A Framework for Data-Driven Robotics

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

Abstract—We present a framework for data-driven robotics that makes use of a large dataset of recorded robot experience and scales to several tasks using learned reward functions. We show how to apply this framework to accomplish three different object manipulation tasks on a real robot platform. Given demonstrations of a task together with task-agnostic recorded experience, we use a special form of human annotation as supervision to learn a reward function, which enables us to deal with real-world tasks where the reward signal cannot be acquired directly. Learned rewards are used in combination with a large dataset of experience from different tasks to learn a robot policy offline using batch RL. We show that using our approach it is possible to train agents to perform a variety of challenging manipulation tasks including stacking rigid objects and handling cloth.

I. INTRODUCTION

Deep Learning (DL) has successfully advanced many fields of Artificial Intelligence (AI), including computer vision [1], [2], speech recognition [3]–[5], natural language processing [6], and Reinforcement Learning (RL) [7], [8]. The success of DL 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 [9], COCO [10] and CLEVR [11]. 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 [8], [12], [13]. The OpenAI Five DOTA bot [13] 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 [7]. This stands in contrast to robotics, where we lack abundant data since designing a policy typically requires execution on a real robot, which cannot be accelerated beyond real time. This limits the effectiveness of Deep RL in robotics.

This paper presents a framework to bring the power of Deep RL to robotics through an approach that makes the generation and handling of a large dataset tractable. While previous attempts required several robots operating in parallel [14], [15], our framework requires only a single robot.

To generate enough data to feed the data hungry machinery of Deep RL with a single robot we record experience continuously and persistently, regardless of the purpose or quality of the behavior. This means that for a new task our data is necessarily off policy, and is also typically off task as

This work is done at Deepmind. Videos are available on this website: https://sites.google.com/view/data-driven-robotics/

Fig. 1: Left: illustration of the proposed framework (see Fig. 2 for more details). Top right: reward annotation. Bottom right: stacking objects and lifting cloth manipulation tasks. well. Nevertheless, we show that the accumulated experience can be reused to learn new tasks. Reuse is made possible by automatically annotating historical data using task-specific learned reward model.

Our framework introduces three novel components:

1) The NeverEnding Storage (NES) system, which captures and stores all experience generated by the robot, regardless of purpose.

2) The reward sketching procedure for eliciting reward functions from humans, so that past experience can be automatically annotated for new tasks.

3) The use of off-policy batch RL allows us to train as many policies as possible given the computation budget, using data that is generated using different behavior policies without further execution on the robot.

The NES system captures all the camera and sensor data generated by the robot. This includes demonstrations (by human teleoperators) of various tasks, behaviors generated by trained policies for these tasks (successful or not), as well as experience generated by scripted or random policies. If the robot is moving, then the experience is being captured by NES.

Our approach to task specification relies on human judgments about progress toward the goal to define task-specific reward functions. Annotations are elicited from humans in the form of per-timestep reward annotations using a process we call reward sketching (Fig. 1, top right). 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 reward model, which is then used to annotate the remainder of the database automatically. The use of a learned reward function in this way allows us to repurpose an arbitrary amount of past experience using a fixed amount of annotation effort per task.
TABLE I: Comparison between several frameworks in robotics, showing different challenges that arise in each of them. We address these limitations with our newly proposed framework. (*Sim2real does not require reward from the real world, but does require reward in the simulated environment.)

|                  | Classic | Pure RL | Sim2real | Ours |
|------------------|---------|---------|----------|------|
| State Estimation | Yes     | Yes     | No       | No   |
| Environment Reward | No      | Yes     | Yes*     | No   |
| System Identification | Yes   | No      | Yes      | No   |
| World Model      | Yes     | No      | Yes      | No   |
| Domain Gap       | No      | No      | Yes      | No   |

Table I compares our framework to other common approaches in robotics. Through learning of both reward functions and controllers we avoid the need for explicit state estimation, which is an essential component of classic robotics. RL systems also typically rely on state estimation to deliver rewards, even when they learn policies from pixels. System identification and environment modelling arise in any situation where explicit models (either of the robot or of the task) are required and significant effort is needed to build these models in both classical robotics and sim2real. Similarly, unlike the sim2real setting we have no perceptual domain gap to address, since we always train on experience collected directly from the robot.

