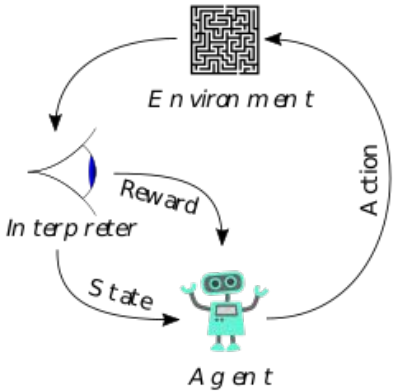
Multi-Agent Reinforcement Learning: Systems for Evaluation and Relations to Complex Systems

Part 1: Multi-Agent Reinforcement Learning Systems for Evaluation

Reinforcement Learning



- Machine learning for “Optimal Control”
  - Computer games
  - Robots
  - Stock trading
  - Elevator power management

Given a Partially Observable Markov Decision Process (POMDP):
S is a set of states in an environment, $s \in S$
A is a set of actions, $a \in A$
$T(s'|s,a)$ is a set of conditional transitional probabilities
$R : S \times A \to \mathbb{R}$ is the reward function, $r \in R$
Ω is a set of observations, $o \in \Omega$
$O(o|s',a)$ is the set of conditional observation probabilities,
$\gamma \in [0,1]$ is the discount factor

The goal is to learn a policy π:
$\pi(a,s) = \mathbb{P}[A_t = a|O_t = o]$
Such that $argmax_{\pi \in \Pi} G = E(\sum_{t=0}^{\infty} \gamma^t r_t)$
Where Π is the set of all possible policies,
and where G is the “return”

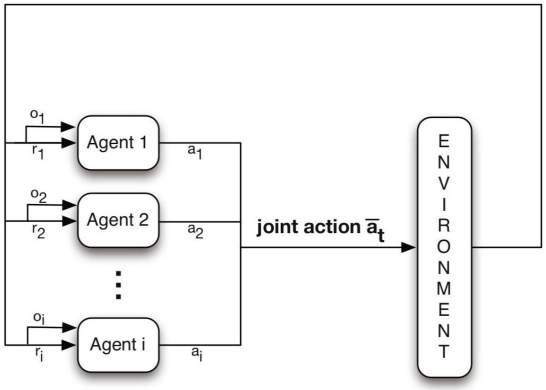
A Brief History of Modern Reinforcement Learning

- “Learning many Atari games is a very good milestone for reinforcement learning”
  - https://gym.openai.com/envs/#atari
  - Arcade Learning Environment
    - Graphical
    - Moderately challenging to humans
    - Clear reward
    - Diverse and unique
- DeepMind did this with the DQN
- Elon Musk thought the robot apocalypse was upon us
  - OpenAI Gym
    - Democratization
    - Make RL accessible to university researchers

What if you want to learn chess or Starcraft 2?

What if you want to coordinate Amazon warehouse robots?

Multi-Agent Reinforcement Learning (MARL)



Source: Nowe, Vrancx & De Hauwere 2012

All multi-agent system can, though not always should, be modeled as a Partially Observable Stochastic Game ("POSG"):

- $\mathcal{S}$ is the set of possible *states*.
- N is the *number of agents*. The *set of agents* is $[N]$.
- $\mathcal{A}_i$ is the set of possible *actions* for agent i.
- $P\colon \mathcal{S} \times \prod_{i\in[N]} \mathcal{A}_i \times \mathcal{S} \to [0,1]$ is the (stochastic) *transition function*.
- $R_i\colon \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N \times \mathcal{S} \to \mathbb{R}$ is the *reward function* for agent i.
- Ω_i is the set of possible *observations* for agent i.
- $O_i\colon \mathcal{A}_i \times \mathcal{S} \times \Omega_i \to [0,1]$ is the *observation function*.

Policy Variations in Multi-Agent Reinforcement Learning

Centralized:

- Agents synthetically pass information beyond the confines of the environment
  - Often by sharing a policy

Decentralized:

- Agents learn individually and only communicate during training

Different parts of the policy can be one or the other, and the mode chosen can vary between training and execution

- E.g. centralized training decentralized execution
- Centralization at all stages equivalent to the single agent case, degenerate case

Reward Scheme Variations in Multi-Agent Reinforcement Learning

- Cooperative: Rewards entirely incentivizes agents to work together
- Competitive (zero sum): Agents compete against each other
- Mixed sum: Agents can benefit from working together and competing

- Fundamental goal remains the same as single agent RL for each agent: find the best policy to achieve the maximum total expected discounted reward

PettingZoo

- “Gym for multi-agent reinforcement learning”
  - MARL is an incredibly engineering effort to do things in, similar to RL before GYM
- Simple universal API that’s very similar to Gym
  - Making it this simple was remarkable difficult
- https://www.pettingzoo.ml/envs
- https://github.com/PettingZoo-Team/PettingZoo

Figure 1: Basic Usage of Gym

```
import gym
env = gym.make('CartPole-v0')
observation = env.reset()
for _ in range(1000):
    env.render()
    action = policy(observation)
    observation, reward, done, info = env.step(action)
env.close()
```

