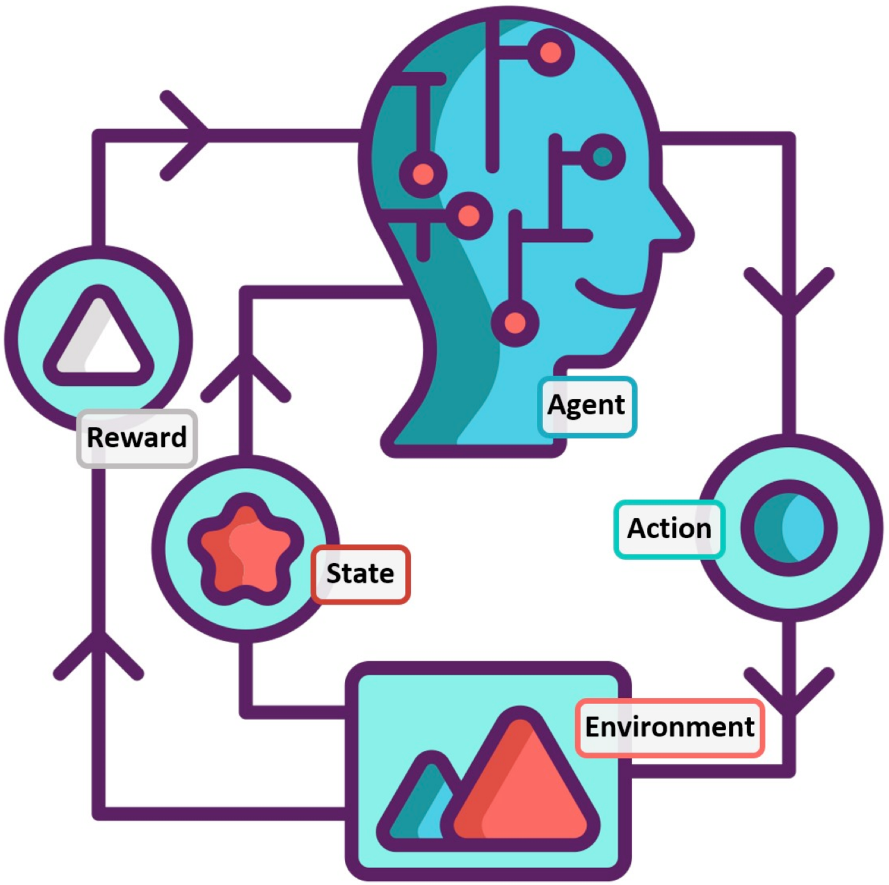
Reinforcement Learning Tutorial

Hands on Reinforcement Learning



Kishansingh Rajput

Data Science Department

Thomas Jefferson National Accelerator Facility

Kishan@jlab.org

Tuesday, November 1, 2022

Jefferson Lab

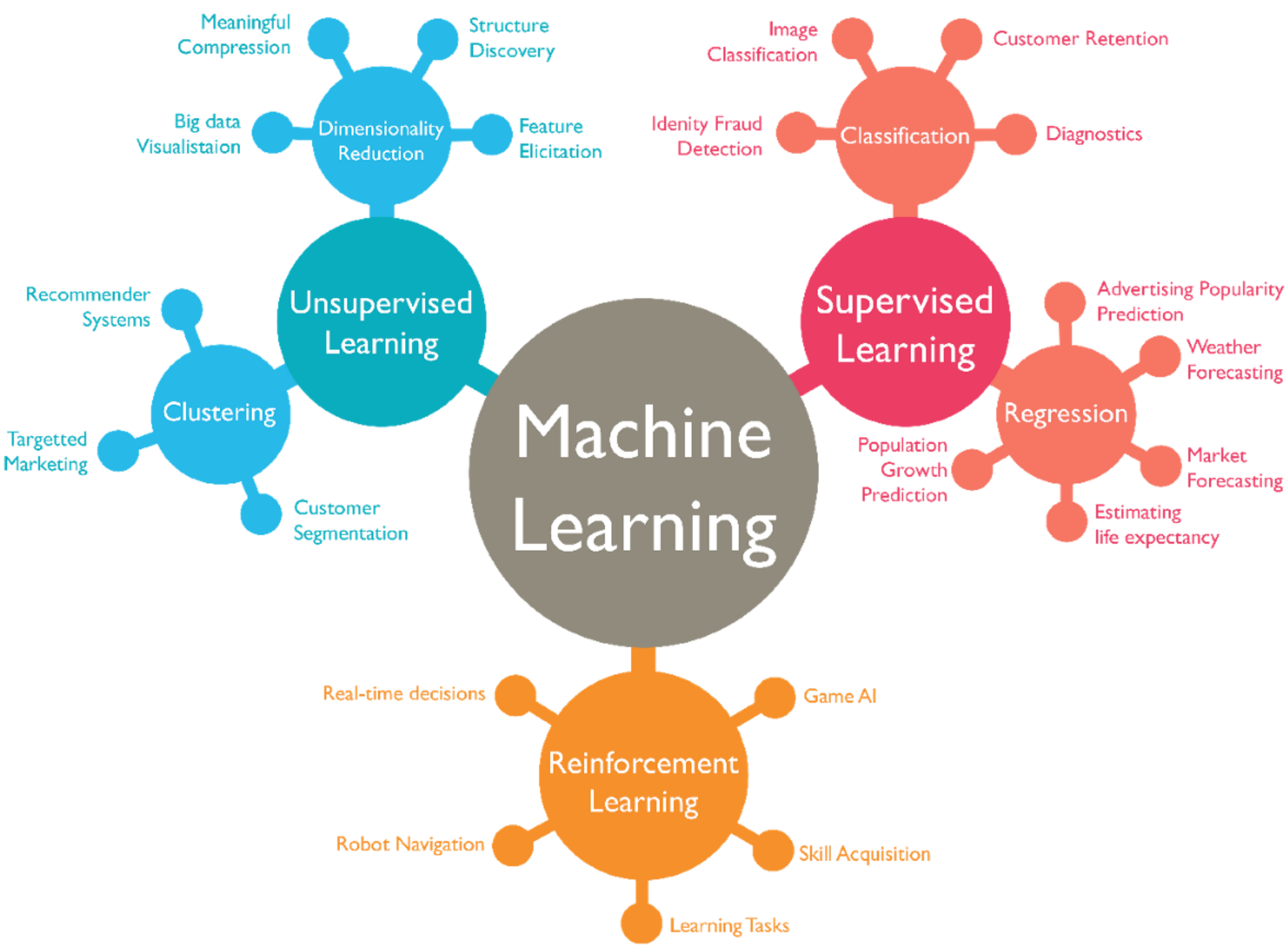
U.S. DEPARTMENT OF ENERGY | Office of Science

JSA

Outline

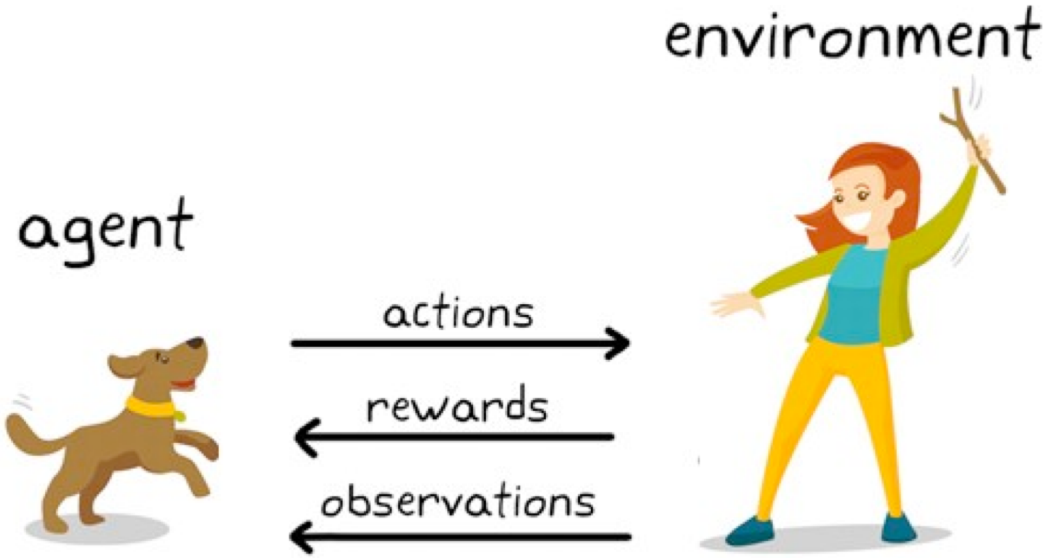
- Introduction to Reinforcement Learning (RL)
- Types of RL algorithms
- Q functions and Bellman Equation
- Introduction to Double Deep Q Networks (DDQN)
- Exploration vs Exploitation
- OpenAI gym, and stable-baseline
- CartPole OpenAI gym environment
- FNAL Booster GMPS regulator OpenAI gym environment
- **Hands on code:**
  - **DDQN agent code**
  - **Train DDQN on Benchmark OpenAI CartPole Environment**
  - **Training DDQN to regulate GMPS (Simplified Env)**

