Reinforcement Learning

Arash Sajjadi

Shahid Beheshti University[1]

Workshop of Data Science
January 20, 2023

Outline

Table of Contents

Introduction

Reinforcement learning is a part of machine learning that tries to do things to maximize the rewards of actions in a particular environment. Reinforcement learning algorithms aim to find the best action in a given environment. This type of machine learning can learn to perform the learning process and achieve its goals even in complex and uncertain environments. Just like the human brain, the system is rewarded for good choices and penalized for bad decisions, and it learns from each action.

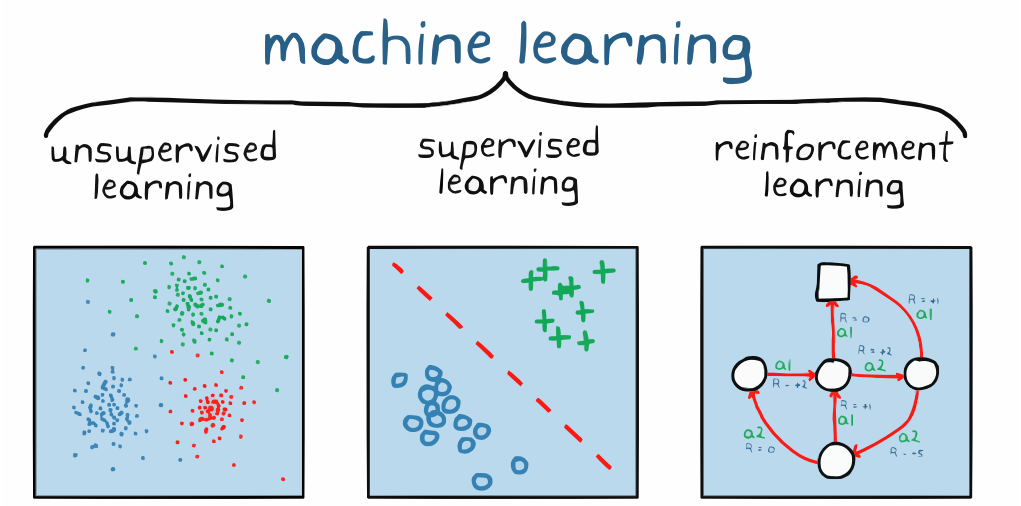


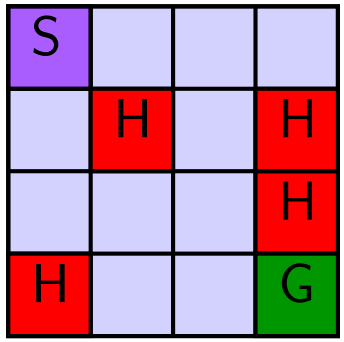
Table of Contents

Tabular methods

Frozen lake game(*Simplified version*): The goal of this game is to go from the starting state (S) to the goal state (G) by walking only on frozen tiles (F) and avoid holes (H).

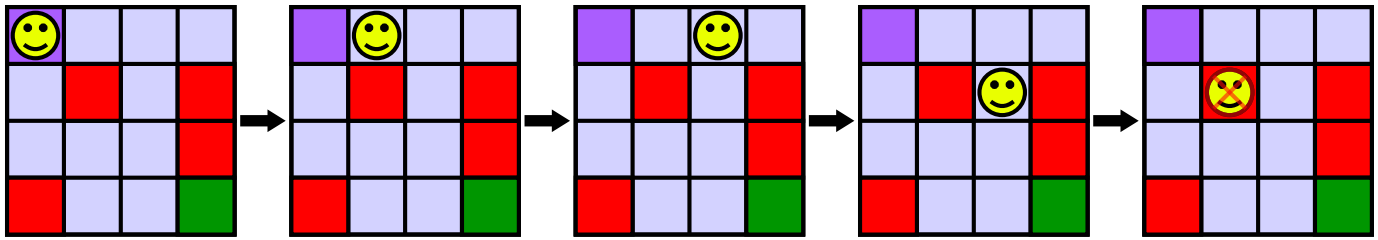
Num	Observation (State)
0 - 15	For 4x4 square, counting each position from left to right, top to bottom

Num	Action	
0	Move Left	$\leftarrow$
1	Move Down	$\downarrow$
2	Move Right	$\rightarrow$
3	Move Up	$\uparrow$

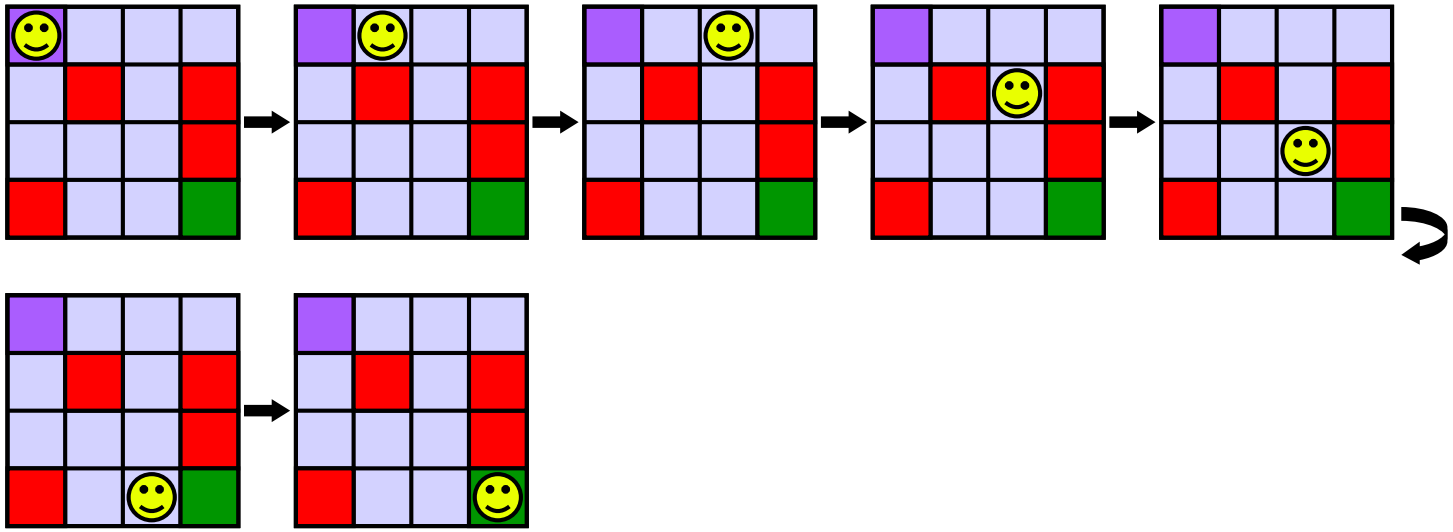


Tabular methods

1. In most random executions, the agent does not reach the goal and falls into the trap of one of the holes.



2. In rare cases, the agent reaches the goal. Note that achieving the goal is not necessarily the shortest path. (*Evolutionary solution*)



