Supplementary Material for Submission: Learning by Playing – Solving Sparse Reward Tasks from Scratch

Martin Riedmiller [*1] **Roland Hafner** [*1] **Thomas Lampe** [1] **Michael Neunert** [1] **Jonas Degrave** [1]
Tom Van de Wiele [1] **Volodymyr Mnih** [1] **Nicolas Heess** [1] **Tobias Springenberg** [1]

Abstract

We provide details on the experimental setup as well as additional experiments, highlighting different aspects of the SAC agent. A supplementary video for the experiments can be found at https://youtu.be/mPKyvocNe_M

A. Details on the Experimental Setup

A.1. Simulation

For the simulation of the Jaco robot arm the numerical simulator MuJoCo [1] was used – using a model we identified from our real robot setup.

The simulation was run with a numerical time step of 10 milliseconds, integrating 5 steps, to get a control interval of 50 milliseconds for the agent. In this way we can resolve all important properties of the robot arm and the object interactions in simulation.

The objects that are used are based on wooden toy blocks. We use a cubic block with side lengths of 5 cm (red object) and a cuboid with side lengths of 5cm x 5cm x 8cm (green block). For the banana stacking experiment a combination of 3 different geometric (capsule shaped) primitives with radius 2.5 cm are used, resulting in a banana shaped object of 12 cm in length (replacing the red object).

All experiments made use of an experiment table with sides of 60 cm x 30 cm in length, which is assumed to be the full working space for all experiments. The objects are spawned at random on the table surface. The robot hand is initialized randomly above the table-top with a height offset of up to 20 cm above the table (minimum 10 cm) and the fingers in an open configuration. The simulated Jaco is controlled by raw joint velocity commands (up to 0.8 radians per second) in all 9 joints (6 arm joints and 3 finger joints). All experiments run on episodes with 360 steps length (which gives a total simulated real time of 18 seconds per episode). For the SAC-X experiments we schedule 2 intentions each episode, holding the executed intention fixed for 180 steps.

For the feature based experiments in simulation we make use of the proprioceptive features that the Jaco robot can deliver (see Table 1). In addition, for the default simulation experiments, we use features from the objects in the scene, that are computed directly in simulation (see table 2). This gives a total of 56 observation entries. For the cleanup experiment, we add the lid angle and lid angle velocity, which gives a total of 58 observations for this experiment. For the pixel experiments, we use two RGB cameras with an resolution of 48 x 48 (see table 3) in combination with the proprioceptive features (table 1).

Table 1. Proprioceptive observations used in all simulation experiments.

Entry	dimensions	unit
arm joint pos	6	rad
arm joint vel	6	rad / s
finger joint pos	3	rad
finger joint vel	3	rad / s
finger touch	3	N
TCP pos	3	m

Table 2. Object feature observations, used in the default simulation experiments. For the pixel experiments these observations are not used. The pose of the objects is represented as world coordinate position and quaternions. In the table m denotes meters, q refers to a quaternion which is in arbitrary units (au).

Entry	dimensions	unit
object i pose	7	m au
object i velocity	6	m/s, dq/dt
object i relative pos	3	m

[*] Equal contribution [1] Google DeepMind, London, GB. Correspondence to: Martin Riedmiller <riedmiller@google.com>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

[1] MuJoCo: see www.mujoco.org

Table 3. Pixel observations that replace the object observations of table 2 for the pixel experiments.

Entry	dimensions	unit
camera 1	48 x 48 x 3	rgb
camera 2	48 x 48 x 3	rgb

A.1.1. AUXILIARY REWARD OVERVIEW

We use a basic set of general auxiliary tasks for our experiments. Dependent on the type and number of objects in the scene the number of available auxiliary tasks can vary.

- *TOUCH, NOTOUCH*: Maximizing or minimizing the sum of touch sensor readings on the three fingers of the Jaco hand. (see Eq. 12 and Eq. 13)
- *MOVE(i)*: Maximizing the translation velocity sensor reading of an object. (see Eq. 11)
- *CLOSE(i,j)*: distance between two objects is smaller than 10cm (see Eq. 1)
- *ABOVE(i,j)*: all points of object i are above all points of object j in an axis normal to the table plane (see Eq. 2)
- *BELOW(i,j)*: all points of object i are below all points of object j in an axis normal to the table plane (see Eq. 6)
- *LEFT(i,j)*: all points of object i are bigger than all points of object j in an axis parallel to the x axes of the table plane (see Eq. 4)
- *RIGHT(i,j)*: all points of object i are smaller than all points of object j in an axis parallel to the x axes of the table plane (see Eq. 7)
- *ABOVECLOSE(i,j)*, *BELOWCLOSE(i,j)*, *LEFTCLOSE(i,j)*, *RIGHTCLOSE(i,j)*: combination of relational reward structures and *CLOSE(i,j)* (see Eq. 3, 8, 5, 9)
- *ABOVECLOSEBOX(i)*: *ABOVECLOSE(i,box object)*

We define the auxiliary reward structures, so that we can - in principle - compute all the required information from one or two image planes (two cameras looking at the workspace). Replacing the world coordinates referenced above with pixel coordinates.

