Accelerating Reinforcement Learning with Suboptimal Guidance [⋆]

Eivind Bøhn [*] **Signe Moe** [*,**] **Tor Arne Johansen** [**]

[*] *SINTEF Digital, Oslo, Norway (e-mail: {eivind.bohn, signe.moe}@sintef.no).*
[**] *Centre for Autonomous Marine Operations and Systems, Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway (email: {signe.moe, tor.arne.johansen}@ntnu.no)*

Abstract: Reinforcement learning in domains with sparse rewards is a difficult problem, and a large part of the training process is often spent searching the state space in a more or less random fashion for learning signals. For control problems, we often have some controller readily available which might be suboptimal but nevertheless solves the problem to some degree. This controller can be used to guide the initial exploration phase of the learning controller towards reward yielding states, reducing the time before refinement of a viable policy can be initiated. To achieve such an exploration guidance while also allowing the learning controller to outperform the demonstrations provided to it, Nair et al. (2017) proposes to use a "Q-filter" to select states where the agent should clone the behaviour of the demonstrations. The Q-filter selects states where the critic deems the demonstrations to be superior to the agent, providing a natural way to adjust the guidance in a manner that is adaptive to the proficiency of the demonstrator. The contribution of this paper lies in adapting the Q-filter concept from pre-recorded demonstrations to an online guiding controller, and further in identifying shortcomings in the formulation of the Q-filter and suggesting some ways these issues can be mitigated — notably by replacing the value comparison baseline with the guiding controller's own value function — reducing the effects of stochasticity in the neural network value estimator. These modifications are tested on the OpenAI Gym Fetch environments, showing clear improvements in adaptivity and yielding increased performance in all robotics environments tested.

Keywords: Deep Reinforcement Learning, Non-Linear Control Systems, Robotics

1. INTRODUCTION

Reinforcement learning (RL) (Sutton and Barto, 2018) is a field of machine learning concerned with sequential decision making problems. The theory and methods in RL has produced some impressive results the last few years, ranging from high-level reasoning tasks such as game-playing (Silver et al., 2016; OpenAI et al., 2019) to control of fast dynamics such as actuation in robotics (Gu et al., 2016). RL has received much attention and interest due to its framework being very general, in principle capable of discovering a strategy for nearly any problem as long as one can define some notion of utility of different states of the system.

This utility measure, called the reward function in the RL framework, is central to both the definition of the problem and the performance of the algorithm. Often, a sparse reward function [1] is preferable, as they are easier to formulate, easier to estimate, and importantly, there is less room for ambiguity and misinterpreting the objective of the task. The obvious downside is however that only a small region of the problem space actually gives a learning signal to the agent, and a significant portion of the training time is spent exploring the state space (often in a random manner) until a minimally viable strategy (i.e. one that consistently is able to reach reward-yielding states) is found, from which further progress can be made.

Often in robotics and other control applications one already has a controller, which due to e.g. nonlinearities in the dynamics, uncertainties in model parameters, or strict computational requirements might be suboptimal. This controller could be used to guide the learning controller towards a minimally viable strategy, thereby reducing the long initial exploration phase. Further, guiding in this manner can offer more explainable behaviour from the learning controller, in the sense that its behaviour is close to that of the guiding controller. In turn, this approach may also lead to worse asymptotic performance, as the learning controller is searching for an optimum of the policy space in the vicinity of the guiding controller, while the global optimum might lie somewhere else. Getting to the global optimum might entail climbing several unattractive regions of less optimal policies, which might not be achiev-

[⋆] The first author is financed by "PhD Scholarships at SINTEF" from the Research Council of Norway (grant no. 272402). The third author was supported by the Research Council of Norway (grant no. 223254 NTNU AMOS).

[1] A sparse reward function is one that only yield signals in some subset of the state space, e.g. in the subset defined as the set of goal states, and is typically constant elsewhere.

able with the usual reinforcement learning optimization instruments.

