Robot Learning

Deep Reinforcement Learning: Tutorial

Glen Berseth

Université de Montréal and Mila Québec AI Institute

October 15, 2025

Outline

Why Robot Learning With DeepRL?

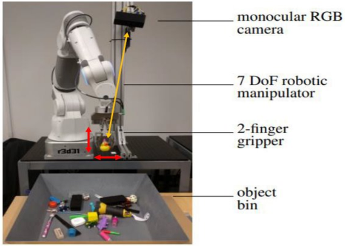
Supervised Learning vs Reinforcement Learning

Model-Based Reinforcement Learning

Model-Free Reinforcement Learning

Creating an RL Environment for a Robot

Why Robot Learning With DeepRL?

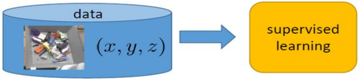


Option 1:

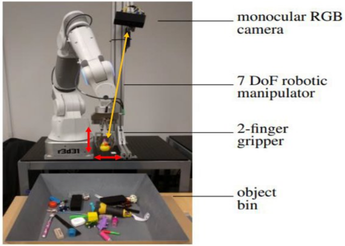
Understand the problem, design a solution

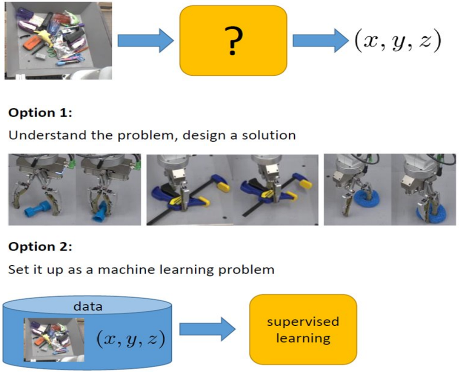
Option 2:

Set it up as a machine learning problem



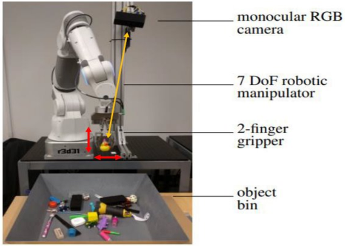
Why Robot Learning With DeepRL?





- There are many situations where traditional models are challenged
  - Large state spaces
  - Non-linear dynamics
  - Discontinuous contacts

Why Robot Learning With DeepRL?

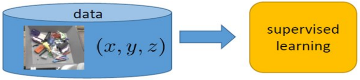


Option 1:

Understand the problem, design a solution

Option 2:

Set it up as a machine learning problem



- There are many situations where traditional models are challenged
  - Large state spaces
  - Non-linear dynamics
  - Discontinuous contacts

What Problem is DeepRL Solving?

No feature engineering!

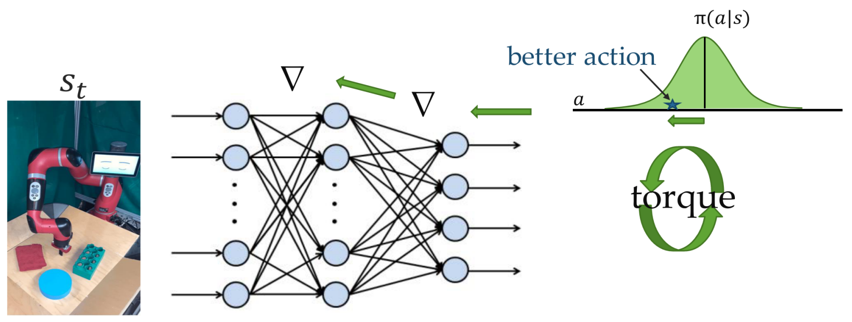


Figure: Deep Learning and Reinforcement Learning

- The perception and planning problem in a more general way.

What Problem is DeepRL Solving?

