Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

Learning from demonstrations and task rewards

Spring 2020, CMU 10-403

Katerina Fragkiadaki

Learning from demonstrations

Pros

- Can much accelerate trial-and-error learning by suggesting good actions to try
- Can help us train initial safe policies, to deploy in the real world

Cons

- Time consuming
- May include suboptimal, noise and diverse ways to perform the task
- When you imitate, you cannot surpass the “expert”.

Learning from task rewards

Pros

- Cheap supervision
- Optimizes the right end task, as encoded in the task rewards

Cons

- Super sample inefficient - impossible to have in the real world right now
- Initial policy is random thus unsafe to deploy in the real world

Learning from demonstrations and task rewards

Goals

- More sample efficient that RL alone
- Good/safe initial performance
- Outperform the human expert

Challenges for kinesthetic demonstrations

- Handling expert sub optimality

Additional challenges for learning from video demonstrations

- requires visual perception
- requires handling mismatch between imitator and demonstrator action spaces

Learning from demonstrations and task rewards

Goals

- More sample efficient that RL
- Good/safe initial performance
- Outperform the human expert

Challenges for kinesthetic demonstrations

- Handling expert sub optimality

Additional challenges for learning from video demonstrations

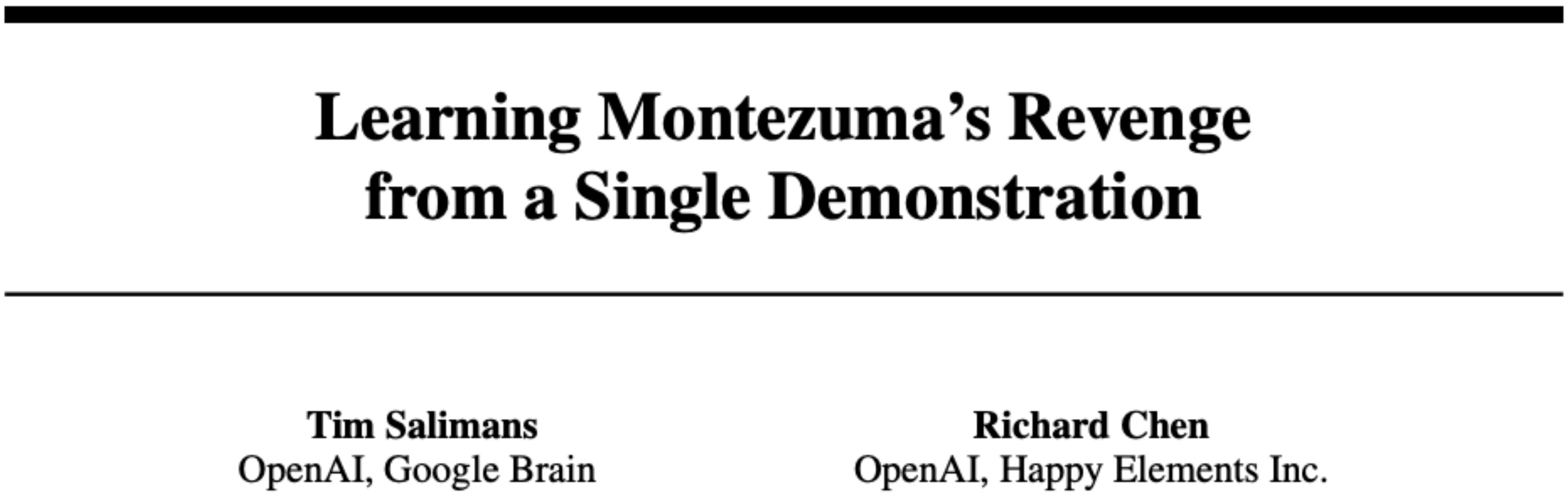
- requires visual perception
- requires handling mismatch between imitator and demonstrator action spaces

Learning from demonstrations and task rewards

- Initialize the replay buffer with demos (which will be later either removed, or kept forever) and start your model-free RL method
- Pre-train the model-free RL method (a policy and a consistent with it value function) with a demonstration only buffer, then fine-tune it.
- Combine imitation and task rewards
- Exploit the temporal structure, and step progressively earlier and earlier in time along a trajectory, to solve progressively longer horizon tasks, as opposed to solving them at once.

Learning from demonstrations and task rewards

- Initialize the replay buffer with demos (which will be later either removed, or kept forever) and start your model-free RL method
- Pre-train the model-free RL method with a demonstration only buffer, then fine-tune it.
- Combine imitation and task rewards
- Exploit the temporal structure, and step progressively earlier and earlier in time along a trajectory, to solve progressively longer horizon tasks, as opposed to solving them at once.



Learning Montezuma's Revenge from a Single Demonstration

Tim Salimans
OpenAI, Google Brain

Richard Chen
OpenAI, Happy Elements Inc.

- What makes model-free RL hard is distant in time and sparse rewards
- Insights:
  - Do not imitate the trajectory actions! Just reset from a state along the trajectory instead from the initial state.
  - Reset progressively earlier in time. Make the task easier to solve by decomposing it into a curriculum of subtasks requiring short action sequences.

The hopelessness of learning from distant rewards

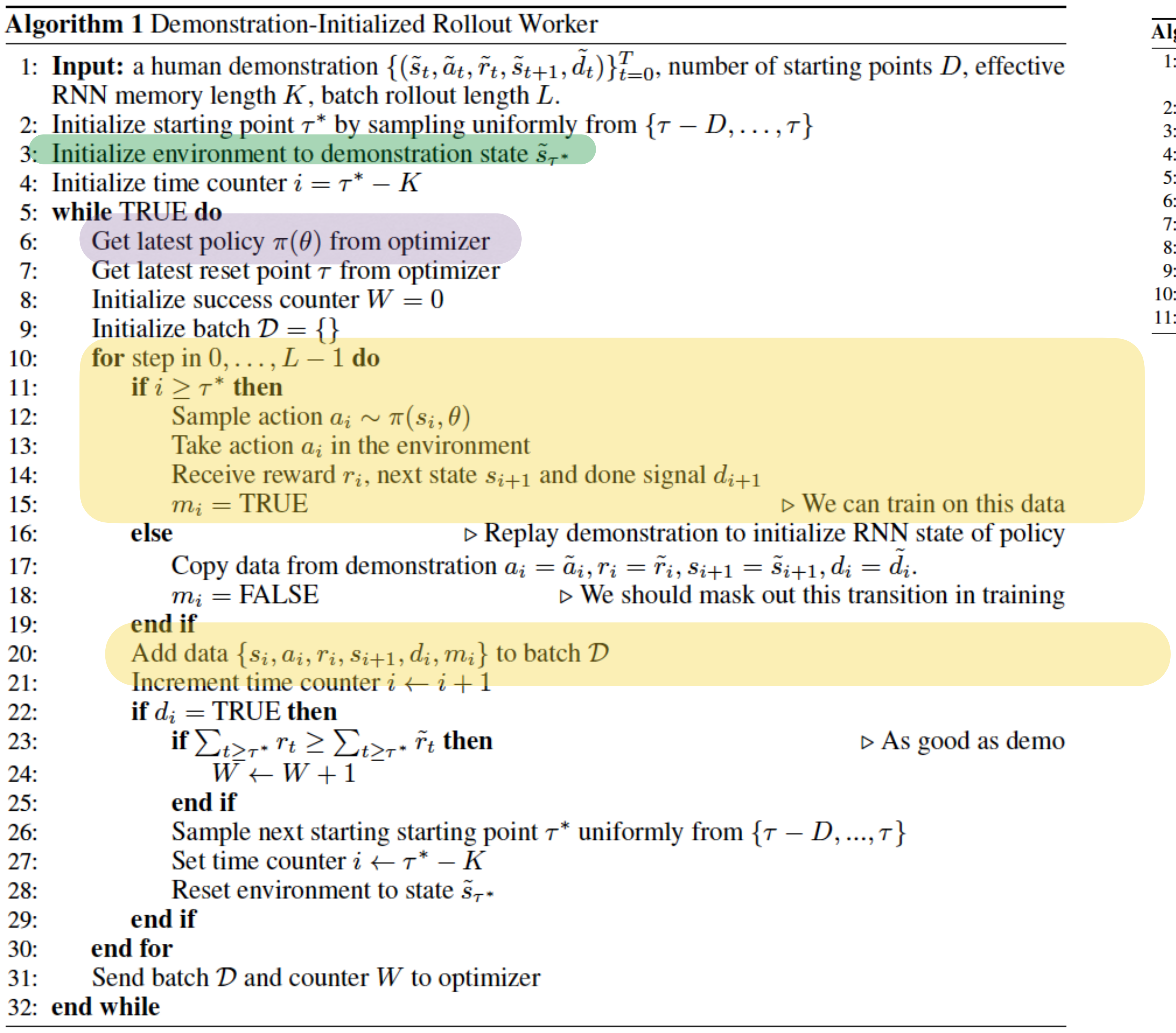
Imagine we need 5 steps till the first reward (getting a key).

