Banana’s Project

This project implemented three approaches to solve Bananas reinforcement problem.

1. Double - Dueling Reinforcement Learning
2. Dueling Reinforcement learning
3. Eligibility Trace with Neural Networks

In all three cases we approximate the q function with a neural network as shown below:

```
    super(QNetwork, self).__init__()
    self.seed = torch.manual_seed(seed)
    self.fc1 = nn.Linear(state_size, fc1_units)
    self.fc2 = nn.Linear(fc1_units, fc2_units)

    #two fully connected layers
    self.fc3_adv = nn.Linear(fc2_units,64)
    self.fc4_val = nn.Linear(fc2_units,64)

    #advantage stream
    self.fc_advantage = nn.Linear(64,action_size)
    #value stream
    self.fc_value = nn.Linear(64,1)


def forward(self, state):
    """Build a network that maps state -> action values."""
    m = nn.ELU()
    x = m(self.fc1(state))
    x = m(self.fc2(x))

    fc_adv = m(self.fc3_adv(x))
    fc_value = m(self.fc4_val(x))

    return self.fc_advantage(fc_adv) + self.fc_value(fc_value)
```

Image 1. Model.py

In this way we use more layers 24 - 32 - 64 and used a nonlinear activation function elastic linear unit finally you can see that we implemented the dueling architecture.

The hyperparameters over those models were:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-3                # learning rate
UPDATE_EVERY = 4        # how often to update the network
LAMBDA =0.7
```

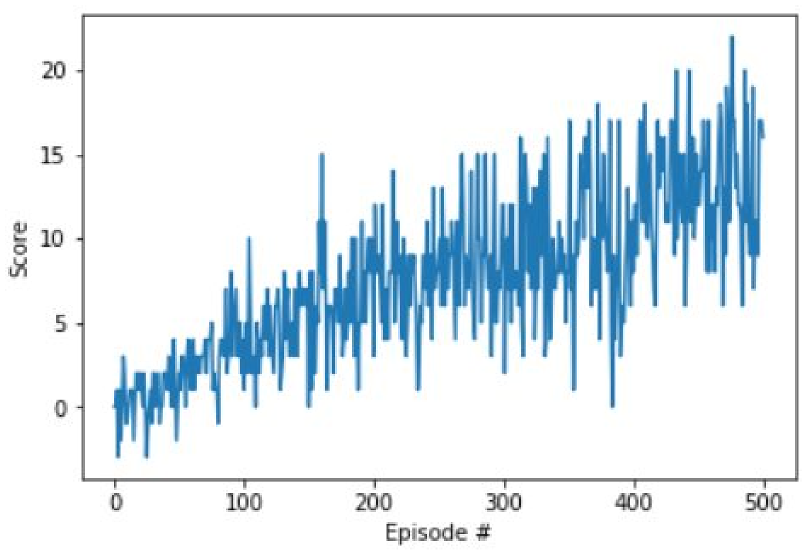
We use the default hyperparameter in first and second case for Eligibility Traces we reduce the size of the buffer replay to 20000 and a lower gamma 0.80.

Results

Double - Dueling Reinforcement Learning

The image below shows that we achieve the desired score of 13.0 in approx. 501 episodes

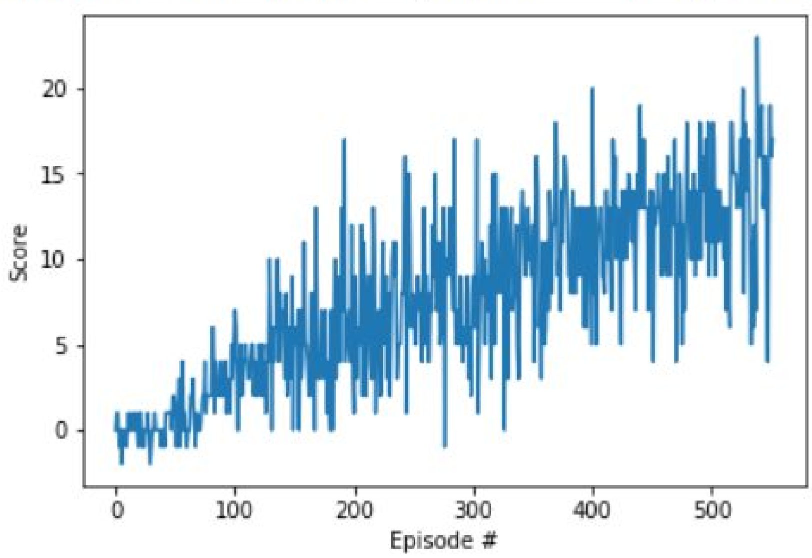
```
Episode 100     Average Score: 1.84
Episode 200     Average Score: 5.49
Episode 300     Average Score: 8.06
Episode 400     Average Score: 9.24
Episode 500     Average Score: 12.96
Episode 501     Average Score: 13.04
Environment solved in 401 episodes!     Average Score: 13.04
```



Dueling Reinforcement learning

There is not such a big difference at least in this case between double-dueling and dueling architectures