II. RELATED WORK

RL: RL has a long history in robotics [14]–[20] but applying RL to robots inherits all the general difficulties of applying RL to the real world [21]. Most published works either rely on state estimation for a specific task, or work in a very data limited regime to learn from raw observations. These methods typically entail highly engineered reward functions. Our work handles these limitations. Besides, we go beyond the usual scale of RL methods in robotics.

Among the RL for robotics literature, QT-Opt [15] is the closest approach to ours. The authors collected a dataset of over 580,000 grasps over the course of several weeks with 7 robots. The resulting distributed Q-learning agent shows remarkable generalization to different objects. Yet, the whole system focuses on a single task: grasping. Grasping is particularly well-suited for hand-engineering a reward function and scripting policies for collecting data. However, neither hand-engineered reward functions nor scripting policies are easy to design for many tasks. So, relying on these techniques limits the applicability of the method. In contrast, in our work we learn the reward functions. Besides, experiences for multiple tasks contribute to learning a target task.

Demonstrations: Demonstrations in RL have gained popularity in recent years [22], [23] as they help to address the exploration problem. As in prior works [24]–[26], we use demonstrations as part of the agent experience and train with temporal difference learning in a model-free setting.

Demonstrations can also be incorporated through Behavioral Cloning (BC) [27], [28], but this requires high-quality consistent demonstrations of the target task. As such, it cannot benefit from heterogeneous data. Moreover, BC policies cannot outperform the human demonstrator.

Sim2real: The sim2real framework aims to seamlessly transfer policies trained in simulated environments to the real world [29]–[33]. These methods have achieved remarkable success in dexterous in-hand manipulation [32] and deformable object manipulation [34]. The challenges of sim2real lie in the need for extensive system identification to construct simulated environments that closely mirror the target task. Even with careful simulation design there remains a domain gap between simulation and reality that needs to be dealt with [30], [34]. In our framework there is no need to simulate the robot or the environment, and, thus, no domain gap.

Reward learning: Learning reward functions using inverse RL dates back to Ng and Russell [35] and has achieved tremendous success recently [36]–[42]. 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 challenge.

Learning from preferences [43]–[48] also has a long history. For example, interactive learning and optimization with human preferences has been applied to animation [49]–[51]. Preference learning is also used in RL for reward learning [52], [53]. Preferences can be extracted with whole episode comparisons [54]–[57] or shorter clip comparisons [58], [59]. Binary success labels of frames are also used to learn reward functions [26], [60].

In our framework we propose a new form of annotation, reward sketching (Section IV-C), that allows perceptual reward learning [61] from any trajectory, not only successful demonstrations.

Batch RL: Batch RL describes the scenario where the learning experience is fixed a priori [62]. We rely on Batch RL for training our agents. This is an active area of research, with a number of recent works aimed at improving its stability [63]–[66]. These additional advancements can further improve our agents.

III. FRAMEWORK OVERVIEW AND MOTIVATION

The general workflow is illustrated in Fig. 2. The distinct properties of our pipeline come from the interaction of the two key features: NES and learning the reward model from sketches. NES allows us to accumulate 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 using our framework has the following steps which are described in turn in the remainder of the section, with more details given in Section IV:

1) A human teleoperates the robot to provide first-person demonstrations of the target task (IV-A).
2) All robot experience, including these demonstrations, is accumulated into NES (IV-B).
3) A subset of data from NES (including the task-specific demos) are annotated by humans with reward sketches for the target task (IV-C).
Learn Q/pi
Evaluate
Execute
Learn R
As illustrated in Fig. 3, user draws a curve indicating the
we use expert knowledge about the intended outcome of the
we circumvent the problem of exploration in RL: Instead of
This allows us to make use of historical data when learning
various tasks, human play data, and accumulated experience
include both the demonstrations provided for the target task,
task to guide the agent.

