RL
Parametric Approximation Approaches

Q-Learning variant of Temporal Difference Learning

The goal of RL is to find a policy π that maps from the set of states S to actions A so as to maximize the expected total discounted future reward

$$V^{\pi}(s) = E^{\pi}\Big(\sum_{t=0}^{\infty} \gamma^t r_t\Big) \tag{1}$$

Alternatively, the values can be parametrized by state and action pairs, denoted by $Q^{\pi}(s, a)$.

$$Q^*(s,a) = \sum_r r P(r|s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q^*(s', a') \tag{2}$$

Temporal difference learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_Q \tag{3}$$

$$\delta_Q = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \tag{4}$$

Model-free Q-learning action selection

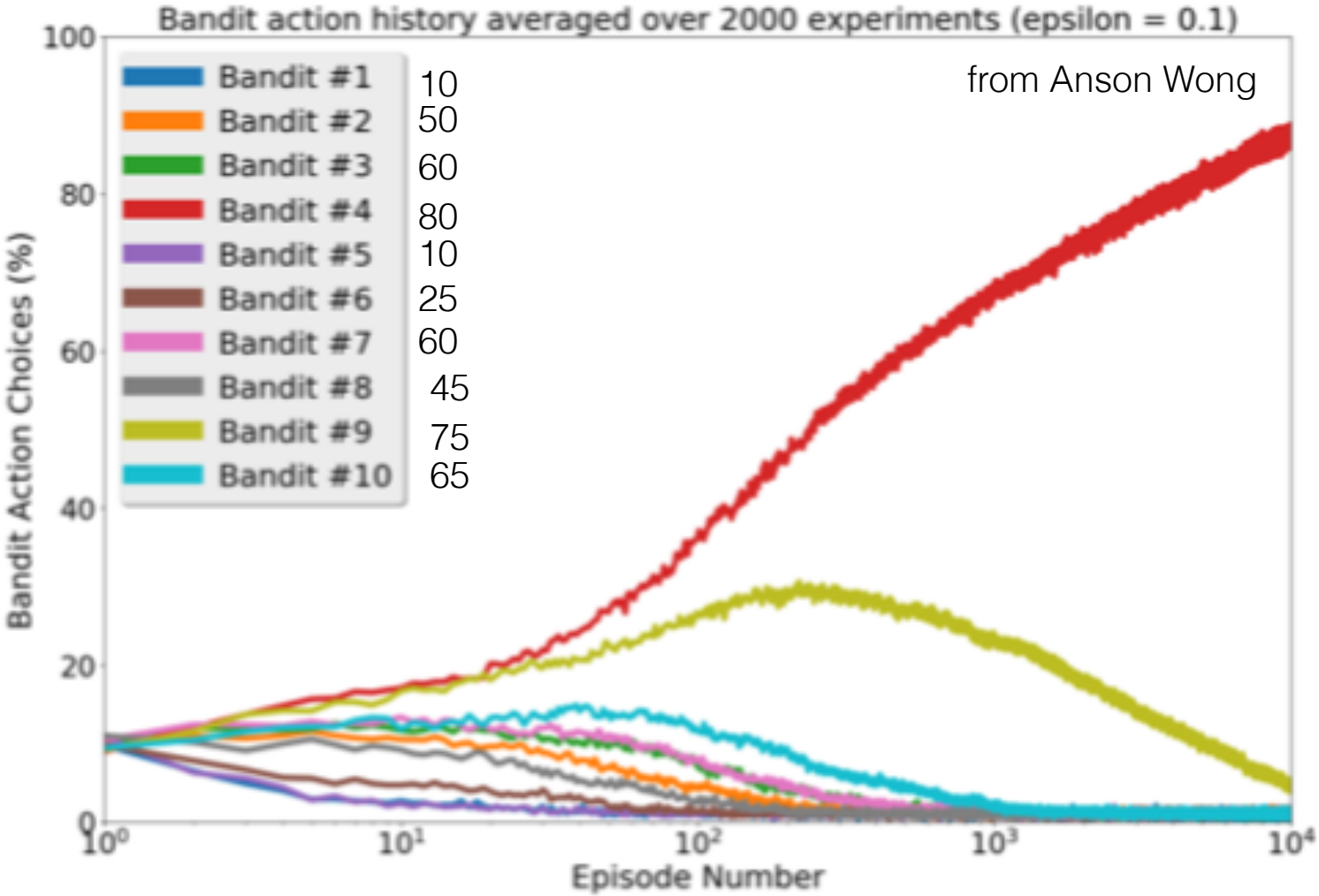


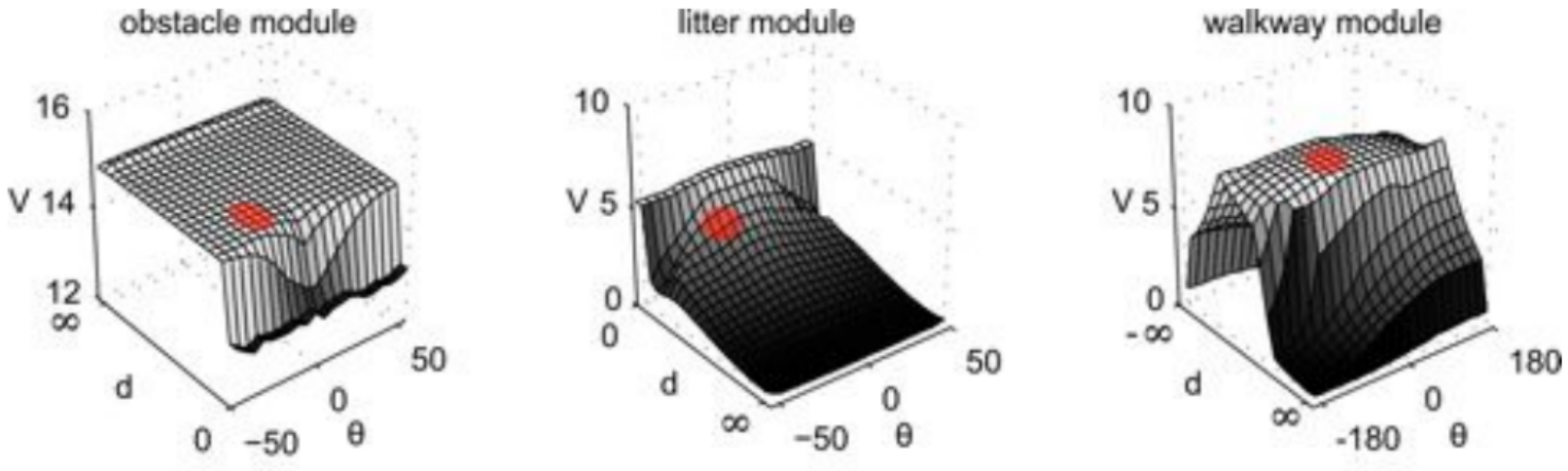
Fig 1) Bandit choices by the epsilon-greedy agent (epsilon = 10%) throughout its training

1. In model-free Q learning, keep sampling actions, sampling the actions with the most rewards more often using the epsilon greedy protocol

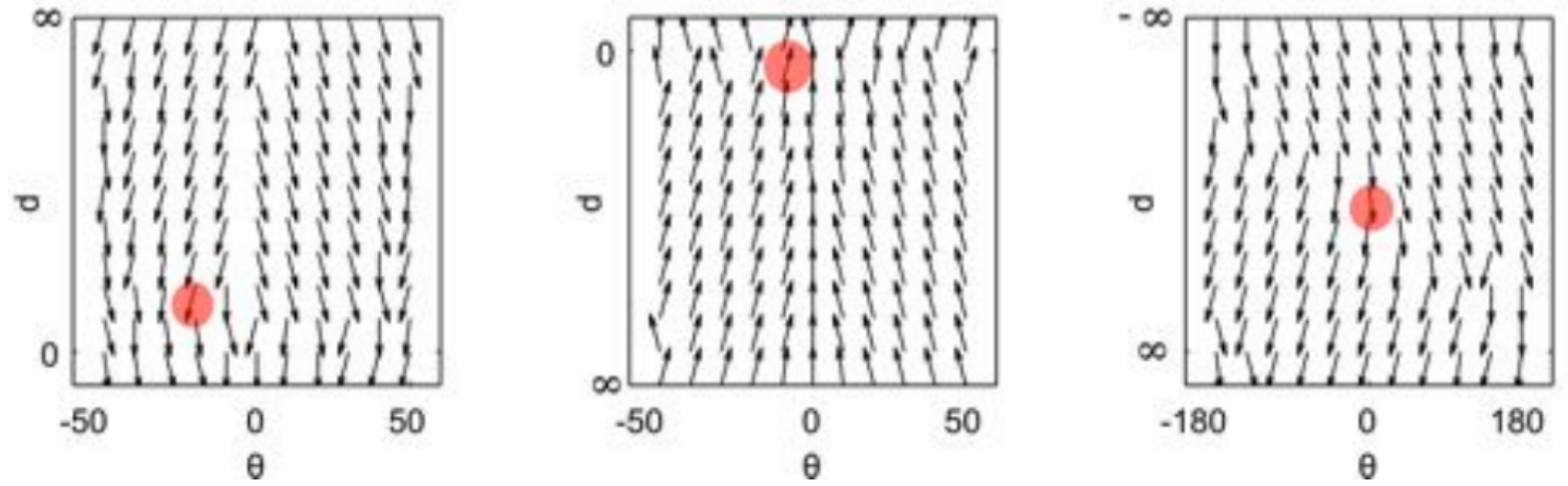
2. sample sequences of a selected length

