# DQN with Replay Memory

June 17, 2024

## 1 DQN with Replay Memory

A Deep Q Network (DQN) is a Q-learning algorithm that uses neural networks to estimate $Q$ - values [Mnih et al., 2013].

For larger state-action spaces, it becomes challenging to maintain and update the Q-table. Thus, the DQN provides an efficient alternative to approximate the $Q$ - values. The DQN takes the state as input and outputs the $Q$ - values for each action.

The DQN updates the parameters of the network based on samples of past experiences which are collected from agent's interaction with environment. The replay experience helps remove any corralation between data by randomly sampling the experiences, which is more efficient than using the consecutive samples. The traditional algorithms make updates using an experience and discard it immediately. The replay experience on the other hand allows each experience to be used for multiple weight updates, thus increasing data efficiency. Third, in on-policy learning, the agent's current parameters influence the data it trains on next. For example, if agent this the best move is to go left, most training data will come from left-side actions. If the optimal move changes to right, the training data will shift to the right. This can create feedback loops, causing the agent to get stuck in a poor strategy or even experience catastrophic divergence. By using experience replay the behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.

## 2 Training the DQN

Initially, all the weights of DQN are initialized randomly. During the training process, the error that needs to be minimized is the difference in the computed $Q$-value $q(s, a)$ and the target $Q$-value $q^*(s, a)$. (* denotes the optimal or the best $Q$ - value based on the optimal policy). Mathematically,

$$error = q^*(s_t, a_t) - q(s_t, a_t) = E[R_{t+1} + \gamma max_{a_{t+1}} q^*(s_{t+1}, a_{t+1})] - q(s_t, a_t)$$

The max term above is calculated by passing $s_{t+1}$ to the DQN again, and selecting the best action. Thus, we need 2 passes.

If we use the same weights for calculating both target $Q$ - value and computed $Q$ - value, we might not be able to converge, this is because, as the we update the weights, the computed $Q$ - value moves a bit closer to the target $Q$ - value, however, since the target $Q$ - value is also calculated using the same weights, it also moves in the same direction as the computed $Q$ - value, hence it is unable to converge. Thus, we use a copy of the network (target network) to calculate the target $Q$ - value, whose weights are updated periodically. This can be done later as per suggestion.

# 3 Algorithm

- Define network.
- Define replay buffer with capacity N.
- Initialize n agents (For now n = 2) (the agents are stored in a list)
- for each episode do
- – Randomly select an agent - $agent_i$
  – $agent_i$ interacts with environment
  – Store experience in buffer
  – Take a uniform random sample from the buffer.
  – Train $agent_i$ using these samples and update the weights.

# References

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs].