Final Notes

Decision Learning

Decision learning involves making optimal choices based on available data. It often uses techniques from reinforcement learning and Bayesian decision theory to model uncertain environments.

Key Concepts

- **Utility-Based Agents:** Learn a utility function on states and use it to select actions that maximize the expected utility.
- **Exploration vs. Exploitation:** Balancing between taking new actions to discover better outcomes and using known successful actions.
- **Markov Decision Process (MDPs):** A framework for modeling decision making in situations where outcomes are partly random and partly controlled by the agent.

Reinforcement Learning

Reinforcement learning is a learning paradigm where an agent learns optimal actions by interacting with an environment and receiving rewards.

Key features of RL:

- **Trial and Error Learning:** The agent explores different actions to determine which yield the highest rewards
- **Delayed Rewards:** The reward may not be immediate but depends on future outcomes.
- **Agent-Environment Interaction**: The agent perceives the state and takes actions, influencing future states and rewards.

RL Components:

- **State (s):** the current state/situation of the agent
- **Action (a):** The possible moves the agent can make
- **Reward (r):** The feedback signal indicating the desirability of a state
- **Policy (π):** The strategy that maps states to actions
- **Value Function (V):** Expected long-term rewards for a state
- **Q-Function (Q):** Expected reward for a state-action pair

---

Reinforcement Learning Bandits

Multi-armed bandits (MABs) model the exploration-exploitation tradeoff in reinforcement learning. It involves choosing among multiple options (arms) with unknown reward distributions or unknown payoffs.

Types of Bandits

Multi-Armed Bandits (MABs) balance **exploration** (trying new options) and **exploitation** (choosing the best-known option) to maximize total reward.

---

1. **Greedy Strategy**

✅ Always picks the arm with the **highest observed reward**.

❌ **No exploration**, so it may get stuck in a suboptimal choice.

🔷 **Best for:** Simple, stationary environments.

---

2. **Optimistic Greedy**

✅ Starts with **high initial estimates**, forcing early exploration.

✅ Eventually behaves like **Greedy** once good arms are found.

❌ Bad if **initial values** are poorly chosen.

🔷 **Best for:** Ensuring initial exploration **without randomness**.

---

3. ϵ-**Greedy Strategy**

✅ **With probability (1 - ε), pick the best arm** (exploitation).

✅ **With probability (ε), pick a random arm** (exploration).

❌ Too much **exploration wastes time**, too little **gets stuck early**.

🔷 **Best for: Constant** exploration, works in **non-stationary** environments.

---

4. **Upper Confidence Bound (UCB)**

✅ Picks the arm with the **highest UCB score**:

$Q(a) + c \cdot \sqrt{\frac{\ln t}{N(a)}}$

✅ **Intelligently balances** reward & uncertainty.

❌ **More complex** than Greedy or **ε**-Greedy.

🔷 **Best for: Efficient exploration** with minimal wasted trials.

Algorithm	**Exploration**	**Exploitation**	**Best for**
Greedy	❌ None	✅ Always	Simple, stationary rewards
Optimistic Greedy	✅ Early only	✅ After initial phase	Ensuring early exploration
ε-Greedy	✅ Randomly	✅ Most of the time	Balancing short & long-term rewards
UCB	✅ Smartly	✅ Focuses on best arms over time	Efficient exploration

---

Final Thoughts

- **For simplicity:** Use **Greedy**.
- **For early exploration:** Use **Optimistic Greedy**.
- **For continuous exploration:** Use **ϵ-Greedy**.
- **For best efficiency:** Use **UCB**.

Reinforcement Learning Temporal Difference (TD) Learning

Temporal Difference (TD) Learning is a **reinforcement learning method** that updates the estimated value of a state by combining **observed rewards** with the **estimated future value** of the next state. Unlike Monte Carlo methods, TD learning **does not wait until the end of an episode** to update values—it learns **incrementally** after each step using bootstrapping.

Key Features:

- **Updates based on partial experience** (no need to wait for episode completion).
- **Uses bootstrapping** (estimates future values based on current estimates).
- **Balances exploration and exploitation** by continuously refining predictions.

TD learning is the foundation of algorithms like **Q-learning** and **SARSA**, enabling efficient decision-making in **unknown or uncertain environments**.

Key Equation:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

where:

- α is the learning rate
- γ is the discount factor
- $V(s)$ is the current estimated value of state s
- r is the reward
- $V(s')$ is the estimated value of the next state s'
- $r + \gamma V(s')$ → **TD Target** or the new expected value
- $V(s') - V(s)$ → **TD Error** or the difference between the current estimate and new estimate

TD methods do not require a model of the environment and are often used in combination with deep learning

---

Q Learning

Q-Learning is a model free reinforcement learning algorithm that learns an optimal action-selection policy.