Value and Policy spaces can be approximated with smooth functions that space the state space, making search more efficient

Value



Policy



A straightforward implementation of RL uses tables of digitized states to save the policies and the values. But this is very expensive. Since the policies and values are smooth functions of the state space, we can do better.

Value Function Approximation via Stochastic Gradient Descent

The main idea is that since the state space is expensive to represent in the form of a discrete table, and the value function V(s) is smooth, we can compress it by choosing a parametric form V_θ e.g.

$$V_\theta = f(x_1, x_2, \theta)$$

Value function learning via bootstrapping in T-D learning

When we use the transition function from state s_i we have to obey the Bellman operator constraint

$$\hat{T}^{\pi} V(s_i) = r_i + \gamma V(s_{i+1})$$

The goal is to fit $\hat{V}_{\theta}$ to the optimal value function

$$J_{V^{\pi}}(\theta) = \|V^{\pi} - \hat{V}_{\theta}\|^2$$

We can express this in terms of samples. So for one sample …

$$\hat{J}_{V^{\pi}}(\theta) = \sum_{j} \left(v_j^{\pi} - \hat{V}_{\theta}(s_j) \right)^2$$

Lots of samples:

$$\hat{J}_{V^\pi}(\theta) = \sum_j \left(v_j^\pi - \hat{V}_\theta(s_j)\right)^2$$

Find the parameters θ by gradient descent

$$\Delta\theta = -\frac{\alpha}{2}\nabla_\theta \left(v_i^\pi - \hat{V}_\theta(s_i)\right)^2$$

$$= +\alpha \left(\nabla_\theta \hat{V}_\theta(s_i)\right)\left(v_i^\pi - \hat{V}_\theta(s_i)\right)$$

$$\Delta\theta = \quad \alpha \left(\nabla_\theta \quad \hat{V}_\theta(s_i)\right) \left(\hat{T}^\pi \hat{V}_\theta \quad (s_i) - \hat{V}_\theta \quad (s_i)\right)$$

$$\Delta\theta = \quad \alpha \left(\nabla_\theta \quad \hat{V}_\theta(s_i)\right) \left(r_i + \gamma \hat{V}_\theta \quad (s_{i+1}) - \hat{V}_\theta \quad (s_i)\right)$$

Exam Question

[10] To reduce the cost of high-dimensional search spaces one can approximate the value function with low degree-of-freedom parameterized functions. Suppose that the value function can be written as

$$V(s) = \sum_{i=1}^{n} \exp(\theta_i - b(s_i))^2$$

where the functions $b_i(s)$ are known. The bootstrapping value function approximation method attempts to choose the θ_i so as to minimize the difference of samples $\hat{V}$,

$$(\hat{V}(s_{t-1}) - (r_t + \gamma \hat{V}(s_t))^2$$

so how this can be done by computing the appropriate derivative.

