Week 1 - Theoretical Work

Team: Game & RL
Aim: Understanding fundamentals of Reinforcement Learning (RL)

What is Reinforcement Learning, and how does it differ from other types of machine learning (e.g., supervised, unsupervised learning)?

Reinforcement learning (RL) is a machine learning (ML) technique that trains software to make decisions to achieve the most optimal results. It mimics the trial-and-error learning process that humans use to achieve their goals. Software actions that work towards your goal are reinforced, while actions that detract from the goal are ignored.

Reinforcement learning vs. supervised learning

- In supervised learning, you define both the input and the expected associated output. For instance, you can provide a set of images labeled dogs or cats, and the algorithm is then expected to identify a new animal image as a dog or cat.
- Supervised learning algorithms learn patterns and relationships between the input and output pairs. Then, they predict outcomes based on new input data. It requires a supervisor, typically a human, to label each data record in a training data set with an output.
- In contrast, RL has a well-defined end goal in the form of a desired result but no supervisor to label associated data in advance. During training, instead of trying to map inputs with known outputs, it maps inputs with possible outcomes. By rewarding desired behaviors, you give weightage to the best outcomes.

Reinforcement learning vs. unsupervised learning

- Unsupervised learning algorithms receive inputs with no specified outputs during the training process. They find hidden patterns and relationships within the data using statistical means. For instance, you could provide a set of documents, and the algorithm may group them into categories it identifies based on the words in the text. You do not get any specific outcomes; they fall within a range.
- Conversely, RL has a predetermined end goal. While it takes an exploratory approach, the explorations are continuously validated and improved to increase the probability of reaching the end goal. It can teach itself to reach very specific outcomes.

How do Markov Decision Processes (MDPs) model the environment in RL?

A Markov Decision Process is a mathematical framework used to describe an environment in decision-making scenarios where outcomes are partly random and partly under the control of a decision-maker. MDPs provide a formalism for modeling decision-making in situations where outcomes are uncertain, making them essential for Reinforcement Learning.

Components of an MDP:

An MDP is defined by a tuple (S,A,P,R,γ) where:

1. States (S): A finite set of states representing all possible situations in which the agent can find itself. Each state encapsulates all the relevant information needed to make a decision.
2. Actions (A): A finite set of actions available to the agent. At each state, the agent can choose from these actions to influence its environment.

3. Transition Probability (P): A probability function $P(s_{t+1} \mid s_t, a_t)$ that defines the probability of transitioning from state s_t to state s_{t+1} after taking action a_t. This encapsulates the dynamics of the environment.

4. Reward Function (R): A reward function $R(s_t, a_t, s_{t+1})$ that provides a scalar reward received after transitioning from state s_t to state s_{t+1} due to action a_t. This reward guides the agent towards desirable outcomes.

5. Discount Factor (γ): A discount factor $\gamma \in [0,1)$ that determines the importance of future rewards. A discount factor close to 1 makes the agent prioritize long-term rewards, while a factor close to 0 makes it focus on immediate rewards.

The agent interacts with the environment modeled as an MDP by observing the current state s_t, taking an action a_t, transitioning to a new state s_{t+1}, and receiving a reward r_t.

The agent's goal is to learn a policy that maximizes the expected cumulative reward, often expressed as the sum of discounted rewards over time.

What are the key components of an RL system (e.g., agent, environment, actions, states, rewards)?

The key components of RL include an agent, an environment, actions taken by the agent, observations of the environment and reward signal. The agent is the learner that interacts with the environment to achieve specific goals. Environment represents the external system with which the agent interacts and receives feedback. A reward is a feedback signal from the environment that indicates how well the agent is doing at step t and the agent's job is to maximize the reward. Examples of reward are $+$ for winning a game and $-$ for losing a game in game playing, $+$ for straight motion and $-$ for falling in humanoid robots. But, sometimes it might be better to sacrifice immediate reward to gain long term reward like refueling stunt chopper that might prevent future crash. In RL the interaction between the agent and the environment unfolds over discrete steps. At each step t, the agent executes action a_t, receives observation o_t and reward r_t. Meanwhile, the environment receives action A_t, emits observation O_{t+1} and reward R_{t+1}.

Other key components in RL are history and state. History is the sequence of observations, actions and rewards. i.e all observable variables up to step t.

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

What happens next depends on the history of actions taken, observation and reward given by environment.