Tabular methods

After training the agent, we should reach the Q table. The mission of this table is to tell us which action is the best in each state. Note that the table is not a Q table. It is simply an interpretation of a Q table.

Tabular methods

To get the values of the Q table, we use the recursive form of the **bellman equation**.

$$Q(s,a) = r + \gamma \times Q(s',a')$$

Algorithm for *deterministic* environment:

---

initialize Q[num_states,num_actions]
observe initial state s
Repeat until terminated:
    select and perform an action
    observe reward r and state s'
    $Q(s,a) = r + \gamma \times Q(s',a')$
    $s = s'$

Tabular methods

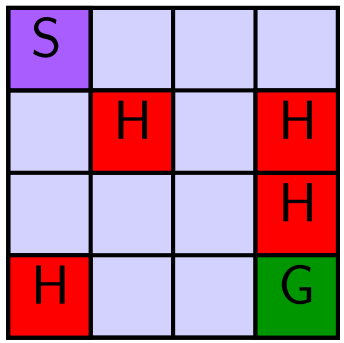
- What if we don't know everything about environment?
- What if we only know set of states and actions?

Q learning as an answer

Frozen lake game: The goal of this game is to go from the starting state (S) to the goal state (G) by walking only on frozen tiles (F) and avoid holes (H). However, the ice is slippery, so you won't always move in the direction you intend (*stochastic environment*)

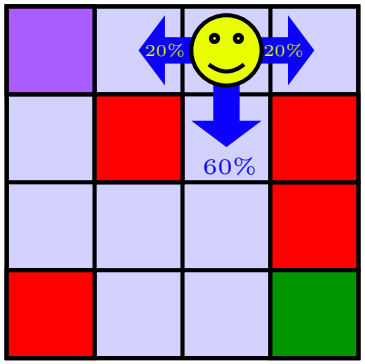
Num	Observation (State)
0 - 15	For 4x4 square, counting each position from left to right, top to bottom

Num	Action	
0	Move Left	$\leftarrow$
1	Move Down	$\downarrow$
2	Move Right	$\rightarrow$
3	Move Up	$\uparrow$



Tabular methods

Temporal difference:



Temporal Difference Learning is an unsupervised learning technique that is very commonly used in reinforcement learning for the purpose of predicting the total reward expected over the future. They can, however, be used to predict other quantities as well

$$TD = [r + \gamma \times \max_{a'} Q(s', a')] - [Q(s, a)]$$
$$Q_t(s, a) = Q_{t-1}(s, a) + \alpha \times TD \Rightarrow$$
$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[r + \gamma \times \max_{a'} Q(s', a')]$$

Q learning

Tabular methods

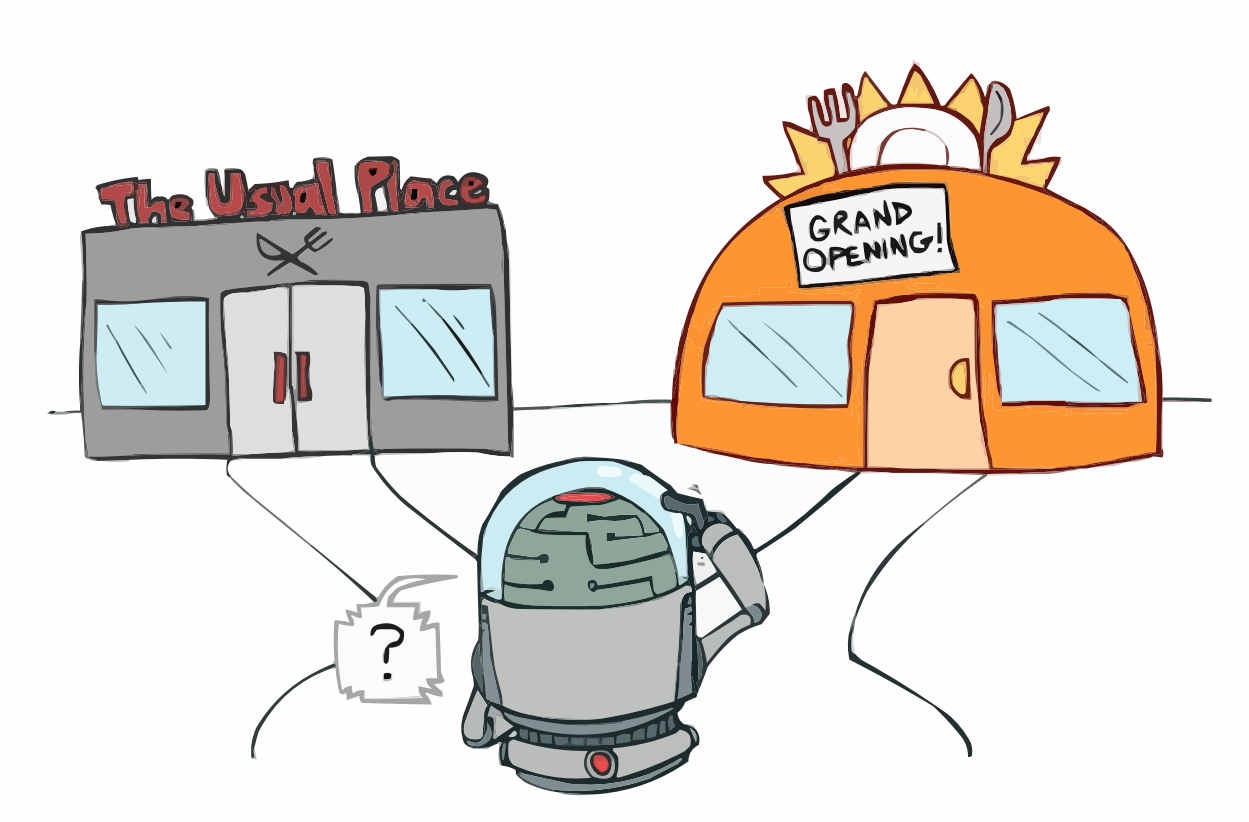
Algorithm for *stochastic* environment:

