Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative rewards. The agent explores different actions, learns from the feedback (rewards or punishments), and improves its strategy over time to achieve the best possible outcome.

Question: Why use reinforcement learning instead of traditional search algorithm?

Reinforcement learning inherently balances between exploring new actions to find more rewarding strategies and exploiting known actions that yield high rewards. Traditional search algorithms generally don't handle this exploration/exploitation trade-off, they usually follow a predetermined path.

In addtion, RL algorithms learn from the consequences of their actions (estimating a value/policy function that serves as a kind of heuristic function), making them ideal for scenarios where it's impractical to model the environment completely before making decisions. Traditional search methods often require a complete and accurate model of the environment.

Exploration

- **Definition**: Exploration refers to the act of trying out different actions to discover their potential rewards. The goal is to gather information about the environment to make better decisions in the future.
- **Purpose**: It helps the agent learn about the rewards associated with various actions and states, especially those that have not been frequently tried.
- **Examples**:
  - Trying a new route in a maze.
  - Playing an unfamiliar move in a game.

Exploitation

- **Definition**: Exploitation refers to the act of choosing actions that are known to yield the highest rewards based on the current knowledge. The goal is to maximize the immediate reward based on what has already been learned.
- **Purpose**: It allows the agent to make the most of its current knowledge to achieve the highest possible reward at a given time.
- **Examples**:
  - Repeating a known profitable strategy in a game.
  - Choosing a familiar and successful route in a maze.

The Exploration-Exploitation Trade-off

The core challenge in reinforcement learning is balancing exploration and exploitation. An agent must explore enough to discover the best actions but must also exploit its current knowledge to maximize rewards. Here are some strategies to manage this trade-off:

On-Policy Learning

On-Policy means that the agent updates the Q-values (or the value function) using the same policy that it uses to select actions. In other words, the policy used for exploration and the policy being improved are the same.

- **Example**: SARSA (State-Action-Reward-State-Action)
  - The agent follows a policy (e.g., epsilon-greedy or softmax) to select actions.
  - The Q-values are updated based on the actions chosen by this same policy.
  - The update rule in SARSA depends on the action a' that the current policy would take in the next state s'.

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

Off-Policy Learning

Off-Policy means that the agent updates the Q-values (or the value function) using a policy different from the one it uses to select actions. The agent can use one policy to explore (behavior policy) and another policy to update the Q-values (target policy).

- **Example**: Q-Learning
  - The agent might use an exploratory policy (e.g., epsilon-greedy) to select actions.
  - The Q-values are updated using the action that maximizes the Q-value of the next state, regardless of the policy used to select the current action.

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

Clarified Explanation

1. **On-Policy Learning**:
   - The agent updates the Q-values using the same policy it follows to choose actions.
   - The updates reflect the expected return following the current policy.
   - Example: In SARSA, if the agent uses a softmax policy to select actions, it updates the Q-values based on the action that the softmax policy would take in the next state.
2. **Off-Policy Learning**:
   - The agent updates the Q-values using a potentially different policy than the one used to choose actions.
   - The updates aim to learn the optimal policy, independent of the current exploration policy.
   - Example: In Q-Learning, the agent might use an epsilon-greedy policy to explore, but it updates the Q-values based on the action that would maximize the Q-value in the next state, which is the action that would be chosen by the optimal policy.

Summary

- **On-Policy**: The same policy is used for both action selection and updating the Q-values.
- **Off-Policy**: Different policies can be used for action selection (exploration) and updating the Q-values (learning the optimal policy).

The discount factor, denoted as γ, is a key parameter in reinforcement learning algorithms that determines the importance of future rewards. It is a number between 0 and 1.

Purpose of the Discount Factor

1. **Future Reward Valuation**: The discount factor controls how much the agent values future rewards compared to immediate rewards. A higher discount factor means future rewards are considered more important, while a lower discount factor places more emphasis on immediate rewards.
2. **Trade-off Between Short-term and Long-term Rewards**: The discount factor helps balance the trade-off between short-term and long-term gains. A discount factor close to 1 encourages the agent to plan for the long term, while a discount factor close to 0 makes the agent focus on immediate rewards.

Choosing the right discount factor (γ) is crucial for the success of a reinforcement learning algorithm, as it directly affects how the agent values future rewards. Here are some guidelines and considerations for choosing the appropriate discount factor:

Guidelines for Choosing the Discount Factor

1. **Nature of the Problem**:
   - **Short-Term vs. Long-Term Goals**:
     - If the problem requires the agent to focus on immediate rewards and short-term goals, a lower discount factor (γ) is appropriate.
     - If the problem involves long-term planning and the future rewards are important, a higher discount factor (γ) is suitable.
2. **Environment Dynamics**:
   - **Stable vs. Unstable Environments**:
     - In stable environments where the future is predictable, a higher discount factor can be beneficial.
     - In unstable or highly dynamic environments where future states are uncertain, a lower discount factor might be safer.
3. **Exploration vs. Exploitation**:
   - A higher discount factor encourages the agent to explore more because future rewards are valued more.
   - A lower discount factor makes the agent exploit immediate rewards more aggressively.
4. **Experimentation and Tuning**:
   - Often, the optimal discount factor is found through experimentation and hyperparameter tuning.
   - Cross-validation or running multiple trials with different discount factors can help identify the most effective value.
5. **Specific Recommendations**:
   - Common values for γ range between 0.9 and 0.99 for long-term planning tasks.
   - For tasks where immediate rewards are more important, values between 0.5 and 0.9 might be used.
   - Very short-term tasks might use even lower values, such as 0.1 or 0.2.

