Executive Summary

- Markov Decision Process
  - Markov Chain $\langle \mathcal{S}, \mathcal{P} \rangle$
  - Markov Reward Process $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
    - MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ + policy π = MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
      - $\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathcal{P}^a_{ss'}$
      - $\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathcal{R}^a_s$
  - Bellman Expectation Equations
  - Bellman Optimality Equations
- Planning (*Lec 3*)
  - Synchronous DP
    - Policy Iteration
      - Policy Evaluation: estimate $v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right]$
        - Iterative PE: $\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$
      - Policy Improvement: generate $\pi' \geq \pi$
        - Greedy PI: $\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a)$
    - Value Iteration: $v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_*\left(s'\right)$
  - Asynchronous DP
    - In-place DP
    - Prioritised Sweeping
    - Real-time DP
  - Approximate DP (using function approximation)
    - Fitted Value Iteration: $\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \hat{v}\left(s', \mathbf{w_k}\right)\right)$ for $s \in \tilde{S} \subseteq \mathcal{S}$
      - train $\hat{v}\left(\cdot, \mathbf{w}_{\mathbf{k}+1}\right)$ using targets $\left\{\langle s, \tilde{v}_k(s)\rangle\right\}$
  - Contraction Mapping
- Reinforcement Learning
  - Model Free
    - Prediction (*Lec 4*)
      - Monte Carlo Policy Evaluation: $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t - V\left(S_t\right)\right)$
      - TD Learning
        - TD(0): $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right)$
        - Forward TD(λ): $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G^\lambda_t - V\left(S_t\right)\right)$
        - Backward TD(λ): $V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$
          - $E_t(s) = \gamma \lambda E_{t-1}(s) + \mathbf{1}\left(S_t = s\right)$
    - Control (*Lec 5*)
      - On-Policy
        - Monte-Carlo Policy Iteration: $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(G_t - Q\left(S_t, A_t\right)\right)$
          - GLIE Monte-Carlo Control
        - SARSA: On-Policy TD Learning based Policy Iteration:
          $Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma Q\left(S', A'\right) - Q(S, A)\right)$
          - Forward Sarsa(λ): $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q^\lambda_t - Q\left(S_t, A_t\right)\right)$
          - Backward Sarsa(λ): $Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$
            - $E_t(s, a) = \gamma \lambda E_{t-1}(s, a) + \mathbf{1}\left(S_t = s, A_t = a\right)$
      - Off-Policy
        - Off-Policy Monte Carlo (*impractical* as too high variance):
          $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \ldots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t - V\left(S_t\right)\right)$
        - Off-Policy TD Learning: $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\left(R_{t+1} + \gamma V\left(S_{t+1}\right)\right) - V\left(S_t\right)\right)$
        - Q-Learning: $Q(S, A) \leftarrow Q(S, A) + \alpha\left(R + \gamma \max_{a'} Q\left(S', a'\right) - Q(S, A)\right)$

- Value Function Approximation (*Lec 6*)
  - Incremental Methods
    - Incremental Prediction $\hat{v}(S, \mathbf{w}) \approx v_\pi(S)$
      - MC: $\Delta\mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_\mathbf{w}\hat{v}(S_t, \mathbf{w})$
      - TD(0): $\Delta\mathbf{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_\mathbf{w}\hat{v}(S_t, \mathbf{w})$
      - Forward TD(λ): $\Delta\mathbf{w} = \alpha\left(G_t^\lambda - \hat{v}(S_t, \mathbf{w})\right)\nabla_\mathbf{w}\hat{v}(S_t, \mathbf{w})$
      - Backward TD(λ): $\Delta\mathbf{w} = \alpha\delta_t E_t$
        - $E_t = \gamma\lambda E_{t-1} + \nabla_\mathbf{w}\hat{v}(S_t, \mathbf{w})$
    - Incremental Control
      - MC: $\Delta\mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_\mathbf{w}\hat{q}(S_t, A_t, \mathbf{w})$
      - TD(0): $\Delta\mathbf{w} = \alpha(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}))\nabla_\mathbf{w}\hat{q}(S_t, A_t, \mathbf{w})$
      - Forward-view $\mathrm{TD}(\lambda)$: $\Delta\mathbf{w} = \alpha\left(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w})\right)\nabla_\mathbf{w}\hat{q}(S_t, A_t, \mathbf{w})$
      - Backward-view $\mathrm{TD}(\lambda)$: $\Delta\mathbf{w} = \alpha\delta_t E_t$
        - $E_t = \gamma\lambda E_{t-1} + \nabla_\mathbf{w}\hat{q}(S_t, A_t, \mathbf{w})$
  - Batch Methods
- Policy Gradient Methods (*Lec 7*)
  - Policy Gradient Thoerem
  - Monte-Carlo PG
    - REINFORCE: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) v_t]$
  - Compatible Function Approximation Theorem
    - Compatible Value Function: $\nabla_w Q_w(s, a) = \nabla_\theta \log \pi_\theta(s, a)$
  - Actor-Critic PG
    - Q AC: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^w(s, a)]$
    - Advantage AC: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^w(s, a)]$
    - TD(0) AC: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)(r + \gamma V^w(s_{t+1}) - V^w(s_t))]$
    - Forward TD(λ) AC: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\left(v_t^\lambda - V^w(s_t)\right)e\right]$
    - Backward TD(λ) AC: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a)\delta_t e_t]$
      - $e_t = \gamma\lambda e_{t-1} + \nabla_\theta \log \pi_\theta(s_t, a_t)$
  - Natural PG
    - Natural AC: $\nabla_\theta^{nat} J(\theta) = G_\theta^{-1}\nabla_\theta J(\theta) = w$
- Model Based (*Lec 8*)
  - Model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ where $\mathcal{P}_\eta \approx \mathcal{P}$ and $\mathcal{R}_\eta \approx \mathcal{R}$
  - Model Learning
    - Supervised learning problem $\{S_t, A_t \to R_{t+1}, S_{t+1}\}_{t=1}^{T-1}$
      - $s, a \to r$: regression
      - $s, a \to s'$: density estimation
  - Table Lookup Model
  - Sample-based Planning
  - Dyna - Integrated Planning and Learning

Part I: Elementary Reinforcement Learning

Lecture 1: Introduction to Reinforcement Learning

The RL Problem

- A reward R_t is a single scalar feedback signal
- **Reward Hypothesis**: *All* goals can be described by the maximisation of expected cumulative reward.
- **Sequential Decision Making**
  - Actions may have long term consequences
  - Reward may be delayed
  - It may be better to sacrifice immediate reward to gain more long-term reward
- Formalism:
  - observation (O_t), action (A_t), reward (R_t)

