EPFL Neuroengineering Laboratory

BIOENG-456 Controlling Behaviors in Animals and Robots

Reinforcement Learning Tutorial

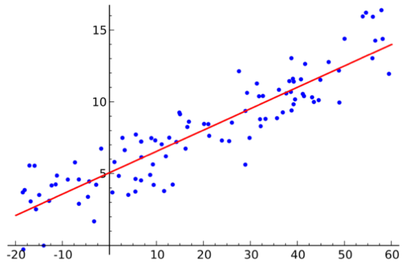
Prof. Pavan Ramdya
TA: Sibo Wang

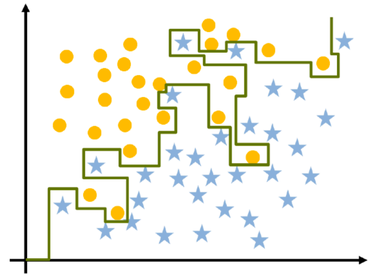
A brief introduction

Types of machine learning tasks

Supervised learning

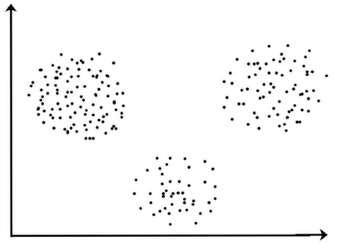
- Goal: predict output given new input
- Training requires input-output pairs
- Example: regression, classification

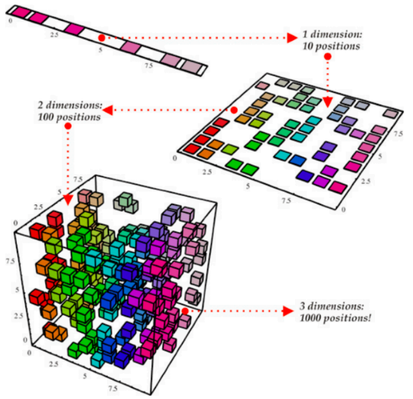




Unsupervised learning

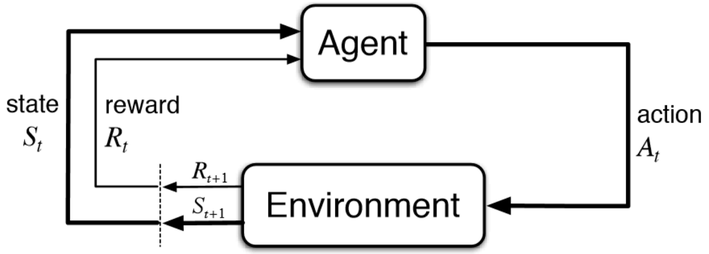
- Goal: discover hidden structures/patterns
- Learning from unlabeled data (no input-output pairs)
- Example: clustering, dimensionality reduction





Reinforcement learning learning

- Goal: learn optimal “policy" (control strategy)
- Learning from interaction with environment
- No input-output pairs, but reward is given
- Example: games, robotics, navigation