The most naive implementation of imitation learning (IL), that is, trying to directly copy the pre-existing controller is seldom successful, mainly due to a data distribution mismatch (Ross et al., 2011). Furthermore, the potential performance of the learning controller is bounded by the performance of the existing controller. The data distribution problem arises as the data collected by the demonstrator will only consist of a subset of the state space, and will offer no guidance on how to course correct back into this subset when the learning controller inevitably makes a small deviation from the controller it is imitating. Small errors therefore tend to accumulate into trajectories that stray far from those produced by the demonstrator. Additionally, when combining IL with RL there needs to be some mechanism in place to allow the learning controller to make different choices than the original controller when appropriate, in order to exceed the performance of the existing controller.

In this paper, we tackle the problem of automatically deciding when the learning controller should imitate the pre-existing controller, and when it should develop its own behaviour. We base our method in the concept of the Q-filter (Nair et al., 2017), which makes the learning controller imitate the pre-existing controller only when the pre-existing controller is deemed to yield a higher reward than what the learning controller would achieve in that situation. The contribution of this work lies in identifying some key issues with the original formulation of the Q-filter and proposing modifications to mitigate these issues, resulting in improved adaptivity to the difficulty of the task and to the proficiency of the pre-existing controller, and thus also improved performance.

The rest of the paper is organized as follows. First, related work is presented and discussed in Section 2, then the requisite background theory for RL and the algorithms used in this paper is presented in Section 3. Our method is outlined in Section 4, and the experiments used to evaluate the methods are outlined in Section 5. The results of the experiments are shown in Section 6, which also discusses the strengths and weaknesses of the proposed method. Finally, Section 7 concludes with our thoughts on the matter and suggestions for further work.

2. RELATED WORK

The DAGGER algorithm (Ross et al., 2011) is an approach to mitigating the data distribution mismatch problem. A learning controller is trained to mimic the behaviour of some expert controller from a demonstration dataset, and then this learning controller is used in the environment to collect more data. The newly collected data is appended to the training set, and the expert demonstrator is asked to label the new data with its action choices. Our method builds upon the DAGGER framework in the sense that we assume access to an online demonstrator, and iteratively expand the dataset with the demonstrators knowledge. However, we account for suboptimality in the demonstrator, consequently our learning controller is only made to mimic the demonstrator in some selected states and this is only one of its optimization objectives. Deeply AggreVaTeD (Sun et al., 2017) is a further extension of DAGGER to continuous state and action spaces and nonlinear function approximation with neural networks (NNs). Both these approaches are pure IL methods and thus do not allow for the agent to surpass the demonstrator.

Another approach to extract knowledge from demonstration data is inverse reinforcement learning (IRL) (Ng and Russell, 2000), in which the aim is to recover the reward function that the agent that generated the dataset was trying to optimize, rather than directly learning how to act from the data. Armed with the reward function, the learning agent can reason on how the expert would act even in situations that are not covered in the dataset, or otherwise develop its own behaviour that is different but also in some sense optimal with respect to the reward function.

Deep deterministic policy gradients from demonstrations (DDPGfD) (Vecerik et al., 2018) is a method for leveraging demonstration data for RL algorithms in domains with continuous state and action spaces. In this work, human generated demonstrations are included in the replay buffer for which a prioritized sampling technique is used to ensure a suitable mix of demonstration data and self collected data in each training batch. They further use a mix of 1-step and n-step return losses for training the Q-networks. They show increased performance over the demonstrations, as well as over regular deep deterministic policy gradients (DDPG), even when the latter uses hand-crafted shaped rewards and DDPGfD uses sparse rewards.

Nair et al. (2017) propose to address the problem of choosing when the learning controller should emulate the demonstrator with the use of the Q-filter. The Q-filter is an indicator function used to select when to apply a behaviour cloning loss on the action chosen by the demonstrator. The filter evaluates to true when the estimated value from the Q-function is higher for the demonstrator action than it is for the action chosen by the learning controller. This provides a natural way to anneal behaviour cloning loss during the training process that is adaptive to the task at hand. Their demonstrations come in the form of a fixed set of trajectories collected by a human demonstrator in a virtual reality version of the environment. In each training batch, a portion of the data is sampled from the regular replay buffer and a portion is sampled from the demonstration buffer, on which behaviour cloning (BC) is applied for the actions selected by the Q-filter. Our method borrows the concept of the Q-filter from this work, suggesting some important adjustments to the concept. Further, their source of demonstrations consists of a fixed dataset, while we assume online access to a guiding controller.