---

initialize Q[num_states,num_actions]
observe initial state s
Repeat until terminated:
    select and perform action a
    observe reward r and state s'
    $Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max Q(s',a')]$
    $s = s'$

Tabular methods

Exploitation vs Exploration



$$a = \begin{cases} \text{optimal } a^* & 1-\epsilon \\ \text{random action} & \epsilon \end{cases}$$

Tabular methods

The exploration-exploitation trade-off: Intuitions and strategies

1. Optimistic initialisation
2. ϵ-greedy
3. Upper-confidence-bound action selection
4. Gradient bandit algorithms

Tabular methods

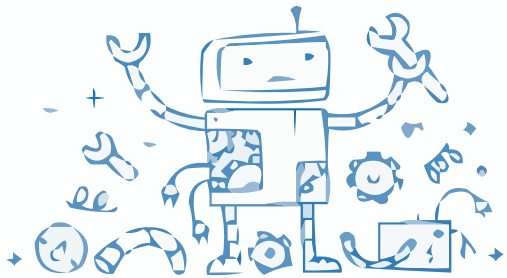
Suppose we are going to train our agent in the Tetris game environment. In this case, how many different states will we have for this game? 10^{60}

Table of Contents

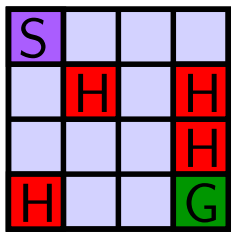
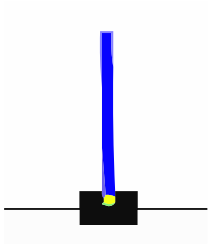
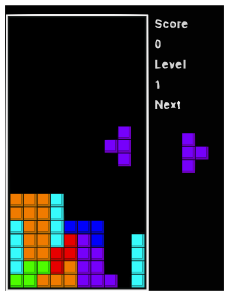
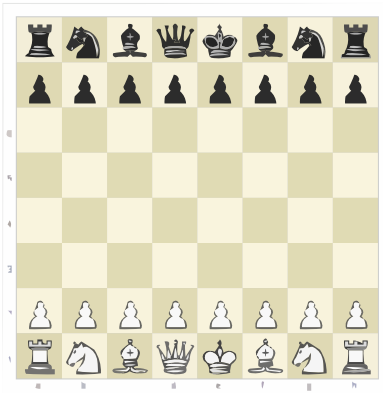
Scaling up

What are the problems with the current approach?

1. The high volume of the set of states
2. The need for mighty computing resources
3. Performance



Scaling up

FrozenLake	Cartpole	Backgammon	Tetris	Chess
				
16	512	10^{20}	$2^{200} \approx 1.6 \times 10^{60}$	$(4.59 \pm 0.38) \times 10^{44}$ with a 95% confidence level
Discrete	Continuous	Discrete	Discrete	Discrete

Scaling up

What is the solution?

- Generalize
- Function approximation
- Understand the rules and apply it to future, similar situations