$$\begin{aligned} p(\text{get first key}) = {} & p(\text{get down ladder 1}) * p(\text{get down rope}) * \\ & p(\text{get down ladder 2}) * p(\text{jump over skull}) * p(\text{get up ladder 3}). \end{aligned}$$

Taking uniformly random actions (naive exploration) in Montezuma's Revenge only produces a reward about once in every half a million steps. This probability decreases

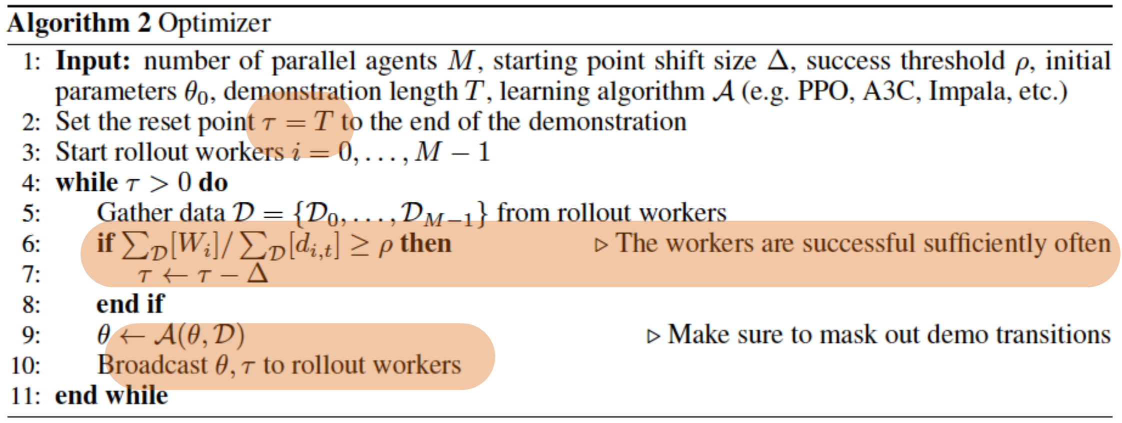
- as the number of actions increases and
- the time-horizon till reward increases.

reset to a state from the demo



Algorithm 1 Demonstration-Initialized Rollout Worker

```
Input: a human demonstration $\{(\tilde{s}_t, \tilde{a}_t, \tilde{r}_t, \tilde{s}_{t+1}, \tilde{d}_t)\}_{t=0}^{T}$, number of starting points $D$, effective
RNN memory length $K$, batch rollout length $L$.
Initialize starting point $\tau^*$ by sampling uniformly from $\{\tau - D, \ldots, \tau\}$
Initialize environment to demonstration state $\tilde{s}_{\tau^*}$
Initialize time counter $i = \tau^* - K$
while TRUE do
    Get latest policy $\pi(\theta)$ from optimizer
    Get latest reset point $\tau$ from optimizer
    Initialize success counter $W = 0$
    Initialize batch $\mathcal{D} = \{\}$
    for step in $0, \ldots, L-1$ do
        if $i \geq \tau^*$ then
            Sample action $a_i \sim \pi(s_i, \theta)$
            Take action $a_i$ in the environment
            Receive reward $r_i$, next state $s_{i+1}$ and done signal $d_{i+1}$
            $m_i$ = TRUE                                   ▷ We can train on this data
        else                       ▷ Replay demonstration to initialize RNN state of policy
            Copy data from demonstration $a_i = \tilde{a}_i, r_i = \tilde{r}_i, s_{i+1} = \tilde{s}_{i+1}, d_i = \tilde{d}_i$.
            $m_i$ = FALSE                      ▷ We should mask out this transition in training
        end if
        Add data $\{s_i, a_i, r_i, s_{i+1}, d_i, m_i\}$ to batch $\mathcal{D}$
        Increment time counter $i \leftarrow i + 1$
        if $d_i$ = TRUE then
            if $\sum_{t \geq \tau^*} r_t \geq \sum_{t \geq \tau^*} \tilde{r}_t$ then                ▷ As good as demo
                $W \leftarrow W + 1$
            end if
            Sample next starting starting point $\tau^*$ uniformly from $\{\tau - D, ..., \tau\}$
            Set time counter $i \leftarrow \tau^* - K$
            Reset environment to state $\tilde{s}_{\tau^*}$
        end if
    end for
    Send batch $\mathcal{D}$ and counter $W$ to optimizer
end while
```



Algorithm 2 Optimizer

```
Input: number of parallel agents $M$, starting point shift size $\Delta$, success threshold $\rho$, initial
parameters $\theta_0$, demonstration length $T$, learning algorithm $\mathcal{A}$ (e.g. PPO, A3C, Impala, etc.)
Set the reset point $\tau = T$ to the end of the demonstration
Start rollout workers $i = 0, \ldots, M-1$
while $\tau > 0$ do
    Gather data $\mathcal{D} = \{\mathcal{D}_0, \ldots, \mathcal{D}_{M-1}\}$ from rollout workers
    if $\sum_{\mathcal{D}}[W_i] / \sum_{\mathcal{D}}[d_{i,t}] \geq \rho$ then   ▷ The workers are successful sufficiently often
        $\tau \leftarrow \tau - \Delta$
    end if
    $\theta \leftarrow \mathcal{A}(\theta, \mathcal{D})$                        ▷ Make sure to mask out demo transitions
    Broadcast $\theta, \tau$ to rollout workers
end while
```

Reset time point
to earlier in time

Solve the subtask from τ
till T with standard model-free RL

I do not start from scratch!
I start from the policy
learnt from the previous
time segment