Reward Sketching: The second step in task specification
is reward sketching. We ask human experts to provide per-
timestep annotations of reward using a custom user interface.
As illustrated in Fig. 3, 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.

The reward sketches allow comparison of perceived value
of any two frames. In addition, the green region in Fig. 3 is
reserved for timesteps where the goal is achieved. For each
task the episodes to be annotated are drawn from NES. They
include both the demonstrations provided for the target task,
as well as experience generated for prior tasks. Annotating
data from prior tasks ensures better coverage of the state
space.

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

Off-policy batch RL: We train an agent using off-policy
pure batch RL. It allows us to learn as many policies as
possible given the computation budget, using data that is
generated using different behavior policies without further
execution on the robot. Besides, the agent is trained by only
using RL objective and it does not use feature pretraining, BC
initialization, any special batch correction terms, or auxiliary
losses. We do, however, find it important to use the historical
data from other tasks.

Execution: Once an agent is trained, we can run it on
the real robot. By running the agent on the robot, we collect
more experience data 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.

At this point we can iterate the workflow to improve the
agent. Typically this involves sketching more data to improve
the reward function. We can choose to sketch the new data
generated by executing the policy on the robot, or more of
the historical data stored in NES. If particular failure cases
are observed during execution, we can teleoperate the robot
to collect examples near these failure modes. Then, we sketch
them to direct an agent in areas of the state space where it
does yet not perform well.

IV. METHODS

A. Teleoperation

Our approach starts by human operators providing first-
person demonstrations of the target tasks on the robot. The
robot is controlled with a 6-DoF mouse or hand-held virtual
reality controllers. 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 as well as proprioceptive inputs.

In addition to full episodes of demonstrations, 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. All demonstrations are stored in NES with corresponding metadata. Note that in later workflow iterations, all demonstrations are used for RL and possibly reward sketching.

B. NeverEnding Storage

NeverEnding Storage (NES) is a database of episode data stored as files on disk and metadata associated with it. Episode data includes video recordings from several cameras in the cage and at the robot wrist (Fig. 4). Metadata includes unique episode ID, operator name, date and time of operation, task ID, episode type, a path to separately stored experience data, and others. Each episode can be marked with an arbitrary set of tags to offer a flexible way to identify sets of relevant episodes. NES also allows to associate an arbitrary number of reward sketches with each episode that could be task-specific. Metadata and reward sketches can be jointly queried with SQL queries to select subsets of data.

C. Reward Sketching

In order to convey the desired behaviour to the agent, a human operator sketches the reward values for several examples of successful and unsuccessful trajectories for each task. For annotation, we select demonstration episodes as examples. This combination gives good coverage of the state space.

Sketching works on an episodic basis. An example of a sketch is shown in Fig. 3. 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 Section IV-B.

D. Reward Learning

Episodes annotated with reward sketches are used to train a reward function in the form of a neural network with parameters $\psi$ 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 to the estimated reward function $r_\psi(x_t)$. Second, if $s(x_t)$ is higher than $s(x_q)$ by a threshold $\mu_s$, then $r_\psi(x_t)$ should be higher than $r_\psi(x_q)$ by another threshold $\mu_r$. These conditions are captured by the following two hinge losses:

$$L_{\text{rank}}(\psi) = \max \{0, r_\psi(x_t) - r_\psi(x_q) + \mu_r\} \mathbb{I}_{s(x_t) - s(x_q) > \mu_s}$$
$$L_{\text{success}}(\psi) = \max \{0, \tau_1 - r_\psi(x)\} \mathbb{I}_{s(x) > \tau_s} + \max \{0, r_\psi(x) - \tau_2\} \mathbb{I}_{s(x) < \tau_s}$$

(1)

The total loss is obtained by adding these terms: $L_{\text{rank}} + \lambda L_{\text{success}}$. In our experiments, we set $\mu_s = 0.2$, $\mu_r = 0.1$, $\tau_s = 0.85$, $\tau_1 = 0.9$, $\tau_2 = 0.7$, and $\lambda = 10$.

