# Applied Reinforcement Learning on Super-Mario-Bros.

A Report

Ludwig-Maximilians-Universität München
Institute of Informatics
Chair of Database Systems and Data Mining

Authors: Xingying Chen and Christian Lemke
Advisor: Dr. Michel Tokic

Date: 02.09.2018

## Content

## Introduction

Reinforcement learning is popular for training an agent to solve a task independently. An agent learns to act in an environment through trials and feedback.. In the reinforcement learning project, we trained a super mario agent with acquired knowledge basically from the theoretical part of the course. Our agent is trained with a Deep Q-Network [1] with Prioritized Replay Buffer [2] and Dueling Network[3]. Eventually, it can survive the first level of the super mario game.

In this report, we will first introduce the theoretical background for the experiment. Then, we will briefly illustrate the environment and the framework of the super mario game. Afterward, we will report the issues we encountered during the training and the corresponding solutions. Finally, we will show the result of our training and conduct short conclusion.

**Theory and Background**

**Deep Q-Learning**

Deep Q-Learning is the deep learning version of Q-Learning, where the state space has high dimensions (e.g. an image). A convolutional neural network is used to downsample the input state and outputs the Q-values for all actions. These outputs are then used for updating the Q-function:

$$Q(s,a) = E[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')].$$

The Convolutional Neural Network in [1] has three convolutional layers and two fully connected layers. There are no pooling layers because they drop spatial dependencies of the objects in the image.

The network is updated with batch stochastic gradient descent:

$$\nabla L_i(\theta) = E[(r + \gamma \max_{a'} Q(s',a';\theta') - Q(a,s,\theta))\nabla Q(a,s,\theta)],$$

where $\theta'$ is the parameters of a fixed target network and $\theta$ is the parameter of the deep Q-network.

The input of the convolutional neural network is set to be a stack of four frames. With only one frame, it is hard to recognize the moving direction of the objects.

**Improvements**

**Experience Replay**

In online reinforcement learning, agents discard the observation immediately after an update. This is a waste of resource where experience gaining is difficult. Rare experience should be reused. Another drawback is that the learned experiences have strong temporal correlation which speaks against the i.i.d assumption of many stochastic gradient-based algorithms.
Experience Replay is introduced by [4] to solve the addressed issues. Experience is stored in a buffer, making it possible to reuse experience and break the temporal correlation by uniformly sampling from experience of different time steps. Specifically, it uses a large sliding window replay memory to store the tuple (s, a, r, s’). [4]

**Prioritized Replay Buffer**

With a replay buffer, experience is sampled uniformly, which means rare experience is rarely used for updates. In fact, certain transitions may provide more information for the agent. To identify the valuable transitions, Prioritized Replay Buffer introduces a priority measure for each experience. An ideal measure would be the amount the agent can learn from the transition which is not instantly available. A quantifiable measure uses the temporal-difference (TD) error of a transition. This fits online RL algorithms which already applies TD-error in the training process. [2]

In detail, the transition with the largest absolute TD error will be chosen for replay. New transitions without a TD error will acquire the maximal priority to ensure at least one update. In practice, a binary heap is used for the priority queue. The time for getting the experience with maximal priority requires $O(1)$ and inserting a new experience requires $O(logN)$.

However, there are still some drawbacks of this greed TD-error prioritization. First, TD errors are only updated for replayed transitions. Transitions with low TD error have little chance to be updated because more often they are discarded from the sliding window memory. Also, it is sensitive to noise from the stochastic rewards and approximation errors from bootstrapping. Finally, experience with initial high TD errors is sampled and updated frequently. The system tends to over-fitting.

To interpolate between greedy prioritization and uniform random sampling, the priority measure is adjusted to:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha},$$

for all transitions i, where the exponent $\alpha$ is the prioritization rate. $\alpha$ = 0 indicates a uniform random sampling. Usually, $p_i = |\delta_i| + \epsilon$ with $\epsilon$, a small positive constant, to prevent transitions with zero error from not being visited.

