Overview

Original Code:

The agent sometimes loops around itself or collides with itself, especially at higher levels.

The state representation and reward function may not fully guide the agent toward optimal behavior.

The neural network architecture might be too simple to capture complex patterns.

Hyperparameters may not be ideal for the complexity of the environment.

Optimized Code:

Enhanced reward function to encourage efficient movement toward the food and discourage self-collisions.

Expanded state representation to provide the agent with more information about the environment.

Modified neural network architecture with additional layers and different activation functions.

Adjusted hyperparameters (learning rate, batch size, epsilon decay rate) for better learning.

Implemented a target network for stability using Double DQN.

I'll compare the two codes by focusing on the key components:

Reward Function

State Representation

Neural Network Architecture

Hyperparameters

Advanced Techniques (Double DQN)

Training Loop and Other Adjustments

1. Reward Function

Original Code

Collision Penalty: -10 for any collision (wall, self, obstacle).

Food Reward: +10 for eating food.

Else: 0 for other moves.

Issues:

The agent doesn't receive feedback on its movements unless it eats food or collides.

This lack of intermediate rewards may lead the agent to develop suboptimal behaviors, like looping or inefficient paths.

Optimized Code

Increased Collision Penalty: The penalty for collisions is increased to -20 to emphasize avoiding collisions.

Approaching Food: Agent receives a +1 reward when moving closer to the food.

Moving Away from Food: Agent receives a -1 penalty when moving away from the food.

2. State Representation

Original Code

The state consisted of 11 inputs:

Danger straight, right, and left (3 inputs).

Current direction (4 inputs).

Food location relative to the head (4 inputs): whether the food is left, right, up, or down from the snake's head.

Limitations:

Lacks information about distances to obstacles or the snake's tail.

Doesn't provide the agent with sufficient context to make optimal decisions in complex environments.

Optimized Code

Enhancements:

Expanded the state representation to 12 inputs, including:

Danger indicators (same as before).

Move direction (same as before).

Food relative position: Now includes normalized distances in both x and y directions.

Placeholders for Tail and Obstacle Proximity: Added slots for future implementation.

Bias Term: An optional input set to 1 to aid the neural network.

3. Neural Network Architecture

Original Code

The neural network had a simple architecture:

Input layer matching the size of the state.

One hidden layer with 256 neurons.

Output layer with 3 neurons (representing possible actions).

Activation function: ReLU.

Limitations:

May not have sufficient capacity to model the complexity of the environment.

ReLU activation can suffer from the "dying ReLU" problem.

Optimized Code

Enhancements:

Architecture Modifications:

Increased the number of hidden layers to three.

Each hidden layer has 256 neurons.

Added dropout layers to prevent overfitting.

Changed activation functions to LeakyReLU.

The dying ReLU problem is a common issue encountered when using the Rectified Linear Unit (ReLU) activation function in neural networks. It refers to the phenomenon where neurons become inactive and only output zero for any input, effectively "dying" during training. This means that once a neuron enters this state, it can no longer contribute to learning, which can degrade the performance of the neural network.

4. Hyperparameters

Original Code

Issues:

Learning rate might be too high for the complexity of the environment.

Batch size may not be optimal for stable learning.

Epsilon decay strategy may not balance exploration and exploitation effectively.

Optimized Code

Impact:

Learning Rate: A lower learning rate helps prevent overshooting minima and allows finer adjustments to network weights.

Batch Size: A larger batch size reduces variance in updates, leading to more stable training.

Epsilon Decay: Exponential decay allows for a smoother transition from exploration to exploitation, ensuring the agent continues to explore new strategies over time.

5. Advanced Techniques (Double DQN)

Original Code

Used standard DQN without a target network.

Issues:

Overestimation Bias: Standard DQN can overestimate action values, leading to suboptimal policies.

Learning Instability: Without a target network, the model's predictions can become unstable.

Optimized Code

Implementing Double DQN, During training, the target network is used to compute the target Q-values for the next states, while the main network is updated.

Yaret nshlha

6. Training Loop and Other Adjustments

Original Code

Trained for a set number of games (e.g., up to 1000 games).

Limited handling of device assignments, which could lead to mismatches between CPU and GPU tensors.

Optimized Code

Adjustments:

Increased Training Duration: The training loop now runs for up to 5000 games to allow the agent to learn from more experiences.

Device Management:

Ensured all tensors and models are consistently assigned to the correct device (CPU or GPU).

Impact:

Extended Training: Allows the agent more opportunities to refine its policy.

Device Consistency: Prevents runtime errors due to device mismatches and ensures efficient computation.

Monitoring: Enables the observation of improvements over time, aiding in debugging and parameter tuning.

Mlhash lzma