Trustworthy AI Autonomy
M5-1 Trustworthy RL-Generalization

Ding Zhao
Assistant Professor
Carnegie Mellon University
Plan for today

• Working on real robots
  • Setting of robots, human supervision etc
  • Continuous state actions (DDPG/SAC)
  • Delay-aware RL
  • Non-stationary context-aware RL
  • Generalization and meta learning
Working with real robots

• Experiment design
  • Facilitating continuous operation
  Round-the-clock operation

Figure 3. Close-up of our robot grasping setup in our setup (left) and about 1000 visually and physically diverse training objects (right). Each robot consists of a KUKA LBR IIWA arm with a two-finger gripper and an over-the-shoulder RGB camera.
Working with real robots

• Experiment design

• Facilitating continuous operation
  Round-the-clock operation

• Non-stationarity due to environment changes

• a consistent performance drop of 5% in as little as 800 grasps executed on a single robot.

Working with real robots

- Experiment design
  - Facilitating continuous operation
    Round-the-clock operation
  - Non-stationarity due to environment changes
  - The human can reset the scene, stop the robot in unsafe situations, and simply restart and reset the robot on failures.

https://www.youtube.com/watch?v=FmMPHL3TcrE&t=85s
Safe Reinforcement Learning via Human Intervention

- Trial without Error

- May not be scalable for real world implementation

- However, people still do it e.g. self-driving companies

Figure 1: HIRL scheme. At (1) the human overseer (or Blocker imitating the human) can block/intercept unsafe actions $a$ and replace them with safe actions $a^*$. At (2) the overseer can deliver a negative reward penalty $r^*$ for the agent choosing an unsafe action.

Figure 2: In Pong (left) it’s a catastrophe if the agent (green paddle) enters the Catastrophe Zone. In Space Invaders (center), it’s a catastrophe if the agent shoots their defensive barriers (highlighted in pink box). In Road Runner (right), it’s a catastrophe if Road Runner touches the Coyote.

Recap: two ways to compute the optimal policy

- Parameterize the policy
  - Gradient ascent

\[ J(\theta, \mathcal{D}_{\pi_\theta}) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid \pi_\theta \right] \]

\[ \theta^* = \arg \max_\theta J(\theta, \mathcal{D}_{\pi_\theta}) \]

\[ \theta_{i+1} = \theta_i + \alpha \nabla_{\theta} J(\theta) \bigg|_{\theta=\theta_i}, \theta_i \rightarrow \theta^* \]

\[ a_t \sim \pi_{\theta_i}(\cdot \mid s_t) \]

- Parameterize the value function \( Q \)
  - Dynamic programming

\[ Q_{\phi^*}(s_t, a_t) = \mathbb{E} \left[ r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi^*}(s_{t+1}, a_{t+1}) \right] \]

\[ e^\pi_\phi = Q^\pi_\phi(s_t, a_t) - \mathbb{E} \left[ r(s_t, a_t) + \gamma \max_{a_{t+1}} Q^\pi_\phi(s_{t+1}, a_{t+1}) \right] \]

\[ L(\phi, \pi) = \mathbb{E} \left[ \frac{1}{2} e^\pi_\phi \right], (\phi^*, \pi^*) = \arg \min_{\phi, \pi} L(\phi, \pi) \]

- With the “greedy method”, i.e., \( \pi(a_t \mid s_t) = \max_a Q_{\phi}(s_t, a) \)

- \( \phi_i + 1 = \phi_i - \alpha \nabla_{\phi} L(\phi) \bigg|_{\phi=\phi_i}, \phi_i \rightarrow \phi^*, \pi_i \rightarrow \pi^* \)
Handle continuous action space

- Issue of DQN: With continuous action space, we need to solve $a_t = \arg \max_a Q(s_t, a)$ every timestep in Q-learning.

- When there are a finite number of discrete actions, the max poses no problem, because we can just compute the Q-values for each action separately and directly compare them. (This also immediately gives us the action which maximizes the Q-value.)

- But when the action space is continuous, we can’t exhaustively evaluate the space, and solving the optimization problem is highly non-trivial. Using a normal optimization algorithm would make calculating $\max_a Q^*(s, a)$ a painfully expensive subroutine. And since it would need to be run every time the agent wants to take an action in the environment, this is unacceptable.