State is the information used to determine what happens next and this state is the function of the history. It is a representation of the environment at a specific time that contains all the relevant information needed to predict what will happen next. In RL, the state s_t is a function of the history H_t.

$$s_t = \mathcal{F}(H_t)$$

- The environment state is the environment's private state representation which is usually not visible to the agent.
- Agent state is the agent's internal representation which is the information used to train RL algorithm.
- Information state is the accumulation of all useful information from the history. It is also commonly known as Markov state. The Markov state defines that the future is independent of the past given the present. i.e. Once the state is known, the history is not useful anymore. A state is called a Markov state if it satisfies the Markov property, which states that the future is independent of the past given the present state. Formally:

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

  This means that the future state s_{t+1} depends only on the current state s_t and the action a_t, and not on the history H_t.

Explain the exploration vs. exploitation dilemma in RL and why it is important.

A key challenge that arises in reinforcement learning (RL) is the trade-off between exploration and exploitation. This challenge is unique to RL and doesn't arise in supervised or unsupervised learning.

In reinforcement learning, whenever agents get a situation in which they have to make a difficult choice between whether to continue the same work or explore something new at a specific time, then, this situation results in Exploration-Exploitation Dilemma because the knowledge of an agent about the state, actions, rewards and resulting states is always partial.

Exploration is any action that lets the agent discover new features about the environment, while exploitation is capitalizing on knowledge already gained. If the agent continues to exploit only past experiences, it is likely to get stuck in a suboptimal policy. On the other hand, if it continues to explore without exploiting, it might never find a good policy.

An agent must find the right balance between the two so that it can discover the optimal policy that yields the maximum rewards.

Epsilon-Greedy Method

The ϵ-greedy method is a simple and widely used algorithm in reinforcement learning (RL) for balancing exploration and exploitation while selecting actions.

In the ϵ-greedy method, an agent typically chooses the action with the highest estimated reward (exploitation). However, with a probability ϵ (epsilon), the agent instead chooses an action at random (exploration). This ensures that the agent does not get stuck in a local optimum and continues to explore other possible actions that might lead to better rewards in the long term.

Mathematically, the action a_t at time step t is chosen as follows:

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

Here, $Q(s_t, a)$ is the estimated reward for taking action a in state s_t. The parameter ϵ typically starts high and is decreased over time as the agent learns more about the environment, allowing the agent to favor exploitation more as its knowledge improves.

What Are the Applications of Multi-Agent Reinforcement Learning (MARL) in Computer Games, and What Are the Key Challenges of This Approach?

Multi-Agent Reinforcement Learning (MARL) refers to a subfield of reinforcement learning where multiple agents interact within a shared environment, learning and adapting their strategies based on both their experiences and the actions of other agents. Each agent aims to optimize its own performance, either through collaboration, competition, or a mix of both, depending on the specific application or environment.

Applications of MARL in Computer Games

- **Collaborative Agents:** MARL can be utilized in games where multiple agents must work together towards a common goal. For instance, in a team-based strategy game, different AI-controlled units can collaborate to achieve objectives like defending a base or attacking an opponent.
- **Competitive Agents:** MARL is also applicable in competitive scenarios, where agents represent different players or teams competing against each other. Each agent learns to outmaneuver its opponents by adapting its strategies based on the actions of others.

- **Non-Player Character (NPC) Behavior:** MARL can enhance the behavior of NPCs by allowing them to learn from interactions with both the environment and other NPCs or players, leading to more dynamic and realistic in-game behaviors.
- **Simulation of Social Dynamics:** In games that simulate social interactions or economics, MARL can model complex interactions between multiple entities, each with its own goals and strategies.

Key Challenges of MARL

- **Scalability:** As the number of agents increases, the complexity of the environment grows exponentially, making it difficult for each agent to learn effectively.
- **Credit Assignment:** Determining which agent's actions contributed to a particular outcome can be challenging, especially in collaborative environments where multiple agents are involved in achieving a goal.
- **Coordination and Communication:** Ensuring that agents can effectively communicate and coordinate their actions is a significant challenge, particularly in environments where they must work together to achieve common objectives.
- **Non-Stationarity:** In a multi-agent setting, the environment becomes non-stationary from the perspective of each agent, as the policies of other agents are continuously changing. This makes it harder for each agent to learn a stable policy.

How Can RL Agents Be Used to Dynamically Adjust the Perceived Difficulty Level by Human Players?