Q-Learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a\prime} Q(s\prime, a\prime) - Q(s,a)]$$

where:

- $Q(s,a)$ is the action-value function.
- α (learning rate) controls how much new information overrides old estimates.
- γ (discount factor) determines the importance of future rewards.
- r is the reward received for taking action a in state s.
- $\max_{a\prime} Q(s\prime, a\prime)$ represents the best possible reward for the next state.

Off-policy Learning: Learns the best policy regardless of the agent's actions.

Exploration Function: Uses an exploration strategy (e.g. ϵ-greedy) to balance learning and exploitation

Convergence Guarantee: Given sufficient exploration, Q-learning converges to the optimal policy in **finite Markov Decision Processes (MDPs)**

Approximate Q Learning

In large or continuous state spaces, storing all $Q(s,a)$ values in a table is impractical. Approximate Q-learning replaces the table with a function approximator, such as a neural network.

Key Concepts:

- **Function Approximation:** Uses linear function, neural networks, or decision trees to estimate Q-values **{** $Q(s,a) = w^T \cdot \phi(s,a)$ **}**
- **Deep Q-Networks (DQN):** Uses deep learning to approximate Q-values

Advantages:

- Handles continuous and high-dimensional state-action spaces
- Reduces memory usage compared to Q-tables
- Enables learning in complex real-world environments (e.g. robotics, autonomous behicles)

Ethics

The development of AI brings ethical considerations regarding its impact on society.

Key Ethical Issues:

- **Job Loss Due to Automation:** AI may replace human jobs in various industries
- **Bias in AI systems:** Training data biases can lead to discriminatory outcomes
- **Lack of Accountability:** Autonomous AI decision may lack human oversight
- **AI in Warfare:** Autonomous weapons raise moral and legal questions
- **Existential Risks:** AI surpassing human intelligence could pose risks

AI advancements pose **ethical challenges** that must be addressed to ensure **fairness, safety, and accountability**.

Key Ethical Issues:

1. **Job Loss Due to Automation:**

- AI-powered automation can **replace human workers** in industries like manufacturing, customer service, and transportation.
- Requires strategies for **workforce reskilling** and **universal basic income** discussions.

2. **Bias in AI Systems:**

- AI models **inherit biases** from their training data, leading to **discrimination**.
- Example: **Facial recognition bias**—AI systems misidentifying people of certain ethnic backgrounds.
- Solution: Implementing **fair AI training datasets** and **bias audits**.

3. **Lack of Accountability:**

- Autonomous AI decisions can be **opaque and difficult to interpret**.
- Raises questions about **who is responsible** when AI makes harmful decisions (e.g., **self-driving car accidents**).

4. **AI in Warfare:**

- Autonomous weapons introduce **ethical dilemmas**.
- Concerns include **lack of human oversight**, **AI-driven targeting errors**, and **escalation of conflicts**.

5. **Existential Risks:**

- **Superintelligent AI** could surpass human intelligence, leading to **unpredictable consequences**.
- Calls for **AI governance**, international **regulations**, and **alignment research** to ensure AI systems align with human values

---

Bias in AI

Bias in AI arises when machine learning models **reflect and amplify prejudices** present in their training data, leading to **unfair or discriminatory outcomes**.

Causes of AI Bias

- **Biased Training Data:** If historical data contains **discrimination** (e.g., biased hiring records), AI models will **replicate** these patterns.
- **Algorithmic Bias:** Certain **modeling choices** can unintentionally favor one group over another.
- **User Bias:** Developers and users may unknowingly **reinforce existing biases** by interpreting AI results **subjectively**.

Examples of AI Bias

- **Facial Recognition Issues:** AI systems have shown **higher error rates** for **darker skin tones** due to **unbalanced datasets**.
- **Loan & Hiring Decisions:** AI-driven hiring or lending tools may **discriminate** based on **gender, race, or socioeconomic status**.
- **Healthcare Disparities:** AI models trained on **limited demographics** can fail to diagnose conditions accurately for **underrepresented groups**.

Mitigating AI Bias

1. **Diverse & Representative Training Data:** Ensuring datasets **include all demographics** to reduce bias.
2. **Bias Audits & Fairness Testing:** Regularly evaluating AI models for **discriminatory patterns**.
3. **Interpretable AI Models:** Making AI decisions **transparent** and **explainable**.
4. **Human Oversight & Ethical AI Development:** Involving **experts from diverse fields** to **monitor AI fairness**.

