Report for Reinforcement Learning Project : Pac-Man

Mario LARSEN

***Abstract*— Make an abstract that summarises the report. Keep in mind that your report should not exceed 4 pages (without appendix and references) with a few figures in it. You don't need to write 4 pages to have a good report but to ask the good questions about your agent. You can also add interesting figures in the appendix if you need to. If you have used any source, use the references to add them.**

I. Presentation and Choice of the game

A. *Presentation of the game*

Pac-Man is one of the oldest video game. The player have to evolve inside a maze without being catch by four ghosts. The player win when he visited every places at least once (represented with pacgums to eat) before the end of the timer. Every ghosts have a different approach to follow Pac-Man. The ghosts artificial intelligence are very simple but efficient. The variety of the assaults by those bots make the game pretty hard (in particular without the super-pacgums power). Wining shows the player superiority to the AI. I found interesting to make a neural network reaching this goal. The distant goal with this project is to train the ghosts and Pac-Man at the same time to see which role is advantaged.

B. *Existing environment*

In the first place, a version of this game exist for Atari. It is Ms. Pac-Man. This version exist as a gym environment. In the one hand, is a good approach to begin in particular to test the implementation of the agent. In the other hand, the game is ugly, has a huge state space and need a deep neural network to master. My computer can't dell with it.

I found various emulation for python online. One is a project by Berkeley university to learn reinforcement learning. I try to use it but the program is to be completed, is full of bugs due to the age of the project and is way to complex for me. An other one is made by freegames, use turtle and is only 70 lines long. I use this one as a canvas for my own environment.

C. *My environment*

I develop 12 version of the environment. Some are adapted for human player and other for openai training. For the last version, I choose a small map, no super-pacgum effects, one to four random ghosts , multiples life (one by default). The state space is a 3D-matrix with the size of the map (11x11). The value of the matrix are length 4 vectors representing the presence (with a one instead of a zero) of respectively, wall, pacgums, Pac-Man and ghosts. The shape is then (11, 11, 4). The actions are the 4 aims gives to Pac-Man. Other model suggest having only to turn right or left work better.

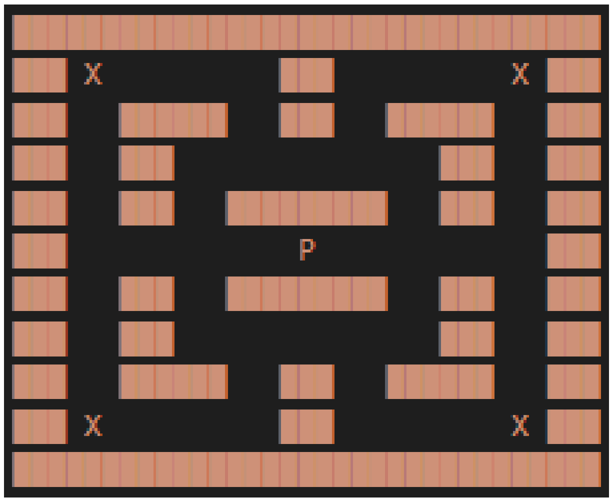

Fig. 1. The maze map

Every step, we move Pac-Man according to its aim (action). If he can't, he is punish (by a negative reward of 10). The ghosts moves randomly without getting back or stopping. If one catch Pan-Man, the game end and the reward if fix to -100. Meanwhile, when Pac-Man eat a pacgums, the score increase by one and the reward is fix to the current score. This mean that the lasts pacgums worth more than the firsts. If he eat them all, the reward gain a additional 100 and the game end. The last way to end the game is a timer fix at 150 step. Knowing there is 51 pacgums, this give plenty of time to finish the game.

II. Implementation of your agent

A. *Agent design*

I based my design on the Deep Q Network Reinforcement Learning model given as an example. Q learning links state of the environment and action given to reward of those action in those state. We than use those links to predicts what action to take to maximise the reward. Since Pac-Man present a wild variety of states, we can't list them all and calculate all possibles outcome for all possible combination of actions. The calculation of the quality of each action given a state is predict by a neural network. The quality name Q-value is related to the reward of this action and the future rewards predicted.

We play the game. At each state, the network predict a Q-value for each possible action. The highest quality one is taken. The environment, make a step return the reward and the next state. We do this in a loop to gather data and see how much reward our DQN can get.

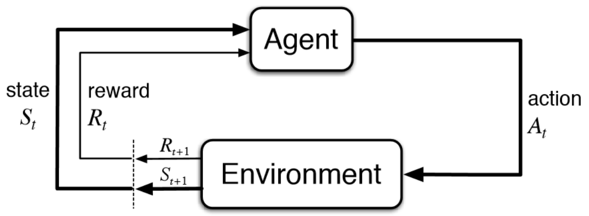

Fig. 2. Main loop of Q learning

To train our network, we store related states, actions, rewards, next states and a Boolean true if the game end. the network predict the Q-value related to those states and actions. We compute the actual Q-value is given by the reward if it the end of the game. If not we add the Q-value of the next state, time a constant call gamma. Gamma is chosen between 0 to 1 to express the importance of long term strategy. This next Q-value is predicted by a frozen copy of the Train-Net call Target-Net. The loss is the distance between actual and predicted Q-values. With it, we apply gradient descent on our network variables.

The optimizer Adam, his lr at 0.01, the 100 copy step of Train-Net to Target-Net and the batch size of 32 were already implemented and they are not relevant to change for the moment.