Answer

Policy Gradient Approximation

(From John Shulman's dissertation)

Markov Decision Process

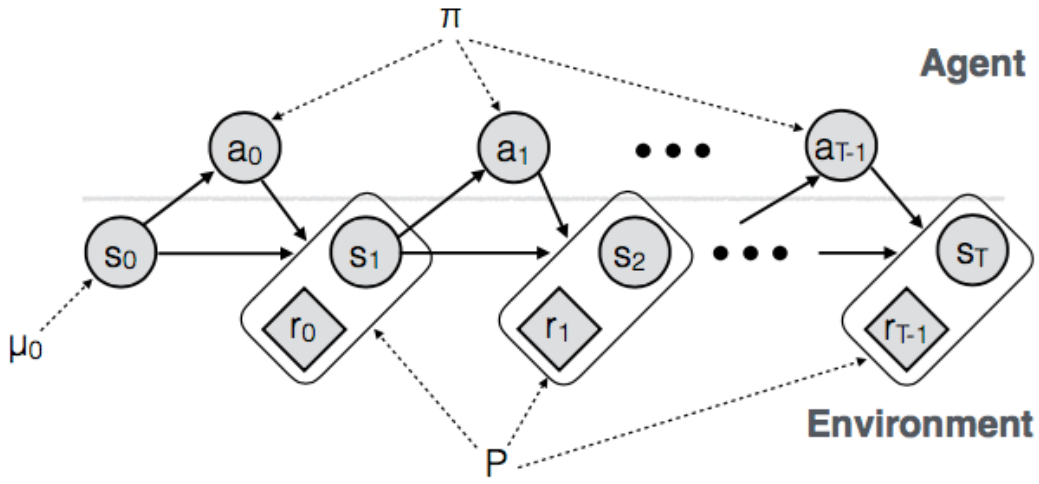
- $\mathcal{S}$: **state space**, a set of states of the environment.
- $\mathcal{A}$: **action space**, a set of actions, which the agent selects from at each timestep.
- $P(r, s' \mid s, a)$: a transition probability distribution. For each state s and action a, P specifies the probability that the environment will emit reward r and transition to state s'.

In certain problem settings, we will also be concerned with an **initial state distribution** $\mu(s)$, which is the probability distribution that the initial state s_0 is sampled from.

Derivative Free Optimization

Derivative free RL algorithms work by perturbing the policy and accepting the result if it leads to greater reward

$$
\begin{aligned}
s_0 &\sim \mu(s_0) \\
a_0 &\sim \pi(a_0 \mid s_0) \\
s_1, r_0 &\sim P(s_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid s_1) \\
s_2, r_1 &\sim P(s_2, r_1 \mid s_1, a_1) \\
&\dots \\
a_{T-1} &\sim \pi(a_{T-1} \mid s_{T-1}) \\
s_T, r_{T-1} &\sim P(s_T \mid s_{T-1}, a_{T-1})
\end{aligned}
$$

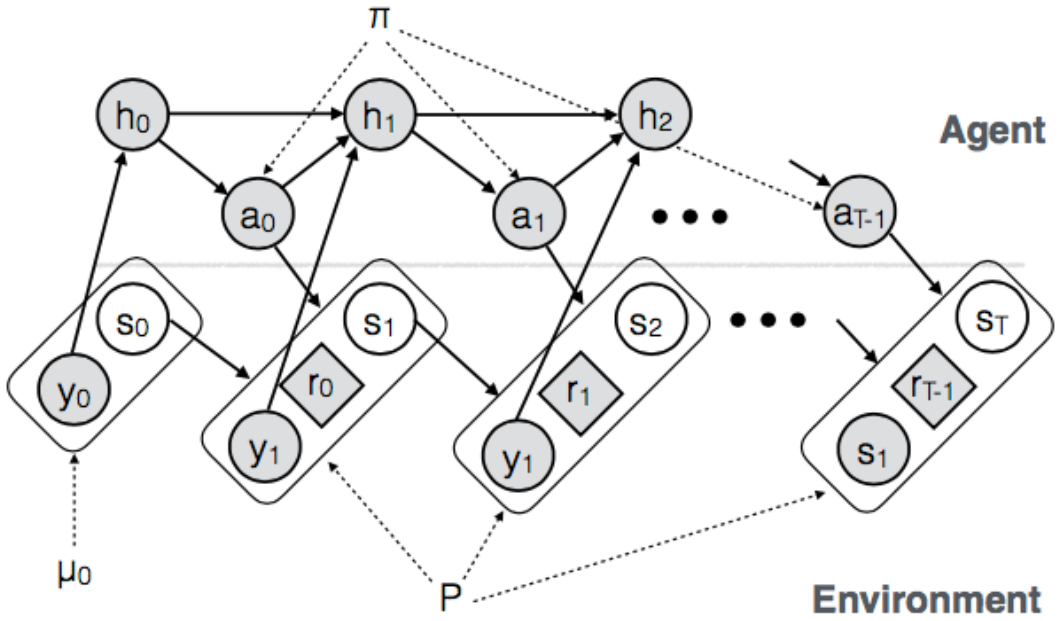


The goal is to find a policy π that optimizes the expected total reward per episode.

