UNIVERSITÀ DEGLI STUDI DI PADOVA

Homework #3
Deep Reinforcement Learning

Monaco Saverio - 2012264

Neural Networks and Deep Learning - Professor: A. Testolin

***Abstract*—The final homework concerns Deep-Q Learning, an algorithm of Deep Reinforcement Learning based on *Bellman equation* for decision processes.**
Two environments of OpenAI Gym are tested with the Deep-Q Learning algorithm: "CartPole-v1" where a 2-dimensional platform must balance straight a pole, and a custom environment of a popular mobile game "FlappyBird".
For the latter, two different state-space encodings are probed, a smaller state that employs 2 distances variables, and a bigger encoding that contains the game visual output.

DEEP-Q LEARNING

Deep-Q Learning algorithm belongs to the class of Deep Learning problems of Reinforcement Learning. In this branch of Deep Learning, an agent can interact with the surrounding environment (simulated or real). By the resulting state it is possible to give the agent a reward, namely a parameter that it tries to maximize.
The goal of this algorithm is to find an optimal *policy*, namely a rule that tells what action to make, given the state of the system. The way to determine the *policy function* is not unique, Deep-Q Learning relies on *Bellman equation* with the hypothesis that every instance is a Markov Process.

For both environments, two optimizations are implemented to the vanilla algorithm:

- *Experience Replay:* The agent does not learn while it acts on the environment. It follows the last policy for an arbitrary number of episodes, then it updates its policy sampling from its past [state, action] pairs. This helps the convergence of the algorithm by having a more i.i.d like data samples.
- *Target Network:* Two similar networks are implemented for a more stable update rule

A final note about Deep-Q Learning theory regards the *Exploitation vs. Exploration* problem, namely wether to choose to follow the current policy to get the most reward, or to perform non-optimal actions according to the current policy in order for the agent to improve its knowledge about actions with more logn term benefits. For the training of all the following models, a parameter called "T" (temperature) was implemented, the higher T is, the more exploration is performed. This parameter is gradually decreased using an exponential rule.

I. CARTPOLE-V1

The first environment taken into consideration is "CartPole-v1". It consists in a cart that can move on a 1-dimensional rail. At the centre of the cart, a pole is pinned to one of its end. The goal is to avoid the pole to fall by moving the cart left or right along the rail.

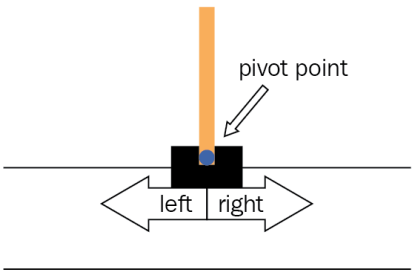

Fig. 1: CartPole-v1 graphical representation

A. *Methods*

For the CartPole environment the dimension of the observation space, namely the dimension

of the state-vector that completely defines every state, is 4-dimensional[2]:

- Cart x-position: range = [-4.8, +4.8]
- Cart x-velocity: range = [-∞,+∞]
- Pole Angle: range ∼ [-24°, +24°]
- Pole Angular Velocity: range = [-∞,+∞]

The Cart can only either move left or right, the action space is then 2 dimensional.

The Policy and Target network are simple linear Networks:

- *Input Layer*
  - input_size = 4 (state space dimension);
  - output_size = 128;
- *Hidden Layer*
  - input_size = 128
  - output_size = 128;
- *Output Layer*
  - input_size = 128
  - output_size = 2 (action space dimension);

Between each layer, `nn.Tanh()` activation function was chosen.

The goal was to find an optimal model that required the least amount of episodes as possible to learn how to balance the pole. For this reason, during the training of each model, after every update of the weights, the network was tested on 10 games with 0 temperature, if it could balance the pole until the end (500 frames), the model was said to have learned.
Various models were tested with different hyperparameters using Optuna, in the search of the model that could ace the 10 games with the least amount of learning as possible.

Parameters for Optuna Optimization:

Trials:	30
Epochs:	1000
Regularization:	[1e-5, 1e-3]
Lr:	[1e-4, 1e-3]
Optimizer:	Adam, SGD
Gamma:	[.95, 1]
Initial T:	[1, 10]

B. *Results*

Best parameters:

Regularization:	0.0002
Lr:	0.0003
Optimizer:	Adam
Gamma:	0.976
Initial T:	2.107

Learned in 500 episodes

II. FlappyBird

Flappy Bird was a popular game for smartphone devices in the year 2014. The player must control the height of a bird by tapping on the screen, resulting in a flutter of its wing, while the bird's x-position is always fixed.
The game consists in avoiding incoming obstacle, namely vertical pipes that have a hole at a random height, in order to keep playing, the player must tap the screen accordingly to enter each pipe's hole, never touching the obstacles.