Sensor Motor Loop

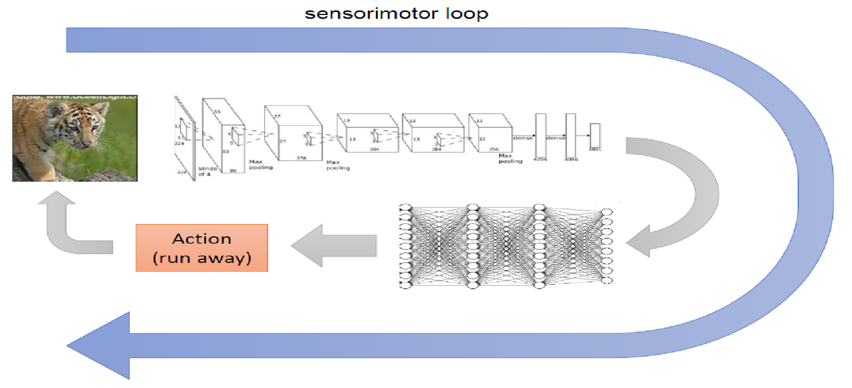


Figure: Sensory motor loop

- RL agents collect their own data to solve a task
  - No need for expert data

Supervised Learning vs Reinforcement Learning

Supervised learning

- given $\mathcal{D} = \{x_i, y_i\}$
  - learn to predict y_i given x_i, $y \leftarrow f(x)$
- Assumptions in supervised learning
  - Data is Independent and Identically Distributed (IID)
    - This is rarely the case in the real world
  - True optimal action y is known
- Example:
  - $L(\theta) = ||f(x|\theta) - y||^2$

Supervised Learning vs Reinforcement Learning

Supervised learning

- given $\mathcal{D} = \{x_i, y_i\}$
  - learn to predict y_i given x_i, $y \leftarrow f(x)$
- Assumptions in supervised learning
  - Data is Independent and Identically Distributed (IID)
    - This is rarely the case in the real world
  - True optimal action y is known
- Example:
  - $L(\theta) = ||f(x|\theta) - y||^2$

Reinforcement Learning

- Previous outputs influence future inputs
  - Data is not IID
- Optimal action y is known
  - Instead, we have a scalar reward function
- **reward** function
  - $r \leftarrow R(s, a)$
  - **weighted regression**
- Example:
  - $L(\theta) = ||f(s|\theta) - a||^2 R(s, a)$

What is Reinforcement Learning

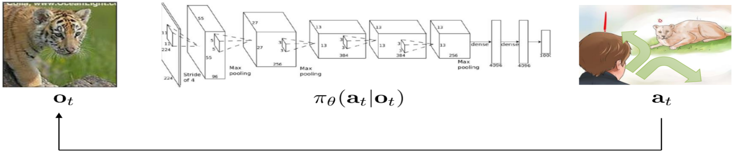


Figure: First terms

- a_t - Action
- $\mathbf{a}_t$ - Continuous action
- $\mathbf{s}_t$ - State
- $\mathbf{o}_t$ - Observation

What is Reinforcement Learning

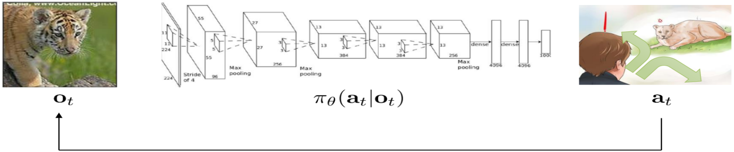


Figure: First terms

- a_t - Action
- $\mathbf{a}_t$ - Continuous action
- $\mathbf{s}_t$ - State
- $\mathbf{o}_t$ - Observation

- $\pi(\mathbf{a}_t|\mathbf{o}_t, \theta)$ policy
- $\pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$ fully observed policy

What is Reinforcement Learning

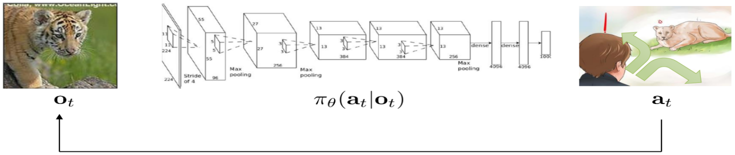


Figure: First terms

- a_t - Action
- $\mathbf{a}_t$ - Continuous action
- $\mathbf{s}_t$ - State
- $\mathbf{o}_t$ - Observation
- $\pi(\mathbf{a}_t|\mathbf{o}_t, \theta)$ policy
- $\pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$ fully observed policy

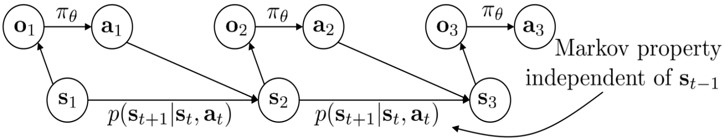


Figure: Markov property

Reinforcement Learning Optimization

$\arg\max_{\theta^*} \mathbb{E}_{\tau\sim p(\tau|\theta)}\left[\sum_t^T r(\mathbf{s}_t, \mathbf{a}_t)\right]$