$$\underset{\pi}{\text{maximize}}\, E_\tau[R \mid \pi]$$

$$\text{where } R = r_0 + r_1 + \cdots + r_{\text{length}(\tau)-1}$$

$$
\begin{aligned}
s_0, y_0 &\sim \mu_0 \\
a_0 &\sim \pi(a_0 \mid h_0) \\
s_1, y_1, r_0 &\sim P(s_1, y_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid h_1) \\
s_2, y_2, r_1 &\sim P(s_2, y_2, r_1 \mid s_1, a_1) \\
&\cdots \\
a_{T-1} &\sim \pi(a_{T-1} \mid h_{T-1}) \\
s_T, y_T, r_{T-1} &\sim P(s_T, y_T, r_{T-1} \mid s_{T-1}, a_{T-1})
\end{aligned}
$$



This process is called a partially observed Markov decision process (POMDP). The partially-observed setting is equivalent to the fully-observed setting because we can call the observation history h_t the state of the system. That is, *a POMDP can be written as an MDP* (with infinite state space). When using function approximation, the partially observed setting is not much different conceptually from the fully-observed setting.

Cross Entropy: A simple and effective DFO method

Algorithm 1 Cross Entropy Method

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
for iteration $= 1, 2, \ldots$ **do**
  Collect n samples of $\theta_i \sim N(\mu, \text{diag}(\sigma))$
  Perform one episode with each θ_i, obtaining reward R_i
  Select the top p% of samples (e.g. $p = 20$), which we'll call the **elite set**
  Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new μ, σ.
end for
Return the final μ.

Policy Gradients

Policy gradient methods work by repeatedly estimating the gradient of the policy's performance with respect to its parameters.

$$\nabla_\theta E_x[f(x)] = E_x[\nabla_\theta \log p(x \mid \theta) f(x)].$$

This equation can be derived by writing the expectation as an integral:

$$\begin{aligned}\nabla_\theta E_x[f(x)] &= \nabla_\theta \int dx\, p(x \mid \theta) f(x) = \int dx\, \nabla_\theta p(x \mid \theta) f(x) \\ &= \int dx\, p(x \mid \theta) \nabla_\theta \log p(x \mid \theta) f(x) = E_x[f(x) \nabla_\theta \log p(x \mid \theta)].\end{aligned}$$

$$\hat{g} = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log p(x_i \mid \theta) f(x_i)$$

$$\Delta\theta = \epsilon \hat{g}$$

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau[\nabla_\theta \log p(\tau \mid \theta) R(\tau)]$$

$$\begin{aligned} p(\tau \mid \theta) =& \mu(s_0)\pi(a_0 \mid s_0, \theta)P(s_1, r_0 \mid s_0, a_0)\pi(a_1 \mid s_1, \theta) \\ & P(s_2, r_1 \mid s_1, a_1) \ldots \pi(a_{T-1} \mid s_{T-1}, \theta)P(s_T, r_{T-1} \mid s_{T-1}, a_{T-1}), \end{aligned}$$

where μ is the initial state distribution. When we take the logarithm, the product turns into a sum, and when we differentiate with respect to θ, the terms $P(s_t \mid s_{t-1}, a_{t-1})$ terms drop out as does $\mu(s_0)$. We obtain

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) R(\tau) \right]$$

One t sample:

$$\nabla_\theta E_\tau[r_t] = E_\tau\left[\sum_{t'=0}^{t} \nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta) r_t\right]$$

$$\nabla_\theta E_\tau[R(\tau)] = E_\tau\left[\sum_{t=0}^{T-1} r_t \sum_{t'=0}^{t} \nabla_\theta \log \pi(a_{t'} \mid s_{t'}, \theta)\right]$$
$$= E_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'}\right].$$

Algorithm 2 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta) \hat{A}_t$
end for

$$V^\pi(s) = \mathbb{E}\left[r_t + r_{t+1} + \cdots + r_{T-1} \mid s_t = s,\ a_{t:(T-1)} \sim \pi\right]$$

$$b(s) \approx V^\pi(s)$$