Reinforcement learning

BOISSIN Thibaut

Table of contents

I Problem selection **2**
1 Taxi problem . . . 2
2 2048 . . . 2

II Value Iteration **2**
1 Taxi problem . . . 3
2 2048 game . . . 4

III Policy Iteration **5**
1 Taxi problem . . . 5
2 2048 game . . . 6

IV Q learning with Neural Network approximation **7**
1 modification of the original algorithm . . . 7
2 method . . . 7
3 results . . . 8

V Conclusion **8**

I. Problem selection

For this assignment two problems have been chosen. In order to choose those, two main characteristics has been used in order to compare and contrast the results : having one stochastic and one deterministic problem would have been interesting. Also it was required to choose a problem with a *small* number of states and an other with a *large* number of states.

1. Taxi problem

The first chosen problem is the Taxi problem, here is the definition of the problem according to the openAI website :

> *This task was introduced in [Dietterich2000] to illustrate some issues in hierarchical reinforcement learning. There are 4 locations (labeled by different letters) and your job is to pick up the passenger at one location and drop him off in another. You receive +20 points for a successful drop-off, and lose 1 point for every time-step it takes. There is also a 10 point penalty for illegal pick-up and drop-off actions.*

This problem is interesting as it is fully deterministic and have a reasonable amount of states (compared to the second problem). The problem parameters are the following :

- size of the grid : 5*5
- amount of passengers to pick : 5
- amount of possible destinations : 4

There are then $5 * 5 * 5 * 4 = 500$ states for this problem.

2. 2048

The second problem chosen is the the puzzle game 2048. 2048 is played on a gray 4x4 grid, with numbered tiles that slide smoothly when a player moves them using the four arrow keys. Every turn, a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided.
This problem is stochastic as the new tiles appears randomly on the board. Also the amount of state is huge compared to the previous problem. According to this website [1] there are approximately 1.3 trillion possible states for a 4x4 board. In order to being able to run value and policy iteration on this problem, the size of the board will be reduced to 3x3 (where the is "only" 25.176.014 different states). Assuming each board can be encoded with a 4bytes per tiles, so that is would require $4 * 3 * 3 = 36$ bytes for each board. Storing the entire state space would then require 906444504 bytes (approx 1G) which is big but affordable. The main problem is the exploration of these states would take too long if done in a naive way, we then need a clever way to explore only a subset of the elements.

II. Value Iteration

The value iteration algorithm make use of the Bellman update in order to compute iteratively the utility of each state (based on it's discounted reward). This algorithm is guaranteed to converge in a finite time if $\gamma < 1$.

1. http ://jdlm.info/articles/2017/12/10/counting-states-enumeration-2048.html

1. Taxi problem

The following graph plot the learning curve of the algorithm, the score is the average reward made on 30 episodes. The second graph plot the distance between the current value and the optimal value. The distance is defined as the sum of the absolute differences (please note that this means the plotted values are dependent to the number of states, this is why we can not compare the absolute values obtained on the Taxi problem with those obtained on the 2048).

