README.md

Project 1

Deep Reinforcement Learning Udacity Nanodegree

Task

Navigate unknown environment based on the rewards awarded using deep reinforcement learning.

Input

Vector of size 37.

Rewards

- 0 - no banana was picked up
- 1 - yellow banana was picked up
- -1 - black banana was picked up

Solution

We used code from previous code example as Deep Q-Network agent repo link.

Parameters

- eps - random exploration probability
- eps_step - decrement in eps
- mx - number of maximal training iterations
- replay_buffer_size - this variable was introduced because training in the beginning was very slow due to very few traning rewards. I considered only 10 previous states to be relevant for reward. The replay buffer is consisting of 10 steps before non zero reward.

```
eps = 1.
eps_step = 15
mx = 1800

replay_buffer_size = 10*200
total_scores = []
total_states, total_actions, total_rewards, total_next_states, total_dones = [[], [], [], [], []]
```

- Decrement in exploration parameter eps.

```
    if i <= eps_step:
        eps = 0.6
    elif i <= 2*eps_step:
        eps = 0.5
    elif i <= 3*eps_step:
        eps = 0.4
    elif i <= 4*eps_step:
        eps = 0.3
    elif i <= 5*eps_step:
        eps = 0.2
    elif i <= 6*eps_step:
        eps = 0.1
    elif i <= 7*eps_step:
        eps = 0.05
    else:
        eps = 0.
```

Augmentation of training data

Code below is recording relevant training data when reward was non zero.

```
s_b, a_b, r_b, ns_b, d_b = [[], [], [], [], []]
for s, a, r, ns, d in zip(states, actions, rewards, next_states, dones):
    s_b.append(s)
    a_b.append(a)
    r_b.append(r)
    ns_b.append(ns)
    d_b.append(d)

    if len(s_b) > 10:
        s_b.pop(0)
        a_b.pop(0)
        r_b.pop(0)
        ns_b.pop(0)
        d_b.pop(0)

    if r != 0:
        total_states += s_b
        total_actions += a_b
        total_rewards += r_b
        total_next_states += ns_b
        total_dones += d_b

    if len(total_states) > replay_buffer_size:
        total_states = total_states[(len(total_states) - replay_buffer_size):]
        total_actions = total_actions[(len(total_actions) - replay_buffer_size):]
        total_rewards = total_rewards[(len(total_rewards) - replay_buffer_size):]
        total_next_states = total_next_states[(len(total_next_states) - replay_buffer_size):]
        total_dones = total_dones[(len(total_dones) - replay_buffer_size):]
```

Training

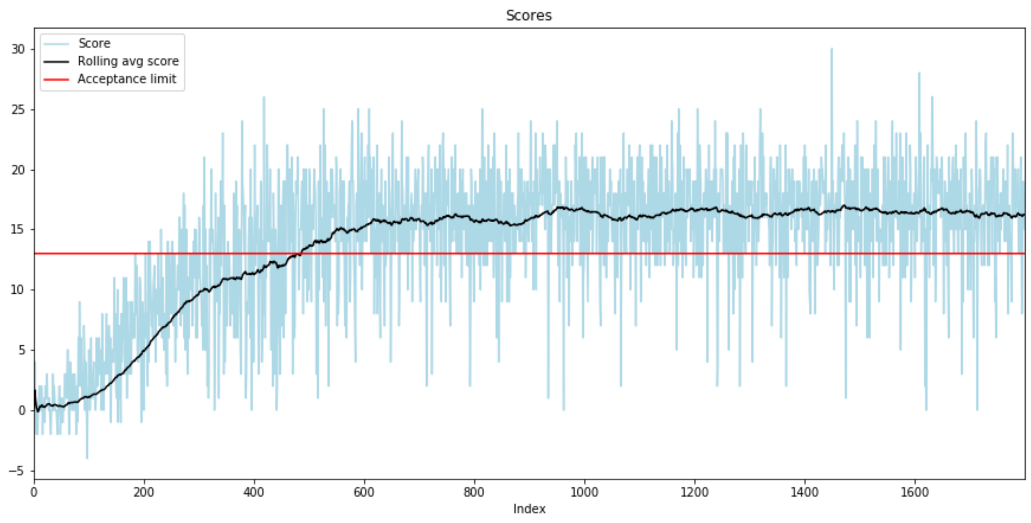
Code below is example of training. We tried multiple training strategies. To reproduce achieved results parameters above might need to be changed and run `python3 train.py` (assuming all of the `requirements.txt` are fullfilled).

