Artificial Intelligence: Probabilistic & learning techniques

Chris Pal

Project (INF8225)

Farnoush Farhadi & Juliette Tibayrenc

21 Apr. 2016

Contents

Introduction

Our AI course has allowed us to discover Markov Decision Processes, which we decided to further explore in our project. Looking for an application for them, we stumbled upon Mastronarde & Van der Schaar's article, 'Fast Reinforcement Learning for Energy-Efficient Wireless Communications'[1], and we chose it as the basis for our work.

The article presents the two researchers' work in developing an algorithm that would enable systems to adopt the best policy to quickly send data packets in the most efficient way possible, an important goal to achieve since the majority of today's surfing is done on smartphones whose battery does not last longer than a day (at best).

In this report, we will first introduce the article, with a quick review of the state of the art at the time it was written (2011), then summarize it. We will continue with a presentation of some theoretical elements, highlighting the differences between what we saw in class and the processes on which the article and our implementations are based, and the way the reward matrix used is calculated (a process that does not appear in the article). Finally we will talk about our own implementation of classic Q-learning and one of its variants, speedy Q-learning, and compare the results we obtain with the results obtained with the authors'algorithm, plus the results we get by using value iteration and policy iteration.

1. Reference article

1.1 Literature review

1.2 Article summary

2. Theoretical elements

2.1 Value iterating & policy iterating vs. Q-learning

In our project, we compare the performance of several algorithms against that of the authors. These algorithms have to be evaluated along several criteria, among which the elements that need to be known for them to work, the speed with which they are executed (partly dependent on the efficency of the implementation and the language used) and the number of iterations their internal loop is executed for.

These algorithms are value iteration, policy iteration, classic Q-learning and speedy Q-learning.

2.1.1 Value iteration

One method to find an optimal policy is to find the optimal value function by determining a simple iterative algorithm called value iteration. The value iteration method is referenced by Bellman equation for MDP problems which is obtained simply by turning the Bellman optimality equation into an update rule. If there exist n possible states, then we have n Bellman equations, one per each state. The n equations contain n unknowns which are the utilities of theses states.

We aim to solve these simultaneous equations to find the utility values. These equations are non-linear, since the max operator is not a linear one. Hence, it can be determined by a simple iterative algorithm that can be shown as follows:

$$V_{i+1}(s) = R(s) + \gamma \mathrm{max_a} \sum_s P(s, a, s') V_i(s')$$

where i is the iteration number. Value iteration starts at $i = 0$ and then iterates, repeatedly computing V_{i+1} for all states s, until V converges. Value iteration formally requires an infinite number of iteration to converge to V^*. In practice, it stops once the value function changes by a small amount in a sweep which satisfies the Bellman equations. This process is called a value update or Bellman update/backup. The algorithm for value iteration is as follows (from Artificial Intelligence, 3rd edition, French version [2]):

Note that the value iteration backup is identical to the policy evaluation except that it requires the max operator to be taken over all actions.

2.1.2 Policy iteration

In policy iteration algorithm, the agent manipulates the policy directly, instead of finding it indirectly via the optimal value function. The policy often becomes exactly optimal long before the utility estimation have converged to their true values. This phenomena leads us to another way of finding optimal policies, called policy iteration. Policy iteration picks an initial action arbitrarily by taking rewards on states as their utilities and computing a policy according to the maximum expected utility. Then it iteratively performs two steps: firstly, it determine the values through computing the utility of each state given the current action, and then policy improvement by updating the current policy if possible. To detect when the algorithm has converged, it should only change the policy if the new action for some state improves the expected value; that is, it should stabilize the policy. The algorithm for policy iteration is as follows:

At the end of algorithm, we test the stable variable; if it is 'True', algorithm terminates, otherwise, control jumps to the

Policy Evaluation and continues in an iterative manner. Policy iteration often converges in few iterations. The policy improvement theorem guarantees

that these policies works better than the original random policy and achieves to the goal states in the minimum number of steps.

2.1.3 Classic Q-learning

Q-learning is part of a sec-

ond cat- e- gory of al- go- rithms that do not pre- sup- pose the full knowl- edge of the sys- tem. Con- trary to value it- er- a- tion and pol- icy it- er- a- tion, the en- vi- ron- ment does not need to

be known for the algorithm to work. The algorithm presented by the authors of our reference article is similar in its requirements, though it goes further and actually uses

the knowledge we do have about the environment, which makes it more efficient.

Q-learning helps us find the optimal policy by using the interactions between the system and the en-

vi-
ron-
ment.
In
it,
we
com-
pute
an
es-
ti-
mate
of
the
cu-
mu-
la-
tive
dis-
counted
re-
ward
for
each
ac-
tion
ex-
e-
cuted
in
each
state.
The
al-
go-
rithm
for
one
step
is
as
fol-
lows
(from
Ar-
ti-

fi-
cial
In-
tel-
li-
gence,
3rd
edi-
tion,
French
version[2]:

Algorithm 3 Q-learning algorithm for one step

Data: s' /* current state */, r' /* reward signal */
Result: Q /* action values table indexed by states and actions */, T /* frequency of state-action couples */, s, a, r /*previous state, action and reward */
if *s final* **then**
  $Q(s, Empty) = r'$
if $s \neq 0$ **then**
  $T(s, a) + +$
  $Q(s, a) += \alpha \cdot T(s, a)(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
$s, a, r = s', _{a'} f(Q(s', a'), T(s', a')), r'$
return a

2.2 Reward matrix

3. Comparing algorithms

3.1 Classic Q-learning

3.2 Speedy Q-learning

3.3 Additional remarks

Further work & conclusion

Bibliography

[1] N. Mastronarde and M. Van der Schaar, "Fast reinforcement learning for energy-efficient wireless communication," *IEEE Transactions on Signal Processing*, vol. 59, 12 2011.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence, Prentice Hall, 3rd ed., 2010.