Robotics: Science and Systems 2020
Corvalis, Oregon, USA, July 12-16, 2020

Scaling data-driven robotics with reward sketching and batch reinforcement learning

Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova,
Scott Reed, Rae Jeong, Konrad Żołna, Yusuf Aytar, David Budden, Mel Vecerik,
Oleg Sushkov, David Barker, Jonathan Scholz, Misha Denil, Nando de Freitas, Ziyu Wang
Deepmind

***Abstract*—By harnessing a growing dataset of robot experience, we learn control policies for a diverse and increasing set of related manipulation tasks. To make this possible, we introduce reward sketching: an effective way of eliciting human preferences to learn the reward function for a new task. This reward function is then used to retrospectively annotate all historical data, collected for different tasks, with predicted rewards for the new task. The resulting massive annotated dataset can then be used to learn manipulation policies with batch reinforcement learning (RL) from visual input in a completely off-line way, *i.e.*, without interactions with the real robot. This approach makes it possible to scale up RL in robotics, as we no longer need to run the robot for each step of learning. We show that the trained batch RL agents, when deployed in real robots, can perform a variety of challenging tasks involving multiple interactions among rigid or deformable objects. Moreover, they display a significant degree of robustness and generalization. In some cases, they even outperform human teleoperators.**

I. Introduction

Deep learning has successfully advanced many areas of artificial intelligence, including vision [39, 26], speech recognition [24, 46, 4], natural language processing [17], and reinforcement learning (RL) [49, 63]. The success of deep learning in each of these fields was made possible by the availability of huge amounts of labeled training data. Researchers in vision and language can easily train and evaluate deep neural networks on standard datasets with crowdsourced annotations such as ImageNet [58], COCO [45] and CLEVR [33]. In simulated environments like video games, where experience and rewards are easy to obtain, deep RL is tremendously successful in outperforming top skilled humans by ingesting huge amounts of data [63, 69, 9]. The OpenAI Five DOTA bot [9] processes 180 years of simulated experience every day to play at a professional level. Even playing simple Atari games typically requires 40 days of game play [49]. In contrast, in robotics we lack abundant data since data collection implies execution on a real robot, which cannot be accelerated beyond real time. Furthermore, task rewards do not naturally exist in the real-world robotics as it is the case in simulated environments. The lack of large datasets with reward signals has limited the effectiveness of deep RL in robotics.

This paper presents a data-driven approach to apply deep RL effectively to learn to perform manipulation tasks on real robots from vision. Our solution is illustrated in Fig. 1. At its heart are three important ideas: (i) efficient elicitation of user preferences to learn reward functions, (ii) automatic annotation of all historical data with any of the learned reward functions, and (iii) harnessing the large annotated datasets to learn policies purely from stored data via batch RL.

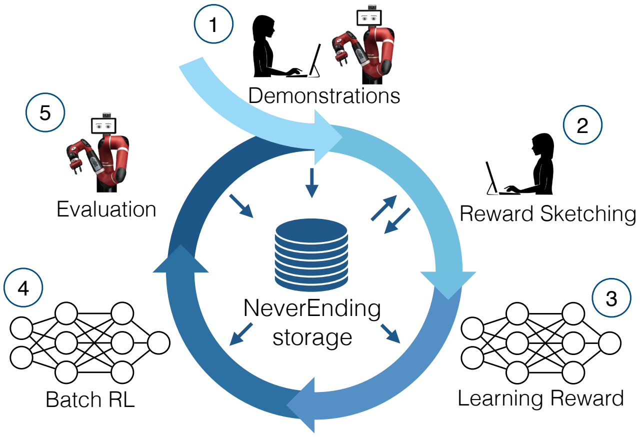

Fig. 1: Our cyclical approach for *never-ending* collection of data and continual learning of new tasks consists of five stages: (1) generation of observation-action pairs by either teleoperation, scripted policies or trained agents, (2) a novel interactive approach for eliciting preferences for a specific new task, (3) learning the reward function for the new task and applying this function to automatically label all the historical data, (4) applying batch RL to learn policies purely from the massive growing dataset, without online interaction, and (5) evaluation of the learned policies.

Existing RL approaches for real-world robotics mainly focus on tasks where hand-crafted reward mechanisms can be developed. Simple behaviours such as learning to grasp objects [35] or learning to fly [23] by avoiding crashing can be acquired by reward engineering. However, as the task complexity increases, this approach does not scale well. We propose a novel way to specify rewards that allows to generate reward labels for a large number of diverse tasks. Our approach relies on human judgments about progress towards the goal to train task-specific reward functions. Annotations are elicited from humans in the form of per-timestep reward annotations using a process we call *reward sketching*, see Fig. 2. The sketching procedure is intuitive for humans, and allows them to label many timesteps rapidly and accurately. We use the human annotations to train a ranking reward model, which is then used to annotate all other episodes.

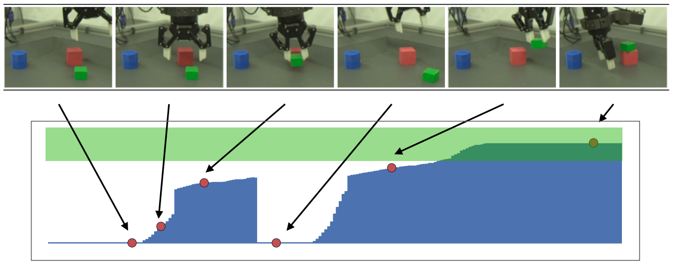

Fig. 2: Reward sketching procedure. Sketch of a reward function for `stack_green_on_red` task. A video sequence (top) with a *reward sketch* (bottom), shown in blue. Reward is the perceived progress towards achieving the target task. The annotators are instructed to indicate successful timesteps with reward high enough to reach green area.