In the following equations a definition of all rewards is given. Let $d(o_i, o_j)$ be the distance between the center of mass of the two objects, $\max_a(o_i)$ and $\min_a(o_i)$ denote the maximal (or minimal) pixel locations covered by object i in axis $a \in \{x, y, z\}$.

$$r_{C(i,j)}(\mathbf{s},\mathbf{a}) = \begin{cases} 1 & \text{iff } d(o_i, o_j) \le 10cm \\ 0 & \text{else,} \end{cases} \tag{1}$$

$$r_{A(i,j)}(\mathbf{s},\mathbf{a}) = \begin{cases} 1 & \text{iff } \max_z(o_j) - \min_z(o_i) \le 0 \\ 0 & \text{else,} \end{cases} \tag{2}$$

$$r_{AC(i,j)}(\mathbf{s},\mathbf{a}) = r_{A(i,j)}(\mathbf{s},\mathbf{a}) * r_{C(i,j)}(\mathbf{s},\mathbf{a}) \tag{3}$$

$$r_{L(i,j)}(\mathbf{s},\mathbf{a}) = \begin{cases} 1 & \text{iff } \max_x(o_j) - \min_x(o_i) \le 0 \\ 0 & \text{else,} \end{cases} \tag{4}$$

$$r_{LC(i,j)}(\mathbf{s},\mathbf{a}) = r_{L(i,j)}(\mathbf{s},\mathbf{a}) * r_{C(i,j)}(\mathbf{s},\mathbf{a}) \tag{5}$$

$$r_{B(i,j)}(\mathbf{s},\mathbf{a}) = r_{A(j,i)}(\mathbf{s},\mathbf{a}) \tag{6}$$
$$r_{R(i,j)}(\mathbf{s},\mathbf{a}) = r_{L(j,i)}(\mathbf{s},\mathbf{a}) \tag{7}$$
$$r_{BC(i,j)}(\mathbf{s},\mathbf{a}) = r_{AC(j,i)}(\mathbf{s},\mathbf{a}) \tag{8}$$
$$r_{RC(i,j)}(\mathbf{s},\mathbf{a}) = r_{LC(j,i)}(\mathbf{s},\mathbf{a}) \tag{9}$$
$$\tag{10}$$

In addition to these 'object centric' rewards, we define MOVE, TOUCH and NOTOUCH as:

$$r_{MOVE(i)}(\mathbf{s},\mathbf{a}) = \begin{cases} |v(o_i)| & \text{iff } |v(o_i)| \ge 3\frac{mm}{s} \\ 0 & \text{else,} \end{cases} \tag{11}$$

$$r_{T(i)}(\mathbf{s},\mathbf{a}) = \begin{cases} |\sum_{i \in 1,2,3} f_i| & \text{iff } |\sum_{i \in 1,2,3} f_i| \le 1N \\ 1 & \text{else,} \end{cases} \tag{12}$$

$$r_{NT(i,j)}(\mathbf{s},\mathbf{a}) = \begin{cases} 1 & \text{iff } r_{T(i)}(\mathbf{s},\mathbf{a}) \le 0.1 \\ 0 & \text{else,} \end{cases} \tag{13}$$

Two objects were used in the experiments, yielding a set of 13 general auxiliary rewards that are used in all simulation experiments.

A.1.2. External Task Rewards

For the extrinsic or task rewards we use the notion of STACK(i), for a sparse reward signal that describes the property of an object to be stacked. As a proxy in simulation we use the collision points of different objects in the scene to determine this reward. where $col(o_i, o_j) = 1$ if object i and j in simulation do have a collision – 0 otherwise. We can derive a simple sparse reward from these signals as

$$r_{STACK(i)}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{iff } (1 - col(GROUND, o_i)) \\ & *(1 - col(ROBOT, o_i)) \\ & *col(o_j, o_i) = 0 \\ 0 & \text{else.} \end{cases} \tag{14}$$

For the cleanup experiments we use an additional auxiliary reward for each object, ABOVE_CLOSE_BOX (ACB), that accounts for the relation between the object and the box:

$$r_{ACB(i)}(\mathbf{s}, \mathbf{a}) = r_{AC(i,BOX)}(\mathbf{s}, \mathbf{a}). \tag{15}$$

As additional extrinsic reward, we use a sparse INBOX(i) reward signal, that gives a reward of one if the object i is in the box; INBOXALL, that gives a signal of 1 only if all objects are in the box; and a OPENBOX, which yields a sparse reward signal when the lid of the box is lifted higher then a certain threshold,

$$r_{INBOX(i)}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{iff } o_i \text{ is in box} \\ 0 & \text{else,} \end{cases} \tag{16}$$

$$r_{INBOXALL}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{iff all objects in box} \\ 0 & \text{else,} \end{cases} \tag{17}$$

$$r_{OPENBOX}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{iff } \theta_{lid} \geq 1.5 \\ 0 & \text{else,} \end{cases} \tag{18}$$

This gives 15 auxiliary reward signals and 4 extrinsic reward signals for the cleanup experiment.