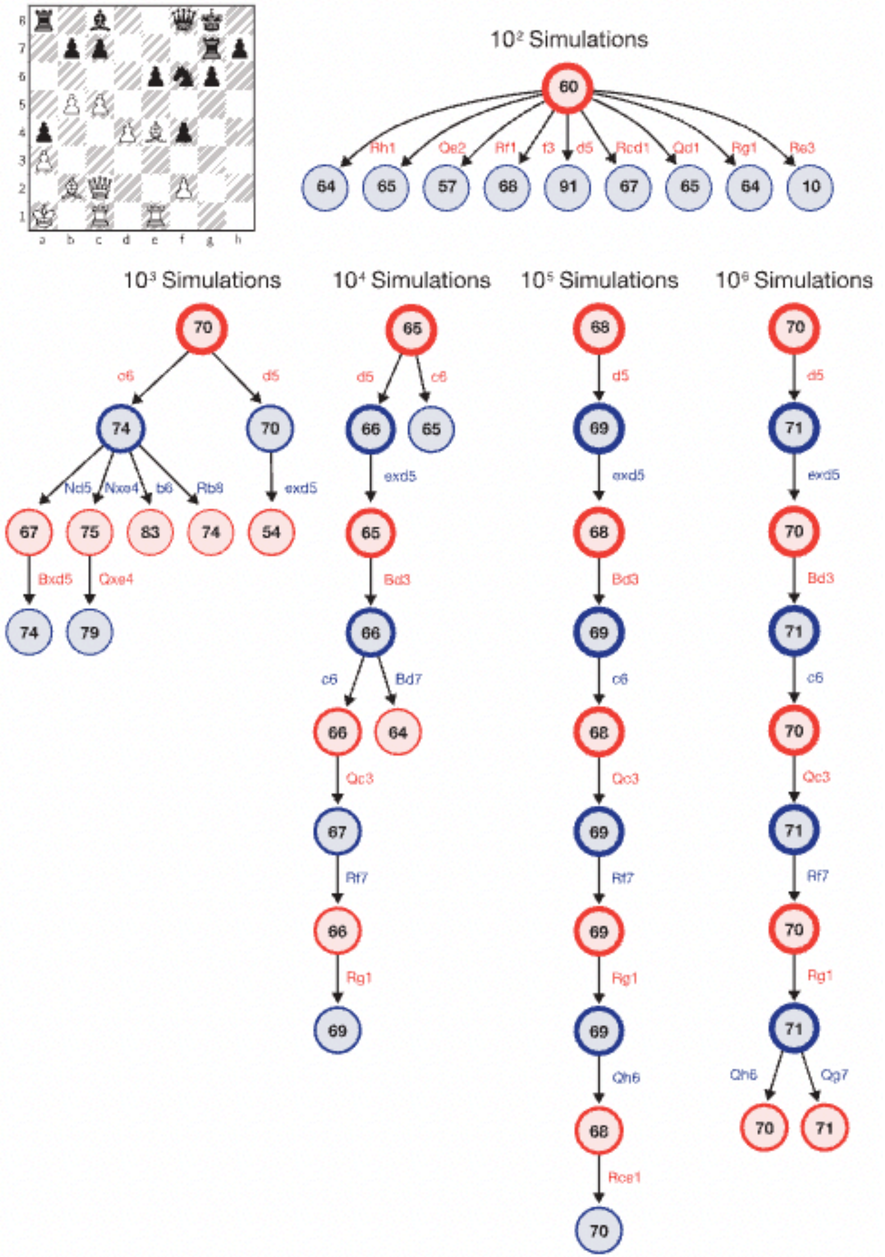


COMPUTER SCIENCE

A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play

David Silver[1,2*†], Thomas Hubert[1*], Julian Schrittwieser[1*], Ioannis Antonoglou[1], Matthew Lai[1], Arthur Guez[1], Marc Lanctot[1], Laurent Sifre[1], Dharshan Kumaran[1], Thore Graepel[1], Timothy Lillicrap[1], Karen Simonyan[1], Demis Hassabis[1†]

The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. By contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go by reinforcement learning from self-play. In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess), as well as Go.



David Silver et al. ,A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.Science362,1140-1144(2018).DOI:10.1126/science.aar6404