Dynamic Difficulty Adjustment Using RL Agents:

- **Real-Time Adaptation:** RL agents can be trained to adjust the game's difficulty in real-time by analyzing the player's performance. For example, if a player is struggling, the RL agent might reduce the difficulty by weakening opponents or providing extra resources, and if the player is excelling, it might increase the difficulty to maintain a challenging experience.
- **Personalized Gameplay Experience:** By continuously monitoring player behavior, RL agents can personalize the difficulty to match the skill level of individual players, ensuring that the game remains engaging without being frustrating or too easy.
- **Behavioral Cloning and Imitation Learning:** RL agents can use techniques like behavioral cloning to mimic the behavior of human players, adjusting the difficulty to mirror the strategies and tactics typically used by players at different skill levels.
- **Balancing Challenge and Enjoyment:** The key challenge in dynamic difficulty adjustment is balancing the challenge to ensure the game remains enjoyable. RL agents need to be trained not just to adjust difficulty mechanically but to do so in a way that maximizes player satisfaction, which may involve learning complex patterns of player engagement and frustration.

How does Q-Learning work, and what role does the Q-table play in decision-making?

What is Q-Learning?

Q-learning is a reinforcement learning algorithm that finds an optimal action-selection policy for any finite Markov decision process (MDP). It helps an agent learn to maximize the total reward over time through repeated interactions with the environment, even when the model of that environment is not known.

How Q-Learning Works?

1. **Learning and Updating Q-values:** The algorithm maintains a table of Q-values for each state-action pair. These Q-values represent the expected utility of taking a given action in a given state and following the optimal policy after that. The Q-values are initialized arbitrarily and are updated iteratively using the experiences gathered by the agent.

2. **Q-value Update Rule:** The Q-values are updated using the formula:

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

   Where:

   - s is the current state.
   - a is the action taken.
   - r is the reward received after taking action a in state s.
   - s' is the new state after taking action a.
   - a' is any possible action from the new state s'.
   - α is the learning rate ($0 < \alpha \leq 1$).
   - γ is the discount factor ($0 \leq \gamma < 1$).

3. **Policy Derivation:** The policy determines what action to take in each state and can be derived from the Q-values. Typically, the policy chooses the action with the highest Q-value in each state (exploitation), though sometimes a less optimal action is chosen for exploration purposes. Once the Q-table has been sufficiently updated through exploration, the optimal policy π^* can be derived by choosing the action with the highest Q-value for each state:

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

4. **Exploration vs. Exploitation:** Q-learning manages the trade-off between exploration (choosing random actions to discover new strategies) and exploitation (choosing actions based on accumulated knowledge). Techniques like the epsilon-greedy strategy, where the agent mostly takes the best-known action but occasionally tries a random action, often manage the balance between these.

5. **Convergance:** Under certain conditions, such as ensuring all state-action pairs are visited an infinite number of times, Q-learning converges to the optimal policy and Q-values that give the maximum expected reward for any state under any conditions.

Role of the Q-Table in Decision-Making

- **Value Storage:** The Q-table stores the expected rewards (Q-values) for each state-action pair. This table is central to the Q-Learning algorithm as it represents the agent's knowledge about the environment and the rewards associated with different actions.

- **Action Selection:** During the decision-making process, the agent uses the Q-table to choose actions. It typically selects actions that have the highest Q-value for the current state, which reflects the expected cumulative reward based on past experiences.

- **Learning from Experience:** The Q-table is continuously updated based on the agent's interactions with the environment. As the agent explores and receives feedback (rewards), the Q-values are adjusted to better estimate the true expected rewards of actions. This iterative process helps the agent to learn an optimal policy over time.

- **Policy Improvement:** The Q-table enables the agent to refine its policy. By updating the Q-values, the agent improves its ability to predict the best actions to take in various states, leading to better decision-making as more experiences are accumulated.

What are Deep Q-Networks (DQNs), and how do they improve upon traditional Q-Learning?

What is Deep Q-Learning?

The deep Q-learning model breaks the chain in order to find the optimal Q-value function. It determines this by combining Q-learning and a neural network. The uses of the deep Q-learning algorithm can be stated as finding the input and the optimal Q-value for all possible actions as the output.

In deep Q-learning, past experiences are stored in memory and the future action depends on the Q-network output. It is how the Q-network calculates the Q-value at state s_t. Similarly, the target network (neural network) determines the Q-value for state s_{t+1} (next state) to stabilize the training.

