0.1 Definitions

- Return (G_t) is the total discounted reward

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Value function: expected return starting from state s

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

- State value function: expected return starting from state s and following policy π

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) \end{aligned}$$

- Action value function: expected return starting from state s, taking action a and following policy π

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \end{aligned}$$

- Policy: distribution over actions given states

$$\pi(a|s) = \mathcal{P}[A_t = a | S_t = s]$$

- Optimal state-value function : maximum value function over all polices

$$v_*(s) = \max_\pi v_\pi(s)$$

- Optimal action-value function is maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

- Off-policy: Q-values are updated using the Q-value of the next state s' and the greedy action a'. It estimates the return for state-action pairs assuming a greedy policy were following.
- On-policy: Q-values are updated using the Q-value of the next state s' and the *current policy's* action a'. It estimates the return for state-action pairs assuming the current policy continues to be followed.
- Online learning: alternate between optimizing a policy and using that policy to collect more data

0.2 Markov Decision Processes

- describes an environment for reinforcement learning
- environment is fully observable
- the future is independent of the past given the present (e.g. $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$)
- a Markov Process is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
  - $\mathcal{S}$ is a set of states
  - $\mathcal{P}$ is a state transition probability matrix
- a Markov reward process is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{P}$ is a state transition probability matrix
  - $\mathcal{R}$ is a reward function
  - γ is a discount factor, $\gamma \in [0, 1]$
- a Markov decision process (MDP) is a Markov reward process with decisions, $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - γ is a discount factor, $\gamma \in [0, 1]$
- For any MDP, there exists an optimal policy π_* better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
- If we know $q_*(s, a)$, we immediately have the optimal policy.
- Bellman Optimality Equation is non-linear
- Many iterative solution methods:
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

0.3 Value-iteration

- it is a method of computing an **optimal MDP policy** and its value

0.4 Q-learning

- Off-policy Temporal Difference Control
- One step Q-learning:
-
$$q(s_t, a_t) = q(s_t, a_t) + \alpha[r_{t+1} + \gamma \text{max}_a q(s_{t+1}, a) - q(s_t, a_t)]$$

Algorithm 1: Q-learning

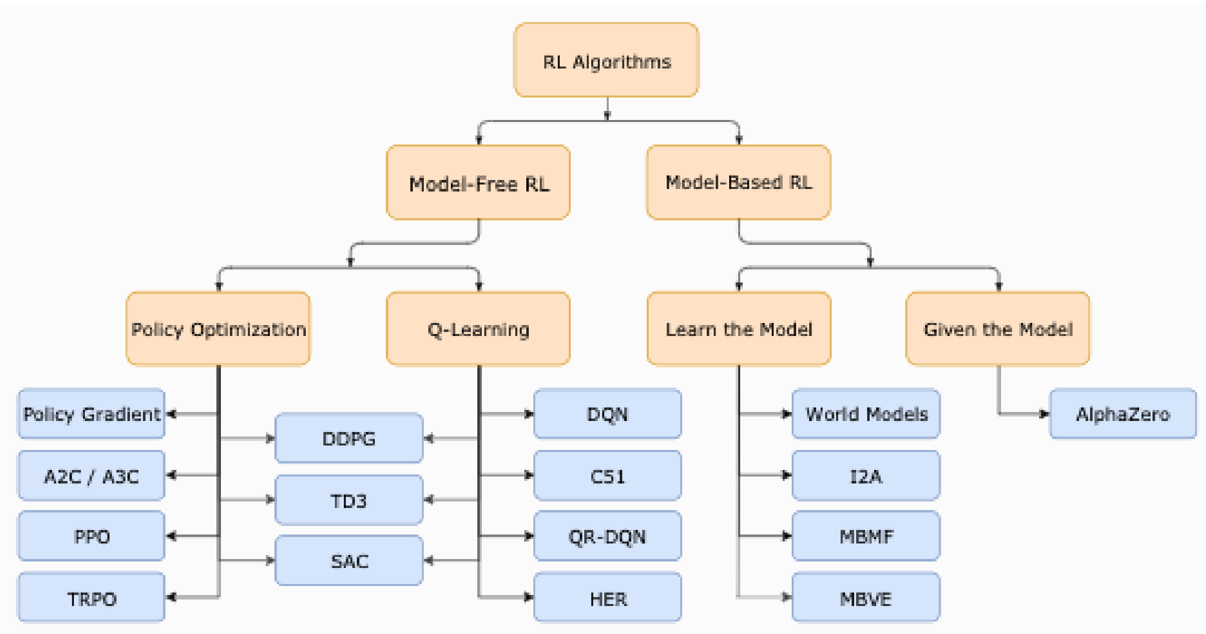
Initialize $q(s, a) \forall s \in \mathcal{S}, \forall s \in \mathcal{A}, q(\text{terminal-state}) = 0$;
for *each episode* **do**
  Initialize s;
  for *each step in episode* **do**
    Choose a from s using policy derived from q (e.g. ϵ-greedy);
    Take action a and observe r, s';
    $q(s, a) = q(s, a) + \alpha[r + \gamma \text{max}_a q(s', a) - q(s, a)]$;
    $s = s'$;
  end
end