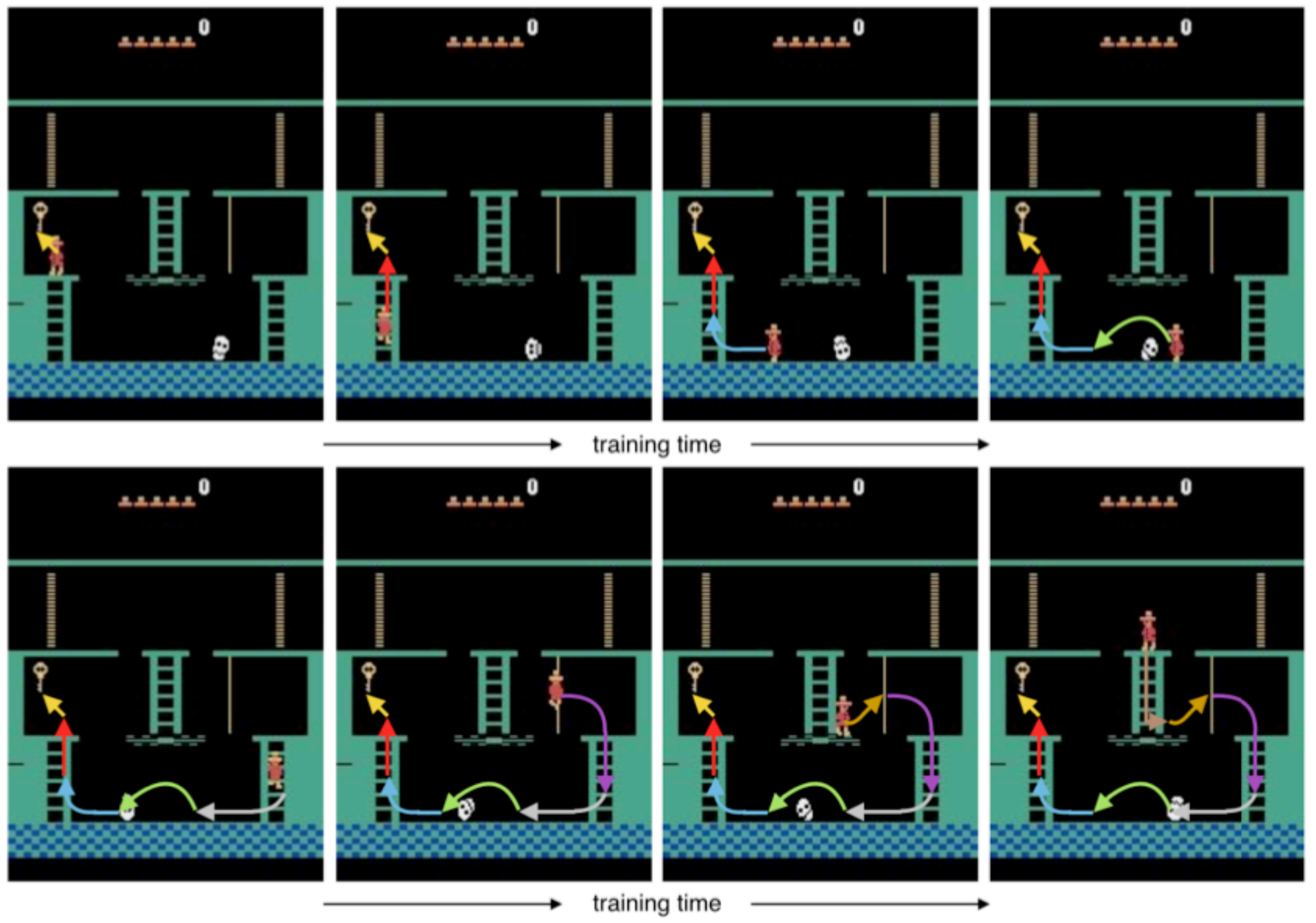
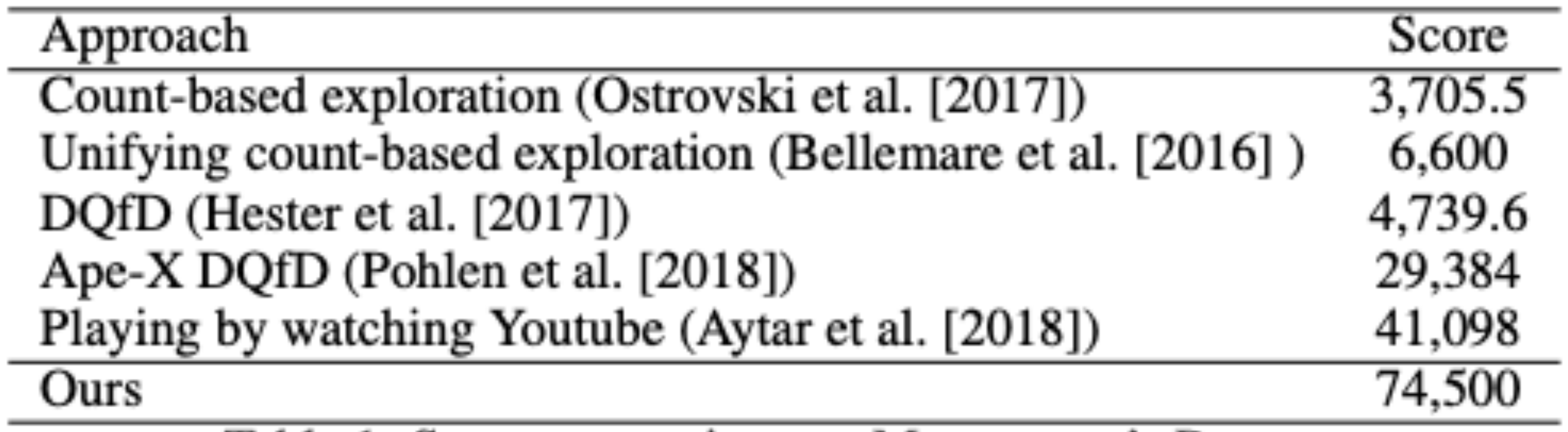


Figure 1: Impression of our agent learning to reach the first key in Montezuma's Revenge using RL and starting each episode from a demonstration state. When our agent starts playing the game, we place it right in front of the key, requiring it to only take a single jump to find success. After our agent has learned to do this consistently, we slowly move the starting point back in time. Our agent might then find itself halfway up the ladder that leads to the key. Once it learns to climb the ladder from there, we can have it start at the point where it needs to jump over the skull. After it learns to do that, we can have it start on the rope leading to the floor of the room, etc. Eventually, the agent starts in the original starting state of the game and is able to reach the key completely by itself.

Learning Montezuma's Revenge from a Single Demonstration, Salimans and Chen



Approach	Score
Count-based exploration (Ostrovski et al. [2017])	3,705.5
Unifying count-based exploration (Bellemare et al. [2016])	6,600
DQfD (Hester et al. [2017])	4,739.6
Ape-X DQfD (Pohlen et al. [2018])	29,384
Playing by watching Youtube (Aytar et al. [2018])	41,098
Ours	74,500

Table 1: Score comparison on Montezuma's Revenge

Learning from demonstrations and task rewards

- Initialize the replay buffer with demos (which will be later either removed, or kept forever) and start your model-free RL method
- Pre-train the model-free RL method with a demonstration only buffer, then fine-tune it.
- Combine imitation and task rewards
- Exploit the temporal structure, and step progressively earlier and earlier in time along a trajectory, to solve progressively longer horizon tasks, as opposed to solving them at once.

Off-policy RL

- Off-policy RL learns from data collected under a behavioral policy different than the current policy. In what we have seen thus far, “off-policy” transitions are generated from earlier versions of the current policy, they are thus heavily correlated to the current policy. Not that much *off-policy* after all.

- Batch RL learns from **a fixed experience buffer that does not grow with data collected from a near on policy exploratory policy**. This is truly off-policy..

Thing to remember: off-policy RL (e.g., Q learning or DDPG)do not work with truly off policy experience tuples.

Off-policy model-free RL from demonstrations

- Problem with this?
- Convergence of off-policy methods relies on the assumption of visiting each (s,a) pair infinitely many times. Demonstrations are highly biased transitions of the environment, and violate that assumption.
- The states and actions in the demonstrations generally have higher Q-values than other states and actions. Q-learning will push up Q(s; a) in a sampled state s. However, since the values or the bad actions are not observed, the Q-function has no way of knowing whether the action itself is good, or whether all actions in that state are good, so the demonstrated action will not necessarily have a higher Q-value than other actions in the demonstrated state.

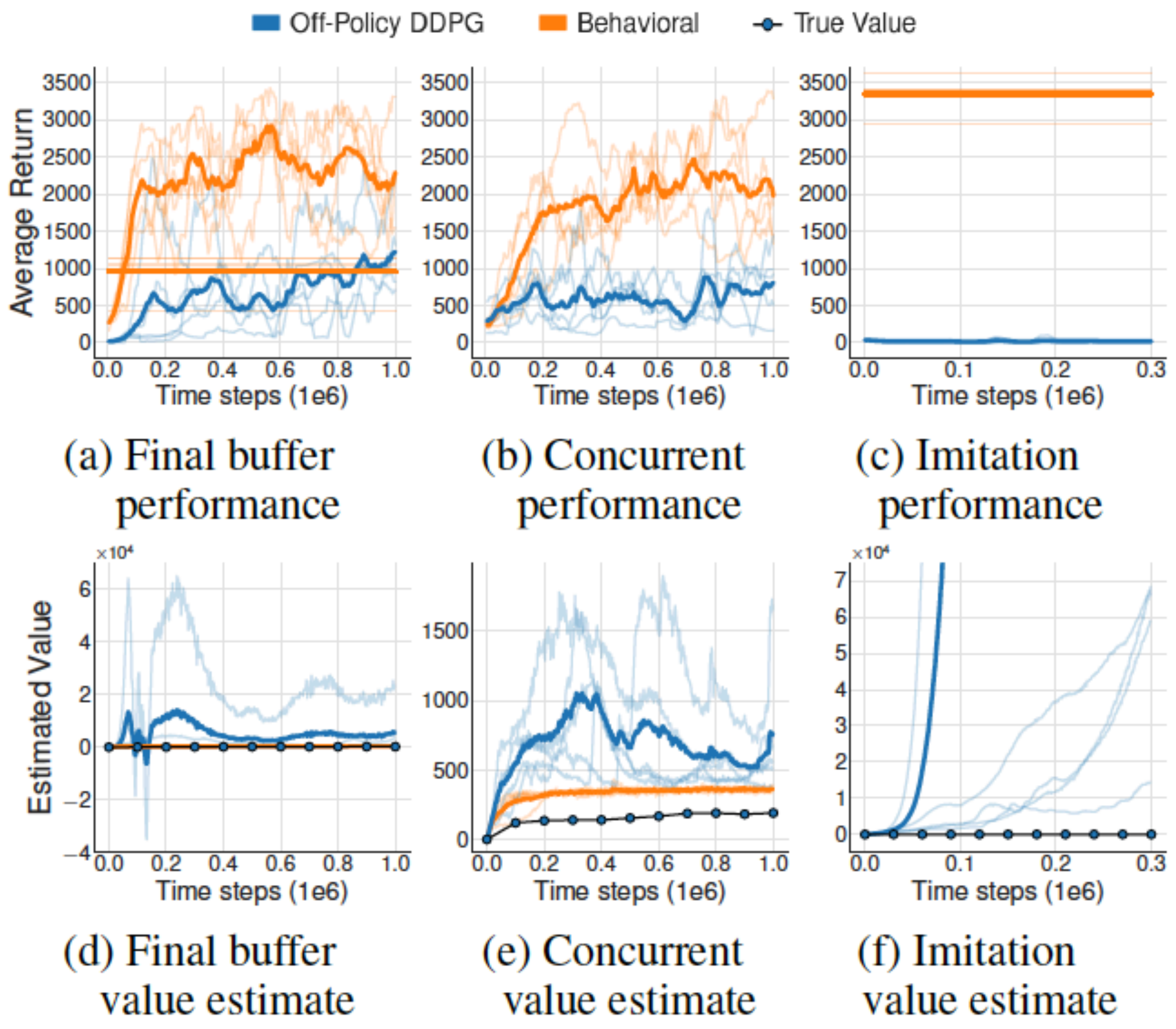
Reinforcement Learning from Imperfect Demonstrations, Gan et al.