A.2. Real Robot

On the real robot we use a slightly altered set of auxiliary rewards to account for the fact that the robot does not possess touch sensors (so TOUCH and NOTOUCH cannot be used). Additionally, a distance based reward for reaching is added to reduce the amount of training time needed. For the pick up experiment we used the following rewards: OPENED, CLOSED, LIFTED(block) and AT(hand,block), defined as:

- *OPENED, CLOSED*: maximal if the angle of the finger motors, $\theta_{fingers} \in [0.0,\ 0.8]$, is close to its minimum respectively maximum value. (see Eq. 19 and 20)
- *LIFTED(i)*: maximal if the lowest point of object i is at a height of 7.5cm above the table, with a linear shaping term below this height. (see Eq. 21)
- *AT(i, j)*: similar to CLOSE(i,j) in simulation but requiring objects to be closer; maximal if the centers of i and j are within 2cm of each other; additionally uses a non-linear shaping term when further apart. (equivalent to CLOSE$_{1cm}$(i,j) in Eq. 22)

The rewards are defined as followed:

$$r_{OPENED}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{iff } \theta_{fingers} \leq 0.1 \\ 0 & \text{else,} \end{cases} \tag{19}$$

$$r_{CLOSED}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{iff } \theta_{fingers} \geq 0.7 \\ 0 & \text{else,} \end{cases} \tag{20}$$

$$r_{LIFTED(i)}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1.5 & \text{iff } min_z(i) > 7.5cm \\ 0. & \text{iff } min_z(i) < \ 0.5cm \\ \frac{min_z(i)}{7.5} & \text{else,} \end{cases} \tag{21}$$

For all other rewards based on the relation between two entities i and j, we use a shaped variant of CLOSE that is parametrized by a desired distance ϵ. Let $d(i, j)$ be the distance between the center i and some target site j.

$$r_{CLOSE_x(i,j)}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1.5 & \text{iff } d(i,j) < \epsilon \\ 1 - tanh^2(\frac{d(i,j)}{10}) & \text{else,} \end{cases} \tag{22}$$

In an extended experiment, the agent is trained to bring the object to a specified target position, as well as to hover it above it. For this, we added several more rewards based on a fixed target site.

- *CLOSE(i, j), AT(i, j)*: maximal if the center of object i is within 10cm respectively 1.5cm of the target j. (equivalent to CLOSE$_{10cm}$(i,j) and CLOSE$_{1.5cm}$(i,j) in Eq. 22)
- *ABOVE_CLOSE(i, j), ABOVE_AT(i, j)*: maximal if the center of object i is within 10cm respectively 2cm of a site 6cm above the target j. (equivalent to CLOSE$_{10cm}$(i,j+6cm) and CLOSE$_{2cm}$(i,j+6cm) in Eq. 22)

B. Additional model details

For the SAC-X experiments we use a shared network architecture to instantiate the policy for the different intentions. The same basic architecture is also used for the critic Q value function. Formally, θ and ϕ in the main paper thus consist of the parameters of these two neural networks (and gradients for individual intentions wrt. these model parameters are averaged).

In detail: the stochastic policy consists of a layer of 200 hidden units with ELU units (Clevert et al., 2015), that is shared across all intentions. After this first layer a LayerNorm (Ba et al., 2016) is placed to normalize activations (we found this to generally be beneficial when switching between different environments that have differently scaled observations). The LayerNorm output is fed to a second shared layer with 200 ELU units. The output of this shared stack is routed to blocks of 100 and 18 ELU units followed by a final tanh activation. This output determines the parameters for a normal distributed policy with 9 outputs (whose variance we allow to vary between 0.3 and 1 by transforming the corresponding tanh output accordingly). For the critic we use the same architecture, but with 400 units per layer in the shared part and a 200-1 head for each intention. Figure 1 shows a depiction of this model architecture (where intention 0 would correspond to the main task). For the pixel based experiments a CNN stack consisting of two convolutional layers (16 feature maps each, with a kernel size of 3 and stride 2) processes two, stacked, input images of 48 x 48 pixels. The output of this stack is fed to a 200 dimensional linear layer (again with ELU activations) and concatenated to the output of the first layer in the above described architecture (which now only processes proprioceptive information).

The intentions are 1 hot encoded and select which head of the network is active for the policy and the value function. Other network structures (such as feeding the selected intention into the network directly) worked in general, but the gating architecture described here gave the best results – with respect to final task performance – in preliminary experiments.

Training of both policy and Q-functions was performed via ADAM (Kingma & Ba, 2015) using a learning rate of $2 \cdot 10^{-4}$ (and default parameters otherwise). See also the next section for details on the algorithm.