As an additional feature, it copies the Q-value count as the training dataset for each iteration of the Q-value in the Q-network, thereby blocking abrupt increases in the Q-value count.

The deep Q-learning algorithm relies on neural networks and Q-learning. In this case, the neural network stores experience as a tuple in its memory with a tuple that includes ⟨State, Next State, Action, Reward⟩.

A random sample of previous data increases the stability of neural network training. As a result, deep Q-learning uses another concept to boost the agent's performance—experience replay, which stores experiences from the past.

The target network uses experience replay to determine the Q-value, while the Q-network uses it for training. Calculating the loss becomes easier when the squared difference between the target and predicted Q-values is known. The equation is given below:

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

To understand deep Q-learning better, we need to break it down into these steps:

1. First, the agent is informed about the environmental state.
2. The agent uses Q-values of possible actions based on the provided state.
3. The agent applies a Q-value to the action, resulting in higher rewards and lower punishment.
4. We then follow up on rewards and next steps.
5. The agent keeps track of previous experiences from the action in an experience replay memory.
6. The agent uses experience replay memory to train the networks.
7. We continually repeat steps 2 through 6 according to the agent's state.

Why 'deep' Q-learning?

The Q-learning algorithm creates a cheat sheet for agents in a simple but quite powerful manner. By doing so, the agents can determine exactly what action to take.

Consider an environment with 10,000 states and 1,000 actions per state. Wouldn't the cheat sheet be too long? There would be a table with 10 million cells. Eventually, things would spiral out of control. Thus, it would be impossible to estimate the Q-value for new states based on previously explored states.

Two main problems would arise:

- As the number of states increases, the amount of memory required to save and update the table would expand.
- It would become impossible to explore every state to create the required Q-table in the time required.

A neural network becomes helpful in approximating the Q-value function in deep Q-learning. The state is taken as the input, and Q-values for all possible actions are generated as the output.

Improvements Over Traditional Q-Learning:

- Scalability: DQNs can handle environments with large state spaces, such as those with high-dimensional inputs like images or continuous state spaces, which would be infeasible for traditional Q-Learning.
- Generalization: By using a neural network, DQNs can generalize learned behaviors to new, unseen states, making them more adaptable to dynamic and complex environments.
- Stable Learning: Experience replay and the use of a target network help stabilize the learning process, reducing the risk of divergence and improving the convergence rate compared to traditional Q-Learning.

The following steps are involved in reinforcement learning using deep Q-learning networks (DQNs):

1. User memories get stored with past experiences. This feature is called experience replay.
2. A Q-network decides the next action based on its maximum output.
3. In the loss function, the mean square error is the difference between the target Q-value Q^* and the predicted Q-value.
4. To prevent the neural network from affecting the distribution of states, actions, rewards, and next_states it encounters, deep Q-learning uses experience replay to learn in small batches. Moreover, the agent doesn't need training after each step.

Describe Policy Gradient Methods and their applications in complex environments like games.

Policy: A policy is defined as the probability distribution of actions given a state.

Policy Gradient Methods are a class of algorithms in reinforcement learning (RL) that directly optimize the policy, which is a mapping from states to actions, by maximizing the expected cumulative reward. Unlike value-based methods like Q-Learning, which learn a value function and derive a policy indirectly, policy gradient methods learn the policy directly. This makes them well-suited for environments with high-dimensional or continuous action spaces, where defining a Q-value for each possible action is impractical.

Vanilla Policy Gradient

In value-based methods, our objective was to estimate the action values to derive good policies. Policy Gradient methods focus directly on optimizing the policy with respect to the objective function, which is the expected discounted return.

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

- The policy is parameterized by θ.
- The trajectory τ is random and follows an unknown probability distribution ρ_θ.
- R is the reward function given a state S_t and an action A_t.

Our goal is to maximize this objective function depending on θ. Naturally, to optimize the policy, policy gradient methods use gradient ascent. The gradient of the objective function with respect to the policy parameters θ is calculated and used to update the policy:

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

where α is the learning rate.

To find the gradient of the objective above which contains the expectation, one can use the following theorem:

The Policy Gradient Theorem: The derivative of the expected reward is the expectation of the product of the reward and the gradient of the log of the policy π_θ:

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

where we represent the total reward for a given trajectory τ as $r(\tau)$

REINFORCE Algorithm

The REINFORCE algorithm is one of the simplest policy gradient methods. It uses the following update rule:

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

where $R(\tau)$ is the cumulative reward of the trajectory. The gradient is estimated using Monte Carlo sampling.

