Report

Learning Algorithm

The Agent was designed using **Deep Q Networks Algorithm.** In order to successfully learn the environment, the agent uses **Experience Replay** in which the agent learns from the experience it had in the past. Deep Q-learning algorithm with experience replay is better depicted in the following pseudocode.

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**

  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 $\phi_{t+1} = \phi(s_{t+1})$

    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

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

    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

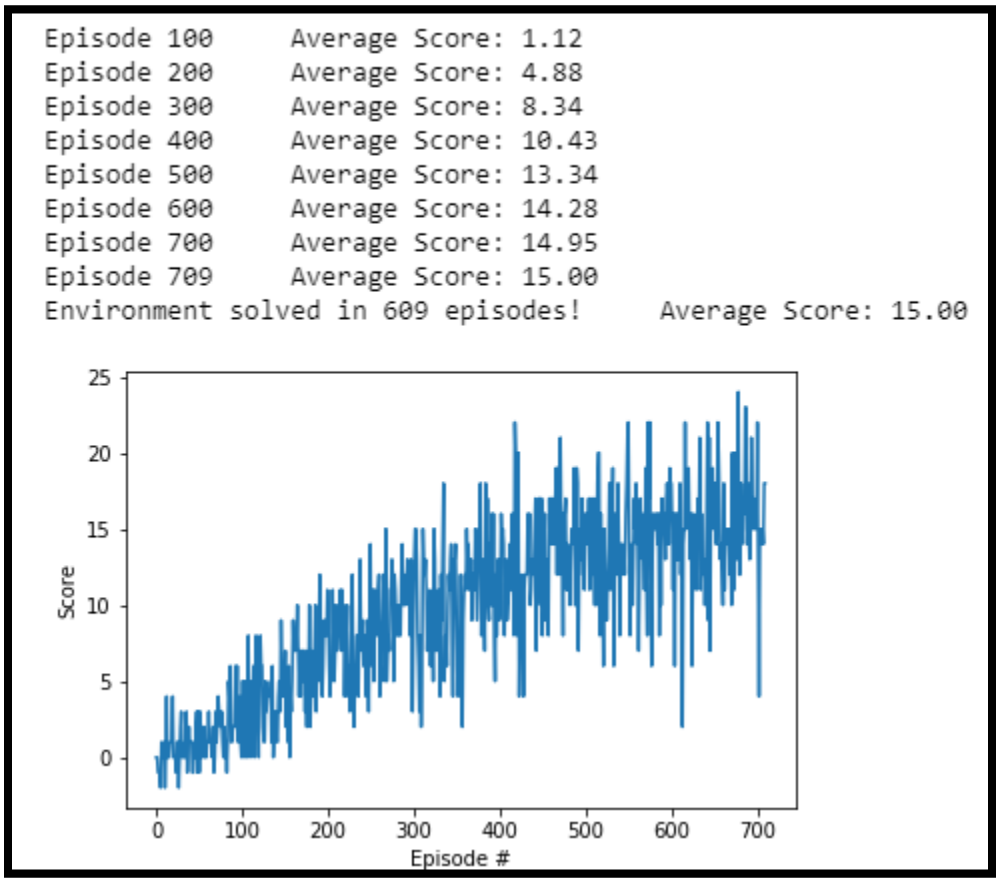
Figure 1 Deep Q-Learning Algorithm. Source: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

The algorithm is initiated by creating a replay memory buffer *D* to store agent's experience. The length of this memory buffer can be specified as hyperparameter *N*. In this algorithm, we use neural networks to store action-value function *Q*. Thus, we also initialize random weights θ for the neural networks. We also need to initialize another neural networks Q' with random weights Θ^{-} which is the clone weight of the previous network. Now we are ready for the training with for M iterations. Note that M is another our hyperparameter here. The algorithm above use images as input. The image is encoded as x. Several images are stacked into s which need to be preprocessed using CNN $\emptyset_1 = \emptyset(s_1)$. In our case, the preprocessed has been done in the environment side. Therefore, we need only to focus on the state embeddings. In each iteration, we let the agent play in the environment with maximum timestep denoted as T which is also our hyperparameter. In every timesteps, the agent picks an action α_t based on Ɛ-greedy policy from *Q* at current environment state s_t. After taking the action, the environment gives a new state s_{t+1} and a reward r_t. The agent then saves the sequence of (s_t, α_t, r_t,

s_{t+1}) in the replay buffer *D*. The agent then samples random minibatch of experience sequence (s_t, α_t, r_t, s_{t+1}) from the replay buffer *D* to calculate the TD error y_j which will be used to perform the gradient descent for the network parameter Θ. In the end of current timestep, the network Q is cloned to target network Q'. The execution is continued until all iterations M are reached.

Training Process

The average score of 13 was reached at 500 iterations as depicted in the chart below:



In the end, the average score of 15.00 was reached at 609 iterations. Further iterations seem to make slight score improvement as the trend begins to plateau since the 500 more iterations.

Hyperparameters

Below is the table of all hyperparameters along with their training values and descriptions.

Hyperparameter	Value	Description
Replay memory size	100000	The capacity of buffer *N*
Minibatch size	64	The length of experience sequence from the replay buffer used for training
Discount factor	0.99	Discount factor gamma used for Q-learning update
tau	0.001	soft update of target parameters

Learning rate	0.0005	The learning rate used for the optimizer
Update frequency	4	how often to update the network
max iterations	1500	The maximum iterations/episodes
Max timesteps	1000	The maximum timesteps
eps_start	1.0	starting value of epsilon, for epsilon-greedy action selection
eps_end	0.01	minimum value of epsilon
eps_decay	0.995	multiplicative factor (per episode) for decreasing epsilon

Future Improvement

Here are some ideas of future improvement:

1. **Double DQN** for better performance
2. **Prioritized Experience Replay** for optimized learning
3. **Dueling DQN** for optimized learning
4. **Error Term Clipping** to between +1 and -1 to improve stability