- DDPG does not have this problem as it directly approximate $\arg \max_a Q^*(s, a)$ with $\mu_\theta(s)$. No optimization is needed.
Recap: DQN-3.0 algorithm

Randomize actions and training data

1. Take the \( \epsilon \)-greedy method:
   \[
   a_t = \max_a Q_{\phi_i}(s_t, a) \text{ with probability } 1 - \epsilon, \text{ otherwise, choose a random action}
   \]

   observe a dataset \( \{(s_t, a_t, s_{t+1}, r_t)\} \) and add it to \( \mathcal{D} \)

   1. Randomly sample a mini batch from \( \mathcal{D} \)
   2. Calculate Bellman backup for this batch
      \[
      y_t = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1})
      \]

      1. Update the Q function
         \[
         \phi \leftarrow \phi - \alpha \sum_i (\nabla_{\phi} Q_{\phi}(s_t, a_t))(Q_{\phi}(s_t, a_t) - y_t)
         \]
   3. Update Q function:
      Moving average: \( \phi_{i+1} = \rho \phi_i + (1 - \rho)\phi \), e.g. \( \rho = 0.999 \)

The only changes DDPG made

1. Use a deterministic policy to calculate
   \[
   \text{arg max: } a_t = \mu_\theta(s_t) = \text{arg max}_a Q(s_t, a)
   \]

Q function:
\[
\max_{a_{t+1}} Q_{\phi_i}(s_{t+1}, a_{t+1}) \rightarrow Q_{\phi_i}(s_{t+1}, \mu_\theta(s))
\]

2. \( \theta_{i+1} = \theta_i + \nabla_\theta J(\theta) |_{\theta=\theta_i} \)
   
   \[
   \nabla_\theta J(\theta) = \nabla_\theta Q(s_t, a_t) = \nabla_\theta Q(s_t, \mu_\theta(s_t))
   \]

   \[
   = \mathbb{E}_{s_t \sim \mathcal{D}}[\nabla_\theta \mu_\theta(a_t | s_t) \nabla_a Q^\mu(s_t, a) |_{a=\mu_\theta(s_t)}]
   \]

Also called low pass filter/Polyak averaging in literature
DDPG algorithms

Randomize actions and training data

1. Take the \( a(s_t) = \mu_\theta(s_t) \)
   observe a dataset \( \{(s_t, a_t, s_{t+1}, r_t)\} \) and add it to \( \mathcal{D} \)

1. Randomly sample a mini batch from \( \mathcal{D} \)

2. Calculate Bellman backup for this batch
   \[
   y_t = r(s_t, a_t) + \gamma Q_{\phi_i}(s_{t+1}, \mu_\theta(s_t))
   \]

1. Update the Q function
   \[
   \phi \leftarrow \phi - \alpha \sum_t (\nabla_\phi Q_{\phi_i}(s_t, a_t)) (Q_\phi(s_t, a_t) - y_t)
   \]

2. Update the policy
   \[
   \theta \leftarrow \theta + \beta \sum_t [\nabla_\theta \mu_\theta(a | s_t) \nabla_a Q_\phi^\mu(s_t, a) | a = \mu_\theta(s)]
   \]

3. Update Q function and policy
   \[
   \phi_{i+1} = \rho \phi_i + (1 - \rho) \phi, \quad \theta_{i+1} = \rho \theta_i + (1 - \rho) \theta,
   \]

Two tricks to enhance performance

1. Increase stability: use a larger delay to stabilize the learning of Q function
   - use \( \phi_{i-k} \) instead of \( \phi_i \) to compute \( y_t \)

Also called low pass filter/Polyak averaging in literature
DDPG algorithms

Randomize actions and training data

1. Take the \( a(s_t) = \mu_\theta(s_t) \)
   observe a dataset \( \{(s_t, a_t, s_{t+1}, r_t)\} \) and add it to \( \mathcal{D} \)

1. Randomly sample a mini batch from \( \mathcal{D} \)

2. Calculate Bellman backup for this batch
   \[
   y_t = r(s_t, a_t) + \gamma Q_{\phi_i}(s_{t+1}, \mu_\theta(s_t))
   \]

1. Update the Q function
   \[
   \phi \leftarrow \phi - \alpha \sum_t \left( \nabla_\phi Q_{\phi}(s_t, a_t) \right) \left( Q_{\phi}(s_t, a_t) - y_t \right)
   \]

2. Update the policy
   \[
   \theta \leftarrow \theta + \beta \sum_t \left[ \nabla_\theta \mu_\theta(a | s_t) \nabla_a Q_{\mu_i}^\mu(s_t, a) \right]_{a=\mu_\theta(s)}
   \]

