Högskolan i Gävle

Enhancing Snake AI with Deep Q-Learning

Efficient Decision-Making Strategies for Autonomous Apple Collection

By Daniel Danho & William Eriksson

daniel.danho.dd@gmail.com
william99eriksson@outlook.com

Date 2024-01-08

Course: Introduction to Machine Learning HT2023

Instructors: Mohammad Aslani, Hanna Holmgren & Åke Wallin

Abstract

This project explores into implementing Deep Q-Learning within the Snake game, drawing inspiration from the foundational research “Human-level control though deep reinforcement learning” Focused on enhancing the Snake game’s intelligence, the study investigates the potential of Deep Q-learning in training an agent to navigate and optimize gameplay strategies. Employing PyTourch, the project constructs a learning model and overcomes challenges by incorporating knowledge from various sources. Though iterative training sessions, the game agent demonstrates improvements, showcasing the adaptability and efficacy of Deep Q-Learning in decision-making processes within the forced dynamics of the Snake game environment.

Table of Contents

1 Introduction

The application of Deep Q-Learning in gaming environments, showcases the adaptability and effectiveness of reinforcement learning algorithms in mastering complex tasks within constrained environments.

The foundation of Deep Q-Learning, derived from the integration of Q-Learning with deep neural networks, presents a paradigm shift in how AI agents learn to navigate and excel in gaming scenarios. Much like how the k-means algorithm revolutionized clustering, Deep Q-Learning seeks to optimize decision-making processes within the dynamics of gaming environments.

1.1 Background

Deep Q-Learning is a specific algorithm within Deep Reinforcement Learning (DRL). It extends the classic Q-Learning algorithm by using a deep neural network to approximate the Q-value function. The Q-value function essentially tells the agent what the expected return is for taking an action in each state. Deep Q-Learning has been successfully used in various applications, and famously for playing video games at a high level, like the Atari games, [1]

1.2 Motivation

Our project influences the principles of Deep Q-Learning to train an AI agent to play the Snake game effectively. The theoretical foundations of the Deep Q-Learning algorithm are explained, highlighting its relevance and application in the context of gaming environments.

Moreover, the project entails the implementation of Deep Q-Learning algorithms using frameworks such as TensorFlow or PyTorch within the Snake game environment. This practical implementation aims to demonstrate the system's functionality, showcasing the AI agent's ability to learn, adapt, and optimize its gameplay strategy through interactions with the game environment.

In addition to showcasing the general functionality of the AI agent trained through Deep Q-Learning, the project aims to conduct systematic validation exercises. These validations may involve analyzing metrics such as the agent's average score, its ability to achieve specific objectives (e.g., collecting apples efficiently), and its strength in handling variations within the Snake game environment.

Through this project, we seek to not only explore the theoretical foundations of Deep Q-Learning but also demonstrate its practical application in mastering the Snake game. By employing methods for training and validation, we aim to showcase the potential of Deep Q-Learning in enhancing the AI agent's decision-making and control abilities within gaming environments.

2 Methods

The snake game was implemented in *Python* using the *Visual Studio Code* editor. First the game was created and tested by the group, at this stage it was possible to be play the game using the arrows keys on the keyboard.

2.1 The Model

After this, the model was developed using PyTorch. The model compromises two Python classes: the *Linear QNet* class and the *QTrainer* class.In the *Linear_QNet* class, we define the activation function, which is ReLU, figure 1. This class also constructs the neural network using nn.Module from PyTorch. Additionally, it includes a *save* function, which is used to make the model persistent in training.

The other class in the model, *QTrainer*, is responsible for implementing the hyperparameters. It uses Adam as optimizer and Mean Squared Error (MSE) as the loss criterion. The *train_step* function within this class calculates the Q-value for the model. It does this by considering the current state, potential state following an action, the reward, the gamma value (the exploration) and the action itself.

```
class Linear_QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

Figure 1, The start of the Linear_QNet class

2.2 The Agent

In the agent class we determine the learning rate, batch size and max memory. The agent class imports both the game and the model and uses the classes in the model to train it. In this class we use several functions for the agent like getting the state representation from the game including dangers (collisions), current direction of the snake and the relative position of the food, this is used to decide on what action to perform next. The memory function is used to store the experiences which is then used for training.

The main function in this class is the train function which this function runs the training loop for the agent. It continuously plays the Snake game, making decisions using the Agent class. In each loop iteration, the agent gets the current state, decides on an action, performs the action in the game, and receives the new state and reward. It updates the agent's short-term memory and records the experience in long-term memory. After each game ends (done is True), it trains the long-term memory and updates the game record if a new high score is achieved, figure 2. The model is saved when a new record is set. The scores and mean scores are plotted after each game, providing a visual representation of the agent's performance over time.