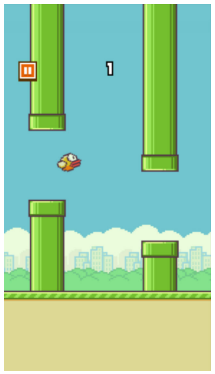

Fig. 2: Gameplay example of the game

A. *FlappyBird-v0*

*1) **Methods**:* The state space are the two parameters of this environment, namely:

- horizontal distance to the next pipe:
  range = [-∞,+∞]
- difference between the player's y position and the next hole's y position:
  range = [-∞,+∞]

While the possible actions are just two:

- Fly upwards (originally the tap of the screen): `action value = 1`

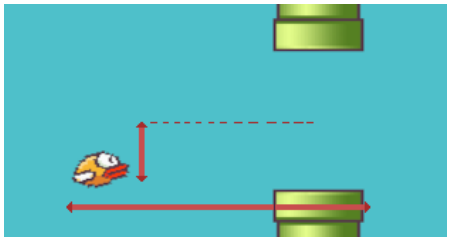

Fig. 3: State space of FlappyBird-v0 environment

- Do nothing `action value = 0`

Regarding the reward and punishes, the agent takes a point for every frame the game has not ended, and it gets a punishment proportionally to the vertical distance of the next hole. This punishment rule was mainly implemented to guide the agent for the first stages of learning, since without it would always fly way up (if the policy is mostly random, the actions are random and equiprobable aswell, however, fluttering the wings affects the y-position way more than doing nothing).

For this environment, the very same Policy and Target Networks class used for `CartPole-v1` were adopted.

Parameters for Optuna Optimization:

Trials:	10
Epochs:	1000
Lr:	[1e-4, 1e-3]
Optimizer:	Adam, SGD
Gamma:	[.95, 1]
Initial T:	[3, 6]

The best model was chosen by letting the agent play 10 times at the end of its training and choose the model with the highest mean score.

*2) **Results**:*

Best parameters:

Lr:	0.0004
Optimizer:	Adam
Gamma:	0.96
Initial T:	5.34

Mean Score on Test Games: 2490

The model seems to be applied successfully in this environment aswell, however, the agent still loses each game eventually when sharp turns must be performed taking into account not only the incoming pipe, but also the following one. However in this environment, the latter information is totally missing and a more extensive learning will not probably result in significative improvements of its executions. Nonetheless it has achieved a performance way above the average human player by usually surpassing the 70th-pipe.

B. *FlappyBird-rgb-v0*

The very same task was performed with the visual game output as state space.
Ideally, this new type of observation space could surpass the lack of information exposed for the environment before.

*1) **Methods**:* State space was reduced through channel selections, maxpooling and cropping due to the long times required during the learning phase. See Appendix C for the state space optimization process.
State space is:

- 14×40 grayscale image

Possible actions are the same as before:

- Fly upwards (originally the tap of the screen): `action value = 1`
- Do nothing `action value = 0`

For the type of state the space data, a convolutional network for both the policy and target network was considered to be more fitting:

- *First convolutional layer*
  - in_channels = 1;
  - out_channels = 8;
  - kernel_size = 5;
  - stride = 2;
  - padding = 1;
- *Second convolutional layer*
  - in_channels = 8;
  - out_channels = 16;
  - kernel_size = 5;
  - stride = 2;
  - padding = 1;
- *First linear layer*
  - input_size = 288
  - output_size = 64
- *Second linear layer*
  - input_size = 64
  - output_size = 2 (action space dimension)

Between each layer, `nn.Tanh()` activation function was chosen.

Due to the longer training times, the hyperparameters optimization was limited to just 5 trials:

Parameters for Optuna Optimization:

Trials:	5
Epochs:	1600
Lr:	[1e-4, 1e-3]
Optimizer:	Adam, SGD
Gamma:	[.95, 1]
Initial T:	[3, 6]

*2) **Results**:*

Best parameters:

Lr:	???
Optimizer:	???
Gamma:	???
Initial T:	???

Mean Score on Test Games: ???

REFERENCES

[1] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[2] "GitHub CartPole-v1 Documentation". In: (). URL: https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py#L75.