In assisted deterministic policy gradients (AsDDPG) Xie et al. (2018) propose a novel architecture for incorporating a simple controller to guide the initial exploration phase. In their architecture, the critic module has an additional branch that estimates the Q-values of the pre-existing controllers action and the actors action for the state in question. Based on these estimates the method greedily chooses the one with the highest Q-value, and applies this action as the exploration action during training. In this way, the guiding controller is used to show the actor some primitives it can use to aid exploration. As in our method, online access to the guiding controller is assumed, but the

Q-filter is applied to select actions for exploration instead of to shape the gradients of the optimization procedure.

3. BACKGROUND

3.1 Reinforcement Learning

We consider the RL problem in a Markov decision process (MDP) framework. The MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, R, \mathcal{T}, \gamma \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $R(s, a)$ is the reward function, $\mathcal{T}$ is the state transition function describing the evolution of the states as a function of time and actions, and $\gamma \in [0, 1]$ is the discount factor, weighing the relative importance of immediate and future rewards. We denote a trajectory τ as a sequence of state and actions, e.g. one episode of the problem task, and the return as $R(\tau) = \sum_{(s_t,a_t)\sim\tau} \gamma^t R(s_t, a_t)$, where $\sim$ signifies that the left hand side is sampled from the distribution on the right hand side. The RL objective is to find a policy π, i.e. a function that generates actions from states, which maximizes the sum of expected future rewards weighed by the discount factor γ, i.e. the expected return in an episodic setting. We denote this objective as $\max_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} [R(\tau)]$, where we use subscript to indicate θ is the parameterization of the policy. Central to many RL algorithm is the concept of the value of a state, given by the value function $V^\pi(s) = \mathbb{E}_{\tau\sim\pi} [R(\tau) \mid s_0 = s]$. The value function measures the expected return of being in a given state and from there always acting according to the policy π. If we further leave the first action in the trajectory free, we get the state-action value function, often called the Q-function $Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi} [R(\tau) \mid s_0 = s,\ a_0 = a]$. We consider in this work a further extension to the MDP framework in the form of a specified goal state, from a set of possible goals $\mathcal{G}$. Schaul et al. (2015) show that value functions conditioned on this additional goal parameter can successfully be trained to generalize to unseen goals.

3.2 Deep Deterministic Policy Gradient

To address the challenges posed by tasks with continuous state and action spaces, Lillicrap et al. (2015) introduced DDPG. DDPG is an off-policy, model-free, policy-gradient, actor-critic algorithm which concurrently estimates the Q-function Q^{π_θ} and an actor that aims at solving for the actions which maximize the Q-function. These actor and critic functions are implemented as NNs, and are trained off-policy from experience collected by the actor interacting with the environment. This experience is stored in a replay buffer $\mathcal{D}$, from which we sample minibatches $\mathcal{B}$ to train the actor and the critic. More concretely, the critic is trained in a supervised manner to regress on 1-step returns according to the Bellman optimality equation:

$$y_t = R_t + \gamma Q_{\theta^Q}(s_{t+1}, \pi_\theta(s_{t+1})) \tag{1}$$

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{(s_t,a_t,R_t,s_{t+1})\sim\mathcal{B}} \left[(y_t - Q_{\theta^Q}(s_t, a_t))^2\right] \tag{2}$$

where θ^Q is the parameterization of the Q-function. The actor attempts to find the optimal policy according to the critic through the relationship (3). Since this actor is deterministic, a stochastic behaviour policy π^b (4) is used to collect experience in order to improve exploration, which is obtained by adding noise that is normally distributed with moments that are hyperparameters of the optimization problem. Finally, the actor is trained by optimizing the objective (5).