- use only latest episode for training
- use only enhanced training data
- use both

```
# use latest episode for training
agent.learn((FloatTensor(states),
             LongTensor(actions),
             FloatTensor(rewards),
             FloatTensor(next_states),
             FloatTensor(dones)),
            (1.-(1./action_size)))
# use enhanced training data
agent.learn((FloatTensor(total_states),
             LongTensor(total_actions),
             FloatTensor(total_rewards),
             FloatTensor(total_next_states),
             FloatTensor(total_dones)),
            (1.-(1./action_size)))
```

We have recorded results below.

- Training with 20 episodes spent on each exploration level. Random action was taken with probability from 0.6 - 0.05.
- During this training we found that rolling average stopped improving shortly after objective was reached (rolling average of 13).

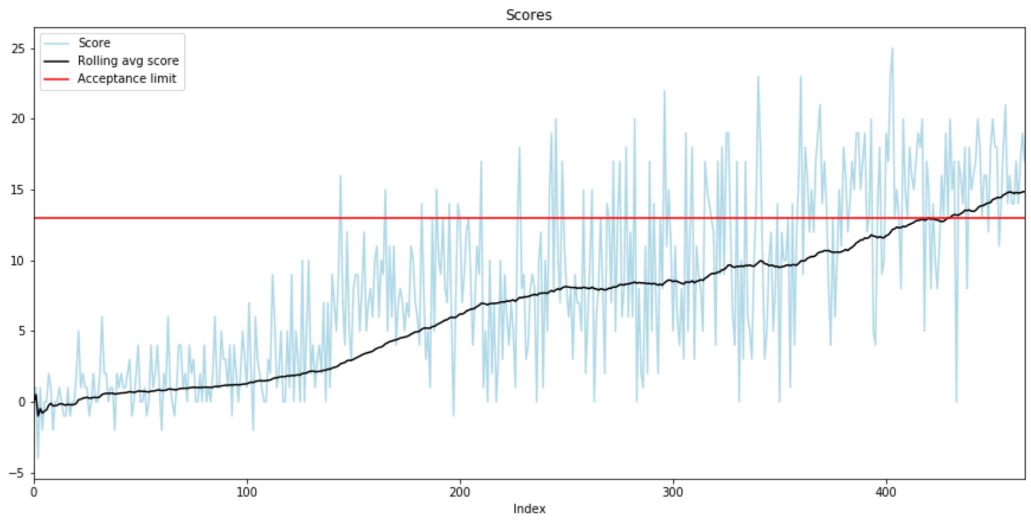


We have introduced early stopping criterion.

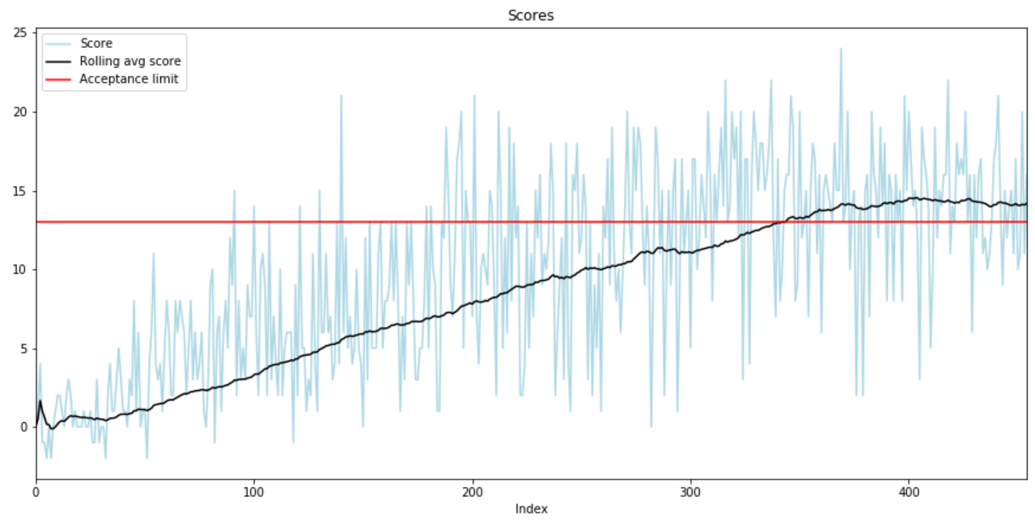
```
if i > 200 and i > stop + 50:  # i > 200 don't consider stopping criteria if still exploring
    print('Training finished no improvements in score recorded in 50 episodes')
    break
```

This criterion will stop training if no improvement in rolling average was recorded in the last 50 episodes.