Scaling up

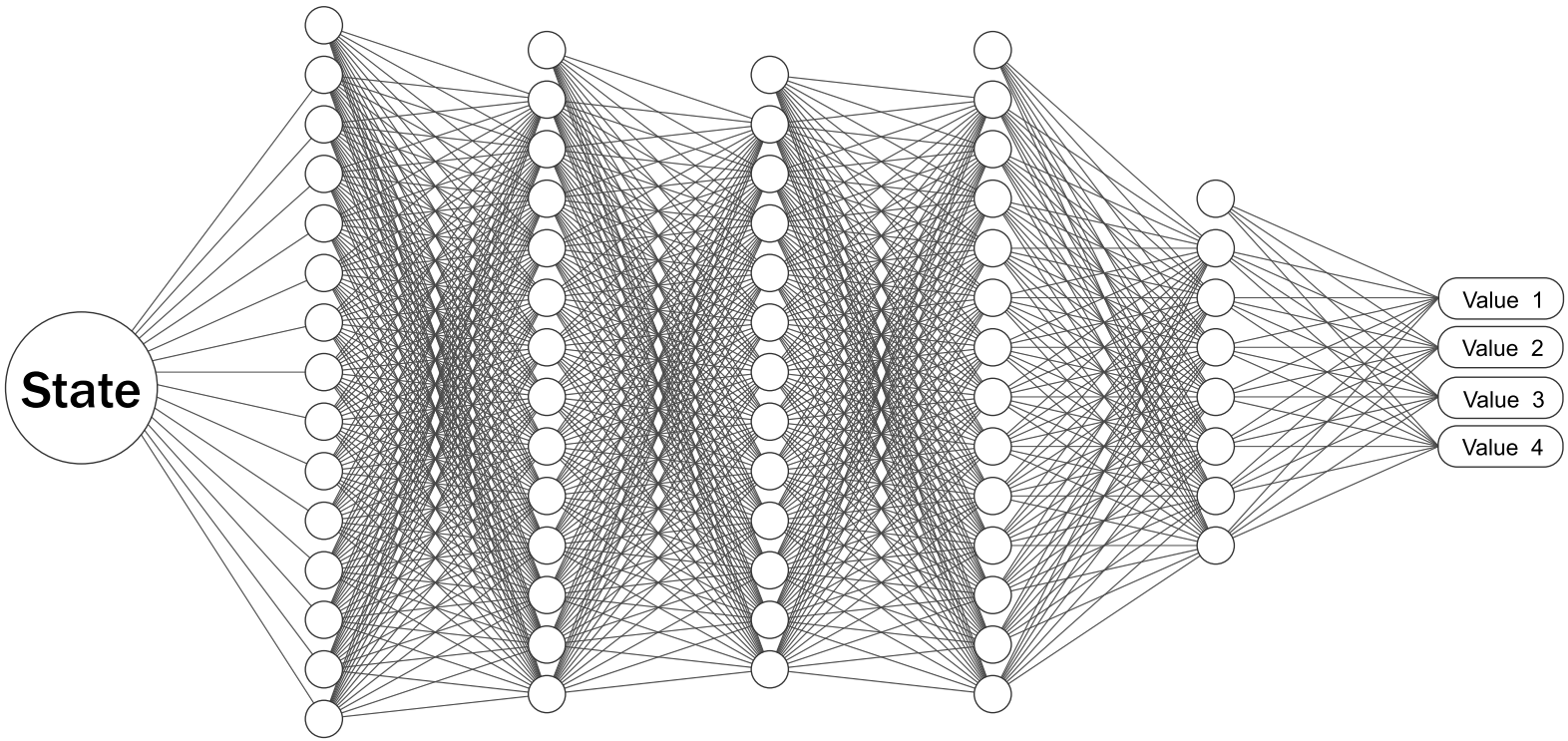


Table of Contents

DQN

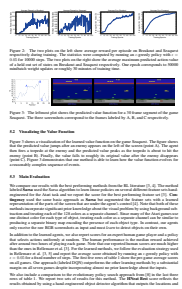
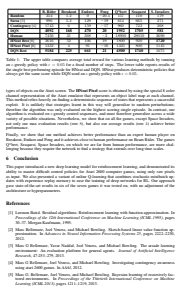
Deep Q learning

Paper "Playing Atari with Deep Reinforcement Learning"







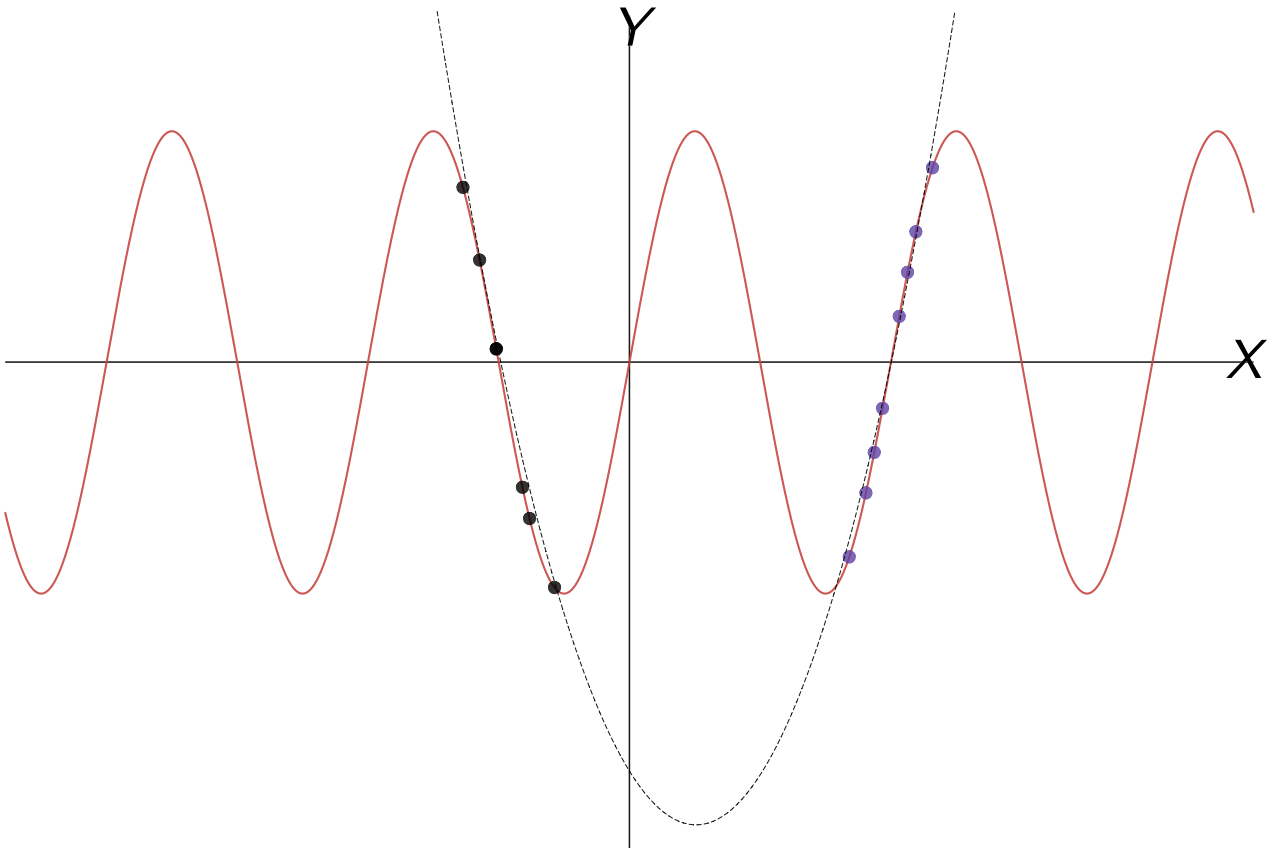




Problems listed in Deepmind's paper:

1. highly correlated data *Part 1, third paragraph (Page 1)*
2. non-stationary distributions *Part 1, forth paragraph (Page 1-2)*