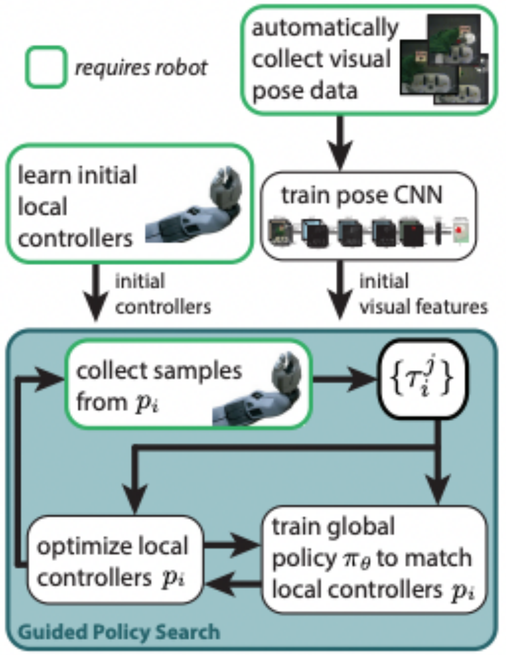
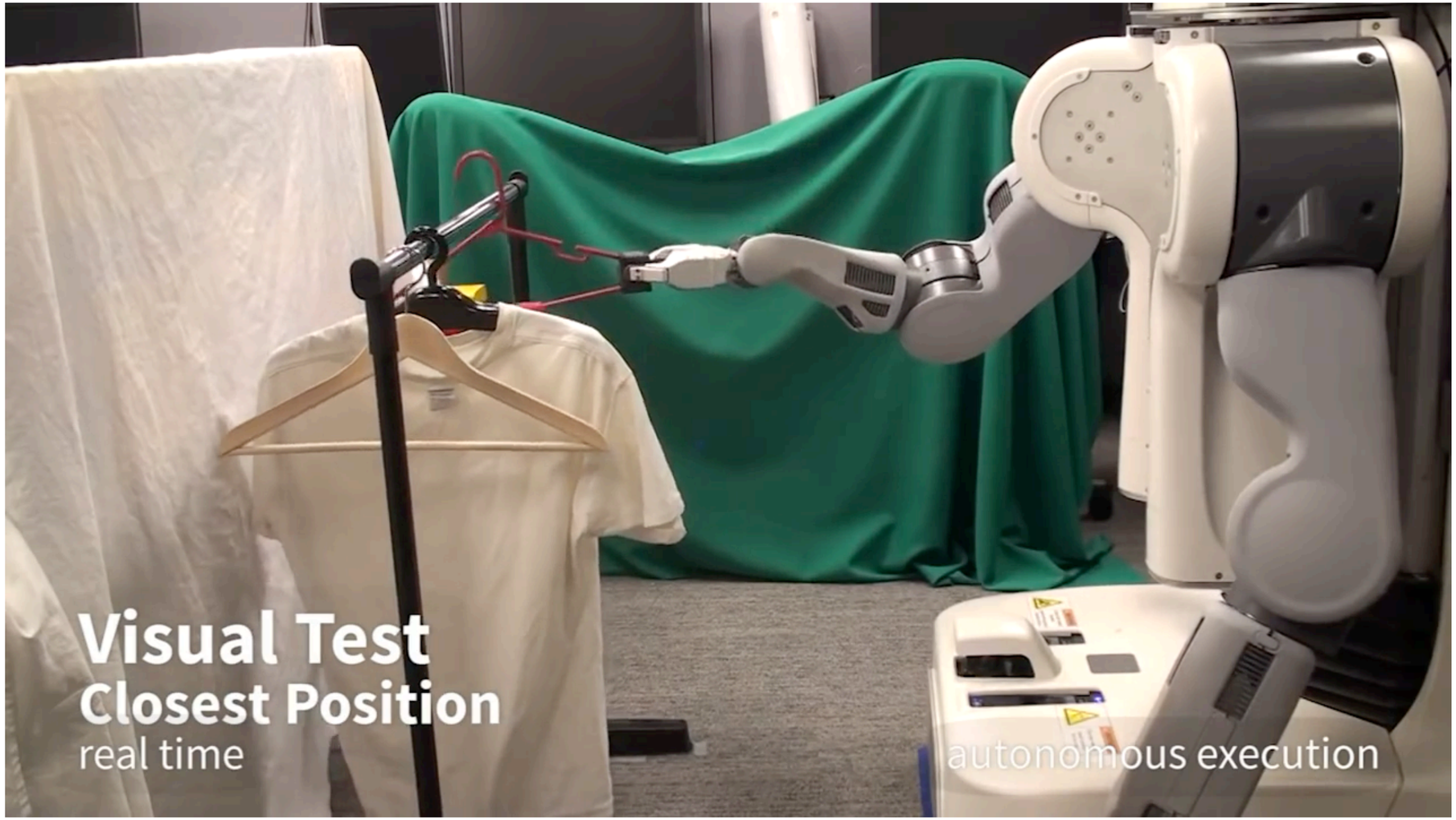


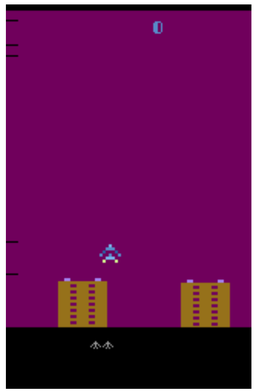
Figure 3: Diagram of our approach, including the main guided policy search phase and initialization phases.