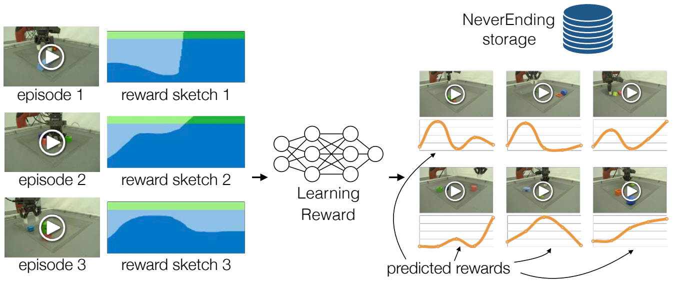

Fig. 3: Retrospective reward assignment. The reward function is learned from a limited set of episodes with reward sketches. The learned reward function is applied to a massive dataset of episodes from NeverEnding Storage. All historical episodes are now labelled with a newly learned reward function.

To generate enough data to train data-demanding deep neural network agents, we record experience continuously and persistently, regardless of the purpose or quality of the behavior. We collected over 400 hours of multiple-camera videos (Fig. 6), proprioception, and actions from behavior generated by human teleoperators, as well as random, scripted and trained policies. By using deep reward networks obtained as a result of reward sketching, it becomes possible to retrospectively assign rewards to any past or future experience for any given task. Thus, the learned reward function allows us to repurpose a *large* amount of past experience using a *fixed* amount of annotation effort per task, see Fig. 3. This large dataset with task-specific rewards can now be used to harness the power of deep batch RL.

For any given new task, our data is necessarily off-policy, and is typically off-task (*i.e.*, collected for other tasks). In this case, batch RL [41] is a good method to learn visuomotor policies. Batch RL effectively enables us to learn new controllers without execution on the robot. Running RL off-line gives researchers several advantages. Firstly, there is no need to worry about wear and tear, limits of real-time processing, and many of the other challenges associated with operating real robots. Moreover, researchers are empowered to train policies using their batch RL algorithm of choice, similar to how vision researchers are empowered to try new methods on ImageNet. To this end, we release datasets [16] with this paper.

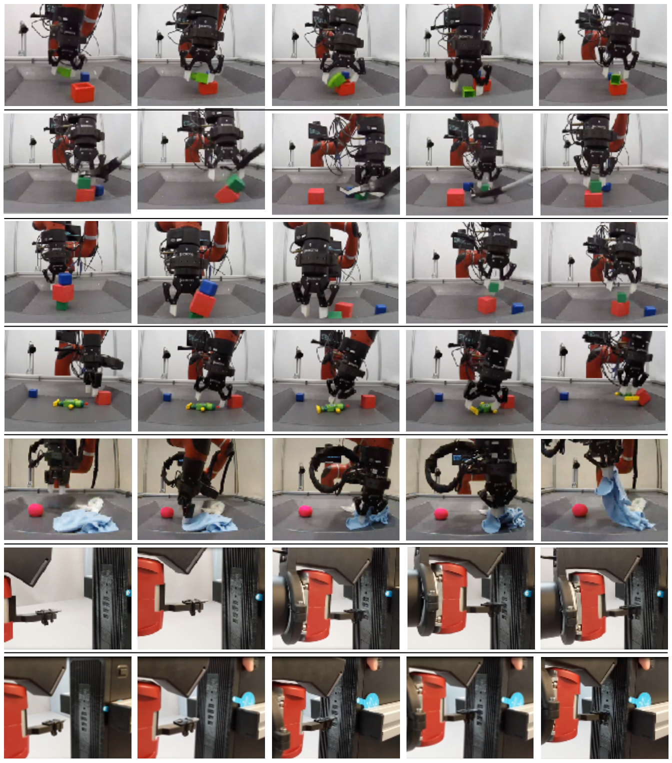

Fig. 4: Each row is an example episode of a successful task illustrating: (1) the ability to recover from a mistake in `stack_green_on_red` task, (2) robustness to adversarial perturbations in the same task, (3) generalization to unseen initial conditions in the same task, 4) generalizing to previously unseen objects in a `lift_green` task, (5) the ability to lift deformable objects, (6) inserting a USB key, (7) inserting a USB key despite moving the target computer.

The integration of all the elements into a scalable system to tightly close the loop of human input, reward learning and policy learning poses substantial engineering challenges. Nevertheless, this work is essential to advance the data-driven robotics. For example, we store all robot experience including demonstrations, behaviors generated by trained policies or scripted random policies. To be useful in learning, this data needs to be appropriately annotated and queried. This is achieved thanks to a design of our storage system dubbed NeverEnding Storage (NES).

This multi-component system (Fig. 1) allows us to solve a variety of challenging tasks (Fig. 4) that require skillful manipulation, involve multi-object interaction, and consist of many time steps. An example of such task is stacking arbitrarily shaped objects. In this task, small perturbations at the beginning can easily cause failure later: The robot not only has to achieve a successful grasp, but it must also grasp the first object in a way that allows for safe placement on top of the second object. Moreover, the second object may have a small surface area which varies how demanding the task is. Learning policies directly from pixels makes the task more challenging, but eliminates the need for feature engineering and allows for additional generalization capacity. While some of our tasks can be solved effectively with scripted policies, learning policies that generalize to arbitrary shapes, sizes, textures and materials remains a formidable challenge, and hence the focus of this paper is on making progress towards meeting this challenge.

As shown in Fig. 4, the policies learned with our approach solve a variety of tasks including lifting and stacking of rigid/deformable objects, as well as USB insertion. Importantly, thanks to learning from pixels, the behaviour generalizes to new object shapes and to new initial conditions, recovers from mistakes and is robust to some real-time adversarial interference. Fig. 9 shows that the learned policies can also solve tasks more effectively than human teleoperators. *To better view our results and general approach, we highly recommend watching the accompanying video on the project website.*

The remainder of this paper is organized as follows. Sec. II introduces the methods, focusing on reward sketching, reward learning and batch RL, but also provides the bigger context highlighting the engineering contributions. Sec. III is devoted to describing our experimental setup, network architectures, benchmark results, and an interactive insertion task of industrial relevance. Sec. IV explores some of the related work.

II. Methods

The general workflow is illustrated in Fig. 1 and a more detailed procedure is presented in Fig. 5. NES accumulates a large dataset of task-agnostic experience. A task-specific reward model allows us to retrospectively annotate data in NES with reward signals for a new task. With rewards, we can then train batch RL agents with all the data in NES.