Machine Learning



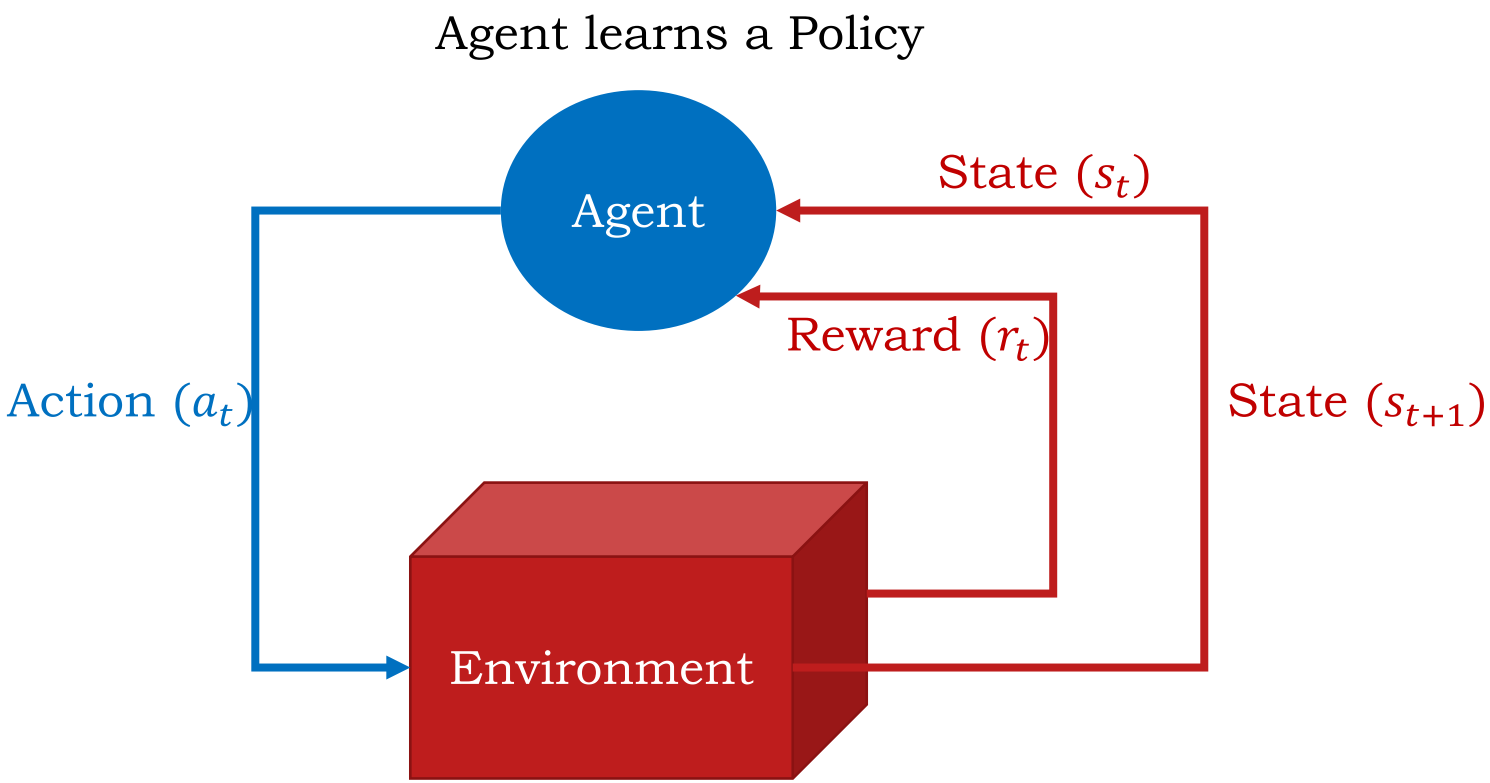
Reinforcement Learning

- Learning from interaction with an environment to achieve/maximize a long-term goal related to the state
- The goal is defined by the reward function
- The agent needs to be able to observe the environment and take actions to modify it's state in order to achieve the goal