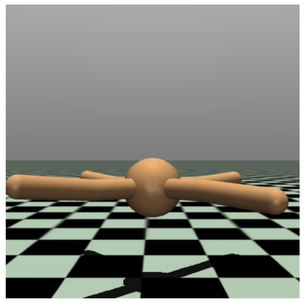


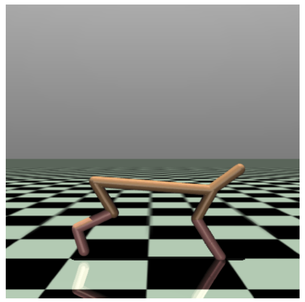
Levine, Sergey, et al. "End-to-end training of deep visuomotor policies." *The Journal of Machine Learning Research* 17.1 (2016): 1334-1373.

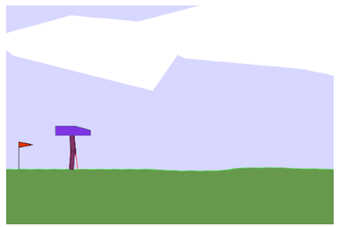
Practical aspects of RL in general

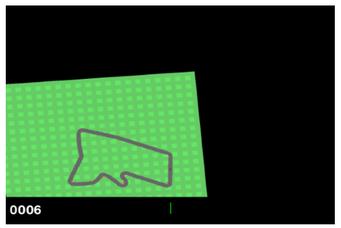
A variety of tasks and algorithms

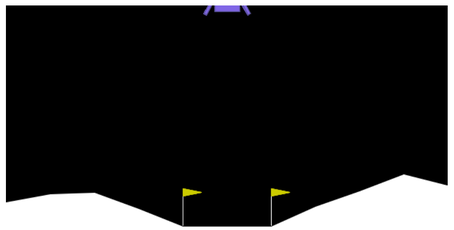


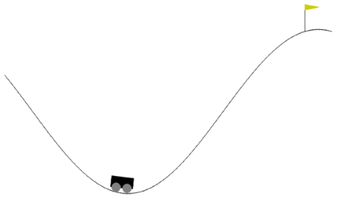


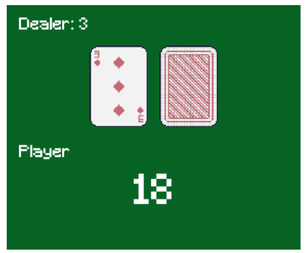


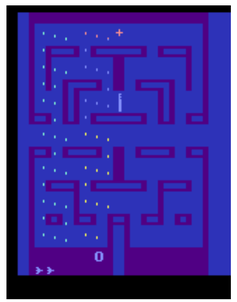










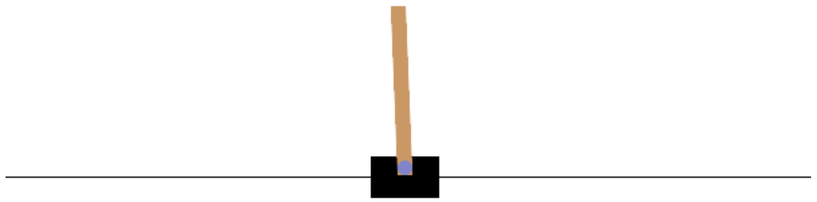


Tasks

Algorithms

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
ARS [1]	✓	✓	✗	✗	✓
A2C	✓	✓	✓	✓	✓
DDPG	✓	✗	✗	✗	✓
DQN	✗	✓	✗	✗	✓
HER	✓	✓	✗	✗	✓
PPO	✓	✓	✓	✓	✓
QR-DQN [1]	✗	✓	✗	✗	✓
RecurrentPPO [1]	✓	✓	✓	✓	✓
SAC	✓	✗	✗	✗	✓
TD3	✓	✗	✗	✗	✓
TQC [1]	✓	✗	✗	✗	✓
TRPO [1]	✓	✓	✓	✓	✓
Maskable PPO [1]	✗	✓	✓	✓	✓

Example Task: Cart Pole



Cart Pole: A cart that can move left or right, with a pole placed on it held by a hinge joint

Goal: keep the pole upright for as long as possible (task fails if pole banks by >±12°)

- What is the action space?