The procedure for training an agent to complete a new task has the following steps which are described in turn in the remainder of the section:

A. A human teleoperates the robot to provide first-person demonstrations of the target task.
B. All robot experience, including demonstrations, is accumulated into NES.
C. Humans annotate a subset of episodes from NES (including task-specific demos) with reward sketches.
D. A reward model for the target task is trained using the fixed amount of labelled experience.
E. An agent for the target task is trained using all experience in NES, using the predicted reward values.
F. The resulting policy is deployed on a real robot, while recording more data into NES.
G. Occasionally we select an agent for careful evaluation, to track overall progress on the task.

A. Teleoperation

To specify a new target task, a human operator first remotely controls the robot to provide several successful (and occasionally unsuccessful[1]) examples of completing the task. By employing the demonstration trajectories, we facilitate both reward learning and reinforcement learning tasks. Demonstrations help to bootstrap the reward learning by providing examples of successful behavior with high rewards, which are also easy to interpret and judge for humans. In RL, we circumvent the problem of exploration: Instead of requiring that the agent explores the state space autonomously, we use expert knowledge about the intended outcome of the task to guide the agent. In addition to full episodes of demonstrations, when an agent controls the robot, *interactive interventions* can be also performed: A human operator can take over from, or return control to, an agent at any time. This data is useful for fixing particular corner cases that the agents might encounter.

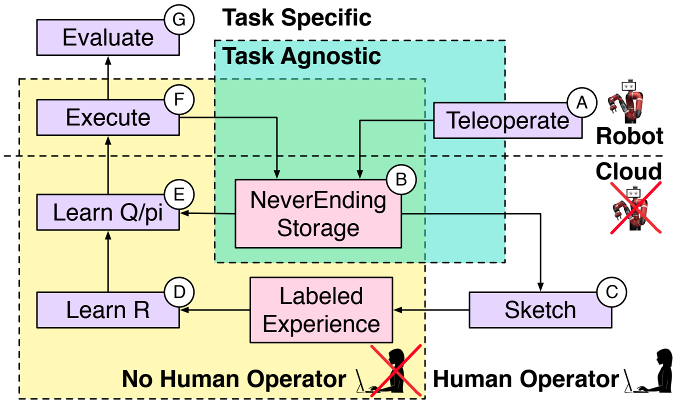

Fig. 5: Structure of the data-driven workflow. Each step is described in Sec. II and the figure highlights which steps are performed on the robot or not, involving human operator or not and if they are task-specific or task-agnostic.

The robot is controlled with a 6-DoF mouse with an additional gripper button (see the video) or hand-held virtual reality controller. A demonstrated sequence contains pairs of observations and corresponding actions for each time step t: $((x_0, a_0), \ldots, (x_t, a_t), \ldots, (x_T, a_T))$. Observations x_t contain all available sensor data including raw pixels from multiple cameras as well as proprioceptive inputs (Fig. 6).

B. NeverEnding Storage

NES captures all of the robot experience generated across all tasks in a central repository. This allows us to make use of historical data each time when learning a new target task, instead of generating a new dataset from scratch. NES includes teleoperated trajectories for various tasks, human play data, and experience from the execution of either scripted or learned policies. For every trajectory we store recordings from several cameras and sensors in the robot cage (Fig. 6). The main innovation in NES is the introduction of a rich metadata system into the RL training pipeline. It is implemented as a relational database that can be accessed using SQL-type queries. We attach environment and policy metadata to every trajectory (e.g., date and time of operation), as well as arbitrary human-readable labels and reward sketches. This information allows us to dynamically retrieve and slice the data relevant for a particular stage of our training pipeline.

C. Reward Sketching

The second step in task specification is *reward sketching*. We ask human experts to provide per-timestep annotations

[1] We notice that in our dataset around 15% of human demonstrations fail to accomplish the task at the end of the episode.

of reward using a custom user interface. As illustrated in Fig. 2, the user draws a curve indicating the progress towards accomplishing the target task as a function of time, while the interface shows the frame corresponding to the current cursor position. This intuitive interface allows a single annotator to produce hundreds of frames of reward annotations per minute.

To sketch an episode, a user interactively selects a frame x_t and provides an associated reward value $s(x_t) \in [0, 1]$. The sketching interface allows the annotator to draw reward curves while "scrubbing" through a video episode, rather than annotating frame by frame. This efficient procedure provides a rich source of information about the reward across the entire episode. The sketches for an episode $\{s(x_t)\}|_{t=1}^{T}$ are stored in NES as described in Sec. II-B.

The reward sketches allow comparison of perceived value of any two frames. In addition, the green region in Fig. 2 is reserved for frames where the goal is achieved. For each task the episodes to be annotated are drawn from NES. They include both the demonstrations of the target task, as well as experience generated for prior tasks. Annotating data from prior tasks ensures better coverage of the state space.

Sketching is particularly suited for tasks where humans are able to compare two timesteps reliably. Typical object manipulation tasks fall in this category, but not all robot tasks are like this. For instance, it would be hard to sketch tasks where variable speed is important, or with cycles as in walking. While we are aware of these limitations, the proposed approach does however cover many manipulation tasks of interest as shown here. We believe future work should advance interfaces to address a wider variety of tasks.

D. Reward Learning

The reward annotations produced by sketching are used to train a reward model. This model is then used to predict reward values for all experience in NES (Fig. 3). As a result, we can leverage all historical data in training a policy for a new task, without manual human annotation of the entire repository.

Episodes annotated with reward sketches are used to train a reward function in the form of neural network with parameters ψ in a supervised manner. We find that although there is high agreement between annotators on the relative quality of timesteps within an episode, annotators are often not consistent in the overall *scale* of the sketched rewards. We therefore adopt an intra-episode ranking approach to learn reward functions, rather than trying to regress the sketched values directly.