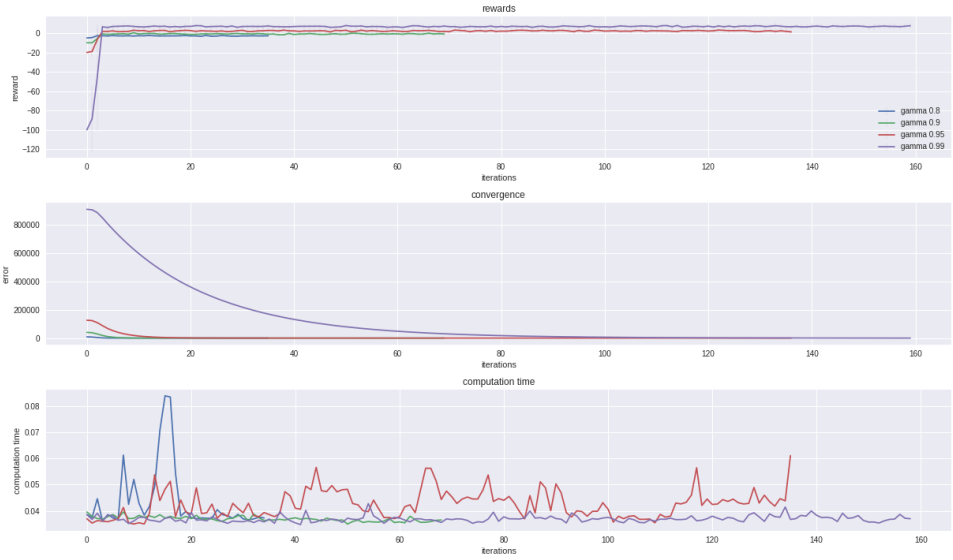

FIGURE 1 – scores and error evolution over iterations

As we can see having $\gamma \to 1$ improve the results as the utility of each state take long term rewards into account. In order to illustrate this we can define a " disount time horizon" as following : $gamma^{discountTimeHorizon} = 0.5$. This "discount time horizon" can give a measure of how far the agent takes long term reward into account. We can compute this quantity for the various values of γ :

- $\gamma = 0.8 \implies timeHorizon = 3$
- $\gamma = 0.9 \implies timeHorizon = 6$
- $\gamma = 0.95 \implies timeHorizon = 13$
- $\gamma = 0.99 \implies timeHorizon = 68$

As the taxi problem takes place in a 5x5 world, we can typically imagine that high reward actions (such as deposit/pickup), will take place in approximately than 10 time steps (Manhattan distance between two opposite corners of the world). Also the rewards constructed such that having more than 20 time-step between two drop-off make the action useless. As we have $\gamma < 0.95 \implies timeHorizon < 10$, it would explain why the obtained policies with $\gamma = 0.8$ and $\gamma = 0.9$ have negative average rewards.
Also we can note that the computation time is fairly constant over the iterations, and quite independent of gamma. This was as expected because the algorithm complexity is :

$$O(\#_{iterations} * \#_{states} * \#_{actions})$$

This shows an important thing about this algorithm : in order for this to work, we must be able to enumerate all states of the MDP which is easy in this case, be it makes the algorithm not very scalable as we have to store all the transitions and their probability.
We can finally observe an interesting fact : even if the values have not converged yet, the extracted policy does. This highlight the fact that the policy only take into account the $\arg\max_{action}[value(state)]$

2. 2048 game

In contrast to the taxi problem the 2048 has a very high amount of different states. Despite that was said in the introduction, even solving the 3x3 game has not been possible. This is was due to the fact that we also need to store all the transitions and their probabilities. We then moved to the 2x2 game. With this size the amount of states has been reduced to 1554 (without taking symmetry and rotation into account). Although there is a reasonable amount of states, the transitions matrix still need to be computed. The transition matrix has the following structure :

- each row correspond to a state
- each column correspond action
- each element is the list of the possible output of the action. The format of an output is as following : (probability, next state, reward, done)

In order to find the row corresponding to a specific state, the following formula has been applied :

$$\begin{aligned} row_{state} = \quad & board[0,0] \\ & + 6 * board[0,1] \\ & + 6^2 * board[1,0] \\ & + 6^3 * board[1,1] \end{aligned}$$

The 6 comes from the fact each tile can have 6 different values : 0, 2, 4, 8, 16, 32 (it is impossible to get a 64 on a 2*2 game). We then have an other problem : this way of indexing states, add in the transition matrix some inaccessible states (a board full of 16 is not accessible). As the value iteration iterate over the entire transition matrix, we don't want these unreachable states to interfere. To do so some DFA minimization algorithm could have been used but here simply replacing all unreachable state by a sink which end the game works also.
Now we have the the transition matrix we can apply the value iteration algorithm. The following figure shows the results obtained :

