Bandit Problem
The greedy method performs significantly worse in the long run because it often gets stuck performing suboptimal actions. (Exploitation vs. exploration dilemma)
Softmax (choose action a at t-th play with prob:)

$$\frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{n} e^{Q_t(i)/\tau}}$$

NewEstimate = OldE + StepSize[Target - OldE]
StepSize never too small or large, constant for non stationary problem(weight recent)
Markov Reward Process (MRP): a Markov process with reward values $< S, \mathscr{P}, \mathscr{R}, \gamma >$

$$\mathscr{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$$

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

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Bellman Equation for MRP

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v\left(S_{t+1}\right) \mid S_t = s\right]$$

$$v(s) = \mathscr{R}_s + \gamma \sum_{s' \in \mathscr{S}} \mathscr{P}_{ss'} v\left(s'\right)$$

Markov Decision Process (MDP): MRP with decisions, a tuple $< S, A, \mathscr{P}, \mathscr{R}, \gamma >$

$$\pi(a \mid s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

$$v_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$$

$$q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$$

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

$$\pi_*(a \mid s) = \begin{cases} 1 \text{ if } a = \operatorname{argmax}_{a \in \mathscr{A}} q_*(s, a) \\ 0 \text{ otherwise} \end{cases}$$

Bellman Expectation Equation for MDP

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s\right]$$

$$q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right]$$

Bellman Optimality Equation

$$v_*(s) = \max_a q_*(s, a)$$

$$q_*(s, a) = \mathscr{R}_s^a + \gamma \sum_{s' \in \mathscr{S}} \mathscr{P}_{ss'}^a v_*\left(s'\right)$$

Model-based RL
Dynamic Programming (DP) (Full Backup)
Iterative Policy Evaluation / Prediction

$$v_{k+1}(s) = \sum_{a \in \mathscr{A}} \pi(a \mid s)\left(\mathscr{R}_s^a + \gamma \sum_{s' \in \mathscr{S}} \mathscr{P}_{ss'}^a v_k\left(s'\right)\right)$$

Policy improvement: $\pi' = \text{greedy}\left(v_\pi\right)$
Policy Iteration / Control
Iterative policy evaluation & Greedy policy improve.
Converge to v_*, π_*.
Generalized Policy Iteration (GPI)
Any policy evaluation + Any policy improvement.
Converge to v_*, π_*.
Value Iteration / Control

$$v_{k+1}(s) = \max_{a \in \mathscr{A}}\left(\mathscr{R}_s^a + \gamma \sum_{s' \in \mathscr{S}} \mathscr{P}_{ss'}^a v_k\left(s'\right)\right)$$

Model-free RL
Monte Carlo RL (Sample Backup)
MC Policy Evaluation / Prediction
empirical mean return instead of expected return
First Visit MC Policy Evaluation, Every Visit MC Policy Evaluation. (in an episode)

$$N(s) \leftarrow N(s) + 1 \qquad S(s) \leftarrow S(s) + G_t$$

$$V(s) = S(s)/N(s)$$

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t - V\left(S_t\right)\right)$$

Incremental MC Updates: Update after each episode. Constant step useful in non-stationary
MC Policy Improvement
Greedy policy improvement over V(s) needs model, over Q(s,a) is model free.
MC Policy Iteration / Control
MC policy evaluation, $Q = q_\pi$ & ϵ-greedy policy improvement. Converge to q_*, π_*
GLIE MC Policy Iteration
Evaluation:

$$N\left(S_t, A_t\right) \leftarrow N\left(S_t, A_t\right) + 1$$

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \frac{1}{N\left(S_t, A_t\right)}\left(G_t - Q\left(S_t, A_t\right)\right)$$

Improvement:
$\epsilon \leftarrow 1/k$
$\pi \leftarrow \epsilon - \text{greedy}(Q)$ Converge to $q_*(s, a)$
On policy: learn π from experience sampled from π
Off policy: learn π from experience sampled from μ
Temporal Difference (TD) (Sample Backup)
TD Prediction

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right)$$

Learn online every step. Low variance, some bias.

$$\hat{\mathscr{P}}_{s,s'}^a = N(s, a, s')/N(s, a) \quad \hat{\mathscr{R}}_s^a = \sum r_s^a/N(s, a)$$

TD Control: Prediction & e-greedy
Sarsa (On-Policy) Prediction

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma Q\left(S', A'\right) - Q(S, A)\right)$$

Q-Learning (Off-Policy) Prediction

$$Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma \max_{a'} Q\left(S', a'\right) - Q(S, A)\right)$$

TD(λ) Prediction

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V\left(S_{t+n}\right)$$

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t^{(n)} - V\left(S_t\right)\right)$$

Forward-view TD($\boldsymbol{\lambda}$) (TD(1) is MC)

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

$$V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t^\lambda - V\left(S_t\right)\right)$$

Eligibility Trace

$$E_0(s) = 0$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + \mathbf{1}\left(S_t = s\right)$$

Backward-view TD($\boldsymbol{\lambda}$)

$$\delta_t = R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)$$

$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

Off-line: Updates accumulating within episode, apply in batch at end of episode. The sum of offline updates (forward = backward)

$$\sum_{t=1}^{T}\alpha\delta_t E_t(s) = \sum_{t=1}^{T}\alpha\left(G_t^\lambda - V\left(S_t\right)\right)\mathbf{1}\left(S_t = s\right)$$

On-line: Update every time step. TD(0) (forward = backward), TD($\boldsymbol{\lambda}$) (forward != backward).