    - **Agent** interacts with the **Environment**
- **History**: sequence of observations, actions, rewards
    - $H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$
    - Decides what happens next:
        - agent selects action
        - environment selects observations/rewards.
- **State**: $S_t = f(H_t)$
    - **Environment State** (S^e_T) - environment's private representation
    - **Agent State**(S^a_T) - agent's internal representation
    - **Information State** or **Markov State** - contains all the useful information from the history
        - Once Markov state S_t is known, history may be thrown away
            - $H_{1:t} \to S_t \to H_{t+1:\infty}$
            - State is sufficient statistic of future
        - Eg. S^e_T and H_t
- **Fully Observable Environments**
    - Agent directly observes environment state $O_t = S^t_a = S^t_e$
    - Formally: *MDP*.
- **Partially Observable Environments**
    - Agent indirectly observes environment.
    - Formally: *POMDP*
    - Agent must construct its own state representation S^a_t
        - Complete history: $S^a_t = H_t$
        - Beliefs of environment state: $S^a_t = \left(\mathbb{P}\left[S^e_t = s^1\right], \ldots, \mathbb{P}\left[S^e_t = s^n\right]\right)$
        - Recurrent neural network: $S^a_t = \sigma\left(S^a_{t-1} W_s + O_t W_o\right)$
- **Components of an RL Agent**
    - Policy (π): agent's behaviour function
        - Deterministic or Stochastic
    - Value function: how good is each state and/or action
        - $v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$
    - Model: agent's representation of the environment
        - $\mathcal{P}$: state transition matrix: $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
        - $\mathcal{R}$: rewards: $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
        - Not required
    - Categorisation 1:
        - **Value Based**: Only value Function
        - **Policy Based**: Only Policy
        - **Actor Critic**: Both Policy and Value Function
    - Categorisation 2:
        - **Model Free**
        - **Model Based**
- Fundamental problems in sequential decision making
    - **Reinforcement Learning**:
        - The environment is initially unknown
        - The agent interacts with the environment
        - The agent improves its policy
    - **Planning**:
        - The model/dynamics of the environment is known
        - The agent performs computations with its model (without any external interaction)
        - The agent improves its policy
- Exploration and Exploitation
    - **Exploration** finds more information about the environment
    - **Exploitation** exploits known information to maximise reward
    - Both are important in RL
- Prediction and Control
    - **Prediction**: evaluate the future given a policy
    - **Control**: find the best policy to optimise the future
    - typically need to solve *prediction* to solve *control*

Lecture 2: Markov Decision Processes

- Almost all RL problems can be formalised as MDPs,
- **Markov Property** - state captures all relevant information from history
- **Markov Process or Chain**: memoryless random process: $\langle \mathcal{S}, \mathcal{P} \rangle$
  - $\mathcal{S}$ - finite set of states
  - $\mathcal{P}$: State Transition Matrix: $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- **Markov Reward Process**: $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{R}$: reward function: $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
    - reward from exiting a state
  - $\gamma \in [0, 1]$: discount factor
- **Return**: total discounted reward from time-step t: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- Why discount?
  - Mathematically convenient to discount rewards
  - Avoids **infinite returns** in cyclic Markov processes
  - Preference for more immediate reward
  - Uncertainty about the future may not be fully represented
  - May use $\gamma = 1$ (no discounting)
    - If all sequences terminate
    - Average reward formulation
- **Value Function**: $v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$
- **Bellman Equation for MRPs**
  - $v(s) = \mathbb{E}\left[R_{t+1} + \gamma v\left(S_{t+1}\right) \mid S_t = s\right]$
  - $v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v\left(s'\right)$
  - in matrix form: $v = \mathcal{R} + \gamma \mathcal{P} v$
  - Solving for v
    - Analytically - $O(n^3)$
    - Iterative Methods
      - Dynamic programming
      - Monte-Carlo evaluation
      - Temporal-Difference learning
- **Markov Decision Processes**: $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - $\mathcal{A}$ - finite set of actions
  - $\mathcal{P}$: state transition matrix: $\mathcal{P}^a_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
  - $\mathcal{R}$: reward function: $\mathcal{R}^a_s = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- π: Policy: $\pi(a \mid s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$
  - Policies are stationary (time-independent).
- Given an $\mathrm{MDP} \mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
  - The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
  - The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
    - $\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathcal{P}^a_{ss'}$
    - $\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a \mid s) \mathcal{R}^a_s$
- Value functions:
  - **state-value function**: $v^\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right]$
  - **action-value function**: $q^\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right]$
- **Bellman Expectation Equation for MDPs**
  - for state-value function
    - $v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s\right]$
    - $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s, a) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi\left(s'\right)\right)$
  - for action-value function
    - $q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right]$
    - $q_\pi(s, a) = \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} v_\pi\left(s'\right) = \mathcal{R}^a_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \sum_{a' \in \mathcal{A}} \pi\left(a' \mid s'\right) q_\pi\left(s', a'\right)$
  - in matrix form: $v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$
- **Optimal Value Functions**
  - $v_*(s) = \max_\pi v_\pi(s)$

    - $q_*(s,a) = \max_\pi q_\pi(s,a)$
        - This can be used to make decision on which action to take
- **Optimal Policy**: π_*
    - *Partial Ordering* over policies:
    - Define a partial ordering over policies: $\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$
    - For any MDP,
        - There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
        - $v_{\pi_*}(s) = v_*(s)$
        - $q_{\pi_*}(s,a) = q_*(s,a)$
    - **Deterministic Optimal Policy**
        - Always exists for any MDP i.e. in fully observed setting
            - One such π_* can be obtained from q_*
        - May not exist in partially observed setting.
            - Eg. when there is state-aliasing or when value function approximation is used.
- **Bellman Optimality Equation for** v_*
    - $v_*(s) = \max_a q_*(s,a) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$
    - $q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a')$
    - max instead of expectation/summation over $a \sim \pi(a \mid s)$
    - Only Iterative methods (No analytical solution)
        - Dynamic PRogramming
            - Value Iteration
            - Policy Iteration
        - Q-learning
        - Sarsa
- **Extensions to MDPs**
    - Infinite and continuous MDPs
    - **Partially observable MDPs (POMDPs)**: $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$
        - $\mathcal{O}$: finite set of observations
        - $\mathcal{Z}$: observation function: $\mathcal{Z}_{s'o}^a = \mathbb{P}\left[O_{t+1} = o \mid S_{t+1} = s', A_t = a\right]$
        - Belief State
    - **Undiscounted Average Reward MDPs**

Lecture 3: Planning by Dynamic Programming

Introduction

- **Planning**:
    - The MDP/model/dynamics of the environment is known
    - The agent performs computations with its model to improve it's policy.
- MDP satisfy satisfy both requirements of DP
    - Optimal substructure - bellman equation
    - Overlapping subproblems properties - value function
- DP assumes full knowledge of MDP and is used for planning
    - For prediction:
        - *Input*: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π or MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
        - *Output*: value function v_π
    - For control:
        - *Input*: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
        - *Output*: optimal value function v_* and optimal policy π_*

Policy Evaluation

- **Iterative Policy Evaluation**
    - *Solution*: iterative application of Bellman expectation backup
    - $v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$
    - in matrix form: $\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$
    - synchronous or asynchronous backups

