Chef's Hat - Reinforcement Learning Competition

Udi Daniel
Orr Moran
Tal Hollander

Chef's Hat Card Game

Chef's Hat is a card game which is divided into rounds, each round (or shift) divided into three phases - start of the shift, making pizzas and the end of the shift. Each player gather points by making pizzas and the first player that achieves 15 points wins the game.

A shift starts when all the cards (there are 68 ingredient cards, and 11 different ingredients) are distributed among the players. Players then exchange cards according to their ranks (for example, the Chef which is the most skilled player in the kitchen gets the rarest ingredients from the Dishwasher, who has the lowest rank in the kitchen).

In the pizzas making phase, players are using their ingredients cards to make pizzas on the same pizza base, and at the end of each shift points are given to the players and the roles are re-assigned according the the ranking of current shift.

Q-Learning, Deep Q-Learning and Double Deep Q-Learning

The approach we've chosen to use to develop our agent is using Q-Learning, and especially Deep and Double Deep Q-Learning. Q-Learning is a reinforcement learning algorithm, meant to learn the quality of actions, telling the agent what action to take according to the current state of the agent or the environment (Q-Values).

Deep Q-Learning (or "DQN") is an algorithm that combines Q-Learning with deep neural networks the reinforcement algorithm to work for complex, high-dimensional environments.

In a DQN, The next action is determined by the maximum output of the Q-network and the loss function here is mean squared error of the predicted Q-value and the target Q-value – Q* (This is basically a regression problem). DQN uses "experience replay", a biologically inspired mechanism that uses a random sample of prior actions (from a stored replay buffer) instead of the most recent action to proceed. This removes correlations in the observation sequence and smooths changes in the data distribution.

Iterative updates adjust Q towards target values that are only periodically updated, further reducing correlations with the target. Because the future maximum approximated action value in Q-learning is evaluated using the same Q function as in current action selection policy, in noisy environments Q-learning can sometimes overestimate the action values, slowing the learning.

A variant called Double Q-learning was proposed to correct this. Double Q-learning is an off-policy reinforcement learning algorithm, where a different policy is used for value evaluation than what is used to select the next action. In practice, two separate value functions are trained in a mutually symmetric fashion using separate experiences.

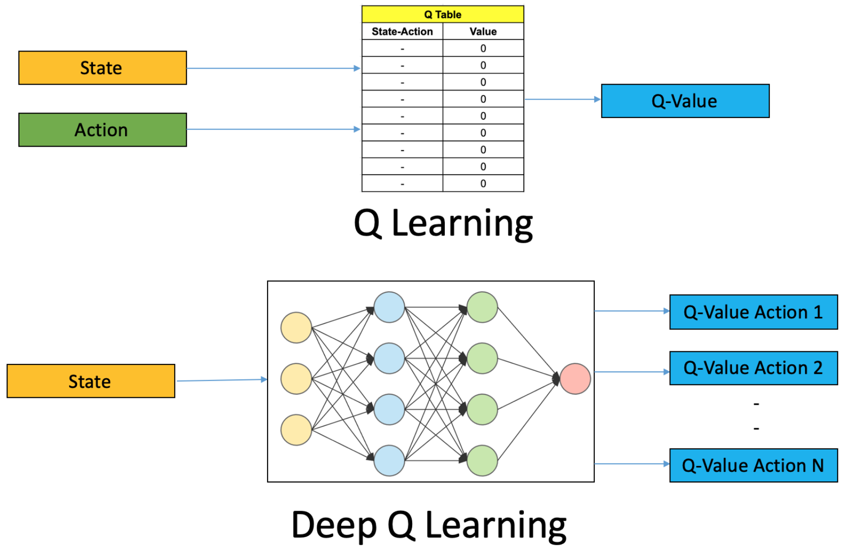

Figure 1: Q-Learning and Deep Q-Learning

Training Process

Our training setup include 4 Learning Agents (D-DQN) and one Experience Replay Buffer.
All the Agents are using the same buffer to write their experience and to draw random samples for training.

Hyperparameters

Network Parameters:
layer1 size = 64
layer2 size = 64
activation = Relu

Experience Replay:
max buffer size = 100000
min buffer size = 1000

Exploration Exploitation:
epsilon start value = 1.0
epsilon decay factor = 0.995
epsilon min value = 0.2

Learning Parameters:
batch size = 64
optimizer = Adam

optimizer learning rate = 0.0005
update policy network interval = 4
update target network interval = 100
gamma = 0.995
tau = 0.001

Code

Our code is accessible in the following Github repository: https://github.com/udid/CHEFS_HAT_CUP