Figure 2: Basic Usage of PettingZoo

```
from pettingzoo.butterfly import pistonball_v0
env = pistonball_v0.env()
env.reset()
for agent in env.agent_iter(1000):
    env.render()
    observation, reward, done, info = env.last()
    action = policy(observation, agent)
    env.step(action)
env.close()
```

SuperSuit

- Preprocessing is almost always done to adapt actions/rewards/observations from an environment to whatever the policy function and learning methods need
- It's an essential feature of all RL
  - It's really hard to to a good job
  - It's really easy to create small bugs
  - Everyone does it themselves
- There should probably be a library for that
- https://github.com/PettingZoo-Team/SuperSuit

Multiplayer Support for the Arcade Learning Environment

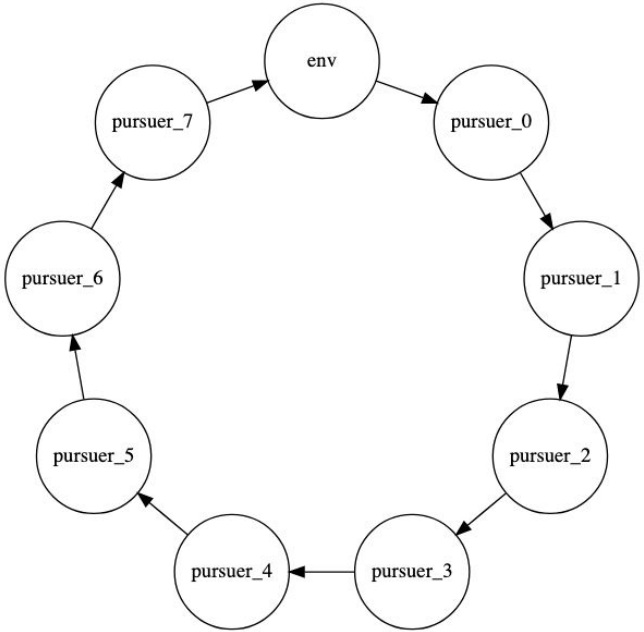
- The ALE is how RL interfaces with Atari games
- We extended the ALE to support multiplayer Atari games
- Turns out learning these is *hard* (ongoing work)
  - MuZero

Ice Hockey (2 player)
on the Atari 2600

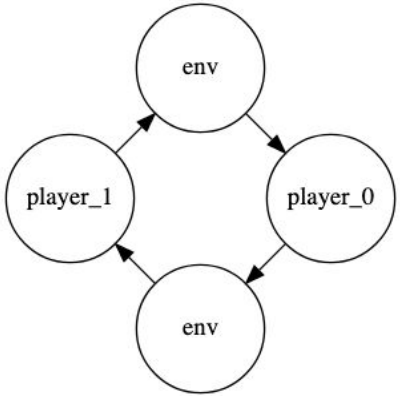
Modeling games in MARL

- POSGs
  - Assume agents step strictly simultaneously
  - Generally used model in MARL, except when games are strictly turn based (e.g. chess)
    - Many other models exist, like multi-agent MDPs, that are strict subsets of POSGs
- Extensive form games (EFGs)
  - Assume reward only happens at end of game
  - Generally used model in MARL for strictly turn based games
- How do you model games if you want to build an API that can handle both?

Agent Environment Cycle Diagrams



Pursuit



Chess

Agent Environment Cycle (“AEC”) Games

- POSG, but agents update sequentially
  - Not a disaster to model turn based or strictly parallel games
  - Constant cycle moving through agent and environment actions
- Provably equivalent model to POSGs
- Basis of PettingZoo API

Formalism of an AEC game:
S is a set of states in an environment, $s \in S$
N is number of agents
Ξ is the set of all agents and the environment agent
$\mathbb{A}_i$ is a set of actions, $a \in A$
$T_i : Sx\mathbb{A}_i \rightarrow S$ is the transition function for agents
$P : SxS \rightarrow [0, 1]$ is the transition function of the environment
R_i is the reward function for agent i
Ω_i is the set of possible observations for agent i
$O_i : S \times \Omega_i \rightarrow [0, 1]$ is the observation function for agent i
$v : S \times \Xi \times \mathbb{A}\Xi \rightarrow [0, 1]$ is the next agent function

Modeling computer games as POSGs doesn't make conceptual sense

- Imperfect tie breaking in competitive snake
  - http://doublesnakegame.com/
  - What if the snakes hit each other or eat the food at the same time?
- Updating is always sequential unless you do crazy parallelization stuff
- Writing code that assumes you do is an excellent way to get race conditions
  - Social sequential delima game race condition
- The AEC games removes this opportunity for incredibly subtle bugs and better aligns with games in practice

POSGs don’t allow access to reward information that should be available

- Reward is emitted from specific agent or environment actions
- If the game steps sequentially, you can get reward at different time points from each source
  - POSGs smash all this together
- When you view reward from this perspective, there’s often lots of things in it that shouldn’t be there that are very hard to figure out
  - Pursuit environment (2 line order switch, 10x performance improvement)
- You can also do “free” curriculum learning with this information