Policy Iteration

- **Policy Iteration**
  - **Policy Evaluation**: Any policy evaluation algorithm to estimate $v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right]$
    - Eg. *Iterative Policy Evaluation*
    - Don't need to converge to v_π -> Other algorithms
    - different stopping criterion in iterative policy evaluation:
      - ϵ-convergence of value function
      - stop after k iterations
        - $k = 1$ gives *value iteration*
  - **Policy Improvement**: Any policy improvement algorithm to generate $\pi' \geq \pi$
    - Eg. *Greedy Policy Improvement*
- **Greedy Policy Improvement**:
  * Acting greedily with respect to v_π: $\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a)$
  * Improves policy
  * taking greeedy action in next step only
    * $q_\pi\left(s, \pi'(s)\right) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
  * improves value function if acting greedily in all steps
    * $v_\pi(s) \leq q_\pi\left(s, \pi'(s)\right) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s\right] \leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s)$
  * If improvement stops: $V_\pi(s) = V_*(s)$ and π is optimal.
  * $q_\pi\left(s, \pi'(s)\right) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$
  * Bellman Optimality Equations are satisfied
  * *Policy Iteration with Greedy Policy Improvement always converges*

Value Iteration

- **Principle of Optimality**: For a policy $\pi(a \mid s)$ and any state s, $v_\pi(s) = v_*(s)$ *iff* for any state s' reachable from s, $v_\pi\left(s'\right) = v_*\left(s'\right)$.
- *Algorithm*: Directly solve for v_* by iteratively applying this update to all states:
  - **Bellman Optimality Backup**: $v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*\left(s'\right)$
  - May be synchronous or asynchronous.
- *Intuition*: start with final rewards and work backwards.
- No explicit policy unlike policy iteration.
  - Intermediate value functions may not correspond to any policy.
- Still works with loopy, stochastic MDPs.
- Equivalent to Policy Iteration with single iteration of Iterative Policy Evaluation and Greedy Policy Improvement.

Summary of Synchronous DP Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O\left(mn^2\right)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O\left(m^2 n^2\right)$ per iteration

Extensions to Dynamic Programming

- **Asynchronous Dynamic Programming Algorithms**
  - Synchronous DP uses synchronous backups i.e. all states are backed up in parallel
  - Ideas for Asynchonous DP:
    - **In-place dynamic programming**
      - Synchronous VI keeps two copies of value function: $v_{new}(s)$ and $v_{old}(s)$
        - for all s:
          - $v_{\text{new}}(s) \leftarrow \max_{a \in \mathcal{A}}\left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_{\text{old}}\left(s'\right)\right)$

        - $v_{new}(s) \leftarrow v_{old}(s)$
      - In-place VI only keeps one copy:
        - for all s:
          - $v(s) \leftarrow \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v\left(s'\right)\right)$
    - **Prioritised sweeping**
      - *Bellman Error*: $\left|\max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v\left(s'\right)\right) - v(s)\right|$
      - Backup state with largest remaining Bellman Error
    - **Real-time dynamic programming**
      - Idea: Only backup states that are relevant to agent
      - Use agent's experience to guide the selection of states
      - At each time-step sample S_t, A_t, R_{t+1} and backup state St
  - Backup type
    - **Full Width Backup**
      - For each backup (sync or async)
        - Every successor state and action is considered
        - Requires knowledge of the MDP transitions and reward function
      - Used by DP
      - Problem: *Curse of Dimensionality*- even one full-width backup may be too expensive
    - **Sample Backup**
      - Sample rewards and transitions $\langle S, A, R, S' \rangle$
      - *Pros*:
        - Model-free: no advance knowledge of MDP required
        - Breaks the curse of dimensionality through sampling
        - Cost of backup is constant, independent of $n = |S|$
  - **Approximate DP**
    - Apply DP to a function approximator $\hat{v}(s; w)$ for the value function
    - Example: **Fitted Value Iteration**
      - Repeat at iteration k:
        - Sample states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
        - For each state $s \in \tilde{\mathcal{S}}$, estimate target value using Bellman optimality equation,
          - $\tilde{v}_k(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \hat{v}\left(s', \mathbf{w_k}\right)\right)$
        - Train next value function $\hat{v}\left(\cdot, \mathbf{w_{k+1}}\right)$ using targets $\{\langle s, \tilde{v}_k(s)\rangle\}$

Contraction Mapping

- l_∞ norm distance between value functions: $\|u - v\|_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$
- *Bellman Expectation Backup is a Contraction*
  - Bellman expectation backup operator $T^\pi: T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v$
  - T^π is a γ-contraction: $\left\|T^\pi(u) - T^\pi(v)\right\|_\infty \leq \gamma \|u - v\|_\infty$
- *Bellman Optimality Backup is a Contraction*
  - Bellman optimality backup operator $T^*: T^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$
  - T^* is a γ-contraction: $\left\|T^*(u) - T^*(v)\right\|_\infty \leq \gamma \|u - v\|_\infty$
- **Contraction Mapping Theorem**
  - For any metric space $\mathcal{V}$ that is complete and hence closed under an operator $T(v)$, where T is a γ-contraction,
    - T converges to a *unique fixed point* at a *linear convergence rate* of γ
  - T^π converges to v_π (a fixed point of T^π by Bellman expectation equation)
    - Iterative policy evaluation converges on v_π
    - Policy iteration converges on v_*
  - T^* converges to v_* (a fixed point of T^* By Bellman Optimality equation)
    - Value iteration converges on v_*

Lecture 4: Model-Free Prediction

Monte-Carlo Learning

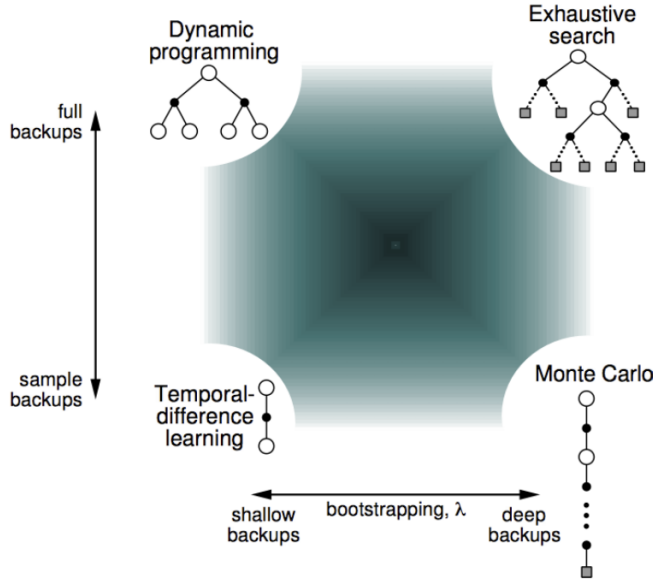
- **Monte Carlo Methods**
  - MC methods learn directly from complete episodes of experience

