Deep Reinforcement Learning – Project 1 : Navigation
Jovan Jovanovic

Goal of the project is to learn agent to collect yellow and avoid blue bananas. Without any prior knowledge he is trying to explore and exploit the environment to learn the rules and master the environment. Solving the environment consist of collecting average score above +13. Since the environment is designed to be MDP problem suits for Reinforcement Learning.

Q learning

The Q-Learning is effective TD method that uses an update rule that attempts to approximate the optimal state - value function at every time step :

$$Q: S \times A \to \mathbb{R}$$

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{(1-\alpha) \cdot Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)$$

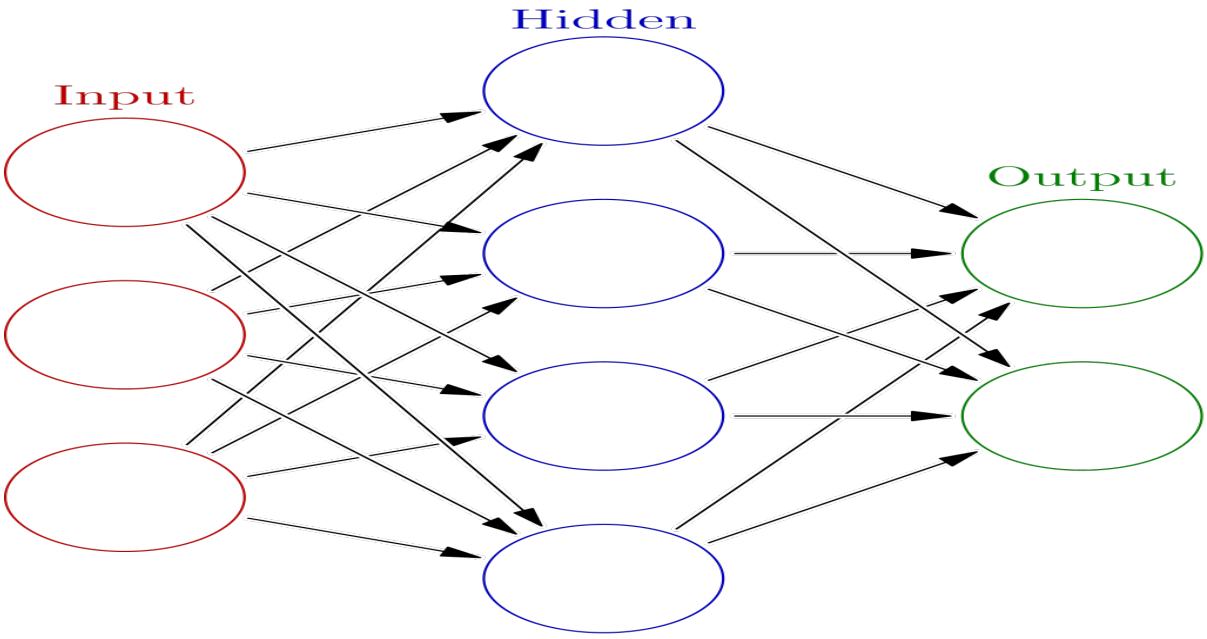
Q learning is very popular RF learning algorithm that can be used when number of states is finite and when we have prior knowledge about various Q values for corresponding states.

Deep Q learning

Since our environment is finite in theory but consist of a lot of states we cannot use table approach. Better fit for this problem is Q function approximation. Reinforcement learning is unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy and the data distribution, and the correlations between Q and the target values.
The technique used *experience replay*, a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent action to proceed.

Neural networks

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules.



Double DQN

The max operator in standard Q-learning and DQN uses the same values both to select and to evaluate an action. This makes it more likely to select overestimated values, resulting in overoptimistic value estimates. In Double Q-learning (van Hasselt 2010), two value functions are learned by assigning experiences. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can untangle the selection and evaluation in Q-learning and rewrite DQN's target as

DQN:

```
target = reward + gamma * dqn_target(next_state).max(dim=1, keepdim=True)[0]
```

DoubleDQN:

```
selected_action = dqn(next_state).argmax(dim=1, keepdim=True)
target = reward + gamma * dqn_target(next_state).gather(1, selected_action)
```

Replay buffer

List of collected (state,action,reward,next state) touples that are used for the training of neural net that approximate Q function.

Prioritized Replay Buffer

Problem with using standard Replay Buffer is that we sample data from there absolutely randomly. When using Prioritized Replay Buffer we can overcome this problem and give priority to the data that is really useful for learning. In other words we are trying to measure the error of particular experience (state, action, reward, next_state) and based on that error give priority to that experience to be selected in learning process.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Here we control prioritization with parameter alpha where in alpha = 0 corresponds to the uniform case. With this way of sampling we introduce bias cause we now have correlation between observations. Therefore we need to correct this by using importance-sampling (IS) weights

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$$

Introducing another hyperparameter beta which controls level of uniformly random sampling.

Noisy Network

NoisyNet is an exploration method that learns perturbations of the network weights to drive exploration. The key insight is that a single change to the weight vector can induce a consistent, and potentially very complex, state-dependent change in policy over multiple time steps.

If linear layer in standard neural network looks like this:

$$y = wx + b,$$

Where x is an input, w is weight and b is bias.

The corresponding noisy linear layer is defined as:

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b,$$

$\mu^w \in R^{q \times p}, \mu^b \in R^q, \sigma^w \in R^{q \times p}$ and $\sigma^b \in R^q$ are learnable and epsilon ones are random values.