B.1. Stochastic Value Gradient for Learned Intentions

The following presents a detailed derivation of the stochastic value gradient – Equation (9) in the main paper – for learning the individual intention policies. Without loss of generality, we assume Gaussian policies for all intentions (as used in all our experiments). We can then first reparaemeterize the sampling process for policy $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}_t, \mathcal{T})$ as

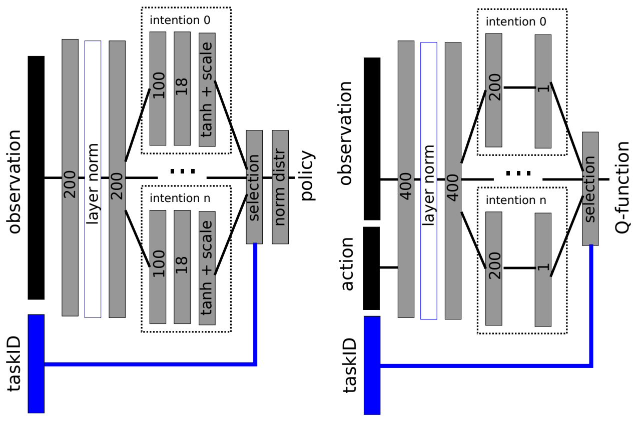

Figure 1. Schematics of the fully connected networks used to parameterize policy distribution and Q-functions for each intention.

$g_\theta(\mathbf{s}_t, \epsilon_{\mathbf{a}})$, where $\epsilon_{\mathbf{a}}$ is a random variable drawn from an appropriately chosen base distribution. That is, for a Gaussian policy we can use a normal distribution (Kingma & Welling, 2014; Rezende et al., 2014) $\epsilon_{\mathbf{a}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, with I denoting the identity matrix. More precisely, let $\sim \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}_t, \mathcal{T}) = \mathcal{N}(\mu_\theta(\mathbf{s}_t), \sigma^2_\theta(\mathbf{s}_t))$, then $g_\theta(\mathbf{s}_t, \epsilon_a) = \mu_\theta(\mathbf{s}_t) + \sigma_\theta(\mathbf{s}_t) * \epsilon_{\mathbf{a}}$. With this definition in place we can re-write the gradient as

$$\begin{aligned}
&\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\
&\approx \sum_{\substack{\mathcal{T} \in \mathfrak{T} \\ \tau \sim B}} \nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t, \mathcal{T}) \\ \mathbf{s}_t \in \tau}} \Big[\hat{Q}^{\pi}_{\mathcal{T}}(\mathbf{s}_t, \mathbf{a}; \phi) + \alpha \log \pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s}_t, \mathcal{T}) \Big], \\
&= \sum_{\substack{\mathcal{T} \in \mathfrak{T} \\ \tau \sim B}} \mathop{\mathbb{E}}_{\substack{\epsilon_{\mathbf{a}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ \mathbf{s}_t \in \tau}} \Big[\nabla_g \hat{Q}^{\pi}_{\mathcal{T}}(\mathbf{s}_t, g_\theta(\mathbf{s}_t, \epsilon_{\mathbf{a}}); \phi) \nabla_{\boldsymbol{\theta}} g_\theta(\mathbf{s}_t, \epsilon_{\mathbf{a}}) \\
&\quad + \alpha \nabla_g \log \pi_{\boldsymbol{\theta}}(g(\mathbf{s}_t, \epsilon_{\mathbf{a}})|\mathbf{s}_t, \mathcal{T}) \nabla_{\boldsymbol{\theta}} g_\theta(\mathbf{s}_t, \epsilon_{\mathbf{a}}) \Big].
\end{aligned} \tag{23}$$

C. SAC-Q algorithm

To allow for fast experimentation we implement our algorithm in a distributed manner, similar to recent distributed off-policy implementations from the literature (Gu et al., 2017; Horgan et al., 2018). In particular, we perform asynchronous learning and data acquisition in the following way: Except for the real world experiment, in which only a single robot – one actor connected to 10 learners – is used, we launch 36 actor processes that gather experience. These actors are connected to 36 learners (we used a simple 1-to-1 mapping) and send experience over at the end of each episode. To allow for fast learning of the scheduling choices each actor also performs Monte Carlo estimation of the Scheduling rollouts (i.e. it keeps its own up-to-date scheduler). The complete procedure executed by each actor is given in Algorithm 3.

The learners then aggregate all collected experience inside a replay buffer and calculate gradients for the policy and Q-function networks, as described in Algorithm 2.

Each learner then finally sends gradients to a central parameter server, that collects $G = 36$ gradients, updates the parameters and makes them available for both learners and actors; see the algorithm listing in Algorithm 1.

Note that this setup also makes experimentation on a real robot easy, as learning and acting (the part of the procedure that needs to be executed on the real robot) are cleanly separated.

Algorithm 1 SAC-Q (parameter server)

```
  Input: $G$ number of gradients to average
  Initialize parameters $\theta, \phi$
  while True do
    initialize N = 0
    initialize gradient storage $d_\theta = \{\}, d_\phi = \{\}$
    while $N < G$ do
      receive next gradients from learner $i$
      $d_\phi^{\mathcal{T}} = d_\phi^{\mathcal{T}} \cup \{\delta\phi^i\}$
      $d_\theta = d_\theta \cup \{\delta\theta^i\}$
    end while
    update parameters with averages from gradient store:
    $\phi = \text{ADAM\_update}(\phi, \frac{1}{|d_\phi|} \sum_{\delta_\phi \in d_\phi} \delta_\phi)$
    $\theta = \text{ADAM\_update}(\theta, \frac{1}{|d_\theta|} \sum_{\delta_\theta \in d_\theta} \delta_\theta)$
    send new parameters to workers
  end while
```

