WELCOME TO THE TUCSON PYTHON MEETUP

Many thanks to our sponsors:

Tek Systems,
Neuro-ID,
& Connect Coworking

What is Open AI Gym?

- A collection of environments conducive to creating, validating, and comparing reinforcement learning algorithms
  - AI Gym is to reinforcement learning (RL) as Imagenet is to supervised learning
- Environments include: legacy control theory, Atari games, 3D locomotion (requires external libraries), robot manipulation
- Free (for the most part)

Basics of Reinforcement Learning

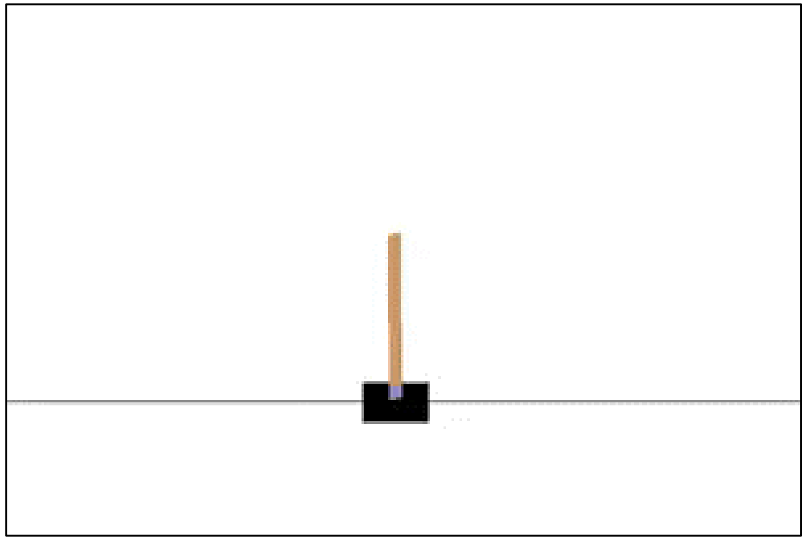
- We have a player (agent) that can act on some world state and receive rewards (or penalties) for entering states as the result of taking actions
- We want our player to perform a set of actions in this environment that maximizes reward
    - The actions that an agent takes when confronted with a given state is called a **policy**
- Assume that time is discrete, or snapshot-able
- Need three things: states, actions, rewards
- **State**: some representation of the world at a given time
- **Actions**: moves that can be made by the player while in a state at a given time; taking an action can cause the world state to change
- **Reward**: points that the player receives from taking actions while in a state

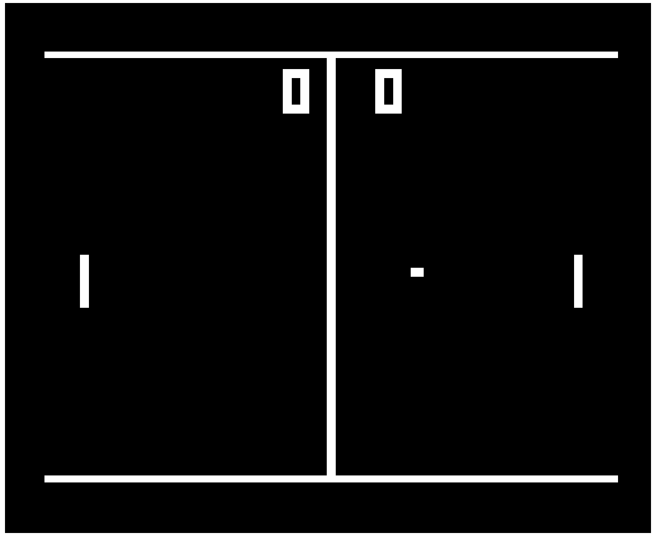
States, Actions, Rewards

Basics of Reinforcement Learning

- States can be large or small, `discrete` or *continuous*
  - A state is just a representation of the environment at a given time; a representation could be as small as a position and velocity measurement or as large as pure pixels from an image
- Actions can be `discrete` or *continuous*
- Rewards are typically `discrete`
  - Rewards are given at every time-step
  - We say that rewards are more desirable the sooner they're received (rewards that arrive later are discounted by a factor $0 \leq \gamma \leq 1$)
- There is usually a terminal state after which the player receives zero reward
  - E.g. Mario gets to the castle at the end of each level

Basics of Reinforcement Learning - Examples





Basics of Reinforcement Learning

- We receive a reward at each timestep, but the player's total reward for an episode isn't necessarily the sum of all rewards
  - We have to include the discount factor (because we prefer points soon to points far in the future)
  - The total episode reward is:

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^N r_N = \sum_{i=0}^{N} \gamma^i r_i$$

  - The reward from time t until the end of the episode is:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{N-t} r_N$$

Q-Learning In, Like, 3 Slides

- Q is a function, $Q(\mathrm{state}_t, \mathrm{action}_t)$, which is the maximum discounted reward the player can expect to achieve by taking an action in the current state from the current time, t, until the player reaches the terminal state
  - If we're in some state_t, and we take some action_t, Q tells us what the *maximum* reward is when we reach the terminal state
- We can learn Q exactly or approximately
  - Exact methods are useful when state representations and actions are low-dimensional, and it doesn't take forever to reach a terminal state
  - Approximate methods are suited for high-dimensional states and actions

Q-Learning In, Like, 3 Slides

- Exact (tabular) update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \right)$$

  α – learning rate
  r_t – reward from taking action a_t while in state s_t
  γ – discount factor for rewards
  $\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$ – we pick an action at time t+1 that maximizes our future reward from that point on

- This method requires that we keep a matrix/database of states and actions and rewards

Q-Learning In, Like, 3 Slides

- Approximate update (minimize this loss):

$$L = (r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \; - Q(s_t, a_t))^2$$

- Both of the values for Q come out of our approximator
  - We feed in a state, s_t, and get future discounted reward approximations for each action, a_t
  - We take one of the actions, a_t, and our environment returns a new state, s_{t+1}, and a reward for taking that action, r_t
  - We feed in the new state, s_{t+1}, and pick the largest action that our approximator predicts (this value replaces the max)

Open AI Gym

- Install it

  pip install gym
    to get the basic environments, or
  pip install "gym[atari]"
    to get the basic environments and the Atari environments

    and if you get errors (I did), use this guy's github repo
  pip install --no-index -f https://github.com/Kojoley/atari-py/releases atari_py

    https://stackoverflow.com/questions/42605769/openai-gym-atari-on-windows

- Use it
  - There are a ton of environments you can use
  - [demo]