Comparative Analysis of Q-learning and SARSA in the Flappy Bird Game with Enhanced Reward Functions

Phan Duy Kha, Nguyen Dinh Huy

1 Introduction

This report analyzes the performance of Q-learning and SARSA in training an agent for the Flappy Bird game. An enhanced reward function is utilized to encourage prolonged flight, reward bonus actions, and penalize collisions. The structure of the report includes a description of the game mechanics, methodology, experimental evaluation, and a discussion on results and future improvements.

2 Game Description and Mechanics

The Flappy Bird game offers a challenging environment for Reinforcement Learning experiments. It is implemented using `pygame` and comprises the following key components:

2.1 Game Elements and Dynamics

- **Visual Elements:** The game features a background, moving ground, pipes as obstacles, and a bird controlled by the agent. A bonus element ("gift") is also included.
- **Dynamics:**
  - The bird is affected by gravity and can flap to gain altitude.
  - Pipes move leftwards at a constant speed; new pipes are generated as older ones exit the screen.
  - The gift moves along with the pipes; when collected, it provides an extra reward.

2.2 State Representation and Actions

- **State Representation:** The game state is represented as a tuple that includes the relative positions of obstacles and bonus elements.
- **Actions:**
  - **Action 0 (Do Nothing):** The bird descends due to gravity.
  - **Action 1 (Flap):** The bird flaps to gain altitude.

2.3 Enhanced Reward Function

- A positive reward is given for successfully passing a pipe.
- A bonus reward is provided for collecting a gift, while missing the gift incurs a minor penalty.
- A significant penalty is applied upon collision or when the bird goes out of bounds.

3 Methodology

This section describes the overall approach including environment setup, agent design, and the learning algorithms.

3.1 Environment and Agent Architecture

- **Environment:** The game is simulated using `pygame`, which manages state updates, action effects, and reward assignments.
- **Agent:** The agent uses a Q-table implemented as a dictionary, where the keys are state tuples and the values are arrays of Q-values for each action. States are added dynamically as they are encountered.
- **Action Selection:** A greedy or ϵ-greedy strategy is used, with ϵ gradually decayed to balance exploration and exploitation.

3.2 Learning Algorithms

Both algorithms share the same framework for state discretization and Q-table management, differing mainly in their update rules:

3.2.1 Q-learning

- **Principle:** An off-policy method that updates the Q-value using the maximum estimated future reward from the next state:

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

- **Behavior:** This approach is optimistic, leading to faster convergence but potentially riskier strategies.

3.2.2 SARSA

- **Principle:** An on-policy method that updates the Q-value based on the action actually taken in the next state:

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

- **Behavior:** By updating based on the actual action, SARSA tends to produce a more conservative policy that reduces collision risks.

3.3 Implementation Details

- **State Discretization:** Continuous states are discretized by combining the positions of pipes and gifts into a tuple, ensuring the Q-table remains manageable.
- **Training Process:** The training proceeds through multiple episodes, with the exploration rate ϵ decaying gradually from high to low to shift from exploration to exploitation.
- **Reward Structure:** As detailed above, the reward function provides specific numerical feedback to guide the learning process: **+200** for passing a pipe, **+500** for collecting a gift, **-50** for missing a gift, **-1000** for collision or going out of bounds.

4 Experimental Evaluation

4.1 Setup

Key parameters for the training process include: Learning rate (α) 0.1; Discount factor (γ) 0.9; Exploration rate (ϵ) initially high, decaying over time; Number of training episodes 1000

4.2 Performance Metrics

Performance is evaluated using:

- **Average Score per Episode:** Reflects overall agent performance.
- **Convergence Speed:** Indicated by the stabilization of Q-values.
- **Policy Stability:** Measured by the consistency of decisions during testing.
- **Bonus Collection Frequency:** Frequency of collecting bonus rewards from gifts.

4.3 Results and Discussion

Although detailed results (charts and tables) are not included here, preliminary observations indicate:

- Q-learning converges faster but may adopt riskier strategies.
- SARSA demonstrates a more stable learning process with fewer collisions due to its on-policy updates.
- The enhanced reward function is crucial for guiding the agent toward an effective policy.

5 Discussion and Conclusion

5.1 Discussion

Comparing the two algorithms:

- **Q-learning:** Converges quickly by optimistically estimating future rewards, but may lead to risky actions.
- **SARSA:** Provides a more conservative and stable policy by updating based on the actual actions taken.
- The enhanced reward function delivers effective multi-dimensional feedback that is essential for successful learning.

5.2 Limitations and Future Work

Despite promising results, several limitations remain:

- The discretization of continuous states may not capture all dynamics of the game.
- Further parameter tuning could improve overall performance.
- Future research may explore more complex reward structures and advanced state representation techniques.

5.3 Conclusion

The report demonstrates that both Q-learning and SARSA can effectively train an agent in the Flappy Bird game with an enhanced reward scheme. Each algorithm has its strengths: Q-learning offers faster convergence while SARSA ensures safer, more stable policies. The analysis lays a strong foundation for further improvements and deeper research into reinforcement learning strategies.