  Discrete: {0: push cart to the left, 1: push cart to the right}

- What is the observation space?

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

- What is the reward function?

  R = number of time steps run before termination

Example Algorithm: Policy Gradient

A trajectory $\tau = ((s_0, a_0), \ldots, (s_T, a_T))$ of states s_i and actions a_i at each step i

A reward function $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$

A policy π implemented by a neural network with parameters θ: π_θ

We want to find θ to maximize reward: $\max_\theta \mathbb{E}_{\pi_\theta} R(\tau)$

We do this via gradient descent:

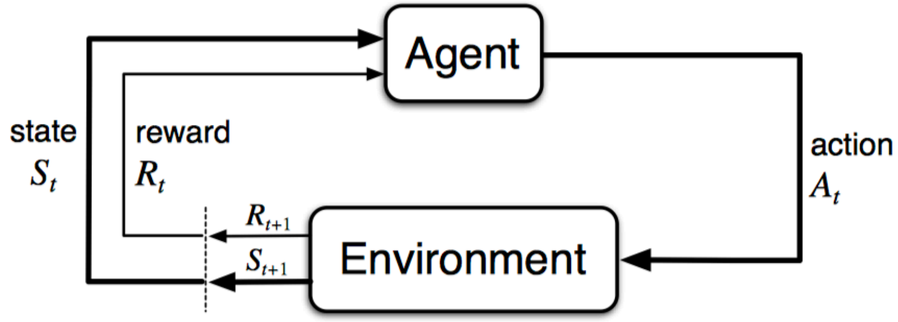
Compute gradient $\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau)$

Update parameters: $\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau)$

After much simplification: $\nabla_\theta \mathbb{E}_{\pi_\theta} R(\tau) = \mathbb{E}_{\pi_\theta} \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} R(s_{t'}, a_{t'}) \right)$

expected future reward

repeated for the whole simulation



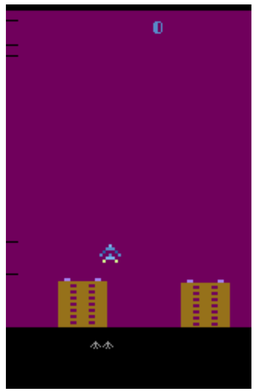
Vanilla policy gradient is almost never used

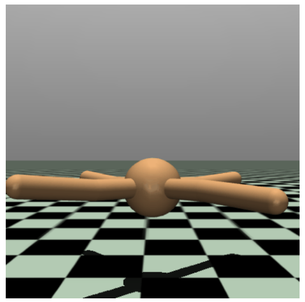
A large family of policy-gradient-based methods:

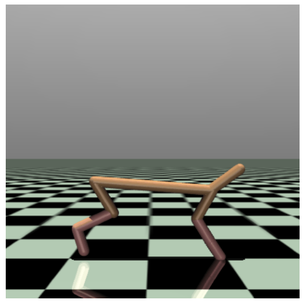
- REINFORCE
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)
- Deterministic Policy Gradient (DPO)
- Advantage Actor-Critic (A2C)
- …

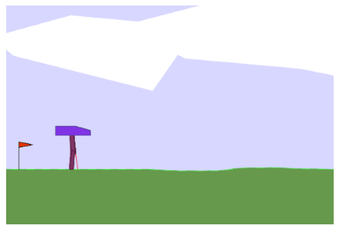
Janis Klaise, “Reinforcement learning with policy gradients in pure Python” https://www.janisklaise.com/post/rl-policy-gradients/

