega(s, s') - a\|. \tag{10}$$

Algorithm 2 Cycle

function C(s, s'_τ)
  if imitation **then**
    $q = Q_\theta(s, s'_\tau)$
    $s'_f = f_\phi(s, q)$
  else
    $a = I_\omega(s, s'_\tau)$
    $s'_f = f_\phi(s, a)$
  end if
end function

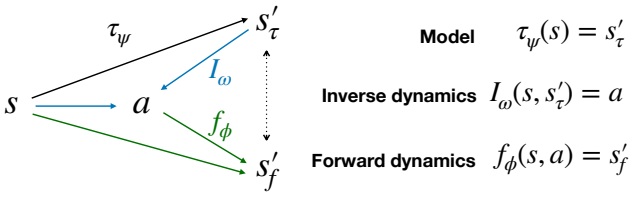


Figure 5. Illustration of the cycle consistency for training D3G. Given a state s, $\tau(s)$ predicts the next state s'_τ (black arrow). The inverse dynamics model $I(s, s'_\tau)$ predicts the action that would yield this transition (blue arrows). Then a forward dynamics model $f_\phi(s, a)$ takes the action and current state to obtain the next state, s'_f (green arrows).

4.3.1. Cycle consistency

DDPG has been shown to overestimate the values of the critic, resulting in a policy that exploits this bias (Fujimoto et al., 2018). Similarly, with the current formulation of the D3G loss, $\tau(s)$ can suggest non-neighboring states that the critic has overestimated the value for. To overcome this, we regularize τ by ensuring the proposed states are reachable in a single step. In particular, we introduce an additional function for ensuring cycle consistency, $C(s, \tau_\psi(s))$ (see Algorithm 2). We use this regularizer as a substitute for training interactions with τ. As shown in Figure 5, given a state s, we use $\tau(s)$ to predict the value maximizing 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 $f(s, a)$ to obtain the final next state, s'_f. In other words, we regularize τ to make predictions that are consistent with the inverse and forward dynamics models.

To train the forward dynamics model, we compute:

$$\mathcal{L}_\phi = \|f_\phi(s, a) - s'\|. \tag{11}$$

We can then compute the cycle loss for τ_ψ:

$$\mathcal{L}_\psi = -Q_\theta(s, C(s, \tau_\psi(s)) + \beta\|\tau_\psi(s) - C(s, \tau_\psi(s))\|. \tag{12}$$

The second regularization term further encourages prediction of neighbors. The final target for training Q becomes:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', C(s', \tau_{\psi'}(s'))) \tag{13}$$

We train each of these models concurrently. The full training procedure is described in Algorithm 1.

4.3.2. A note on training dynamics models

We found it useful to train the models τ_ψ and f_ϕ to predict the difference between states $\Delta = s' - s$ rather than the next state, as has been done in several other works (Nagabandi et al., 2018; Goyal et al., 2018; Edwards et al., 2018). As such, we compute $s'_\tau = s + \tau(s)$ to obtain the next state from $\tau(s)$, and $s'_f = s + f(s, a)$ to obtain the next state prediction for $f(s, a)$. We describe this implementation detail here for clarity of the paper.

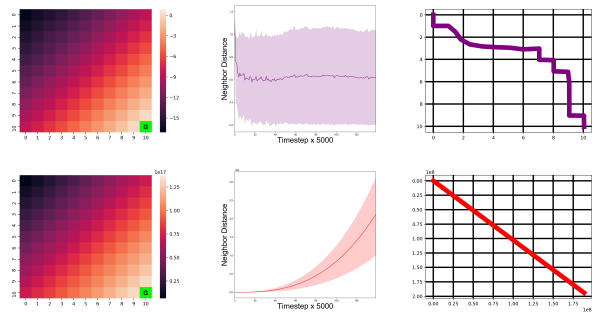


Figure 6. Gridworld experiments for D3G (top) and D3G⁻ (bottom). The left column represents the value function $Q(s, \tau(s))$. The middle column represents the average nearest neighbor predicted by τ when s was initialized to $\langle 0, 0\rangle$. These results were averaged over 5 seeds with 95% confidence intervals. The final column displays the trajectory predicted by $\tau(s)$ when starting from the top left corner of the grid.

5. D3G properties and results

We now describe several experiments that aimed to measure different properties of D3G. We include full training details of hyperparameters and architectures in the appendix.

5.1. Example of D3G in a gridworld

We first evaluate D3G within a simple 11x11 gridworld with discrete states and actions (Figure 6). The agent can move a single step in one of the cardinal directions, and obtains a reward of -1 until it reaches the goal. Because D3G uses an inverse dynamics model to determine actions, it is straightforward to apply it to this discrete setting.