Algorithm 2 SAC-Q (learner)

```
  Input: $N_{\text{learn}}$ number of learning iterations, $\alpha$ entropy regularization parameter Fetch initial parameters $\theta, \phi$
  initialize N = 0
  while $N < N_{\text{learn}}$ do
    update replay buffer $B$ with received trajectories
    for k=0,1000 do
      sample a trajectory $\tau$ from $B$
      // compute gradients for policy and Q
      $\delta_\phi = \frac{1}{|\mathfrak{T}|} \sum_{\mathcal{T} \in \mathfrak{T}} \nabla_\phi L(\phi)$
      $\delta_\theta = \nabla_\theta L(\theta)$
      send $(\delta_\theta, \delta_\phi)$ to parameter server
      wait for parameter updates
      fetch new parameters $\phi$, $\theta$
    end for
    // update target networks
    $\phi' = \phi$, $\theta' = \theta$
    $N = N + 1$
  end while
```

Algorithm 3 SAC-Q (actor)

```
  Input: $N_{\text{trajectories}}$ number of total trajectories requested, $T$ steps per episode, $\xi$ scheduler period
  initialize N = 0
  // Initialize Q-table
  $\forall \mathcal{T}_h, \mathcal{T}_{0:h-1} : Q(\mathcal{T}_{0:h-1}, \mathcal{T}_h) = 0$, $M_{\mathcal{T}_h} = 0$
  while $N < N_{\text{trajectories}}$ do
    fetch parameters $\theta$
    // collect new trajectory from environment
    $\tau = \{\}, h = 0$
    for t=0,T do
      if $t \pmod{\xi} \equiv 0$ then
        $\mathcal{T}_h \sim P_{\mathcal{S}}(\mathcal{T} | \mathcal{T}_{0:h-1})$
        $h = h + 1$
      end if
      $a_t \sim \pi_\theta(\mathbf{a}_0 | \mathbf{s}_t, \mathcal{T}_h)$
      // execute action and collect all rewards
      $\bar{r} = [r_{\mathcal{A}_1}(\mathbf{s}_t, \mathbf{a}_t), \ldots, r_{|\mathcal{A}|}(\mathbf{s}_t, \mathbf{a}_t), r_{\mathcal{M}}(\mathbf{s}_t, \mathbf{a}_t)]$
      $\tau \leftarrow \tau \cup \{(\mathbf{s}_t, \mathbf{a}_t, \bar{r}, \pi_\theta(\mathbf{a}_0 | \mathbf{s}_t, \mathcal{T}_h))\}$
    end for
    send $\tau$ and schedule decisions $\mathcal{T}_{0:H}$ to learner
    // update Monte Carlo Q for scheduler$P_S$
    for h=0:H do
      $M_{\mathcal{T}_h} = M_{\mathcal{T}_h} + 1$
      $Q(\mathcal{T}_{0:h-1}, \mathcal{T}_h) \mathrel{+}= \frac{R^{\tau}_{\mathcal{M}}(\mathcal{T}_{h:H}) - Q(\mathcal{T}_{0:h-1}, \mathcal{T}_h)}{M}$
    end for
    $N = N + 1$
  end while
```

D. Additional Experiment Results

D.0.1. A detailed look at the SAC-Q learning process

In Figure 2 we show the reward statistics over the full set of auxiliary and extrinsic tasks for both SAC-U (left) and SAC-Q (right) when learning the stacking task. While our main goal is to learn the extrinsic stacking task, we can observe that the SAC-X agents are able to learn all auxiliary intentions in parallel. In this example we use a set of 13 auxiliary intentions which are defined on the state of the robot and the two blocks in the scene as in Section A.1.1. These are *TOUCH*, *NOTOUCH*, *MOVE(1)*, *MOVE(2)*, *CLOSE(1,2)*, *ABOVE(1,2)*, *BELOW(1,2)*, *LEFT(1,2)*, *RIGHT(1,2)*, *ABOVECLOSE(1,2)*, *BELOWCLOSE(1,2)*, *LEFTCLOSE(1,2)*, *RIGHTCLOSE(1,2)*. In addition we have the extrinsic reward, which is defined as *STACK(1)* in this case. SAC-U (shown in the top part of the figure) will execute all intentions in a uniform order. Some of the intention goals (such as for *NOTOUCH*, *WEST*, *EAST*) can be valid starting states of an episode and will see their reward signals very early in the learning process. Other reward signals, such as *MOVE* and *TOUCH*, are more difficult to learn and will lead to rich interaction with the environment which are, in turn, a requirement for learning even more difficult intentions. In this example, after *NORTH* and *NORTHCLOSE* are learned, *PILE(1)* can be learned reliably as well.