B. Personnal choices

At the stage I am, I want my Network to learn to walk without hi ting the wall or the ghosts. I choose to reduce the gamma to a little 0.2 because long term does not really matter. Since the stage do not vary over the game, I need some action to be random for discovering new states and rewards. I choose an epsilon and a decay proportional to the number of iteration I am ready to compute. The probability of choosing a random action is given by epsilon time decay power iteration number. This way it decline over time from close to 1 to a minimum set close to 0. I set a number maximal of experiences to keep in memory while training to 10000 and a minimum to 1000.

C. Neural Network

For the design of the neural network I could just let lots of longs dense layers do the jobs but it is able to teach Pac-Man how to navigate. I had try to implement a stack of convolutions 2D layers before some dense ones but I realize how much parameter it was. I particular, I can't use max pooling to reduce the dimension because of the index importance in this game.

I decided to think of the simplest solution I could build myself to know how to navigate with those data. There are 4 position of wall, ghosts and pacgum around Pac-Man to take care of to evolve without trouble. This mean 16 case represented by a 2x2 matrix.

I chose to build the network with a 2D convectional layer with a 3x3 mask. The activation was a ReLU and the depth fix to 32. I was not hoping, it find the optimal solution so I add extra freedom parameters. Before the dense output layer activated by softmax, I placed a global max pooling (and a flatten) to ensure that if Pac-Man index is found, nothing else mater. With only 1312 weights to choose, I may just have to train my network on some thousand games to find a stable solution.

III. Evaluation of your agent

I use tree way to see how my agent is doing. I compute the average loss and rewards and make a graph out of them. The last one is to see by myself a game play by the agent. The firsts tests where not conclusive but it was more due to coding flows than the actual model.

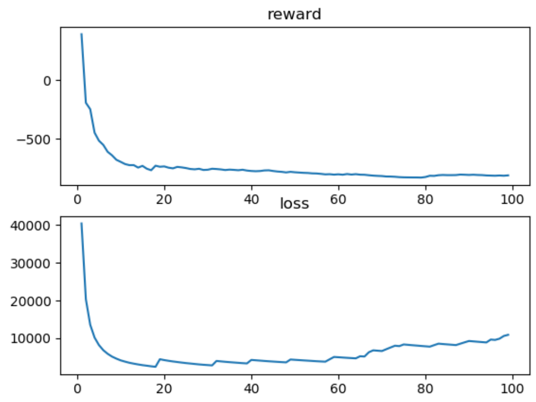

Fig. 3. Trouble child

The seconds were conducted on a classical Convolutional + Dense. If the loss was wonderfully declining, it hide lots of problems in the implementation. The reward is a good illustration of this issue. The execution was so slow, it tooks one full night and day to get this result.

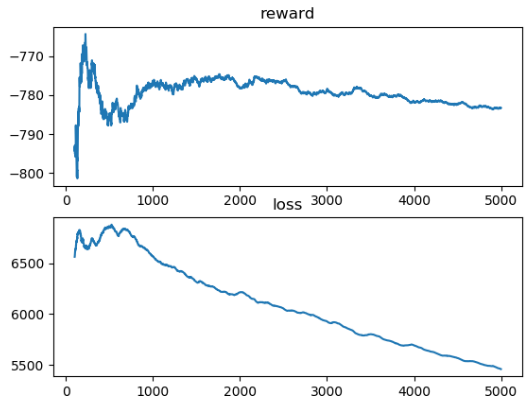

Fig. 4. 2 Conv2D 3 Dense

After seeing those results, I decide once again to fully remodel the agent and the environment. The latest test was conducted with the model detail in the previous section. With only one ghost, the results are now coming in real time. I'm happy to see the reward increasing with the epsilon decreasing but the loss keep being a little high and does not seem to move. There might be some issue.

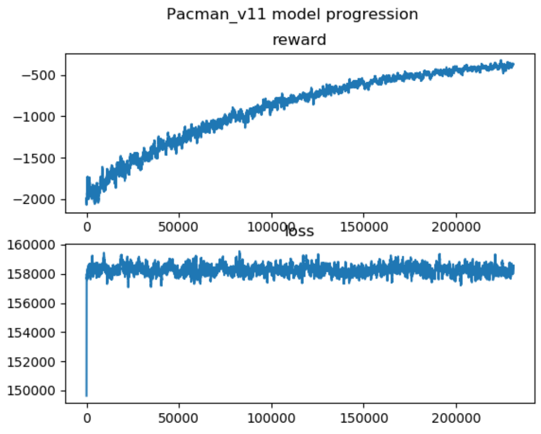

Fig. 5. Conv2D GlobalMaxPooling Dense

Looking at the model behaviour thanks to the render function of the environment, I can see the agent keeps getting back and forth in the void. It successfully learn to avoid lots of walls and the ghost but only in certain direction. He also loved pacgum above his head but don't mind the rest of them. This was a good approach to teach the agent some basic ideas of moving but with a step back I can see the limits.

All my agents for the moment are way dumber then those one could found on the internet. In particular knowing ghosts don't follow Pac-Man.

APPENDIX

There is still a lot of work to do. The next model will combine the two approach. The input will be split in the one hand to a Conv2D (64, (2, 2)) and a GlobalMaxPool2D and in the other hand for a two layers Dense (100). Adding the two results before the output dense will according to Multiplicative Weights Update theory, give the agent way more knowledge than the best of the two method. After that maybe, it would be able to eat all pacgums...

REFERENCES

[1] *http://ai.berkeley.edu/project$_o$verview.html*
[2] *https://github.com/grantjenks/free-python-games/blob/master/freegames/pacman.py*
[3] *https://www.youtube.com/watch?v=QpyHYRBKy8Ut=370s*
[4] *https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288*