3. Update Q function and policy
   \[
   \phi_{i+1} = \rho \phi_i + (1 - \rho) \phi, \quad \theta_{i+1} = \rho \theta_i + (1 - \rho) \theta,
   \]

Also called low pass filter/Polyak averaging in literature

Two tricks to enhance performance

1. Increase stability: use a larger delay to stabilize the learning of Q function
   - use \( \phi_{i-k} \) instead of \( \phi_i \) to compute \( y_t \)

2. Use add a Gaussian noise to the deterministic policy to make the data more i.i.d and covers more scenarios
   - \( a(s_t) = \mu_\theta(s_t) + \mathcal{N}(0, \sigma^2) \)
   - Two approaches: TD3 and SAC
Improve the DDPG with TD3

• Overestimation problem
  • While DDPG can achieve great performance sometimes, it is frequently brittle with respect to hyperparameters and other kinds of tuning. A common failure mode for DDPG is that the learned Q-function begins to dramatically overestimate Q-values (spikes), which then leads to the policy breaking. **Twin Delayed DDPG (TD3)** is an algorithm that addresses this issue by introducing three critical tricks:

1. Clipped Double-Q Learning. TD3 learns two Q-functions instead of one (hence “twin”), and uses the smaller of the two Q-values to form the Bellman error loss functions. Particularly,
   \[
   \phi^{(i)} \leftarrow \phi^{(i)} - \alpha \sum_t (\nabla_{\phi^{(i)}} Q_{\phi^{(i)}}(s_t, a_t))(Q_{\phi^{(i)}}(s_t, a_t) - y_t), \quad i = \{1, 2\}.
   \]
   \[
   y_t = r(s_t, a_t) + \gamma \min_i Q_{\phi^{(i)}}(s_{t+1}, \mu_\theta(s_{t+1}))
   \]

2. “Delayed” Policy Updates.: use \(\phi_{i-k}\) instead of \(\phi_i\) to compute \(y_t\). Usually \(k = 1\) or 2.

3. Target Policy Smoothing: Use add a Gaussian noise to the deterministic policy
   \[
   a(s_t) = \mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi, \quad \xi \sim \mathcal{N}(0, I).
   \]
   Usually add clips to make the action range feasible. Or normalize it with \(a_t = \tanh \left( \mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi \right)\)
SAC

• Idea 2: Increase the entropy of the policy distribution $\pi(a_t | s_t)$ to encourage exploration
  
  • Use a stochastic policy: $a_t \sim \pi_\theta(a_t | s_t) = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi)$, $\xi \sim \mathcal{N}(0, I)$.
  
  • Entropy: $H(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\!-\!log \pi_\theta(a_t | s_t)]$
    
    • Entropy is a quantity which, roughly speaking, says how random a random variable is. If a coin is weighted so that it almost always comes up heads, it has low entropy; if it’s evenly weighted and has a half chance of either outcome, it has high entropy.
    
    • Reward: $r(s_t, a_t) \Rightarrow r(s_t, a_t) + \alpha H(\pi(a_t | s_t))$

  • This has a close connection to exploration-exploitation trade-off: increasing entropy results in more exploration, which can accelerate learning later on.

  • The Bellman Equation becomes
    
    $Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[ \sum_{t' \geq t} \gamma^{t'-t} r_{t'} + \alpha \sum_{t' \geq t+1} \gamma^{t'-t} H(\pi(a_t | s_t)) \big| s_t, a_t \right] = \mathbb{E}_\pi \left[ r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) + \alpha H(\pi(a_t)) \right]$

    • The remaining is similar to TD3.
SAC algorithms

Randomize actions and training data

1. Take the $a_t = \tanh \left( \mu_\theta(s_t) + \sigma_\theta(s_t) \odot \xi \right)$, $\xi \sim \mathcal{N}(0, I)$
   observe a dataset $\{(s_t, a_t, s_{t+1}, r_t)\}$ and add it to $\mathcal{D}$

1. Randomly sample a mini batch from $\mathcal{D}$

2. Calculate Bellman backup for this batch
   $y_t = r(s_t, a_t) + \gamma (\min_{i=1,2} Q_{\phi(i)}(s_{t+1}, a_{t+1}) + \alpha H(\pi(a_t | s_t)))$, $a_{t+1} \sim \pi_\theta(\cdot | s_t)$

1. Iteratively calculate $\phi$ from $\phi_i$
   $\phi^{(i)} \leftarrow \phi^{(i)} - \alpha \sum_t (\nabla_{\phi^{(i)}} Q_{\phi^{(i)}}(s_t, a_t))(Q_{\phi^{(i)}}(s_t, a_t) - y_t)$, $i = \{1, 2\}$