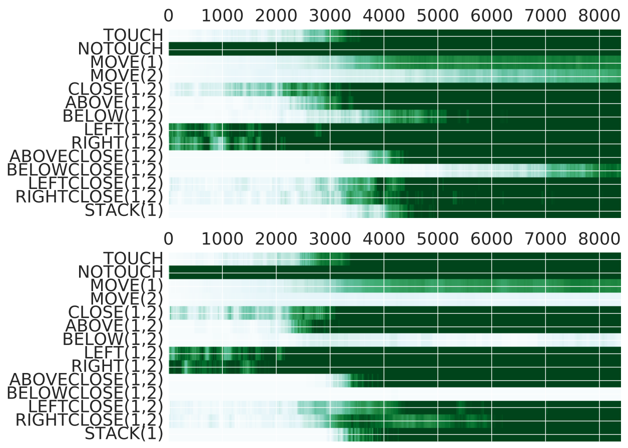

Figure 2. Comparison of full auxiliary and extrinsic set of intentions learned of SAC-U (top) and SAC-Q (bottom) over the training process. The x axis is episodes per actor and the color intensity encodes the obtained reward for each depicted intention.

The SAC-Q agent in contrast tries to select only auxiliary tasks that will help to collect reward signals for the extrinsic intentions. In the bottom plot in Figure 2, we can see that by ignoring the auxiliaries *MOVE(2)*, *SOUTH* and *SOUTHCLOSE*, SAC-Q manages to learn the extrinsic task faster.

The learned distribution of Q values at the end of training can also be seen in Figure 3 (plotted for pairs of executed intentions). We can observe that executing the sequence (*STACK(1)*, *STACK(1)*), gives the highest value, as expected. But SAC-Q also found other sequences of intentions that will help to collect reward signals for *STACK(1)*.

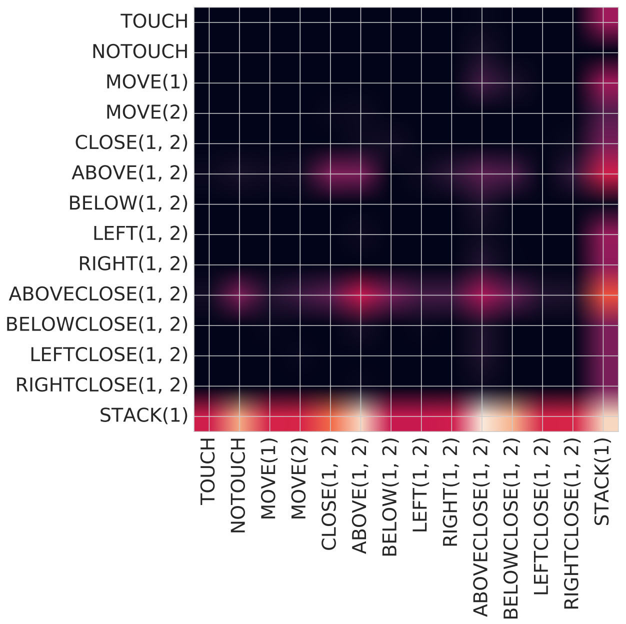

Figure 3. SAC-Q learned Q value distribution for the scheduler. We plot the Q-values after training for pairs of executed intentions. That is, the Q value after first executing the intention denoted by the row names and then executing the intention denoted by the column name. Lighter colors here indicate a higher extrinsic stacking reward.

A full set of plots for the clean-up tasks is also shown in Figures 4 to 7, comparing the SAC-U and SAC-Q results over all auxiliaries and extrinsic tasks. While SAC-Q and SAC-U both learn all tasks, only SAC-Q manages to learn the most difficult sparse clean-up task. As shown in the plots, the learned scheduler is more efficient in learning the auxiliaries, as well as the extrinsic tasks, at least in the beginning of the learning process. In later stages, SAC-Q will try to concentrate on intentions that will help it solve the extrinsic tasks, and therefore may disregard some of the less important auxiliaries (e.g. CLOSE(1,2)).