Truly off policy RL does not work

Final buffer: We train a DDPG agent for 1 million time steps, adding N (0, 0.5) Gaussian noise to actions for high exploration, and store all experienced transitions. This collection procedure creates a dataset with a diverse set of states and actions, with the aim of sufficient coverage.

Concurrent: We concurrently train the off-policy and behavioral DDPG agents, for 1 million time steps. To ensure sufficient exploration, a standard N (0, 0.1) Gaussian noise is added to actions taken by the behavioral policy. Each transition experienced by the behavioral policy is stored in a buffer replay, which both agents learn from. As a result, both agents are trained with the identical dataset.

Imitation: A trained DDPG agent acts as an expert, and is used to collect a dataset of 1 million transitions, and populates a buffer, from which the off policy agent learns.



(a) Final buffer performance (b) Concurrent performance (c) Imitation performance

(d) Final buffer value estimate (e) Concurrent value estimate (f) Imitation value estimate

- DDPG (behavioral): (what we have seen in the course) a DDPG policy based on which actions are selected (with small exploration noise) and the experience buffer is populated

- (Truly) Off-policy DDPG: a DDPG policy that uses experience tuples from the buffer, *it does not influence in any way the data collected in the buffer*

  Conclusion: The DDPG update rule does not work anymore when used on data NOT collected based on the exploratory version of the policy.

Learning from demonstrations and task rewards

- Initialize the replay buffer with demos (which will be later either removed, or kept forever) and start your model-free RL method
- Pre-train the model-free RL method with a demonstration only buffer, then fine-tune it using both demo+reinforcement
- Combine imitation and task rewards
- Exploit the temporal structure, and step progressively earlier and earlier in time along a trajectory, to solve progressively longer horizon tasks, as opposed to solving them at once.

Idea: add a supervised margin loss

Standard DQN loss:

$$\mathscr{L}_{QL}(Q) = \left(\underbrace{\left[R(s,a) + \gamma \max_{a'} Q(s',a') \right]}_{\text{target}} - Q(s,a) \right)^2$$

The margin loss ensures the expert action has higher Q (by a margin) than the rest of the actions.

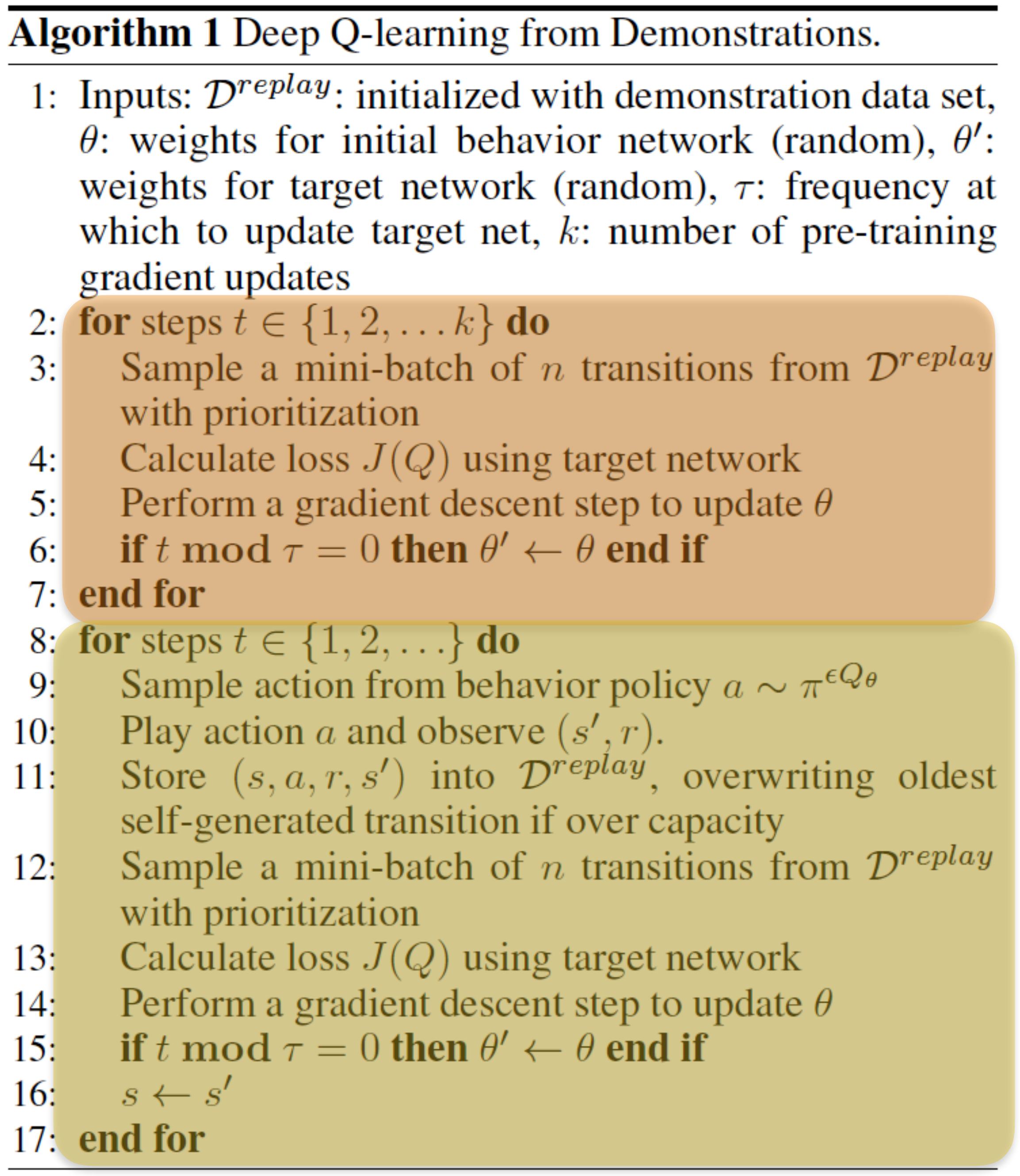
$$\mathscr{L}_{margin}(Q) = \left(\max_{a \in A} [Q(s,a) + \ell(a_E, a)] - Q(s, a_E) \right)^2$$

$\ell(a_E, a_E) = 0$ and $\ell(a_E, a) > 0,\ a \neq a_E$

This extra loss permits to pretrain solely on demo buffer in the beginning (truly off policy)

Deep Q learning from demonstrations, Hester et al.

V1.0: add a margin loss



Algorithm 1 Deep Q-learning from Demonstrations.

