Reward Machines for Vision-Based Robotic Manipulation

Alberto Camacho and Jacob Varley and Andy Zeng and Deepali Jain and Atil Iscen and Dmitry Kalashnikov

***Abstract*—Deep Q learning (DQN) has enabled robot agents to accomplish vision based tasks that seemed out of reach. Despite recent success stories, there are still several sources of computational complexity that challenge the performance of DQN. We place the focus on vision manipulation tasks, where the correct action selection is often predicated on a small number of pixels. We observe that in some of these tasks DQN does not converge to the optimal Q function, and their values do not separate well optimal and suboptimal actions. In consequence, the policies obtained with DQN tend to be brittle and manifest a low success rate, especially in long horizon tasks. In this work we show the benefits of Reward Machines (RMs) for Deep Q learning (DQRM) in vision based robot manipulation tasks. Reward machines decompose the task at an abstract level, inform the agent about their current stage along task completion, and guide them via dense rewards. We show that RMs help DQN learn the optimal Q values in each abstract state. Their policies are more robust, manifest higher success rate, and are learned with fewer training steps compared with DQN. The benefits of RMs are more evident in long-horizon tasks, where we show that DQRM is able to learn good-quality policies with six times times fewer training steps than DQN, even when this is equipped with dense reward shaping.**

I. Introduction

Deep reinforcement learning (RL), and in particular DQN, has succeeded across different domains (e.g, Atari games from pixels [1], board games [2], and robot control from pixels [3]). In this work we evidence a limitation of standard DQN in the context of vision based robotic manipulation tasks expressed in a necessity of having to solve two related subproblems: 1) learning useful latent state features, and 2) learning a policy that is conditioned on such features. DQN solves the two subproblems at once, while we show value in decoupling them. Intuitively, some of these latent features describe state properties at an abstract level and may be indicative of the current stage of a task. This collection of features defines an *abstract state*, a notion that can be exploited to decompose long-horizon tasks into simpler subtasks. Abstract states are an additional supervision signal that complement commonly-used reward signals in RL. Our main insight in this paper is in revealing how crucial an explicit notion of abstract states can really be. Even in some very simple tasks, DQN fails to learn good-quality policies unless abstract states are provided. This phenomenon occurs even when the information encoded in abstract states is already available within state observations.

We address the subproblems above (developing useful state features & learning a policy conditioned on such features) with a hybrid approach to RL, where the agent has access to a set of *feature detectors*. These features enable the agent to track it's current abstract state, and policies need be conditioned on fewer latent features of the low-level state and are easier to learn. Then, we construct dense reward functions that facilitate learning sub-policies conditioned on abstract states. Our approach makes use of *reward machines*, a mathematical structure that provides a principled mechanism to drive the agent along different abstract states, and equips them with a dense reward structure that guides exploration toward task completion. We show how to construct a reward machine from demonstrations, in a way that centers Q-Values around zero—which is beneficial to deep learning methods. We refer to this approach of augmenting a DQN with a reward machine as DQRM.

Our experiments evidence the limits of standard DQN, which was unable to learn good-quality policies even in very simple vision-based robotics tasks. Reward machines help overcome such limitations by providing supervision signals from abstract states and reward shaping. Abstract states alone

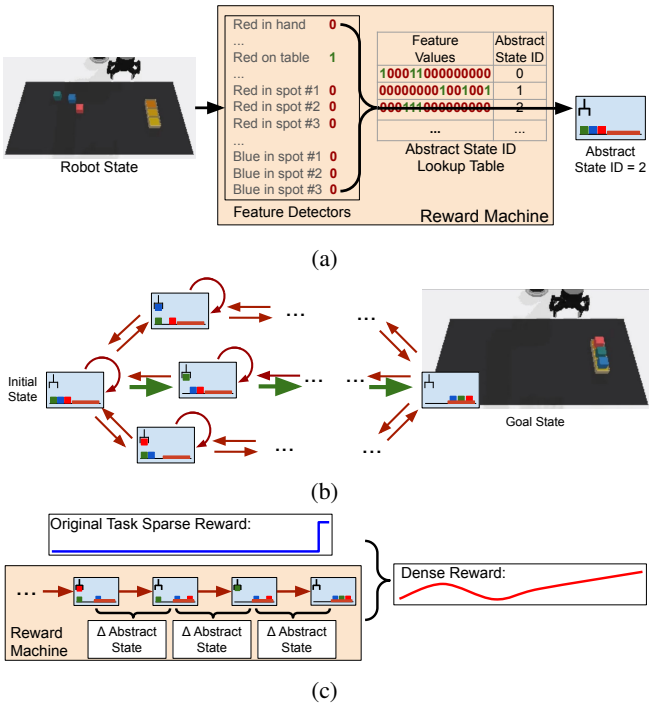


Fig. 1: Reward machines can be leveraged to produce useful policy features, define tasks, and create reward functions. a) Reward machines are used to compute the current abstract state id from a set of feature detectors. This abstract state id can be directly input into a policy as an extra source of information. b) Tasks are accomplished by moving from a starting abstract state id to a goal abstract state id, and multiple paths can exist. c) The reward machine can smooth out a sparse task reward by densifying the reward signal as the policy transitions between abstract states. The agent is rewarded for moving towards the goal abstract state and penalized for moving away.