$$\pi(s) = \arg\max_a Q^\pi(s, a) \tag{3}$$

$$\pi^b(s) = \pi(s) + \mathcal{N}(\mu, \sigma) \tag{4}$$

$$\mathcal{L}(\theta) = \max_\theta \mathbb{E}_{s\sim\mathcal{D}} [Q_{\theta^Q}(s, \pi_\theta(s))] \tag{5}$$

3.3 Twin Delayed Deep Deterministic Policy Gradients

Twin delayed DDPG (TD3) (Fujimoto et al., 2018) improves upon DDPG in several ways, notably by alleviating sources of overestimation in the critic network, and introduces more robustness towards hyperparameters. It does this by maintaining two separate networks estimating the critic Q-function, and using the lesser of these two as the target for regression (1). Further they suggest updating the policy networks less frequently to allow the value functions more time to converge before they are used to update the policy.

3.4 Hindsight Experience Replay

The environments considered in this paper have sparse rewards, requiring a directed effort over several actions to reach regions of the state space where any learning signal is received at all. To reach learning signals in such scenarios more quickly, Andrychowicz et al. (2017) proposed hindsight experience replay (HER), in which the experience contained in the replay buffer is retroactively fitted with a new goal state that was actually achieved sometime during the episode. In this way, by pretending that some state the actor was successful in achieving was the goal state, the algorithm receives more frequent learning signals and convergence is accelerated.

4. METHOD

4.1 Problem Description

In our setup we assume online access to a pre-existing sub-optimal controller which is capable of reaching goal states from a nonempty set of initial states. We also assume that we have access to the Q-function of this controller, denoted by Q^G. However, we do not assume prior access to demonstration trajectories from this controller.

4.2 Proposed Method

We modify the actor objective function of DDPG style algorithms (5) by adding a BC loss term (6) weighted by λ_{BC}, and as in Nair et al. (2017) we selectively apply this loss with the indicator function conditioned on the Q-filter:

$$\mathcal{L}_{BC}(\theta) = \sum_{i=1}^{N_b} \|a_i - \pi_\theta(s_i)\|_2 \, \mathbb{1}_{Q^G(s_i,a_i) > Q^{\pi_\theta}(s_i,\pi_\theta(s_i))} \tag{6}$$

$$\mathcal{L}(\theta) = \max_\theta \mathbb{E}_{s\sim\mathcal{B}} [Q_{\theta^Q}(s, \pi_\theta(s)) - \lambda_{BC}\mathcal{L}_{BC}] \tag{7}$$

$$\text{where } \mathbb{1}_{A>B} = \begin{cases} 1 & \text{if } A > B \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$Q^{\pi_{\theta_0}} \leftarrow Q^G \tag{9}$$

Here N_b is the number of samples in each training batch, i.e. we use the online availability of the pre-existing controller to apply this term to all samples in the batch. This does not add considerable overhead, as the pre-existing controller's action can be evaluated once and then saved to the replay buffer alongside the other data. [2] In this work we suggest to modify the original formulation of the Q-filter by replacing the left hand side of the condition with the pre-existing controller's own estimated Q-function, which is kept static throughout the training process. As the TD3 algorithm has two separate Q-networks, one can look at different ways of combining these to make up the Q-filter. In this work we have chosen to only use the Q-network that is used in the optimization objective (5) as a basis for the comparison.

The motivation behind this modification is based on two key insights about the formulation of the original Q-filter in Nair et al. (2017):

First, the quantities compared are very similar, especially for problems with a sparse reward function and problems with low sampling time as is common in control applications. Each side of the inequality consists of the immediate reward of the state resulting from applying the given action in the current state, and the total value of the trajectory produced by the agent from this new state. With low sampling time each action is applied only for a short time, and each state in a sequence is similar to the previous even for very different actions. The immediate rewards for the two controllers actions are therefore also similar, especially for sparse rewards where they will in most cases be identical. The difference in the total reward of the two trajectories which are produced by the same controller from a similar starting state should also in most cases be minute. Great accuracy is therefore needed to correctly assess the superior action. By using the guiding controllers own Q-function on the other side of the inequality we compare the total value of two trajectories produced by different controllers, thus the difference should be greatly increased, allowing for easier discrimination.