Dueling network

The dueling network automatically produces separate estimates of the state value function and advantage function, without any extra supervision. Intuitively, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state. This is particularly useful in states where its actions do not affect the environment in any relevant way.

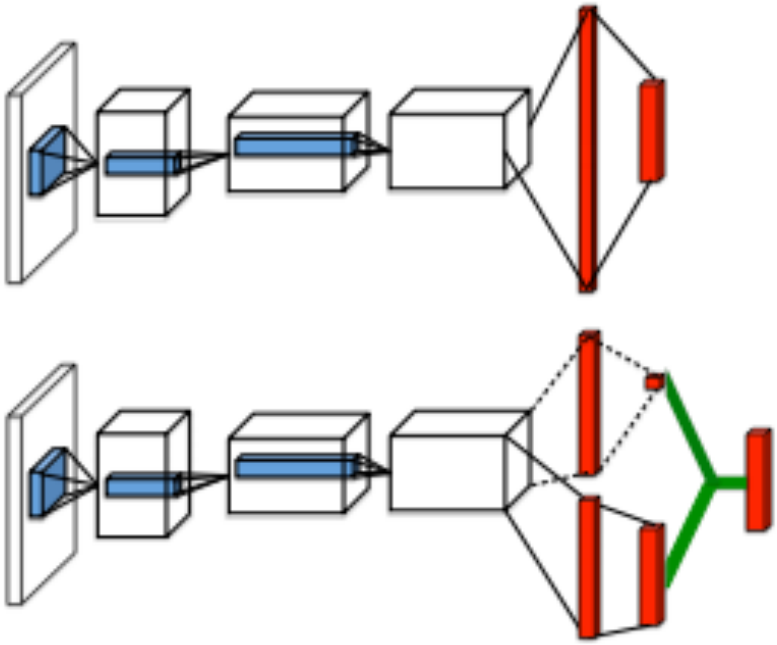


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

Final algorithm

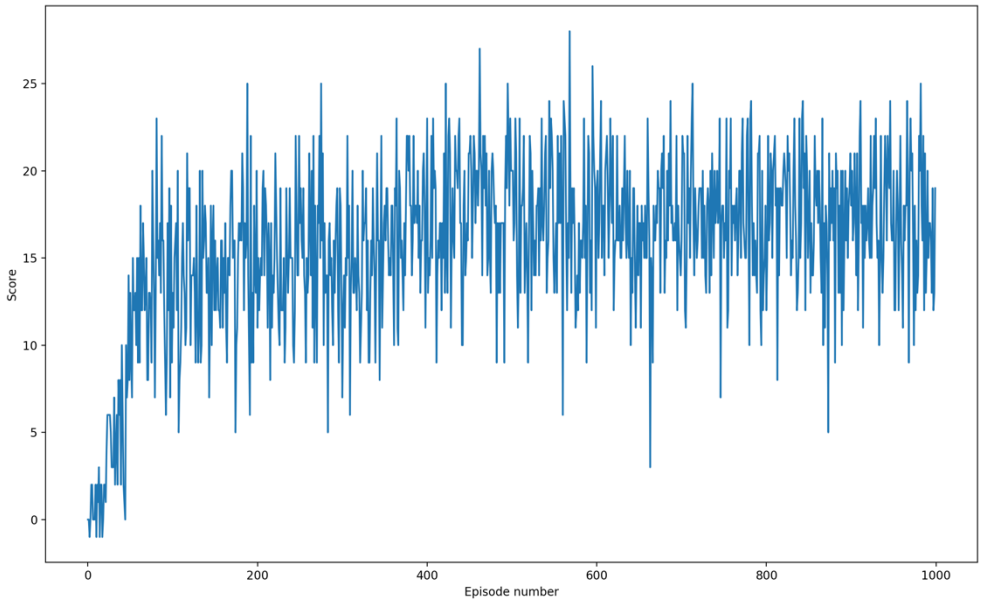
Final solution combines Deep QN with double DQN and noisy networks. When tried with Dueling network training starts to be bit slower and solution converges in 50 episodes more. Here is provided list of hyperparameters:

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-2              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
ALPHA = 0.4   #prioritization level
BETA = 0.6   #importance sampling level
PRIOR_EPS = 1e-6  #guarantees that we don't lose some experiences
```

All hyperparameter values are found in experimental way and it is subject of further work to optimize them. Exploration is controlled through noisy net and therefore additional parameter epsilon is not needed. The environment can be solved in between 140 and 400 episodes and best results goes over 17 after between 500 and 700 episodes.

Best results are achieved with priority experience replay. Environment is solved in only 143 episodes and after 500 episodes score is over 17. Perhaps with tweaking alpha and beta we can get even better results.

```
States have length: 37
Episode 100     Average Score: 8.255
Episode 143     Average Score: 13.00
Environment solved in 143 episodes!     Average Score: 13.00
Episode 200     Average Score: 14.19
Episode 300     Average Score: 14.95
Episode 400     Average Score: 15.64
Episode 500     Average Score: 17.85
Episode 526     Average Score: 17.90
```



For standard replay buffer it's needed around 170 episodes.

Ideas for improvement

Hyperparameter search – probably some grid search can help in finding proper hyperparameters

Rainbow – combining current solution with dueling nets, multi-step learning and categorical dqn can give more improvements