[3] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

III. Appendix

A. *CartPole-v1 hyperparameter optimization*

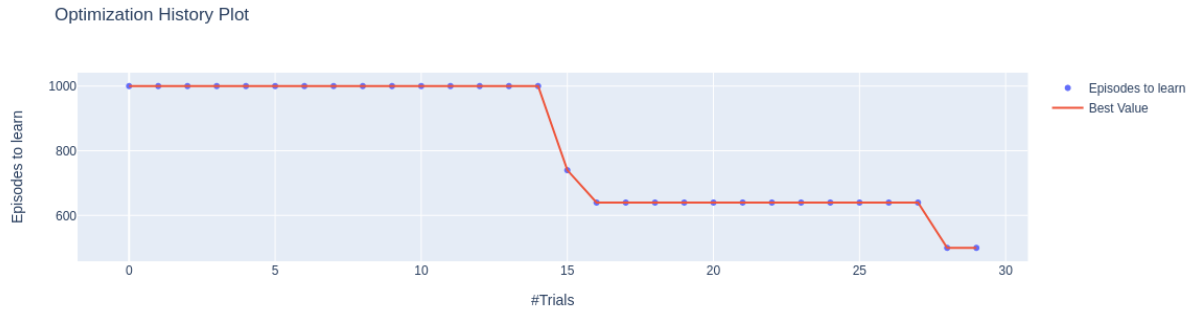

Fig. 4: Hyperparameter optimization of CartPoe-v1 environment in Optuna

During the training of each model, every 10 episodes their policies were tested in 10 games with T=0. The best model was chosen by finding the one that takes the least amount of episodes to score 500 on all the 10 games with zero temperature.

The search of the best parameters was further optimized by interrupting the learning of all the models that were taking longer, with respect to the best model found to date, to reach the above requirement, that is the reason why the curve of the figure 4 is not-increasing.

B. *FlappyBird-v0 hyperparameters optimization*

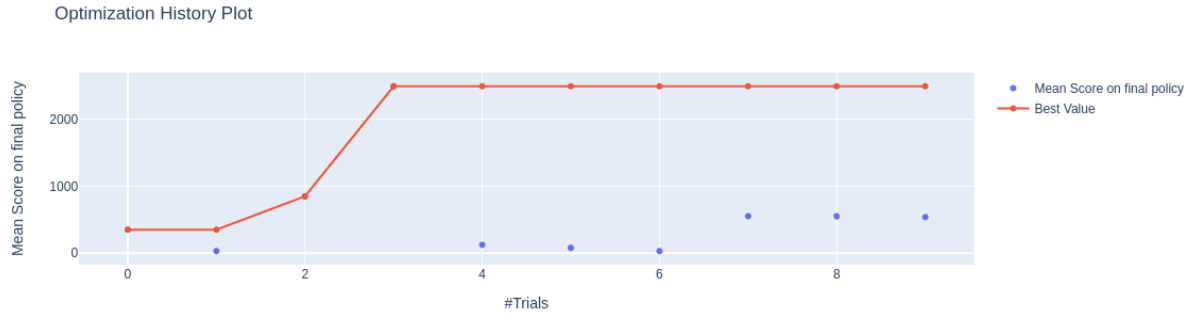

Fig. 5: Final Mean Scores of FlappyBird-v0 models

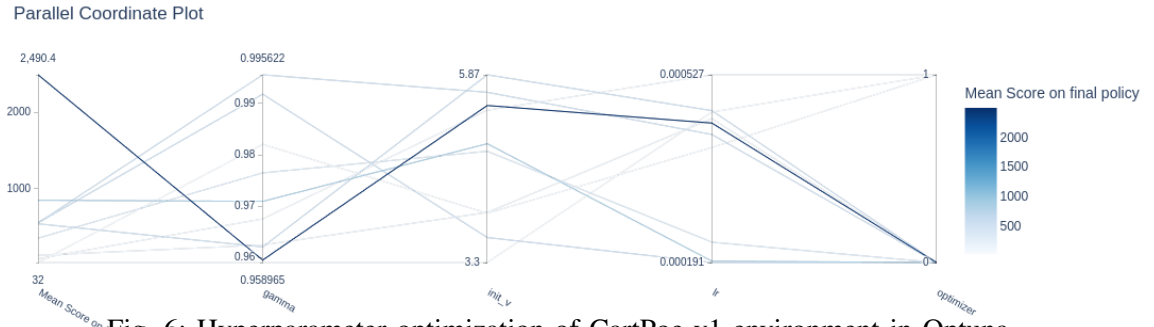

Fig. 6: Hyperparameter optimization of CartPoe-v1 environment in Optuna

C. *Coordinate Plot of FlappyBird-v0 hyperparameters*

For the environment FlappyBird-rgb-v0, the original state space corresponds to the game visual output with a screen size of 288×512, the dimension of the state space is then 288×512×3, that is 442,368.

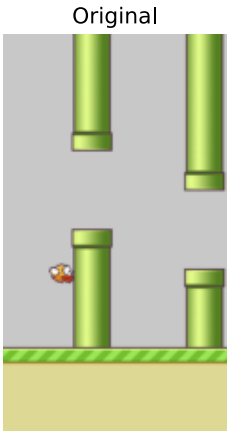

Fig. 7: Original example of State space of FlappyBird-rgb-v0 environment

For curiosity, this encoding of the state space was tested, resulting in several seconds for every time the function `update_step()` was called. Considering the large amount of time it would take to train the model enough epochs with this state space encoding, the process was killed and optimizations were considered.