```python
def train():
    plot_scores = []
    plot_mean_score = []
    total_score = 0
    record = 0
    agent = Agent()
    game = SnakeGame()
    while True:
        #Get old state
        state_old = agent.get_state(game)

        #Get move
        final_move = agent.get_action(state_old)

        #Perform move
        reward, done, score = game.play_step(final_move)
        state_new = agent.get_state(game)

        #Train short memory
        agent.train_short_memory(state_old, final_move, reward, state_new, done)

        #Remember
        agent.remember(state_old, final_move, reward, state_new, done)

        if done:
            #train the long memory, plot result
            game.reset()
            agent.n_games += 1
            agent.train_long_memory()

            if score > record:
                record = score
                agent.model.save()

            print('Game: ', agent.n_games, 'Score: ', score, 'Record: ', record)

            plot_scores.append(score)
            total_score += score
            mean_score = total_score/agent.n_games
            plot_mean_score.append(mean_score)
            plot(plot_scores, plot_mean_score)
```

Figure 2, the train function

3 Results

The AI agent's performance showcased a noteworthy progression across successive game instances. In the early stage of the training process containing the first 100 games, the AI agent presented a modest performance, achieving an average score of approximately 2 points. This phase primarily reflects the early stages of learning, where the agent dealt with understanding game dynamics and formulating effective strategies, figure 3.

However, as the training progressed and the AI agent engaged with the game environment, a remarkable advancement in performance became obvious. Subsequently, around 300 games, the AI agent showcased significant improvements, achieving an average score of around 20 points. This notable rise in performance signifies the agent's learning capability, adaptability, and refinement of decision-making strategies over time, figure 3.

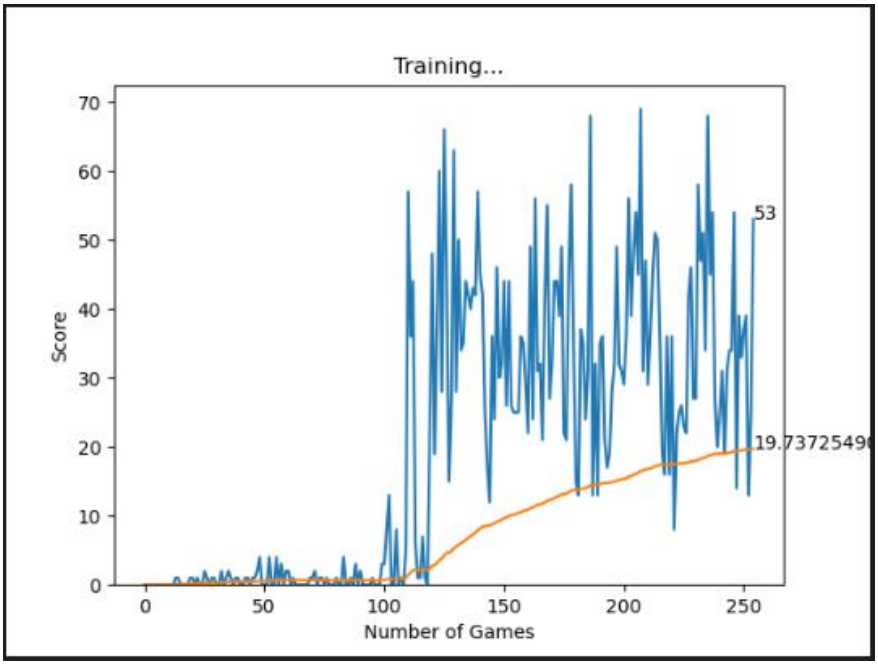

Figure 3, Results over time for the agent

This trend illustrates the AI agent's ability to learn from experiences, refine its gameplay strategies, and progressively improve its performance within the Snake game environment through Deep Q-Learning. The observed progression from modest scores to significantly higher averages exemplify the efficacy and learning capabilities of the implemented Deep Q-Learning model.

Including such empirical findings not only demonstrates the effectiveness of the Deep Q-Learning approach but also provides a clear tale of the agent's learning curve and improvement over successive game iterations.

4 Discussion

Our project initiated into implementing Deep Q-Learning in the snake game, inspired by the challenges we discovered in existing methods and our study of alternative solutions. The foundational article [1], “Human-level control though deep reinforcement learning which provided valuable insights that guided our approach. Instead of playing the same Atari games as in the paper we decided on applying it to a newer but still simple game to test.

4.1 Understanding the Problem and Discovering Solutions

We encountered challenges in typical gaming approaches. This triggered our desire to investigate further into alternative solutions. Deep Q-Learning stood out due to its ability to learn directly from visual data, aligning with principles discussed in the foundational article.

4.2 Project Journey

Using PyTorch, we constructed the game's learning framework. The process presented challenges, but our progress was influenced by the knowledge gathered from different studies and methods. These resources helped us refine our program as well as enhancing its learning capabilities.

4.3 Reflection on the learning process

Our project progression, shaped by the insights from the foundational article, highlighted the revolutionary potential of reinforcement learning in gaming environments. The gradual improvement witnessed in the game's performance underscored the adaptability and effectiveness of the Deep Q-Learning.

In summary, our project's venture into Deep Q-Learning for the Snake game revealed both challenges and successes. The iterative learning process reflected human-like adaptability, demonstrating the method's promise in enhancing gaming agent intelligence.

5 Referenser

[1] V. Mnih, K. Kavukcoglu, D. Silver, A. A. Rusu, J. Veness och e. al., ”Human-level controll through deep reinforcement learning,” Nature publishing group, 2015.