Install **python, tensorflow, gym** (e.g. with pip)

Download **ppo_tuto.py** from your mailbox

Approximate Policy Iteration and PPO Implementation

MDP notations

- MDP: (S, A, R, gamma, P)
  - R(s,a): reward
  - P(s’|s,a): transition proba.

- Given $Q_\pi(s,a)=E[r(s_{0,}a_0),\gamma r(s_{1,}a_1),\gamma^2 r(s_{2,}a_2),\ldots|s_0=s,a_0=a]$

  - Goal: Find $\pi^{max}=argmax_\pi J(\pi)=argmax_\pi E_{s\sim p_{0,}a\sim\pi} Q_\pi(s,a)$

Policy iteration

- Given policy q:
  - Policy evaluation step: compute $Q_q(s,a)$
  - Policy improvement step: generate new policy π s.t. $E_\pi[Q_q(s,a)] \geq E_q[Q_q(s,a)]$ for all s
    - e.g. $\pi(s) = argmax_a Q(s,a)$
- Policy improvement implies $J(\pi) \geq J(q)$

Approximate policy iteration

- For large state spaces:
  - Policy evaluation: use function approximation for Q_q(s,a)
    - Regression problem… fine
  - Policy update: use function approximation for policies
    - e.g. $\pi(a|s) = Normal(neuralnet(s), \Sigma)$
    - Cannot ensure that $E_{\pi}[Q_q(s,a)] \geq E_q[Q_q(s,a)]$ is true for all s!

Staying close to data policy

- Workaround: improve in expectation

$$E_{s\sim\pi}[E_{a\sim\pi}[Q_q(s,a)]]\propto J(\pi)$$

  - Impractical because of the expectation over the state distribution of pi
- Switch state distribution to that of q
  - $E_{s\sim q}[E_{a\sim\pi}[Q_q(s,a)]]$
  - Can guarantee improvement in J only if q and pi are close! (improve in never reached states otherwise)

Kakade et al., Approximately optimal approximate reinforcement learning, ICML 2002

API summary

- Generate data from q
- Policy evaluation: Approximate Q_q(s,a)
- Policy update: maximize $E_{s\sim q}[E_{a\sim\pi}[Q_q(s,a)]]$
  - But make sure that q and pi are close!

PPO

- Policy update is PPO's key step:
  - $L_{PPO}(\pi)=E_{s,a\sim q}[min(I(s,a)Q_q(s,a),c(I(s,a),\epsilon)Q_q(s,a))]$
  - I(s,a) = pi(a|s) / q(a|s)
  - c(x, e) = min(max(x, 1 - e), 1 + e)), clips x to [1-e, 1+e]
- $E_{s,a\sim q}[I(s,a)Q_q(s,a)]=E_{s\sim q}[E_{a\sim\pi}[Q_q(s,a)]]$
- The min and the clipping are what prevents q and pi from deviating too much from each other

Let’s implement PPO

- PPO is straightforward to implement
- Policy evaluation: can use any from the literature
- Policy update: code and optimize via gradient ascent $L_{PPO}(\pi) = E_{s,a \sim q}[min(I(s,a)Q_q(s,a), c(I(s,a),\epsilon)Q_q(s,a))]$

Three step tutorial

#0: Implement a Gaussian policy with mean given by a neural network in tensorflow

#1: Perform policy evaluation via standard MSE regression

#2: Update policy following PPO’s loss

Policy evaluation: regression

- We will use the advantage function for update
  - Let $V_\pi(s)=E_{a\sim\pi}[Q_\pi(s,a)]$ and $A_\pi(s,a)=Q_\pi(s,a)-V_\pi(s)$
- We will only learn V and estimate A from it
- Learn V as regression problem:
  - Let V(s), value given by a neural network
  - Minimize $E_{s\sim q}\left(V(s)-V_s^{target}\right)^2$
  - V_s^{target} can be the sum of rewards over one path

Policy evaluation: targets

- V_s^{target} can be the sum of rewards over one path
- V_s^{target} can be the first reward + V of next state (TD(0) method)
- V_s^{target} can be the first two rewards + V of next next state
- V_s^{target} can be an average of all such estimates (TD(lambda) method)

Policy update

- Policy update of PPO:
  - $L_{PPO}(\pi)=E_{s,a\sim q}[min(I(s,a)A_q(s,a),c(I(s,a),\epsilon)A_q(s,a))]$
  - I(s,a) = pi(a|s) / q(a|s)
  - c(x, e) = min(max(x, 1 - e), 1 + e)), clips x to [1-e, 1+e]