Specifically, given two frames x_t and x_q in the same episode, we train the reward model to satisfy two conditions. First, if frame x_t is (un)successful according to the sketch $s(x_t)$, it should be (un)successful according the estimated reward function $r_\psi(x_t)$. The successful and unsuccessful frames in reward sketches are defined by exceeding or not a threshold τ_s, the (un)successful frames in the predicted reward exceed (or not) a threshold τ_{r1} (τ_{r2}). Second, if $s(x_t)$ is higher than $s(x_q)$ by a threshold μ_s, then $r_\psi(x_t)$ should be higher than $r_\psi(x_q)$ by another threshold μ_r. These conditions are captured by the following two hinge losses:

$$\begin{aligned}\mathcal{L}_{rank}(\psi) &= \max\{0, r_\psi(x_t) - r_\psi(x_q) + \mu_r\} \mathbb{1}_{s(x_q)-s(x_t)>\mu_s}\\ \mathcal{L}_{success}(\psi) &= \max\{0, \tau_{r1} - r_\psi(x)\} \mathbb{1}_{s(x)>\tau_s} +\\ &\qquad \max\{0, r_\psi(x) - \tau_{r2}\} \mathbb{1}_{s(x)<\tau_s}\end{aligned}$$

The total loss is obtained by adding these terms: $\mathcal{L}_{rank} + \lambda\mathcal{L}_{success}$. In our experiments, we set $\mu_s = 0.2$, $\mu_r = 0.1$, $\tau_s = 0.85$, $\tau_{r1} = 0.9$, $\tau_{r2} = 0.7$, and $\lambda = 10$.

E. Batch RL

We train policies using batch RL [41]. In batch RL, the new policy is learned using a single batch of data generated by different previous policies, and without further execution on the robot. Our agent is trained using only distributional RL [7], without any feature pretraining, behaviour cloning (BC) initialization, any special batch correction terms, or auxiliary losses. We do, however, find it important to use the historical data from other tasks.

Our choice of distributional RL is partly motivated by the success of this method for batch RL in Atari [1]. We compare the distributional and non-distributional RL alternatives in our experiments. We note that other batch RL methods (see Sec. IV) might also lead to good results. Because of this, we release our datasets [16] and canonical agents [28] to encourage further investigation and advances to batch RL algorithms in robotics.

We use an algorithm similar to D4PG [7, 28] as our training algorithm. It maintains a value network $Q(x_t, h_t^Q, a \,|\, \theta)$ and a policy network $\pi(x, h_t^\pi \,|\, \phi)$. Given the effectiveness of recurrent value functions [36], both Q and π are recurrent with h_t^Q and h_t^π representing the corresponding recurrent hidden states. The target networks have the same structure as the value and policy networks, but are parameterized by different parameters θ' and ϕ', which are periodically updated to the current parameters of the original networks.

Given the Q function, we update the policy using DPG [62]. As in D4PG, we adopt a distributional value function [8] and minimize the associated loss to learn the critic. During learning, we sample a batch of sequences of observations and actions $\{x_t^i, a_t^i, \cdots, x_{t+n}^i\}_i$ and use a zero start state to initialize all recurrent states at the start of sampled sequences. We then update ϕ and θ using BPTT [70].

Since NES contains data from many different tasks, a randomly sampled batch from NES may contain data mostly irrelevant to the task at hand. To increase the representation of data from the current task, we construct fixed ratio batches, with 75% of the batch drawn from the entirety of NES and 25% from the data specific to the target task. This is similar to the solution proposed in previous work [54], where fixed ratio batches are formed with agent and demonstration data.

F. Execution

Once an agent is trained, we can run it on the real robot. By running the agent, we collect more experience, which can be used for reward sketching or RL in future iterations. Running the agent also allows us to observe its performance and make judgments about the steps needed to improve it.

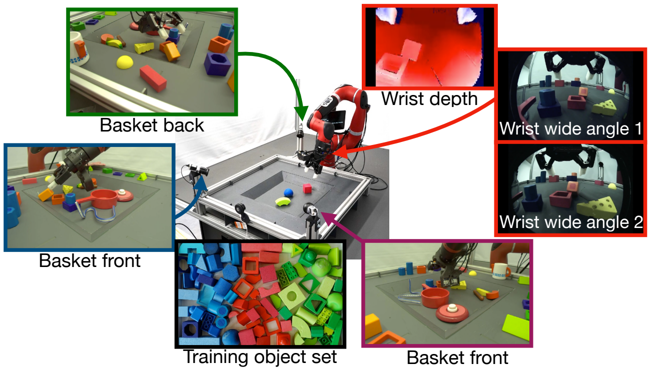

Fig. 6: The robot senses and records all data acquired with its 3 cage cameras, 3 wrist cameras (wide angle and depth) and proprioception. It also records its actions continuously. The robot is trained with a wide variety of object shapes, textures and sizes to achieve generalization at deployment time.

In early workflow iterations, before the reward functions are trained with sufficient coverage of state space, the policies often exploit "delusions" where high rewards are assigned to undesired behaviors. To fix a reward delusion, a human annotator sketches some of the episodes where the delusion is observed. New annotations are used to improve the reward model, which is used in training a new policy. For each target task, this cycle is typically repeated 2–3 times until the predictions of a reward function are satisfactory.

III. Experiments

A. Experimental Setup

Robotic setup: Our setup consists of a Sawyer robot with a Robotiq 2F-85 gripper and a wrist force-torque sensor facing a 35×35 cm basket. The action space has six continuous degrees of freedom, corresponding to Cartesian translational and rotational velocity targets of the gripper pinch point and one binary control of gripper fingers. The agent control loop is executed at 10Hz. For safety, the pinch point movement is restricted to be in a $35 \times 35 \times 15$ cm workspace with maximum rotations of 30°, 90°, and 180° around each axis.

Observations are provided by three cameras around the cage, as well as two wide angle cameras and one depth camera mounted at the wrist, and proprioceptive sensors in the arm (Fig. 6). NES captures all of the observations, and we indicate what subset is used for each learned component.

