Can the replayed sequences during sleep can be modelised by a reinforcement learning algorithm? – Rapport de stage

Paul Ecoffet

15 avril 2016

Abstract

On the neurobiology side, replays of behavioral sequences have been observed in the hippocampus during sharp wave ripples complexes and is thought to be a mechanism involed in memory consolidation and learning. On the computer science side, some reinforcement learning algorithms also use replays of behavioral sequences to improve the learning speed. One can ask if this algorithms behave in the same way that hippocampus replays of sequences, and if these algorithms can model the replay mechanisms observed in the hippocampus. For instance, can those algorithms explain why certain sequences are replayed instead of others? We'll show that that ** we'll see **.

Introduction

Reinforcement learning

Reinforcement Learning is a kind of learning mechanism where an agent tries to maximise a reward by doing some specific action in an environment (R. S. Sutton & Barto, 1998). The environment is defined by set of *states* (noted $\mathcal{S}$) in which the agent can be. In each state, the agent can do an *action* from its *action set* (noted $\mathcal{A}$). When doing a specific action a in a specific state s, the agent receives a *reward* (noted r) and go in a new state $s' \in \mathcal{S}$. The reward is a number which can be positive or negative. The goal of the agent is to maximise the amount of reward it receives over time. The agent must do the best action in a given state to get the most reward in the long run. This mean that it might be worthful to go in a state which gives no reward if it can lead to new states that give a high reward. The table which links each state to an action is called a *policy*. Reinforcement learning are methods to find the best policies in a given environment.

This kind of environment and task is called a *Markov Reward Process*. This process can be used to model a wide variety of situation such as maze solving tasks or workload management (R. S. Sutton & Barto, 1998; Vanseijen & Sutton, 2015).

For instance, we can model experiences on rats who does maze solving task with markovian reward process. For instance, if we model a tree maze, each state is a node of the tree, the action of the rat is to take a edge. When it does so, he goes to another node (a new state), and receive the amount of food pellets in this node. Figure 1 shows an exemple of such maze. In this model, the goal of the rat is to eat as much food pellets as possible. The agent must maximise its amount of reward over time.

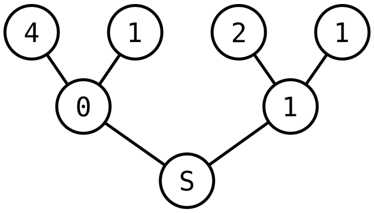


Figure 1: Exemple of an environment with states and rewards
Each number is the reward associated with the state. The goal of the agent is to maximise the amount of reward it gets in an episode. Therefore, he has to choose the action “go left” to get the 0, then choose “go left” again to get the 4.

If the agent knows his environment perfectly, it is possible to find the best policy to maximise the amount of reward the agent gets in the long run analytically.

To know if it is “interesting” to go into a state s, the agent can compute the *value* V_π of it. The value of a state is the expected reward the agent will receive if he is in this state plus the future rewards it can get from future states according to its policy π. Rewards from future states are discounted by a coefficient γ.

$$V_\pi(s) = r_{s,a} + \gamma \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \tag{1}$$

With complete knowledge of the rewards r, the transition table of state according to an action (ie. the fact that I will be in s' if I do action a while in s), the value of every state can be computed according to the agent's policy. It is then possible to compute the best policy possible analytically or iteratively with Expectation-Maximisation (EM) algorithm (R. S. Sutton & Barto, 1998).

When the environment is not known by the agent, the value function cannot be computed and it must be approximated. The agent explore the environment and observe for each state the reward it gets, the next state it will be according to its action and update a value function approximation $\tilde{V}_\pi$. Different methods such as Monte Carlo or Temporal Difference (TD) provide way to approximate

V. Then, the policy can be updated with an EM algorithm on $\tilde{V}_\pi$ or use another policy improvement method.

Temporal Difference algorithm

Temporal difference (TD) algorithms are methods that do not require a perfect knowledge of the environment. The TD(0) algorithm (Algorithm 1) is the most basic TD learning algorithm. TD(0) algorithm computes an approximation of $\tilde{V}_\pi$ after each action of the agent, using the equation 2.