- **Adjusting the bias**

To estimate expected value with a stochastic update, it is important that those updates are sampled from the same distribution as the expected. However, the prioritized replay changes the distribution. Therefore, importance-sampling (IS) weights are introduced to correct this bias:

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta.$$

N is the size of the replay buffer; the exponent $\beta$ grows linearly from $\beta_0$ to 1 at the end of the learning process. Weights are stabilized by normalization with the maximal weight $\frac{1}{max_i w_i}$. Instead of $\delta_i$, $w_i\delta_i$ is used to update the Q-learning network.

In the implementation, O(N) is not efficient enough for sampling. The cumulative density function can be approximated by a piecewise linear function with k segments of same probability. Specifically, it first samples a sagement and then samples uniformly from this segment.

**Dueling Network**

Dueling network is introduced by [3] to generalize learning across actions. For each state and action pair, it computes the advantage $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, measuring the importance of each action in the given state. For example, when there are no enemies in front of Mario, moving right or jumping right is relatively the same. [3]

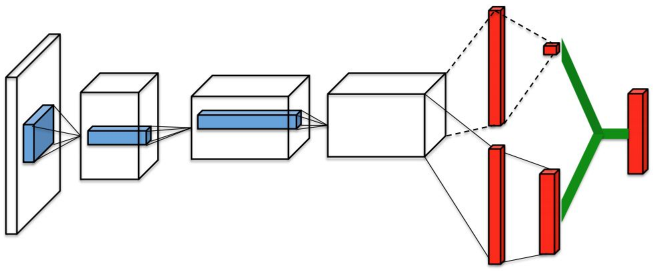

Figure 1: Two streams separately estimate the state-value V(s) function and the advantage A(s, a) for each action.[3]

On the basis of a Deep-Q-Network, a dueling network uses an additional stream to estimate the advantage of each action. Propagated through two streams of fully connected layers, the outputs of the advantage function and the value function are combined to produce the $Q$ function. The Q-function can be computed in different ways.

The first one is $Q(s,a) = V(s) + (A(s,a) - max_{a' \in A} A(s,a)')$. When the best action is chosen, the second term with brackets equals to zero and V(s) is left. The semantic of the value function is preserved.

The second one is $Q(s,a) = V(s) + (A(s,a) - \sum_{a'} A(s,a')/|A|)$. Although it loses the semantics of value function, the optimization is more stable because the advantages change only as fast as the mean. In the experiment in [], the first one performs similar to a simpler version of the second one. Therefore, the second one is adopted in practice.

Dueling Network and Prioritized Replay Buffer optimize different aspects of the training process. Therefore, it is reasonable to combine them. The experiment in [3] also confirms that the combination of Dueling Network and Prioritized Replay Buffer results in the best performance.

## The Environment of Super Mario Bros.

The environment for the reinforcement learning problem is the video game of "Super Mario Bros.". The arcade game was published 1985 by Nintendo. In the game, the player's objective is to control Mario in the side-scrolling landscape of the Mushroom Kingdom to rescue Princess Toadstool from her hijacker Bowser. During the game, Mario has to avoid hazards such as enemies and abysses. [5]

For this work, we use the OpenAI Gym compatible environment of "Gym-Super-Mario-Bros." in version 3.0.0. The game runs on the nes-py emulator, is easy to install via pip command and is accessible thru the OpenAI gym framework interface. [4]

The following sections describe the game specific setup of the reward function, as well as the observation and action space.

### Observation Space

In reinforcement learning, the observation space contains all information of the environment. The Agent has to choose its next action based on the observation and has to learn and adapt between new observations.

In Gym-Super-Mario-Bros. the observation space consists of raw pixel values in RGB color space. The amount of pixels depends on the reconfigurable rendering resolution setup. Like in the work of Mnih et al. [6], we set the resolution to 84x84 and reduce the color space from RGB to gray color space. Overall this results in 7056 discrete pixel values as observation space. Figure 1 shows an example of the observation space.