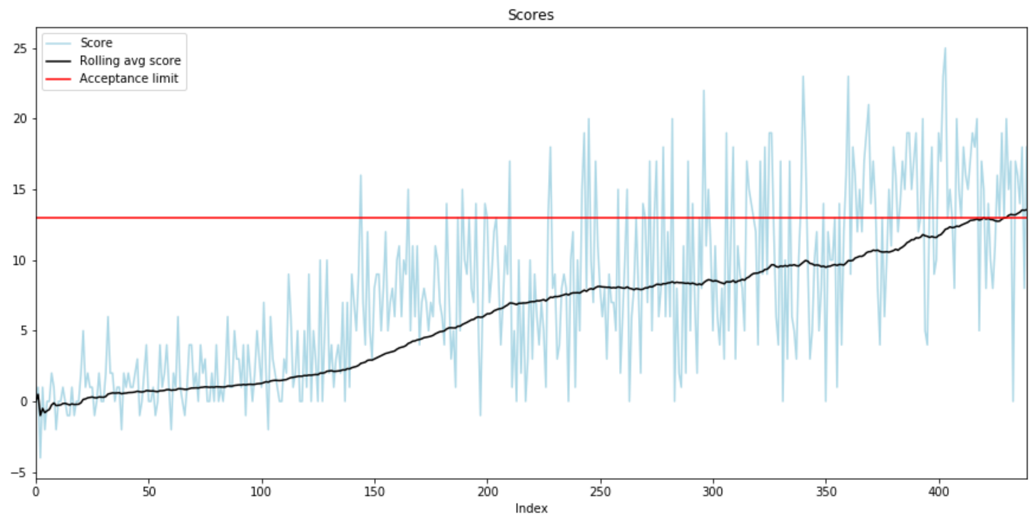
- Training with 15 episodes spent on each exploration level. Random action was taken with probability from 0.6 - 0.05. No data augmentation.



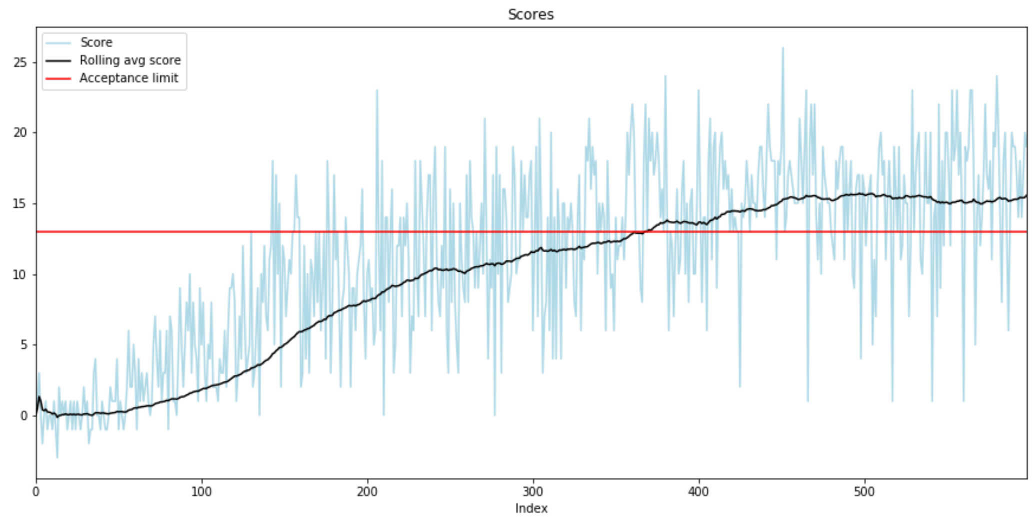
- Training with 5 episodes spent on each exploration level. Random action was taken with probability from 0.6 - 0.05.



- Training with 15 episodes spent on each exploration level. Random action was taken with probability from 0.6 - 0.05.