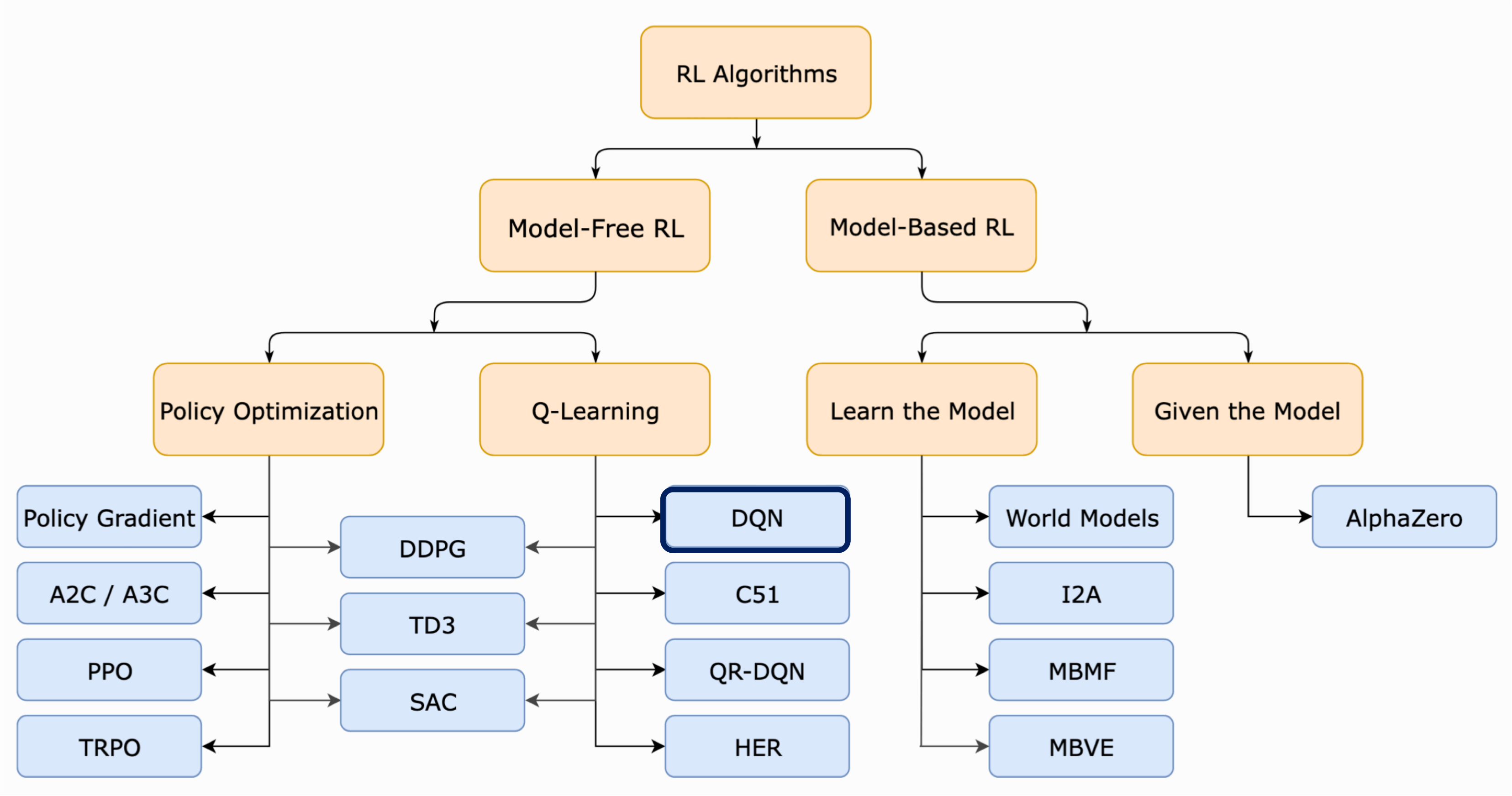


Reinforcement Learning Example – KDNuggets

Reinforcement Learning



RL algorithms



Model-Based RL: The agent can predict the reward for some action before actually performing it thereby planning what it should do.