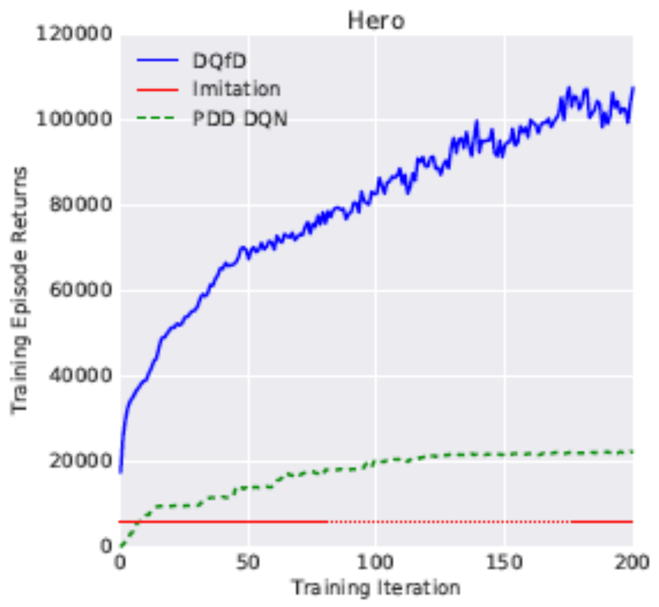
```
Inputs: $\mathcal{D}^{replay}$: initialized with demonstration data set, $\theta$: weights for initial behavior network (random), $\theta'$: weights for target network (random), $\tau$: frequency at which to update target net, $k$: number of pre-training gradient updates
for steps $t \in \{1, 2, \dots k\}$ do
    Sample a mini-batch of $n$ transitions from $\mathcal{D}^{replay}$ with prioritization
    Calculate loss $J(Q)$ using target network
    Perform a gradient descent step to update $\theta$
    if $t$ mod $\tau = 0$ then $\theta' \leftarrow \theta$ end if
end for
for steps $t \in \{1, 2, \dots\}$ do
    Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$
    Play action $a$ and observe $(s', r)$.
    Store $(s, a, r, s')$ into $\mathcal{D}^{replay}$, overwriting oldest self-generated transition if over capacity
    Sample a mini-batch of $n$ transitions from $\mathcal{D}^{replay}$ with prioritization
    Calculate loss $J(Q)$ using target network
    Perform a gradient descent step to update $\theta$
    if $t$ mod $\tau = 0$ then $\theta' \leftarrow \theta$ end if
    $s \leftarrow s'$
end for
```

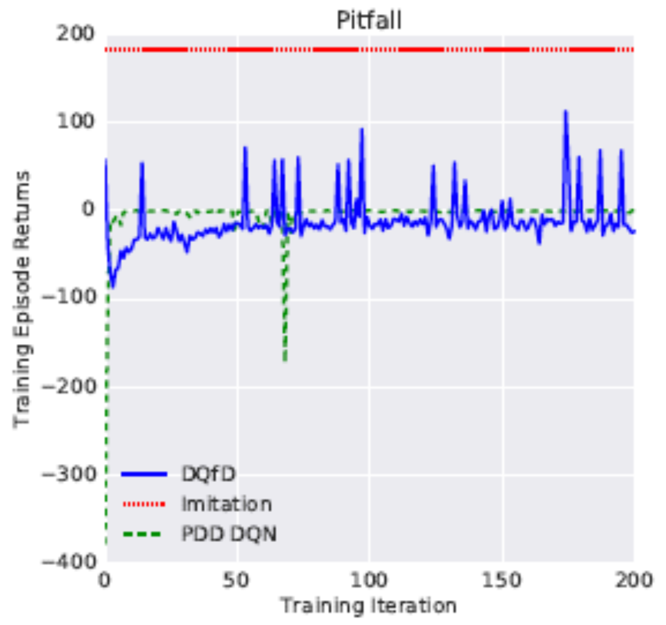
Pretraining only with demos

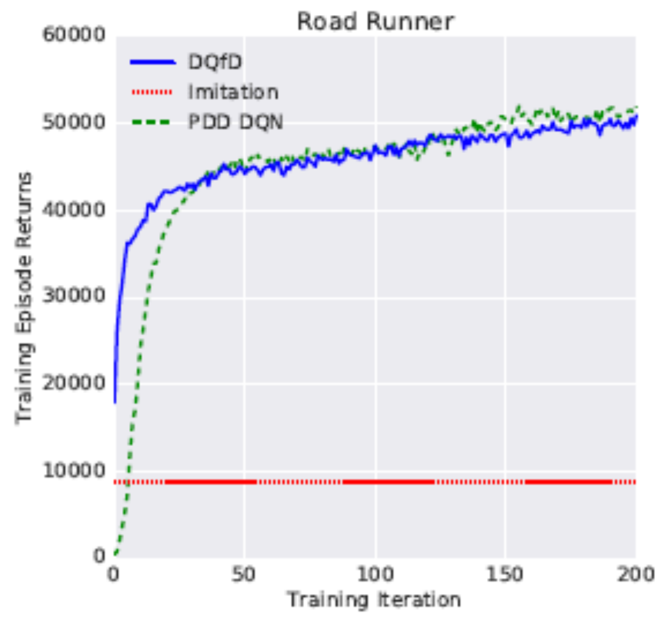
(using DQN and classification losses)

Fine-tuning jointly with demo+self generated transition in the replay buffer

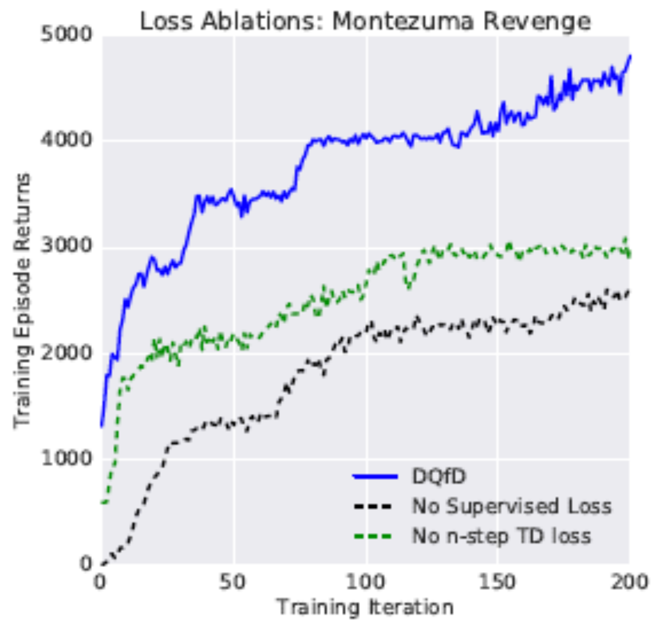
Deep Q learning from demonstrations, Hester et al.