E. Off-policy batch RL

We use an algorithm similar to that of D4PG [67] as our main training algorithm, but with a recurrent state. It maintains an online value network $Q(x_t, h_\pi^t, a_t | \theta)$ and an online policy network $\pi(x_t, h_\pi^t | \phi)$. Given the effectiveness of recurrent values functions [68], both $Q$ and $\pi$ are recurrent with $h_\pi^0$ and $h_\pi^t$ representing the corresponding recurrent hidden states. The target networks are of the same structures as the value and policy network, but are parameterized by different parameters $\theta'$ and $\phi'$ which are periodically updated to the current parameters of the online networks.

Given the $Q$ function, we update the policy using DPG [69]:

$$\nabla_\phi Q(x_t, h_\pi^t, \pi(x_t, h_\pi^t | \phi) | \theta)$$

(2)

As in D4PG, instead of using a scalar $Q$ function, we adopt a distributional value function [70]. We refer the readers to the original paper for details on the loss for learning distributional value functions which we use to compute the gradient for learning the critic:

$$\nabla_\theta L_{\text{dist}}(Q(x_{t+1}, h_{t+1}^Q, \hat{a}_{t+1} | \theta'), r_\psi(x_t), \gamma, Q(x_t, h_\pi^t, a_t | \theta)))$$

(3)

with $\hat{a}_{t+1} = \pi(x_{t+1}, h_\pi^t | \phi')$

During learning, we sample a batch of sequences of observations and actions $\{x_t^i, a_t^i, \cdots, x_{t+1}^i\}$, and use a zero start state to initialize all recurrent states at the beginning of sampled sequences. We then update $\phi$ and $\theta$ following gradients defined in (2) and (3) respectively using BPTT [71].

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 [25], where fixed ratio batches are formed with agent and demonstration data.
F. Execution

After policies are trained, we choose some of them for execution on the robot. No matter how successful the episodes are, the data is accumulated in NES to train next models.

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.

V. EXPERIMENTS

A. Experimental Setup

Robotic setup: Our experimental 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 seven degrees of freedom, corresponding to Cartesian translational and rotational velocity targets of the gripper pinch point and gripper fingers. The agent control loop is executed at 10 Hz. For safety, the pinch point movement is restricted to be in a 35 × 35 × 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. 4). NES captures all of the observations, and 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.

Tasks and datasets: We focus on two subsets of NES, containing data recorded during manipulations of:

1) three colored objects, one of each color: red, green and blue (rgb dataset, Fig. 3);
2) three deformable objects: a soft ball, a rope and a piece of cloth (deformable dataset, Fig. 1, bottom right).

The rgb dataset is used to learn policies for two tasks: lift_green and stack_green_on_red, the deformable dataset is used for the pull_cloth_up task. Final statistics for both datasets are presented in Table II. Both datasets were grown progressively by iterating the process in Fig. 2. Each episode lasts for 200 steps (20 seconds) unless an episode is terminated earlier for safety reasons.

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 moments in time. 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 final datasets degrades performance of the final agents.

The two datasets contain a significant number of teleoperated episodes, although the majority are recorded via interactive teleoperation (section IV-A), and thus required 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 pull_cloth_up tasks, respectively. Approximately 90% of the episodes are used for training and 10% for validation. Notice that not all sketches were obtained at once, but they were accumulated over several iterations of the process in Fig. 2.

Reward network architecture: The reward network is a non-recurrent residual network with a spatial softmax layer (Fig. 5). The spatial softmax layer [20] produces a list of 64 (x, y) coordinates. Proprioceptive features are embedded using linear layers, layer-normalized [72], and concatenated with the camera encoding vectors. As the sketched values are in the range of [0, 1], the reward network ends with a sigmoid non-linearity.

