Report

The solution is based on the Deep Q-Network (DQN) approach, which is a variant of the Q-Learning algorithm. It is based on the solution provided as part of the DQN exercise from the Udacity Deep Reinforcement Learning Nanodegree. The source for the base is available on https://github.com/udacity/deep-reinforcement-learning/tree/master/dqn .

The main difference is here that the vector of the environment state is used as input for the learning which is returned after the agent performed one of the available options.

1. Model Description (qnetwork_local):

```
QNetwork (
  (fc1): Linear(in_features=37, out_features=64, bias=True), weights=((
64, 37), (64,)), parameters=2432
  (fc2): Linear(in_features=64, out_features=64, bias=True), weights=((
64, 64), (64,)), parameters=4160
  (fc3): Linear(in_features=64, out_features=4, bias=True), weights=((4
, 64), (4,)), parameters=260
)
```

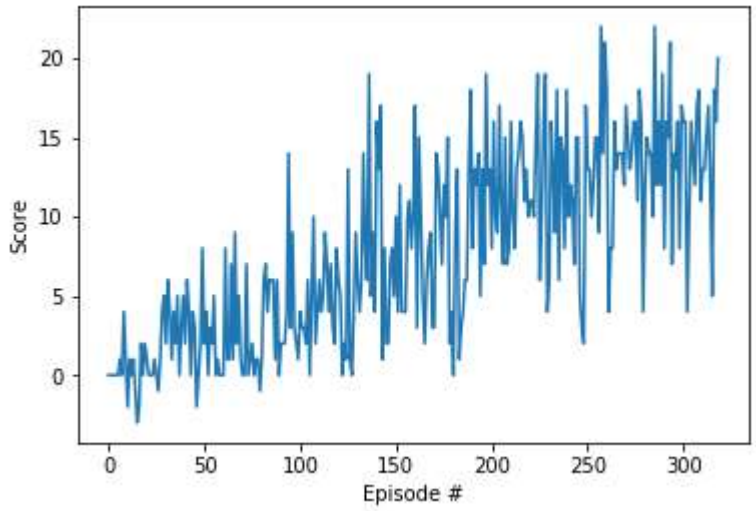
The summary above provides the description of the underlying deep neural network used in the DQN approach as part of this solution.

2. Hyperparameter:

Parameter	Value
n_episodes (int): maximum number of training episodes	**2000**
max_t (int): maximum number of timesteps per episode	**1000**
eps_start (float): starting value of epsilon, for epsilon-greedy action selection	**1.0**
eps_end (float): minimum value of epsilon	**0.01**
eps_decay (float): multiplicative factor (per episode) for decreasing epsilon	**0.95**

3. Training Summary:

```
Episode 100    Average Score: 2.22
Episode 200    Average Score: 7.29
Episode 300    Average Score: 12.61
Episode 319    Average Score: 13.00
Environment solved in 219 episodes!  Average Score: 13.00
```



The plot above shows the agents score achieved during the training episodes. The agent achieved an average score >13.0 after 12 episodes.

4. Ideas for optimizing the solution:

- **Optimize hyperparamters:** During the development of the exercise I experimented with the hyperparameters (especially the decay factor decreasing epsilon. I strongly believe that there is plenty of room of further optimization here with automating the experimentation efforts.
- **Take a different learning approach:** Alternatively to the current approach, learning from pixels is a more 'direct' approach and might lead to better and more accurate results
- **Changing the Algorithm:** As an option Double DQ Networks or Dueling DQ networks could be used to get a solution here. Implementing Prioritized Experience Replay has also potential to improve the results.