Model-Free RL: The agent needs to carry out the action to see what happens and learn from it.

Q function and Bellman Equation

Optimal Action-value function $\boldsymbol{Q^*(s,a)}$, which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the *optimal* policy in the environment.

Q function and optimal action

$$a^*(s) = \arg\max_a Q^*(s,a)$$

Bellman equation

$$Q^*(s,a) = r(s_t, a_t) + \gamma \times r(s_{t+1}, a_{t+1}) + \gamma^2 \times r(s_{t+2}, a_{t+2}) \ldots . + \gamma^n \times r(s_n, a_n)$$

$$\boxed{Q^*(s,a) = \underset{s\sim P}{E}[r(s_t, a_t) + \gamma \times \underset{a_{t+1}}{max} Q^*(s_{t+1}, a_{t+1})]}$$

- The $Q^*(s,a)$ is the return value starting at state s' and taking action a, plus the value of wherever you land next
- $r(s,a)$ is the immediate reward for s, a combination
- γ is the discount factor

Key point

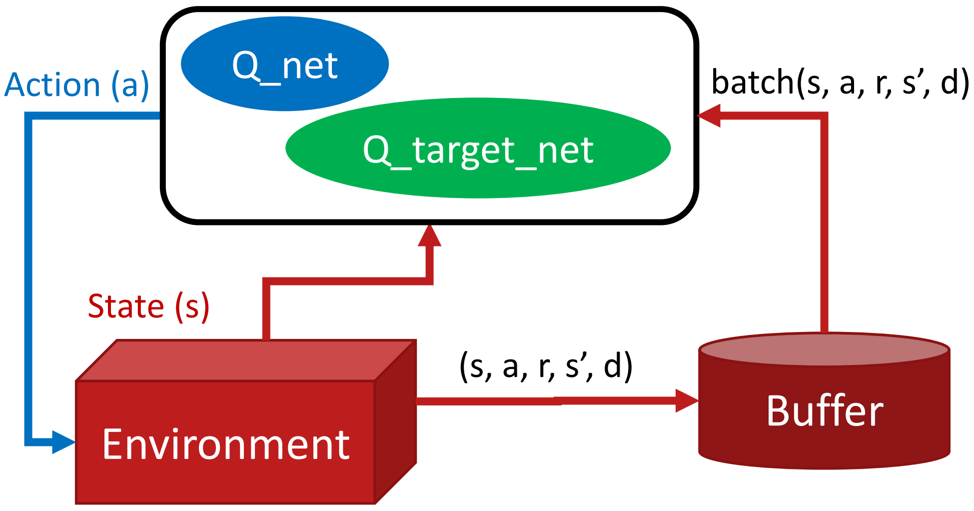
<u>Bellman equation</u>

$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

- r(s, a) is an immediate reward for action a at state s
- This indicate how good the action a is at state s
- The goal of the agent is to maximize the cumulative reward
- Actions may have long term consequences, reward may be delayed
- Sometimes better to sacrifice immediate reward to gain more long term reward

Double Deep Q-Network (DDQN)

A neural network is used to learn the Q-values corresponding to different actions at a given state



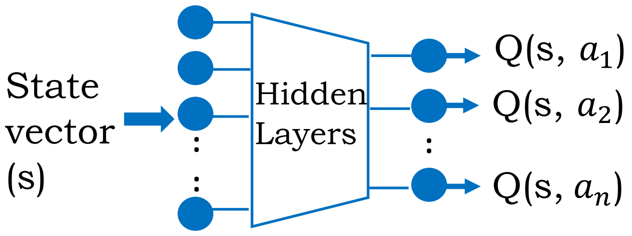
Training Q network (update weights)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^{Q} - Q(S_t, A_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t)$$

Y_t^Q comes from Bellman equation as

$$Y_t^{Q} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)$$

We use a second network (Q target network) to estimate the target Q values. The weights of this network θ'_t are updated by copying the weights of Q network every X steps.



