# MS-Pacman\_v0 OpenAI GYM DQN-Learning

Author: Parsa Pourzargham

e-mail: [lum2ce@inf.elte.hu](mailto:lum2ce@inf.elte.hu)

Neptun ID: LUM2CE

## Abstract

## Introduction

The problem which this article tries to solve is an Atari game called Pacman, in which, the agent must learn how to maximize its reward in the environment provided by GYM python library where agent must learn how to collect the maximum number of dots, while avoiding the moving ghosts in the closed maze.

convolutional neural network is utilized to train the agent which maps the input state to actions. The motivation of applying the DQN algorithm on this problem is to see how good this algorithm can perform in this specific task and to check it can reach a near-human level performance and beat human record in the game. Also, to find out how much benefit the model gets by utilizing the neural networks alongside with the Q-Learning, compared to more basic solutions like vanilla-Q-Learning.

The input to the algorithm is the in-game grayscale screen images, fed to the neural network that output the probability of each action also known as Q values.

## Related work

Breaking your report into sections can make it much easier to read. Main sections

## Problem description

In the game, Pacman is the agent which has to collect as much seed as possible while avoiding 4 ghosts moving to catch it in the closed maze field. The agent has 9 possible actions to take in each step:

* NOOP(No Operatoin,not moving in the maze)
* Up
* Down
* Left
* Right
* UpRight
* UpLeft
* DownRight
* DownLeft

The environment for the problem is provided by the GYM library in python programming language. It takes the action from the agent and returns the new state of the environment after performing the action, as well as the immediate reward of the taken action as a feedback to the agent.

## Methodes

As mentioned, the algorithm used to train the agent is Deep-Q-Learning, DQN in short. DQN is also a reinforcement algorithm which is derived from the classical Q-Learning algorithm. Unlike classical Q-learning algorithm which maps each state-action pair to its corresponding Q-Value using a table called Q-table, DQN maps the state-action pairs to their corresponding Q-value using a deep neural network.

The overall workflow of the algorithm is as following:

1. The neural network takes the in-game screen of the game as the input and calculate the tensor of probabilities of actions. To obtain a tensor of probabilities, the activation function of the las layer (fully connected layer with output number of the action space size) is set to None.
2. As the networks take the current state of the environment to give the probability tensor of actions, a grayscale image of the game is taken and fed into the network to calculate the corresponding Q-values.
3. After obtaining the probability tensor, the action with the highest probability is chosen. It is delivered by the function argmax ().
4. Having chosen the action in the previous step, the action is given to GYM and it provides us with the new state of the environment after taking the action, as well as the immediate reward corresponding to the taken action and whether the game is finished or not.
5. After obtaining the above-mentioned parameters from the environment, the parameters are stored in a buffer in the format of (s,a,r,s’,d). s denotes the state before taking the action, a denote the taken action, s’ denotes the new state after taking the action and d stand for the Boolean which decide the game is done or not.
6. After finishing the step 5, the algorithm needs to calculate the loss to train the network. r’ and a’ are required to calculate the loss. a’ could be obtained by feeding s’ to the network, while r’ is discounted from r. now, the loss can be calculated and hence, the loss can be obtained.
7. Finally when the training process is finished, the network can be used by the agent to play the game and evaluate the score of the agent. This is done by feeding the current state to the network and choose the action according to the probability tensor provided by the network.

Remarks:

* Updating the model weights frequently in every step results to low performance of the model as it updates the weights when nothing much has changed. As a result, the model weights are updated every 4 steps.
* Two networks are used to train the agent in this algorithm. Although both have the same architecture, they have different set of weights and different purposes in the model. Two networks are called main network and target network. the main network weights are updated after each 4 steps in the training process, while the main network weights are copied to the target network every 100 steps. The correct frequency of copying the main network weight to the target network is obtained by trial and error to 100 steps. According to [], having these two different networks is beneficial in terms of stability in the learning process and helps the model to learn effectively.

## Experiments and Discussion

The idea and parts of the formating for this template came from the Association for Learning Technology (UK) ALT-C 2004 Research Paper Format Template.

## Conclusion, further work

## References