Second, as explained the Q-function needs to be highly accurate to properly assess the values of the actions. The comparison will therefore be greatly impacted by the randomness inherent in an unconverged neural network. Since the actors objective (5) in the DDPG algorithm is precisely to find the maximization of actions over the Q-function, it will quickly learn to optimize the unconverged Q-network, and therefore seemingly (most times erroneously) offering better options than the guiding controller. The action comparison performed in the Q-filter is consequently highly stochastic for much of the learning process. Nair et al.'s implementation will henceforth be referred to as the naive method.

In practice, the naive Q-filter therefore tends to prefer the guiding controllers actions infrequently in the beginning, and converges to select the guiding controller as superior with a non-zero frequency. We want the learning controller to aim to emulate the pre-existing controller to a large degree in the beginning, and then have the BC loss taper off to allow the learning controller to surpass the pre-existing controller. With the original formulation of the Q-filter however, the BC loss does not seem to be strategically applied.

We obtain Q^G by running the TD3 algorithm with the guiding controller as the actor, until the objective (5) stabilizes. Since the actor in this case is static, all data is on-policy and n-step updates can be used to learn the Q-function for faster and more accurate convergence. [3] See Section 6 for a discussion on other ways of obtaining Q^G. In order for the actor's and the guiding controller's Q-functions to have comparable magnitudes, the actor's Q-network is initialized to that of the guiding controller (9). This also provides a starting point that should be closer to optimal than a random initialization.

5. EXPERIMENTS

We train and evaluate our method on the OpenAI Gym Fetch environments (Plappert et al., 2018), which are based on the MuJoCo physics simulator (Todorov et al., 2012). The FetchReach environment is easy to solve for the RL algorithms even without the use of a guiding controller, and as such has been excluded from the experiments. For each environment a test set consisting of 100 random initial states and goal states has been constructed, and the agents are evaluated on this set every 20k time steps. We evaluate our method against the original formulation of the Q-filter, against an approach where the weight of the BC loss is linearly decayed with time steps, as well as an unguided approach using TD3 + HER. We run each experiment with five different random seeds and show mean results with one standard deviation confidence bounds.

5.1 Guiding Controllers

The guiding controllers are handcrafted proportional controllers with some conditional logic, e.g. move over block, then lower hand and grip block etc. The controllers exhibit varying proficiency levels, from a controller capable of achieving any goal in the test set for the pick-and-place environment, to a controller achieving merely 21% of the goals in the slide task.

5.2 Configuration

Due to limited computational resources we employed the TD3 algorithm instead of DDPG, as convergence of the latter is often dependent on running several actors in parallel. To increase sample efficiency we use HER with the future goal selection strategy, generating four imagined goals for every real experience sample. We use the Stable-Baselines (Hill et al., 2018) implementation of the algorithms, running on an I7-9700k 8-core CPU and an RTX 2070 GPU.

Following the original paper introducing the Fetch environments (Andrychowicz et al., 2017), we use a decay factor of 0.98 and clip the regression Q-targets (1) to the ranges possible in the given environment, and also bootstrap on environment termination due to time limit. We use the hyperparameters from TD3, with an actor and critic consisting of two fully-connected hidden layers of

[2] Training with a guiding controller increases the wall clock time of the training process by about 5% compared with normal training in our experiments.

[3] Convergence was achieved after less than 100k time steps in our experiments for all environments.

400 and 300 nodes respectively, a learning rate of 0.001 for both actor and critic, and a replay buffer size of 10^6. Training is run every 1000 time steps of the environment for 1000 gradient steps, with a batch size of 100. We apply an $\mathcal{L}_2$ regularization term with strength 0.01 to the pre-activation values of the actors output layer, in order to mitigate vanishing gradients issues. The BC loss term weighting factor λ_{BC} (7) is set to 2.