2. Iteratively calculate $\theta$ from $\theta_i$
   $\theta \leftarrow \theta + \beta \nabla_\theta \sum_t [ (\min_{i=1,2} Q_{\phi^{(i)}}(s_t, \pi_\theta(a_t | s_t)) + \alpha H(\pi(a_t | s_t))]$

3. Update Q function and policy
   $\phi_{i+1} = \rho \phi_{i} + (1 - \rho) \phi$, $\theta_{i+1} = \rho \theta_{i} + (1 - \rho) \theta$

 Also called low pass filter/Polyak averaging in literature
Recap: popular RL algorithms

RL Algorithms

Model-Free RL
- Policy Optimization
  - Policy Gradient
    - A2C / A3C
    - PPO
    - TRPO
  - TD3
  - SAC
- Q-Learning
  - DDPG
  - DQN
  - C51
  - QR-DQN
  - HER

Model-Based RL
- Learn the Model
  - World Models
    - I2A
    - MBMF
  - MBVE
- Given the Model
  - AlphaZero

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Agent type</th>
<th>Policy</th>
<th>Policy type</th>
<th>Monte Carlo (MC) or Temporal difference (TD)</th>
<th>Action space</th>
<th>State space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular Q-learning (= SARSA max) Q-learning lambda</td>
<td>Value-based</td>
<td>Off-policy</td>
<td>Pseudo-deterministic (epsilon greedy)</td>
<td>TD</td>
<td>Discrete only</td>
<td>Discrete only</td>
</tr>
<tr>
<td>SARSA</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Pseudo-deterministic (epsilon greedy)</td>
<td>TD</td>
<td>Discrete only</td>
<td>Discrete only</td>
</tr>
<tr>
<td>DQN</td>
<td>Value-based</td>
<td>Off-policy</td>
<td>Pseudo-deterministic (epsilon greedy)</td>
<td>Discrete only</td>
<td>Discrete or continuous</td>
<td></td>
</tr>
<tr>
<td>N-step DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Double DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Noisy DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Prioritized Replay DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Dueling DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Categorical DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Distributed DQN (C51)</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>NAF + continuous DQN</td>
<td>Value-based</td>
<td>On-policy</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>CEM</td>
<td>Policy-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>REINFORCE (Vanilla policy gradient)</td>
<td>Policy-based</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Continuous</td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>Policy gradient softmax</td>
<td>Policy-based</td>
<td>Stochastic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Natural Policy Gradient</td>
<td>Policy-based</td>
<td>Stochastic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRPO</td>
<td>Actor-critic</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Discrete or continuous</td>
<td>Discrete or continuous</td>
<td>Discrete or continuous</td>
</tr>
<tr>
<td>PPO</td>
<td>Actor-critic</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>Discrete or continuous</td>
<td>Discrete or continuous</td>
<td>Discrete or continuous</td>
</tr>
<tr>
<td>Distributed PPO</td>
<td>Actor-critic</td>
<td>Continuous</td>
<td></td>
<td></td>
<td>Continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>A3C / A3C</td>
<td>Actor-critic</td>
<td>On-policy</td>
<td>Stochastic</td>
<td>TD</td>
<td>Discrete or continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>DDPG</td>
<td>Actor-critic</td>
<td>Off-policy</td>
<td>Deterministic</td>
<td>Continuous only</td>
<td>Continuous or Continuous</td>
<td>Continuous or Continuous</td>
</tr>
<tr>
<td>TD3</td>
<td>Actor-critic</td>
<td>Continuous</td>
<td></td>
<td></td>
<td>Continuous</td>
<td>Discrete or continuous</td>
</tr>
<tr>
<td>D3PG</td>
<td>Actor-critic</td>
<td>Continuous</td>
<td></td>
<td></td>
<td>Continuous</td>
<td>Discrete or continuous</td>
</tr>
<tr>
<td>SAC</td>
<td>Actor-critic</td>
<td>Continuous</td>
<td></td>
<td></td>
<td>Continuous</td>
<td>Discrete or continuous</td>
</tr>
<tr>
<td>ACER</td>
<td>Actor-critic</td>
<td>Continuous</td>
<td></td>
<td></td>
<td>Discrete or Continuous</td>
<td>Discrete or Continuous</td>
</tr>
<tr>
<td>ACKTR</td>
<td>Actor-critic</td>
<td>Continuous</td>
<td></td>
<td></td>
<td>Discrete or Continuous</td>
<td>Discrete or Continuous</td>
</tr>
</tbody>
</table>

https://www.datamachinist.com/reinforcement-learning/part-2-reinforcement-learning-algorithms/
Domain randomization

• Sim-to-real: Dactyl was trained entirely in simulation and transfers its knowledge to reality

https://openai.com/blog/learning-dexterity/
A Distributed workers collect experience on randomized environments at large scale.