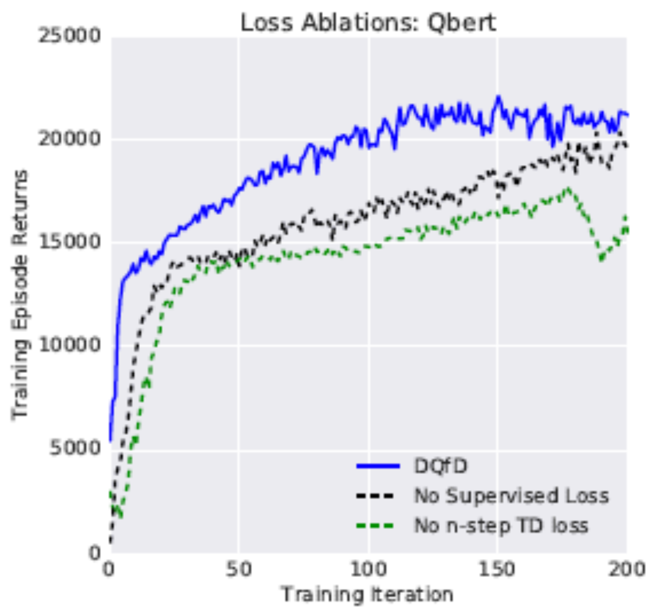




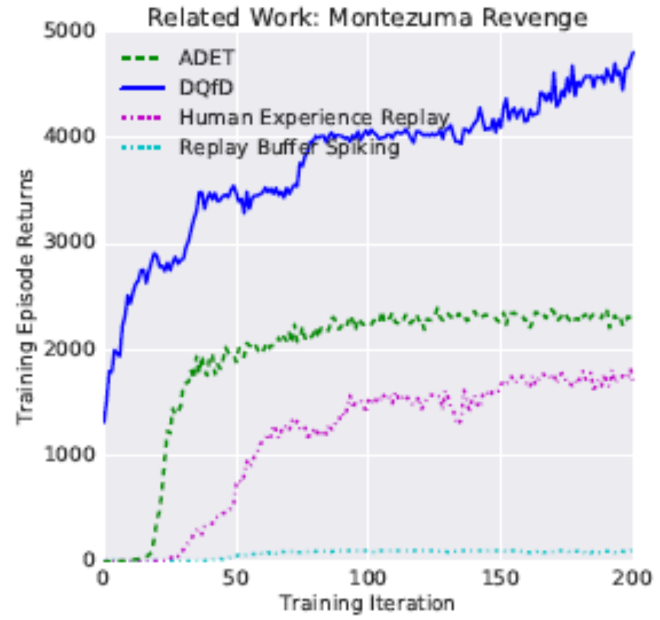


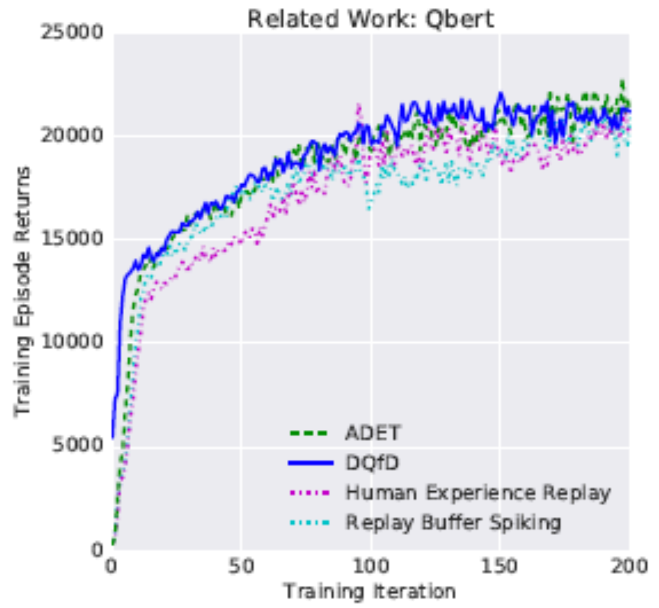
Outperforms both RL alone (PDD DQN) and imitation alone





Margin loss essential





Outperforms both just initializing the replay buffer with demos (RBS) as well as keeping demos around in the buffer (HER).

Deep Q learning from demonstrations, Hester et al.

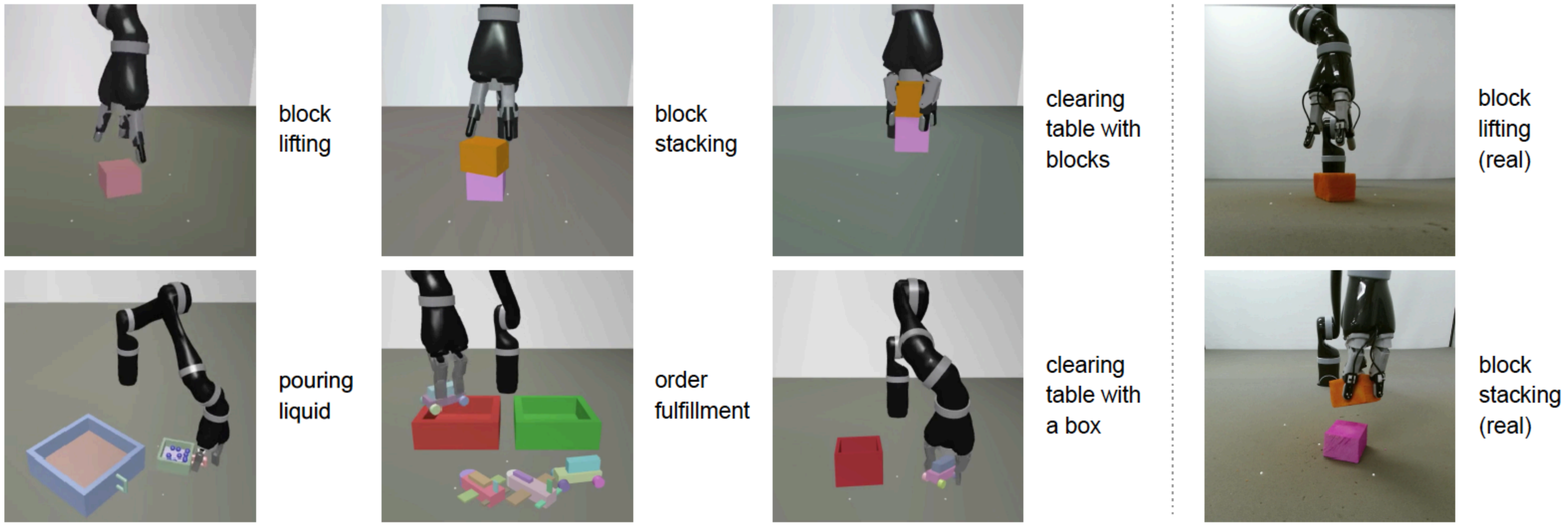
Learning from demonstrations and task rewards

- Initialize the replay buffer with demos (which will be later either removed, or kept forever) and start your model-free RL method
- Pre-train the model-free RL method with a demonstration only buffer, then fine-tune it.
- Combine imitation and task rewards
- Exploit the temporal structure, and step progressively earlier and earlier in time along a trajectory, to solve progressively longer horizon tasks, as opposed to solving them at once.

Reinforcement and Imitation Learning for Diverse Visuomotor Skills

Yuke Zhu[†] Ziyu Wang[‡] Josh Merel[‡] Andrei Rusu[‡] Tom Erez[‡] Serkan Cabi[‡]
Saran Tunyasuvunakool[‡] János Kramár[‡] Raia Hadsell[‡] Nando de Freitas[‡] Nicolas Heess[‡]
[†]Computer Science Department, Stanford University, USA
[‡]DeepMind, London, UK

Input: 30 kinesthetic trajectories (sequences of (s,a,r)) for each of the tasks.



Reinforcement and Imitation Learning for Diverse Visuomotor Skills

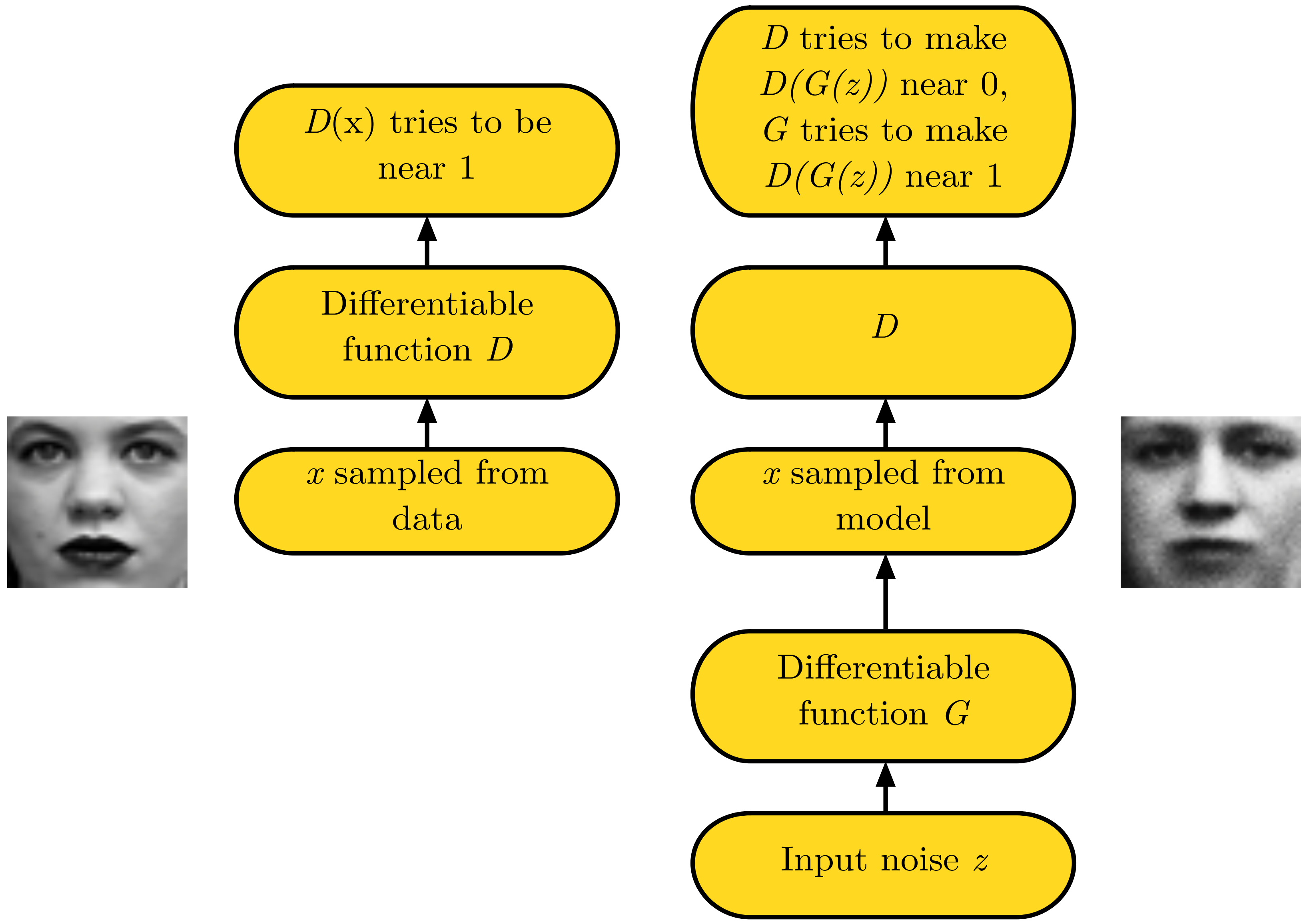
Yuke Zhu[†] Ziyu Wang[‡] Josh Merel[‡] Andrei Rusu[‡] Tom Erez[‡] Serkan Cabi[‡]
Saran Tunyasuvunakool[‡] János Kramár[‡] Raia Hadsell[‡] Nando de Freitas[‡] Nicolas Heess[‡]
[†]Computer Science Department, Stanford University, USA
[‡]DeepMind, London, UK