The linear schedule is set to be fully decayed after 500k time steps, with an initial value equal to that of the other methods. This schedule is not optimal for all environments, but a static schedule was chosen to highlight some of the issues with this approach.

5.3 Ablation Studies

It is conceivable that the improvements of the our method stem mainly from the knowledge encoded into Q^G and the initialization of Q^{π_θ} to this. A pretrained Q^G already contains information about which action would maximize the rewards when further following the same action choices as the pre-existing controller at each state, and the optimization procedure implemented by the actor could take advantage of this information to achieve some of the same benefits that the BC loss offers. To test this hypothesis, we ran one version of our method without the BC loss term, and one version of the naive method which was initialized to Q^G as in (9).

6. RESULTS AND DISCUSSION

6.1 Results of Experiments

Figure 1 illustrates the unwanted behaviours of the naive Q-filter as described in section 4: it typically starts low and converges to some value in the range of 0.3 - 0.5. Our method on the other hand delivers on its promises of adaptivity: it copies a large portion of the actions produced by the optimal guiding controller in the pick-and-place environment, selects only some of the actions produced by the pre-existing controllers in the slide and push environments, and decreases the frequency of imitation as the agent improves. The two Q-filter approaches have the same asymptotic performance for the push and slide environments, but our Q-filter method has considerable more success in the pick-and-place environment. The linear method matches the performance of the adaptive methods for the push environment, but falls a bit short in the pick-and-place environment, and is only able to find any success for a single seed in the most difficult slide task. The linear scheduling method seems to be in general the method with the quickest initial learning, due to indiscriminately imitating all actions from the guiding controller at the start, while the two Q-filter approaches seem to share a similar time at which improvements begin.

Note that the unguided baseline agent using only TD3 + HER is unable to achieve any success at all in the time frame considered here, except for a single seed for the FetchSlide environment. By comparing with the guided methods in Figure 1, one can clearly see how much one can accelerate the learning curve and get better consistency by incorporating knowledge from existing solutions to the task in the training process of RL agents. Our results for the unguided TD3 + HER method are considerably worse than the DDPG + HER results reported in Andrychowicz et al. (2017) and Plappert et al. (2018), in which they show that these methods are able to find working approaches without guidance in the Fetch environments, albeit at a much slower pace than the guided approaches in this paper. This discrepancy might stem from their work leveraging extensive computational resources to run many data-collecting agents in parallel, which generates more uncorrelated and diverse exploration data, and that these methods are reliant on this trick. We also did not perform specific hyperparameter optimization for the unguided approach, but rather used hyperparameters reported in previous work as described in Section 5.

The results of the ablation experiments are shown in Figure 2. These experiments show that initializing to Q^G is not enough in and of itself to explain the performance differences shown in Figure 1, as our method is significantly degraded by the removal of the BC loss term, while the naive method is largely unaffected by the addition of the initialization procedure. These results suggests that the observed performance gains from our method stem in large part from an improved comparison in the Q-filter. However, since the model without the BC term in the ablation experiments and the unguided model in Figure 1 differ only in initialization, it is clear that the algorithm is capable of utilizing the information contained in Q^G in some way to achieve more success.

6.2 Evaluation of Results

While our implementation of the Q-filter does lead to more deliberate application of the BC loss, it does introduce a credit assignment problem: Since the two Q-functions that are compared correspond to two different controllers, it is unclear which actions in the action sequences generated by the controllers are responsible for the observed difference in value. Imitating only the initial action is therefore a potential source of error, but introducing some randomness and perturbation to the training process in RL is a known measure to avoid local minima, and therefore this might not be a significant issue.

Even though the linear scheduling method is less complex than the other methods it can sometimes achieve similar or better performance in terms of takeoff time and asymptotic performance. The slide environment clearly illustrates that this approach require careful tuning, as the method is in this case unable to represent a policy capable of achieving any goals for most seeds. The Q-filters on the other hand are adaptive and provide reasonable performance for any task with little tuning, and in particular our method will imitate a good guiding controller to a larger degree than a bad guiding controller, unlike the linear scheduling method. Our version of the Q-filter does however require additional information in the form of Q^G, and the linear method might therefore be preferable in some scenarios despite its limitations.

