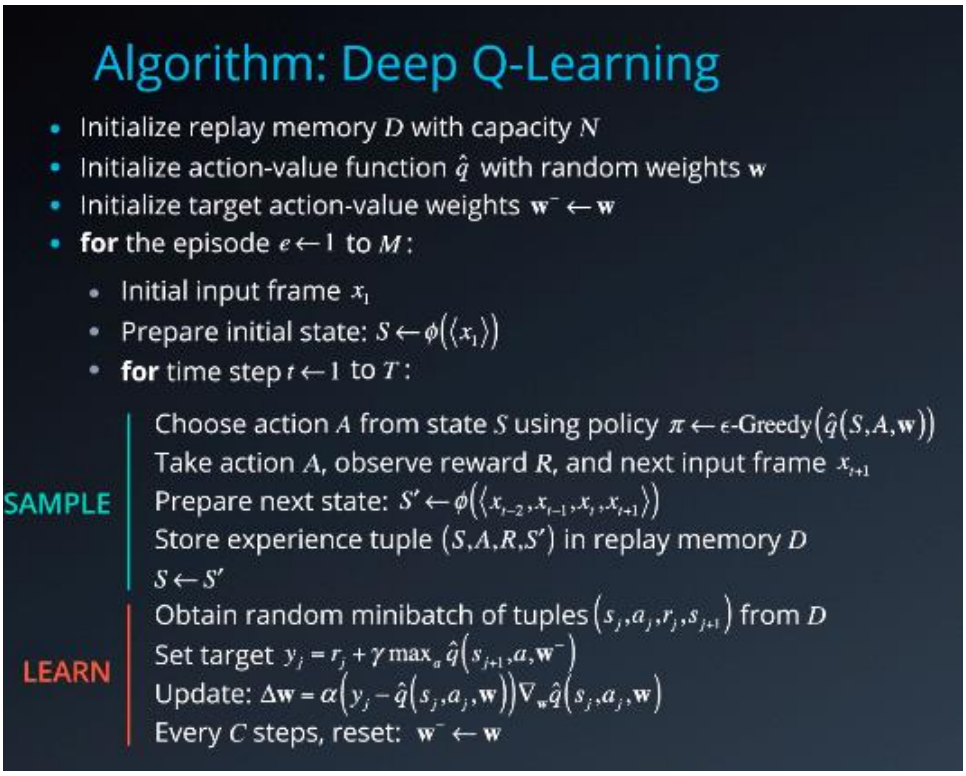
Model: DQN with experience replay. Based on paper: Human-level control through deep reinforcement learning

(https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf).

DQN with experience replay



Source: Udacity https://www.youtube.com/watch?time_continue=180&v=MqTXoCxQ_eY

The DQN algo use a technique known as experience replay in which we store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$, in a data set $D_t = \{e_1, \ldots, e_t\}$, pooled over many episodes (where the end of an episode occurs when a terminal state is reached) into a replay memory. During the inner loop of the algorithm, it apply Q-learning updates, or minibatch updates, to samples of experience, (s,a,r,s') ~ U(D), drawn at random from the pool of stored samples.

This approach has several advantages over standard online Q-learning.

- First, each step of experience is potentially used in many weight updates, which allows for greater data efficiency.
- Second, learning directly from consecutive samples is inefficient, owing to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates.
- Third, when learning on policy the current parameters determine the next data sample that the parameters are trained on. For example, if the maximizing action is to move left then the training samples will be dominated by samples from the left-hand side; if the maximizing action then switches to the right then the training distribution will also switch. It is easy to see how unwanted feedback loops may arise and the parameters could get stuck in a poor local minimum, or even diverge catastrophically. By using experience replay the behaviour distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.

Note that when learning by experience replay, it is necessary to learn off-policy (because our current parameters are different to those used to generate the sample), which motivates the choice of Q-learning.

In practice, the algorithm only stores the last N experience tuples in the replay memory, and samples uniformly at random from D when performing updates. This approach is in some respects limited because the memory buffer does not differentiate important transitions and always overwrites with recent transitions owing to the finite memory size N. Similarly, the uniform sampling gives equal importance to all transitions in the replay memory. A more sophisticated sampling strategy might emphasize transitions from which we can learn the most, similar to prioritized sweeping.

DQN use a separate network for generating the targets y_j in the Q-learning update to improve the stability of the results. More precisely, every C updates it clone the network Q to obtain a target network $\hat{Q}$ and use $\hat{Q}$ for generating the Q-learning targets y_j for the following C updates to Q. This modification makes the algorithm more stable compared to standard online Q-learning, where an update that increases $Q(s_t,a_t)$ often also increases $Q(s_{t+1}, a)$ for all a and hence also increases the target y_j, possibly leading to oscillations or divergence of the policy. Generating the targets using an older set of parameters adds a delay between the time an update to Q is made and the time the update affects the targets y_j , making divergence or oscillations much more unlikely.