REINFORCEMENT LEARNING

SAMI KHAIRY
ILLINOIS TECH

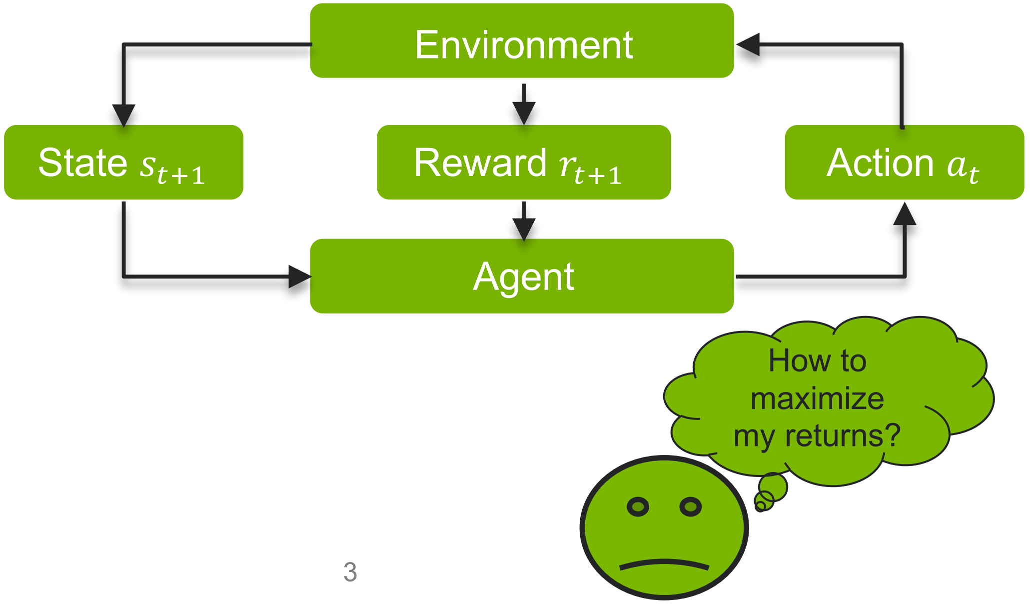
08/09/2019

OUTLINE

- Reinforcement learning, in a nutshell
- On-policy and Off-policy learning
- Q-learning
- Deep Q-Network (DQN)
- Policy Gradient Methods

WHAT IS REINFORCEMENT LEARNING?

- A branch of machine learning that attempts to formalize the nature of learning in human beings.
- An autonomous agent learns how to map situations to actions in an interactive environment, by trial and error.
- Environment gives the agent a reward signal, which guides the learning process.
- Based on the signal, agent reinforces the action, or avoids it at future encounters.



RL AS A MARKOV DECISION PROCESS

- Classical formalization of sequential decision making: actions influence immediate rewards, next states, and future rewards.
- **Def**: An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$ such that environment dynamics are Markovian, p: $\mathcal{S} x \mathcal{R} x \mathcal{S} x \mathcal{A} \rightarrow [0,1]$, $p(s', r | s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$.

- At each t, agent observes environment state S_t, selects action A_t, receives reward R_{t+1}, arrives at S_{t+1}.
- **Trajectory**: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$
- **Return at** t, $\mathrm{G_t} = \sum_{k=0}^{T} R_{t+k+1}$ or
- $\mathrm{G_t} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$.
- **Objective**: maximize long-term expected total rewards by choice of a **policy (strategy).**
- In **Model-free RL,** $p(s', r | s, a)$ are not known to the agent!