These experiments examine if D3G learns meaningful values, predicts neighboring states, and makes realistic transitions toward the goal. We additionally investigate the merits of using a cycle loss.

We first visualize the values learned by D3G and D3G without cycle loss (D3G⁻). The output of QSS increases for both methods as the agent moves closer to the goal (Figure 6). This indicates that D3G can be used to learn meaningful QSS values. However, D3G⁻ vastly overestimates these values[2]. Hence, it is clear that the cycle loss helps to reduce

[2]One seed out of five in the D3G⁻ experiments did yield a good value function, but we did not witness this problem of overestimation in D3G.

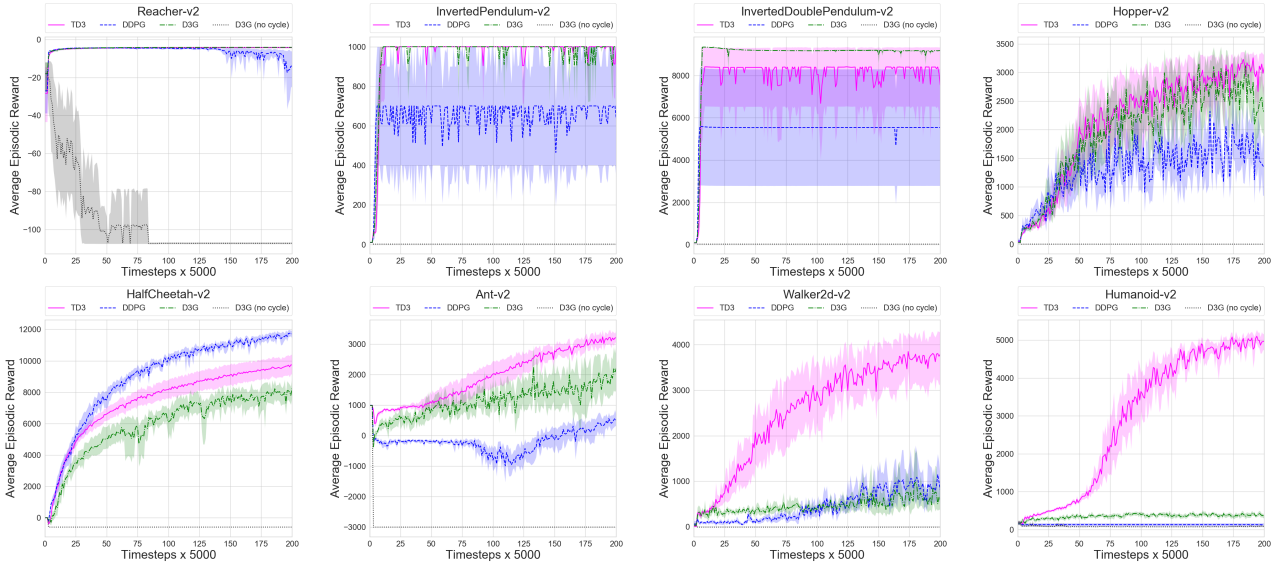


Figure 7. Experiments for training TD3, DDPG, D3G⁻ and D3G in MuJoCo tasks. Every 5000 timesteps, we evaluated the learned policy and averaged the return over 10 trials. The experiments were averaged over 10 seeds with 95% confidence intervals.

overestimation bias.

Next, we evaluate if $\tau(s)$ learns to predict neighboring states. First, we set the agent state to $\langle 0, 0 \rangle$. We then compute the minimum Manhattan distance of $\tau(\langle 0, 0 \rangle)$ to the neighbors of $N(\langle 0, 0 \rangle)$. This experiment examines how close the predictions made by $\tau(s)$ are to neighboring states.

In this task, D3G is able to predict states that are no more than one step away from the nearest neighbor on average (Figure 2). However, 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⁻ simply 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 neighbors of s.

5.2. D3G can be used to solve control tasks

We next evaluate D3G in more complicated MuJoCo tasks from OpenAI Gym (Brockman et al., 2016). These experiments examine if D3G can be used to learn complex control tasks, and the impact of the cycle loss on training. We compare against TD3 and DDPG.