- Finite horizon case

$\arg\max_{\theta^*} \mathbb{E}_{\tau\sim p(\tau|\theta)}\left[\sum r(s, a)\right]$

- Infinite horizon case

Reinforcement Learning Optimization

$\arg\max_{\theta^*} \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t^T r(\mathbf{s}_t, \mathbf{a}_t) \right]$

- Finite horizon case

$\arg\max_{\theta^*} \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum r(s, a) \right]$

- Infinite horizon case

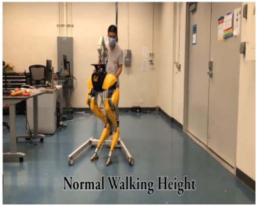
- Reinforcement Learning uses Expectations

Figure: Discontinuous Rewards

- $r(s, a)$ - **not smooth**
- $\pi(a = fall|s, \theta)$
- $\mathbb{E}_{\pi(\cdot|s,\theta)}[r(s, a)]$ - smooth wrt θ

Deep Reinforcement Learning Success Stories

Image courtesy: The Guardian



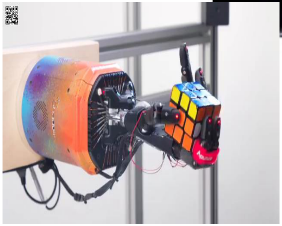


Figure: Success Stories
(Silver *et al.*, 2016; OpenAI *et al.*, 2019; Berner *et al.*, 2019; Li *et al.*, 2021)

Reinforcement Learning Objective

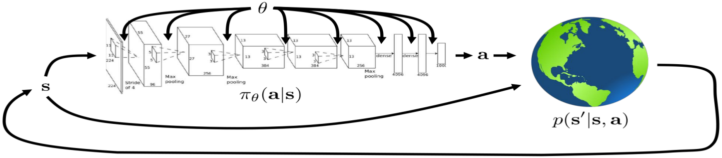


Figure: Reinforcement Learning Environment

Reinforcement Learning Objective

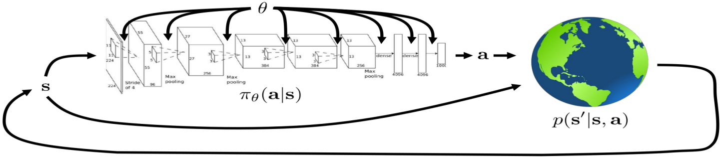


Figure: Reinforcement Learning Environment

- Distribution over trajectories $p(\tau|\theta)$ using chain rule of probability

$$\underbrace{p(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T|\theta)}_{p(\tau|\theta)} = \underbrace{p(\mathbf{s}_1)}_{\text{unknown}} \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t,\theta)\, \underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}_{\text{unknown}} \tag{1}$$

Reinforcement Learning Objective

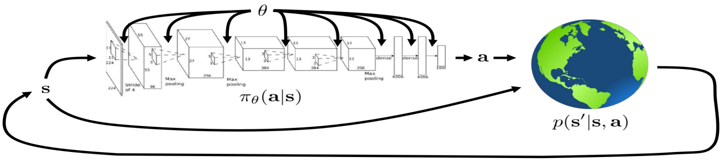


Figure: Reinforcement Learning Environment

- Distribution over trajectories $p(\tau|\theta)$ using chain rule of probability

$$\underbrace{p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T|\theta)}_{p(\tau|\theta)} = \underbrace{p(\mathbf{s}_1)}_{\text{unknown}} \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t, \theta) \underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{unknown}} \tag{1}$$

- RL objective is over this distribution

$$\arg\max_{\theta^*} \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{2}$$