$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

https://arxiv.org/pdf/1509.06461.pdf

Exploration vs Exploitation

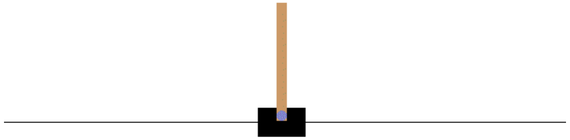
- Exploit previous experiences
  - Store (s, a, r, s', d) tuples to a Buffer to build dataset (D)
  - Train the agent on the dataset (D)
- Explore new actions/states (make agent more robust)
  - To explore new actions add small noise to the actions
  - With new exploration the reward might decrease temporarily during training
  - Slowly decrease the amount of noise as training progress

RL Environment

- OpenAI Gym standards
  - https://github.com/openai/gym/blob/master/gym/core.py
  - Env base class

```
class Env(Generic[ObsType, ActType]):
        def __init__()
        def step(self, action: ActType) -> Tuple[new_state, reward, done, info]:
        def reset(self, *) -> Tuple[state, info]:
        def render(self) -> Optional[Union[RenderFrame, List[RenderFrame]]]:
        def close(self) [Necessary cleanup]
```

CartPole OpenAI gym environment



Action Space	Discrete(2)
Observation Shape	(4,)
Observation High	[4.8 inf 0.42 inf]
Observation Low	[-4.8 -inf -0.42 -inf]
Import	`gym.make("CartPole-v1")`

The action is a ndarray with shape (1,) which can take values {0, 1} indicating the direction of the fixed force the cart is pushed with.

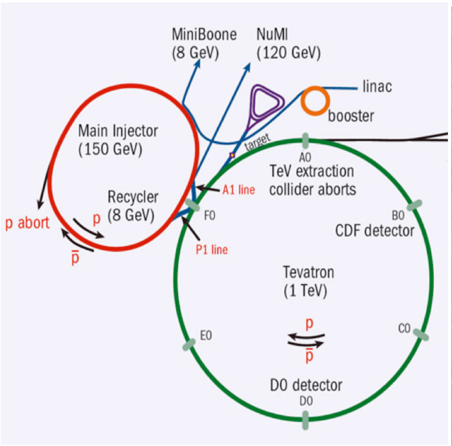
The observation is a ndarray with shape (4,) with the values corresponding to the following positions and velocities:

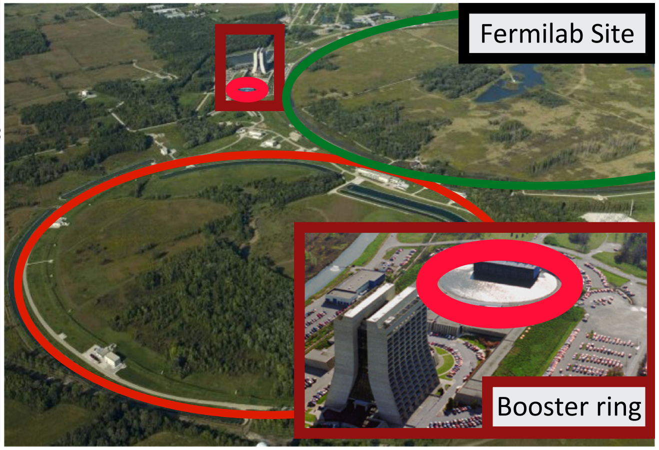
Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

Since the goal is to keep the pole upright for as long as possible, a reward of +1 for every step taken, including the termination step, is allotted.

1. Termination: Pole Angle is greater than ±12°
2. Termination: Cart Position is greater than ±2.4 (center of the cart reaches the edge of the display)
3. Truncation: Episode length is greater than 500 (200 for v0)

GMPS Regulator environment





Courtesy: Christian Herwig

The Booster receives the 400 MeV (kinetic energy) beam from the Linac

It is then accelerated to 8GeV with the help of booster cavities and Combined-function bending and focusing electromagnets known as gradient magnets.