Addressing AI bias is **essential** for creating **equitable, reliable, and trustworthy AI systems**.

Bayesian Games

Bayesian Games extend traditional game theory to scenarios where players have incomplete information, meaning they are uncertain about other players' characteristics, such as their preferences or strategies. Players use beliefs about these unknowns, typically modeled as probability distributions, to make decisions and strategize, incorporating the uncertainty into their strategy selection.

Key Concepts:

- **Type:** A player's private information, such as their preferences or abilities
- **Beliefs:** Probability distributions over other players' types
- **Bayes-Nash Equilibrium:** A strategy profile where each player maximizes their expected utility given their beliefs

---

Bayesian Theorem

Bayes' theorem describes how to update beliefs based on new evidence.

Formula:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

Where:

- $P(H|E)$ is the probability of hypothesis H given evidence E
- $P(E|H)$ is the likelihood of evidence given the hypothesis
- $P(H)$ is the prior probability of the hypothesis
- $P(E)$ is the probability of the evidence

Applications of the Bayes' Theorem

- **Spam filtering: C**lassifies emails as spam or not spam
- **Medical Diagnosis:** Determines the probability of disease given test results.
- **AI Decision Making:** Updates AI models based on new observations

---

P3

1. Value Iteration Agent

Approach:

Implemented **batch-style value iteration**, where each state's value at iteration `k` is updated using values from iteration `k-1`. This ensures stability and prevents bias from partial updates.

Reasoning:

- Ensures **convergence** to optimal values.
- Computes optimal **value function** first, then extracts policy.
- Used **Bellman's equation** to iteratively update values.
- Handled cases where states **lack available actions** (terminal states).
- Chose **argmax over Q-values** to derive the best policy.

---

2. Bridge Crossing Analysis

Approach:

Modified the **noise** parameter while keeping the discount factor unchanged. Lowering noise makes the agent's movements **more deterministic**, allowing it to **safely cross the bridge**.

Reasoning:

- Higher noise leads to unintended movements, making bridge crossing risky.
- Lower noise ensures the agent follows the optimal path **without falling into the chasm**.
- Kept discount factor unchanged to maintain long-term reward consideration.

---

3. Policy Optimization on DiscountGrid

Approach:

Adjusted **discount, noise, and living reward** to shape the agent's behavior:

1. **Risking the cliff:** High discount, low noise, neutral living reward.
2. **Avoiding the cliff:** High discount, low noise, slight negative living reward.
3. **Distant exit, risking cliff:** Very high discount, low noise, neutral living reward.
4. **Distant exit, avoiding cliff:** High discount, low noise, small negative living reward.
5. **Avoiding all exits:** High negative living reward to discourage termination.

Reasoning:

- **High discount** encourages long-term reward maximization (favoring distant exits).
- **Lower noise** ensures precise movement.
- **Living reward tuning** prevents premature termination or excessive stalling.

---

4. Asynchronous Value Iteration

Approach:

Implemented **cyclic updates**, where states are updated **one at a time** in order, looping back after reaching the last state.

Reasoning:

- More **efficient memory usage** than batch updates.
- Works well when **quick approximations** are needed before full convergence.
- Handled terminal states by ensuring **no unnecessary updates**.

---

5. Prioritized Sweeping Value Iteration

Approach:

Used a **priority queue** to update **high-impact states first**—states with the **largest difference** between current and expected value.

Reasoning:

- **Prioritizing high-error states speeds up convergence.**
- **Efficient updates** compared to standard asynchronous value iteration.
- **Tracked predecessors** of each state to propagate changes effectively.
- Used a **threshold** to prevent unnecessary updates for stable states.

---

6. Q-Learning Agent

Approach:

Implemented **model-free reinforcement learning**, updating Q-values based on **actual interactions** with the environment rather than a predefined MDP.

Reasoning:

- **No prior knowledge of transitions needed**—agent learns purely from experience.
- Used **ε-greedy exploration** to balance **learning new actions vs. exploiting known best actions**.
- **Breaks ties randomly** to prevent deterministic biases in unexplored states.
- Ensured **Q-values initialize at zero** and update with:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a\prime} Q(s\prime, a\prime) - Q(s,a)]$$

---

7. Epsilon-Greedy Action Selection

Approach:

Implemented **ε-greedy exploration** in getAction(). The agent selects:

- A **random action** with probability ϵ (to explore).
- The **best-known action** (highest Q-value) with probability `1 -` ϵ (to exploit).