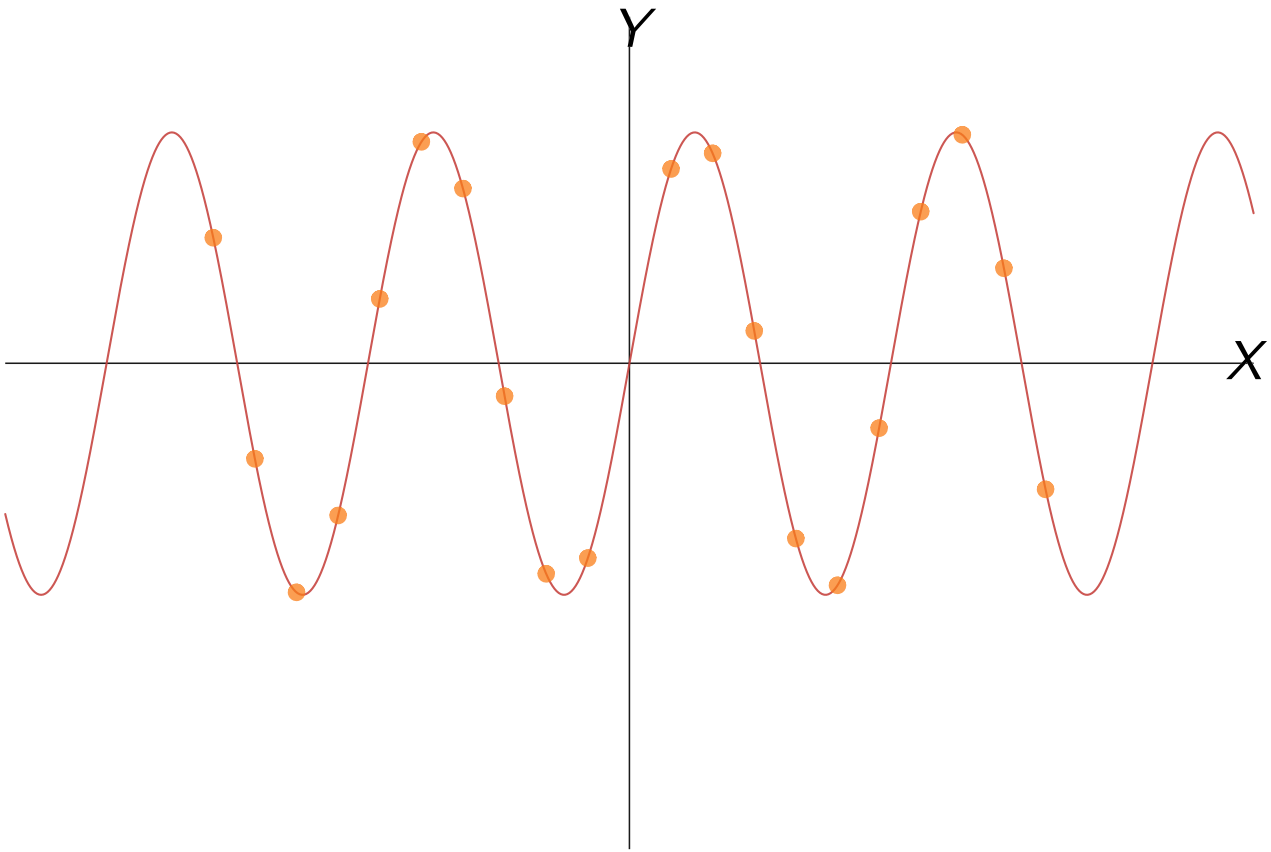
DQN

highly correlated data



DQN

highly correlated data



DQN

non-stationary distributions

$$Q(s,a) = r + \gamma \times \max_{a'} Q(s',a')$$

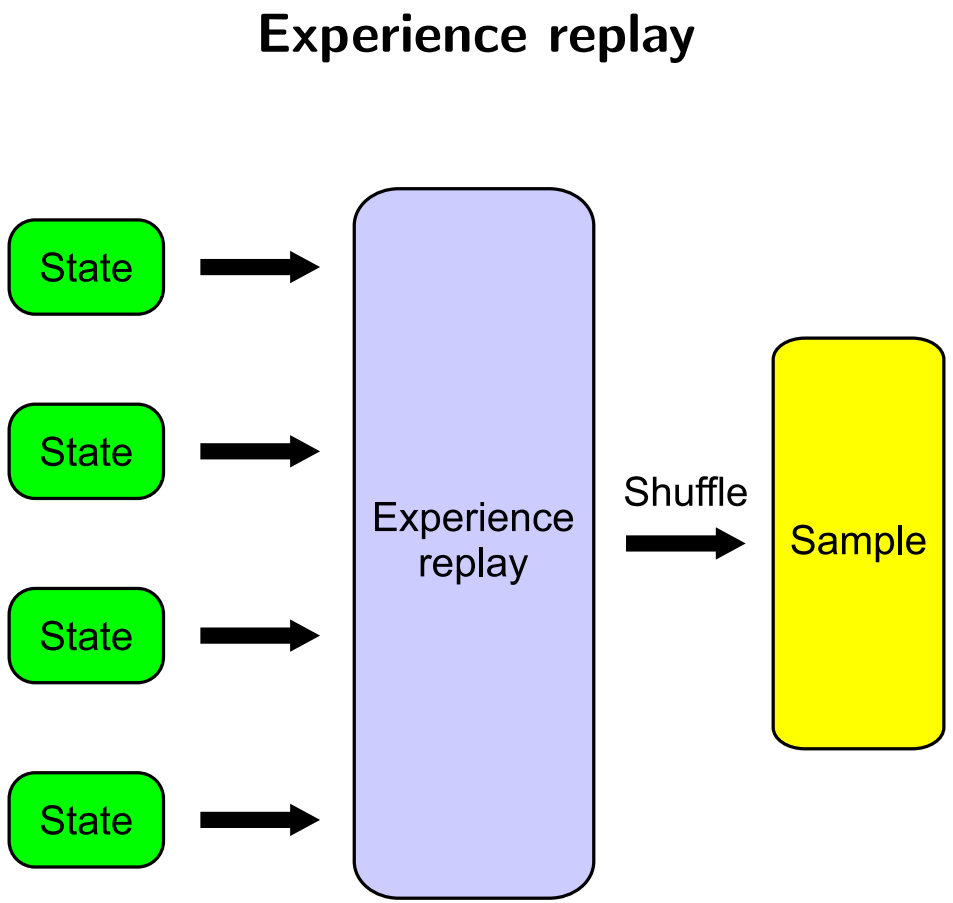
DQN

The solution to these two problems was in a dissertation written in 1993 by a doctor of philosophy named Long-Ji Lin.