In several tasks, D3G is able to perform as well as TD3 and significantly outperforms DDPG (Figure 7). Without the cycle loss, D3G⁻ is not able to accomplish any of the tasks. D3G does perform poorly in Humanoid-v2 and Walker2d-v2. Interestingly, DDPG also performs poorly in these tasks. Nevertheless, we have demonstrated that D3G can indeed be used to solve difficult control tasks. This introduces a new research direction for actor-critic, enabling training a dynamics model, rather than policy, whose predictions optimize the return. We demonstrate in the next section that this model is powerful enough to learn from observations obtained from completely random policies.

5.3. D3G enables learning from observations obtained from random policies

Imitation from observation is a technique for training agents to imitate in settings where actions are not available. Traditionally, approaches have assumed that the observational data was obtained from an expert, and train models to match the distribution of the underlying policy (Torabi et al., 2018; Edwards et al., 2019). Because $Q(s, s')$ does not include actions, we can use it to *learn* from observations, rather than imitate, in an off-policy manner. This allows learning from observation data from completely random policies.

To learn from observations, we assume we are given a dataset of state observations, rewards, and termination conditions obtained by some policy π_o. We train D3G to learn QSS values and a model $\tau(s)$ offline without interacting with the environment. One problem is that we cannot use the cycle loss described in Section 4, as it relies on knowing the executed actions. Instead, we need another function that allows us to cycle from $\tau(s)$ to a predicted next state.

To do this, we make a novel observation. *The forward dynamics model f does not need to take in actions to predict the next state*. It simply needs an input that can be used as a clue for predicting the next state. We propose using $Q(s, s')$ as a replacement for the action. Namely, we now train the

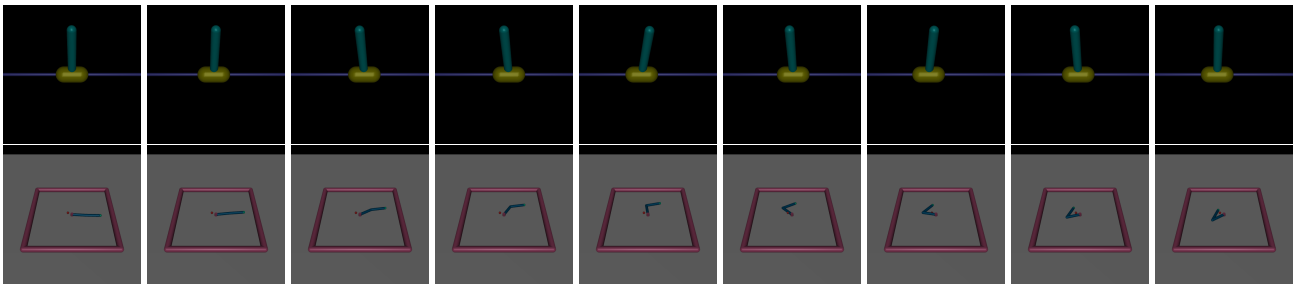


Figure 8. D3G generated plans learned from observational data obtained from a completely random policy in InvertedPendulum-v2 (top) and Reacher-v2 (bottom). To generate the plans, we first plugged the initial state from the column in the left into $C(s, \tau(s))$ to predict the next state s'_f. We then plugged this state into $C(s'_f, \tau(s'_f))$ to hallucinate the next state. We visualize the model predictions after every 5 steps. In the Reacher-v2 environment, we set the target (ball position) to be constant and the delta between the fingertip position and target position to be determined by the joint positions (fully described by the first four elements of the state) and the target position. This was only for visualization purposes and was not done during training. Videos are available at http://sites.google.com/view/qss-paper.

Table 1. Learning from observation results. We evaluated the learned policies every 1000 steps for 100000 total steps. We averaged 10 trials in each evaluation and computed the maximum average score. We then average of the maximum average scores for 10 seeds.

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

forward dynamics model with the following loss:

$$\mathcal{L}_\phi = \|f_\phi(s, Q_{\theta'}(s, s')) - s'\|. \tag{14}$$

Because Q is changing, we use the target network $Q_{\theta'}$ when learning f. We can then use the same losses as before for training QSS and τ, except we utilize the cycle function defined for imitation in Algorithm 2.

We argue that Q is a good replacement for a because for a given state, different QSS values often indicate different neighboring states. While this may not always be useful (there can of course be multiple optimal states), we found that this worked well in practice.

To evaluate this hypothesis, we trained QSS in InvertedPendulum-v2 and Reacher-v2 with data obtained from expert policies with varying degrees of randomness. We first visualize predictions made by $C(s, \tau(s))$ when trained from a completely random policy (Figure 8). Because $\tau(s)$ aims to make predictions that maximize QSS, it is able to hallucinate plans that solve the underlying task. In InvertedPendulum-v2, $\tau(s)$ makes predictions that balance the pole, and in Reacher-v2, the arm moves directly to the goal location. As such, we have demonstrated that $\tau(s)$ can be trained from observations obtained from random policies to produce optimal plans.