Consider a grid world where an agent has two choices:

1. **Immediate Reward**: Move to a state that gives a reward of 1 immediately.
2. **Future Reward**: Move through a series of states that eventually lead to a reward of 10 after several steps.

High Discount Factor ($\gamma = 0.9$):

- The future reward is not discounted much.
- The agent values the future reward almost as much as the immediate reward.
- It will likely choose the path leading to the reward of 10.

Low Discount Factor ($\gamma = 0.1$):

- The future reward is heavily discounted.
- The agent values the immediate reward much more than the future reward.
- It will likely choose the immediate reward of 1, ignoring the better long-term reward of 10.

Mathematical Insight

If γ is low, say $\gamma = 0.1$, the total discounted reward R is computed as:

$$R = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots$$

For future rewards, the discounting effect is substantial:

- $\gamma^1 = 0.1$
- $\gamma^2 = 0.01$
- $\gamma^3 = 0.001$

Thus, a reward of 10 received after 3 steps would be valued as:

$$10 \times 0.1^3 = 10 \times 0.001 = 0.01$$

This is much less significant than a reward of 1 received immediately.

Action Selection Policies

It dictates how an agent chooses actions based on the state it is in. The goal is to balance exploration (trying new things) with exploitation (using known information to achieve higher rewards).

1. Greedy - The agent always chooses the action with the highest estimated reward. This method is purely exploitative and does not explore, which can lead to suboptimal learning because the agent might get stuck in local optima.

2. ε-Greedy

- Exploitation: With probability 1 - ε, the agent chooses the action that has the highest estimated reward based on current knowledge.
- Exploration: With probability ε, the agent randomly selects any action. This random action selection allows the agent to explore new strategies.
- Over time, ε decaies meaning the agent explores less as it becomes more confident in its knowledge.

3. Softmax - This method provides a more sophisticated way to handle the trade-off between exploration and exploitation

- Actions are selected according to the probabilities proportional to the exponential of their estimated values.
- The probability of selecting action a in state s is given by: $P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_b e^{Q(s,b)/\tau}}$
- τ is a temperature parameter that controls the level of exploration. High values lead to more exploration, low values make the behavior closer to the greedy strategy.

Other method:

Simple estimation methods, such as averaging, can indeed be used as part of action selection strategies in reinforcement learning. These methods involve maintaining and updating estimates of the expected rewards for each action based on the observed rewards. Here are a few ways these simple estimation techniques can be implemented:

1. Sample Average Method

- **Description**: This method uses the sample mean of the observed rewards for each action to estimate its value.
- **Formula**:

$$Q(a) = \frac{1}{N(a)} \sum_{i=1}^{N(a)} r_i$$

where $N(a)$ is the number of times action a has been selected, and r_i are the rewards received from selecting action a.

2. Incremental Implementation

- **Description**: Instead of maintaining a list of all rewards and calculating the average, an incremental approach updates the estimate after each action.
- **Formula**:

$$Q_{n+1}(a) = Q_n(a) + \frac{1}{N(a)}(r_n - Q_n(a))$$

where $Q_n(a)$ is the estimated value of action a after n observations, and r_n is the reward received after the n-th selection of action a.

A Q-table is a data structure used in Q-learning, a type of reinforcement learning. It stores the estimated values (Q-values) of taking a particular action in a given state. Each entry in the Q-table corresponds to a state-action pair and represents the expected cumulative reward for that pair. The agent uses the Q-table to select the best actions based on the highest Q-values, aiming to maximize its total rewards over time.

Conclusion

After experimenting with different policies for each update rules, we can say that softmax tends to offer better performance than ε-Greedy.

Higher ε / τ -> higher instability.

Performance of Q-learning and SARSA are similar with softmax as the action selection policy.

Performance of Q-learning is better than SARSA with ε-Greedy as SARSA seems more sensitive to the ε.

Softmax tends to have better results than ε-Greedy in general due to the action selection mechanisum:

- The problem with ε-Greedy is that, when it chooses the random actions, it considers all actions equally good (uniform distribution), even though certain actions are worse than others (eg. illegal pick-up or drop-off).
- Softmax on the other hand selects the random actions with probabilities proportional to their current values, hence prioritise actions that can lead to better value.