The authors are affiliated with Robotics at Google, New York.

can be effectively exploited by DQN to improve on sample efficiency and policy quality, but it is in the combination of using abstract states and rewards together that DQRM manifests a substantially improved performance.

The main contributions of this work are 1) A novel DQRM implementation specific to vision based robotic manipulation tasks. 2) An investigation of a surprising failure of vanilla DQN on vision based robotics tasks (difficulty inferring an appropriate next action when it is indicated by a small number of pixels) and demonstrate how DQRM handles this failure case. 3) A seres of blations on a variety of vision based robot manipulation tasks.

II. Related Work

Task Decomposition in RL reduces tasks into subtasks that an agent can efficiently learn independently and in parallel—[4]. Hierarchical RL (HRL) is one such framework that decomposes planning and control into high-level and low-level policies. Recent options-based methods [5] learn options from reference data [6], [7] through interactions with the environment [8], [9]. Designing good options still requires manual engineering, and is hard to scale with task complexity.

State Abstractions are used to reduce the search space in MDPs, by aggregating states that share certain properties. A number of techniques exist to compute MDP abstractions [10], [11]—e.g., bi-simulations [12]—, with trade-offs between state compression and RL performance [13]. In this work, we are not concerned with planning in the abstract state. Rather, we augment state observations with abstract states as a useful extra supervision signal for deep RL.

Reward Machine (RM) decompose tasks into finite-state machines that encode reward functions for MDPs [14]. An RM is characterized by a set of internal abstract states forming a graph, which the agent navigates through different modes of operation. Abstract state transitions indicate different sub-policies, each associated with an individual reward function. The graph structure of RMs can be exploited toward more sample-efficient RL [14]–[16]. The abstract states of an RM do not need to directly correlated with the progression of one specific task. This property makes them practical for real robots, where various simple hardware signals can be used as features—e.g., end effector states, gripper opening, suction flow readings, proximity sensor readings, joint encoders, etc.

RL for Vision-Based Manipulation policies that map pixels to actions have shown promise for of tasks including grasping [3], [17]–[20], stacking [21], pushing [22], tossing [23], and even a Rubik's cube [24]. Many of these vision-based RL systems leverage handcrafted features beyond pure rgb observations. For example, the gripper height and status [3], a hash of the current state [25], and a CNN prediction of the current pose and state of the cube [24]. Beyond just incorporating additional features into the observation, we demonstrate RMs as a principled and effective means to leverage additional state features to improve the learning efficiency of RL for manipulation tasks.

III. Background

Consider a Markov decision process (MDP): at any given state $s_t \in \mathcal{S}$ at time t, the agent (robot) chooses and executes an action $a_t \in \mathcal{A}$ according to a policy $\pi(s_t)$, then transitions to a new state s_{t+1} according to a transition probability $Pr(s_t, a_t)$ and receives a reward $r(s_t, a_t, s_{t+1}) \in \mathbb{R}$. The goal of our reinforcement learning (RL) problem is to find an optimal policy π^* that maximizes the expected γ-discounted sum of future rewards $\mathbb{E}_\pi \Sigma \gamma^t r(s_t, a_t, s_{t+1})$.

In this work, we investigate off-policy deep Q-learning [1] to train a greedy policy $\pi(s_t)$ that chooses actions by maximizing a parameterized Q-function $\arg\max_{a \in \mathcal{A}} Q_\theta(s_t, a)$ which measures the expected reward of taking action a in state s_t, where θ denotes the weights of a neural network (architecture described in Sec. V). Formally, at each training iteration i, our learning objective is to minimize:

$$\mathcal{L}_i = |Q_{\theta_i}(s_t, a_t) - y_t|$$

$$y_t = r(s_t, a_t, s_{t+1}) + \gamma Q_{\theta_i}(s_{t+1}, \arg\max_{a \in \mathcal{A}} Q_{\theta_i}(s_{t+1}, a_{t+1}))$$

where transitions (s_t, a_t, r_t, s_{t+1}) are uniformly sampled from a replay buffer. Our exploration strategy is ϵ-greedy, with $\epsilon = 0.1$. Our MDPs include a set of terminating goal states $g \in \mathcal{G} \subseteq \mathcal{S}$ for which $Q(g, a) = 0$.

