RL vs Deep-RL in Games

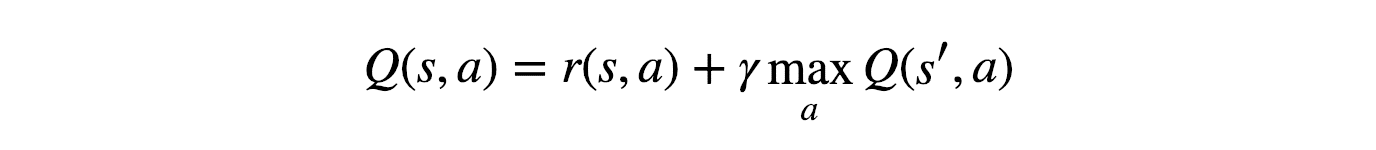
## Introduction

### RL and Deep-RL

**Agent-Environment interaction**

* Main essence of Reinforcement learning is an iterated interaction of an agent and an environment
  + Agent sends actions
  + Environment send states and rewards

**Q-learning**

* Maintiain a table of Q-Values:
  + Key: state-action pair
  + Value: determined by update rule
* Action is chosen with epsilon-greedy policy
  + Chooses random action with probability eps : Exploration
  + Chooses best known action (from q-values) with probability 1-eps : Exploitation
* Q-Values are updated according to:
* 
  + Q-Value of current state and action =
  + Reward of current state and action +
  + Discount factor \* maximum reward possible in the next state
    - Higher discount factor means a higher focus on long term rewards

**Rainbow**

* Deep-RL and DQN
  + Instead of a table of values, train a deep neural network to give a Q-value for a state-action pair
  + Trained using gradient descent to minimise loss function
  + A black text with a white background

    Description automatically generated
  + On ‘Replay Buffer’ – Transitions <S\_t, A\_t, R\_t+1, γ\_t+1, S\_t+1>
  + R\_(t+1) : Rewards of current state and action +
  + Discount factor \* (max reward of next state) - (from *target network*  - copy of online network, not directly trained)
  + Q-Value of previous state and action
* Rainbow’s Improvements
  + The rainbow agent combines a handful of improvements to DQN, namely:
    - Double Q-Learning
      * Modifies loss function to prevent overestimation of next state reward (agent will not always pick best)
    - Prioritised Replay
      * Samples ‘better transitions’ from replay buffer
    - Dueling Networks
      * Separates Values of states and Advantages of actions and combines
    - Multi-Step Learning
      * Considers *n* steps in the future for loss function
    - Distributional RL
      * NN learns approximate distribution of rewards rather than expected value
    - Noisy Nets
      * Introduces noisy linear layer to address limitations of eps-greedy policy
        + Situations where many actions are needed before first reward

### Mariokart Wii

**Emulator**

* Dolphin build with embedded python
  + Allows running scripts with access to Dolphin API

**State Space**

* Represented by a thruple
  + Current Velocity
  + Current race% -> how far around the track the agent is
  + Miniturbo Charge

**Action Space**

* Represented by a dict
  + A Button - Bool
  + B Button - Bool
  + Up Button - Bool
  + Control stick X value – [0,64,128,192, 255]

## What I have done

### Emulator

**RL**

**Environment**

* Get next state
* Checks if state is terminal state
* If Yes
  + Resets variables
  + Loads savestate
* If No
  + Gets next action by eps-greedy policy
  + Calculates reward – speed, race%, mt
  + Updates q-table
  + Updates total reward
  + Updates previous state info
  + Sends action to dolphin

**Agent**

* Action Space
  + Every permutation of controller
* State Space
  + Not initialised – (float, float, int) has many permutations
  + Implemented helper functions to handle new states (not visited before)
* Eps-greedy
  + Returns tuple of action taken and whether it is Exploring or Exploiting
* Update Q
  + Returns q-value of (state, action)

Parameter Tuning

* Epsilon : Currently constant through training, high to encourage exploration of wide Q-Space
* Gamma: high to encourage focus on future rewards – hopefully completing lap
* Alpha – Deterministic environment : 1

**Deep-RL**

**Environment**

Very similar inputs except DQN takes pixel values as well as reward value

Socket setup

* Dolphin’s integrated python is very basic – only officially supports standard library
* To allow for pixel data processing and NN support, send frame data and reward to, and receive action from Rainbow agent running in separate terminal using sockets

Frame Processor

* As soon as dolphin draws a frame a png of it is saved to dolphin’s /dumps/frames folder
* I access this frame and downsample and greyscale to speed up processing

(https://github.com/benjaminjmiddleton/mkw\_ai\_env/blob/main/README.md)

## What I have to show

Agent running with Q-Learning

Graph with x = episode, y = (episode length, reward)

## Evaluation

Value-based evaluation – reward

Time-based evaluation – total frames

## Next steps

State-space alterations – less accurate but quicker learning

Tune reward function – values chosen arbitrarily – improve weightings