Basic Reinforcement Learning Loop: (1) Collect Data

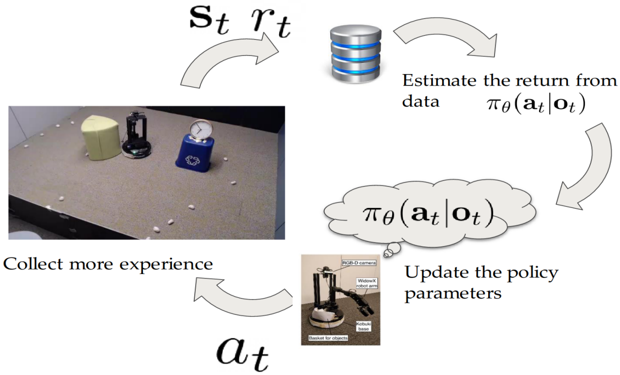


Figure: Sensory motor loop

Basic Reinforcement Learning Loop: (1) Collect Data

Collect Data

```
import gym
env = gym.make("LunarLander-v2") ## Create an instance of the control envi
observation, info = env.reset(seed=42, return_info=True) ## Reset the envi
buff = [] ## Array to store experience
for _ in range(1000):
    env.render()  ## Render the environment if desired
    action = policy(observation)  # User-defined policy function
    next_observation, reward, done, info = env.step(action) ## Take a step i
    buff.append([observation, action, reward, next_observation])
    observation = next_observation
    if done:
        observation, info = env.reset(return_info=True) ## Reset if the robot

env.close()
```

Basic Reinforcement Learning Loop: (2) Estimate Return/Score

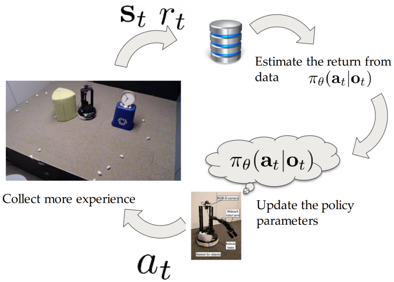


Figure: Sensory motor loop

Basic Reinforcement Learning Loop: (2) Estimate Return/Score

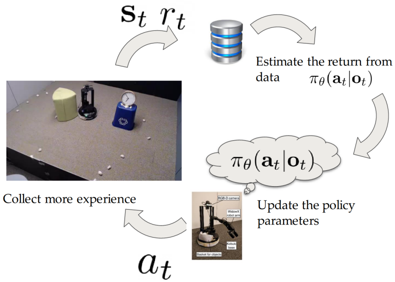


Figure: Sensory motor loop

Estimate the return for θ

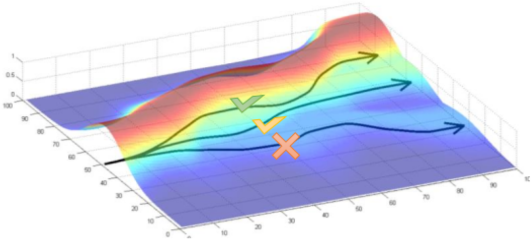


Figure: Policy Gradient

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \quad (3)$$

Examples: Reinforce (Williams, 1992; Sutton *et al.*, 2000)

Basic Reinforcement Learning Loop: (2) Estimate Return/Score

Estimate Return

```
# create list at each index (t') is gamma^(t') * r_{t'}
discounted_rewards = discounts * rewards

# scalar: sum_{t'=0}^T gamma^(t') * r_{t'}
sum_of_discounted_rew = sum(discounted_rewards)

# list where each entry t contains the same thing
# it contains sum_{t'=0}^T gamma^t' r_{t'}
discounted_returns = np.ones_like(rewards) * sum_of_discounted_rew

# For each (s_t, a_t), discounted sum of rewards over trajectory
# Aka: value of (s_t, a_t) = sum_{t'=0}^T gamma^t' r_{t'}
q_values = np.concat([self._discounted_return(r) for r in rews_list])
```