The Q-function of the pre-existing controller need not be obtained in a complicated manner such as Q-learning. Often, one has some historical data of usage of this controller, which one could use to estimate the expected sum of rewards in an entirely offline manner. Furthermore, one might not need a function approximator as complicated

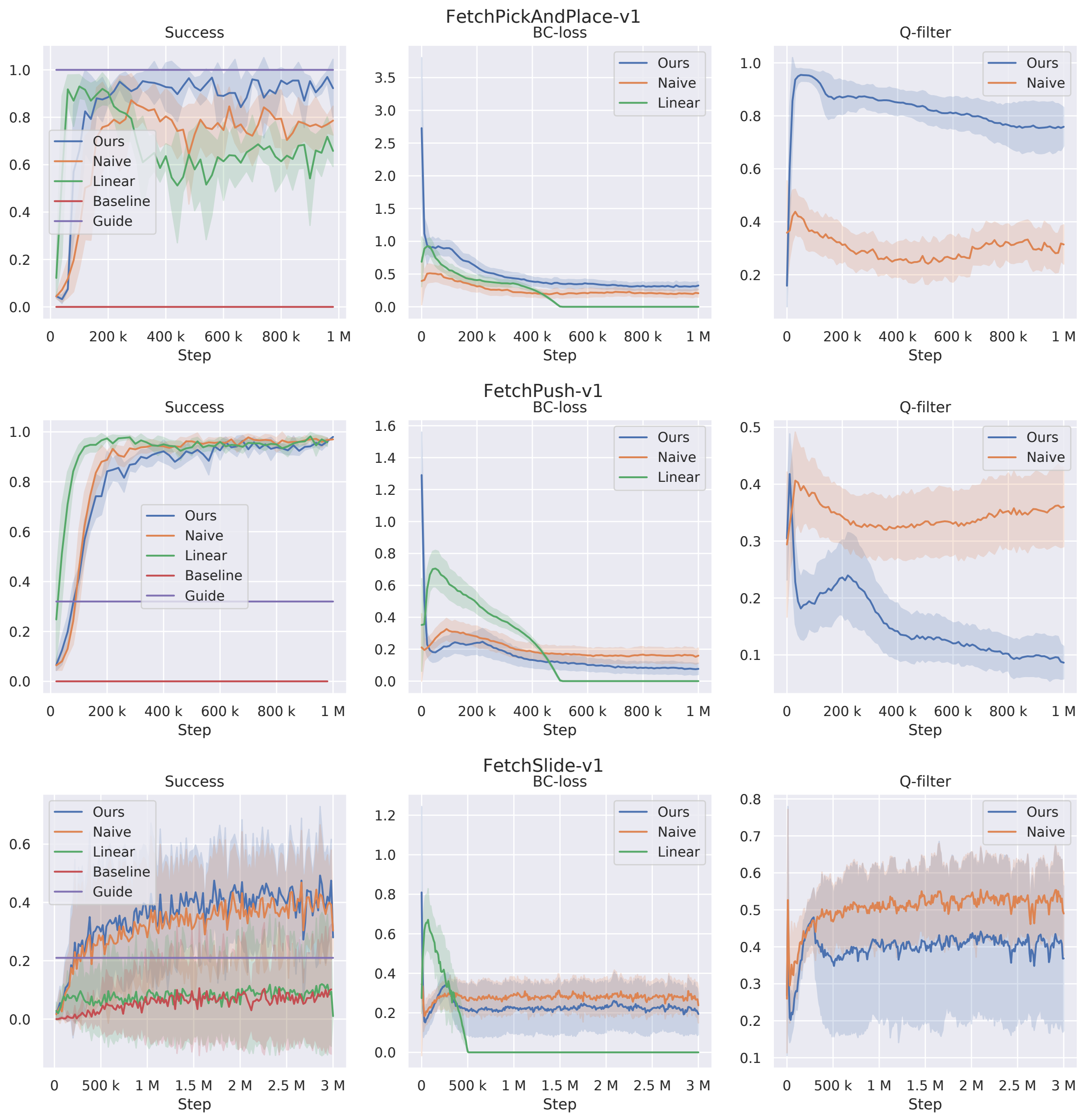

Fig. 1. Results of experiments: The graphs show for each environment the success rate on the test set, the BC loss, and the fraction of actions for which the indicator function evaluates to true in each minibatch.