Applications of Policy Gradient Methods

Gaming: In the gaming industry, developers employ policy gradient methods to develop AI agents capable of playing complex strategy games such as chess, Go, and video games. RL agents can become superhuman performers in these games by learning to outperform human specialists through self-play or contact with human players. By continuously learning and adapting to opponents' strategies, RL agents demonstrate remarkable strategic intelligence and adaptability, enhancing the gaming experience for players and pushing the boundaries of AI capabilities.

Robotics: Policy gradient methods play a crucial role in training agents to execute intricate manipulation tasks in robotics. It takes excellent flexibility and suppleness to open doors, pour drinks, and grab objects. By directly optimizing the policy, RL agents can learn to perform these tasks from raw sensory inputs, such as camera images or tactile feedback. It enables robots to operate more autonomously in dynamic environments, enhancing their utility in industrial, healthcare, and domestic settings.

Finance: In the financial sector, practitioners utilize policy gradient methods to develop sophisticated trading strategies for stock market prediction, portfolio optimization, and algorithmic trading. With these techniques, RL agents can simulate trading decisions as a series of steps that they must take to limit risk and maximize profits. By learning from historical market data and adapting to changing market conditions, RL agents can identify profitable trading opportunities and execute trades precisely, contributing to improved investment performance and risk management.

Healthcare: Policy gradient methods have significant applications in healthcare, particularly in developing personalized treatment plans for patients with chronic diseases such as diabetes and cancer. Using patient data, RL agents can make the best medical care recommendations based on individual characteristics such as lifestyle, medical history, and genetics. This personalized approach to healthcare delivery promises to improve patient outcomes, reduce treatment-related adverse effects, and optimize resource allocation within healthcare systems.

What are some real-world examples of RL applications in computer games?

Here are some examples of how reinforcement learning has been applied in gaming:

- **StarCraft II (AlphaStar)**

  Developed by: DeepMind

  Description: AlphaStar is an RL agent designed to play the real-time strategy game StarCraft II. It demonstrated superhuman performance by learning strategies through deep reinforcement learning and imitation learning, managing the game's complex decision space, including resource management, combat, and strategy.

- **Ms. Pac-Man**

  Developed by: Microsoft Research (using Hybrid Reward Architecture)

  Description: Microsoft Research used reinforcement learning to create an AI that achieved a perfect score in the arcade game Ms. Pac-Man. The AI was built using a hybrid reward architecture, which broke down the game's objectives into smaller tasks that were handled by different RL agents.

- **First-Person Shooter (FPS) Games**

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

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

How did RL contribute to the development of AlphaGo and AlphaZero?

Reinforcement Learning (RL) played a pivotal role in the development of AlphaGo and AlphaZero, two groundbreaking AI systems by DeepMind that achieved unprecedented success in the complex game of Go and other board games.

AlphaGo

AlphaGo was the first AI to defeat a world champion Go player, and its development marked a significant milestone in AI research. The core of AlphaGo's success lies in the combination of two key techniques: deep neural networks and reinforcement learning.

- **Policy Network and Value Network:** AlphaGo used two neural networks: a policy network to predict the next move and a value network to evaluate the position on the board. These networks were trained using supervised learning on a dataset of human expert games and then refined using reinforcement learning through self-play.
- **Reinforcement Learning via Self-Play:** After the initial training, AlphaGo improved its performance by playing against itself (self-play). This self-play process used a technique called *policy gradient* methods to adjust the weights of the neural networks based on the outcomes of these games. The policy network was trained to maximize the expected outcome (winning), and the value network was trained to predict the probability of winning from any given board position
- **Monte Carlo Tree Search (MCTS):** AlphaGo combined RL with MCTS, a method that simulates possible future moves to evaluate the best course of action. The policy network guided the tree search, and the value network provided evaluations for the board positions, significantly improving the efficiency and strength of the search

AlphaZero

AlphaZero, a more generalized version of AlphaGo, extended the use of RL to other games like chess and shogi, demonstrating a more versatile and powerful approach.