Basic Reinforcement Learning Loop: (3) Update The Policy

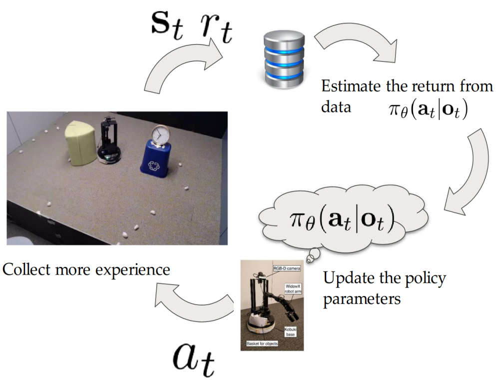


Figure: Sensory motor loop

Update the policy

Basic Reinforcement Learning Loop: (3) Update The Policy

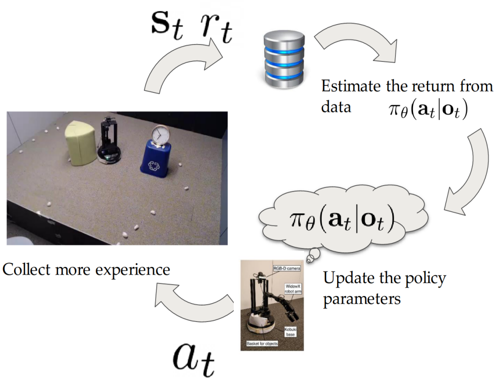


Figure: Sensory motor loop

Update the policy

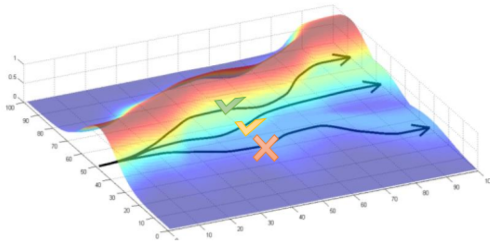


Figure: Policy Gradient

- $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- α is the learning rate

Model-Based Reinforcement Learning

- Distribution over trajectories $p(\tau|\theta)$ using chain rule of probability

$$\underbrace{p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T|\theta)}_{p(\tau|\theta)} = \underbrace{p(\mathbf{s}_1)}_{\text{unknown}} \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t, \theta)\, \underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{learn this}} \tag{4}$$

Start by training a model.

Must train a Model

- Model-Based Reinforcement Learning (MBRL)
- Why learn a model?
  - For most problems, the dynamics are unknown
  - If we have $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$ we can plan
- Then all we need to do is learn $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$, that should be *easy*.

Must train a Model

- MBRL
- Why learn a model?
  - For most problems, the dynamics are unknown
  - If we have $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$ we can plan
- Then all we need to do is learn $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$, that should be *easy*.

Basic MBRL

1. Collect experience $< \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t > \in \mathcal{D}_{\text{train}}$ from the environment with $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$
2. Train θ to minimize $\sum_i ||f(\mathbf{s}_t, \mathbf{a}_t, \theta) - \mathbf{s}_{t+1}||$
3. Use $f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$ to plan high reward trajectories

(Wang *et al.*, 2018)

How Well Does Basic MBRL Work?

How Well Does Basic MBRL Work?

- Not that well, why?

How Well Does Basic MBRL Work?

- Not that well, why?

Basic MBRL

1: Collect experience $< \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t > \in \mathcal{D}_{\text{train}}$ from the environment with $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t)$
2: Train θ to minimize $\sum_i ||f(\mathbf{s}_t, \mathbf{a}_t, \theta) - \mathbf{s}_{t+1}||$
3: Use $f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$ to plan high value trajectories

- Goal: Move higher
- But: $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t) \neq \pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$

- Problem grows with model complexity

How to train a forward model

- How to reduce $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t) \neq \pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$
- Ideas?

How to train a forward model

- How to reduce $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t) \neq \pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$
- Ideas?
- Need more on policy data [Dagger](Ross *et al.*, 2011)

How to train a forward model

- How to reduce $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t) \neq \pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$
- Ideas?
- Need more on policy data [Dagger](Ross *et al.*, 2011)

OnPolicy MBRL

1: Collect experience $< \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t >\in \mathcal{D}_{\text{train}}$ from the environment with $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t)$
2: **while** true **do**
3: Train θ to minimize $\sum_i ||f(\mathbf{s}_t, \mathbf{a}_t, \theta) - \mathbf{s}_{t+1}||$
4: Use $f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$ to plan high value trajectories
5: Collect experience $< \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t >\in \mathcal{D}_{\text{train}}$ from the environment with $f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$
6: **end while**

(Deisenroth and Rasmussen, 2011; Chua *et al.*, 2018; Hafner *et al.*, 2019)

How to train a forward model

- How to reduce $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t) \neq \pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$
- Ideas?
- Need more on policy data [Dagger](Ross *et al.*, 2011)

OnPolicy MBRL

1: Collect experience $< \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t > \in \mathcal{D}_{\text{train}}$ from the environment with $\pi_{\text{rand}}(\mathbf{a}_t|\mathbf{s}_t)$
2: **while** true **do**
3: Train θ to minimize $\sum_i ||f(\mathbf{s}_t, \mathbf{a}_t, \theta) - \mathbf{s}_{t+1}||$
4: Use $f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$ to plan high value trajectories
5: Collect experience $< \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t > \in \mathcal{D}_{\text{train}}$ from the environment with $f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$
6: **end while**

(Deisenroth and Rasmussen, 2011; Chua *et al.*, 2018; Hafner *et al.*, 2019)

- What is wrong with this algorithm?
  - Hint: What objective is it optimizing?