$$\hat{V}_\pi(s) = r + \gamma \tilde{V}_\pi(s') \tag{2}$$

The $\hat{V}_\pi(s)$ which is computed takes only into account the next state s' the agent happens to be in after taking its action, and not all the possible outcomes as $V_\pi(s)$ does. $\tilde{V}_\pi(s)$ is updated slightly toward $\hat{V}_\pi(s)$ so that in the long run, $\tilde{V}_\pi(s)$ takes into account every outcome r and s' possible, according to their probability of occurence. To do so, a *learning rate* parameter $\alpha \in]0,1[$ is set, and $\tilde{V}_\pi(s)$ is updated with the formula in equation 3.

$$\tilde{V}_\pi(s) = \tilde{V}_\pi(s) + \alpha(\hat{V}_\pi(s) - \tilde{V}_\pi(s)) \tag{3}$$

TD learning and reinforcement learning is fully covered in R. S. Sutton and Barto (1998).

Algorithm 1: Canonical TD(0)

```
Input: current $\tilde{V}$ (usually $\vec{0}$), first state $s$, policy $\pi$, learning rate $\alpha$,
       discount factor $\gamma$
Output: Updated $\tilde{V}$
while $s$ is not end do
    $a \leftarrow$ action according to $\pi$ for $s$
    Do $a$, observe the reward $r$ and the new state $s'$
    $\delta \leftarrow r + \gamma \tilde{V}(s') - \tilde{V}(s)$
    $\tilde{V}(s) \leftarrow \tilde{V}(s) + \alpha\delta$
    $s \leftarrow s'$
end
```

It is interesting to note that the Temporal Difference δ, which correspond to the error between the expected value of a state and the actual value observed $(\hat{V}_\pi(s) - \tilde{V}_\pi(s))$, has been correlated with the dopaminergic response of neurons by Hollerman and Schultz (1998).

Learning by replay

In reinforcement learning

Reinforcement learning methods based on TD(0) like Sarsa or Q-Learning converge slowly and need a lot of samples to be efficient. These algorithms use the sample once to improve their solution, then discard it (Adam, Busoniu, & Babuska, 2012).

To improve the efficiency of reinforcement learning methods, one solution is to replay past experiences. Vanseijen and Sutton (2015) offers a good analysis about learning by replay. The goal of replay is to use the maximum of information an experience offers. Each sample is used several times using the current knowledge of the agent, improving the solution several times. Compared to TD(0), replay techniques give a better convergence to the optimal solution with the same number of experiences. Reinforcement learning methods using replay are more expensive than TD(λ) both in memory and in computations, though the memory and computational power needed can be reduced a lot as Vanseijen and Sutton (2015) shows.

Vanseijen and Sutton (2015) give a canonical TD(0) algorithm with replay. This algorithm puts in a list $\mathcal{M}$ all the states, reward and transition the agent have observed at each step. After each action, the algorithm use the new value function estimation it has to replay previous states to compute from the beginning of the experience a new value function estimation given these previous states (in $\mathcal{M}$) and the current knowledge it has ($\tilde{V}_\pi$).

Note that starting from now, I will no longer use π as an indice for the value function to thow that the value function depends on the policy π, but I will put as indice the time step at which the estimation has been done. $\tilde{V}_4$ correspond to the estimation of the value function the algorithm gets after its forth action (of course, according to a policy π). It is also understated that $\tilde{V}_{t+1} = \tilde{V}_t$ for every state except from the one for whom the function has been updated.