Reasoning:

- **Encourages exploration early**, leading to better learning.
- **Balances random exploration and greedy exploitation** for efficient learning.
- **Prevents convergence to suboptimal policies** due to early overfitting.

---

8. Bridge Crossing Revisited

Approach:

Tested different \epsilon (exploration rate) and \alpha (learning rate) to find values ensuring a **99% probability** of learning the optimal policy in **50 episodes**.

Reasoning:

- A **random Q-learner** (ϵ = 1) initially explores but doesn't guarantee optimal policy.
- **Lower ϵ (e.g., 0.1) and moderate α (e.g., 0.5)** allow **sufficient exploration** but ensure **fast convergence**.
- If no such combination exists, return `"NOT POSSIBLE"`

---

9. Q-Learning in Pacman

Approach:

Used PacmanQAgent with ϵ = 0.05**,** α = 0.2**,** γ = 0.8 for training over **2000 episodes**, followed by testing.

Reasoning:

- **Training Phase:**
  - Exploration ensures Pacman **learns optimal action-values**.
  - ϵ > 0 causes **occasional suboptimal moves** but allows broad state-space coverage.
- **Testing Phase:**
  - ϵ = 0, α = 0 → **Pacman exploits learned policy** to maximize wins.
- **Performance Goal:**
  - Wins **≥ 80%** in testing.
  - **Longer training (1000+ episodes)** stabilizes the learned policy.

---

10. Approximate Q-Learning

Approach:

Replaced Q-table with **linear function approximation**:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \cdots + w_n f_n(s,a)$$

Weights updated using gradient descent:

$$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$$

where **difference** is the Q-learning error term.

Reasoning:

- **Generalizes across states**, making learning feasible for **large state spaces**.
- **Feature extraction (e.g., distance to ghosts, food locations)** helps in **state abstraction**.
- **Faster convergence than tabular Q-learning**, since **similar states share feature weights**.
- **Performance Goal:**
  - **SmallGrid** → Nearly 100% win rate.
  - **MediumGrid** & **ClassicGrid** → Efficient learning with only **50 training games**.

Table

Aspect	**Reinforcement Learning (RL)**	**Reinforcement Learning with Temporal Difference (RL TD)**	**Q-Learning**	**Approximate Q-Learning**	**Multi-Armed Bandits (MABs)**	**Decision Learning**	**POMDPs**
Learning Approach	Trial-and-error learning with feedback	Similar to RL but uses TD to update estimates with future rewards	Model-free, off-policy learning of state-action values	Extends Q-learning by approximating Q-values (e.g., via function approximation)	Focus on selecting actions to maximize reward, typically with multiple options	Learning optimal decisions based on given conditions	Deals with decisions under uncertainty and partial observability
Knowledge of Environment	Often no model of the environment (model-free)	Same as RL, but updates based on TD error	Model-free, learns from experience	Model-free, learns from experience with approximations	Assumes a fixed number of actions and rewards, no state transitions	Assumes decisions based on dynamic conditions or states	Incomplete information about the environment or system state
State Information	Often assumes full observability of state	Assumes full observability of state	Assumes full observability of state	Assumes full observability of state	No state-based modeling, only actions and rewards	Assumes some state information or context	Deals with partial observability of states

Action Selection	Agent chooses actions to maximize cumulative rewards	Action selection based on current value estimates and TD updates	Chooses actions based on learned Q-values for state-action pairs	Uses approximations (e.g., function approximation) for action-value estimates	Action selection based on reward maximization, no state info	Actions based on learned optimal decision policies	Actions based on beliefs and states, sometimes through policies
Exploration vs Exploitation	Balances exploration (trying new actions) and exploitation (using best-known actions)	Balances exploration and exploitation using TD error updates	Balances exploration and exploitation using Q-value updates	Same as Q-learning, but includes function approximation for large state spaces	Exploration is a key element to determine best actions	Decisions made by learning from past decisions and rewards	Balances exploration and exploitation under uncertainty
Example Algorithms	SARSA, Q-learning, Deep RL	TD(λ), SARSA(λ), Q-learning with TD	Q-learning	Deep Q-Network (DQN), Function Approximation Q-learning	ε-Greedy, UCB, Thompson Sampling	Markov Decision Process (MDP), POMDP algorithms	POMDP-based algorithms (Value iteration, Policy iteration)
Type of Problem	Markov Decision Processes (MDPs)	Same as RL, but uses temporal difference learning	Markov Decision Processes (MDPs)	Markov Decision Processes (MDPs) with large state spaces	Single-state decision problems with multiple actions	Sequential decision problems	Partially Observable Markov Decision Processes (POMDPs)