Model-Free Reinforcement Learning

- Distribution over trajectories $p(\tau|\theta)$ using chain rule of probability

$$\underbrace{p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T|\theta)}_{p(\tau|\theta)} = \underbrace{p(\mathbf{s}_1)}_{\text{Unknown}} \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t, \theta) \underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{Now unknown}} \tag{5}$$

- RL objective is over this distribution

$$\arg\max_{\theta^*} \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{6}$$

- **MBRL is not optimizing for the RL objective.**
  - (Joseph *et al.*, 2013; Farahmand *et al.*, 2017; Janner *et al.*, 2019; Grimm *et al.*, 2020; Lambert *et al.*, 2020; Nikishin *et al.*, 2022)

The Policy Gradient

$$\theta^* = \arg\max_\theta \underbrace{\mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]}_{J(\theta)} \tag{7}$$

- How can we use this?

The Policy Gradient

$$\theta^* = \arg\max_{\theta} \underbrace{\mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]}_{J(\theta)} \tag{7}$$

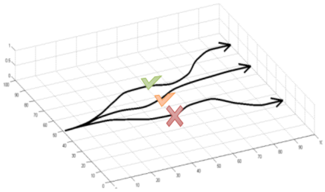


Figure: Simple policy Gradient

- How can we use this?
- Approximate with samples from the environment

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\theta)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_n^N \sum_t^T r(\mathbf{s}_{n,t}, \mathbf{a}_{n,t}) \tag{8}$$

The Policy Gradient

$$\theta^* = \arg\max_\theta \underbrace{\mathbb{E}_{\tau\sim p(\tau|\theta)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)} \tag{7}$$

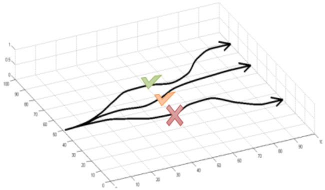


Figure: Simple policy Gradient

- How can we use this?
- Approximate with samples from the environment

$$J(\theta) = \mathbb{E}_{\tau\sim p(\tau|\theta)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_n^N \sum_t^T r(\mathbf{s}_{n,t}, \mathbf{a}_{n,t}) \tag{8}$$

- **Unbiased** estimate of the expected value
- Simple to perform direct gradient ascent

Examples: Reinforce (Williams, 1992; Sutton *et al.*, 2000)

Reducing Variance: Baselines

- $\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla \log p(\tau) r(\tau)$
- Average reward
  - $b_t = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$
  - Reweight trajectories by their average performance

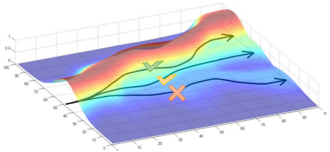


Figure: Policy Gradient

Reducing Variance: Baselines

- $\nabla_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla \log p(\tau) r(\tau)$
- Average reward
  - $b_t = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$
  - Reweight trajectories by their average performance

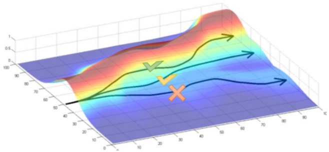


Figure: Policy Gradient

- Will this change the optimal policy?
- $\mathbb{E}[\nabla_\theta \log p(\tau|\theta) b] = \int p(\tau) \nabla_\theta \log p(\tau|\theta) b d\tau$
  - Use identity
- $\int \nabla_\theta p(\tau|\theta) b d\tau = b \nabla_\theta \int p(\tau|\theta) d\tau = b \nabla_\theta 1 = 0$
  - Same optimal policy