A Reward Machine (RM) is a decision process with a set of abstract states U: at a given $u_t \in U$, the RM receives a truth assignment σ_t (a set of propositions that hold true in s_t, which can be expressed as a binary vector), then moves to the next abstract state $u_{t+1} = \delta(u_t, \sigma_t)$ according to the state-transition function δ, and outputs a reward function. In this paper, we only consider reward functions of the type $\rho(u_t, u_{t+1})$

When paired with an MDP, the RM uses a labeling function $L : S \to 2^{AP}$ to map from states to truth assignments $\sigma_t = L(s_t)$. At a given RM state u_t, if the agent executes action a_t to transition from state s_t to s_{t+1} in the MDP, then the RM transitions to state $u_{t+1} = \delta(u_t, F(s_{t+1}))$, and the agent receives a reward $r(s_t, a_t, s_{t+1})$ where $r = \rho(u_t, u_{t+1})$.

Note that the RM may exhibit a different graph structure than the MDP, depending on the labeling function and truth assignments—multiple MDP states s can map to a single RM state u, or the agent can transition to s in different RM states. As shown in Deep Q-learning with RMs (DQRM) [14], these redundancies can be exploited to improve the sample efficiency of off-policy Q-learning. Specifically, DQRM learns one subpolicy per state in the RM, where each RM transition is a simpler subtask. Transitions sampled from the replay buffer are then redistributed across subpolicies trained in parallel. This form of task decomposition is guaranteed to converge to optimal policies, and has been shown to learn faster and achieve better performance than the Hierarchical RL options framework [5]. The benefits of DQRMs are most apparent for non-Markovian tasks. In this work, we show that RMs can also be useful for task decomposition in Markovian tasks, and that they have additional benefits beyond task decomposition.

IV. Reward Machines from Demonstrations

We address the problem of constructing RMs for an MDP with goal states, and present a method to construct RMs from demonstrations that encourage a DQRM agent to achieve such goals. Our method complements recent work, which construct RMs by hand or from a high-level logical specification of the intended behavior of the agent [14], [15]. Demonstrations are sequences of states $s_0, s_1, \ldots, s_n$ that show how a certain task—e.g., stacking a tower of blocks—can be completed, ending in one of the *goal* states. Demonstrations can be often acquired with a minimal technical knowledge, compared to constructing RMs by hand or designing logical specifications.

We construct an RM in two stages. First, we construct an *abstract planning graph* by mapping demonstrations into sequences of abstract states. In a second stage, we equip such graph with a reward function to obtain a RM. We introduce the necessary concepts below. The first thing we need is a set of *feature detectors*, that map state observations s into the subset of propositions $\sigma \in 2^{AP}$ that hold in s. Each proposition in AP represents a boolean state property—e.g., whether the end effector is holding an object. We presume those features detectors are given. With them, we define the labeling function L that we need to communicate with the RM. In the following, we say that $\sigma = L(s)$ is the *abstraction* of state s in AP by L. We further presume that the set of features are rich enough to discriminate between goal and non-goal states—i.e., goal and non-goal states must be mapped into different abstract states.

Stage 1: Constructing an abstract planning graph. The *abstract demonstration* associated to a demonstration $s_0, s_1, \ldots, s_n$ is a sequence $\sigma_0, \sigma_1, \ldots, \sigma_n$, where σ_i is the abstraction of s_i into AP by L, for each $i \in \{1, \ldots, n\}$. A set of demonstrations defines the *abstract planning graph* (V, E). The set of nodes, V, is the set of abstract states σ that appear in the abstract demonstrations. The set of directed edges, E, connect σ with σ' if, and only if, σ and σ' are two consecutive abstract states in some abstract demonstration. This construction presumes that the task is Markovian. In the general case, our methods could be used in conjunction with existing methods that construct graph-like structures that capture non-Markovian behavior (e.g., [26], [27]).