- **No Prior Human Knowledge:** Unlike AlphaGo, which was initially trained on human games, AlphaZero started with random play and learned entirely through self-play, using reinforcement learning to continuously improve. It employed the same deep neural network architecture and MCTS as AlphaGo, but without any initial human input
- **Generalized Learning:** AlphaZero's architecture allowed it to learn and master any two-player game with perfect information (like chess and shogi) through reinforcement learning. The process involved the agent playing millions of games against itself, gradually improving its strategy by learning from the outcomes of these games
- **Efficiency and Performance:** AlphaZero demonstrated that reinforcement learning could be used to achieve superhuman performance across multiple domains. Its success in mastering games without human data showed the potential of RL for developing AI that can surpass human expertise by learning purely from experience

What tools and frameworks (e.g., OpenAI Gym) are available for experimenting with RL in games?

When experimenting with reinforcement learning (RL) in games, several tools and frameworks can be very helpful:

- **OpenAI Gym:** A popular toolkit for developing and comparing RL algorithms. It provides a variety of environments, including classic control problems and Atari games, which are great for testing RL agents.
- **Unity ML-Agents Toolkit:** This toolkit allows you to train RL agents within Unity environments. It provides a rich set of features for creating complex simulations and games.
- **Stable Baselines3:** A set of reliable implementations of RL algorithms built on top of PyTorch, designed to work seamlessly with OpenAI Gym environments.
- **RLlib:** Part of the Ray ecosystem, RLlib is a scalable RL library that supports various environments and algorithms. It's useful for both research and production.
- **DeepMind Lab:** A 3D learning environment that enables research into RL and AI. It offers complex environments for testing RL algorithms.
- **Gym Retro:** A library that allows you to run classic video games from the SEGA Genesis and other consoles within OpenAI Gym environments.
- **CoppeliaSim (formerly V-REP):** A versatile robot simulation environment that supports RL and is useful for simulating robotic tasks and environments.
- **Google Research Football:** An RL environment focused on soccer, where agents can be trained to play and strategize in football games.
- **SuperMarioBros:** A library that provides an environment for the classic Super Mario Bros game, allowing you to train RL agents to play and learn from the game.

Discuss the challenges faced in implementing RL in real-time or complex computer games

- **Opacity or Non-interpretability:** Modern neural networks often lack understandable reasoning steps and have lower performance in terms of activation methods compared to human intelligence.
- **Poor Generalization to Samples Outside Training Distribution:** Neural networks can struggle with data distributions different from the training set. For example, a DRL agent might need to re-learn a game if its parameters change, unlike humans who can quickly adapt using past experiences.
- **Data Inefficiency:** Neural networks typically require large amounts of data, which can increase model complexity and computational demands. For instance, a DRL system might need millions of frames in a video game to achieve high performance, whereas humans may need far fewer frames.
- **Develop High-Level Abstractions Necessary for Causal or Analogical Reasoning:** Challenges in neural networks include abstraction and reasoning. Neurosymbolic AI aims to address these by incorporating human-like reasoning and causal principles. Tests such as Stanford's "Strength of Evidence" have highlighted the need for improved causal reasoning in AI.
- **Low Sample Efficiency:** Efficient algorithms should quickly learn and utilize experiences to improve performance. Current DRL methods often suffer from poor sample efficiency, leading to slower performance enhancement.
- **Limited Accuracy and Transferability:** While DRL methods can achieve high accuracy in familiar tasks, their transferability to different domains or datasets is limited. There is a need for methods that improve accuracy and transferability across various applications.
- **Issues of Reproducibility:** As neural networks grow in complexity, reproducing results becomes challenging. Methods like the minimal traces model, experiment tracking, and metadata platforms are being developed to enhance reproducibility.

- **Implementation in Real Life Scenarios:** RL agents often rely on trained data rather than exploring real environments. Techniques to bridge the "reality gap" include mimicking desired behavior, using verified simulations, and designing robust algorithms.

How do reward functions influence the behavior of an agent in a game environment?