The example of computation of $\tilde{V}_\pi$ using replay proposed by Vanseijen and Sutton (2015) is given in equations 4 and 5. Equation 4 shows the updates of a canonical TD(0) algorithm, as in algorithm 1.

$$\begin{aligned}
\tilde{V}_1(s_0) &\leftarrow \tilde{V}_0(s_0) + \alpha(r_1 + \gamma\tilde{V}_0(s_1) - \tilde{V}_0(s_0)) \\
\tilde{V}_2(s_1) &\leftarrow \tilde{V}_1(s_1) + \alpha(r_2 + \gamma\tilde{V}_1(s_2) - \tilde{V}_1(s_1)) \\
\tilde{V}_3(s_2) &\leftarrow \tilde{V}_2(s_2) + \alpha(r_3 + \gamma\tilde{V}_2(s_3) - \tilde{V}_2(s_2))
\end{aligned} \tag{4}$$

The idea of replay is to use the knowledge got after some time. For instance, let $\tilde{V}_* = \tilde{V}_3$. Then, it is possible to compute a new estimate of the value function from scratch using this knowledge and past experiences:

$$\begin{aligned} \tilde{V}_1'(s_0) &\leftarrow \tilde{V}_0(s_0) + \alpha(r_1 + \gamma \tilde{V}_*(s_1) - \tilde{V}_0(s_0)) \\ \tilde{V}_2'(s_1) &\leftarrow \tilde{V}_1'(s_1) + \alpha(r_2 + \gamma \tilde{V}_*(s_2) - \tilde{V}_1'(s_1)) \\ \tilde{V}_3'(s_2) &\leftarrow \tilde{V}_2'(s_2) + \alpha(r_3 + \gamma \tilde{V}_*(s_3) - \tilde{V}_2'(s_2)) \end{aligned} \tag{5}$$

$\tilde{V}'$ is reasonably more accurate that $\tilde{V}$, as stated by Vanseijen and Sutton (2015). Using the same method, $\tilde{V}''$ can be computed using the same method and with $\tilde{V}_* = \tilde{V}_3'$ and so on, depending of the computational budget of the algorithm. The algorithm 2 can therefore be constructed (Vanseijen & Sutton, 2015).

Algorithm 2: TD(0) with Replay

```
Input: $\tilde{V}_{init}$ (usually $\vec{0}$), first state $s$, policy $\pi$, learning rate $\alpha$, discount
       factor $\gamma$, computational budget $k$
Output: Updated $\tilde{V}$
$\tilde{V} \leftarrow \tilde{V}_{init}$
$\mathcal{M} \leftarrow \emptyset$ // The list of observed samples
while $s$ is not end do
    $a \leftarrow$ action according to $\pi$ for $s$
    Do $a$, observe the reward $r$ and the new state $s'$
    append $(s, r, s')$ to $\mathcal{M}$
    for $i \leftarrow 1$ to $k$ do
        $\tilde{V}_* \leftarrow \tilde{V}$
        $\tilde{V} \leftarrow \tilde{V}_{init}$
        foreach $(s, r, s') \in \mathcal{M}$ (from oldest to newest) do
            $\delta \leftarrow r + \gamma \tilde{V}_*(s') - \tilde{V}(s)$
            $\tilde{V}(s) \leftarrow \tilde{V}(s) + \alpha\delta$
        end
    end
    $s \leftarrow s'$
end
```

Planning and replay?

RL algorithms with planning use a model of their environment to infer the value of a behaviour without actually doing it. The model is commonly a transition table between the possible states in the environment according to the action done by the agent. For exemple, if the agent has to solve a maze and has access to a transition table, then it can look up the table and know that if he is in the case $(3, 5)$ and it does the action “go to the east”, it will be in $(4, 5)$, without actually doing the action.

The model can be either directly accessible (given by the developer) or inferred by the agent. When the model is inferred, the agent builds explicitly the transition table according to the experiences he has in its environment. In this document, we will mainly talk about the Dyna algorithms. Dyna is a class of RL algorithms where the model is inferred by the agent. There is two phase during an iteration of Dyna. The first phase is when the agent interacts with its environment: it observes its state and reward, do an action according to its policy and go to the next state. It also updates its model of the environment according to this new experience. The second part is the *planning* phase. During the planning phase, the agent improves its value function estimation with simulations according to its model: The agent generates a sample state, an experience, according to some rules (a probability distribution) and uses its model to compute the value of this sample thanks its model. Then, it updates its solution according to this simulation (R. S. Sutton, Szepesvári, Geramifard, & Bowling, 2012). Algorithm 3 from R. S. Sutton et al. (2012) is the canonical form of a Dyna algorithm. In this algorithm and next algorithms, F and b represent the model of the environment built by the agent. F is a transition matrix. If ϕ is the current feature vector of the state the agent is, then $\phi' = F\phi$ is the estimated feature vector of the next state the agent will be according to its policy π. b is the expected reward per feature activation vector. For instance, if all the features observed by the agent are at 0 except from the i-th feature is at 2 and $b_i = 0.25$, then the expected reward for this state according to the agent's model is $\hat{r} = 0.5$.The expected reward from a feature vector according to the model of the agent is given by equation 6.