Stage 2: Constructing a RM. The planning graph gives us almost all the ingredients to construct a RM with elements $\langle U, u_0, \Sigma, \delta, \rho \rangle$. Intuitively, the dynamics of the RM are an abstraction the MDP through L. The states in the RM are in one-to-one correspondence with the elements returned by the labeling function L. That is, each RM state $u_\sigma \in U$ is in correspondence with an abstract state $\sigma \in 2^{AP}$. For simplicity, we confuse RM states and observations and write $u_\sigma = \sigma$. The initial state of the RM is the abstraction of the initial state s_0 of the MDP, i.e., $u_0 = L(s_0)$. Recall that the dynamics of a RM updates their state along with the new state observation, $u' = \delta(u, L(s'))$. Therefore, the input alphabet in our RM is the set of values that L can take, i.e., 2^{AP}; and the transition function maps observations into abstract states in a trivial way: $\delta(u_\sigma, \sigma') = u_{\sigma'}$. It remains to define the RM reward function $\rho(u, u')$ that enforces the agent to reach goal states. There are many ways of designing such a reward function. A *naive* way of doing it is with a function $\rho^{sparse}(u, u') = \mathbb{1}_{goal}(u')$ that evaluates to 1 if u' is a goal, and zero otherwise. This is not a good idea, because rewards may be very sparse. The challenge becomes designing a non-sparse reward function that guides the RL agent. We note that via potential-based reward shaping, we obtain a family of denser reward functions that induce the same behavior as the naive reward function ρ^{sparse}, which are:

$$\rho(u, u') = \gamma Pot(u') - Pot(u), \text{ where } u' = \delta(u, L(s')).$$

Here, $Pot : U \to \mathbb{R}$ can be *any* potential function that takes bounded values and that satisfies the following two conditions: (i) the potentials in goal states are constant; and (ii) the potentials in goal states are positive.

A. Potentials for Zero-Valued Q Function

In our work we use the potential function $Pot^*(u) := \gamma^{dist(u)}$ that evaluates to 1 in goal states, and vanishes exponentially with their distance to the goal—here, $dist(u)$ is the minimal distance from u to a goal state in the abstract planning graph, and γ is the discount factor of the MDP.

$$Pot^*(u) := \gamma^{dist(u)}$$
$$\rho^*(u, u') := \gamma Pot^*(u') - Pot^*(u)$$

The RM reward function ρ^*_u has a good property: it takes negative values when the agent is "moving away from the goal" in the RM graph, slightly less negative values when the agent "loops" into RM states, and evaluates to exactly *zero* when the agent is "moving closer to the goal". If we further presume that the agent can transition deterministically between different RM states with only one single action at a time (which is true in our experimental setup), then the optimal value function becomes:

$$V^*(s) := \max_a Q^*(s, a) = 0$$

We observe that doing RL in an MDP whose optimal Q values are zero can be advantageous, especially to *deep* RL. Bringing target Q values within a range around 0 benefits training very large DQNs, as it reduces the Internal Covariate Shift from gradient descent optimization [28], [29]. This leads to more stable training since smooth L1 loss gradients are thereby less likely to excessively perturb convolutional network weights [30]. Whereas the same potential-based reward function had been used in [31] and [15], their good properties were not identified and exploited before.

V. DQRMs for Vision-Based Manipulation

In this work, we study DQRMs in the context of vision-based robotic pick-and-place tasks using a UR5 robot arm equipped with a Robotiq 2F-85 parallel-jaw gripper. Our environment is simulated in PyBullet, with fixed RGB-D camera generating top-down views of a $1 \times 0.5\text{m}^2$ planar workspace. Our MDP is partially observable, where the input to our policies are visual observations of the workspace in the form of

RGB images $o_t = \mathbb{R}^{320\times160\times3}$. With respect to the actions, we consider a discrete-time planar parameterization of pick-and-place, where each pixel coordinate (u, v) of the observation o_t corresponds to a picking or placing action executed at that location via camera-to-robot calibration (where depth is used to compute the z height of the target location).

We model our discrete Q-function as a deep Q network that takes as input o_t and outputs a dense pixel-wise prediction of Q values for all actions. This is made fast by using fully convolutional networks (FCNs) [32]—equivalent to standard DQNs sampled over all possible actions, but with the benefit of (i) fast parallelization from highly optimized convolutional operations, and (ii) inductive biases from translational equivariance [33]. Specifically, our DQN is implemented as two independent feed-forward 43-layer Residual Networks (ResNets) [34] – one that outputs a dense pixel-wise prediction of Q values for picking, and the other for placing. During training (with Huber loss, see Sec. III), we pass gradients only through the single pixel and ResNet from which the value prediction of the executed action was computed. All other pixels backpropagate with 0 loss. This architecture has previously been shown to improve learning efficiency for vision-based DQNs [21], [35], [36].