In reinforcement learning (RL), reward functions play a crucial role in shaping the behavior of an agent within a game environment. Here's how they influence the agent's behavior:

- **Guiding Behavior:** The reward function provides feedback to the agent about the quality of its actions. Positive rewards encourage the agent to repeat those actions, while negative rewards (or penalties) discourage them. This feedback helps the agent learn which actions lead to desirable outcomes.
- **Defining Objectives:** The reward function defines the goals and objectives for the agent. For example, in a game, a reward function might give points for collecting items or reaching certain locations, guiding the agent to focus on these goals.
- **Influencing Exploration vs. Exploitation:** The design of the reward function can impact the balance between exploration (trying new actions) and exploitation (relying on known successful actions). A well-designed reward function encourages exploration of promising strategies while still focusing on rewarding high-performing actions.
- **Encouraging Efficient Strategies:** A reward function can drive the agent to find efficient strategies by rewarding faster completion of tasks or optimal resource usage. For example, in a racing game, a reward function might give higher scores for completing laps in less time or with fewer mistakes.
- **Shaping Long-Term Behavior:** The reward function can influence the agent's long-term strategy and planning. For example, it might be designed to provide rewards based on cumulative achievements or future potential, guiding the agent to consider long-term outcomes rather than just immediate rewards.

What Are the Effects of Reward Shaping Techniques Specifically Developed for a Game on the Performance of RL Agents?

Reward Shaping

Definition – Reward Shaping: *Reward shaping* is the use of small intermediate 'fake' rewards given to the learning agent to assist in more rapid convergence.

In numerous applications, it is possible to have prior knowledge of what constitutes a good solution. For instance, in a simple navigation task, it is evident that moving towards a reward of $+1$ and away from a reward of -1 are likely to represent good solutions.

To accelerate the learning process and enhance the final solution, the reinforcement learner can be guided towards the desired behavior.

This objective can be achieved by slightly modifying the reinforcement learning algorithm, providing additional information to assist the agent, while also ensuring optimality is maintained.

This supplementary information is referred to as *domain knowledge*—pertinent details about the domain that the human modeler is aware of during the construction of the model to be solved.

Shaped Reward

In TD learning methods, we update a Q-function when a reward is received. For example, for 1-step Q-learning:

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

The approach to reward shaping is not to modify the reward function or the received reward r, but to just give some additional reward for some actions:

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

The purpose of the function is to give an additional reward $F(s,s')$ when any action transitions from state s to state s'. The function $F : S \times S \rightarrow \mathbb{R}$ provides *heuristic domain knowledge* to the problem that is typically manually programmed.

We say that $r + F(s,s')$ is the **shaped reward** for an action.

Further, we say that $G^{\Phi} = \sum_{i=0}^{\infty} \gamma^i \left(r_i + F(s_i, s_{i+1}) \right)$ is the shaped reward for the entire episode.

If we define $F(s,s') > 0$ for states s and s', then this provides a small positive reward for transitioning from s to s', thus encouraging actions that transition from s to s' in future exploitation. If we define $F(s,s') < 0$ for states s and s', then this provides a small negative reward for transitioning from s to s', thus discouraging actions that transition like this in future exploitation.

Effects of Reward Shaping on RL Performance:

- **Accelerated Learning:** By providing additional feedback in the form of intermediate rewards, reward shaping can significantly speed up the learning process. The agent receives more frequent signals about its progress, reducing the time it takes to learn effective strategies.
- **Guided Exploration:** Reward shaping can guide the agent towards more promising areas of the state space by making certain actions more attractive. This reduces the likelihood of the agent getting stuck in suboptimal policies or exploring irrelevant parts of the environment.
- **Improved Performance:** Well-designed reward shaping can lead to improved overall performance by encouraging behaviors that align more closely with the desired outcome of the game. This can result in more competitive or human-like AI behavior.
- **Risk of Overfitting:** One potential downside is that if the reward shaping is too specific to the training environment, the agent may overfit to those particular rewards and perform poorly in slightly different scenarios. Care must be taken to ensure that the shaping does not limit the agent's ability to generalize.

References

[1] Amazon Web Services. *What is Reinforcement Learning?* Available at: https://aws.amazon.com/what-is/reinforcement-learning/

[2] GeeksforGeeks. *What is Markov Decision Process (MDP) and its Relevance to Reinforcement Learning?* Available at: https://www.geeksforgeeks.org/what-is-markov-decision-process-mdp-and-its-relevance-to-reinforcement-learning/

[3] Khadka, P. *A Brief Overview to Reinforcement Learning*. Medium. Available at: https://medium.com/@pranjalkhadka/a-brief-overview-to-reinforcement-learning-10ec6b517eb7

[4] Javatpoint. *Exploitation and Exploration in Machine Learning.* Available at: https://www.javatpoint.com/exploitation-and-exploration-in-machine-learning