✓ A dissertation titled "**Reinforcement learning for robots using neural networks.**"

Experience replay

DQN


Experience replay
State
State
State
State
Experience replay
Shuffle
Sample

DQN

Experience replay

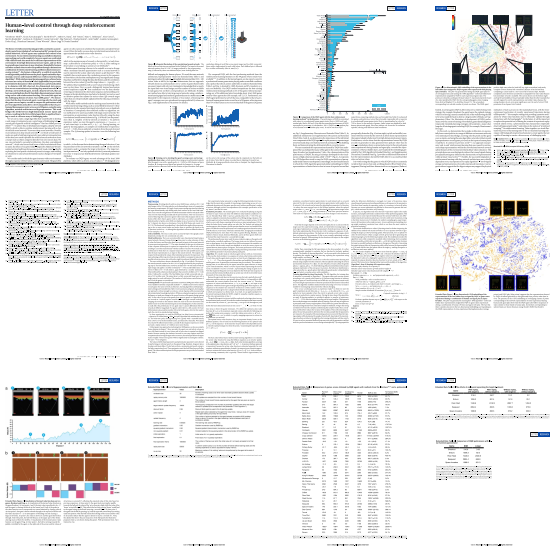
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DQN

Deep Q-Network[1]

Paper "Human-level control through deep reinforcement learning"



The problem listed in Deepmind's paper:

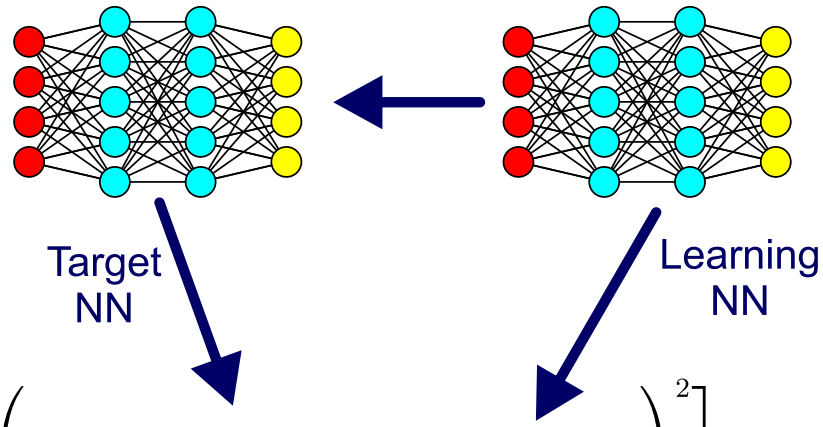
- Non stationary target, *the forth paragraph (Page 1)*

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a; \theta_i^-)}_{\text{Target}} - Q(s, a, ; \theta_i)\right)^2\right]$$

[1] A DQN, or Deep Q-Network, approximates a state-value function in a Q-Learning framework with a neural network.

DQN

✓Non stationary target



$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a; \theta_i^-)}_{\text{Target}} - Q(s, a, ; \theta_i)\right)^2\right]$$

DQN

Detail

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do** → One game

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Take a step

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Experience replay

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$ → Fix Q target

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Table of Contents

DQN Improvements

Double DQN[2]

Paper "Deep Reinforcement Learning with Double Q-learning"

The problem listed in Deepmind's paper:

- Overoptimism (Overestimations)

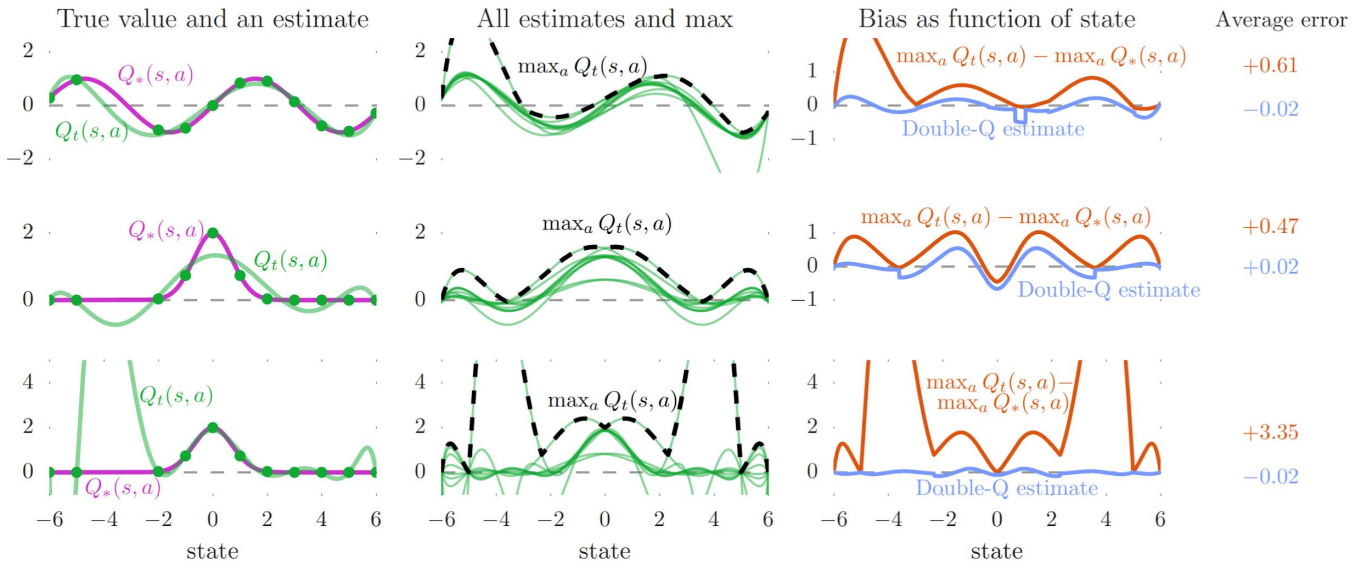
[2] A Double Deep Q-Network, or Double DQN utilises Double Q-learning to reduce overestimation by decomposing the max operation in the target into action selection and action evaluation.