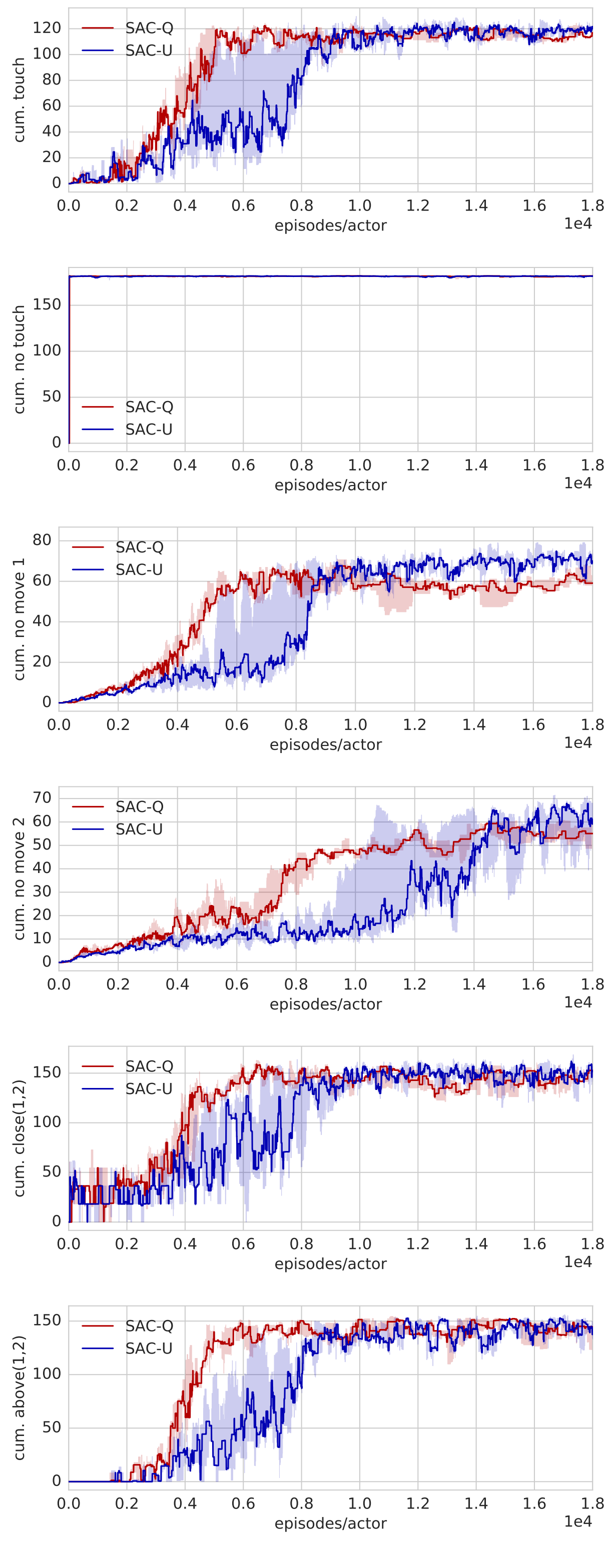

Figure 4. Cleanup experiment, SAC-Q learns all six extrinsic tasks reliably. In addition it reliably learns also to solve the 15 auxiliary tasks in parallel. Part 1: auxiliaries 1-6.

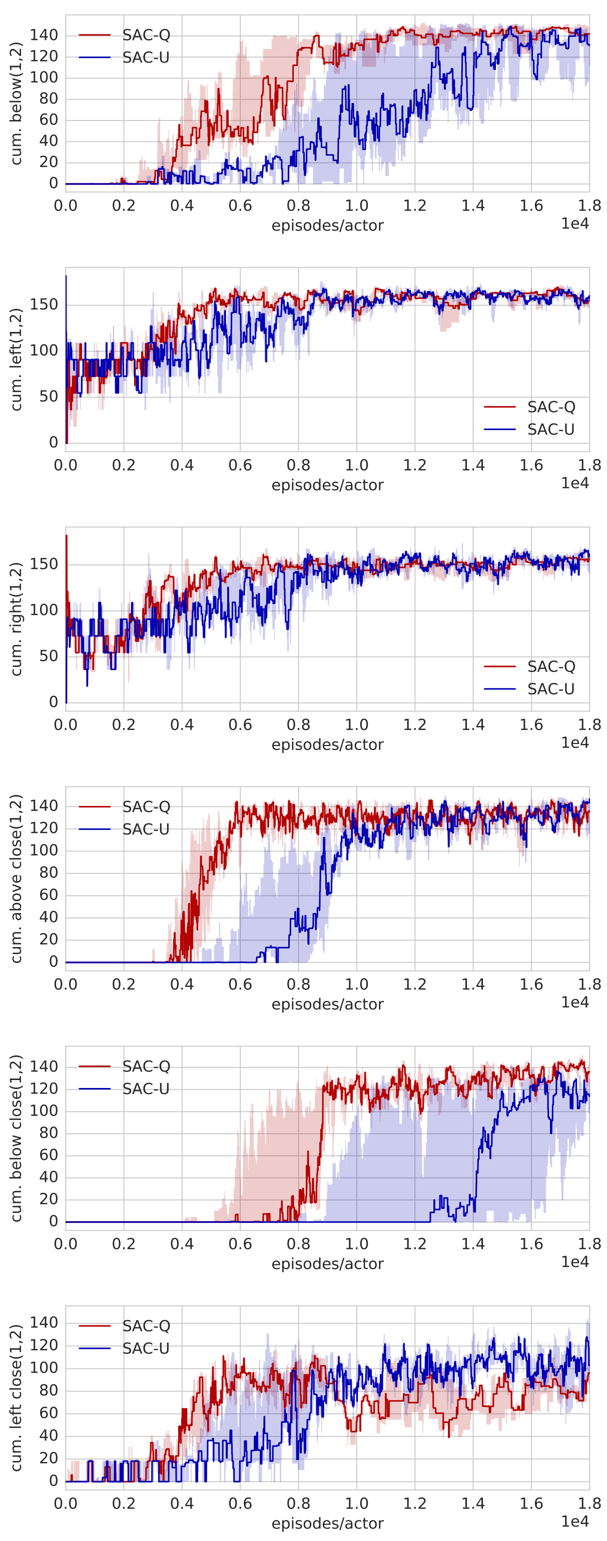

Figure 5. Cleanup experiment, SAC-Q learns all six extrinsic tasks reliably. In addition it reliably learns also to solve the 15 auxiliary tasks in parallel. Part 2: auxiliaries 7-12.