[5] Scribbr. *What is the Exploration vs Exploitation Trade-Off in Reinforcement Learning?* Available at: https://www.scribbr.com/frequently-asked-questions/what-is-the-exploration-vs-exploitation-trade-off-in-reinforcement-learning/#:~:text=Exploration%20is%20any%20action%20that,stuck%20in%20a%20suboptimal%20policy

[6] Naukri Learning. *Epsilon-Greedy Algorithm: A Complete Guide*. Available at: https://www.naukri.com/code360/library/epsilon-greedy-algorithm

[7] Pineau, J. *I've Been Thinking About Multi-Agent Reinforcement Learning (MARL), And You Probably Should Be Too*. Available at: https://towardsdatascience.com/ive-been-thinking-about-multi-agent-reinforcement-learning-marl-and-you-probably-should-be-too-8f1e241606ac

[8] Advanced Operations Research Academy. *Introduction to Hierarchical Multi-Agent Reinforcement Learning (H-MARL)*. Available at: https://advancedoracademy.medium.com/introduction-to-hierarchical-multi-agent-reinforcement-learning-h-marl-aca09c793630

[9] Lievrouw, R. *Hierarchical Multi-Agent Reinforcement Learning: An Overview*. Master's Thesis, Ghent University, 2021. Available at: https://libstore.ugent.be/fulltxt/RUG01/002/945/739/RUG01-002945739_2021_0001_AC.pdf

[10] Towards Data Science. *Reinforcement Learning Explained Visually: Part 4 - Q-Learning Step-by-Step.* Available at: https://towardsdatascience.com/reinforcement-learning-explained-visually-part-4-q-learning-step-by-step-b65efb731d3e

[11] Simplilearn. *What is Q-Learning?* Available at: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning#:~:text=Q%2Dlearning%20is%20a%20reinforcement,that%20environment%20is%20not%20known

[12] Turing. *How are Neural Networks Used in Deep Q-Learning?* Available at: https://www.turing.com/kb/how-are-neural-networks-used-in-deep-q-learning#why-deep-q-learning

[13] Hui, B. *Explaining Policy Gradient Methods in Reinforcement Learning: Part 1 - Reinforce Algorithm.* Medium. Available at: https://bechirtr97.medium.com/explaining-policy-gradient-methods-in-reinforcement-learning-part-1-reinforce-algorithm-1f5f10928ce0

[14] Towards Data Science. *Policy Gradients in a Nutshell.* Available at: https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d

[15] AzoAI. *Policy Gradient Methods in Reinforcement Learning.* Available at: https://www.azoai.com/article/Policy-Gradient-Methods-in-Reinforcement-Learning.aspx

[16] DeepMind. *AlphaStar: Grandmaster-Level in StarCraft II Using Multi-Agent Reinforcement Learning.* Available at: https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

[17] Analytics Vidhya. *How to Train MS Pac-Man with Reinforcement Learning.* Medium. Available at: https://medium.com/analytics-vidhya/how-to-train-ms-pacman-with-reinforcement-learning-dea714a2365e

[18] DeepChecks. *Reinforcement Learning Applications: From Gaming to Real World*. Available at: https://deepchecks.com/reinforcement-learning-applications-from-gaming-to-real-world/

[19] Hui, J. *AlphaGo: How It Works Technically*. Medium. Available at: https://jonathan-hui.medium.com/alphago-how-it-works-technically-26ddcc085319

[20] Winder. *A Comparison of Reinforcement Learning Frameworks: Dopamine, RLlib, Keras-RL, Coach, TRFL, Tensorforce, Coach, and More*. Available at: https://winder.ai/a-comparison-of-reinforcement-learning-frameworks-dopamine-rllib-keras-rl-coach-trfl-tensorforce-coach-and-more/

[21] Savy, L. *Challenges in Deep Reinforcement Learning*. Medium. Available at: https://medium.com/@lotussavy/challenges-in-deep-reinforcement-learning-46ec2eaab3c2

[22] Bowyer, C. M. *Characteristics of Rewards in Reinforcement Learning*. Medium. Available at: https://medium.com/@CalebMBowyer/characteristics-of-rewards-in-reinforcement-learning-f5722079aef5

[23] Gibberblot. *Reward Shaping in Single-Agent Reinforcement Learning*. Available at: https://gibberblot.github.io/rl-notes/single-agent/reward-shaping.html