Week 1
Theoretical Work

Aim: Understanding fundamentals of Reinforcement Learning (RL)

What is Reinforcement Learning (RL)?

Definition: A machine learning technique where software learns to make decisions by trial and error, mimicking human behavior to achieve optimal results. Positive actions are reinforced; negative actions are discouraged.

RL vs. Supervised Learning

- Supervised Learning: Requires labeled data with clear input-output pairs. It predicts outcomes based on patterns in this data.
- Reinforcement Learning: Focuses on achieving a goal without labeled data. It rewards actions leading to the best outcomes, learning through exploration and feedback.

RL vs. Unsupervised Learning

- Unsupervised Learning: Finds hidden patterns in data without predefined outcomes. It groups data based on similarities.
- Reinforcement Learning: Has a specific end goal. It explores different strategies, continuously improving its approach to maximize successful outcomes.

Key Components of an RL System

- Agent: The learner interacting with the environment to achieve goals.
- Environment: The external system with which the agent interacts, providing feedback through rewards.
- Actions: Moves made by the agent to affect the environment.
- States: Information about the environment at a specific time, guiding the agent's decisions.
- Rewards: Feedback indicating the success of the agent’s actions (e.g., + for winning, - for losing).

Additional Concepts

- History: The sequence of all observations, actions, and rewards up to time t.

$$H_t = \{o_1, a_1, r_1, o_2, a_2, r_2, \ldots, o_t, a_t, r_t\}$$

- State: A function of the history, used to predict future actions. In RL, states with the Markov property mean future states depend only on the current state and action, not the full history.

$$P(s_{t+1} \mid s_t, a_t, H_t) = P(s_{t+1} \mid s_t, a_t)$$

Markov Decision Processes (MDPs) in RL

Definition: MDPs are a mathematical framework for modeling decision-making in environments where outcomes are partly random and partly under the agent's control. They are crucial for understanding how RL works in uncertain scenarios.

Components of an MDP

1. States (S): All possible situations the agent can encounter, containing necessary decision-making information.
2. Actions (A): Possible actions the agent can take to influence the environment.
3. Transition Probability (P): Probability of moving from one state to another given an action, representing environment dynamics.
4. Reward Function (R): Scalar reward received after transitioning between states, guiding the agent toward desirable outcomes.
5. Discount Factor (γ): Determines the importance of future rewards, balancing short-term and long-term objectives.

Agent's Objective

- Goal: Learn a policy that maximizes the expected cumulative reward, typically expressed as the sum of discounted rewards over time.

Exploration vs. Exploitation Dilemma in Reinforcement Learning

- Definition: The exploration-exploitation dilemma is a key challenge in reinforcement learning (RL). It involves deciding whether an agent should continue exploiting known rewards (exploitation) or try new actions to discover potentially better rewards (exploration).

Importance:

- Exploration: Helps the agent discover new information about the environment.
- Exploitation: Allows the agent to utilize known information to maximize immediate rewards.
- Balance: Finding the right balance is crucial to avoid getting stuck in suboptimal policies or missing out on discovering optimal ones.

Epsilon-Greedy Method

• Overview: A widely used strategy in RL for balancing exploration and exploitation.

Mechanism:

• The agent usually selects the action with the highest estimated reward (exploitation).

• Occasionally, with probability epsilon , the agent selects a random action (exploration).

Mathematical Expression

$$a_t = \begin{cases} \text{argmax}_a Q(s_t, a), & \text{with probability } 1 - \epsilon \\ \text{random action}, & \text{with probability } \epsilon \end{cases}$$

Parameters:

• Q(s_t, a) : Estimated reward for taking action a in state s_t .

• Epsilon : Tends to decrease over time as the agent gains more knowledge, shifting the focus from exploration to exploitation.

Applications of Multi-Agent Reinforcement Learning (MARL) in Computer Games

Definition: Multi-Agent Reinforcement Learning (MARL) involves multiple agents interacting within a shared environment, where each agent learns and adapts its strategies based on its experiences and the actions of others, either through collaboration, competition, or both.