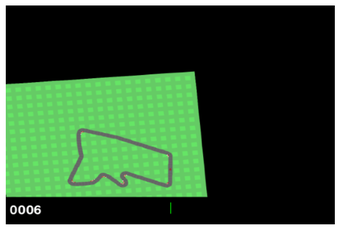
A variety of tasks and algorithms

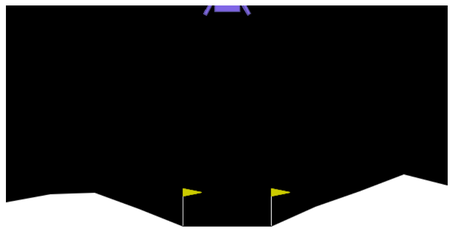


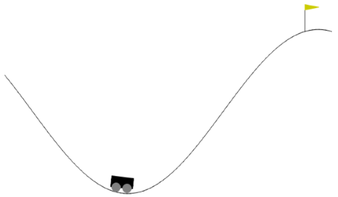


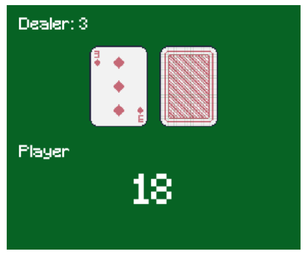


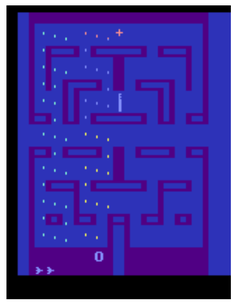












Tasks

Algorithms

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
ARS [1]	✓	✓	✗	✗	✓
A2C	✓	✓	✓	✓	✓
DDPG	✓	✗	✗	✗	✓
DQN	✗	✓	✗	✗	✓
HER	✓	✓	✗	✗	✓
PPO	✓	✓	✓	✓	✓
QR-DQN [1]	✗	✓	✗	✗	✓
RecurrentPPO [1]	✓	✓	✓	✓	✓
SAC	✓	✗	✗	✗	✓
TD3	✓	✗	✗	✗	✓
TQC [1]	✓	✗	✗	✗	✓
TRPO [1]	✓	✓	✓	✓	✓
Maskable PPO [1]	✗	✓	✓	✓	✓

Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations

Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch. It is the next major version of Stable Baselines.

Github repository: https://github.com/DLR-RM/stable-baselines3

Paper: https://jmlr.org/papers/volume22/20-1364/20-1364.pdf

RL Baselines3 Zoo (training framework for SB3): https://github.com/DLR-RM/rl-baselines3-zoo

RL Baselines3 Zoo provides a collection of pre-trained agents, scripts for training, evaluating agents, tuning hyperparameters, plotting results and recording videos.

SB3 Contrib (experimental RL code, latest algorithms): https://github.com/Stable-Baselines-Team/stable-baselines3-contrib

Main Features

- Unified structure for all algorithms
- PEP8 compliant (unified code style)
- Documented functions and classes
- Tests, high code coverage and type hints
- Clean code
- Tensorboard support
- **The performance of each algorithm was tested** (see *Results* section in their respective page)

Name	Box	Discrete	MultiDiscrete	MultiBinary	Multi Processing
ARS [1]	✓	✓	✗	✗	✓
A2C	✓	✓	✓	✓	✓
DDPG	✓	✗	✗	✗	✓
DQN	✗	✓	✗	✗	✓
HER	✓	✓	✗	✗	✓
PPO	✓	✓	✓	✓	✓
QR-DQN [1]	✗	✓	✗	✗	✓
RecurrentPPO [1]	✓	✓	✓	✓	✓
SAC	✓	✗	✗	✗	✓
TD3	✓	✗	✗	✗	✓
TQC [1]	✓	✗	✗	✗	✓
TRPO [1]	✓	✓	✓	✓	✓
Maskable PPO [1]	✗	✓	✓	✓	✓