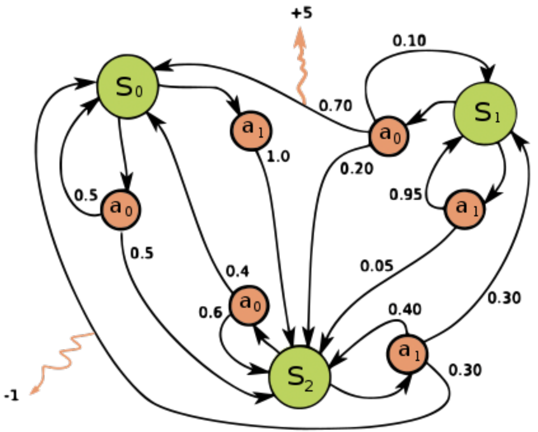


Image source: Wikipedia

RL JARGONS

- A **Policy** is a mapping from states to action probabilities, $\pi(A_t = a|S_t = s)$.
- **State-Value function** of a state s under a policy $\pi, v_\pi(s)$, is the expected return starting in s and following π thereafter, $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s], \mathbb{E}_\pi{\sim}p(r|s)$.
- **Action-Value function** of taking action a in state s under a policy π, $q_\pi(s,a)$, is the expected return starting in s, taking action a, and following π thereafter, $q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$.
- Policy π^* is optimal *iff*, $\mathrm{v}_{\pi^*}(s) \geq v_\pi(s), \forall\pi, s$.
- If agent learns the optimal $v_\pi(s)$, or $q_\pi(s,a)$, the optimal policy can be derived.
- **How to learn** $v_\pi^*(s)$, or $q_\pi^*(s,a)$ **given** $p(s', r|s,a)$ **are not known?**

  → Trade-off between exploration & exploitation.

OFF-POLICY VS ON-POLICY LEARNING

- **On-policy approach**: evaluate and improve the policy that is used to make decisions,
  - Does not reuse old data, sample inefficient, but more stable
  - Policy Gradient Algorithms: VPG, TRPO, PPO
- **Off-policy approach**: evaluate and improve a different policy from that used to generate the data,
  - Reuses old data, sample efficient, but no stability/performance guarantees
  - Q-learning, DQN, DDPG (~hybrid, actor-critic)
- **Exploration strategies,**
  - Exploring Starts: every episode starts in some state-action pair.
  - The Boltzmann approach: $\pi(a|s) > 0, \forall a, s$, e.g., $\pi(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{b \in A(s)} \exp(Q(s,b)/\tau)}$
  - ϵ-greedy: with probability ϵ select a random action at random, with probability $1 - \epsilon$ select the action with the maximal action-value.

TEMPORAL DIFFERENCE POLICY EVALUATION

- TD methods can learn directly from raw experience without a model of the environment's dynamics.
- One-step TD updates the estimated State-Value function every time step, using sample updates based on the observed reward R_{t+1} and estimate of new state $V(S_{t+1})$. Learn on-the-fly,

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

- TD estimation error, $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$.
- α is step-size parameter, e.g. $\alpha_n(a) = \frac{1}{n}$ or $\alpha(a) = \zeta$ (very small constant)

TABULAR Q-LEARNING

- Q-learning directly approximates q_* independent of the policy being followed.
- Converges with probability 1 to π^* and $q_*(s,a)$ *iff* 1) all state-action pairs are visited an infinite number of times, 2) Policy converges in the limit to be greedy w.r.t q_π.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

- Not practical when state space is continuous or large.
  – **Solution:** maintain $q_\pi(s,a)$ as parameterized functions, adjust parameters to match observed returns.
  – **Learning** $q_\pi(s,a)$ from '**samples**' involves **function approximation techniques from supervised learning** (e.g. regression, NNs).

DEEP Q-NETWORK (DQN)

- For discrete action space.
- Uses a DNN or CNN to learn the state-action value function $Q(s, a)$.
- For DNN training to converge, data samples should be *i.i.d*. This is no longer the case in reinforcement learning, since samples are temporally-correlated trajectories.
- DQN mitigates this issue by creating an experience replay buffer to store transition samples, from which a batch of samples are picked uniformly and used for training the NN.
- The network is trained with another target Q-network, which will be used to compute the loss for every action during training. Weights are slowly updated and synchronized with the primary Q-network.

POLICY GRADIENT METHODS

- Q-learning/DQN: learn the action-value function $q_\pi(s,a)$, from which the optimal policy is derived.
- In Policy Gradient Methods, adopt a parametrized policy that can select actions without consulting a state-value function (e.g., θ weights/biases of DNN),

$$\pi(a|s,\theta) = \Pr\{A_t = a|S_t = s, \theta_t = \theta\}$$

- Objective: learn the parameters $\boldsymbol{\theta}$ of $\pi(a|s,\theta)$ by maximizing a performance measure, $J(\pi_\theta)$ ~ average rate of reward, and update parameters by gradient optimization, $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k}$.

HANDS ON TUTORIAL

- Jupyter notebook: https://github.com/argonne-lcf/ATPESC_2019/blob/master/ReinforcementLearning/RL_CARTPOLE_ATPESC_2019.ipynb

USEFUL RESOURCES

- Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.
- Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971*(2015).
- Schulman, John, et al. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

- OpenAI's Baselines, Gym, Spinning Up in Deep RL, https://openai.com/resources/
- DeepMind's Tensorflow RL, https://github.com/deepmind/trfl

THANK YOU