These magnets are powered by the gradient magnet power supply (GMPS)

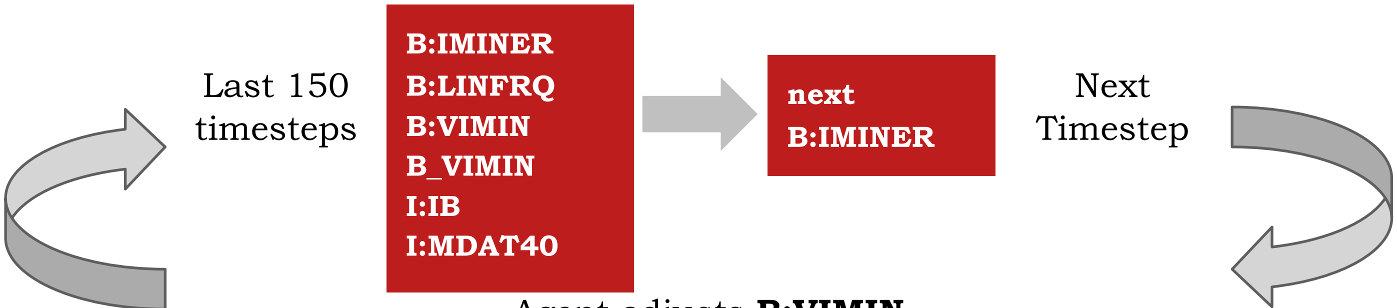
- Other high-current, high-power electrical loads near GMPS varies in time
- Causing unwanted fluctuations of the actual GMPS electrical current and thus fluctuations of the magnetic field in the Booster gradient magnets
- Spread in B-field degrades beam quality, degrades repeatability, & contributes to losses
- A GMPS regulator is required to calculate and apply small compensating offsets in the GMPS driving signal
- A RL agent can be trained to learn an optimal regulator, focusing on reducing the errors

Variables considered to construct env states

B:LINFRQ = 60 Hz line frequency error [mHz]	**B_VIMIN** = Setting to achieve*
I:IB = MI lower bend current [A]	**B:VIMIN** = Prescribed remedy from PID regulator circuit
I:MDAT40 = MDAT measured MI current [A]	**B:IMINER** = 10*(Setting - obsMax)

GMPS Regulator environment

LSTM Surrogate Model



Agent adjusts **B:VIMIN**
(Other variables continue to draw from data)

Environment Setup

- Environment uses LSTM surrogate model to predict next IMINER using last 150 timesteps on all 6 variables
- Agent updates VIMIN (Action: delta VIMIN)
- To build next state the time series is shifted by one and VIMIN, and IMINER are updated as per action and surrogate prediction respectively
- Reward is (-1 * IMINER) since the goal is to minimize IMINER

https://arxiv.org/pdf/2105.12847.pdf

Resources

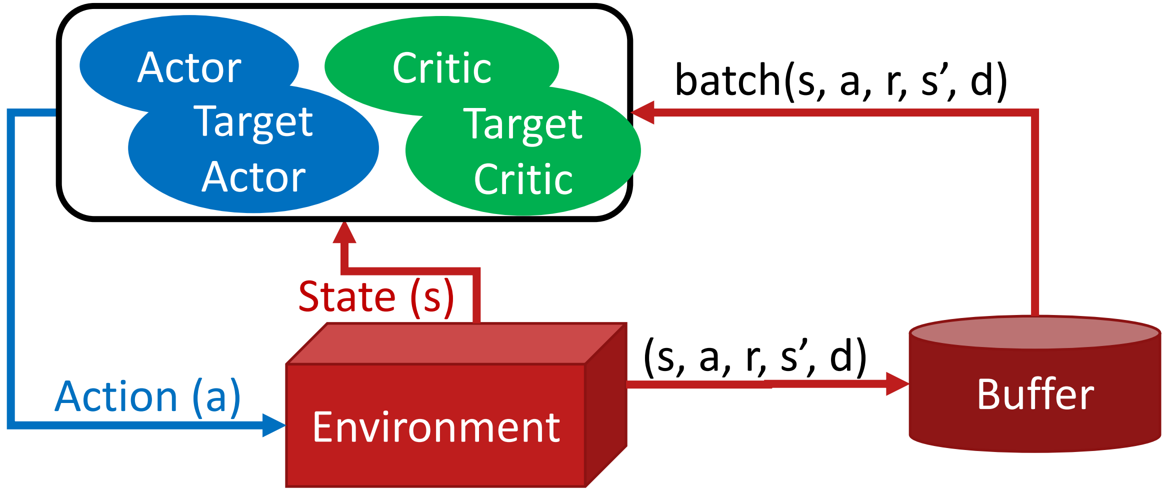
- Make your own custom environment
  https://www.gymlibrary.dev/content/environment_creation/
- Agent libraries
  – Stable-baseline
    - https://stable-baselines.readthedocs.io/en/master/
  – TF-agents
    - https://www.tensorflow.org/agents
- Readings
  – Deep Reinforcement Learning with Double Q-learning
- Tutorial code: https://github.com/JeffersonLab/jlab_datascience_tutorial