        - uses **complete episodes**: no bootstrapping
    - Model-free: no knowledge of MDP transitions / rewards
    - MC uses the simplest possible idea: value = mean return
    - **Can only apply MC to episodic MDPs**
- **Monte-Carlo Policy Evaluation**
    - Goal: learn v_π from episodes of experience under policy π
        - $S_1, A_1, R_2, \ldots, S_k \sim \pi$
    - return = total discounted reward: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$
    - value function = expected return: $v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right]$
    - Monte-Carlo policy evaluation uses empirical mean return instead of expected return
        - $V(s) = S(s)/N(s)$ where
            - $N(s)$ - counter for number of times s is visited
            - $S(s)$ - total return from s
        - By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$
    - **First-Visit** Monte-Carlo Policy Evaluation
        - For *first time-step* t that s is visited in an episode:
            - Increment counter $N(s) \leftarrow N(s) + 1$
            - Increment total return $S(s) \leftarrow S(s) + G_t$
    - **Every-Visit** Monte-Carlo Policy Evaluation
        - For *every time-step* t that s is visited in an episode:
            - Increment counter $N(s) \leftarrow N(s) + 1$
            - Increment total return $S(s) \leftarrow S(s) + G_t$
    - **Incremental Monte-Carlo Updates**
        - After each episode, for each state S_t with return G_t:
            - $N\left(S_t\right) \leftarrow N\left(S_t\right) + 1$
            - $V\left(S_t\right) \leftarrow V\left(S_t\right) + \frac{1}{N(S_t)}\left(G_t - V\left(S_t\right)\right)$
        - In non-stationary problems where it can be useful to forget old episodes.
            - $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t - V\left(S_t\right)\right)$

Temporal-Difference Learning

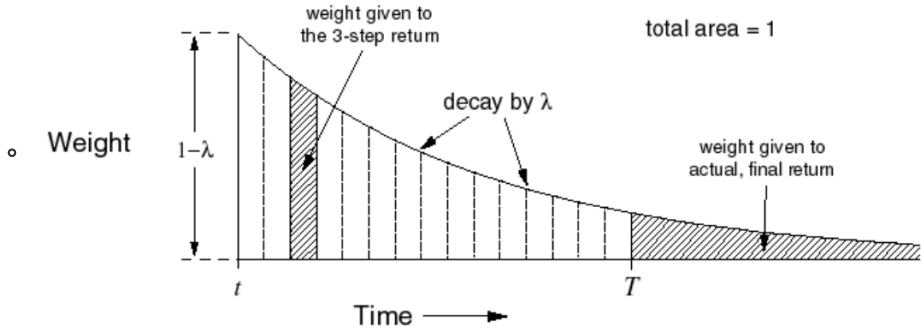
- **TD Methods**
    - TD methods learn directly from episodes of experience
        - works with **incomplete episodes** by bootstrapping
    - Model-free: no knowledge of MDP transitions / rewards
    - TD updates a guess towards a guess
- TD(0):
    - Update $V\left(S_t\right)$ toward estimated return: $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right)$
        - **TD target**: $R_{t+1} + \gamma V\left(S_{t+1}\right)$
        - **TD Error**: $\delta_t = R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)$
- **MC v/s TD**(0)
    - Update:
        - MC: Update toward actual return: $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t - V\left(S_t\right)\right)$
        - TD(0): Update toward estimated return: $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right)$
    - **Bias/Variance Tradeoff**
        - Return G_t is an unbiased estimate of $v_\pi(S_t)$
            - But depends on many random actions, transitions, rewards => higher variance
        - TD target $R_{t+1} + \gamma V\left(S_{t+1}\right)$ is a biased estimate of $v_\pi(S_t)$
            - But depends on one random actions, transitions, rewards => lower variance
    - **Certainty Equivalence of Batch MC and TD**
        - MC converges to solution with minimum mean-squared error
            - Finds the best fit to the observed returns
                - $\sum_{k=1}^{K} \sum_{t=1}^{T_k} \left(G_t^k - V\left(s_t^k\right)\right)^2$
        - TD (0) converges to solution of max likelihood Markov model
            - Solves the MDP $\left\langle \mathcal{S}, \mathcal{A}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, \gamma \right\rangle$ that best fits the data

        - $\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}\left(s_t^k, a_t^k, s_{t+1}^k = s, a, s'\right)$
        - $\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbf{1}\left(s_t^k, a_t^k = s, a\right) r_t^k$
    - **Advantages and Disadvantages of MC vs. TD**
        - *Shallow vs deep Backup*:
            - TD can learn online after every step before knowing the final outcome
            - MC must wait until end of episode when return is known
        - *Episodic v/s Continuing environments*:
            - TD can learn from incomplete episodes and works with non-episodic environments.
            - MC can only learn from complete episodes and only works for episodic (terminating) environments
        - *sample efficiency*: TD is more sample-efficient
        - *bias-variance tradeoff*: MC has higher variance but zero bias
        - *convergence*:
          * MC has good convergence properties even with function approximation
          * TD may not converge with function approximation
          * TD is sensitive to initial value
        - *Markov Property*:
            - TD exploits Markov property and is more efficient in Markov environments
            - MC does not and is more effective in non-Markov environments.
        - **Summary of Backups**:
            - MC: $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t - V\left(S_t\right)\right)$
                - TD(0): $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right)\right)$
                    - DP: $V\left(S_t\right) \leftarrow \mathbb{E}_\pi\left[R_{t+1} + \gamma V\left(S_{t+1}\right)\right]$
        - **Bootstrapping and Sampling**
            - Bootstrapping: update involves an estimate: eg. DP, TD
            - Sampling: update samples an expectation: eg. MC, TD



TD(λ) and Multi-Step Returns

- **n-Step Return**: $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V\left(S_{t+n}\right)$
    - TD(n): $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t^{(n)} - V\left(S_t\right)\right)$
        - $n = 0$ - TD(0)
        - $n = \infty$ - MC
- **λ-return**: $G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$
    - geometric weighting are *memoryless* => doesn't require storing anything extra for all $G_t^{(n)}$