Tasks and datasets: We focus on 2 subsets of NES, with data recorded during manipulation of 3 variable-shape rigid objects coloured red, green and blue (*rgb* dataset, Fig. 6), and 3 deformable objects: a soft ball, a rope and a cloth (*deformable* dataset, Fig. 4, row 5). The *rgb* dataset is used to learn policies for two tasks: `lift_green` and `stack_green_on_red`, and the *deformable* dataset is used for the `lift_cloth` task. Statistics for both datasets are presented in Tab. I which describes how much data is teleoperated, how much comes from the target tasks and how much is obtained by random scripted policies. Each episode lasts for 200 steps (20 seconds) unless it is terminated earlier for safety reasons.

Type	No. Episodes	No. steps	Hours
Teleoperation	6.2 K	1.1 M	31.9
`lift_green`	8.5 K	1.5 M	41.3
`stack_green_on_red`	10.3 K	2.0 M	56.1
`random_watcher`	13.1 K	2.6 M	70.9
Total	37.9 K	7.0 M	193.3

(a) *RGB* dataset.

Type	No. Episodes	No. steps	Hours
Teleoperation	2.8 K	568 K	15.8
`lift_cloth`	13.3 K	2.4 M	66.0
`random_watcher`	6.0 K	1.2 M	32.1
Total	36.5 K	6.9 M	191.2

(b) *Deformable* dataset.

TABLE I: Dataset statistics. Total includes off-task data not listed in individual rows, teleoperation and tasks `lift_green`, `stack_green_on_red`, `lift_cloth` partly overlap.

To generate initial datasets for training we use a scripted policy called the `random_watcher`. This policy moves the end effector to randomly chosen locations and opens and closes the gripper at random times. When following this policy, the robot occasionally picks up or pushes the objects, but is typically just moving in free space. This data not only serves to seed the initial iteration of learning, but removing it from the training datasets degrades performance of the final agents.

The datasets contain a significant number of teleoperated episodes. The majority are recorded via interactive teleoperation (Sec. II-A), and thus require limited human intervention. Only about 600 full teleoperated episodes correspond to the `lift_green` or `stack_green_on_red` tasks.

There are 894, 1201, and 585 sketched episodes for the `lift_green`, `stack_green_on_red` and `lift_cloth` tasks, respectively. Approximately 90% of the episodes are used for training and 10% for validation. The sketches are not obtained all at once, but accumulated over several iterations of the process illustrated in Fig. 1. At the first iteration, the humans annotate randomly sampled demonstrations. In next iterations, the annotations are usually done on agent data, and occasionally on demonstrations or random watcher data. Note that only a small portion of data from NES is annotated.

Agent network architecture: The agent network is illustrated in Fig. 7. Each camera is encoded using a residual network followed by a spatial softmax keypoint encoder with 64 channels [42]. The spatial softmax layer produces a list of 64 (x, y) coordinates. We use one such list for each camera and concatenate the results.

Before applying the spatial softmax, we add noise from the distribution $\mathcal{U}[-0.1, 0.1]$ to the logits so that the network learns to concentrate its predictions, as illustrated with the circles in Fig. 7. Proprioceptive features are concatenated, embedded with a linear layer, layer-normalized [5], and finally mapped through a $\tanh$ activation. They are then appended to the camera encodings to form the joint input features.

The actor network $\pi(x)$ consumes these joint input features directly. The critic network $Q(x, a)$ additionally passes them

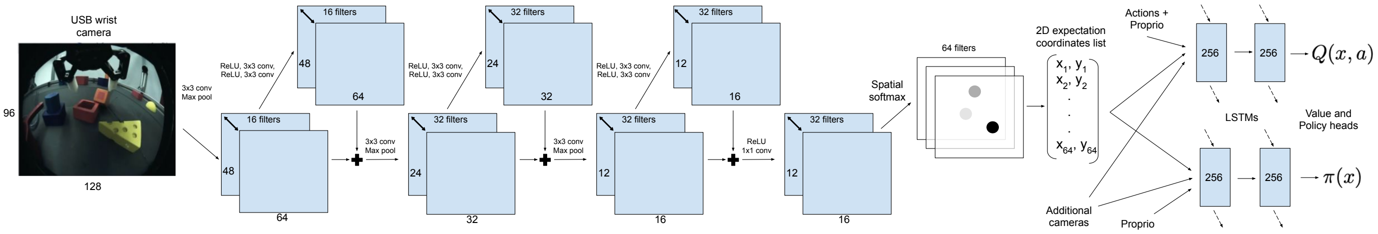

Fig. 7: Agent network architecture. Only the wrist camera encoder is shown here, but in practice we encode each camera independently and concatenate the results.

through a linear layer, concatenates the result with actions passed through a linear layer, and maps the result through a linear layer with ReLU activations. The actor and critic networks each use two layer-normalized LSTMs with 256 hidden units. Action outputs are further processed through a tanh layer placing them in the range $[-1, 1]$, and then re-scaled to their native ranges before being sent to the robot.

The agent for `lift_green` and `stack_green_on_red` tasks observes two cameras, a basket front left camera (80×128) and one of wrist-mounted wide angle cameras (96×128) (Fig. 6). The agent for `lift_cloth` uses an additional back left camera (80×128).

Reward network architecture: The reward network is a non-recurrent residual network with a spatial softmax layer [42] as in the agent network architecture. We also use the proprioceptive features as in the agent learning. As the sketched values are in the range of $[0, 1]$, the reward network ends with a sigmoid non-linearity.

Training: We train multiple RL agents in parallel and briefly evaluate the most promising ones on the robot. Each agent is trained for 400k update steps. To further improve performance, we save all episodes from RL agents, and sketch more reward curves if necessary, and use them when training the next generation of agents. We iterated this procedure 2–3 times and at each iteration the agent becomes more successful and more robust. Three typical episodes from three steps of improvement in `stack_green_on_red` task are depicted in Fig. 8. They correspond to agents trained using approximately 82%, 94% and 100% of the collected data. In the first iteration, the agent could pick a green block, but drops it. In the second iteration, the agent attempts stacking a green block on red, and only in the third iteration it succeeds in it. Next, we report the performance of the final agents.

Evaluation: While the reward and policy are learned from data, we cannot assess their ultimate quality without running the agent on the real robot. That is, we need to evaluate whether our agents learned using the stored datasets transfer to the real robot. As the agent is learned off-line, good performance on the real robot is a powerful indicator of generalization.