BACKUP

Deep Deterministic Policy Gradient (DDPG)

Policy learning μ_θ

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi(s,\mu_\theta(s))\right]$$



Critic is used to approximate the value function, trained using the following loss function

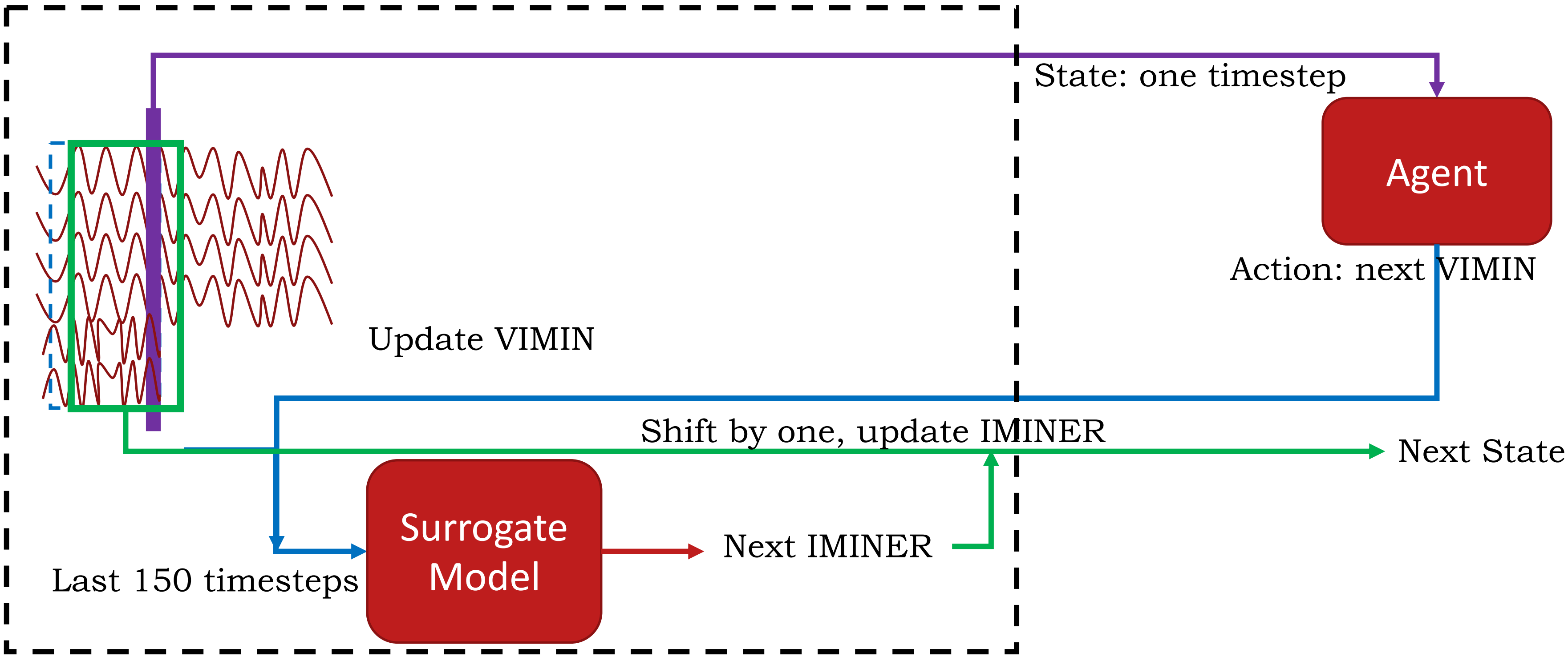
$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

Target network is used to approximate $Q_\phi(s',a')$

As both critic and target critic networks depends on same parameters ϕ, target network is updated with a time delay as

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi, \quad 0 < \rho < 1$$

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s',\mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right]$$


State: one timestep
Agent
Action: next VIMIN
Update VIMIN
Shift by one, update IMINER
Next State
Surrogate Model
Next IMINER
Last 150 timesteps