0.5 Hindsight Experience Replay

0.6 RL Algorithms

0.6.1 Taxonomy

Figure 1: Taxonomy of RL Algorithms



-

- Figure from OpenAI SpinningUp
- Does the agent have access to a model of the environment?
- Having a model alows the agent to plan by thinking ahead
- Ground-truth model is not always available to the agent, must learn model from experience
- In model-free there are two main approaches for training agents: policy optimization and q-learning
- Policy optimization
  - Represents the policy as $\pi_\theta(a|s)$ and optimize the parameters θ by gradient ascent on $J(\pi_\theta$
  - The optimization is almost always performed on-policy
  - Also learn an approximator $V_\phi(s)$
  - Directly optimizes for the thing that you want**
  - Q-learning learns an approximator $Q_\theta(s, a)$ for the optimal action-value function
  - Q-learning is almost always performed off-policy, update uses data collected at any point during training
  - $a(s) = \text{argmax}_a Q_\theta(s, a)$
  - Indirectly optimization, tens to be less stable, more sample efficient because they can reuse data more effectively

0.6.2 Policy Gradients

- Maximize the expected return $J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$
- Optimize policy: $\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|_{\theta_k}$
- Gradient of policy performance $\nabla_\theta J(\pi_\theta)$ is the policy gradient
- $\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \text{log}\pi_\theta(a_t|s_t)R(\tau)]$
- This is the grad log probability and since it is an expectation, we can estimate it with a sample mean by collecting a set of trajectories
- The "loss" for policy gradient algorithm is not the same loss in the typical sensen from supervised learning
- Loss doesn't measure performance, it doesn't mean anything. It is possible to have low loss and have poor policy performance, this is due to policy overfitting to a batch of data
- EGLP or Expected Grad-Log-Prob Lemma states that $E_{x\sim P_\theta}[\nabla_\theta log P_\theta(x)] = 0$
- Reward-to-go policy gradient, we only care about the reward that came after taking an action

- Past rewards add noise to sample estimates of the policy gradient
- EGLP lemme implies that we can add/subtract any function b that only depends on the state from the policy gradient without changing its expectation (this is called a baseline)
- Results in faster and more stable policy learning
- Common baselines:
  - $\Phi_t = R(\tau)$
  - $\Phi_t = \Sigma_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$
  - $\Phi_t = \Sigma_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$
  - $\Phi_t = Q^{\pi_\theta}(s_t, a_t)$ (on-policy action-value function)
  - $\Phi_t = A^{\pi}(s_t, a_t)$ (advantage function, describes how much better or worse an action is compared to other actions)
- Baseline is often approximated using a neural network that is updated concurrently wiht the policy using a mean-squared error

0.6.3 Deep Deterministic Policy Gradient (DDPG)

-

0.7 Batch/Offline Reinforcement Learning

- Most of RL algorithms assume that an agent interacts with an online environment or simulator and learns from its own collected experience. This is expensive and often requires a high-fidelity simulator which can be hard to build
- Offline RL addresses the problem of learning a policy from a fix set of trajectories, without any further interaction with the environment
- In principle, off-policy algorithm can learn from data collected by any policy
- Recent work shows that standard off-policy deep RL algorithms diverge in offline setting
- Removes design of replay buffer and exploration
- Challenging due to distribution mismatch between current policy and offline data collection policy
- Datasets:
  - D4RL
  - Developed tasks that reflect both real-world dataset challenges and real-world applications
  - Autonomous driving, robotics, and other domains
- Batch RL algorithms

  - Quantile Regression DQN (QR-DQN)
  - Random Ensemble Mixture REM
  - Batch Constrained Deep Q-learning (BCQ)
- Batch Constrained deep Q-learning (BCQ) [**bcq**]
  - Run normal Q-learning but in the maximization step, instead of considering max over all possible actions, only consider actions a' such that (s', a') actually appear in the batch of data
  - Train a *generative model* - variational autoencoder - to generate actions that are likely to be from the batch
  - Also a *perturbation model* to perturb the actions

0.8 Curiosity Driven Exploration

0.9 Inverse Reinforcement Learning