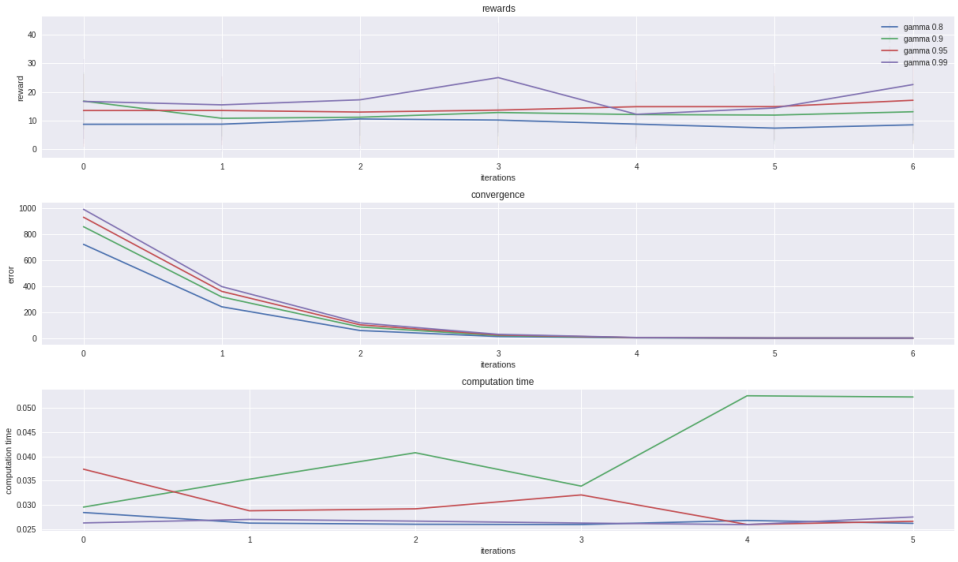

FIGURE 2 – scores and error evolution over iterations

Even if each iteration take longer due to amount of states and the stochastic aspect of the problem, the algorithm converged in a fewer iterations. This may be due to the fact that the utility of the 2048

states is strongly linked to the instant reward, in contrast to the taxi problem. Most of the time, using the Bellman equation to update the value doesn't change the max part of the equation a lot. It is those changes in the best action that delay the convergence.
Also we can see that the stochastic aspect of the 2048, leads to some variations in the results, even with a high number of games and a fixed seed. This seems to be the case especially with values of $\gamma \to 1$. A surprising fact is that we would the average rewards to be more spread, as instant reward are only power of two's.

III. Policy Iteration

According to previous results, we can see that we can get an optimal policy even if the utility function estimation has not converged yet. This is because often one action is clearly better than the others. Then, evaluating directly policy instead of extracting it from the utility function is then relevant. This is was policy iteration does. we can then expect this algorithm to converge in fewer iterations. The cost of each iteration still need to be discussed.

1. Taxi problem

By looking at the results obtained on the value iteration algorithm, we can expect the policy iteration in less than 10 iterations. As the total reward gained from then reached stagnation. Also we expect both algorithms to converge to the same results.
The next figure shows the results of the policy iteration algorithm :

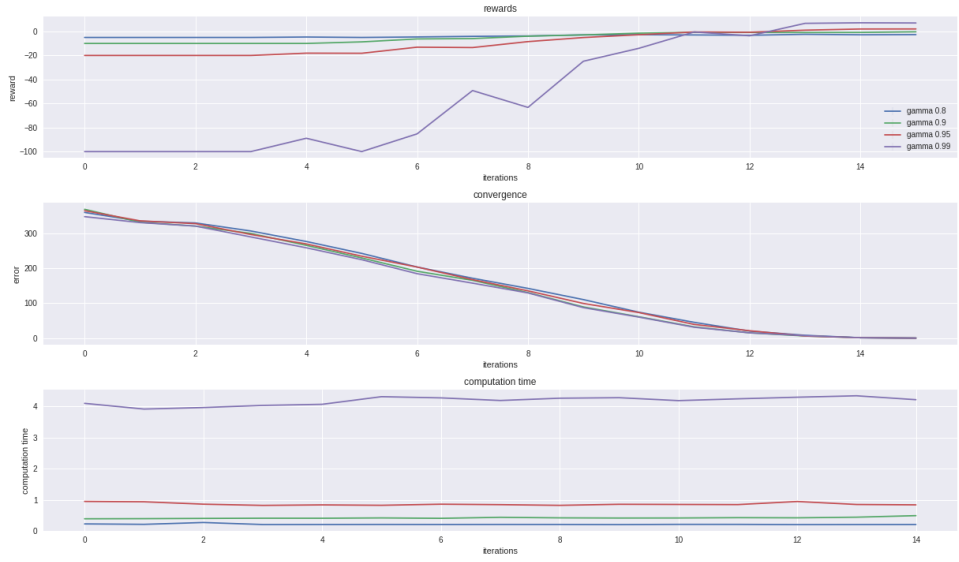

FIGURE 3 – scores and error evolution over iterations

In this case the error is defined as the amount of different values in the policy a current step and the final policy.
As expected the algorithm converge in a very few iterations, as the algorithm stop before the full convergence of the Utility function. We can also note that each iteration is much heavier. This is comprehensible as the each iteration require to compute the value function of a given policy (using an algorithm similar to value iteration, but it rely on the current policy instead of and estimation of