Sarsa($\boldsymbol{\lambda}$) Control

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1}R_{t+n} + \gamma^n Q\left(S_{t+n}, A_{t+n}\right)$$

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q_t^{(n)} - Q\left(S_t, A_t\right)\right)$$

Forward-view Sarsa($\boldsymbol{\lambda}$)

$$q_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}q_t^{(n)}$$

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q_t^\lambda - Q\left(S_t, A_t\right)\right)$$

Eligibility Trace for state-action pair

$$E_0(s,a) = 0$$

$$E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}\left(S_t = s, A_t = a\right)$$

Backward-view Sarsa($\boldsymbol{\lambda}$)

$$\delta_t = R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

Q($\boldsymbol{\lambda}$) Control

$$R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n \max_a Q_t\left(S_{t+n}, a\right)$$

$$E_t(s,a) = \mathbf{1}\left(S_t = s, A_t = a\right) + \gamma\lambda E_{t-1}(s,a) \times \mathbf{1}\left(Q_{t-1}\left(S_t, A_t\right) = \max_a Q_{t-1}\left(S_t, a\right)\right)$$

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q_t\left(S_{t+1}, a'\right) - Q_t\left(S_t, A_t\right)$$

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t E_t(s,a)$$

Function Approximation

Motivation: curse of dimensionality. Solution: Estimate value function with function approximation

$$\nabla_{\mathrm{w}} J(\mathrm{w}) = \left(\frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_1}, \ldots, \frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_n}\right)^T$$

$$\Delta\mathbf{w} = -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w})$$

Stochastic Gradient Descent

Find $\mathbf{w}$ to min $J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(S) - \hat{v}(S,\mathbf{w})\right)^2\right]$

GD: $\Delta\mathbf{w} = \alpha\mathbb{E}_\pi\left[\left(v_\pi(S) - \hat{v}(S,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})\right]$

SGD: $\Delta\mathbf{w} = \alpha\left(v_\pi(S) - \hat{v}(S,\mathbf{w})\right)\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$

RL Prediction with Value Approximation

MC: $\Delta\mathbf{w} = \alpha\left(G_t - \hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$

TD(0): $\Delta\mathbf{w} = \alpha\left(R_{t+1} + \gamma\hat{v}\left(S_{t+1},\mathbf{w}\right) - \hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$

Forward TD(λ): $\Delta\mathbf{w} = \alpha\left(G_t^\lambda - \hat{v}\left(S_t,\mathbf{w}\right)\right)\nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$

Backward TD(λ):

$$\delta_t = R_{t+1} + \gamma\hat{v}\left(S_{t+1},\mathbf{w}\right) - \hat{v}\left(S_t,\mathbf{w}\right)$$

$$E_t = \gamma\lambda E_{t-1} + \nabla_{\mathbf{w}}\hat{v}\left(S_t,\mathbf{w}\right)$$

$$\Delta\mathbf{w} = \alpha\delta_t E_t$$

Feature vector: $\mathbf{x}(S) = (\mathbf{x}_1(S), \ldots, \mathbf{x}_n(S))^T$

$\hat{v}(S,\mathbf{w}) = \mathbf{x}(S)^\top\mathbf{w} \quad J(\mathbf{w}) = \mathbb{E}_\pi\left[\left(v_\pi(S) - \mathbf{x}(S)^\top\mathbf{w}\right)^2\right]$

$\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w}) = \mathbf{x}(S) \quad \Delta\mathbf{w} = \alpha\left(v_\pi(S) - \hat{v}(S,\mathbf{w})\right)\mathbf{x}(S)$

RL Control with Value Function Approximation

Approximate policy evaluation $\hat{q}(S,A,\mathbf{w}) \approx q_\pi(S,A)$ & e-greedy policy improvement

Policy Gradient ($\pi_\theta(s,a) = \mathbb{P}[a \mid s, \theta]$)

Policy Objective Function (find θ maximize $J(\theta)$)

Episodic: $J_1(\theta) = V^{\pi_\theta}\left(s_1\right) = \mathbb{E}_{\pi_\theta}\left[v_1\right]$ Start value

Continuing: $J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s)V^{\pi_\theta}(s)$ Ave value