- 

- **Forward-view TD(λ)**: $V\left(S_t\right) \leftarrow V\left(S_t\right)+\alpha\left(G_t^\lambda-V\left(S_t\right)\right)$
- **Eligibility Traces**
  - **Credit Assignment problem:**
    - **Frequency heuristic**: assign credit to most frequent states
    - **Recency heuristic**: assign credit to most recent states
  - Eligibility Traces combine both heuristics
    - $E_0(s)=0$
    - $E_t(s)=\gamma \lambda E_{t-1}(s)+\mathbf{1}\left(S_t=s\right)$
- **Backward-view TD(λ)**: $V(s) \leftarrow V(s)+\alpha \delta_t E_t(s)$
  - Idea:
    - Keep an eligibility trace for every state s
    - Update $V(s)$ in proportion to TD-error δ_t and eligibility trace $E_t(s)$
  - Backward-view provides mechanism for using **λ-return** that allows
    * Updating online, every step, from incomplete sequences
  - Eligibility trace allows earlier states and actions in the episode to get updates even if the reward comes at the end of the episode.
    - λ controls how far back in the episode the updates go and hence the bias-variance tradeoff as further back you go more random variables will the updates pass through.
  - When $\lambda=0$, only current state is updated
    - $E_t(s)=\mathbf{1}\left(S_t=s\right)$
    - $V(s) \leftarrow V(s)+\alpha \delta_t E_t(s)$
      - Equivalent to $\mathrm{TD}(0)$ update: $V\left(S_t\right) \leftarrow V\left(S_t\right)+\alpha \delta_t$
  - When $\lambda=1$, credit is deferred until end of episode
    - In an episodic environment, it has the same update as TD(λ)
    - The sum of offline updates is identical for forward-view and backward-view $TD(\lambda)$
      - $\sum_{t=1}^T \alpha \delta_t E_t(s)=\sum_{t=1}^T \alpha\left(G_t^\lambda-V\left(S_t\right)\right) 1\left(S_t=s\right)$
- **TODO**: Finish This
-

Offline updates	$\lambda=0$	$\lambda \in (0,1)$	$\lambda=1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\|\|$	$\|\|$	$\|\|$
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda=0$	$\lambda \in (0,1)$	$\lambda=1$
Backward view	TD(0)	TD(λ)	TD(1)
	$\|\|$	$\not\|\|$	$\not\|\|$
Forward view	TD(0)	Forward TD(λ)	MC
	$\|\|$	$\|\|$	$\|\|$
Exact Online	TD(0)	Exact Online TD(λ)	Exact Online TD(1)

Lecture 5: Model-Free Control

Introduction

- **On-policy learning**: Learn about policy π from experience sampled from π
- **Off-policy learning**: Learn about policy π from experience sampled from μ

On-Policy Monte-Carlo Control

- **Model-Free Policy Iteration**
  - First attempt:
    - *Policy Evaluation*: Monte-Carlo Policy Evaluation
    - *Policy improvement*: Greedy policy improvement over $V(s)$
  - *Problem 1*: Greedy policy improvement over $V(s)$ requires model of MDP
    - $\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$
    - *Solution*: Use $Q(s,a)$ instead as Greedy policy improvement over it is model-free
      - $\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s,a)$
  - *Problem 2*: If acting greedily with Monte-Carlo policy evaluation will not explore all states and end up with incorrect estimates of the value function.
    - *Solution*: Ensure Exploration
      - ϵ**-Greedy Exploration**
        - With probability $1 - \epsilon$ choose the greedy action
        - With probability ϵ choose an action at random
        - ϵ**-Greedy Policy Improvement**: For any ϵ -greedy policy π, the ϵ -greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$
- **Monte-Carlo Policy Iteration**
  - *Algorithm*:
    - *Policy Evaluation*: Monte-Carlo Policy Evaluation, $Q = q_\pi$
    - *Policy improvement*: ϵ-Greedy policy improvement over $Q(s,a)$
  - *Problem*: Monte-Carlo is inefficient
    - *Solution*: find $Q \approx q_\pi$ using a few or even just one episode
- **Greedy in the Limit with Infinite Exploration (GLIE)**
  - All state-action pairs are explored infinitely many times,
    - $\lim_{k \to \infty} N_k(s,a) = \infty$
  - The policy converges on a greedy policy,
    - $\lim_{k \to \infty} \pi_k(a \mid s) = \mathbf{1}\left(a = \underset{a' \in \mathcal{A}}{\operatorname{argmax}} Q_k(s,a')\right)$
  - Example: ϵ-greedy is GLIE if ϵ decays to zero as $\epsilon_k = 1/k$
- **GLIE Monte-Carlo Control**
  - *Algorithm*:
    - Sample kth episode using $\pi : \{S_1, A_1, R_2, \ldots, S_T\} \sim \pi$
    - For each state S_t and action A_t in the episode,
      - $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
    - Improve policy based on new action-value function
      - $\epsilon \leftarrow 1/k$
      - $\pi \leftarrow \epsilon$-greedy(Q)
  - Just generates a single episode in each control (outer) loop iteration
  - The statistics collected in the policy evaluation step will be for changing (improving) policies.
  - Converges to $q_*(s,a)$

Sarsa: On-Policy Temporal-Difference Learning

- Pros of TD over MC:
  - Lower variance
  - Online
  - Incomplete sequences
- Idea: Use TD instead of MC in control loop
- **SARSA**:
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S', A') - Q(S,A)\right)$
    - A and A' are both chosen using the *same* policy derived from the $Q(S,A)$
      - eg. ϵ-greedy
  - Algorithm:
    - **Every time-step**:
      - *Policy evaluation*: Sarsa, $Q \approx q_\pi$

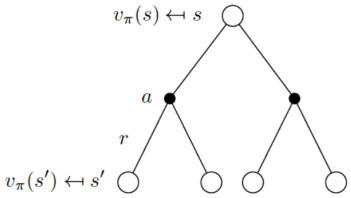
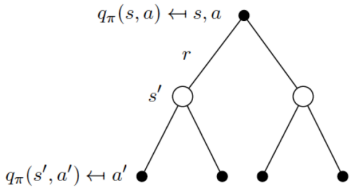
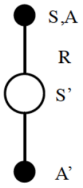
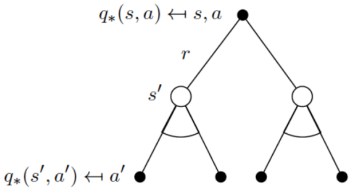
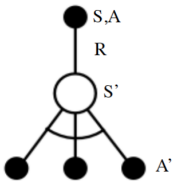
        - *Policy improvement*: ϵ-greedy policy improvement
    - Converges to $q_*(s,a)$
- **n-Step Sarsa**
    - n-step Q-return: $q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q\left(S_{t+n}\right)$
    - n-step Sarsa updates $Q(s,a)$ towards the n -step Q-return
    - $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q_t^{(n)} - Q\left(S_t, A_t\right)\right)$
        - n = 1: Sarsa
        - n = ∞: MC
- **Sarsa(λ)**
    - $q_t^\lambda = (1-\lambda)\sum_{n=1}^\infty \lambda^{n-1} q_t^{(n)}$
    - **Forward-view Sarsa(λ)**: $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q_t^\lambda - Q\left(S_t, A_t\right)\right)$
    - **Backward-view Sarsa(λ)**: $Q(s,a) \leftarrow Q(s,a) + \alpha \delta_t E_t(s,a)$
        - $\delta_t = R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)$
        - Use an Eligibility Trace for each state-action pair
            - $E_0(s,a) = 0$
            - $E_t(s,a) = \gamma\lambda E_{t-1}(s,a) + \mathbf{1}\left(S_t = s, A_t = a\right)$
        - $Q(s,a)$ is updated **for every state s and action a** in proportion to TD-error δ_t and eligibility trace $E_t(s,a)$

