Neural Network and Q-Learning Implementation for Flappy Bird Game

Drumia Sebastian, Solomon Claudia

12.01.2025

1. Neural Network Architecture

The neural network serves as a **function approximator** to estimate the Q-values for the agent's actions. Below is a detailed breakdown:

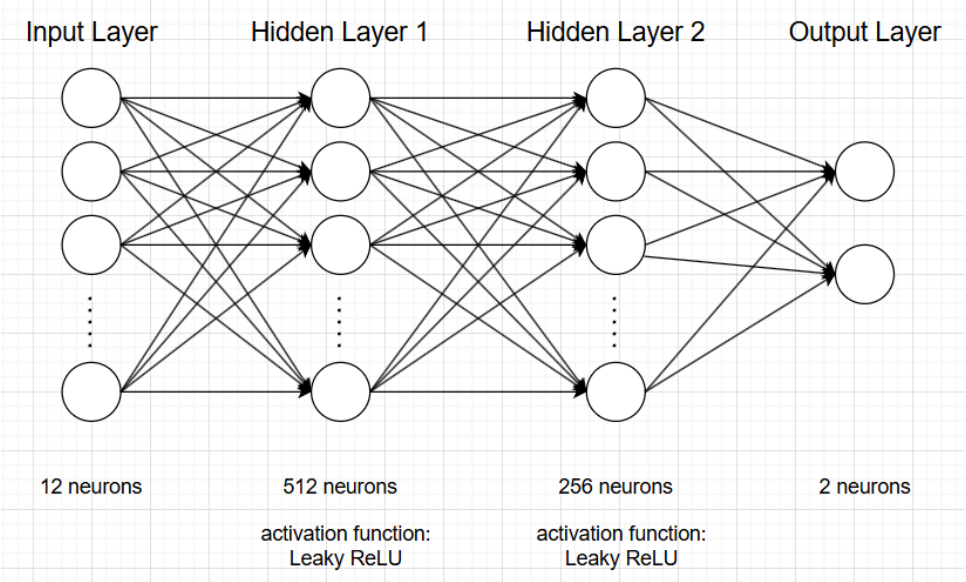

Figure 1: Neural network architecture with 12 input parameters, two hidden layers (512 and 256 neurons), and an output layer with two actions. The activation function for the hidden layers is Leaky ReLU

1.1 Structure

- **Input Layer:**
  - The input consists of **12 parameters**, each describing a specific aspect of the game state:
    (1) Horizontal position of the last pipe.
    (2) Vertical position of the last top pipe.
    (3) Vertical position of the last bottom pipe.
    (4) Horizontal position of the next pipe.
    (5) Vertical position of the next top pipe.
    (6) Vertical position of the next bottom pipe.

(7) Horizontal position of the next-next pipe.
(8) Vertical position of the next-next top pipe.
(9) Vertical position of the next-next bottom pipe.
(10) Player's vertical position.
(11) Player's vertical velocity.
(12) Player's rotation.

- **Hidden Layers:**
  - The network has two hidden layers:
    - The **first hidden layer** contains **512 neurons**.
    - The **second hidden layer** has **256 neurons**.
- **Output Layer:**
  - The network outputs Q-values for each of the two possible actions:
    - **0:** Do nothing.
    - **1:** Flap.

1.2 Activation Function

- **Leaky ReLU** ($\alpha = 0.01$) is used in the hidden layers:
  - Allows a small gradient for negative values, avoiding "dead neurons" (neurons stuck at zero).
  - Enhances learning stability, especially for deeper networks.

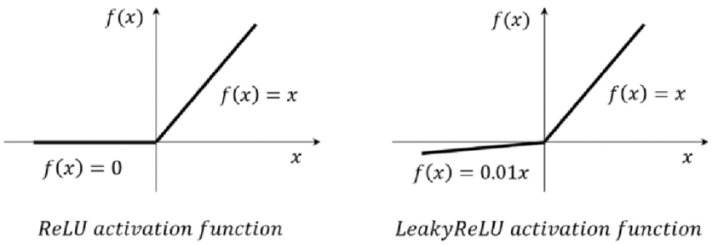

Figure 2: Graph comparison between normal ReLU and Leaky ReLU

- No activation is applied to the **output layer** since it represents raw Q-values used for decision-making.

1.3 Training Details

- **Loss Function:** Mean Squared Error (MSE) is used to compare predicted Q-values with target Q-values.
- **Optimizer:** Adam optimizer with a learning rate of 0.0001.