Agent architecture: The final agent for lift_green and stack_green_on_red tasks observes two cameras, a basket front left camera (80 × 128) and one of wrist-
Fig. 5: Agent network architecture. Note that the wrist camera encoder is shown, but in practice we encode each camera independently and concatenate the results.

| Agent                  | Normal | Hard | Unseen |
|------------------------|--------|------|--------|
| Our approach           | 80%    | 80%  | 50%    |
| No random watcher data | 80%    | 70%  | 20%    |
| No other task data     | 0%     | 0%   | 0%     |
| Non-distributional     | 30%    | 20%  | 10%    |

**TABLE III: Performance on the lift_green task.**

| Agent                  | Normal | Hard | Unseen |
|------------------------|--------|------|--------|
| Our approach           | 60%    | 40%  | 40%    |
| No random watcher data | 50%    | 30%  | 30%    |
| No other task data     | 0%     | 10%  | 0%     |
| Non-distributional     | 20%    | 0%   | 0%     |

**TABLE IV: Performance on the stack_green_on_red task.**

mounted wide angle cameras (96×128) (Fig. 4). The agent for pull_cloth_up also uses additional basket back left camera (80×128).

The agent network is illustrated in Fig. 5. Each camera is encoded using a residual network followed by a spatial softmax keypoint encoder with 64 channels. We use one such column for each camera and concatenate the results. Before applying the spatial softmax, noise from the distribution $U[-0.1,0.1]$ was added to the logits.

Proprioceptive features are concatenated, embedded using a linear layer, layer-normalized [72] and passed through a tanh activation and appended to the camera encodings to form joint the input features.

The actor network consumes these joint input features directly. The critic network additionally passes them through a linear layer, concatenates the the result with action features (obtained by passing the action through a linear layer), and passes the result through an additional linear layer with ReLU activation. Everything is then fed through a two layer layer-normed LSTMs with 256 hidden units each, followed by actor or critic heads. Action outputs are processed through a tanh layer putting them in the range $[-1,1]$, and then rescaled to their native ranges before being sent to the robot.

**Training:** We train multiple RL agents in parallel and briefly evaluate 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 to get the final agents that we report here.

**Evaluation:** Finally, we run 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:

- **normal** – basic rectangular green blocks well represented in training data, large red object close to center in starting position;
- **hard** – more diverse objects that are less well represented in training data, smaller red objects with diverse starting locations;
- **unseen** – green objects unseen during training with a large red object.

Each condition specifies 10 different initial positions of the objects (set by 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.

Figure 6 shows examples of the initial conditions. We use the same 3 evaluation sets for both the lift_green and stack_green_on_red tasks. To evaluate the pull_cloth_up task, we randomize initial conditions at every trial. As a quality metric, we measure the rate of successfully completed episodes, where success is judged by the human operator.

**B. Results**

Results on the rgb dataset are summarized in Table III and IV. 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. The success rate of our agent for pull_cloth_up task in 50 episodes with randomized initial conditions is 74%.

Our results compares favorably with those of Zhu et al. [41], where block lifting and stacking success rates are 64% and 35%. Wulfmeier et al. [73] also attempted the block stacking task. But instead of learning directly from pixels, they rely on QR code based state estimation of fixed set of cubes, whereas our policies can handle a variety of objects.

To understand the importance of accumulating robot experience in NES, we run the following ablations. First,
we train an agent using only task-specific data that resembles the conventional RL approach. This strategy is interesting because although the amount of training data is reduced, the similarity between training data and target behavior is increased (i.e. the training data is more on-policy). Second, we train an agent while excluding only the random watcher data. As this data is unlikely to contain episodes relevant to the task, we are interested in knowing how much it contributes to the final performance.

Tables III and IV show the results of these ablations. Remarkably, using only a task-specific dataset dramatically degrades the performance of the policy. 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 [65], [67], we found distributional value functions to be very helpful.

For qualitative results we refer the reader to the accompanying video that demonstrates the robustness of our agents. They successfully deal with adversarial perturbations by a human operator, stacking several unseen and non-standard objects and lifting toys, such as a robot and a pony. Last but not least, our agents move faster and are more efficient compared to a human operator.