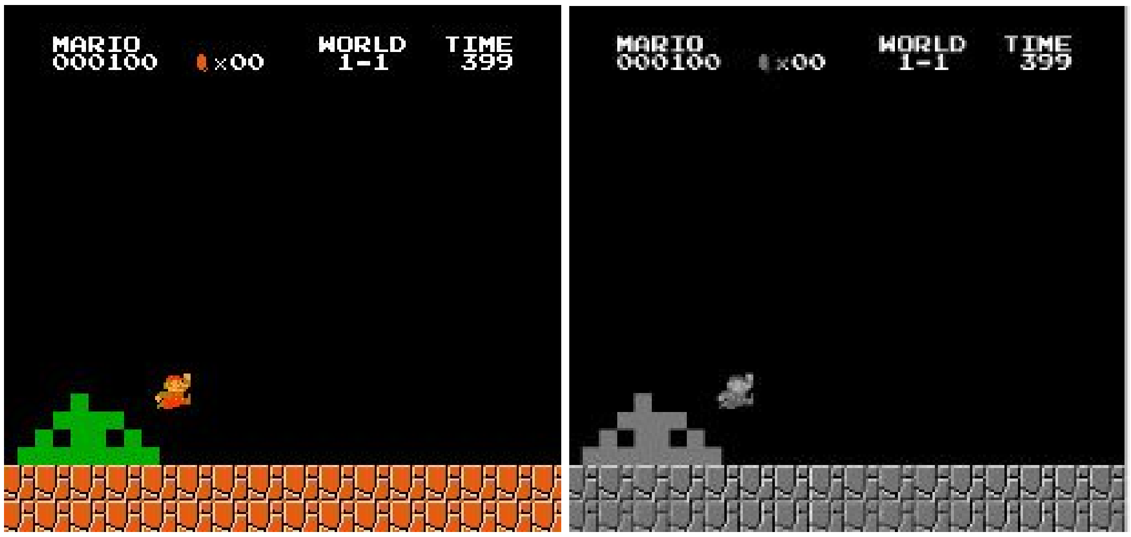

Figure 1: Two screen captures of the game with a resolution of 84x84. The left one shows the raw observation space with RGB color space. The right image shows the same observation after the color reduction to gray values. This image represents one state of the environment's observation space.

**Action Space**

In reinforcement learning, the action space defines the different actions the agent can perform to interact with the environment. The game of Super Mario Bros. is originally controlled with a control pad with four buttons for the movement direction, one button for jumping and one to fire fireballs. In this environment, we reduce the number to seven discrete actions. These actions allow the agent to move to left and right, stay in position, jump up and to the right as well as fire fireballs.

**Reward Function**

The reward function defines the value for executing the right actions in the environment. The agent has to learn the best actions on a given state by a try and error process. It indirectly defines the agents behavior.

In the environment of Super Mario Bros. the reward function consists of three components. The first component rewards the movement in left and right direction. For each step, the positive x direction change of Mario's position $\Delta_x$ is calculated and used as reward. The second component is the negative reward for the degreasing time. The ingame time is represented as a countdown in seconds and displayed at the top right of the screen. The agent gets an negative reward $\Delta_t$ for each elapsed second. The last component is the death penalty $p_{\text{death}}$. The agent gets a negative reward of 25 for dying by running into enemies or jumping into holes. The following equation represents the sum of the reward components:

$$\text{reward} = \Delta_x - \Delta_t - p_{\text{death}}$$

Note that the reward function does not include any reward for increasing the ingame score for collecting coins or killing enemies. The reward function does more represent a behavior for a so called speed run scenario.

## Methodology

In this section, we present our Artificial Intelligence solution for the Super Mario Bros. environment using the introduced Deep Q-learning techniques. We describe our observed difficulties and problems for solving Super Mario Bros. with reinforcement learning and present our attempts and solutions to solve them.

### Frame Stack

The environment's observation space has a high dimensionality because of its low level pixel frame representation. This observation space causes two main problems. First, the high dimensionality needs to be reduced to enable a efficient learning process. To reduce the dimensionality we followed the solution of Mnih et al. [3] and used the same Convolutional Neural Network (CNN) architecture. Second, the static frame informations are not enough for controlling a real time video game. Since the algorithm is using the markov property, the agent has no possibility to remember last observations of the environment. Without this ability the agent is not able to detect any movement in the static images. This is a problem, because without this is required to estimate movement directions and velocities of the objects in the game. We observed this problem while watching the agent failing to learn how to perform a long jump. Long jumps are needed to surpass the tube obstacles in the game and are performed by holding the jump button. The agent had no possibility to detect if Mario is currently jumping upwards or falling downwards. We implemented a frame stack solution with four frames like Mnih et al. [6]. We used a queue to store the last three observations and passed them together with the current frame through the CNN.