- Start episodes by setting the world in states of the demonstration trajectories. Have we seen this before?
- Combine imitation and task rewards
- Asymmetric actor-critic: the value network takes as input the low-dim state of the system and the policy is trained from pixels. What is the benefit of this?
- Co-train the policy CNN with auxiliary tasks: map images to object locations with regression and minimize L2 loss. Any object detection/semantic labelling task would work, e.g., learning to detect the robot's gripper is also a useful auxiliary task for training the visual features.
- Sim2REAL via domain randomization.

Combining imitation and task rewards

$$r(s,a) = \lambda r_{GAIL}(s,a) + (1-\lambda) r_{task}(s,a), \quad \lambda \in [0,1] \, .$$

Generative adversarial networks



D(x) tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

(Goodfellow 2016)

$$\min_{G} \max_{D} V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

Generative Adversarial Imitation Learning

Jonathan Ho
Stanford University
hoj@cs.stanford.edu

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

NIPS 2016

Algorithm 1 Generative adversarial imitation learning

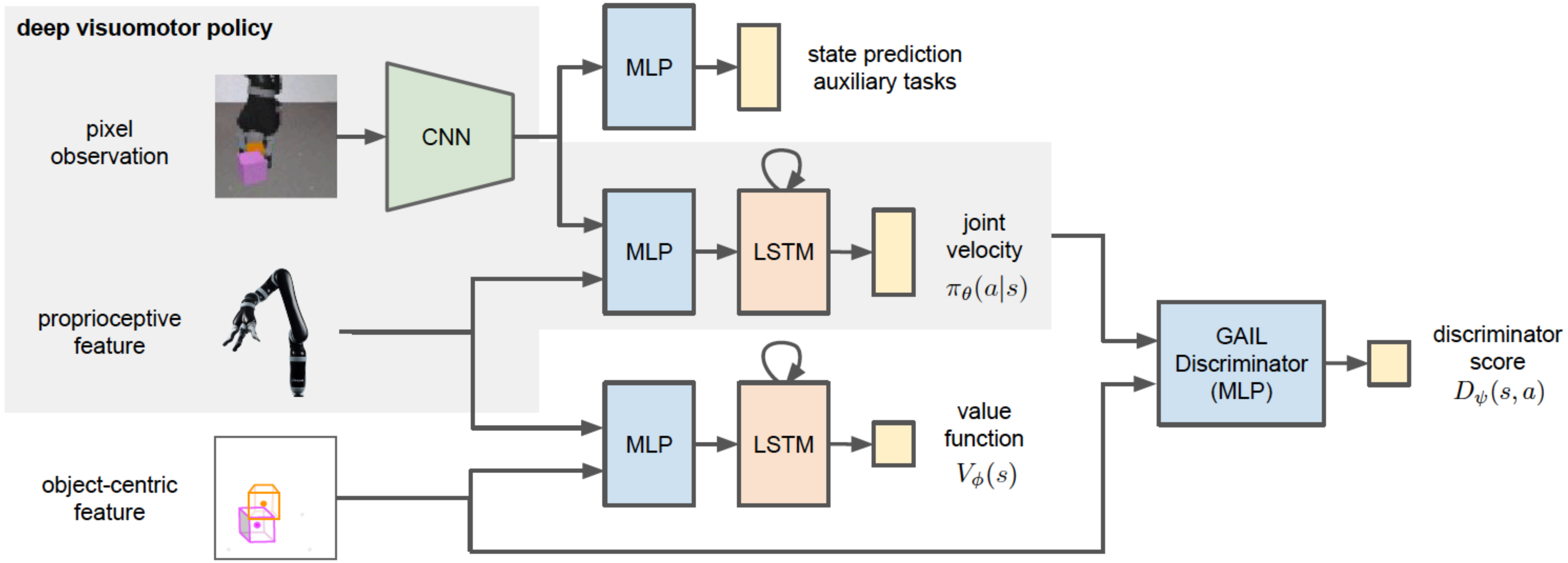
```
1: Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters $\theta_0, w_0$
2: for $i = 0, 1, 2, \ldots$ do
3:     Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4:     Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient

           $\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))]$     (17)

5:     Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$.
       Specifically, take a KL-constrained natural gradient step with

           $\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta)$,
           where $Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$     (18)

6: end for
```

Combining imitation and task rewards

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

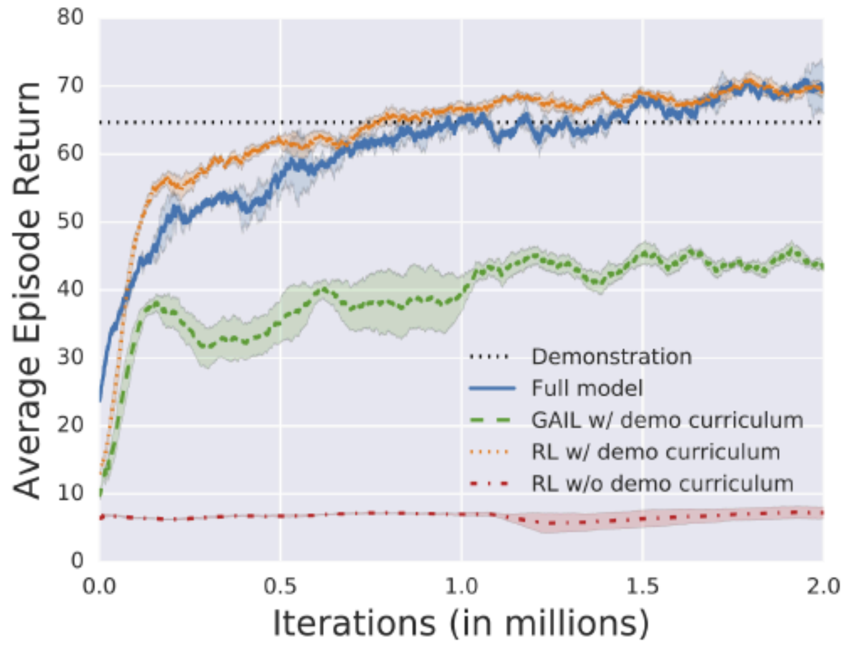
$$r(s, a) = \lambda r_{GAIL}(s, a) + (1 - \lambda) r_{task}(s, a), \quad \lambda \in [0,1] .$$

$$r_{GAIL}(s, a) = -\log(1 - D(s, a))$$

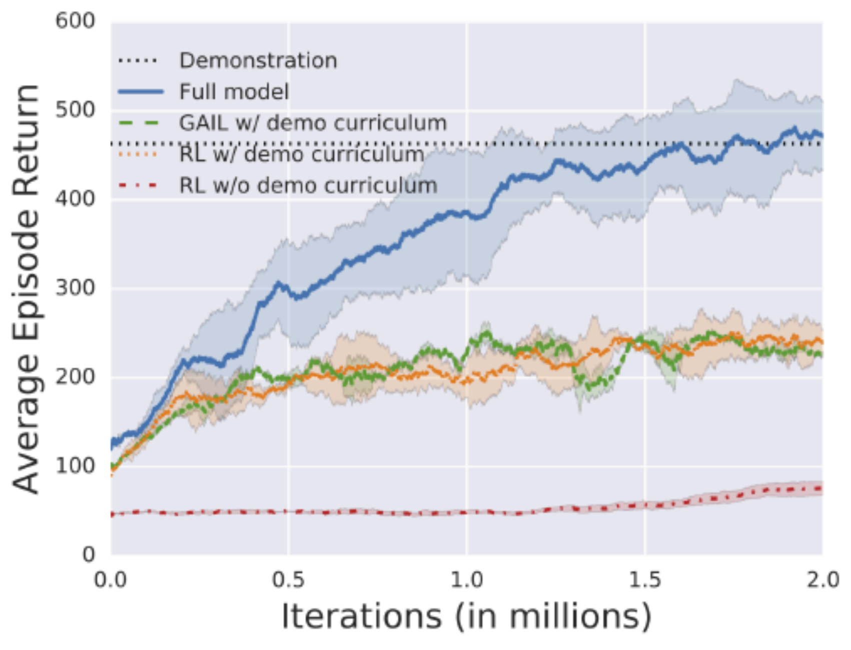


The discriminator takes as input (s,a) tuples and predicts whether they come from a demonstration or the trained policy.