For all tasks, we construct reward machines from demonstrations with the help of hard-coded feature detectors. On a real robot, these features can be obtained by handcrafted heuristics with sensory input, or training CNN's to detect the presence or pose of task-specific objects. Given a robot equipped with RGB-D cameras and simple force sensors, it is feasible to engineer feature detectors that detect whether the end effector is holding a block, the color of such block, and whether a block of certain color is on the table or in the fixture. The value of state features can be used to solve these tasks at an abstract level—e.g., if one block is not on their designated plate, then *pick up* such block; if the end effector is holding a block, then *place* it on their designated plate. Additional details on the environment setup on our website.[1]

In our tasks, all goal states abstract to the same RM state. The reward function $\rho^*(u, u') := \gamma Pot^*(u') - Pot^*(u)$ is computed with potentials $Pot^*(u) := \gamma^{dist(u)}$ that vanish with the goal distance. For example, a state s that has two blocks placed in their designated plate (and the third block is on the table) is two steps away from the goal, and has potential $Pot(L(s)) = \gamma^2$. Moving optimally from this state (i.e., picking up the remaining block) results in zero-valued reward. Sub-optimal moves incur a negative reward. Our setup satisfies the conditions stated in Section IV-A, and therefore the optimal Q values are zero by construction.

We implemented different versions of DQN in TF-agents [37]. Abstract states are encoded with extra input channels appended to the RGB observation, using a one-hot encoding. Batch training was performed by sampling uniformly from an experience replay buffer of size 1000, and a buffer of size 1000 filled with demonstrations. For additional details related to model architecture, encoding of abstract states, and training hyper parameters please refer to the project website.

[1]https://albercm.github.io/vision-dqrm

Fig. 2: Ablation tests of different versions of DQN. The abstract state variables and reward shaping of our DQRM approach provide guidance and greatly improve the performance of DQN on the two block stacking task.

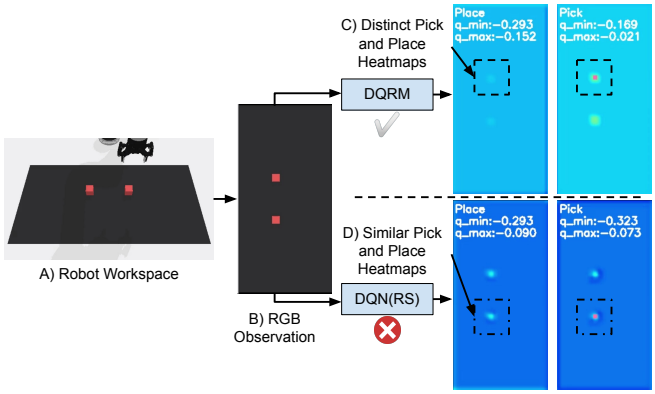


Fig. 3: A) A two block stacking task. B) Top down RGB observation. C) DQRM takes abstract states as part of the input, and is able to make a clear distinction on the right high-level action *pick* (planning) and their low-level parameters (control). D) DQN(RS) does not take abstract states as part of the input, and does not clearly discriminate between high-level actions. The lower performance in DQN(RS) is not due to the lack of capacity of the neural network, nor to the difficulty of the classification task.

- DQN: Baseline with rewards 1 in goal state transitions.
- DQN(RS): DQN with the RMs dense reward shaping.
- DQN(AS): DQN with observations augmented with the RM state (abstract state), and sparse rewards as in DQN.
- DQRM: The combination of DQN(AS) and DQN(RS).

VI. An Intriguing Failure of DQN

Consider a task with two red blocks, where the objective is to stack one on top of another. The solution is conceptually simple: *picking* of one of the blocks, followed by *placing* it on top of the remaining one. The quality of the policies learned with standard DQN is very poor as shown in Figure 2 in part because rewards are sparse. What is more surprising is that a version of DQN with dense potential-based reward shaping (DQN(RS)) is not able to learn good-quality policies either. The Q value heatmaps for the actions *pick* and *place* explain this phenomenon (Figure 3). With two blocks on the table, the Q values for *pick* and *place* look similar to each other and take similar values. This means that the agent may execute a *place* action whilst the right action to execute is *pick*.

Why this is a challenging task to DQN? In this task the branching factor is large, and only a small fraction of the

State:	Two blocks on table		One block on table, one in hand	
Action:	Pick	Place	Pick	Place
Optimal max Q-Value	0	-0.147	-0.21	0
DQRM	-0.021	-0.152	-0.355	-0.061
DQN(RS)	-0.073	-0.090	-0.150	-0.112

TABLE I: Shows the optimal max Q value for the stacking blocks task, with two red blocks with a discount factor of $\gamma = 0.7$. In this task, rewards are reshaped with potentials that vanish exponentially with γ. For DQRM & DQN(RS), max Q-values from an episode are listed for both the pick and place heatmaps which tend to be centered on a block. Notice the DQRM values are much closer to the optimal Q-values. Also the Q-Values for DQN(RS) for pick and place actions are often very close, while pick and place action max Q-values are more distinct for DQRM.