Applications of MARL

1. Collaborative Agents:
 • Usage: MARL enables multiple agents to collaborate towards a common goal in team-based strategy games.
 • Example: AI-controlled units can work together to defend a base or coordinate attacks on opponents.
 2. Competitive Agents:
 • Usage: MARL is applied in competitive scenarios where agents represent different players or teams.
 • Example: Agents learn to outmaneuver opponents by adapting strategies based on other agents' actions.
 3. Non-Player Character (NPC) Behavior:
 • Usage: MARL enhances NPC behavior by allowing them to learn from interactions with the environment, other NPCs, and players.
 • Impact: Leads to more dynamic and realistic in-game behaviors.
 4. Simulation of Social Dynamics:
 • Usage: MARL models complex interactions in games that simulate social or economic dynamics.
 • Example: Multiple entities, each with unique goals and strategies, interact within the game.

Key Challenges of MARL

1. Scalability:
• Issue: Complexity increases exponentially with the number of agents, making learning more difficult.
2. Credit Assignment:
• Issue: Identifying which agent's actions contributed to a particular outcome is challenging, especially in collaborative settings.
3. Coordination and Communication:
• Issue: Effective communication and coordination between agents are difficult to achieve, especially in environments requiring collaboration for common goals.
4. Non-Stationarity:
• Issue: The environment becomes non-stationary for each agent as the policies of other agents continuously change, complicating the learning of stable policies.

Dynamic Difficulty Adjustment Using RL Agents

1. Real-Time Adaptation:
   - Function: RL agents adjust game difficulty in real-time based on player performance.
   - Example: If a player struggles, the agent might lower difficulty by weakening opponents or providing extra resources; if the player excels, the agent increases difficulty to keep the challenge engaging.
2. Personalized Gameplay Experience:
   - Function: RL agents tailor the game's difficulty to individual player skill levels, ensuring a balanced and engaging experience that's neither too easy nor too frustrating.
3. Behavioral Cloning and Imitation Learning:
   - Function: RL agents mimic human player behavior, adjusting difficulty to reflect the strategies used by players at varying skill levels.
4. Balancing Challenge and Enjoyment:
   - Challenge: RL agents must learn to balance difficulty in a way that maximizes player enjoyment, avoiding frustration while maintaining engagement through appropriate challenges.

Q-Learning Overview

Definition: Q-Learning is a type of reinforcement learning algorithm used to find an optimal action-selection policy for any finite Markov decision process (MDP). It helps an agent maximize total rewards over time through interactions with the environment.

How Q-Learning Works

1. Q-Values:

• Definition: Q-values represent the expected utility of taking a specific action in a given state.

• Updating Rule: The Q-value update formula is:

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

Parameters:

- s: Current state
- a: Action taken
- r: Reward received after taking action a
- s’: New state after taking action a
- a’: Any possible action from s’
- alpha: Learning rate
- gamma: Discount factor

2. Policy Derivation:

• Optimal Policy: The best action pi* is selected based on the highest Q-value for a state:

$$\pi^*(s) = \arg\max_a Q(s, a)$$

3. Exploration vs. Exploitation:

• Trade-Off: Balancing between exploring new actions and exploiting known actions to maximize rewards.

4. Convergence:

• Condition: With sufficient exploration, Q-Learning converges to the optimal policy for any state.

Role of the Q-Table in Decision-Making

1. Value Storage: Stores the expected rewards (Q-values) for state-action pairs.
2. Action Selection: Guides decision-making by selecting actions with the highest Q-values based on past experiences.
3. Learning from Experience: Continuously updates the Q-values based on feedback (rewards) from interactions with the environment.
4. Policy Improvement: Refining the policy by improving action predictions as more experiences accumulate.

Deep Q-Networks (DQNs)

What is Deep Q-Learning?

- Combines Q-learning with neural networks to find the optimal Q-value function.
- The model inputs the state and outputs the optimal Q-value for each possible action.

