Report

Agent

The agent implemented in this project uses the approach proposed by the DQN paper along with the two improvements introduced by DeepMind in the Dueling Network Architectures and Double Q-learning papers. The main idea behind all this work is to use *deep neural networks* as non-linear function approximator for the environment in which the agent operates. The goal is to let the network learn a set of weights that can be used to estimate the optimal *Q*(s, a)* action value functions.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \cdots | s_t = s, a_t = a, \pi]$$

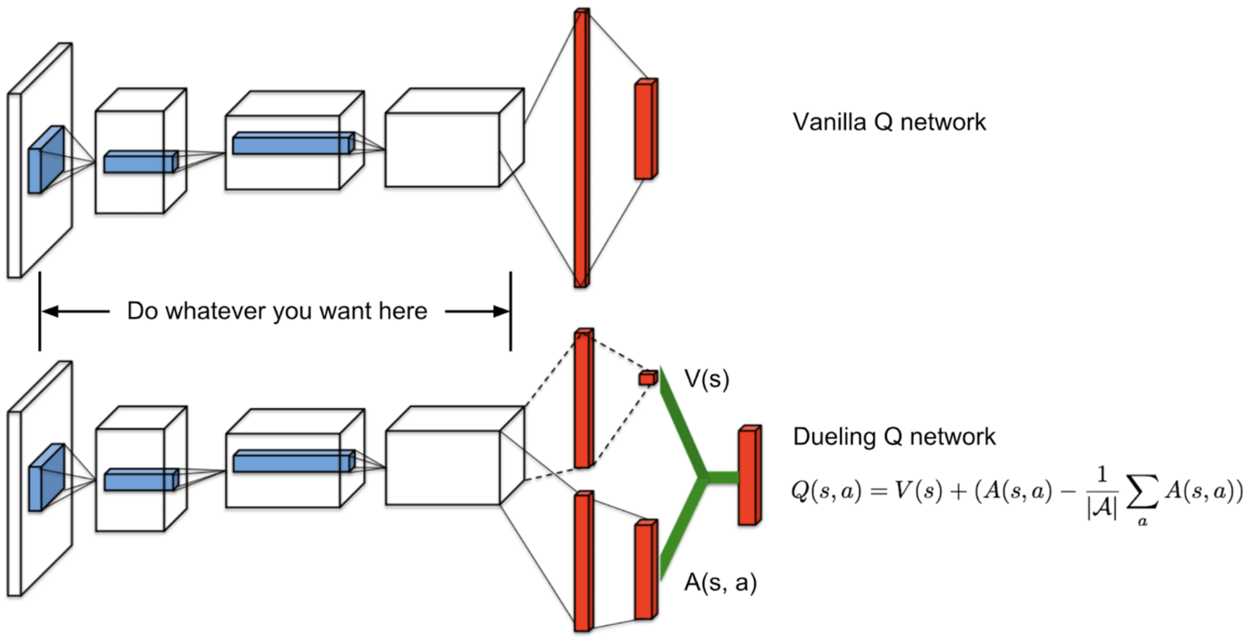
In order to achieve this, we need two components:

- Replay memory: A dataset $D = \{(s_t, a_t, r_{t+1}, s_{t+1})\}$ containing experience tuples that are stored in the exploration phase of the agent and later on used in the learning phase. In this second phase the agent samples randomly for the memory dataset and uses the experience tuples to update its internal weights. This two-phase approach allows to mitigate the correlation effect in the observation sequence.
- Q network: The agent actually has two copies of the same network, namely a local Q_L and target Q_T version. The latter version is updated periodically and is used to provide a fixed target against which to compare the estimated action value function.