DQN Improvements

Problem: Overoptimism

- Q-learning methods are known to be overestimating the Q value
  - Even if the true Q values are given, estimating it by sampling points introduces error, which will be amplified by bootstrap multiple estimations and pick the largest

- Q* is the true value
- Q* is sampled in green points
- Q_t is a polynomial estimate of Q* in different degrees
- Bootstrap several Q_t to get their max line



DQN Improvements

Double Q-Learning: A Solution for Over-Estimation Bias in Q-Learning

Double Q-learning is a variant of Q-learning that addresses the problem of over-estimation bias by using two separate Q-value functions, one for action selection and one for value evaluation. This approach can reduce **over-estimation** and improve **stability** and **performance** in Q-learning algorithm. The formula of double Q-learning can be represented as:

$$Q_1(s,a) = \max(r + \gamma * Q2(s',a'))$$

$$Q_2(s,a) = r + \gamma * \max(Q1(s',a'))$$

Where s is the current state, a is the current action, r is the immediate reward, γ is the discount factor, s' is the next state, and a' is the next action.

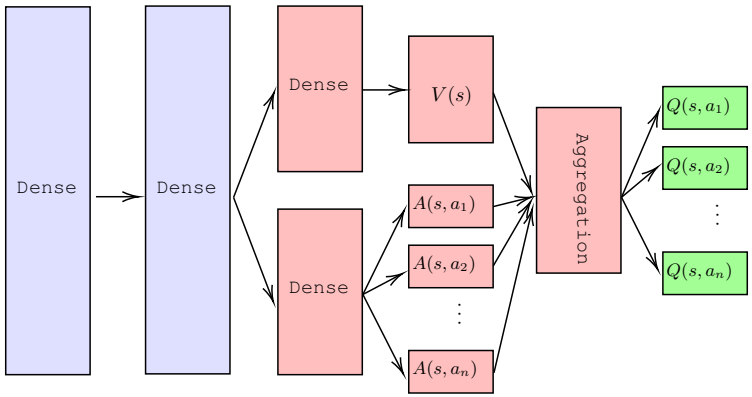
- Q_1 function is used to select the best action to take in the current state
- Q_2 function is used to evaluate the value of the action selected by Q_1

DQN Improvements

Dueling DQN[3]

Paper "Dueling Network Architectures for Deep Reinforcement Learning"

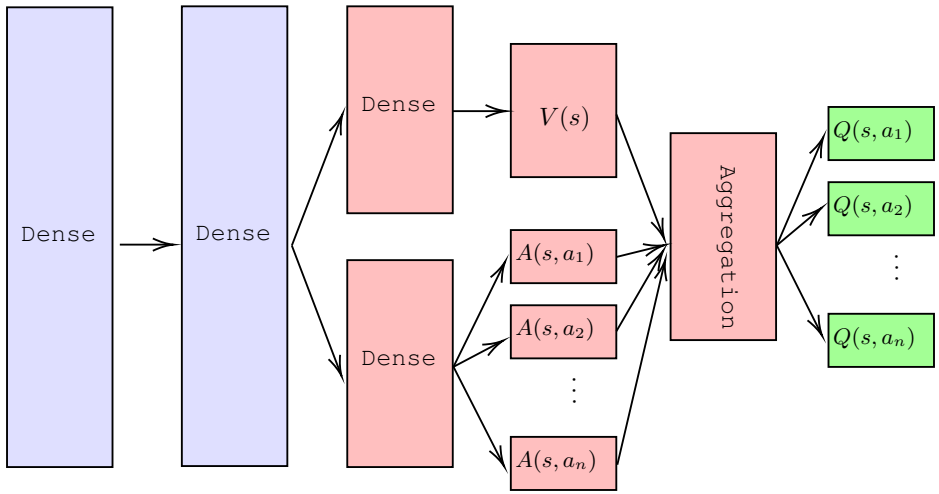
the architecture of a Dueling DQN neural network is composed of two main parts: the value stream and the advantage stream. The **value stream** estimates the value of the *current state*, and the **advantage stream** estimates the *relative value of each action in the current state*. Both streams are then combined to produce the final Q-value function, which estimates the expected future reward for each action in the current state.



[3]Dueling DQN is a variant of Q-learning that improves stability and performance in complex environments by separating Q-value function into value and advantage functions.

DQN Improvements

Dueling DQN: Simplifying Q-value function



Dueling DQN separates the Q-value function into two parts: the value function and the advantage function.

- **Value function**: $V(s) = E[r + \gamma * max(Q(s', a'))|s]$
- **Advantage function**: $A(s, a) = Q(s, a) - V(s)$
- **Final Q-value function**: $Q(s, a) = V(s) + A(s, a)$

Dueling DQN **improves performance** and **stability** in complex environments by simplifying the Q-value function calculation.

Table of Contents

DQN with video output

Combining Computer Vision and Reinforcement Learning for Atari Games

- Combining computer vision and reinforcement learning can **improve the performanc**e of agents in Atari games.
- One popular approach is to use a **CNN** (Convolutional Neural Network) agent to process **visual input**.
- CNNs can extract features from images and improve the agent's ability to **recognize patterns** in the game.
- Using CNNs with RL algorithms such as DQN and DDQN[4] can improve the agent's ability to **learn from raw image input** and play the game **more effectively**.

[4]DDQN (Deep Double Q-Network) is a variant of DQN that addresses the problem of over-estimation bias by using two separate Q-value functions, one for action selection and one for value evaluation. This improves the stability and performance of DQN algorithm.

DQN with video output

Combining Computer Vision and Reinforcement Learning for Improved Performance in Atari Games: Advantages and Disadvantages

Advantages:

- Improved performance in Atari games
- Better recognition of patterns in game
- Ability to learn from raw image input

Disadvantages:

- Increased computational cost
- Requires large amounts of data

Overall, combining computer vision and reinforcement learning, especially using CNNs, can be an effective approach for improving the performance of agents in Atari games, but it comes with increased computational cost and requires large amounts of data.

