Banana Hunter: A Deep Reinforcement Learning Agent

A report by Carlos Gavidia-Calderon

In this report, we describe in detail the algorithm we used to train our agent. We also described how our agent performed after training and some ideas on how to improve this performance.

Learning algorithm

Our agent relies on the deep Q-network (DQN) algorithm proposed by Mnih et al. from Google Deepmind[1]. A DQN agent has the following characteristics:

1. It relies on multi-layered neural networks
2. In addition to the local network, it has an additional target network, used during Q-value updates.
3. During training, it samples experiences uniformly from a finite replay buffer.

The multi-layered architecture allows the agent to deal with complex state spaces, like the Atari games used in the DQN paper.

During training, the loss is defined as the difference between a local Q-value and a target one. The target Q-value is produced by the target network during training. By keeping the target network separate from the local one, DQN avoids destabilizing training with a constantly shifting target.

By sampling randomly experiences from the replay buffer during training, DQN avoids correlations between subsequent states that can affect the training process.

Hyperparameters

The following table contains the hyperparameters used during training.

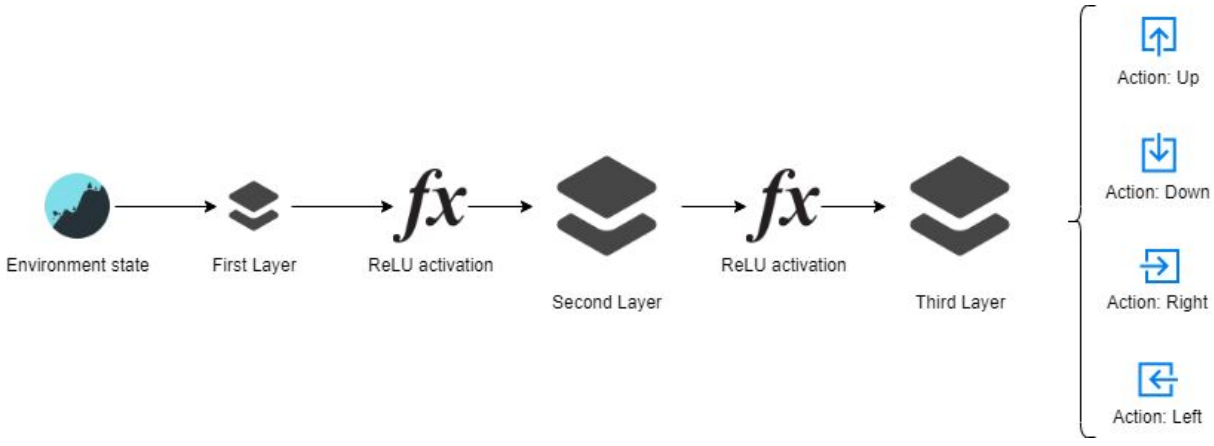
Hyperparameter	Value	Description
Replay buffer size	100000	The maximum number of experiences to store in the replay buffer. When full, newer experiences replace older ones

[1] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.

Learning frequency	4	Our agent does not trigger learning at every step. This parameter defines how often learning is triggered.
Discount factor	0.99	This value represents how much we value future rewards when compared with present ones.
Learning rate	0.0005	This is a parameter of the Adam optimizer. It is used to control how much DQN adjusts the weights of the networks with respect to the gradient of the loss.
Tau	0.001	Periodically, DQN transfers the weights from the local network to the target network. This parameter controls the update process.
Initial epsilon	1.0	Our agent uses an epsilon-greedy policy during training, where epsilon is the probability of choosing an action uniformly at random. This parameter is the value of epsilon at the beginning of training.
Epsilon decay rate	0.995	After each training episode, the value of epsilon decreases according to this parameter.
Minimum epsilon	0.01	This is the minimum value of epsilon during training.

Neural Network Architecture

According to DQN, both the local and the target network have the same architecture, shown in the following figure:



As seen, we rely on three fully connected layers:

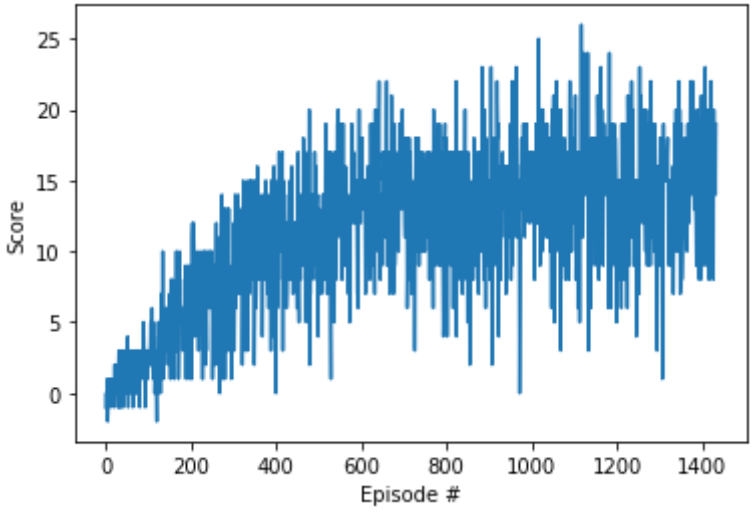
Layer	Input Features	Output Features
First layer	37 (size of the state space)	64

Second layer	64	64
Third layer	64	4 (number of actions)

The first and second layers are followed by a ReLU activation function.

Plot of rewards

The following plot contains the scores obtained by our agent at each training episode. Training stopped when our agent obtained an average score of 15.0 over 100 episodes, which it accomplished at episode 1435 (average score = 15.08).



The following table shows how the average score evolved over training:

Episode	Avg. Score (100 episodes)
100	0.97
200	4.37
300	7.1
400	9.7
500	10.97
600	12.52
700	14.06
800	13.49

900	12.57
1000	14.1
1100	14.7
1200	14.71
1300	13.75
1400	14.47

Ideas for Future Work

We have not explored the hyper-parameter space, relying only on the values used in implementations provided during the module.

The same applies to the network's architecture. The three-layer structure and the number of nodes per layer were also taken from code provided during the modules. Additional layers with a different number of nodes can potentially accelerate convergence.

Finally, we can also incorporate improvements to the classic DQN algorithm. The code can be extended to support Double-DQN[2] and/or Duelling DQN[3], which outperform DQN according to the literature.

[2] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning." Thirtieth AAAI conference on artificial intelligence. 2016.
[3] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." arXiv preprint arXiv:1511.06581 (2015).