1) Single channel image: The first obvious optimization is to choose a single channel from the whole RGB image, reducing the size of the state space by a factor of 3. Additionally, there are two main areas that contains no visual information, namely the bottom area where the ground is, and the area left to the agent, hence the screen was cropped removing them.
This reduced the size of the state space from 442,368 to 84,000.

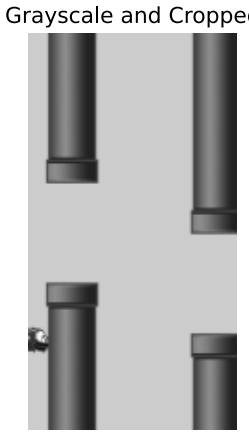

Fig. 8: Single channel (Green) and Cropped

2) Lowering resolution: The major advandage was obtained by maxpooling the state image, lowering its resolutiong. The maximum kernel size (10) was applied to obtain the highest gain in training time, bringing the dimension of the state space from 84,000 down to 840.

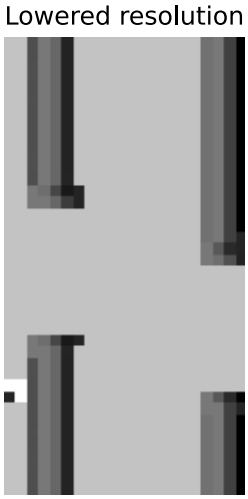

Fig. 9: Maxpooled (10)

3) Further pooling of right side of the screen: With the idea that the attention of the agent should be channeled mostly to the incoming obstacle, the right side of the screen was maxpooled horizontally, merging horizontally the pixels among the ones in the blue area and in the red area of the figure, further lowering the state space dimensionfrom 840 to 560.

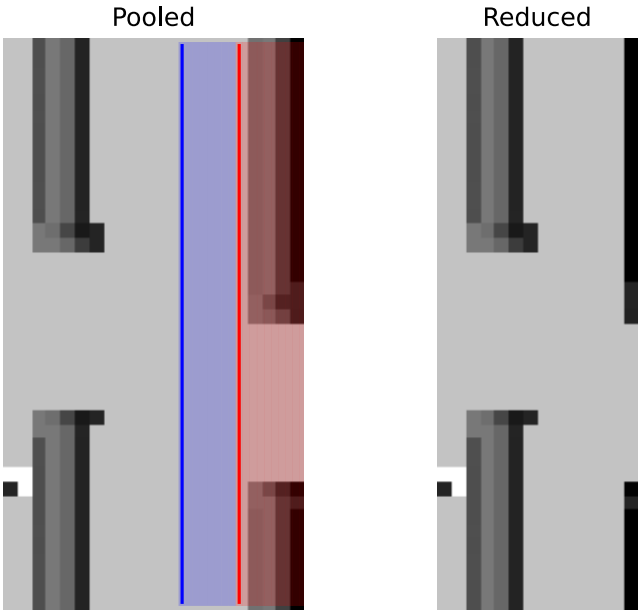

Fig. 10: Horizontal pooling of right (distant) pixels

In the following figure the original and final state space encodings are shown for a direct comparison. All the size reduction of the state space caused a loss of information that can cause the model to converge slower (or not at all), however, it seems that all the essential information are still kept in the reduced encoding.

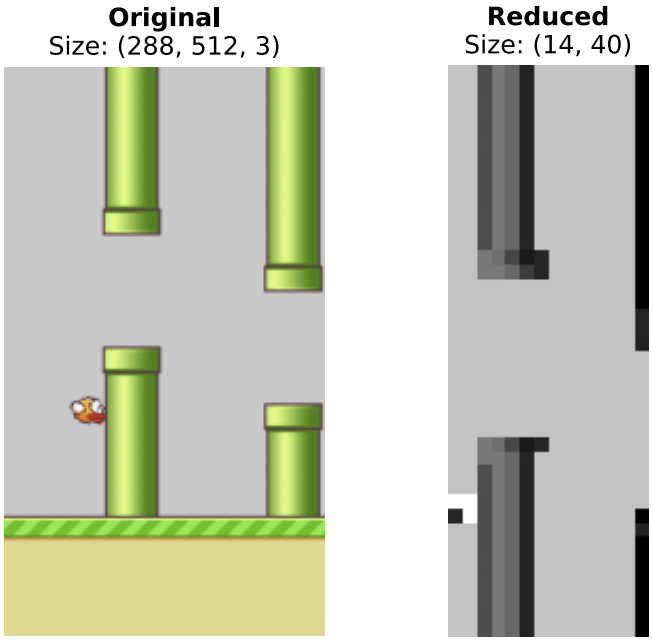

Fig. 11: Full state-space transformation

Total improvement: 442,368 $\rightarrow$ 560