as a NN for this purpose, and some simpler methods such as linear regression might be sufficient. This would remove the possibility of directly initializing the Q-function of the actor to that of the pre-existing controller, but one could simply pretrain the Q-network in a supervised manner on outputs from the pre-existing controller's Q-function to achieve much of the same effect.

7. CONCLUSION

We have shown that our method is capable of accelerating learning using guiding controllers with a wide spread of proficiency levels. An important further work is looking into how the optimality of the guiding controller affects the rate of convergence and asymptotic performance of the learning controller.

Having a pre-existing controller available online opens up many possibilities. If one knowns the stability properties of this controller, one can have the agent explore freely while still in the region of attraction (RoA) of the guiding controller, and then have the guiding controller assume control and stabilize the system when necessary. In this way one can achieve a form of safe training, in the sense that only safe regions of the state space are visited, and risk of damage to the system is minimized. An interesting further work is a study on what fraction of actions the agent needs to control itself for learning to be successful in such a scenario.

REFERENCES

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight Experience Replay. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*.

Fujimoto, S., van Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic

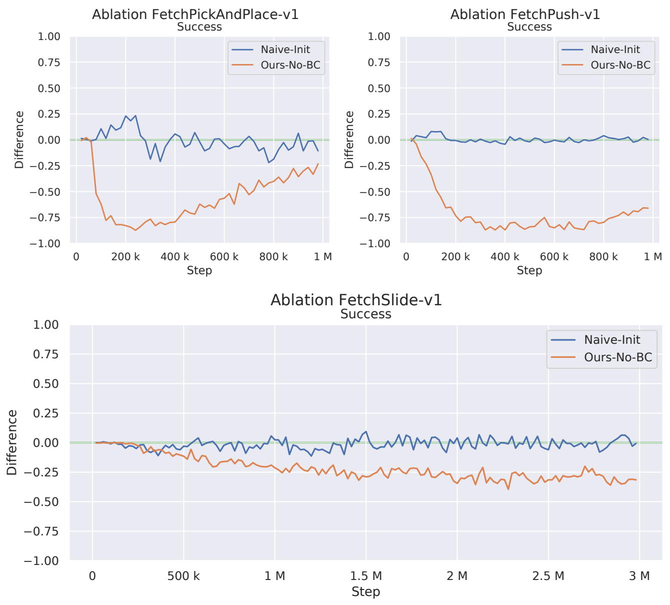

Fig. 2. The difference in performance when including initialization to Q^G in the naive method and when removing the BC loss term from our method, compared to its counterpart in Figure 1 as a baseline.

Methods. In *Proceedings of the 35th International Conference on Machine Learning*.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2016). Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*.

Hill, A., Raffin, A., Ernestus, M., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. `https://github.com/hill-a/stable-baselines`.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016*.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Overcoming Exploration in Reinforcement Learning with Demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

Ng, A.Y. and Russell, S.J. (2000). Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, 663–670. San Francisco, CA, USA.

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H.P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv:1912.06680*.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv preprint arXiv:1802.09464*.

Ross, S., Gordon, G.J., and Bagnell, J.A. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning*.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.v.d., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.

Sun, W., Venkatraman, A., Gordon, G.J., Boots, B., and Bagnell, J.A. (2017). Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction. In *Proceedings of the 34th International Conference on Machine Learning*.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

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

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2018). Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv:1707.08817 [cs]*.

Xie, L., Wang, S., Rosa, S., Markham, A., and Trigoni, N. (2018). Learning with Training Wheels: Speeding up Training with a Simple Controller for Deep Reinforcement Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6276–6283.