Off-Policy Learning

- *Idea*:
    - Evaluate target policy $\pi(a \mid s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
    - While following behaviour policy $\mu(a \mid s)$
        - $\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$
- *Advantages*:
    - Learn from observing humans or other agents
    - Re-use experience generated from old policies
    - Learn about optimal policy while following exploratory policy
    - Learn about multiple policies while following one policy
- **Importance Sampling**: $E_{x\sim p(x)}[f(x)] = \int p(x) f(x) dx = E_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right]$
- **Importance Sampling for Off-Policy Monte-Carlo**
    - $G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$
        - Multiply importance sampling corrections along whole episode
    - $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(G_t^{\pi/\mu} - V\left(S_t\right)\right)$
    - Cannot use if μ is zero when π is non-zero
    - *Problem*: Importance sampling can dramatically increase variance => **Cannot off-policy Monte-Carlo in practice**
- **Importance Sampling for Off-Policy TD**
    - $V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha\left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\left(R_{t+1} + \gamma V\left(S_{t+1}\right)\right) - V\left(S_t\right)\right)$
        - Only need a single importance sampling corrections
- **Q-Learning**
    - **No importance sampling**
    - *Idea*:
        - Next action is chosen using behaviour policy $A_{t+1} \sim \mu\left(\cdot \mid S_t\right)$
        - But successor action taken from target policy $A' \sim \pi\left(\cdot \mid S_t\right)$
        - $Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(R_{t+1} + \gamma Q\left(S_{t+1}, A'\right) - Q\left(S_t, A_t\right)\right)$
    - Q-learning allows both behaviour and target policies to improve
    - The target policy π is greedy w.r.t. $Q(s,a)$: $\pi\left(S_{t+1}\right) = \underset{a'}{\operatorname{argmax}}\, Q\left(S_{t+1}, a'\right)$
    - The behaviour policy μ can be ϵ -greedy w.r.t. $Q(s,a)$
    - **Q-learning target**: $R_{t+1} + \gamma Q\left(S_{t+1}, A'\right) = R_{t+1} + \max_{a'} \gamma Q\left(S_{t+1}, a'\right)$
    - $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma \max_{a'} Q\left(S', a'\right) - Q(S,A)\right)$

Summary

Table 1: Relationship Between DP and TD

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $v_\pi(s)$	 **Iterative Policy Evaluation** $V(s) \leftarrow \mathbb{E}\left[R + \gamma V\left(S'\right) \mid s\right]$	 **TD Learning** $V(S) \overset{\alpha}{\leftarrow} R + \gamma V\left(S'\right)$
Bellman Expectation Equation for $q_\pi(s,a)$	 **Q-Policy Iteration** $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q\left(S', A'\right) \mid s, a\right]$	 **Sarsa** $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q\left(S', A'\right)$
Bellman Expectation Equation for $q_*(s,a)$	 **Q-Value Iteration** $Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q\left(S', a'\right) \mid s, a\right]$	 **Q-Learning** $Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q\left(S', a'\right)$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

Part II: Reinforcement Learning in Practice

Lecture 6: Value Function Approximation

Introduction

- So far we have represented value function by a *lookup table*
- *Problem*: with large MDPs,
  - Too many states and/or actions to store in memory
  - Too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation
    - $\hat{v}(s; w) \approx v_\pi(s)$ or $\hat{q}(s, a; w) \approx q_\pi(s, a)$
  - *Generalise* from seen states to unseen states
  - Update parameter w using MC or TD learning
- Will consider **differentiable** function approximators,
  - Linear Combination of Features
  - Neural Network
- The training method should be suitable for data that is
  - **non-stationary**- if estimating v_π, the policy π will keep changing
  - **non-iid**- because it's not supervised learning, data arrives in a trajectory

Incremental Methods

- Let $J(w)$ be the differentiable function of parameter vector w.

Incremental Prediction Algorithms

- Let $x(S) = [x_1(S), \ldots, x_n(S)]^T$ be the state feature vector

- **Linear Value Function Approximation**:
  - $\hat{v}(S, \mathbf{w}) = \mathbf{x}(S)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S) \mathbf{w}_j$
  - $J(\mathbf{w}) = \mathbb{E}_\pi \left[\left(v_\pi(S) - \mathbf{x}(S)^\top \mathbf{w} \right)^2 \right]$
    - Quadratic in parameters $\mathbf{w}$ => SGD converges to global optimum
  - Update rule:
    - $\nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) = \mathbf{x}(S)$
    - $\Delta \mathbf{w} = \alpha \left(v_\pi(S) - \hat{v}(S, \mathbf{w}) \right) \mathbf{x}(S)$
  - Update Complexity $=$ step-size $\times$ prediction error $\times$ feature value
- **Incremental Prediction Algorithms**
  - Train $\hat{v}\left(S, \mathbf{w}\right) \approx v_\pi(S)$
  - supervised learning on "training data": $\left\langle S_1, Y_1 \right\rangle, \left\langle S_2, Y_2 \right\rangle, \ldots, \left\langle S_T, Y_T \right\rangle$
    - where Y_t is an sample/estimate of true value $v_\pi(S_t)$
    - different algorithms use different estimates:
      - MC: Uses return G_t as an unbiased, noisy sample
      - $\mathrm{TD}(0)$: Uses TD-target $R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right)$ for a biased sample
      - $\mathrm{TD}(\lambda)$: Uses λ -return G_t^λ for a biased sample
  - MC: $\Delta \mathbf{w} = \alpha \left(G_t - \hat{v}\left(S_t, \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$
    - Monte-Carlo evaluation converges to a local optimum
      - Even with non-linear value function approximation
  - $\mathrm{TD}(0)$: $\Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right) - \hat{v}\left(S_t, \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$
    - Converges (close) to global optimum for linear value function approximation
  - **Forward-view** $\mathrm{TD}(\lambda)$: $\Delta \mathbf{w} = \alpha \left(G_t^\lambda - \hat{v}\left(S_t, \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$
  - **Backward-view** $\mathrm{TD}(\lambda)$: $\Delta \mathbf{w} = \alpha \delta_t E_t$
    - $\delta_t = R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right) - \hat{v}\left(S_t, \mathbf{w}\right)$
    - $E_t = \gamma \lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{v}\left(S_t, \mathbf{w}\right)$
    - This update can be applied online, to incomplete sequences.
  - Forward view and backward view linear $\mathrm{TD}(\lambda)$ are equivalent