the value function). In this case we can see that the cost of one iteration is considerable (1s against 0.05s for the value iteration). We can note that in this case the gain on the iteration amount is overridden by the cost of each iteration. This is probably a result of a very low ϵ threshold on the value function estimation (the estimation stops on a $10^{-}10$ stagnation). The cost of an iteration is also very dependent of the γ value ; As seen on the value iteration process, higher values of gamma makes the value function estimation converge slower.
Also the results of the policy iteration has been compared to the result of the value iteration :

```
divergence between policy iteration and value iteration for gamma= 0.8  :  0
divergence between policy iteration and value iteration for gamma= 0.9  :  0
divergence between policy iteration and value iteration for gamma= 0.95  :  0
divergence between policy iteration and value iteration for gamma= 0.99  :  0
```

Remark. *The divergence above is also defined as the amount of different values between the two compared policies.*

As we can see both algorithms converge to the same policies.

2. 2048 game

Like for the values iteration algorithm we used the explicit MDP of 2048 with a 2x2 board. The following figure shows the results of the policy iteration :

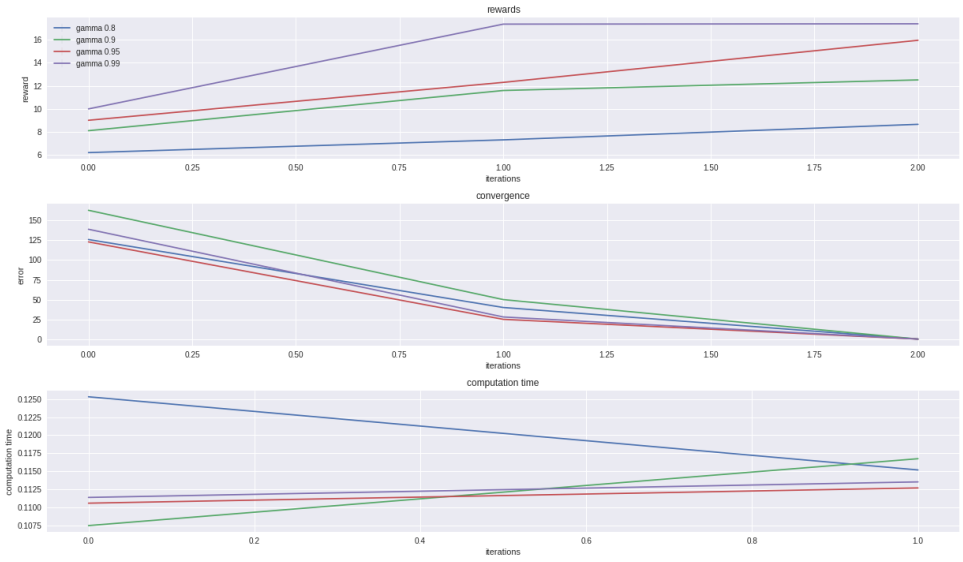

FIGURE 4 – scores and error evolution over iterations

As expected the algorithm did converge faster, also we can note that a single iteration is less expensive. In this case the computation time for both algorithms is roughly the same, which let us suppose that the policy iteration, would give better performance on bigger stochastic problem. Strangely, the relation between the value of γ and the computation, seems not to be respected. Explaining this is quite difficult as the algorithm converge in only 3 iterations.
As for the taxi problem, the resulting policy is the same as the policy extracted from the value iteration algorithm.

IV. Q learning with Neural Network approximation

The Q-learning use a derivation of the value iteration, but instead of approximate the utility function, we approximate the Q function. This change has one major advantage : this algorithm does not need an explicit modeling of the system (as the update step don't require the probability distribution anymore). But the this has a significant drawback, the lack of information about the model make the algorithm converge more slowly. Here we are going to explore a variant of the Q learning algorithm, in which the Q function will be approximated with a neural network.
This variant has been chosen after unsuccessful experiments on the 2048 game with a 4x4 board. The main problem is the explosion of the states. In order to prevent this the idea was to use a neural network to approximate the Q function. This is possible as the update of the Q function is very similar to the neural network fitting function. Moreover neural networks allow partial fitting which is mandatory in his case. The main problem for this was the tuning of the parameters, as there is instant feedback, on this way of learning. Also finding a correct size for the neural network is difficult, due to the lack of information about the Q function structure (how to know that the Q function under/over-fit, since we don't have any reference ?)

