The agent-environment interface

- Environment
- Agent
- State
- Action
- Reward
Examples

• Pick-and-place robot
• Mars robot
• Pole-balancing robot
• Supervised learning as reinforcement learning
• Atari games
OpenAI gym environments

- CartPole
- MountainCar
- Pong
- BeamRider
- BipedalWalker
- …
Terminology & Notation

- $s_t$ - state
- $o_t$ - observation
- $a_t$ - action

$p(s_{t+1}|s_t, a_t)$

$\pi_\theta(a_t|o_t)$ - policy

$\pi_\theta(a_t|s_t)$ - policy (fully observed)

Markov property independent of $s_{t-1}$

Images: Bojarski et al. '16, NVIDIA
Policy

• A policy is the agent’s behaviour
• It is a map from state space to action space:
  • Deterministic policy
  • Stochastic policy
Goal of reinforcement learning

• Obtain a policy that maximizes the expected rewards

\[
\pi_\theta(a_t | s_t) = p(s_{t+1} | s_t, a_t)
\]

\[
\theta^* = \arg \max_\theta E_{\pi \sim p_\theta}(\tau) \left[ \sum_t r(s_t, a_t) \right]
\]

\[
\theta^* = \arg \max_\theta E_{(s,a) \sim p_\theta(s,a)}[r(s,a)]
\]

infinite horizon case

\[
\theta^* = \arg \max_\theta \sum_{t=1}^T E_{(s_t,a_t) \sim p_\theta(s_t,a_t)} [r(s_t,a_t)]
\]

finite horizon case
Evaluating the objective

\[ \theta^* = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

\[ J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \]

sum over samples from \( \pi_\theta \)
Approaches To Reinforcement Learning

• **Policy-based RL** (focus of the tutorial)
  • Search directly for the optimal policy
  • This is the policy achieving maximum future reward

• **Value-based RL** (will be discussed in brief)
  • Estimate the optimal value function $Q(s, a)$
  • This is the maximum value achievable under any policy

• **Model-based RL** (will be discussed in brief)
  • Build a model of the environment
  • Plan (e.g. by lookahead) using model
Value-based approach (in brief)

• A Q-value function is a prediction of future reward
  • “How much reward will I get from action a in state s?”

• Q-value function gives expected total reward
  • from state s and action a
  • under policy \( \pi \)
  • with discount factor \( \gamma \)

\[
Q^\pi(s, a) = \mathbb{E} [r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a]
\]

• Q-value functions decompose into a Bellman equation

\[
Q^\pi(s, a) = \mathbb{E}_{s', a'} [r + \gamma Q^\pi(s', a') \mid s, a]
\]
Optimal value function

• An optimal value function is the maximum achievable value

\[ Q^*(s, a) = \max_{\pi} Q^\pi(s, a) = Q^{\pi^*}(s, a) \]

• Once we have optimal Q-value function we can act optimally

\[ \pi^*(s) = \arg\max_a Q^*(s, a) \]
Model-based approach (in brief)
Policy-based approach

- Direct policy differentiation

\[ \theta^* = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

\[ J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)] = \int_0^T \pi_\theta(\tau) r(\tau) d\tau \]

\[ \sum_{t=1}^T r(s_t, a_t) \]

\[ \nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau = E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \]

A convenient identity

\[ \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau) \]
Direct policy differentiation

\[ \theta^* = \arg \max_{\theta} J(\theta) \]

\[ J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)] \]

\[ \nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)r(\tau)] \]

\[ \nabla_\theta \left[ \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right] \]

\[ \nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left( \sum_{t=1}^{T} r(s_t, a_t) \right) \right] \]
Evaluating the policy gradient

\[ J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \]

\[ \nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left( \sum_{t=1}^T r(s_t, a_t) \right) \right] \]

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^T r(s_{i,t}, a_{i,t}) \right) \]

\[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]

REINFORCE algorithm:
1. sample \{\tau^i\} from \pi_\theta(a_t|s_t) (run the policy)
2. \[ \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a_{i,t}^i|s_{i,t}^i) \right) \left( \sum_t r(s_{i,t}^i, a_{i,t}^i) \right) \]
3. \[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]
Reducing Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right)$$

_Causality:_ policy at time $t'$ cannot affect reward at time $t$ when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi(a_{i,t} | s_{i,t}) \left( \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right)$$

$$\hat{Q}_{i,t}$$
Baselines

\[ \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta} \log \pi_{\theta}(\tau) [r(\tau) - b] \]

\[ b = \frac{1}{N} \sum_{i=1}^{N} r(\tau) \]

- Are we allowed to do that?

\[ E[\nabla_{\theta} \log \pi_{\theta}(\tau) b] = \int \pi_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau) b \, d\tau = \int \nabla_{\theta} \pi_{\theta}(\tau) b \, d\tau = b \nabla_{\theta} \int \pi_{\theta}(\tau) \, d\tau = b \nabla_{\theta} 1 = 0 \]

- Subtracting a baseline is unbiased in expectation!
- average reward is not the best baseline, but it’s pretty good!
Policy gradient with automatic differentiation

• Pseudocode example (with discrete actions):

```python
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values - (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states)  # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)

\[
\hat{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(a_{i,t} | s_{i,t}, Q_{i,t})
\]  
q_values
```
Basics of OpenAI gym

Environment

- Has attributes that give environment specifications (e.g. action space, observation space, etc.)
- Environment `step` function gets an action and update the environment for one step
  - It returns four values, observation (i.e. observation in the next time step), reward, done (e.g. agent died!), info (e.g. it might contain the raw probabilities behind the environment’s last state change)
  - Can be rendered or restart by `render`, or `reset` functions

We implement the policy gradient method for CartPole (one of the gym environments).
References