To this end, we conducted controlled evaluations on the physical robot with fixed initial conditions across different policies. For the `lift_green` and `stack_green_on_red` datasets, we devise three different evaluation conditions with varying levels of difficulty:

1) *normal*: basic rectangular green blocks (well represented in the training data), large red objects close to the center;
2) *hard*: more diverse objects (less well represented in the training data), smaller red objects with diverse locations;
3) *unseen*: green objects that were never seen during training, large red objects.

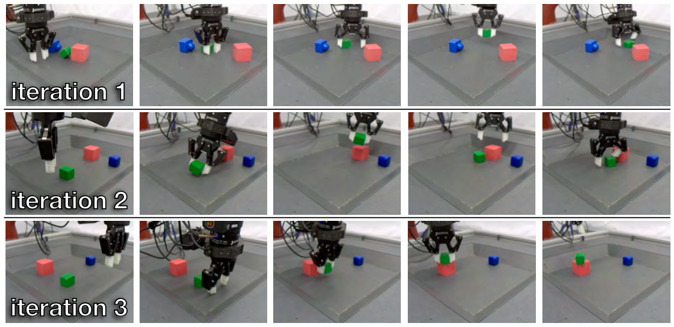

Fig. 8: Iterative improvement of the agent on task `stack_green_on_red`. Each iteration corresponds to a cycle through the steps as shown in Fig. 1. With more training data, the performance of agent improves.

Each condition specifies 10 different initial positions of the objects (set by a human operator) as well as the initial pose of the robot (set automatically). The *hard* and *unseen* conditions are especially challenging, since they require the agent to cope with novel objects and novel object configurations.

We use the same 3 evaluation sets for both the `lift_green` and `stack_green_on_red` tasks. To evaluate the `lift_cloth` task, we randomize the initial conditions at every trial. As a quality metric, we measure the rate of successfully completed episodes, where success is indicated by a human operator.

B. Results

Results on the *rgb* dataset are summarized in Tab. II. Our agent achieves a success rate of 80% for lifting and 60% for stacking. Even with rarely seen objects positioned in adversarial ways, the agent is quite robust with success rates being 80% and 40%, respectively. Remarkably, when dealing with objects that were never seen before, it can lift or stack them in 50% and 40% of cases (see Fig. 4 for examples of such behavior). The success rate of our agent for the `lift_cloth` task in 50 episodes with randomized initial conditions is 74%.

Our results compare favorably with those of Zhu et al. [73], where block lifting and stacking success rates are 64% and 35%. Note that these results are not perfectly comparable due to different physical setups, but we believe they provide some

Agent	Normal	Hard	Unseen
Our approach	**80%**	**80%**	**50%**
No random watcher data	**80%**	70%	20%
Only lift data	0%	0%	0%
Non-distributional RL	30%	20%	10%

(a) `lift_green`

Agent	Normal	Hard	Unseen
Our approach	**60%**	**40%**	**40%**
No random watcher data	50%	30%	30%
Only stacking data	0%	10%	0%
Non-distributional RL	20%	0%	0%

(b) `stack_green_on_red`.

TABLE II: The success rate of our agent and ablations for a given task in different difficulty settings. Recall that out agent is trained off-line.

guidance. Wulfmeier et al. [72] also attempted reward learning with the block stacking task. Instead of learning directly from pixels, they rely on QR-code state estimation for a fixed set of cubes, whereas our policies can handle objects of various shapes, sizes and material properties. Jeong et al. [31] achieve 62% accuracy on block stacking (but with a fixed set of large blocks) using a sim2real approach with continuous 4-DoF control. In contrast, we can achieve similar performance with a variety of objects and more complex continuous 6-DoF control.

To understand the benefits of relabelling the past experience with learned reward functions, we conduct the ablations with fixed reward functions and varying training subsets for RL agents. Firstly, we train the lifting (stacking) policy using only the lifting (stacking) episodes. Using only task-specific data is interesting because the similarity between training data and target behavior is higher (*i.e.*, the training data is more on-policy). Secondly, we train an agent with access to data from all tasks, but no access to the `random_watcher` data. As this data is unlikely to contain relevant to the task episodes, we want to know how much it contributes to the final performance.

Tab. II show the results of these two ablations. Remarkably, using only a task-specific dataset dramatically degrades the policy (its performance is 0% in almost all scenarios). Random watcher data proves to be valuable as it contributes up to an additional 30% improvement, showing the biggest advantage in the hardest case with unseen objects.

We also evaluate the effect of distributional value functions. Confirming previous findings in Atari [1], the results in the last rows of Tab. II show that distributional value functions are essential for good performance in batch RL.

For qualitative results, we refer the reader to the accompanying video and Fig. 4 that demonstrate the robustness of our agents. The robot successfully deals with adversarial perturbations by a human operator, stacking several unseen and non-standard objects and lifting toys, such as a robot and a pony. Our agents move faster and are more efficient compared to a human operator in some cases as illustrated in Fig. 9.

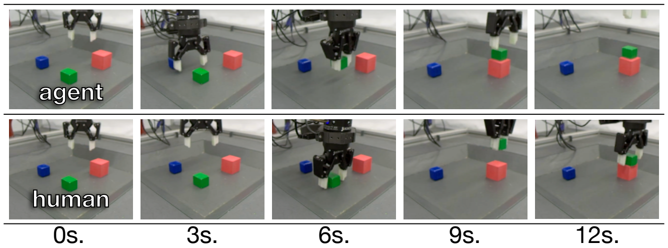

Fig. 9: Agent vs human in `stack_green_on_red` task. We show frames of an episode performed by an agent (top) and a human (bottom) after every 3 seconds. The agent accomplishes the task faster than a human operator.

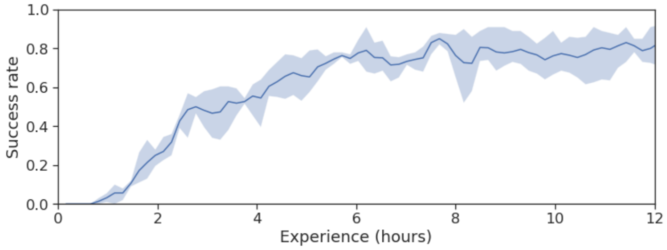

Fig. 10: USB-insertion task success rate during the process of on-line training. It illustrates the rapid progress of training a robot to solve an industrially relevant task.