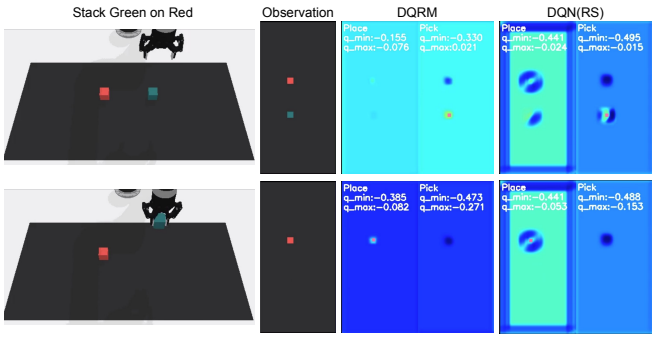


Fig. 4: Green on Red Stacking.

actions are optimal. It appears to be relatively easy to learn that optimal actions correlate with moving the gripper to the location of the blocks. However, it is more challenging to learn a policy that condition actions on the number of blocks in the observation, which is a latent feature.

Why learning the right Q values is difficult? At the beginning of training, we have a replay buffer with 1000 demonstrations. When the DQN is randomly initialized, it begins to execute actions, but will fail most of the time. Since we are uniformly sampling from our replay buffer, we mostly sample transitions from the demos, which train both the pick and place networks to regress to 0 Q-values. Negative samples accumulate over time (as the DQN rolls out), but after a few thousand training iterations, our Q network is now really good at predicting 0 values on red pixels, negative values everywhere else. However, once the robot executes a pick, now it runs into an issue: both pick and place networks still predict 0 values on the remaining red pixels (and negative everywhere else). So it doesn't know which action to select (pick or place). In fact, it is more likely to choose pick again here because the place network may have training data samples that regress it to negative values on red pixels when it accidentally executes a place with an empty gripper. After it executes this failed pick, it now has a corrective training data point with a negative Q value. But the number of transitions where it executes two consecutive picks on red pixels is relatively few in the replay buffer. So it might get stuck at this local minima. This is in contrast to DQRMs where we (i) feed in the abstract state ID as input, and (ii) have target values from the reward shaping, so that there's an input inductive bias that can help us learn a more well behaved pick network and place network.

A closer look into the Q values. The optimal Q values are, by construction of the shaping rewards, zero-valued. Table I shows a more detailed analysis of the values of the optimal Q function, Q^*. We distinguish between *pick* and *place* actions. When the table has two blocks, the optimal action is *pick*, parametrized with the coordinates surrounding the center mass of the blocks on the table. All other actions are suboptimal, and won't grasp a block. When the table has one block (and the gripper is holding the other block), the optimal action is *place*, parametrized with the coordinates surrounding the center mass of the block on the table. Other *place* actions are suboptimal, and will place the block on the table—therefore returning to the initial abstract state. Pick actions are less suboptimal, because the gripper will keep holding one of the blocks. We found that the DQRM learned a Q function that is close to optimal, and separates the Q values of optimal and sub-optimal actions. In contrast, the maximum Q values of the *pick* and *place* networks in DQN(RS) are more similar to each other. DQN(RS) still manifests good task success rates, but the agent sometimes performs the wrong action (e.g., *pick* instead of *place*).

VII. Experiments

In general, DQN has to learn a policy that conditions actions on latent observation features. When meaningful features are difficult to learn (even if salient to a human observer), the overall RL process can become challenging. We saw such phenomenon in Section VI with a simple stacking blocks task.

In our experiments below, we find that DQN(RS) can improve their performance by making latent features more salient (the color and geometry of the blocks). Abstract states in DQRM serve the same purpose: they signal certain state properties that are correlated with optimal actions.

A. A Deeper Look Into Stacking Blocks

We evaluated our algorithms on three variations of the stacking blocks task, listed below, in which we varied the color and geometry of the blocks. We trained each agent for 800 training steps (this is, 32,000 environment interactions).

These artificial variations were done to explore what types of features our Resnet-based architecture fails to distinguish. In or prior experiment DQN(RS) did not perform well (Figure 2), perhaps because the number of blocks on the table is not sufficiently salient to the agent. We wanted to adjust the environment and see what sorts of features would be more salient to DQN(RS). Our premise was that color and geometry could be strong-enough signals to help DQN(RS) achieve good performance, without the need for abstract states. We ran the following additional experiments:

- Stack Green on Red (Figure 4): The objective is to stack the green block on top of the red block.
- Stack Small on Big (Figure 5): The objective is to stack the small block on top of the big block.

