**AI Q-Deep Reinforcement Learning**

**With Yolo Image Detection**

**In a Retro Super Mario Bros environment**

**A short introductory**

Colin Peiffer | Projektkurs Bio-Informatik | Q2

19.12.2023

**Table of Contents**

**Introduction………………………………………………………………………3**

**Computer Vision Experiments with Yolo………………………………..4**

**Q Deep Reinforcement Learning……………….………………………….6**

**Setting up the Final Environment………………………………………….7**

**Backpropagation for Jumps…………………… ……………………………9**

**Putting it all together…………………………………………………………10**

**Video Demonstration………………………………………………………..11**

**Introduction**

I was inspired to create this project by Chrispresso, who created a video demonstrating artificial intelligence in Super Mario Bros. implemented using deep reinforcement learning and image recognition. Intrigued by the world of artificial intelligence, I wanted to explore and experiment with the possibilities that AI technologies offer and decided to start such work as part of my school project.

For my project, I developed a Super Mario Bros artificial intelligence and trained it using a diverse dataset containing sprites of various objects from classic retro Super Mario games. Its main function is to use image recognition to identify friendly and threatening objects. To further enhance the AI's capabilities, it was exposed to video clips from the internet, specifically YouTube, to understand the patterns of experienced players trying to complete a course in record time (it didn’t).

During training, the AI draws on data from previous training sessions to actively tackle the course. The methodology incorporates the reward system used by the Q-Deep reinforcement learning algorithm to reinforce positive behavior and improve the AI's decision-making process. This integrated approach allows the AI to not only effectively navigate the game environment, but also to better understand the optimal strategy for completing the course in the shortest possible time (I didn’t complete it fast).

**Computer Vision Experiments with YOLO**

In order to gather information from games, I considered using YOLO (You Only Look Once), an advanced Deep Learning model for object detection. YOLO is renowned for its speed and efficiency, capable of identifying and classifying various objects in an image in real-time. This technology was intended to enable our AI to recognize key elements of the game, such as enemies, obstacles, and objects.

To integrate object detection via YOLO into the game, a systematic approach was taken for training object recognition. The model requires a large and accurate training dataset to correctly interpret game elements. Initially, I split the spritesheet of Super Mario Bros using a custom Python script. Another script then took these cropped images and placed them randomly on various backgrounds.

To enhance YOLO's ability to generalize and recognize these objects in diverse game contexts, I introduced variations in these images, including changing the background and rotating the objects.

Here are some examples of images generated by the Python script:

**A screenshot of a video game

Description automatically generated**

With each image, the script generates a text file that stores the coordinates of objects relative to the image. Each line, corresponding to an object, consists of several parameters:

* The object class (a unique identifier linked to the object)
* The normalized coordinates of the object's center (x\_center, y\_center)
* Its width (width) and height (height)

This annotation method is essential for supervised learning, as it provides the model with the necessary information to learn how to locate and identify objects in images.

Here is an overview of a .txt file containing labels. In this file, you can find coordinates for a pipe (class 0), Mario (class 1), a block (class 2), and a Goomba (class 3):

A black background with white numbers

Description automatically generated

**Once this dataset was prepared, the next step was to train the YOLO model. The training process involved feeding the model with these images and annotations, enabling YOLO to gradually learn to recognize and locate various objects in the game. This training phase is crucial as it determines the accuracy with which the model will later be able to identify and react to in-game elements in real-time.**

**A screenshot of a video game

Description automatically generated**

***Example of detection in a video of the game (model undertrained)***

**Q-Deep Reinforcement Learning**

**1. Reinforcement Learning (RL):**

Reinforcement Learning is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties based on the actions it takes, and its objective is to maximize cumulative rewards over time. RL involves finding optimal strategies for decision-making through trial and error.

**2. Q-Learning:**

Q-Learning is a specific reinforcement learning algorithm used for making decisions in an environment. It employs a Q-table to store and update the quality (Q-value) of each action for different states. The algorithm iteratively refines its decision-making policy by learning from the rewards obtained during interactions with the environment. Q-Learning is particularly effective for problems with discrete state and action spaces.

**3. Deep Learning:**

Deep Learning is a subfield of machine learning that involves neural networks with multiple layers (deep neural networks). These networks can automatically learn hierarchical representations of data, enabling them to capture intricate patterns and features. Deep Learning has achieved significant success in various tasks such as image and speech recognition, natural language processing, and game playing.

**4. Q-Deep Reinforcement Learning:**

Q-Deep Reinforcement Learning combines Q-Learning with Deep Learning techniques. Instead of using a Q-table to store Q-values, a deep neural network is employed to approximate the Q-function. This allows the algorithm to handle problems with large and continuous state spaces, providing more scalability and flexibility. Q-Deep Reinforcement Learning has been successful in solving complex tasks where traditional Q-Learning may be limited, especially in scenarios with high-dimensional input data, such as images.