Basic Reinforcement Learning Loop: Update Policy

Update Policy

```
def update(self, observations, acs_na, adv_n=None, acs_labels_na=None, qval
    observations = ptu.from_numpy(observations)
    actions = ptu.from_numpy(acs_na)
    adv_n = ptu.from_numpy(adv_n)
    action_distribution = self.policy(observations)
    loss = - action_distribution.log_prob(actions) * adv_n
    loss = loss.mean()
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

Creating an RL Environment for a Robot

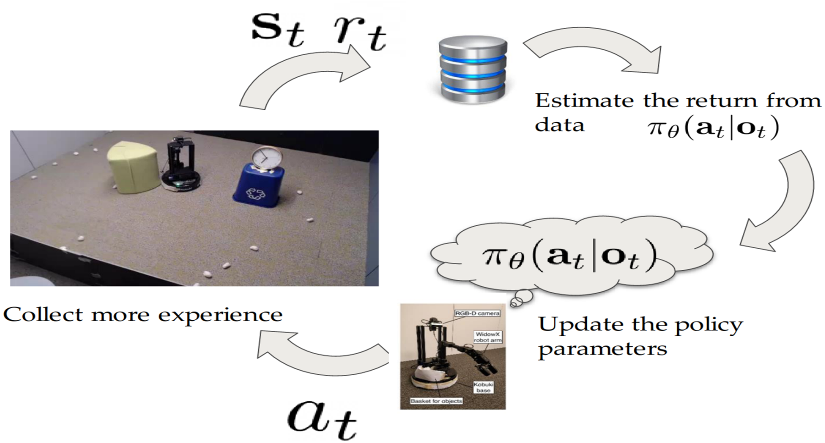


Figure: Robot: sensory motor loop

Load your robot model

- Create a simulated environment for the control loop
  - Or a real environment
- Create a reward function
  - Easy in simulation, often difficult in the real world

OpenAiGym API

```
env = gym.make(env_id)
env = gym.wrappers.RecordEpisodeStatistics(env)
```

DeepRL and Robotics

OpenAIGym Wrappers for Preprocessing