$$\hat{r} = b^\top \phi \tag{6}$$

It is interesting to note that the distribution used to draw samples has no influence on whether the solution will converge or not as long as it covers all the possible features, but has an influence on the speed of convergence. Therefore, choosing a good distribution can greatly improve the performance of Dyna. R. S. Sutton et al. (2012) shows that choosing only already explored samples can have a good impact on convergence speed. Therefore, even if R. S. Sutton et al. (2012) do not explicitly say it, the planning phase is then a *replay* phase. In their paper, they present two algorithms which use replays: Linear Dyna with PWMA prioritized sweeping (LD-PWMA) and Linear Dyna with MG prioritized sweeping (LD-MG). Both these algorithm store all the features the agent encounters in a queue. The features are then drawn from this queue during the planning/replay phase. The idea of prioritized sweeping is to draw the samples in a specific order according to a criterion which will speed up the convergence. PWMA and MG use both a function of the difference between the expected value of the state and its actual value, the value δ, as the criterion to sort the samples. R. S. Sutton et al. (2012) show that Linear Dyna with MG prioritized sweeping (LD-MG) out performs several classic RL algorithms and PWMA-LD in speed of convergence for classical RL problems.

Algorithm 4 is the pseudo code implementation of Linear Dyna with MG prioritized sweeping policy evaluation.

Algorithm 3: Canonical Linear Dyna

```
Input: $\theta_{init}$ (usually $\vec{0}$), first features vector $\phi$, policy $\pi$, learning rate $\alpha$,
       discount factor $\gamma$, computational budget $k$
Output: Updated $\theta$
$\theta \leftarrow \theta_{init}$
while not end do
    $a \leftarrow$ action according to $\pi$ for $\phi$
    Do $a$, observe the reward $r$ and the new state $\phi'$
    $\theta \leftarrow \theta + \alpha(r + \gamma\theta^\top\phi' - \theta^\top\phi)\phi$
    $F \leftarrow F + \alpha(\phi' - F\phi)\phi^\top$
    $b \leftarrow b + \alpha(r - b^\top\phi)\phi$
    $mem \leftarrow \phi'$
    for $i \leftarrow 1$ to $k$ do
        Generate a sample $\phi$ from distribution $\mu$
        $\phi' \leftarrow F\phi$
        $r \leftarrow b^\top\phi$
        $\delta \leftarrow r + \gamma\theta^\top\phi' - \theta^\top(\phi)$
        $\theta \leftarrow \theta + \alpha\delta$
    end
    $\phi \leftarrow mem$
end
```

Vanseijen and Sutton (2015) highlights even more the strong similarity between RL methods using replay and RL methods using planning. First of all, both of them can be considered as model-based algorithms. Indeed, planning methods using an inferred model use an integrated version of samples experienced over time, thus they are by definition model-based algorithms. RL algorithms using experience replay do not store an explicit model, with transition tables and reward estimation. Yet, they store for each sample the state they were in, the reward they received and the next action they did. These data can be considered as a model. They implicitly contains an inferred model of the environnement. Transitions and rewards are available through all the samples the agent experienced and stored in memory (Vanseijen & Sutton, 2015). This is

In addition to that, Vanseijen and Sutton (2015) shows a complete equivalence in the value function approximation at each iteration between a TD(0) inspired algorithm with replay and Linear Dyna, a model-based RL algorithm. Though Linear Dyna is a model based algorithm and thought as a "looking ahead" algorithm and the replay algorithm as a "looking backward" algorithm, they do the exact same calculation. The Linear Dyna variation Vanseijen and Sutton (2015) presents is a batch Linear Dyna. Each iteration of planning compute using an integrated version of all the states and transition experienced. One can ask if different variations of Linear Dyna can also be considered as learning by replay methods. For instance, R. S. Sutton et al. (2012) reviewed different