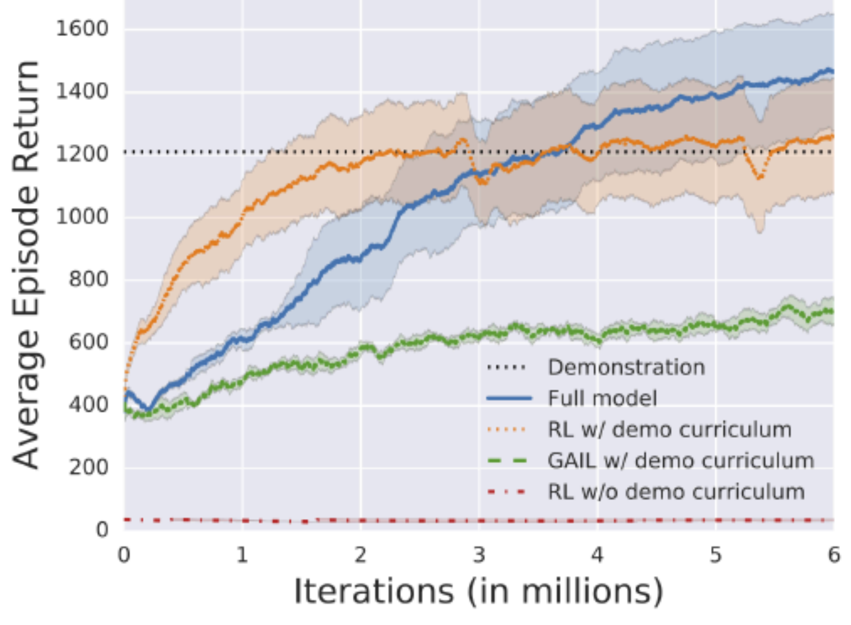
The state only contains the object state, not the gripper state, why?



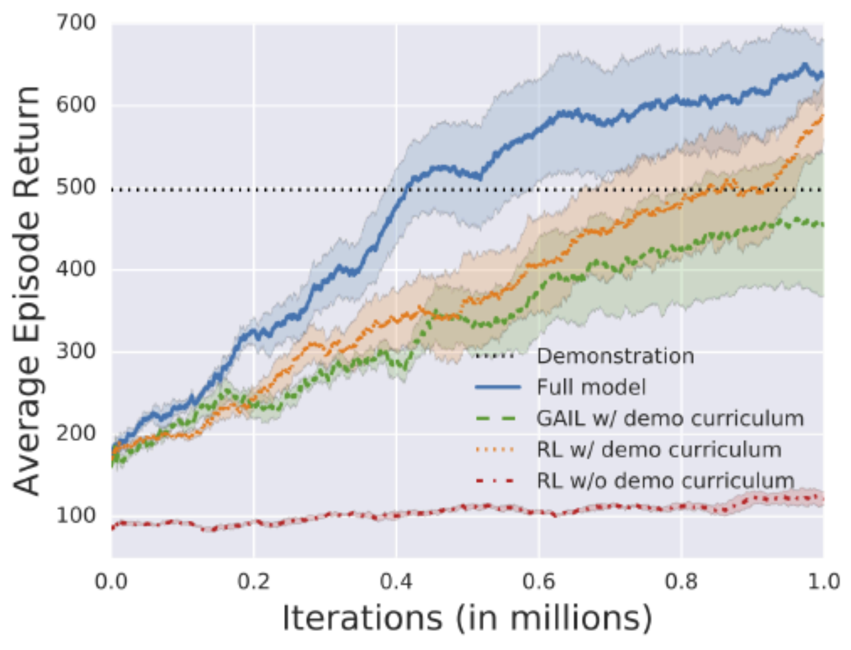
(a) Block lifting



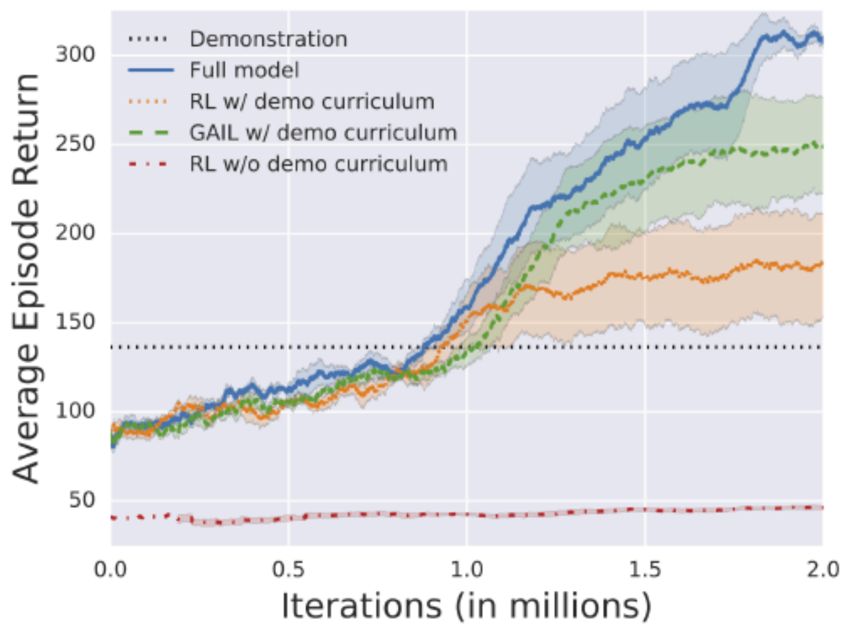
(b) Block stacking



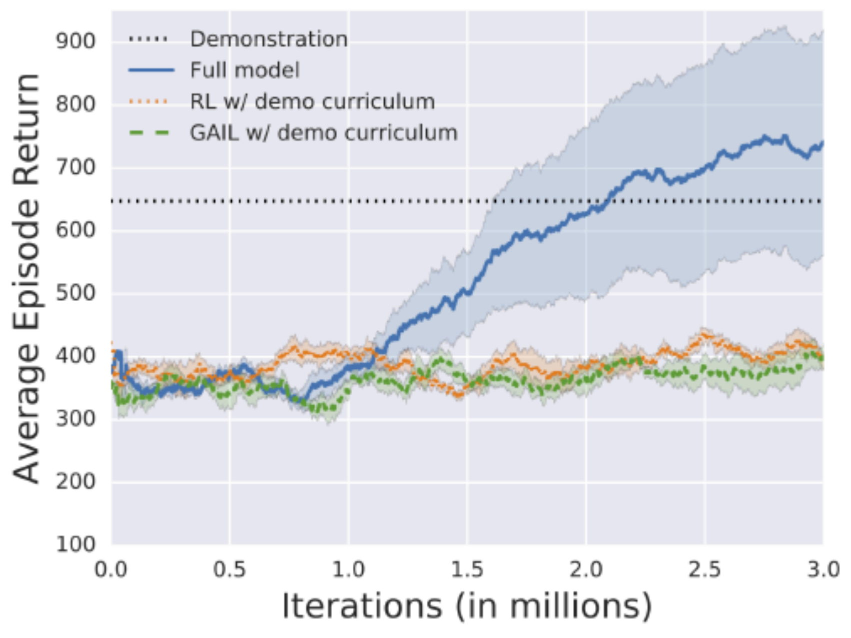
(c) Clearing table with blocks



(d) Clearing table with a box



(e) Pouring liquid



(f) Order fulfillment