```
## Deep Networks like outputs in [-1,1]
env = gym.wrappers.ClipAction(env)
## Deep Networks like inputs in [-1,1]
env = gym.wrappers.NormalizeObservation(env)
env = gym.wrappers.TransformObservation(env, lambda obs: np.clip(obs, -10,
## DeepRL likes rewards [-1,1]
env = gym.wrappers.NormalizeReward(env, gamma=gamma)
env = gym.wrappers.TransformReward(env, lambda reward: np.clip(reward, -10,
```

- This way, learning rates, etc, have meaning

Deep Reinforcement Learning Algorithms

- Model-Based Reinforcement Learning
- Stochastic Policy Gradients
- Reinforce, NPO, TRPO, PPO, Actor-Critic
- Q-Learning
- Approximate Dynamic Programming
- Important to consider the Deep Network ingredient.

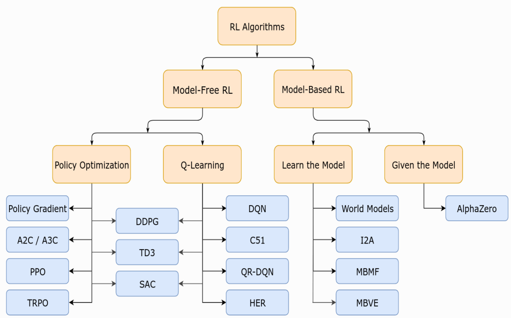


Figure: RL algorithm taxonomy (spi, 2020)

Many RL libraries to use

- Stable Baselines: Good place to start
- cleanrl: simple implementations of RL algorithms
- rlkit: Designed for robotics applications
- tf_agents: Based on deepmind applications
- Many others..

Many RL libraries to use

- Stable Baselines: Good place to start
- cleanrl: simple implementations of RL algorithms
- rlkit: Designed for robotics applications
- tf_agents: Based on deepmind applications
- Many others..

- Learn how to use RL first with simple examples
  - See my class
- Then upgrade to code for real experiments.

Example: Distributed PPO

- Learning to navigate obstacles from vision
- Works across morphologies
- Used distributed computation to speed up training speed
- Not the best motion...
- Heess *et al.* (2017)

Figure: Emergent Behaviours

parkor

Q-learning (off-policy)

- Estimate the action valued function
  - $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{p_\theta}\left[\sum_{t'=t}^{T} r(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \mid \mathbf{s}_t, \mathbf{a}_t\right]$
- Reward for taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and then following policy $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$
- Recursive definition, use dynamic programming
  - $\mathcal{L} = \mathbb{E}_{(s,a,s')\sim p_{\text{data}}} \| Q_\phi(\mathbf{s}_t, \mathbf{a}_t) - (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{a'} Q_\phi(s', a') \|^2$
- How to use
  - Act using $\mathbf{a}_t^* \leftarrow \arg\max_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t, \phi)$
- Stability issues training Q functions
- Policy changes rapidly, Q values unbounded

Example: DQN (Atari)

- Playing Atari with deep reinforcement learning, Mnih *et al.* (2015)
- Q-learning with convolutional neural networks
- Epsilon greedy exploration
- Target Networks

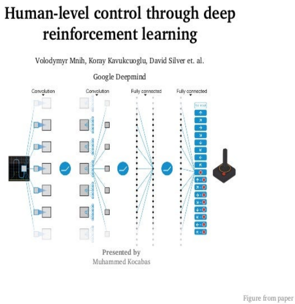


Figure: DQN on Atari

Breakout

Many RL Algorithms

- Different trade-offs
  - Sample efficiency
  - Ease of coding and stability
- Different environment assumptions
  - Episodic vs infinite horizon
  - Continuous/Discrete
  - Deterministic vs Stochastic
- Different types of MDPs
  - Real-world constraints
- When to use which one?

Often Not about Sample Efficiency

- Often about stability
- Will the algorithm converge with enough data?
- What does it converge to?

Often Not about Sample Efficiency

- Often about stability
- Will the algorithm converge with enough data?
- What does it converge to?
- Supervised learning doesn't have these concerns
  - Can always use gradient descent
- Reinforcement learning: often not gradient descent
  - Q-learning: fixed point iteration
  - Model-based RL: model is not optimized for expected reward
  - Policy gradient: is gradient descent, but also often the least efficient

What are we even doing..?

- Value function fitting
  - At best, minimizes error of fit ("Bellman error")
    - Not the same as expected reward
  - At worst, doesn't optimize anything
    - Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear or non-tabular case
- Model-based RL
  - Model minimizes error of fit
    - This does converge
  - No guarantee that better model = better policy
- Policy gradient
  - The only method actually performs gradient descent (ascent) on the true objective

So-called Assumptions

- Common assumption #1: full observability
  - Often assumed when training the value function
  - Or can be compensated with LSTM
- Common assumption #2: episodic learning
  - Often assumed by pure policy gradient methods
  - Assumed by some model-based RL methods
- Common assumption #3: continuity or smoothness
  - Assumed by some continuous value function learning methods
  - Often assumed by some model-based RL methods

DeepRL Tutorial

- cleanrl:
- Setup code here.
  - https://github.com/milarobotlearningcourse/cleanrl/blob/master/roble_install.md

- cleanrl:
- Setup code here.
  - https://github.com/milarobotlearningcourse/cleanrl/blob/master/roble_install.md
- Fix code in ppo_continuous_action.py
  - https://github.com/milarobotlearningcourse/cleanrl/blob/master/cleanrl/ppo_continuous_
  - look for "TODO ##"
  - Ask questions!

Scratch

Scratch

- Possibly some material from Sergey Levine's deepRL course

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. May 2018.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

Amir-Massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-Aware Loss Function for Model-based Reinforcement Learning. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1486–1494. PMLR, 2017.

Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. pages 5541–5552, November 2020.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. pages 1–19, December 2019.

Nicolas Heess, T B Dhruva, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S M Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. July 2017.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. June 2019.

Joshua Joseph, Alborz Geramifard, John W Roberts, Jonathan P How, and Nicholas Roy. Reinforcement learning with misspecified model classes. In *2013 IEEE International Conference on Robotics and Automation*, pages 939–946, May 2013.

Nathan Lambert, Brandon Amos, Omry Yadan, and Roberto Calandra. Objective mismatch in model-based reinforcement learning. February 2020.

Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath. Reinforcement learning for robust parameterized locomotion control of bipedal robots. *International Conference on Robotics and Automation (ICRA 2021)*, 2021.

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

Evgenii Nikishin, Romina Abachi, Rishabh Agarwal, and Pierre-Luc Bacon. Control-Oriented Model-Based reinforcement learning with implicit differentiation. *AAAI*, 36(7):7886–7894, June 2022.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. October 2019.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey

Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Part 2: Kinds of RL algorithms — spinning up documentation. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html, 2020. Accessed: 2023-1-16.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063. proceedings.neurips.cc, 2000.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. NerveNet: Learning structured policy with graph neural networks. February 2018.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3):229–256, May 1992.