Once we learn this model, we can use it to determine how to act in an environment. To do this, given a state s, we use $\tau(s) \rightarrow s'_\tau$ to propose the best next state to reach. In order to determine what action to take, we train an inverse dynamics model $I(s, s'_\tau)$ 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), which aims to learn policies that mimic the data collected from π_o.

As the data collected from π_o becomes more random, D3G significantly outperforms BCO, and is able to achieve high reward when the demonstrations were collected from completely random policies (Table 1). This suggests that D3G is indeed capable of off-policy learning. 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.

6. Related work

We now discuss several works related to QSS and D3G.

Hierarchical reinforcement learning The concept of generating states is reminiscent of hierarchical RL (Barto & Mahadevan, 2003), in which the policy is implemented as a hi-

erarchy of sub-policies. In particular, approaches related to feudal RL (Dayan & Hinton, 1993) rely on a manager policy providing goals (possibly indirectly, through sub-manager policies) to a worker policy. These goals generally map to actual environment states, either through a learned state representation as in FeUdal Networks (Vezhnevets et al., 2017), an engineered representation as in h-DQN (Kulkarni et al., 2016), or simply by using the same format as raw environment states as in HIRO (Nachum et al., 2018). One could think of the $\tau(s)$ 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.

Goal generation 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. For example, Universal Value Function Approximators (Schaul et al., 2015) consider the problem of conditioning action-values with goals that, in the simplest formulation, are fixed by the environment. Recent advances in automatic curriculum building for RL reflects the importance of self-generated goals, where the intermediate goals of curricula towards a final objective are automatically generated by approaches such as automatic goal generation (Florensa et al., 2018), intrinsically motivated goal exploration processes (Forestier et al., 2017), and reverse curriculum generation (Florensa et al., 2017).

Nair et al. (2018) employ goal-conditioned value functions along with Variational autoencoders (VAEs) to generate goals for self-supervised practice and for dense reward relabeling in hindsight. Similarly, IRIS (Mandlekar et al., 2019) trains conditional VAEs for goal prediction and action prediction for robot control. Sahni et al. (2019) use a GAN to hallucinate visual goals and combine it with hindsight experience replay (Andrychowicz et al., 2017) to increase sample efficiency. Unlike these approaches, in D3G goals are always a single step away, generated by maximizing the the value of the neighboring state.

Learning from observation Imitation from Observation (IfO) allows imitation learning without access to actions (Sermanet et al., 2017; Liu et al., 2017; Torabi et al., 2018; Edwards et al., 2019; Torabi et al., 2019; Sun et al., 2019). Imitating when the action space differs between the agent and expert is a similar problem, and typically requires learning a correspondence (Kim et al., 2019; Liu et al., 2019). IfO approaches often aim to match the performance of the expert. D3G aims to *learn*, rather than imitate. T-REX (Brown et al., 2019) is a recent IfO approach that can perform better than the demonstrator, but requires a ranking over demonstrations. Finally, like D3G, Deep Q-learning from Demonstrations learns off-policy from demonstration data, but requires demonstrator actions (Hester et al., 2018).

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, the model τ is trained automatically to make predictions that maximize the return.

Action reduction QSS naturally combines actions that have the same effects. Recent works have aimed to express the similarities between actions to learn policies more quickly, especially over large action spaces. For example, one approach is to learn action embeddings, which could then be used to learn a policy (Chandak et al., 2019; Chen et al., 2019). Another approach is to directly learn about irrelevant actions and then eliminate them from being selected (Zahavy et al., 2018). That work is evaluated in the text-based game Zork. Text-based environments would be an interesting direction to explore as several commands may lead to the same next state or have no impact at all. QSS would naturally learn to combine such transitions.

Successor Representations The successor representation (Dayan, 1993) describes a state as the sum of expected occupancy of future states under the current policy. It allows for decoupling of the environment's dynamics from immediate rewards when computing expected returns and can be conveniently learned using TD methods. Barreto et al. (2017) extend this concept to successor *features*, $\psi^{\pi}(s, a)$. Successor features are the expected value of the discounted sum of d-dimensional *features* of transitions, $\phi(s, a, s')$, under the policy π. In both cases, the decoupling of successor state occupancy or features from a representation of the reward allows easy transfer across tasks where the dynamics remains the same but the reward function can change. Once successor features are learned, they can be used to quickly learn action values for all such tasks. Similarly, QSS is able to transfer or share values when the underlying dynamics are the same but the action label has changed.