Algorithm 4: Linear Dyna with MG prioritized sweeping

```
Input: $\theta_{init}$ (usually $\vec{0}$), first feature vector $\phi$, policy $\pi$, learning rate $\alpha$,
       discount factor $\gamma$, computational budget $k$
Output: Updated $\tilde{\theta}$
$\theta \leftarrow \theta_{init}$
Init PQueue
while not end do
    $a \leftarrow$ action according to $\pi$ for $\phi$
    Do $a$, observe the reward $r$ and the new state $\phi'$
    $\delta \leftarrow r + \gamma\theta^\top\phi' - \theta^\top\phi$
    $\theta \leftarrow \theta + \alpha\delta$
    $F \leftarrow F + \alpha(\phi' - F\phi)\phi^\top$
    $b \leftarrow b + \alpha(r - b^\top\phi)\phi$
    forall $i$ such as $\phi(i) \neq 0$ do
        Put $i$ on the PQueue with priority $|\delta\phi(i)|$
    end
    counter $\leftarrow 0$
    while PQueue is not empty and counter $\leq k$ do
        $\phi' \leftarrow F\phi$
        $r \leftarrow b^\top\phi$
        $\delta \leftarrow r + \gamma\tilde{V}_*(s') - \tilde{V}(s)$
        $\tilde{V}(s) \leftarrow \tilde{V}(s) + \alpha\delta$
    end
    $phi \leftarrow mem$
end
```

Linear Dyna algorithms which look at one possible state per planning iteration. They shows that using already experienced states and prioritizing about states where the difference between the estimated value and the real value was the greatest is an efficient strategy.

In vivo

Place cells Place cells are high level integrative neurons in the hippocampus. They have a specific receptive field called a place field. When the animal is in the receptive field of a place cell, the place cell spikes. It is on these cells that the replay of behavioral sequences is observed during sleep.

Replay Replays of sequences have been observed in the rat hippocampus after navigation tasks (Wilson & McNaughton, 1994; Skaggs & McNaughton, 1996; Davidson, Kloosterman, & Wilson, 2009; Gupta, van der Meer, Touretzky, & Redish, 2010). The place cells whose activations were highly correlated in time are also activated during sleep and respect the same correlation (Wilson & McNaughton, 1994). Replays of place cells activation sequences that occurs during the day can also be observed either forward (Skaggs & McNaughton, 1996) or even backward (Gupta et al., 2010). Coherent sequences that have never been done are also "replayed" by the rat (Gupta et al., 2010). Though, those articles do not propose explantion about why these specific sequences were chosen or about the length of these sequences. Gupta et al. (2010) shows that the recency of the experience is a wrong hypothesis but do not propose other explanation. We will try to provide a criterion to explain why specific sequences are replayed and not others.

to check

- Replay during sleep and during hesitation
- Replay can be either forward or backward
- Replay can also be action that has never be done? −> Planning somewhat?
- Replay aren't necessary about the most recent event?
- Is "surprise" a good crterion?

Model navigation learning

We can model place cells in a reinforcement learning algorithm by using a tiling of the environment (see Figure 2) as Tamosiunaite et al. (2008) have done. Each feature of the feature vector (the state of the agent) represents the level of activation of a tile: a gaussian-like function of the distance of the agent from a kernel. For instance, the i^{th} feature of the feature vector will be the distance from the kernel k_i according to a gaussian function (see Figure 3). The value of the i^{th} feature can be interpreted as the firing rate of a place-cell with a receptive field centered in k_i.