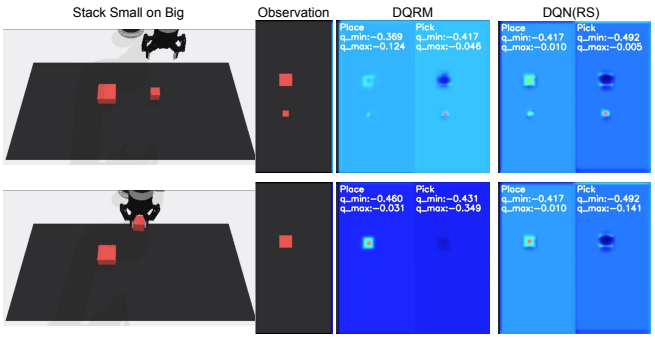


Fig. 5: Small on Big Stacking.

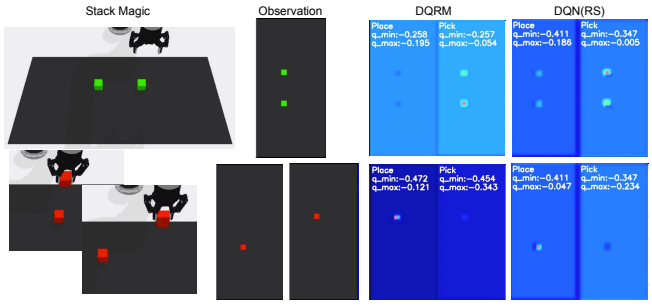
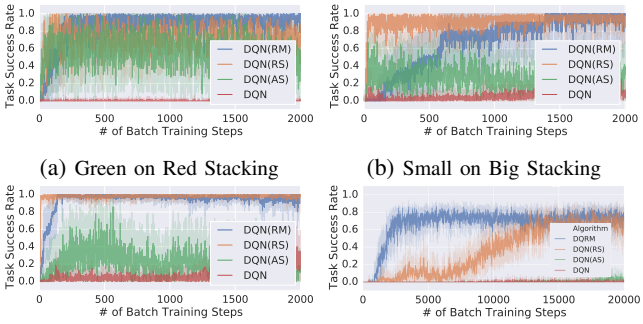


Fig. 6: Magic block stacking.



(a) Green on Red Stacking (b) Small on Big Stacking

(c) Magic Stacking (d) 3 Block Kitting

Fig. 7: Ablations on variations of two block stacking and kitting tasks.

- Stack Magic Blocks (Figure 6): The objective is to stack one block on top another. The block color changes dynamically (magic, since we are in simulation.) based on whether the gripper is holding a block or not.

In the *Stack Green on Red* task we see a similar phenomenon as with the two red blocks, although DQN(RS) is more successful than before. The heatmaps for DQN(RS) are very similar regardless of the number of blocks in the observation as shown in the DQN(RS) column of Figure 4. In contrast, the DQRM heatmaps are clearly different from each other depending on how many blocks are on the table. Without abstract states, DQN(RS) struggles in the case of two similarly sized but different color blocks.

In the *Stack Small on Big* task, rather than changing block color, we change the size of one of the blocks as shown in Figure 5. In this case DQN(RS) is able to learn the task showing that abstract states are not necessarily needed as input when the correct action can be predicated on existence of a uniquely sized object in the scene.

Finally as an extreme example, in *Magic Stacking* in Figure 6 we leverage the fact that we are in simulation to have the color of the blocks changed based on the abstract state. Here DQN(RS) is able to learn as well as DQRM.

What these different variations of the two block stacking task demonstrate is that there are certain features that are more salient to our policy, for example the magic blocks or the large and small blocks. In these cases, a DQN policy with reward shaping is able to learn to a reasonable degree of success. There are many other tasks where these features are less interpretable by a DQN despite being salient to a human observer (two red blocks, red and green block) in these tasks, DQRM is hugely beneficial. Often understanding the abstract state from the raw pixels becomes increasingly difficult as the horizon of the task increases, and DQRM becomes a clear winner. We explore this further in the next section.

B. Kitting Challenge Task

We evaluate the performance of DQRM in a longer-horizon *kitting* task, where the objective is to place three colored blocks in designated bins on a fixture (Figure 1). To construct a RM from demonstrations, we have feature detectors for whether each block is in hand, in it's designated bin or on the table.

In our experiments, DQRM greatly outperformed the other ablations. Figure 7d summarizes our experimental results: DQRM could learn better-quality policies with fewer batch training steps, and their success rate was more stable. DQRM needed only 2,000 batch training steps to converge to a good-quality policy. In comparison, DQN(RS) needed 10,000 batch training steps to achieve an acceptable success rate, and DQN(AS) was not able to learn a good-quality policy. Our results evidence the practicality of RMs, and suggest that their improved performance emanates from the *combination* of abstract states and dense reward shaping.