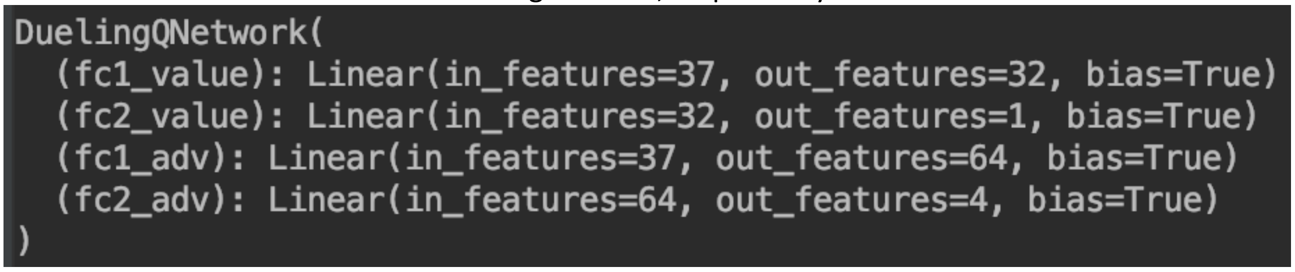
The improvements proposed by DeepMind focus on the Q network. In particular, the difference between the double Q-learning and the vanilla DQN approach is in how they compute the target value y_t of the loss function

Vanilla DQN	**Double DQN**
Using the target network Q_T, from S_{t+1} determine the index of the best action A_{t+1} and its Q-value $y_t = r_t + \gamma \max_a Q_T(S_{t+1}, a')$	Using local network, from S_{t+1} determine the index of the best action A_{t+1} and get the estimated value of the state action pair from the target network. $y_t = r_t + \gamma Q_T(S_{t+1}, \operatorname{argmax}_a Q_L(S_{t+1}, a))$

The dueling approach differs from the DQN method because it separates the computation of the action value function into two estimation branches, one for the state values $V(s)$ and the other for the state-dependent action advantages $A(s,a)$. A clear illustration is provided by the paper itself and is shown below


Vanilla Q network
Do whatever you want here
V(s)
Dueling Q network
$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|\mathcal{A}|}\sum_a A(s,a))$
A(s, a)

In our case the convolution blocks of the network are discarded since we are working with a vectorized representation. Instead we have a simple neural network with only one hidden layer of size 32 and 64 for the value and advantage branch, respectively.



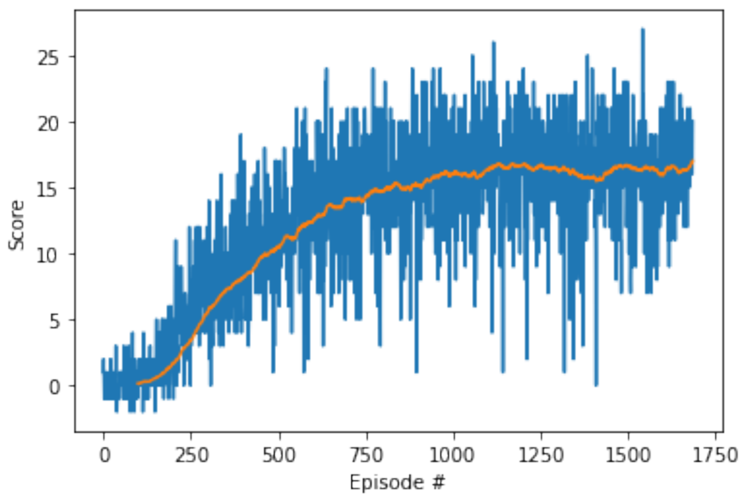
```
DuelingQNetwork(
  (fc1_value): Linear(in_features=37, out_features=32, bias=True)
  (fc2_value): Linear(in_features=32, out_features=1, bias=True)
  (fc1_adv): Linear(in_features=37, out_features=64, bias=True)
  (fc2_adv): Linear(in_features=64, out_features=4, bias=True)
)
```

Results

The agent has been initialized with the following parameters

PARAMETER	**DESCRIPTION**
`BUFFER_SIZE = int(1e5)`	Replay buffer size
`BATCH_SIZE = 128`	Minibatch size
`GAMMA = 0.99`	Discount factor
`TAU = 1e-3`	Discount factor for soft update of target parameters
`LR = 1e-3`	Learning rate
`UPDATE_EVERY = 8`	How often to update the network

Achieving an average score of 17.0 over 100 consecutive episodes after 1585 episodes of training



Future ideas

Despite the quite good performance that the agent achieved the performance, or learning time, can be further improved with the use of a *priority memory.* The idea of such a memory buffer is to sample experience tuple not uniformly. Instead, we should give a higher probability of being selected to those experience tuples that have a higher error. This may speed up the learning process of the agent and allow for faster convergence.