Key Concepts in Deep Q-Learning

1. Q-Value Calculation:
• Uses past experiences stored in memory to calculate Q-values at state s_t .
• The target network calculates the Q-value for the next state s_{t+1} to stabilize training.
2. Target Network:
• A copy of the Q-network used to provide stable targets during training, preventing abrupt changes in Q-values.
3. Experience Replay:
• Stores experiences as tuples (State, Next State, Action, Reward) in memory.
• A random sample from this memory is used to train the neural network, improving stability and performance.
4. Loss Function:
• Measures the squared difference between the target and predicted Q-values:

$$\text{Loss} = \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2$$

Steps in Deep Q-Learning

1. Agent State Awareness: The agent learns about the current environmental state.
2. Action Selection: The agent selects actions based on Q-values.
3. Action Execution: The agent executes the action, receiving rewards or penalties.
4. Experience Tracking: The agent tracks outcomes and updates memory.
5. Network Training: The agent uses experience replay to train the network.
6. Iteration: The process repeats to continually refine the agent’s policy.

Why ‘Deep’ Q-Learning?

Q-Learning Challenges:

• As the number of states and actions increases, the size of the Q-table becomes unmanageable.

• It’s impractical to explore every state and action combination to estimate Q-values.

Solution:

• Neural Networks: Approximates the Q-value function, allowing the system to handle large, complex environments.

• Takes the state as input and outputs Q-values for all possible actions, reducing memory and computational requirements.

Improvements Over Traditional Q-Learning

1. Scalability:

• Deep Q-Networks (DQNs): Handle large state spaces, including high-dimensional inputs like images, which are challenging for traditional Q-learning.

2. Generalization:

• Neural Networks: Enable DQNs to generalize learned behaviors to new, unseen states, making them more adaptable to dynamic environments.

3. Stable Learning:

• Experience Replay & Target Networks: Enhance stability by reducing the risk of divergence and improving convergence rates.

Steps in Reinforcement Learning Using DQNs

1. Experience Replay:
   • Stores user experiences in memory to use for training, improving learning stability.
2. Action Selection:
   • The Q-network decides the next action based on maximum Q-value output.
3. Loss Function:
   • The mean square error between the target Q-value Q* and the predicted Q-value is minimized to improve accuracy.
4. Batch Learning:
   • Experience replay enables batch learning, preventing the neural network from overfitting to specific state-action pairs.

Policy Gradient Methods in Complex Environments

Policy Definition:

- A policy is a probability distribution of actions given a state.

Overview:

- Policy Gradient Methods optimize the policy directly by maximizing expected cumulative rewards. Unlike Q-learning, which derives a policy from a value function, Policy Gradient Methods learn the policy directly.
- Suitable for high-dimensional or continuous action spaces where defining a Q-value for each possible action is impractical.

Vanilla Policy Gradient

Objective:

• Directly optimize the policy with respect to the expected discounted return:

$$\mathcal{J}(\pi_\theta) = \mathbb{E}_{\tau \sim \rho_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t R(S_t, A_t) \right]$$

Key Elements:

• Theta: Policy parameters.
• Tau: Trajectory following an unknown distribution rho_theta.
• R: Reward function.

Optimization

- Gradient Ascent: The policy parameters theta are updated to maximize the objective:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta \mathcal{J}(\pi_{\theta_k})$$

- Alpha: Learning rate.

Policy Gradient Theorem

• The gradient of the expected reward is the expectation of the reward times the gradient of the log of the policy:

$$\nabla \mathbb{E}_{\pi_\theta}\left[r(\tau)\right] = \mathbb{E}_{\pi_\theta}\left[r(\tau)\nabla \log \pi_\theta(\tau)\right]$$

REINFORCE Algorithm

Overview:

- A simple Policy Gradient Method that uses the following update rule:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot R(\tau) \right]$$

- R(tau): Cumulative reward of the trajectory.
- The gradient is estimated using Monte Carlo sampling.

Applications of Policy Gradient Methods

1. Gaming:
 • Application: Used to develop AI agents for playing complex strategy games like chess, Go, and video games.
 • Impact: AI agents can surpass human specialists through self-play, continuously adapting to opponents' strategies, enhancing the gaming experience, and pushing AI capabilities.