VIII. Conclusion

Using reward machines for deep reinforcement learning can improve sample efficiency and the quality of the policies learned. We illustrate the benefits of reward machines in vision based robotic manipulation tasks, which may justify the extra cost of having to construct them. In many tasks, the correct next action is predicated on a small number of pixels, determining whether an object is in hand or already placed correctly. We highlight how DQN struggles when this is the case, even on a very simple two block stacking example. DQRM outperforms DQN by using the two supervisory signals of reward shaping and knowledge of current abstract state coming from the reward machine. These signals complement each other and can both be used to improve policy performance as demonstrated with our DQN(RS) and DQN(AS) ablations. This work opens up exciting opportunities to tackle longer

horizon robotics tasks as demonstrated with our challenging 3 block kitting task.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[3] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv preprint arXiv:1806.10293*, 2018.

[4] J. Karlsson, "Task decomposition in reinforcement learning," in *Proceedings of the AAAI Spring Symposium on Goal-Driven Learning*, 1994.

[5] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[6] J. Merel, A. Ahuja, V. Pham, S. Tunyasuvunakool, S. Liu, D. Tirumala, N. Heess, and G. Wayne, "Hierarchical visuomotor control of humanoids," in *Proceedings of the 7th International Conference on Learning Representations, (ICLR)*, 2019.

[7] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "MCP: Learning composable hierarchical control with multiplicative compositional policies," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 3686–3697.

[8] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Learning sequential motor tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 2626–2632.

[9] J. D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings," in *Proc. of the 35th Intl. Conference on Machine Learning, (ICML)*, 2018, pp. 1008–1017.

[10] L. Li, T. J. Walsh, and M. L. Littman, "Towards a unified theory of state abstraction for MDPs," in *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006.

[11] D. Abel, D. E. Hershkowitz, and M. L. Littman, "Near optimal behavior via approximate state abstraction," in *Proc. 33rd Intl. Conf. on Machine Learning (ICML)*, 2016, pp. 2915–2923.

[12] R. Givan, T. L. Dean, and M. Greig, "Equivalence notions and model minimization in Markov Decision Processes," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 163–223, 2003.

[13] D. Abel, D. Arumugam, K. Asadi, Y. Jinnai, M. L. Littman, and L. L. S. Wong, "State abstraction as compression in apprenticeship learning," in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019, pp. 3134–3142.

[14] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 2112–2121.

[15] A. Camacho, R. Toro-Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning," in *Proc. of the 28th Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, 2019, pp. 6065–6073.

[16] R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith, "Learning reward machines for partially observable reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 15 497–15 508.

[17] U. Viereck, A. ten Pas andz Kate Saenko, and R. P. Jr., "Learning a visuomotor controller for real world robotic grasping using simulated depth images," in *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, 2017, pp. 291–300.

[18] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[19] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[20] B. Wu, I. Akinola, and P. K. Allen, "Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes," in *IEEE/RSJ IROS*, 2019, pp. 1789–1796.

[21] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[22] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv preprint arXiv:1704.03073*, 2017.

[23] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *IEEE Transactions on Robotics*, 2020.

[24] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[25] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[26] G. Giantamidis and S. Tripakis, "Learning moore machines from input-output traces," in *21st Intl. Symposium on Formal Methods (FM)*, ser. LNCS, vol. 9995, 2016, pp. 291–309.

[27] A. Camacho and S. A. McIlraith, "Learning interpretable models in linear temporal logic," in *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*, 2019, pp. 621–630.

[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Intl. Conf. on Machine Learning (ICML)*, 2015, pp. 448–456.

[29] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[30] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 901–909.

[31] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith, "Non-markovian rewards expressed in LTL: guiding search via reward shaping," in *Proceedings of the 10th International Symposium on Combinatorial Search (SoCS)*, 2017, pp. 159–160.

[32] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[33] R. Kondor and S. Trivedi, "On the generalization of equivariance and convolution in neural networks to the action of compact groups," *International Conference on Machine Learning (ICML)*, 2018.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[35] A. Hundt, B. Killeen, H. Kwon, C. Paxton, and G. D. Hager, ""Good robot!": Efficient reinforcement learning for multi-step visual tasks via reward shaping," *arXiv preprint arXiv:1909.11730*, 2019.

[36] J. Wu, X. Sun, A. Zeng, S. Song, J. Lee, S. Rusinkiewicz, and T. Funkhouser, "Spatial action maps for mobile manipulation," *arXiv preprint arXiv:2004.09141*, 2020.

[37] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow," 2018, [Online; accessed 25-June-2019]. [Online]. Available: https://github.com/tensorflow/agents