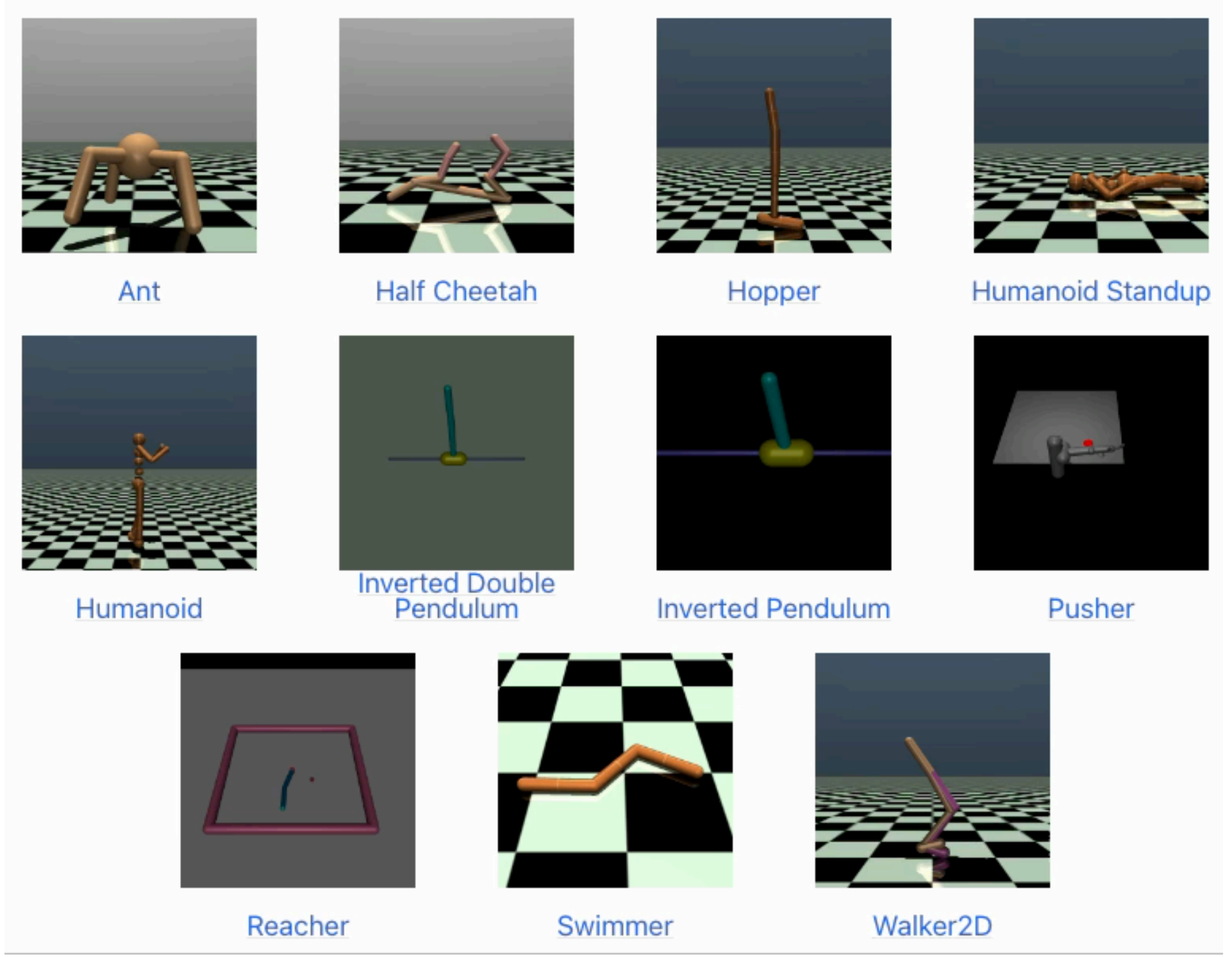
Training models with stable baselines 3 and gym

```python
import gymnasium as gym

from stable_baselines3 import A2C

env = gym.make("CartPole-v1")   ← Define a gym environment

model = A2C("MlpPolicy", env, verbose=1)     ← Define a model
model.learn(total_timesteps=10_000)          ← Train it

vec_env = model.get_env()
obs = vec_env.reset()
for i in range(1000):    ← Roll out your controller
    action, _state = model.predict(obs, deterministic=True)
    obs, reward, done, info = vec_env.step(action)
    vec_env.render()  ← Observe what happens
```

[DEMO 1]

Controlling NeuroMechFly with RL

NeuroMechFly is much higher dimensional than the “toy tasks” (42 DoFs)



What should the RL agent really control?