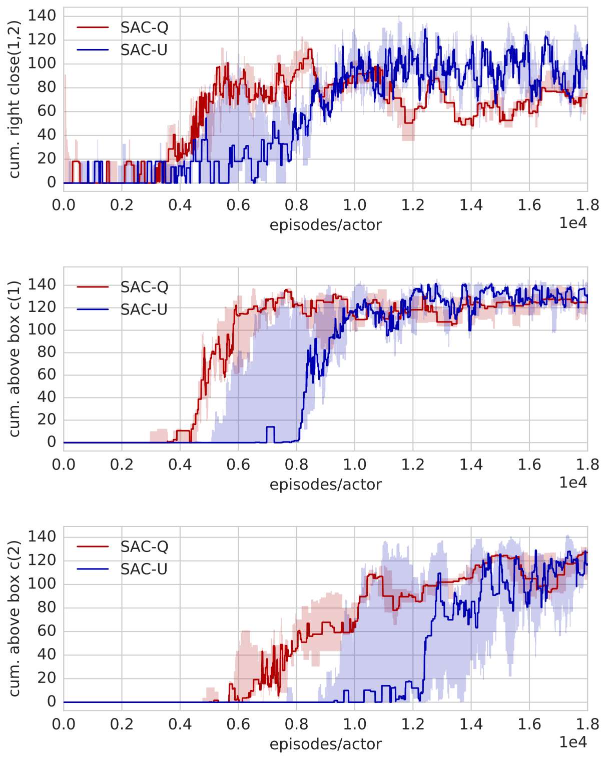

Figure 6. Cleanup experiment, SAC-Q learns all six extrinsic tasks reliably. In addition it reliably learns also to solve the 15 auxiliary tasks in parallel. Part 3: auxiliaries 13-15.

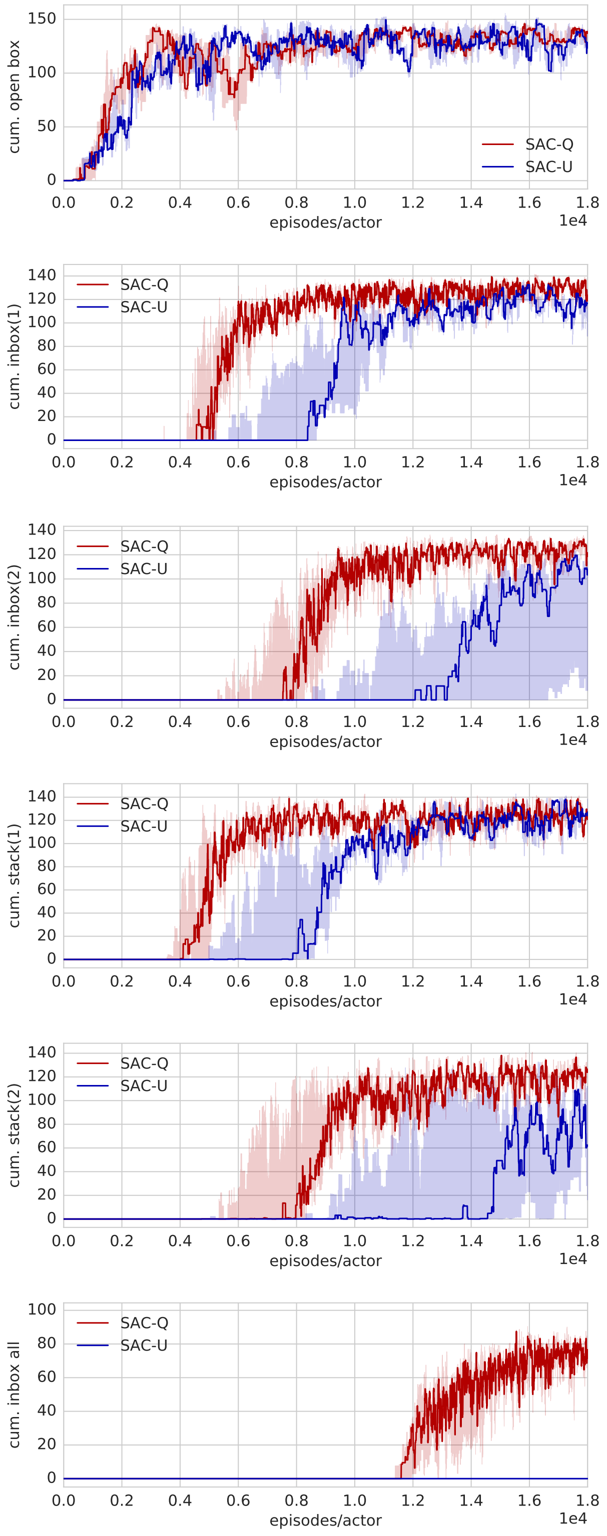

Figure 7. Cleanup experiment, SAC-Q learns all six extrinsic tasks reliably. In addition it reliably learns also to solve the 15 auxiliary tasks in parallel. Part 4: extrinsic tasks.

References

Ba, L. J., Kiros, R., and Hinton, G. E. Layer normalization. *CoRR*, arXiv:abs/1607.06450, 2016.

Clevert, D., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, arXiv:abs/1511.07289, 2015.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. Distributed prioritized experience replay. 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *The International Conference on Learning Representations (ICLR)*, 2014.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.