2. Robotics:
 • Application: Crucial in training agents for complex manipulation tasks, such as opening doors, pouring drinks, and grabbing objects.
 • Impact: By learning from raw sensory inputs (e.g., camera images), these agents operate more autonomously in dynamic environments, increasing their utility in industrial, healthcare, and domestic settings.

3. Finance:

• Application: Utilized for developing trading strategies, portfolio optimization, and algorithmic trading.

• Impact: RL agents simulate trading decisions to maximize profits and minimize risks, learning from historical market data and adapting to market conditions, thereby improving investment performance and risk management.

4. Healthcare:

• Application: Significant in developing personalized treatment plans for chronic diseases like diabetes and cancer.

• Impact: By analyzing patient data, RL agents optimize medical care recommendations based on individual characteristics, improving patient outcomes, reducing adverse effects, and optimizing resource allocation in healthcare systems.

Real-World Examples of RL Applications in Computer Games

1. StarCraft II (AlphaStar)

• Developed by: DeepMind

• Description: AlphaStar is an RL agent designed to play StarCraft II, a real-time strategy game. It achieved superhuman performance by learning complex strategies through deep reinforcement learning and imitation learning. AlphaStar manages various aspects of the game, including resource management, combat, and overall strategy.

2. Ms. Pac-Man

• Developed by: Microsoft Research (using Hybrid Reward Architecture)

• Description: Microsoft Research created an AI that achieved a perfect score in Ms. Pac-Man using reinforcement learning. The AI was built with a hybrid reward architecture, breaking down the game's objectives into smaller tasks managed by different RL agents.

3. First-Person Shooter (FPS) Games

• Example: Doom AI Competitions (e.g., VizDoom)

• Description: Reinforcement Learning has been applied to FPS games like Doom, where AI agents learn to navigate 3D environments, avoid enemies, and engage in combat. These agents are trained using deep Q-learning and policy gradients to improve their performance.

Role of RL in the Development of AlphaGo and AlphaZero

AlphaGo

- Significance: First AI to defeat a world champion Go player, marking a milestone in AI research.

Key Techniques:

1. Policy Network and Value Network:
- Policy Network: Predicted the next move.
- Value Network: Evaluated the board position.
- Training: Initially trained using supervised learning on human expert games, then refined via reinforcement learning through self-play.

2. Reinforcement Learning via Self-Play:
- Process: AlphaGo played against itself (self-play) to improve.
- Policy Gradient Methods: Adjusted neural network weights based on game outcomes to maximize the expected outcome (winning).

3. Monte Carlo Tree Search (MCTS):
- Combination: RL combined with MCTS to simulate possible future moves and evaluate the best course of action.
- Efficiency: Improved the efficiency and strength of the search by evaluating board positions.

AlphaZero

• Advancement: A more generalized version of AlphaGo, applying RL to other games like chess and shogi.

Key Innovations:

1. No Prior Human Knowledge:

• Learning: Started with random play and learned entirely through self-play, improving continuously using RL.

• Architecture: Employed the same deep neural networks and MCTS as AlphaGo, but without any initial human input.

2. Generalized Learning:

• Capability: Able to learn and master any two-player game with perfect information (e.g., chess, shogi) through RL.

• Self-Improvement: Played millions of games against itself, gradually refining its strategy.

3. Efficiency and Performance:

• Outcome: Demonstrated superhuman performance across multiple domains, showcasing RL’s potential to develop AI that can surpass human expertise by learning purely from experience.

Tools and Frameworks for Experimenting with RL in Games

- OpenAI Gym
- Unity ML-Agents Toolkit
- Stable Baselines3
- RLlib
- DeepMind Lab
- Gym Retro
- CoppeliaSim (formerly V-REP)
- Google Research Football
- SuperMarioBros

Challenges in Implementing RL in Real-Time or Complex Computer Games

- Opacity or Non-Interpretability: Modern neural networks often lack understandable reasoning steps and perform poorly in terms of activation methods compared to human intelligence.