7. Conclusion

In this paper, we introduced QSS, a novel form of value function that expresses the utility of transitioning to a state and acting optimal thereafter. To train QSS, we developed Deep Deterministic Dynamics Gradients, which we used to train a model to make predictions that maximized QSS. We showed that the formulation of QSS learns similar values as QSA, naturally learns well in environments with redundant actions, and can transfer across shuffled actions. We additionally demonstrated that D3G can be used to learn complicated control tasks, can generate meaningful plans from data obtained from completely random observational data, and can train agents to act from such data.

Acknowledgements

The authors thank Michael Littman for comments on related literature and further suggestions for the paper. We would also like to acknowledge Joost Huizinga, Felipe Petroski Such, and other members of Uber AI Labs for meaningful discussions about this work. Finally, we thank the anonymous reviewers for their helpful comments.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems 30*, pp. 5048–5058. Curran Associates, Inc., 2017.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pp. 4055–4065, 2017.

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

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Brown, D., Goo, W., Nagarajan, P., and Niekum, S. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, 2019.

Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. S. Learning action representations for reinforcement learning. *arXiv preprint arXiv:1902.00183*, 2019.

Chen, Y., Chen, Y., Yang, Y., Li, Y., Yin, J., and Fan, C. Learning action-transferable policy with action embedding. *arXiv preprint arXiv:1909.02291*, 2019.

Dayan, P. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

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

Edwards, A., Sahni, H., Schroecker, Y., and Isbell, C. Imitating latent policies from observation. In *International Conference on Machine Learning*, pp. 1755–1763, 2019.

Edwards, A. D., Downs, L., and Davidson, J. C. Forward-backward reinforcement learning. *arXiv preprint arXiv:1803.10227*, 2018.

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

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 482–495, 2017.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, pp. 1514–1523, 2018.

Forestier, S., Mollard, Y., and Oudeyer, P.-Y. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Goyal, A., Brakel, P., Fedus, W., Lillicrap, T., Levine, S., Larochelle, H., and Bengio, Y. Recall traces: Backtracking models for efficient reinforcement learning. *arXiv preprint arXiv:1804.00379*, 2018.

Hasselt, H. V. Double q-learning. In *Advances in neural information processing systems*, pp. 2613–2621, 2010.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Kim, K. H., Gu, Y., Song, J., Zhao, S., and Ermon, S. Cross domain imitation learning. *arXiv preprint arXiv:1910.00105*, 2019.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pp. 3675–3683, 2016.

Kurutach, T., Tamar, A., Yang, G., Russell, S. J., and Abbeel, P. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pp. 8733–8744, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, F., Ling, Z., Mu, T., and Su, H. State alignment-based imitation learning. *arXiv preprint arXiv:1911.10947*, 2019.

Liu, Y., Gupta, A., Abbeel, P., and Levine, S. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *arXiv preprint arXiv:1707.03374*, 2017.

Mandlekar, A., Ramos, F., Boots, B., Fei-Fei, L., Garg, A., and Fox, D. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *arXiv preprint arXiv:1911.05321*, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

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

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE, 2018.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pp. 9191–9200, 2018.

Rybkin, O., Pertsch, K., Derpanis, K. G., Daniilidis, K., and Jaegle, A. Learning what you can do before doing anything. *arXiv preprint arXiv:1806.09655*, 2018.

Sahni, H., Buckley, T., Abbeel, P., and Kuzovkin, I. 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.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Schmeckpeper, K., Xie, A., Rybkin, O., Tian, S., Daniilidis, K., Levine, S., and Finn, C. Learning predictive models from observation and interaction. *arXiv preprint arXiv:1912.12773*, 2019.

Sermanet, P., Lynch, C., Hsu, J., and Levine, S. Time-contrastive networks: Self-supervised learning from multi-view observation. *arXiv preprint arXiv:1704.06888*, 2017.

Sun, W., Vemula, A., Boots, B., and Bagnell, J. A. Provably efficient imitation learning from observation alone. *arXiv preprint arXiv:1905.10948*, 2019.

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

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

Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

Torabi, F., Warnell, G., and Stone, P. Recent advances in imitation learning from observation. *arXiv preprint arXiv:1905.13566*, 2019.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3540–3549. JMLR. org, 2017.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., and Mannor, S. Learn what not to learn: Action elimination with deep reinforcement learning. In *NeurIPS*, 2018.