$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s)\sum_a \pi_\theta(s,a)\mathcal{R}_s^a$ Ave reward

per time-step $d^{\pi_\theta}(s) = P^{\pi_\theta}(s \mid \theta)$

Likelihood ratios:

$$\nabla_\theta\pi_\theta(s,a) = \pi_\theta(s,a)\frac{\nabla_\theta\pi_\theta(s,a)}{\pi_\theta(s,a)}$$

$$= \pi_\theta(s,a)\nabla_\theta\log\pi_\theta(s,a)$$

Score function: $\nabla_\theta\log\pi_\theta(s,a)$

Softmax Policy

$$\pi_\theta(s,a) = p(a \mid s,\theta) = \frac{e^{\phi(s,a)^T\theta}}{\sum_a e^{\phi(s,a)^T\theta}}$$

$$\nabla_\theta\log\pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$$

Gaussian Policy $a \sim \mathcal{N}\left(\mu(s), \sigma^2\right)$

$$\nabla_\theta\log\pi_\theta(s,a) = \frac{(a-\mu(s))\phi(s)}{\sigma^2}$$

Generalized MDP (multi-step MDPs)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta\log\pi_\theta(s,a)Q^{\pi_\theta}(s,a)\right]$$

One-step MDP

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$
$$= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\mathcal{R}_{s,a}$$
$$\nabla_\theta J(\theta) = \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_{s,a}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]$$

Monte-Carlo Policy Gradient (REINFORCE)

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta\left(s_t, a_t\right) G_t$$

REINFORCE has variance

Actor-Critic Methods

Use a critic to estimate the action-value function

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

Critic: Update action-value function parameter w

Actor: Update policy parameter θ, in direction suggested by critic.

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$$

Approximating policy gradient introduce bias

Reducing Variance Using a Baseline

$$\nabla_\theta \log \pi_\theta(s,a) B(s) \quad B(s) = V^{\pi_\theta}(s)$$

A good baseline is the state value function.
Rewrite policy gradient using advantage function

$$A_\theta^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]$$

Estimating Advantage Function

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$
$$A(s,a) = Q_w(s,a) - V_v(s)$$

Using TD error

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\delta^{\pi_\theta}\right]$$

Policy gradient many forms

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) v_t\right] \quad \text{REINFORCE}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^w(s,a)\right] \quad \text{Q Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^w(s,a)\right] \quad \text{Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\delta\right] \quad \text{TD Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\delta e\right] \quad TD(\lambda) \text{ Actor-Critic}$$

Upper Confidence Bounds (UCB)

$$\text{UCB1} = \bar{X}_j + C\sqrt{\frac{2\ln n}{n_j}}$$

X_j is estimated reward of choice j

n is number of all plays done so far

n_j is number of times choice j has been tried

Most urgent node has the highest UCB

Monte Carlo Tree Search (MCTS)

A method for finding optimal decisions by taking random samples and building a search tree according to the results.
1.Selection (Tree Descent) until hit tree boundary
2.Expansion 3.Simulation 4.Back-Propagation

Model Learning

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}\left(S_t, A_t, S_{t+1} = s, a, s'\right)$$
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}\left(S_t, A_t = s, a\right) R_t$$

Model-free RL

No model, learn from experience

Model-based RL

Model is known, solve MDP (Policy iteration, Value iteration, MCTS)

Sample-based Planning

Learn a model from real experience. Use model only to generate samples. And apply model-free RL (MC, Sarsa, Q-Learning) on simulated experience.

Dyna

Learn a model from real experience. Use model only to generate samples. And apply model-free RL on real and simulated experience.

Deep Reinforcement Learning

Deep RL: Data is sequential, Algorithm unstable, high correlation between Q-value and target, scale of reward and Q-value unknown.

DQN: Stable (use experience replay, freeze target Q-network, clip reward or normalize network)

Algorithm:

Take action a_t according to ϵ -greedy policy

Store transition $\left(s_t, a_t, r_{t+1}, s_{t+1}\right)$ in replay memory $\mathcal{D}$

Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$

Compute Q-learning targets w.r.t. old, fixed parameters w^-

Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i\left(w_i\right) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; w_i^-\right) - Q\left(s, a; w_i\right)\right)^2\right]$$

Using variant of stochastic gradient descent

Deterministic Policy Gradient (DPG)

DPG may NOT explore full state and action space in continuing situation

Deep Deterministic Policy Gradient (DDPG)

Use experience replay for both actor and critic
Use target Q-network to avoid oscillations

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\left(r + \gamma Q\left(s', \pi\left(s'\right), w^-\right) - Q(s,a,w)\right)\frac{\partial Q(s,a,w)}{\partial w}\right]$$
$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\frac{\partial Q(s,a,w)}{\partial a}\frac{\partial \pi(s,u)}{\partial u}\right]$$

Asynchronous Advantage Actor-Critic (A3C)

A3C uses a global network and multiple agents/ workers where each has its own set of network parameters. Each agent interacts with its own local environment and update the global parameter in an asynchronous manner

In policy gradient, we use advantage function to improve the policy update.
Use Actor-critic architecture.
<u>Why A3C?</u>
The samples an agent gathers are highly correlated, leading to unstable algorithms if nonlinear function approximation (e.g. deep NN) applied.
In DQN & DDPG, we use experience replay to overcome this issue.
A3C runs several agents in parallel, each with its own copy of the environment, and use their samples for updates. Different agents will likely experience different states and transitions, thus avoiding the correlation.
A3C needs much less memory, i.e. no need to store the samples for experience replay.