What should the RL agent do exactly?

Inputs

- Joint states?
- Fly’s global position and orientation?
- Leg contacts?
- How many past steps?

Outputs

- Parameters of the CPGs?
- Parameters of the decentralized control rules?
- Whether a step should be triggered for each leg?
- Joint angles directly?

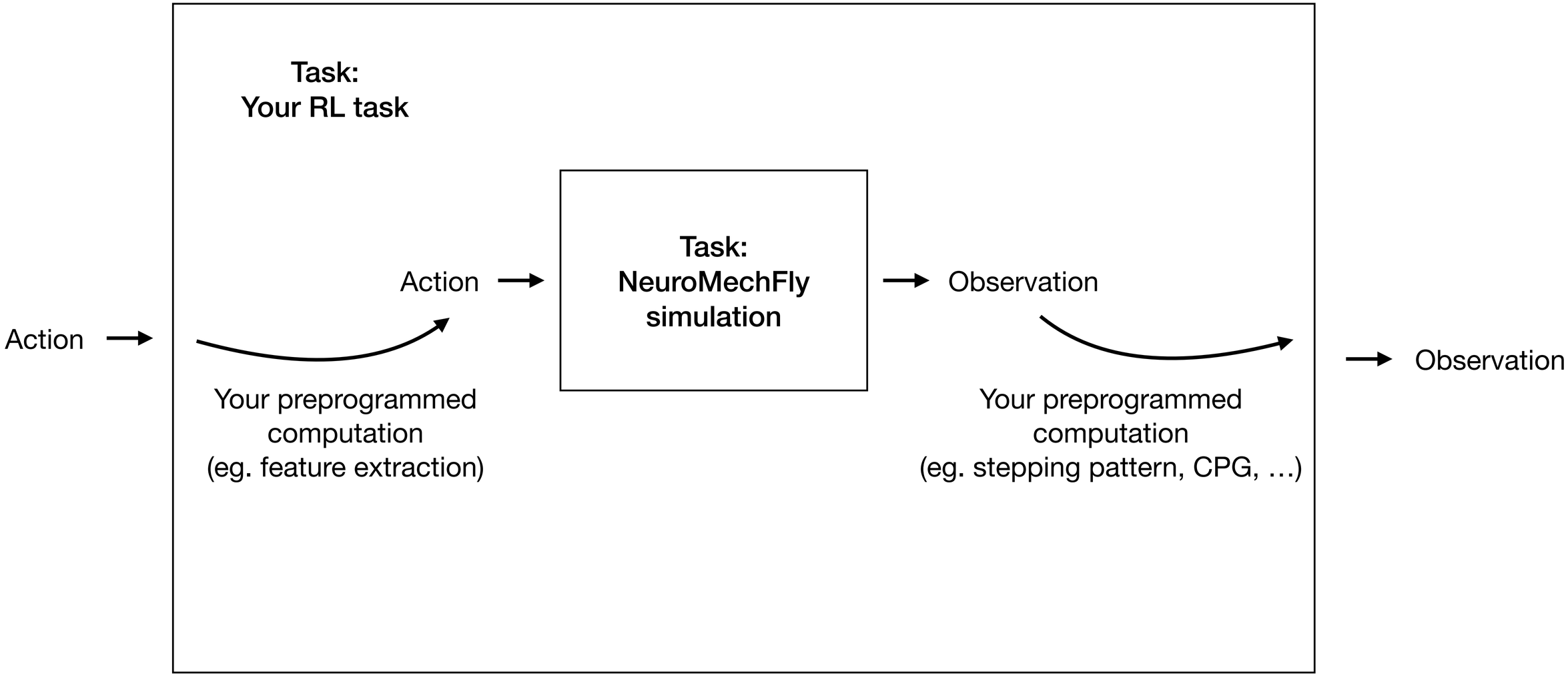
Reward

- Speed?
- Leg contact pattern?
- Stability?

Constraints

- Terminate if all legs off the ground?
- Terminate if fly flipped?

What should the RL agent do exactly?



[DEMO 2]