# Learning algorithm

## Overview of the technique

The agent is implemented using the Deep Q-Learning method (along with Replay Buffer and Fixed Q-Target technique). At each time step, the agent will take an action, store the transition in its replay buffer (Sample) and then sample a minibatch from the buffer to update the primary Q-network with the target Q-network. The target network is updated either after a number of steps by copying the primary network’s weights or slowly update by (a hyperparameter):

target = \* primary + (1-) \* target

The algorithm in detail:

Text

Description automatically generated

Besides, in order to help the agent learn better, I have implemented the algorithm with 2 improvements: Duelling DQN and Double DQN to avoid overestimating problem.

\*) Double DQN:

When learning:

* Vanilla DQN estimate the true value for computing loss by the maximum action-value at the next state, which return from the target network. This can lead to overestimation of action-value.  
  Formula for this true value estimation can be written as:

(Select best action from the target network and evaluate that action on the target network)

* Double DQN solve this by select the best action from the primary network and evaluate that action on the target network.
* Formula for this estimation

\*) Dueling DQN

Its network architecture is different from vanilla DQN. It uses 2 streams, 1 to compute the value of state and 1 to compute the advantage of each action in that state.

Diagram

Description automatically generated

The output of the dueling network is computed as follow (forcing the advantage function estimator to have 0 advantage at the chosen action):

## Network architecture

The network architecture for both primary and target Q-network is described below (the number is # neurons at each hidden layers)

Chart, waterfall chart

Description automatically generated

## Hyperparameters select

Text

Description automatically generated

# Plot of rewards

Chart, line chart

Description automatically generated

# Ideas for future works

The result I have shown is not too good, which solve the problem in about 4k episodes ☹ I think some modifications will make the result better:

* Tuning hyperparameters such as epsilon (start, end, decay) and the architecture of the Q-network probably enhance the learning ability of the agent
* Moreover, many other techniques can be used in order to help the agent learn better such as Prioritized Replay Buffer, Distributing DQN, … or even combine all – Rainbow.