In the project , we use a Banana collector environment , the objective was to collect the maximal number of yellow bananas. Avoiding blue bananas.

- For each yellow banana a reward of +1 is provided
- For each blue banana a reward of -1 is provided

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Given this information, the agent must learn how to best select actions. Four discrete actions are available, corresponding to:

- 0 - move forward.
- 1 - move backward.
- 2 - turn left.
- 3 - turn right.

The task is episodic , the objective is to score of +13 over 100 consecutive episodes.

Approach:

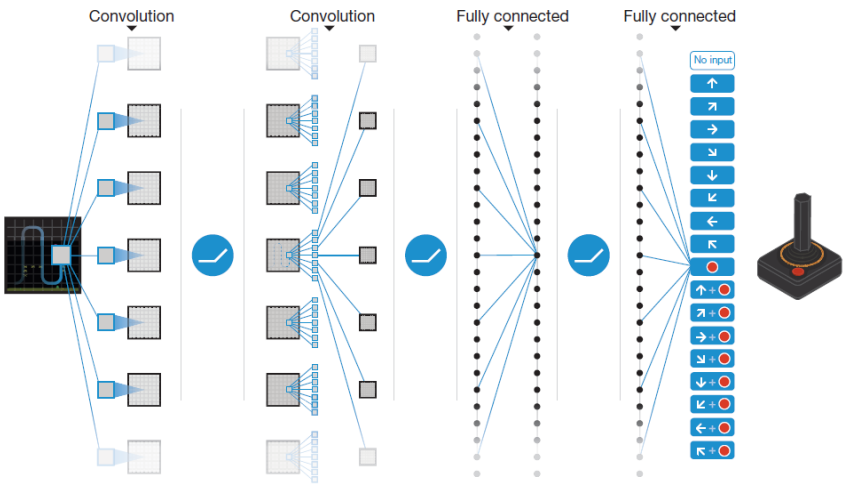
In order to resolve the task, the DQN algorithm was implemented as basis. A second algorith was implemented to improve the learning DDQN.

DQN Algorithm is a multi layer neural network that for a given state s outputs a vector of actions values Q(s,θ), where θ are the parameters of the network. For an n-dimentional state space and an action space containing , actions. This network use experience replay.

DQN.

The provided DQN network was reused in the code.

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-),$$



The architecture consists of three layers:

- 37 neurons in the input layer
- 64 neurons in the first hidden layer, followed by ReLU activation functions
- 64 neurons in the second hidden layer, also followed by ReLU
- 4 neurons in the output layer with no activation function

```
nn.Linear(37, 64)
nn.Relu()
nn.Linear(64, 64)
nn.Relu()
nn.Linear(64, 4)
```

DDQN

As DQN uses the same values to select and to evaluate an action, making the algorithm to be more likely to overestimate values. The Idea of DDQN is to reduce the overestimations by decomposing the may operations in the target into action selection and action evaluation.

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta_i); \theta^-).$$

The DDQN has two identical DQN a local

```
nn.Linear(37, 64)
nn.Relu()
nn.Linear(64, 64)
nn.Relu()
nn.Linear(64, 4)
```

and a target network.

```
nn.Linear(37, 64)
nn.Relu()
nn.Linear(64, 64)
nn.Relu()
nn.Linear(64, 4)
```

double_dqn_targets_q = self.qnetwork_target(next_states) #Q VALUES FOR NEW OBSERVATION

double_dqn_local_q = self.qnetwork_local(states) #Q VALUES FOR THE LOCAL OBSERVATION

best_local_actions= double_dqn_local_q.max(1)[1].unsqueeze(1) #Action with high q value

Q_targets_next = torch.gather(double_dqn_targets, 1, best_local_actions) #select from those Q values computed by target network ,based on action selected by the local

Results:

- The algoritms runs n_repetitions = 10 #repetitions
- 10 random seed to be reused for comparing the two algorithms.
- 600 episodes

DQN

- Episode 506\tAverage Score: 13.05self.double..... False
- Episode 600\tAverage Score: 13.83",
- Finish on 771.166 s

DDQN

- Episode 531\tAverage Score: 13.01self.double..... True
- Episode 600 Average Score: 14.02
- Finish on 808.369 s
- This took more time and the result was lightly improved