B We train a control policy using reinforcement learning. It chooses the next action based on fingertip positions and the object pose.

C We train a convolutional neural network to predict the object pose given three simulated camera images.

https://openai.com/blog/learning-dexterity/
We combine the pose estimation network and the control policy to transfer to the real world.
Learning progress with and without randomizations over years of simulated experience.

https://openai.com/blog/learning-dexterity/
Delay matters in real-world applications

Delays are prevalent in the real world. E.g., control freq of autonomous vehicles > 10 Hz, While the hydraulic brake system delay > 0.4 seconds.

Control of the delayed system

- Delays may not only **degrade the performance** of the agent but also **induce instability** to the dynamic systems. (Gu & Niculescu, 2003)

- The control community has proposed several methods to deal with delayed tasks. The most general approach is the Smith predictor (Astrom et al., 1994):

![Diagram of Smith predictor](image)

Basic idea: feedback control by predicting the future states.

Cons: requires a system model and is sensitive to model errors.
Delayed Markov Decision Process (DMDP)

Experiment #1: the influence of delay

- **SAC (n=0)**: soft actor-critic, SOTA model-free algorithm
- **PETS (n=0)**: The original PETS algorithm in undelayed environment.
- **PETS (n=1)**: The original PETS algorithm in the 1-step delayed environment but ignoring the action delay.
- **W-DATS (n=1)**: PETS algorithm in the delayed environment that wastefully learns the whole dynamics of DMDPs.
- **DATS (n=1)**: the proposed method introduced in Algorithm 2.

Experiment #2: model-based vs model-free

- DATS has stable performance when the delay step increases.
- RTAC degrades significantly as the delay step increases.

DATS: the proposed model-based method.

RTAC: model-free method, the updated SAC. (Ramstedt & Pal, 2019)

Robust Adversarial Reinforcement Learning (RARL)

- Main agent: *protagonist*
- Environment agent: *adversary*
  - Consider *env* as the adversary

**Key idea:**
- Adversarial training as a two-player zero-sum game
  - The protagonist maximizes $\mathbb{E}_{\tau \sim \theta, \psi} [R(\tau)]$
  - The adversary minimizes $\mathbb{E}_{\tau \sim \theta, \psi} [R(\tau)]$
  - Use gradient-descent-ascent-based algorithm to train the protagonist and adversary

\[ \tau \text{ is the trajectories sampled using policy } \theta \text{ and } \psi \]
\[ R(\tau) \text{ is the return of the protagonist} \]

[1] Pinto et al., 2017
Limitations of existing RARL Methods

1. Unstable training

[Dennis et al., 2020]

2. Zero-sum

Existing works: Zero-sum

Encourages the adversary to generate extremely difficult, unsolvable environments

The protagonist may learn an overly conservative strategy or even not be able to learn

Unstable training

Limitations of existing RARL methods

1. [Dennis et al., 2020]

2. Existing work: gradient-descent-ascent

Each agent naively takes partial derivative of its objective w.r.t its parameters

Unstable Training
Unstable training
Robust Reinforcement Learning as a Stackelberg Game via Adaptively-Regularized Adversarial Training

Existing Work

- Zero-sum Adversary
- Gradient-Descent-Ascent

Unsolvable environments

Our Method

- Regularized Adversary
- Stackelberg Game

Unstable training

Encourage Challenging but Solvable Environments

- No-Adv: the protagonist is not aware of the danger
- RARL: the adversary generates unsolvable environments
- RRL-Stack: the adversary generates challenging but solvable environments

![Protagonist](yellow) ![Adversary](yellow)

No-Adv

Improve Training Stability and Robustness

- No-Adv: is stable; not robust
- RARL: diverges at around 20 iterations; not robust
- RRL-Stack: keeps learning robust policies; robust

How to deal with a changing environment?

Safe reinforcement learning for non-stationary environments

What if the contexts are very different?
Worth reading

• Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, Sergey Levine, Sergay, “How to Train Your Robot with Deep Reinforcement Learning – Lessons We’ve Learned,” Journal of Robotics Research (IJRR), 2021