Incremental Control Algorithms

- Let $x(S, A) = [x_1(S, A), \ldots, x_n(S, A)]^T$ be the state-action feature vector
- **Linear Action-Value Function Approximation**:
  - $\hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)^\top \mathbf{w} = \sum_{j=1}^{n} \mathbf{x}_j(S, A) \mathbf{w}_j$
  - Update rule:
    - $\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) = \mathbf{x}(S, A)$
    - $\Delta \mathbf{w} = \alpha \left(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}) \right) \mathbf{x}(S, A)$
- **Incremental Control Algorithms**
  - different algorithms use different estimates for the true value $q_\pi(s, a)$:
    - MC: Uses return G_t as an unbiased, noisy sample
    - $\mathrm{TD}(0)$: Uses TD-target $R_{t+1} + \gamma \hat{v}\left(S_{t+1}, \mathbf{w}\right)$ for a biased sample
    - $\mathrm{TD}(\lambda)$: Uses λ -return G_t^λ for a biased sample
  - MC: $\Delta \mathbf{w} = \alpha \left(G_t - \hat{q}\left(S_t, A_t, \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{q}\left(S_t, A_t, \mathbf{w}\right)$
  - $\mathrm{TD}(0)$: $\Delta \mathbf{w} = \alpha \left(R_{t+1} + \gamma \hat{q}\left(S_{t+1}, A_{t+1}, \mathbf{w}\right) - \hat{q}\left(S_t, A_t, \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{q}\left(S_t, A_t, \mathbf{w}\right)$
  - **Forward-view** $\mathrm{TD}(\lambda)$: $\Delta \mathbf{w} = \alpha \left(q_t^\lambda - \hat{q}\left(S_t, A_t, \mathbf{w}\right) \right) \nabla_{\mathbf{w}} \hat{q}\left(S_t, A_t, \mathbf{w}\right)$
  - **Backward-view** $\mathrm{TD}(\lambda)$: $\Delta \mathbf{w} = \alpha \delta_t E_t$
    - $\delta_t = R_{t+1} + \gamma \hat{q}\left(S_{t+1}, A_{t+1}, \mathbf{w}\right) - \hat{q}\left(S_t, A_t, \mathbf{w}\right)$
    - $E_t = \gamma \lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{q}\left(S_t, A_t, \mathbf{w}\right)$
    - This update can be applied online, to incomplete sequences.
- **Convergence of Prediction Algorithms**
  -

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	TD(0)	✓	✓	✗
	TD(λ)	✓	✓	✗
Off-Policy	MC	✓	✓	✓
	TD(0)	✓	✗	✗
	TD(λ)	✓	✗	✗

    - **Gradient TD Learning**
        - TD diverges when off-policy or using non-linear function approximation because it doesn't follow the gradient of any objective function.
        - Gradient TD follows true gradient of projected Bellman error and fixes convergence issues.
- **Convergence of Control Algorithms**

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

    - (✓) - chatters around near-optimal value function

Batch Methods

- **SGD with Experience Replay**
    - Given experience consisting of ⟨state,value⟩ pairs
    - $\mathcal{D} = \left\{\left\langle s_1, v_1^\pi \right\rangle, \left\langle s_2, v_2^\pi \right\rangle, \ldots, \left\langle s_T, v_T^\pi \right\rangle\right\}$
    - *Repeat*:
        - Sample state, value from experience $\left\langle s, v^\pi \right\rangle \sim \mathcal{D}$
        - Apply SGD update: $\Delta \mathbf{w} = \alpha \left(v^\pi - \hat{v}(s, \mathbf{w})\right) \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$
    - Converges to least squares solution: $\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}}\, LS(\mathbf{w})$
- **DQN**
    - Two ideas prevent DQN from blowing up unlike Sarsa and Q-Learning:
        - **experience replay** - eliminates correlated samples
        - **fixed Q-targets** - eliminates the moving target
    - *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 $\left(s, a, r, s'\right)$ from $\mathcal{D}$
        - Compute Q-learning targets w.r.t. old, fixed parameters w^-
        - Optimise MSE between Q-network and Q-learning targets
            - $\mathcal{L}_i\left(w_i\right) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; w_i^-\right) - Q\left(s, a; w_i\right)\right)^2\right]$
-

Lecture 7: Policy Gradient Methods

Introduction

- **Policy Based RL**
    - Parametrise the policy directly $\pi_\theta(s, a) = P(a \mid s, \theta)$
    - still **model-free**
    - *Pros*:
        - **Better convergence** properties
        - Effective in **high-dimensional or continuous action spaces** as no need to do a max over actions
        - Can learn **stochastic policies**

            - If the environment is partially observed then it is possible that only stochastic policies are optimal and all deterministic policies are subotimal.
                - *Example*: when state aliasing occurs
                - *Example*: if function approximation is used and the features used limit the view of the world
    - *Cons*:
        - Typically converge to a **local optimum**
        - Evaluating a policy is typically **ineffcient and high variance**
            - Resolved using value function in Actor-Critic methods.
- **Policy Objective Functions**
    - For *episodic environments*
        - use the *start value*: $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$
    - For _continuing environments _
        - use the *average value*: $J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
        - Or the *average reward per time-step*: $J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) \mathcal{R}_s^a$
    - where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ
    - Policy gradient is essentially the same for all three

Monte-Carlo Policy Gradient

- **Likelihood-ratio trick**: $\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$
- **Score Function**: $\nabla_\theta \log \pi_\theta(s,a)$
    - *Example*: Softmax Policy: $\pi_\theta(s,a) \propto e^{\phi(s,a)^T \theta}$
        - $\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$
    - *Example*: Gaussian Policy: $a \sim \mathcal{N}\left(\mu(s), \sigma^2\right)$ where $\mu(s) = \phi(s)^T \theta$
    - $\nabla_\theta \log \pi_\theta(s,a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$
- **Policy Gradient for One-step MDPs**
    - Contextual Bandits: state, $s \sim d(s)$, reward, $r = \mathcal{R}_{s,a}$
    - $J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \mathcal{R}_{s,a}$
        - All three objective functions will be equivalent
    - $\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a} = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]$
- **Policy Gradient Theorem**
    - For any differentiable policy $\pi_\theta(s,a)$, for any of the policy objective functions $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$ the policy gradient is
        - $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$
- **REINFORCE: Monte-Carlo Policy Gradient**
    - $\nabla \theta_t = \alpha \nabla_\theta \log \pi_\theta(s,a) v_t$ where the return, v_t, is an unbiased estimator of $Q^{\pi_\theta}(s,a)$

Actor-Critic Policy Gradient

- Reduce Variance using Critic: $Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$
- Actor-critic algorithms maintain two sets of parameters
    - **Critic**: Updates action-value function parameters w
    - **Actor**: Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient
    - $\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$
    - $\Delta \theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$
- Critic is doing policy evaluation. Eg. MC PE, TD learning, $TD(\lambda)$
- **Compatible Function Approximation Theorem**
    - If the following two conditions are satisfied:
        - Value function approximator is **compatible** to the policy: $\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$
            - i.e. *features for the value function approximator are the score function itself*.
                - Eg. if $Q_w(s,a) = \phi(s,a)^T w$ then, $\phi(s,a) = \nabla_\theta \log \pi_\theta(s,a)$
        - Value function parameters w minimise the mean-squared error: $\varepsilon = \mathbb{E}_{\pi_\theta}\left[\left(Q^{\pi_\theta}(s,a) - Q_w(s,a)\right)^2\right]$
    - Then the policy gradient is exact: $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$
- **Reducing Variance Using a Baseline**