DQN with video output

RL-CNN Pong Game Agent

- RL-CNN Pong Game Agent
  - Implements deep RL using CNNs for the game Pong
  - Goal: Train an RL agent to play Pong by watching a video and interacting with the environment
  - Based on:
    - Multiple papers including:
    1. "Playing Atari with Deep RL"
    2. "Human-level control through deep RL"
    3. "Deep RL with Double Q-Learning"
    4. "Dueling Network Architectures for Deep RL"
  - Used to:
    - Improve performance
    - Tackle over-estimation bias in Q-learning
  - Implemented in:Click Here
    - Python
    - Using Pytorch, Numpy, Matplotlib, and Gym
  - To run:
    - Execute `PongVideoOutput.py` to train agent
    - Display results in video form

DQN with video output

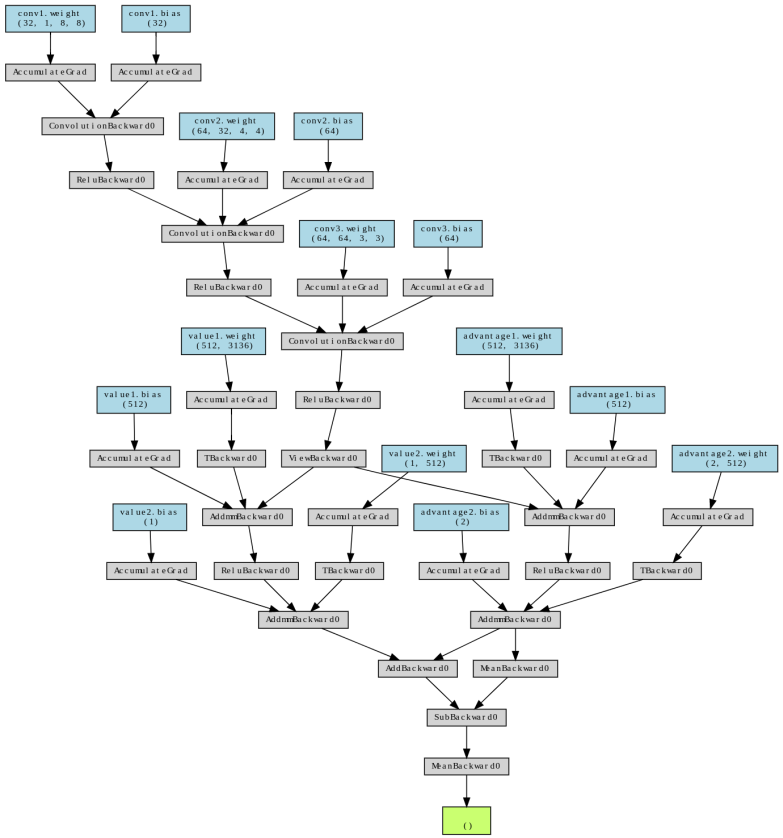


Figure: Visualization of the architecture of the Dueling Double Deep Q-Network (DDDQN) used in the RL-CNN Pong Game Agent project

DQN with video output

Learning Curve of RL-CNN Pong Game Agent

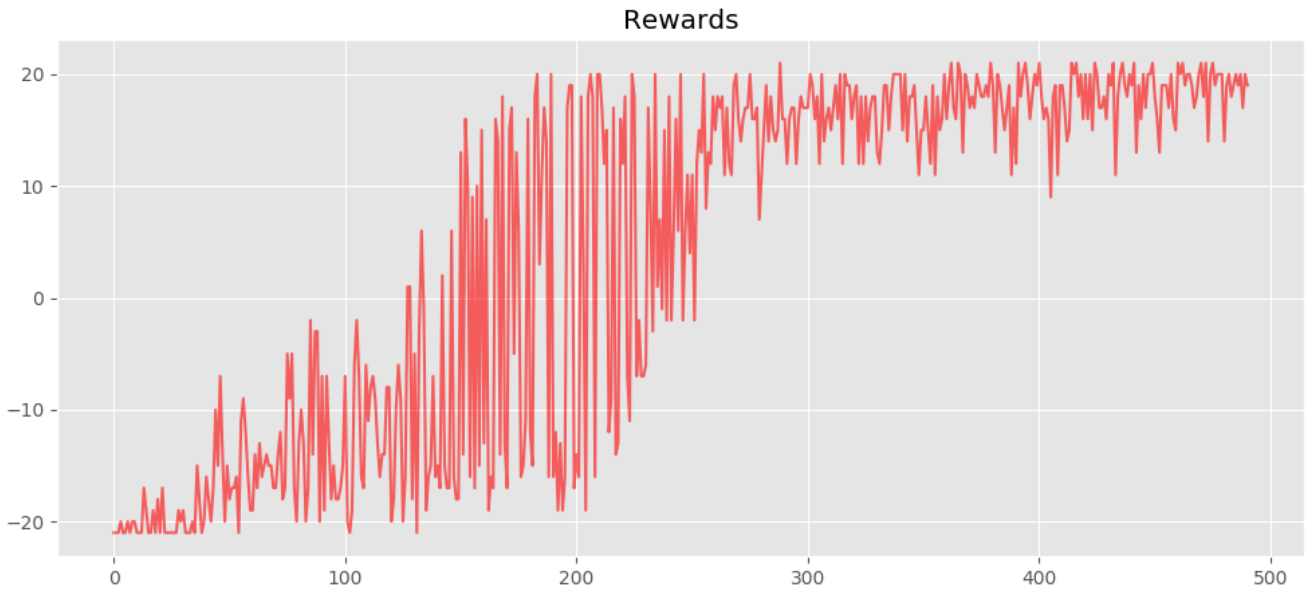


Figure: This plot illustrates the increasing trend of the learning curve of the RL agent in the RL-CNN Pong Game Agent project, showing the agent's improvement in playing the game over time until convergence.

I would like to acknowledge the authors of the papers "Playing Atari with Deep RL", "Human-level control through deep RL", "Deep RL with Double Q-Learning" and "Dueling Network Architectures for Deep RL" for their contributions to the field of deep RL. Special thanks to Prof. Hajiabolhassan for teaching me RL and to the officials and presenters of the ANN course.

Thank you for your attention!

E-mail: Arash.Sajjadi.1@gmail.com