VI. CONCLUSIONS

We proposed a framework for data-driven robotics that makes use of a large dataset of diverse robot experience and reward functions learned from a novel form of human feedback. We presented a successful instantiation of this framework to train policies using pure batch RL. Our experimental results allow us to draw the following conclusions:

1) Reward sketching is an effective way to elicit reward functions, since humans are good at judging progress toward a task goal. Importantly, this approach can be directly applied to many other tasks.

2) Stored robot experience over a long period of time and across different tasks can be efficiently harnessed to learn policies in a completely offline manner.

3) 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 [63].

4) We can tightly close the loop of human input, reward learning and policy learning with a human-in-the-loop workflow. It enables us to learn a new real world skill in a flexible and fast way.

5) Qualitatively we observed that agents trained using our framework accomplish tasks faster than human teleoperation or the BC policies we trained.

Our proposed framework offers a promise to address several shortcomings of other approaches that rely on DL in robotics. Importantly, it is general enough to obtain policies for a variety of robot manipulation tasks as we demonstrated in our experiments. Our policies achieve notable success even in hard settings involving complex object manipulation, soft deformable objects, and previously unseen sets of objects without any need for feature or reward design.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances on Neural Information Processing Systems, 2012, pp. 1097–1105.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in IEEE Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[3] A. Graves and N. Jaitly, “Towards end-to-end speech recognition with recurrent neural networks,” in International Conference on Machine Learning, 2014, pp. 1764–1772.

[4] H. Mas, Z. Xie, D. Jurafsky, and A. 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, 2015, pp. 345–354.

[5] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, et al., “Deep speech 2: End-to-end speech recognition in English and Mandarin,” in International Conference on Machine Learning, 2016, pp. 173–182.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” arXiv preprint arXiv:1810.04805, 2018.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., “Mastering the game of go with deep neural networks and tree search,” nature, vol. 529, no. 7587, p. 484, 2016.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, 2015.

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in European conference on computer vision. Springer, 2014, pp. 740–755.

[11] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick, “CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning,” in IEEE Computer Vision and Pattern Recognition, 2017.

[12] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, et al., “Alphastar: Mastering the real-time strategy game starcraft ii,” DeepMind Blog, 2019.

[13] “OpenAI Five,” https://openai.com/five/.

[14] 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.

[15] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, “Scalable deep reinforcement learning for vision-based robotic manipulation,” in Conference on Robot Learning, 2018, pp. 651–673.

[16] J. Koerber, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” The International Journal of Robotics Research, vol. 32, no. 11, pp. 1238–1274, 2013.

[17] J. Peters and S. Schaal, “Reinforcement learning of motor skills with policy gradients,” Neural networks, vol. 21, no. 4, pp. 682–697, 2008.
The action space is defined as the task is to insert it to a specific computer port (Fig. 7).

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 $8.6^\circ$ rotation in each direction, and the allowed workspace is $8 \times 8 \times 15$ cm with $17.2^\circ$ rotation. This amount of variation is quite significant, as position controlled robots can set the initial state with a much smaller tolerance. Episodes are terminated with a discount of zero if the robot reaches 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. The success is determined by information from tracking the absolute position (note that neither the policy nor the reward function have access to this information).

The average success rate of the agent is shown in Fig. 7 for three replications of the experiment. The agent reached over 80% success rate within 8 hours. During this time, the human annotator provided 65 ± 10 additional reward sketches. This experiment demonstrates that it is possible to learn to perform a complex robotics task from vision using human feedback within a single working day.

ACKNOWLEDGEMENTS

We would like to thank Jan Leike, Borja Ibarz and Konstantinos Bousmalis for their insightful comments along the project and all the DeepMind colleagues who kindly teleoperated the robot for data collection. We would also like to thank the open source community for developing the core set of tools that enabled this work, including ROS [74], Tensorflow [75], Numpy [76], Pandas [77], Matplotlib [78], and Seaborn [79].