- Subtract a baseline function $B(s)$ from the policy gradient
  - Reduce variance, without changing expectation
    - $\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) B(s)\right] = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) B(s) = 0$
- **Advantage function** $A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$
  - Using $B(s) = V^{\pi_\theta}(s)$
  - $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]$
- **Estimating Advantage function**
  - Option 1: Use two function approximators: $V_v(s) \approx V^{\pi_\theta}(s)$, $Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$
  - Option 2: Use a single function approximator: $V_v(s) \approx V^{\pi_\theta}(s)$
    - TD error for the true value function $\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$, is an unbiased estimate of the advantage function.
      * $\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta} \mid s,a\right] = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) = A^{\pi_\theta}(s,a)$
    - approximate TD error: $\delta_v = r + \gamma V_v(s') - V_v(s)$
    - $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \delta_v\right]$

- **Critics at Different Time-Scales**
  - **MC**: $\Delta w = \alpha\left(v_t - V_w(s)\right) \nabla_w V_w(s)$
  - **TD(0)**: $\Delta w = \alpha\left(r + \gamma V(s') - V_w(s)\right) \nabla_w V_w(s)$
  - **Forward-view TD(λ)**: $\Delta w = \alpha\left(v_t^\lambda - V_w(s)\right) \nabla_w V_w(s)$
  - **Backward-view TD(λ)**: $\Delta w = \alpha \delta_t e_t$
    - $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
    - $e_t = \gamma\lambda e_{t-1} + \nabla_w V_w(s)$
    - This update can be applied online, to incomplete sequences.
  - Where $\nabla_w V_w(s) = \phi(s)$ for linear $V_w = w^T \phi(s)$.
- **Actors at Different Time-Scales**
  - Here θ are the parameters for policy π_θ and V_w is the function approximator for V^{π_θ} which in turn is used to estimate A^{π_θ}.
  - **MC**: $\Delta\theta = \alpha\left(v_t - V_w(s)\right) \nabla_\theta \log \pi_\theta(s_t, a_t)$
  - **TD(0)**: $\Delta\theta = \alpha\left(r + \gamma V_w(s_{t+1}) - V_v(s_t)\right) \nabla_\theta \log \pi_\theta(s_t, a_t)$
  - **Forward-view TD(λ)**: $\Delta\theta = \alpha\left(v_t^\lambda - V_v(s_t)\right) \nabla_\theta \log \pi_\theta(s_t, a_t)$
  - **Backward-view TD(λ)**: $\Delta\theta = \alpha \delta_t e_t$
    - $\delta_t = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$
    - $e_t = \gamma\lambda e_{t-1} + \nabla_\theta \log \pi_\theta(s_t, a_t)$
    - This update can be applied online, to incomplete sequences unlike MC or forward-view $\mathrm{TD}(\lambda)$.

Natural Policy Gradient

- A policy can often be reparametrised without changing action probabilities
  - Eg. increasing score of all actions in a softmax policy
- The vanilla gradient is sensitive to these reparametrisations
- **Natural Policy Gradient**
  - parametrisation independent
  - It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount
    - $\nabla_\theta^{nat} \pi_\theta(s,a) = G_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$
  - where G_θ is the **Fisher information matrix**
    - $G_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^T\right]$
  - With a *compatible* function approximator, $A_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)^T w$,
    - $\nabla_\theta J(\theta) = G_\theta w$
  - Natural Policy Gradient: $\nabla_\theta^{nat} J(\theta) = w$
    - i.e. update actor parameters in direction of critic parameters

Summary

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\text{ Actor-Critic}\\
G_\theta^{-1}\nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}
$$

Lecture 8: Integrating Learning and Planning

Introduction

- **Model-free RL**
  - Value-based Methods: learn value function directly from experience
  - Policy-based Methods: learn policy directly from experience
- **Model-Based RL**: learn model directly from experience
  - and use planning to construct a value function or policy

Model-Based Reinforcement Learning

- *Pros*:
  - Easier to estimate model directly from experience than value fn/policy
  - Can effciently learn model by supervised learning methods
  - Can reason about model uncertainty
- *Cons*:
  - First learn a model, then construct a value function =>two sources of approximation error
- **Model**
  - A model $\mathcal{M}$ is a representation of an $\text{MDP}\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}\rangle$ parametrized by η
    - Assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known
    - So $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta\rangle$ where $\mathcal{P}_\eta \approx \mathcal{P}$ and $\mathcal{R}_\eta \approx \mathcal{R}$
  - Typically assume conditional independence between state transitions and rewards
    - $\mathbb{P}\left[S_{t+1}, R_{t+1} \mid S_t, A_t\right] = \mathbb{P}\left[S_{t+1} \mid S_t, A_t\right]\mathbb{P}\left[R_{t+1} \mid S_t, A_t\right]$
- **Model Learning**
  - Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, \ldots, S_T\}$
  - This is a supervised learning problem $\{S_t, A_t \rightarrow R_{t+1}, S_{t+1}\}_{t=1}^{T-1}$
    - Learning $s, a \rightarrow r$ is a regression problem
    - Learning $s, a \rightarrow s'$ is a density estimation problem
  - Find parameters η that minimise empirical loss
- Examples of Models:
  - Table Lookup Model
  - Linear Expectation Model
  - Linear Gaussian Model
  - Gaussian Process Model
  - Deep Belief Network Model
- **Table Lookup Model**
  - Parametric:
    - Count visits $N(s,a)$ to each state action pair
    - $\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)}\sum_{t=1}^{T} 1\left(S_t, A_t, S_{t+1} = s, a, s'\right)$
    - $\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)}\sum_{t=1}^{T} \mathbf{1}\left(S_t, A_t = s, a\right) R_t$
  - Non-parametric:
    - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle$
    - To sample model, randomly pick tuple matching $\langle s, a, ., .\rangle$
  - **Planning with Model**
    - Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta\rangle$
    - Solve the $\text{MDP}\ \langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta\rangle$
      - Algorithms: Value iteration, Policy iteration, Tree search etc.
  - **Sample-based Planning**
    - Use model to **only** to generate samples (*simulated experience*)

        - Apply model-free RL to samples, e.g.: Monte-Carlo control, Sarsa, Q-learning
        - Sample-based planning methods are often more efficient

Integrated Architectures

- Experience
    - Real experience Sampled from environment (true MDP)
        - $S' \sim \mathcal{P}^a_{s,s'}$
        - $R = \mathcal{R}^a_s$
    - Simulated experience Sampled from model (approximate MDP)
        - $S' \sim \mathcal{P}_\eta \left(S' \mid S, A\right)$
        - $R = \mathcal{R}_\eta(R \mid S, A)$
- Model-Free RL
    - No model
    - **Learn** value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
    - Learn a model from real experience
    - **Plan** value function (and/or policy) from simulated experience
- Dyna
    - Learn a model from real experience
    - **Learn and plan** value

Simulation-Based Search

Lecture 9: Exploration and Exploitation

Lecture 10: Case study - RL in games