- Poor Generalization to Samples Outside Training Distribution: Neural networks may struggle with data distributions different from the training set, requiring re-learning when game parameters change.

- Data Inefficiency: Neural networks typically need large amounts of data, increasing model complexity and computational demands.

- Developing High-Level Abstractions for Causal or Analogical Reasoning: Challenges in incorporating reasoning and causal principles, which are essential for improving AI.

- Low Sample Efficiency: Many current DRL methods suffer from poor sample efficiency, leading to slower performance enhancement.

- Limited Accuracy and Transferability: DRL methods often have limited transferability to different domains or datasets, necessitating improved methods for broader application.

- Issues of Reproducibility: Reproducing results becomes challenging as neural networks grow in complexity.

- Implementation in Real-Life Scenarios: Bridging the “reality gap” by mimicking desired behavior and designing robust algorithms is crucial for effective real-world applications.

Influence of Reward Functions on Agent Behavior in a Game Environment

1. Guiding Behavior:
 • Reward functions provide feedback on action quality, encouraging repetition of positive actions and discouraging negative ones.
2. Defining Objectives:
 • They define the goals for the agent, such as collecting items or reaching locations, guiding the agent's focus.
3. Influencing Exploration vs. Exploitation:
 • Reward functions affect the balance between trying new actions (exploration) and relying on known strategies (exploitation).
4. Encouraging Efficient Strategies:
 • They incentivize efficient task completion, such as rewarding faster completion or optimal resource usage.
5. Shaping Long-Term Behavior:
 • Reward functions can guide the agent's long-term strategy, emphasizing cumulative achievements or future potential over immediate rewards.

Effects of Reward Shaping on RL Agents’ Performance

Reward Shaping Definition:

• The use of small intermediate “fake” rewards to guide the learning agent toward more rapid convergence.

Application:

• Prior knowledge about what constitutes a good solution can guide the agent, such as moving toward a reward of +1 and away from -1 in navigation tasks.

Purpose:

• Accelerates the learning process and enhances the final solution by guiding the reinforcement learner toward desired behaviors.

Method:

• Involves slightly modifying the reinforcement learning algorithm to provide additional information, ensuring that optimality is maintained.

Domain Knowledge:

• This supplementary information is referred to as domain knowledge, which includes pertinent details about the domain that the human modeler is aware of during the model’s construction.

Shaped Reward in Reinforcement Learning

Q-Function Update in TD Learning:

• In Temporal Difference (TD) learning, a Q-function is updated when a reward is received:

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

Reward Shaping Approach:

• Instead of modifying the reward function r, additional rewards F(s, s') are provided for specific actions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \underbrace{F(s,s')}_{\text{additional reward}} + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Purpose:

• The function F(s, s’) provides additional rewards when transitioning from state s to s’, using heuristic domain knowledge.

Shaped Reward Definition:

• The sum r + F(s, s’) is considered the shaped reward for an action.

• Overall Shaped Reward for an Episode:

$$G^{\Phi} = \sum_{i=0}^{\infty} \gamma^{i} \left(r_i + F(s_i, s_{i+1})\right)$$

Positive and Negative Shaping:

• F(s, s’) > 0 provides positive reinforcement for transitioning from s to s’.

• F(s, s’) < 0 provides negative reinforcement, discouraging the transition from s to s’.

Effects of Reward Shaping on RL Performance

Accelerated Learning:

• Provides additional feedback, speeding up the learning process by offering more frequent signals about progress.

Guided Exploration:

• Directs the agent toward more promising areas, reducing the likelihood of getting stuck in suboptimal policies.

Improved Performance:

• Encourages behaviors that align closely with the desired outcomes, leading to more competitive or human-like AI behavior.

Risk of Overfitting:

• If reward shaping is too specific, the agent may overfit to those rewards, performing poorly in different scenarios. It's crucial to ensure shaping doesn't limit the agent's ability to generalize.

Thanks for listening!