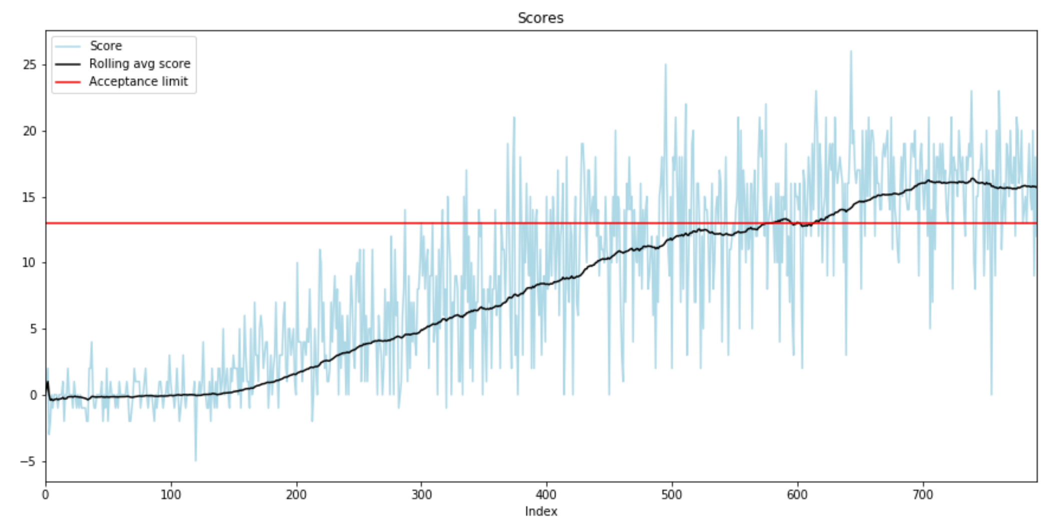


- Training with 20 episodes spent on each exploration level. Random action was taken with probability from 0.6 - 0.05.



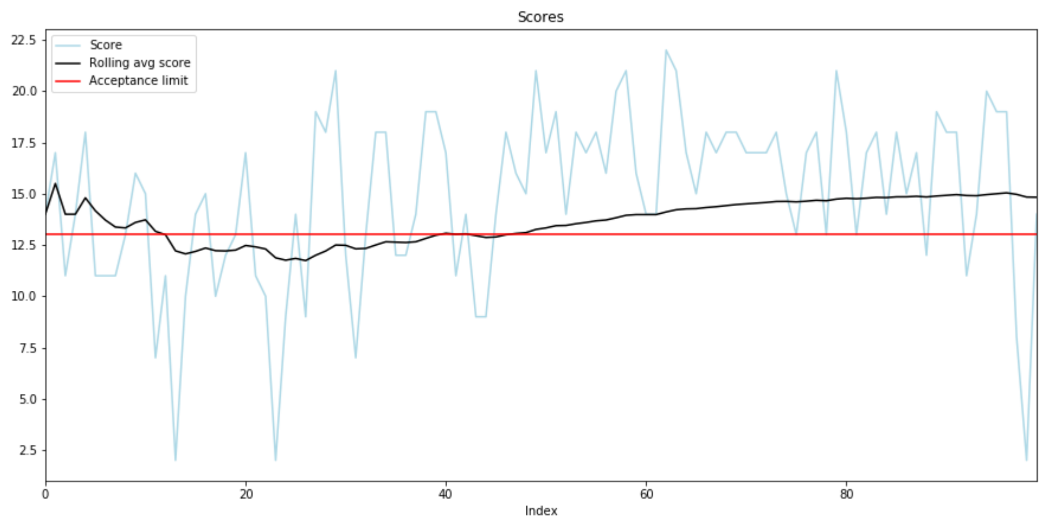
- Training with 20 episodes spent on each exploration level. Random action was taken with probability from 0.6 - 0.05.
- Model was trained on enhanced data each episode data.

- Objective was reached very late in almost 600 episodes



Test

- Test run with model weights loaded from `model/weights_local.torch` and `model/weights_target.torch` .
- To test included model you can run `python test.py` it will generate file data-test.csv with performace data recorded.
- Max rolling average: 15.5
- Max score: 22.0



Conclusions

Objective was reached fastest in about 350 episodes. It was accomplished with the extended training set. Best rolling average was slightly above 17. It was impossible to get above farther above 17. Consistently rolling average was oscilating from 15 to 17 points per episode.

Overall highest score recorded in single episode was 30.