2. Q-Learning Algorithm Implementation

Q-learning is a **model-free reinforcement learning algorithm** used to train the agent by estimating the Q-values for state-action pairs.

2.1 Epsilon-Greedy Exploration

- The algorithm balances **exploration** and **exploitation** using the epsilon-greedy strategy:
  - **Exploration:** Randomly choose an action to discover new strategies.
  - **Exploitation:** Use the action with the highest predicted Q-value from the policy network.
- Epsilon (ϵ) starts at **1.0** and decays over time (**decay rate**) until it reaches a minimum threshold of a chosen **minimum epsilon**.

2.2 Replay Buffer

- A **replay buffer** with a maximum size of **100 000 transitions** is used to store state transitions (state, action, reward, next state, done).
- During training, **batches of 32 samples** are randomly drawn from the buffer to break the temporal correlation between consecutive states, improving generalization.

2.3 Dual Neural Networks

Two separate networks are used to stabilize training:

- **Policy Network:**
  - Used to select actions during gameplay and training.
  - Updated after every batch using gradient descent.
- **Target Network:**
  - Provides stable Q-value targets for training.
  - Synchronized with the policy network (copies its weights and biases) every **10 steps** to prevent instability caused by rapidly changing Q-value targets.

2.4 Update Rule

1. Compute the **target Q-value**:

$$Q_{\text{target}} = \begin{cases} \text{reward}, & \text{if terminal state} \\ \text{reward} + \gamma \cdot \max(Q_{\text{target\_net}}(\text{next state})), & \text{otherwise} \end{cases}$$

   Where $\gamma = 0.99$ is the discount factor.

2. Compute the **current Q-value**:

$$Q_{\text{current}} = Q_{\text{policy}}(\text{state, action})$$

3. Update the policy network by minimizing the loss:

$$\text{Loss} = \text{MSE}(Q_{\text{current}}, Q_{\text{target}})$$

3. Environment Details

The environment is implemented using **Flappy Bird for Gymnasium**, with the **parameterized state representation** (12 features described earlier).

3.1 Action Space

- The agent has **two possible actions**:
  (1) **0:** Do nothing.
  (2) **1:** Flap (jump).

3.2 Reward System

- The agent receives rewards based on its actions and outcomes:
  - **+0.1:** For every frame it survives.
  - **+1.0:** For successfully passing a pipe.
  - **-1.0:** For dying.
  - **-0.5:** For hitting the top of the screen.

4. Training Process and Hyperparameters

These hyperparameters remain the same throughout the first 3 experiments:

- **Batch Size:** 32 transitions per training step
- **Replay Buffer Size:** 100 000 transitions
- **Learning Rate:** 0.0001
- **Discount Factor (γ):** 0.99
- **Target Network Update Frequency:** Every 10 steps

4.1 Experiment 1

Training Parameters:

- **Epsilon Decay Rate:** 0.999995
- **Minimum Epsilon:** 0.1

Experimental results:

- **Best reward:** 25.2
- **Time passed until best reward reached:** 3h 0min 20 sec
- **Numbers of episodes until best reward reached:** 502 398
- **Total time:** 13h
- **Max number of episodes:** 1 000 000

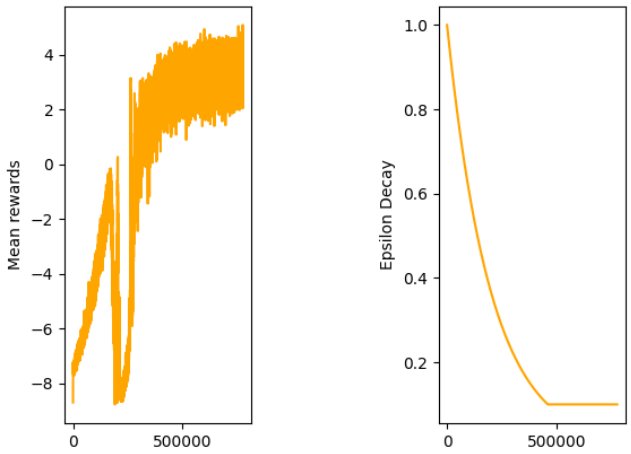

Figure 3: Experiment 1: Epsilon decay and Mean rewards graphs

4.2 Experiment 2

Training Parameters:

- **Epsilon Decay Rate:** 0.999995
- **Minimum Epsilon:** 0.2

Experimental results:

- **Best reward:** 10.9
- **Time passed until best reward reached:** 1h 29min
- **Numbers of episodes until best reward reached:** 318 901
- **Total time:** 4h 32min
- **Max number of episodes:** 587 523

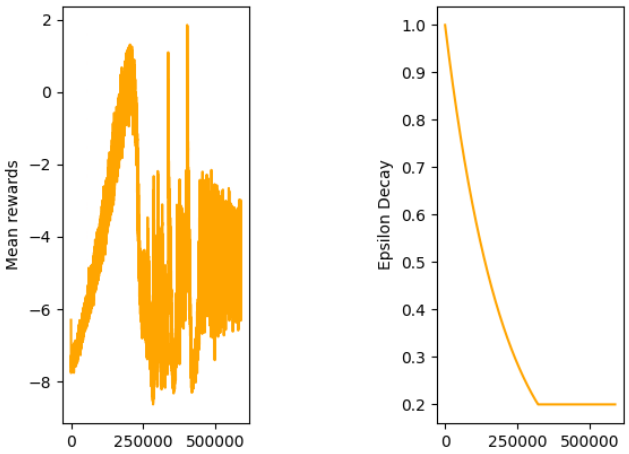

Figure 4: Experiment 2: Epsilon decay and Mean rewards graphs

4.3 Experiment 3

For this experiment, we modified the architecture of the neural networks, as following: instead of 2 hidden layers, we only used one with 512 neurons and the simple ReLU activation function.

Training Parameters:

- **Epsilon Decay Rate:** 0.999995
- **Minimum Epsilon:** 0.2

Experimental results:

- **Best reward:** 9.4
- **Time passed until best reward reached:** 36min
- **Numbers of episodes until best reward reached:** 233 158
- **Total time:** 6h 54min
- **Max number of episodes:** 765 364

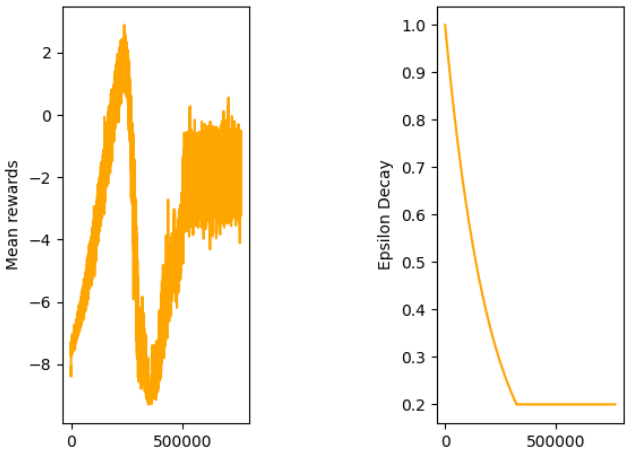

Figure 5: Experiment 3: Epsilon decay and Mean rewards graphs

4.4 Experiment 4

Unlike the first 3 experiments where we used the 12 parameters mentioned above, the forth one uses the lidar flag of the environment set to *True* (180 observations).
Training Parameters:

- **Epsilon Decay Rate:** 0.9999995
- **Minimum Epsilon:** 0.2

Experimental results:

- **Best reward:** 4.4
- **Time passed until best reward reached:** 6h 58min
- **Numbers of episodes until best reward reached:** 774 565
- **Total time:** 7h 48min
- **Max number of episodes:** 781 525

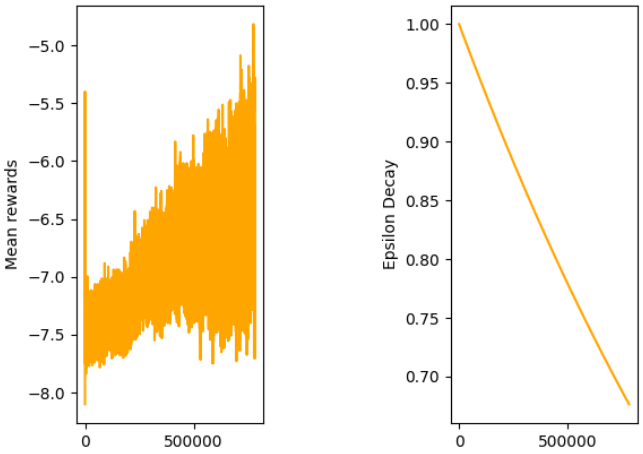

Figure 6: Experiment 4: Epsilon decay and Mean rewards graphs