1. modification of the original algorithm

The original algorithm use the following update equation :

$$\begin{aligned} Q(s_t, a_t)_{k+1} &= (1-\alpha)Q(s_t, a_t)_k + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a)_k] \\ &= Q(s_t, a_t)_k \quad + \alpha \left[r_t + \gamma \max_a (Q(s_{t+1}, a)_k) - Q(s_t, a_t)_k\right] \end{aligned}$$

The second part of the equation can be seen as an error between $r_t + \gamma \max_a(Q(s_{t+1}, a)_k)$ and $Q(s_t, a_t)_k$ so we can replace it with our neural network training rule. Here is a pseudo-code for the neural network update :

$$Q_{nn}.fit([s_t, a_t], r_t + \gamma \max_a(Q_{nn}.predict(s_{t+1}, a)))$$

Although there is absolutely no guarantee for this system to converge. In this particular case, we know that :

- the rewards are fully deterministic as the position of the new tile has no impact on the score, only the action, and the current state does.
- the utility of a state is strongly related to immediate rewards.

The hypothesis made is that if the we start by training a Q function only on instant rewards, before training it with the method described above, we should increase the probability of convergence.

2. method

As the is no guaranties of convergence, the strategy was to make successive training using increasing value for γ by starting with $\gamma = 0$ we ensure the neural network to first learn to approximate immediate rewards, and to progressively approximate long term rewards. The procedure was as the following :

1. 2000 games with $\gamma = 0$
2. 3000 games with $\gamma = 0.2$
3. 4000 games with $\gamma = 0.3$
4. 4000 games with $\gamma = 0.4$

Each step will use the results of the previous step as initial condition. Sadly sklearn does not allow to manually change the learning rate of a neural network, in order to avoid this rate becoming too low, a constant rate of 0.2 has been used. At each step the results will be stored and evaluated, to check if the system do converge or not.

3. results

The following tabular shows the results obtained at each step :

step	average score
1	201
2	205
3	204
4	202

TABLE 1 – Score evolution

As we can see, the results does not improve over the iterations. For now no satisfying reason has been found. There although some potential causes :

- an error in the algorithm modification
- wrong parameters settings (especially the neural network dimension, the value of γ and the learning rate)
- an error in the testing program, as the training program use a python environment with sklearn v0.19 while the gui use sklearn v0.18
- too short training time : currently each game is limited to 1000 move.

V. Conclusion

We can now highlight important points about value iteration, policy iteration and Q learning :

Value Iteration One of the main interest with values iteration is that this algorithm take profits of prior knowledge to have a efficient and accurate estimation of the utility function. The policy is then extracted from the utility function.
There is although some drawbacks :

- all the states and all the transitions needs to be explicit (does not scale for large or continuous state spaces)
- take more iterations to converge even if the extracted policy is already optimal. It usually take longer to converge but this was not the case for our problems as they seems to be too small.
- even if this hasn't been measured in this assignment, the extraction of the policy can be time consuming

Policy iteration The policy iteration give a solution to the 2^{nd} drawback of the value iteration as it compute directly policies. This is done in two steps :

- policy evaluation : by evaluation the utility function based on the current policy
- policy improvement : it used the evaluation of the policy to improve the policy

This processing have two main characteristics : it converge in fewer steps, but those steps are heavier to process.
It then have the following disadvantages :

— The utility of the states can not be used
— sometime the cost of an iteration override the gain over the amount of iterations

Q learning with neural network approximation The use of Q learning allow to avoid the use of prior knowledge, as the Q function include the probabilities on the transitions. The use of a neural network allow to deal with big or continuous state spaces. These advantages come at the cost of some drawbacks :

— the algorithm take much longer to converge
— a fine tuning on the neural network parameter is required
— the proof of convergence is not applicable with the neural network
— it is subject to over/under-fitting