### Episodic Life

In Super Mario Bros. the player has more than one lives before the game is over. In the game the player starts with three lives wich get reduced if Mario dies. If Mario loses a live the game continues at predefined checkpoints. In the used environment, one reinforcement learning episode is in accordance with one game with three lives. This means the death penalty of the reward function can be experienced multiple times in the same episode. This cases a problem with the concept of the death penalty. The learning algorithm is able to experiences the death penalty and is still able gain rewards, which lowers its effect and leads to unpredictable rewards. We used an so called "EpisodikLiveWrapper" in our python implementation to change the behavior to an on live per episode environment. With this change we observed, that the agent does more likely to learn to jump over abysses to avoid the death penalty.

### Small Gamma

The $\gamma$-parameter is set between 0 and 1 and sets the learners influence of future rewards. The lower the value the less the agent looks into the future. The jump and run side scrolling game of Super Mario Bros. highly depends on reaction speed and accuracy. A lower $\gamma$-parameter does encourage a behavior for such attributes by only rewarding the latest states immediately before the hazzard. We observed, that the agent is more likely to learn to avoid enemies and to jump over abysses with a low $\gamma$-parameter of 0.5.

## Results

This section describes the setup of our best results. Our best model is able to solve the first level of Super Mario Bros. The table 1 shows the used parameters of the Deep Q-Learning Algorithm.

**Table 1: The Parameters for the Deep Q-Learning Algorithm.**

| **Parameter** | **Value** |
|---|---|
| Learning Rate | 0.0001 |
| Max Timesteps | 100000 |
| Buffer Size | 50000 |
| Exploration Fraction | 0.3 |
| Final Exploration Rate | 0.1 |
| Learning Starts | 25000 |
| $\gamma$ | 0.5 |

We used the tool of tensorboard to observe the average reward of 100 episodes. The result is displayed in figure 3.

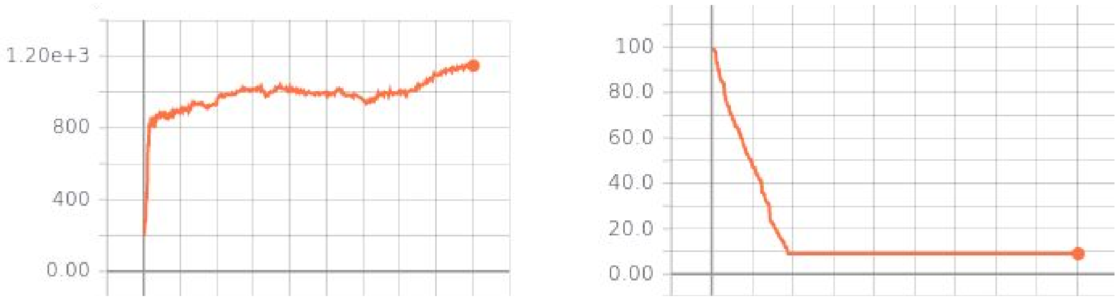

Figure 3: This two figures show the tensorboard output of the training process. Left is the average reward of 100 episodes and right is the exploration rate.

The training process with 100000 steps executed 452 episodes and took 1 hour and 16 minutes. A Video of the best run is available at https://github.com/ChristianLemke/mario_ai [7].

## Conclusion

To conclude, the optimization techniques, frame stack, prioritized replay buffer and dueling network yield positive results and improve the convergence. Also, it is important that the epsilon decays till a small value in the end. The penalty of death should be taken into account as soon as the agent is died once. $\gamma$-value can also influence the convergence of the network. In our trials, small $\gamma$ values yield better results.

## Reference

[1] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., ... & Osband, I. (2017). Deep Q-learning from Demonstrations. arXiv preprint arXiv:1704.03732.

[2] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952.

[3] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

[5] Kauten, Christian. (2018). Super Mario Bros for OpenAI Gym, URL: https://github.com/Kautenja/gym-super-mario-bros.

[6] Wikipedia, Accessed at 31.09.18, URL: https://en.wikipedia.org/wiki/Super_Mario_Bros.

[7] Xingying Chen and Christian Lemke. (2018). Mario AI. URL: https://github.com/ChristianLemke/mario_ai/.