C. Interactive Insertion

An alternative way to obtain a policy is to perform data collection, reward learning and policy learning in a tight loop. Here, the human operator interactively refines the learned reward function on-line at the same time when a policy is learned. In this experiment, the policy is learned from scratch without relying on historical data and batch RL, which is possible in less data-demanding applications. In this section, we present an example of this approach applied to industrially relevant task: insert a USB key into a computer port.

We consider 6-DoF velocity control. The velocity actions are fed to a stateful safety controller, which uses a previously learned model to limit excess forces and a Mujoco inverse kinematics model to infer target joint velocities. Episodes are set to last 15 seconds with 10 Hz control, for a total of 150 steps. Both the policy and reward model use wrist camera images of size 84×84 pixels.

At the start of each episode, the robot position is set within a $6 \times 6 \times 6$ cm region with $8.6°$ rotation in each direction, and the allowed workspace is $8 \times 8 \times 15$ cm with $17.2°$ rotation. This is known to be significant amount of variation for such task. Episodes are terminated with a discount of zero when the robot reached the boundary of the workspace. For faster convergence, a smaller network architecture is chosen with 3 convolutional layers and 2 fully connected layers. At the start of the experiment, 100 human demonstrations are collected and annotated with sketches.

This experiment is repeated 3 times. The average success rate of the agent as a function of time is shown in Fig. 10. The agent reaches over 80% success rate within 8 hours. During this time, the human annotator provides 65 ± 10 additional reward sketches. This experiment demonstrates that it is possible to solve an industrial robot task from vision using human feedback within a single working day.

Two successful episodes of USB insertion are shown in Fig. 4 in two last rows. In the first example the robot successfully inserts a key using only pixel inputs. As only vision input is used during training and actions are defined with respect to the wrist frame, the resulting policy is robust to unseen positional changes. In the second example, the agent (which is trained on the unperturbed state) can perform insertion despite moving the input socket significantly.

IV. Related Work

RL has a long history in robotics [37, 53, 34, 25, 42, 43, 35]. However, applying RL in this domain inherits all the general difficulties of applying RL in the real world [18]. Most published works either rely on state estimation for a specific task, or work in a very limited regime to learn from raw observations. These methods typically entail highly engineered reward functions. In our work, we go beyond the usual scale of application of RL to robotics, learn from raw observations and without predefined rewards.

Batch RL trains policies from a fixed dataset and, thus, it is particularly useful in real-world applications like robotics. It is currently an active area of research (see the work of Lange et al. [41] for an overview), with a number of recent works aimed at improving the stability [22, 30, 1, 40].

In the RL-robotics literature, QT-Opt [35] is the closest approach to ours. The authors collect a dataset of over 580,000 grasps for several weeks with 7 robots. They train a distributed Q-learning agent that shows remarkable generalization to different objects. Yet, the whole system focuses on a single task: grasping. This task is well-suited for reward engineering and scripted data collection policies. However, these techniques are not easy to design for many tasks and, thus, relying on them limits the applicability of the method. In contrast, we collect the diverse data and we learn the reward functions.

Learning reward functions using inverse RL [52] achieved tremendous success [20, 27, 44, 21, 48, 73, 6]. This class of methods works best when applied to states or well-engineered features. Making it work for high-dimensional input spaces, particularly raw pixels, remains a great challenge.

Learning from preferences has a long history [66, 50, 19, 64, 14, 32]. Interactive learning and optimization with human preferences dates back to works at the interface of machine learning and graphics [10, 11]. Preference elicitation is also used for reward learning in RL [65, 71]. It can be done by whole episode comparisons [2, 3, 12, 59] or shorter clip comparisons [13, 29]. A core challenge is to engineer methods that acquire many preferences with as little user input as possible [38]. To deal with this challenge, our reward sketching interface allows perceptual reward learning [60] from any, even unsuccessful trajectories.

Many works in robotics choose to learn from demonstrations to avoid hard exploration problems of RL. For example, supervised learning to mimic demontrations is done in BC [55, 56]. However, BC requires high-quality consistent demonstrations of the target task and as such, it cannot benefit from heterogeneous data. Moreover, BC policies generally cannot outperform the human demonstrator. Demonstrations could be also used in RL [51, 57] to address the exploration problem. As in prior works [67, 54, 68], we use demonstrations as part of the agent experience and train with temporal difference learning in a model-free setting.

Several recent large-scale robotic datasets were released recently to advance the data-driven robotics. Roboturk [47] collects crowd-sourced demonstrations for three tasks with the mobile platform. The dataset is used in the experiments with online RL. MIME [61] dataset contains both human and robot demonstrations for 20 diverse tasks and its potential is tested in the experiments with BC and task recognition. RoboNet [15] database focuses on transferring the experience across objects and robotic platforms. The large-scale collection of the data is possible thanks to scripted policies. The strength of this dataset is evaluated in action-conditioned video prediction and in action prediction. Our dataset [16] is collected with demonstrations, scripted policies as well as learned policies. This paper is the first to show how to efficiently label such datasets with rewards and how to apply batch RL to such challenging domains.

V. Conclusions

We have proposed a new data-driven approach to robotics. Its key components include a method for reward learning, retrospective reward labelling and batch RL with distributional value functions. A significant amount of engineering and innovation was required to implement this at the present scale. To further advance data-driven robotics, reward learning and batch RL, we release the large datasets [16] from NeverEnding Storage and canonical agents [28].

We found that reward sketching is an effective way to elicit reward functions, since humans are good at judging progress toward a goal. In addition, the paper also showed that storing robot experience over a long period of time and across different tasks allows to efficiently learn policies in a completely off-line manner. Interestingly, diversity of training data seems to be an essential factor in the success of standard state-of-the-art RL algorithms, which were previously reported to fail when trained only on expert data or the history of a single agent [22]. Our results across a wide set of tasks illustrate the versatility of our data-driven approach. In particular, the learned agents showed a significant degree of generalization and robustness.