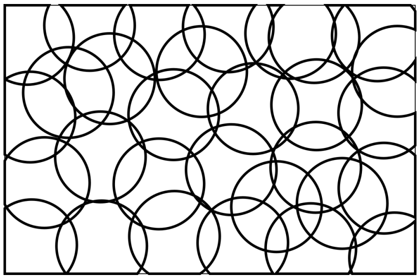
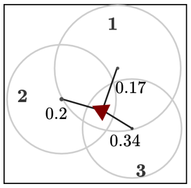


Figure 2: Tiling of an environment
Each disk can be thought as the receptive field of a place cell



$$\phi = \begin{pmatrix} 0.17 \\ 0.2 \\ 0.34 \end{pmatrix}$$

Figure 3: Features representation

What are the sequences that are replayed? How are they selected?

Gupta et al. (2010) shows that the sequences that are replayed are not necessarly the most recent experiences. It shows also that the sequences that are replayed can be either forward or backward. In forward replays, the sequence is replayed in the same order as the sequence the rat has done. In backward replays, the sequence is done in the reverse order. What is the explanation for the selection of the sequence which is replayed, and its order? No criteria are proposed in Gupta et al. (2010). Yet, we can make interesting hypothesis knowing that R. S. Sutton et al. (2012) suggests that there are very good samples to replay in a Dyna algorithm, and those sample are the sample where the difference between the expected value of a state and its real value is the greatest. It is consistant with Gupta et al. (2010) which states that the most common replays are the one that leads to the reward or comes from the reward.

Method

- Pourquoi x place-cells?
- Pourquoi ce radius?
- Quel densité?
- Quel type d'activation?

I try to reproduce the results of Gupta et al. (2010) and see if a Linear Dyna with prioritized sweeping as developped in R. S. Sutton et al. (2012) can reproduce the results observed *in vivo*. To do so, I have modelised the maze used by Gupta et al. (2010). The source code is available at ** github **. It is a continuous world.

Results

Conclusion

Limitations

References

Adam, S., Busoniu, L., & Babuska, R. (2012, March). Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(2), 201–212. doi:10.1109/TSMCC.2011.2106494

Davidson, T. J., Kloosterman, F., & Wilson, M. A. (2009, August). Hippocampal replay of extended experience. *Neuron*, *63*(4), 497–507. doi:10.1016/j.neuron.2009.07.027

Gupta, A. S., van der Meer, M. A., Touretzky, D. S., & Redish, A. D. (2010, March). Hippocampal replay is not a simple function of experience. *Neuron*, *65*(5), 695–705. doi:10.1016/j.neuron.2010.01.034

Hollerman, J. R. & Schultz, W. (1998). Dopamine neurons report an error in the temporal prediction of reward during learning. *Nature neuroscience*, *1*(4), 304–309. Retrieved April 20, 2016, from http://www.nature.com/neuro/journal/v1/n4/abs/nn0898_304.html

Skaggs, W. E. & McNaughton, B. L. (1996, March 29). Replay of neuronal firing sequences in rat hippocampus during sleep following spatial experience. *Science (New York, N.Y.) 271*(5257), 1870–1873.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement learning: an introduction* (Nachdr.). Adaptive computation and machine learning. Cambridge, Mass.: MIT Press.

Sutton, R. S., Szepesvári, C., Geramifard, A., & Bowling, M. P. (2012). Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*. Retrieved March 18, 2016, from http://arxiv.org/abs/1206.3285

Tamosiunaite, M., Ainge, J., Kulvicius, T., Porr, B., Dudchenko, P., & Wörgötter, F. (2008). Path-finding in real and simulated rats: on the usefulness of forgetting and frustration for navigation learning. *J. Comp. Neuroscience*, *25*, 562–582. Retrieved April 25, 2016, from http://www.physik3.gwdg.de/cns/modules/BibtexModule/uploads/PDF/tamosiunaiteaingekulvicius2008.pdf

Vanseijen, H. & Sutton, R. (2015). A deeper look at planning as learning from replay. In *Proceedings of the 32nd international conference on machine learning (ICML-15)* (pp. 2314–2322). Retrieved April 1, 2016, from http://machinelearning.wustl.edu/mlpapers/paper_files/icml2015_vanseijen15.pdf

Wilson, M. A. & McNaughton, B. L. (1994, July 29). Reactivation of hippocampal ensemble memories during sleep. *Science (New York, N.Y.) 265*(5172), 676–679.