```
Episode 100	Average Score: 0.96
Episode 200	Average Score: 4.86
Episode 300	Average Score: 7.30
Episode 400	Average Score: 9.91
Episode 500	Average Score: 12.12
Episode 552	Average Score: 13.04
Environment solved in 452 episodes!	Average Score: 13.04
```



Eligibility Trace with Neural Networks

In this case i implemented Eligibility Trace as shown in https://arxiv.org/pdf/1810.09967.pdf

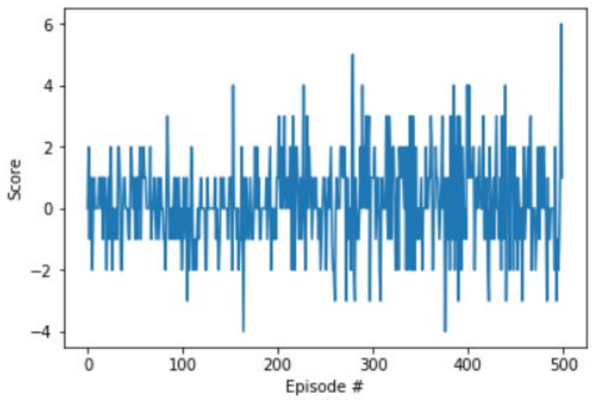
Algorithm 1 DQN(λ)

procedure REFRESH(l)
    for transition $(\hat{s}_k, a_k, r_k, R_k^\lambda, \hat{s}_{k+1}) \in l$ processing back-to-front **do**
        if $\text{terminal}(\hat{s}_{k+1})$ **then**
            Update $R_k^\lambda \leftarrow r_k$
        else
            Get adjacent transition $(\hat{s}_{k+1}, a_{k+1}, r_{k+1}, R_{k+1}^\lambda, \hat{s}_{k+2})$ from l
            Update $R_k^\lambda \leftarrow r_k + \gamma[\lambda R_{k+1}^\lambda + (1-\lambda)\max_{a \in \mathcal{A}} Q(\hat{s}_{k+1}, a)]$
        end if
    end for
end procedure

Initialize replay memory D to capacity N, parameter vector θ randomly
Initialize state $\hat{s}_0 = \phi(o_0)$, episode start $t_{start} = 0$, transition list $L = \emptyset$
for $t \in \{0, ..., T_{max} - 1\}$ **do**
    if $t \equiv 0 \bmod C$ **then** REFRESH(D) **end if**
    Execute action $a_t = \begin{cases} a \sim U(\mathcal{A}) & \text{with probability } \epsilon \\ \text{argmax}_{a \in \mathcal{A}} Q(\hat{s}_t, a; \theta) & \text{otherwise} \end{cases}$
    Receive reward r_t and new observation o_{t+1}
    Approximate state $\hat{s}_{t+1} \leftarrow \phi(o_{t_{start}}, ..., o_{t+1})$
    Append transition $(\hat{s}_t, a_t, r_t, R_t^\lambda, \hat{s}_{t+1})$ to L *// Set R_t^λ arbitrarily – will be updated upon episode termination*
    if $\text{terminal}(\hat{s}_{t+1})$ **then**
        $\hat{s}_{t+1} \leftarrow \phi(o_{t+1})$
        $t_{start} \leftarrow t + 1$
        REFRESH(L); store L in D; $L \leftarrow \emptyset$
    end if
    Sample random minibatch of transitions $(\hat{s}_j, a_j, r_j, R_j^\lambda, \hat{s}_{j+1})$ from D
    Improve Q-function $\theta \leftarrow \theta - \alpha \nabla_\theta \left[R_j^\lambda - Q(\hat{s}_j, a_j; \theta)\right]^2$
end for

```
Episode 100	Average Score: 0.14
Episode 200	Average Score: -0.19
Episode 300	Average Score: 0.44
Episode 400	Average Score: 0.38
Episode 500	Average Score: 0.25

Environment solved in 400 episodes!	Average Score: 0.25
```



But as you can see in the report graph above the results oscillate a lot. a reason for this is the replay memory D and the temporal replay memory interchange the values very often making too difficult to the neural network to learn also the default configuration of replay size is too big for DQN(lambda).

Future Work

Is to implement the same algorithm but using convolutional architecture instead of a NN this could improve our implementation of Eligibility Traces because we will better features to approximate the q function.