This approach has its limitations. For example, it involves a human-in-the-loop during training which implies additional cost. The reward sketching procedure is not universal and other strategies might be needed for different tasks. Besides, the learned agents remain sensitive to significant perturbations in the setup. These open questions are directions for future work.

Acknowledgments

We would like to thank all the colleagues at DeepMind who teleoperated the robot for data collection.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. *arXiv preprint arXiv:1907.04543*, 2019.

[2] Riad Akrour, Marc Schoenauer, and Michèle Sebag. APRIL: Active preference learning-based reinforcement learning. In *ECMLPKDD*, pages 116–131, 2012.

[3] Riad Akrour, Marc Schoenauer, Michele Sebag, and Jean-Christophe Souplet. Programming by feedback. In *International Conference on Machine Learning*, pages 1503–1511, 2014.

[4] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in English and Mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[6] Nir Baram, Oron Anschel, Itai Caspi, and Shie Mannor. End-to-end differentiable adversarial imitation learning. In *International Conference on Machine Learning*, pages 390–399, 2017.

[7] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations*, 2018.

[8] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458, 2017.

[9] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[10] Eric Brochu, Nando de Freitas, and Abhijeet Ghosh. Active preference learning with discrete choice data. In *Advances on Neural Information Processing Systems*, pages 409–416, 2007.

[11] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian interactive optimization approach to procedural animation design. In *SIGGRAPH Symposium on Computer Animation*, pages 103–112, 2010.

[12] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, pages 783–792, 2019.

[13] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances on Neural Information Processing Systems*, pages 4299–4307, 2017.

[14] Wei Chu and Zoubin Ghahramani. Preference learning with Gaussian processes. In *International Conference on Machine Learning*, pages 137–144, 2005.

[15] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. RoboNet: Large-scale multi-robot learning. In *Conference on Robot Learning*, 2019.

[16] DeepMind. Sketchy data, 2020. URL https://github.com/deepmind/deepmind-research/tree/master/sketchy.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186, 2019.

[18] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[19] Stephen E Feinberg and Knley Larntz. Log-linear representation for paired and multiple comparison models. *Biometrika*, 63(2): 245–254, 1976.

[20] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[21] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[22] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv e-prints*, art. arXiv:1812.02900, 2018.

[23] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *International Conference on Intelligent Robots and Systems*, pages 3948–3955, 2017.

[24] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772, 2014.

[25] Roland Hafner and Martin Riedmiller. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[27] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances on Neural Information Processing Systems*, pages 4565–4573, 2016.

[28] Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio GÃşmez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[29] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in Atari. In *Advances on Neural Information Processing Systems*, pages 8011–8023, 2018.

[30] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[31] Rae Jeong, Yusuf Aytar, David Khosid, Yuxiang Zhou, Jackie Kay, Thomas Lampe, Konstantinos Bousmalis, and Francesco Nori. Self-supervised sim-to-real adaptation for visual robotic manipulation. *arXiv preprint arXiv:1910.09470*, 2019.

[32] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems*, 25(2), 2007.

[33] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Computer Vision and Pattern Recognition*, 2017.

[34] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. Learning force control policies for compliant manipulation. In *International Conference on Intelligent Robots and Systems*, pages 4639–4644, 2011.

[35] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673,

2018.

[36] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2018.

[37] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[38] Yuki Koyama, Issei Sato, Daisuke Sakamoto, and Takeo Igarashi. Sequential line search for efficient visual design optimization by crowds. *ACM Transactions on Graphics*, 36(4):1–11, 2017.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances on Neural Information Processing Systems*, pages 1097–1105, 2012.

[40] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy Q-learning via bootstrapping error reduction. In *Advances on Neural Information Processing Systems*, pages 11761–11771, 2019.

[41] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[42] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[43] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[44] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances on Neural Information Processing Systems*, pages 3812–3822, 2017.

[45] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755, 2014.

[46] Andrew Maas, Ziang Xie, Dan Jurafsky, and Andrew Ng. Lexicon-free conversational speech recognition with neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 345–354, 2015.

[47] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. RoboTurk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893, 2018.

[48] Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[50] F Mosteller. Remarks on the method of paired comparisons: I. the least squares solution assuming equal standard deviations and equal correlations. *Psychometrika*, 16:3–9, 1951.

[51] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics & Automation*, pages 6292–6299, 2018.

[52] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, pages 663–670, 2010.

[53] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[54] Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerík, et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

[55] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances on Neural Information Processing Systems*, pages 305–313, 1989.

[56] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *IEEE International Conference on Robotics & Automation*, pages 3758–3765, 2018.

[57] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *Robotics, Science and Systems*, 2018.

[58] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[59] Dorsa Sadigh, Anca D. Dragan, Shankar Sastry, and Sanjit A. Seshia. Active preference-based learning of reward functions. In *Robotics, Science and Systems*, 2017.

[60] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. *Robotics, Science and Systems*, 2017.

[61] Pratyusha Sharma, Lekha Mohan, Lerrel Pinto, and Abhinav Gupta. Multiple interactions made easy (MIME): Large scale demonstrations data for imitation. In *Conference on Robot Learning*, pages 906–915, 2018.

[62] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

[63] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[64] Hal Stern. A continuum of paired comparison models. *Biometrika*, 77:265–273, 1990.

[65] Malcolm J. A. Strens and Andrew W. Moore. Policy search using paired comparisons. *Journal of Machine Learning Research*, 3: 921–950, 2003.

[66] LL Thurstone. A law of comparative judgement. *Psychological Review*, 34:273–286, 1927.

[67] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[68] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *IEEE International Conference on Robotics & Automation*, pages 754–760, 2019.

[69] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al.

Alphastar: Mastering the real-time strategy game StarCraft II. *DeepMind Blog*, 2019.

[70] Paul J Werbos et al. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[71] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

[72] Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Regularized hierarchical policies for compositional transfer in robotics. *arXiv preprint arXiv:1906.11228*, 2019.

[73] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *Robotics, Science and Systems*, 2018.