**Setting up the Final Environment**

In parallel with data extraction via YOLO, I explored another way to retrieve the game state. Ultimately, I decided to emulate Super Mario Bros with "nes-py," an emulator that has a Python API providing access to the console's RAM.

During the NES era, engineers programmed games with significant memory constraints. For better control, each variable was manually allocated; consequently, the memory locations of interest remain consistent. In fact, RAM maps of Super Mario Bros exist, ensuring reliable data retrieval. After some research on YouTube, I found a video addressing the same topic that directed me to a very interesting GitHub repository.

A screenshot of a video game

Description automatically generatedA screenshot of a computer

Description automatically generated

*The Github Repository*

*The Video in Question*

**Firstly, I developed a debug window to verify that the computer perceives the game correctly:**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

*Original Render*

*Vision seen by RAM*

This method of data retrieval is reliable, efficient, and easy to manipulate. It provides exceptional comfort and time savings compared to detection via YOLO, albeit with code less adaptable to other Mario games.

With the data at our disposal, the next step is to set up a training environment. The specifications for this environment are as follows:

* Detect the end of a game session
* Instantaneously reload the beginning of the level
* Know when Mario moves forward, backward, or encounters an enemy

These simple functionalities allow me to restart a level in a loop each time Mario dies or wins. Furthermore, the data on Mario enables us to associate a reward with each action. To my luck, the gym-super-mario-bros library provides an API based on nes-py that fulfills the entire set of specifications.

**Backpropagation for Jumps**

I have implemented a specific backpropagation mechanism for jumps to reinforce the AI's learning about the consequences of its actions. This process is particularly crucial for situations where Mario dies after a jump, as it allows retroactively penalizing decisions that led to this failure.

The API I developed provides two key states essential for this mechanism: "grounded" (Mario is on the ground) and "floating" (Mario is in the air, typically jumping). I use these states to determine when Mario starts a jump and when he lands. For this purpose, two attributes in my training system, ‘self.just\_hit\_ground’ and ‘self.just\_jumped’, are used to mark these events.

The backpropagation logic is divided into 3 steps:

1. Jump and Landing Detection:
   * If the previous state was "floating" and the current state is "grounded," it means Mario has just landed.
   * Conversely, if the previous state was "grounded" and the current state is "floating," it indicates that Mario has just jumped.
2. Jump Buffer Management:
   * From the moment Mario jumps (when ‘self.just\_jumped’ is activated), I start recording in a buffer all state-action combinations until he lands or dies.
   * If Mario successfully lands (‘self.just\_hit\_ground’ activated), the buffer is completely cleared as the jump has been successful.
   * If Mario dies before landing, the state-action combinations recorded in the buffer are used for backpropagation.
3. Applying Death Penalty:
   * When Mario dies while jumping, a penalty of -1 is retroactively applied to all Q values stored in the buffer. This means that for each state-action recorded since the beginning of the jump, I decrease the corresponding Q value in the Q-Table.
   * This penalty aims to discourage sequences of actions that led to failure, guiding the AI toward safer and more effective jumping strategies.

**Putting it all together**

The AI's learning process for playing the Mario course involves a combination of computer vision techniques and reinforcement learning. Initially, the AI leverages YOLO, a deep learning model, to extract information from the game screen. This includes detecting key elements such as enemies, obstacles, and objects, which is essential for the AI to understand the game environment.

To enhance its understanding, the AI simultaneously emulates the Super Mario Bros game using "nes-py," providing access to the console's RAM. This allows the AI to gather information about the game state, such as Mario's position, actions, and surroundings.

The AI's learning is facilitated through a Q-learning approach, implemented using a Q-Table structure. This table is organized as a multi-dictionary, where each key represents a specific game state, and the values are dictionaries containing actions and their corresponding Q-values. During initialization, all Q-values are set to zero, creating a blank canvas for the AI to learn through trial and error.

To guide the AI towards effective strategies, rewards and penalties are carefully tuned. The AI is trained to detect the end of a game session, reload the level instantly, and recognize Mario's movements, including forward, backward, and interactions with enemies.

A crucial aspect of the AI's learning is the backpropagation mechanism for jumps. Specific states, such as being grounded or floating, are utilized to track Mario's jump actions. A buffer system is implemented to record state-action combinations during jumps, and a penalty is retroactively applied if Mario dies during a jump. This encourages the AI to avoid actions that lead to failure, guiding it towards safer and more effective jumping strategies.

The overall learning architecture is organized into distinct classes, each responsible for specific aspects such as training, reading NES memory, and managing game states. The AI's training environment is easily configurable through a settings file.

The AI learns to play the Mario course by combining computer vision for data extraction, emulating the game for state information, and utilizing Q-learning with carefully tuned rewards and penalties. The backpropagation mechanism for jumps plays a crucial role in refining the AI's decision-making, making it more strategic and cautious in its gameplay.

**Video Demonstration**

**[](https://www.youtube.com/embed/omlwjMuns40?feature=oembed)**

The AI is comfortably able to complete the course without dying