Automatic adaptation of learning parameters in reinforcement learning

Nicolas Saunier

February 5, 2001

Abstract. Reinforcement learning is a powerful way of training a system to perform a certain task. The performance of a reinforcement learning system is dependent on the setting of a number of parameters that controls the learning process, and the optimal setting depends on the task and the environment. The brain seems to have a sort of metalearning mechanism which adjusts their learning parameters so that efficient learning is accomplished in an unknown environement. What we would like to find in this paper is a way of automatically tuning the learning parameters for a reinforcement learning system. We will first investigate the effect of reward prediction time-scale in continous temporal difference (T.D.) learning algorithm, the non-normalized and the normalized versions. We then propose a method for evolving the time scale parameter. We test the method in the task of pendulum swing-up with limited torque. At last we will discuss the possibility of using our algorithm for other learning parameters.

1 Introduction

Reinforcement learning is a way of training a machine to perform a certain task by trial and error. In contrast to supervised learning, where the system is trained to follow a pre-defined behaviour, in reinforcement learning, the system is given a reward according to its behaviour: based on this reward, the learner will buid an internal evaluation of the environmental state. Behaving optimally according to this evaluation, the learner should achieve the goal.

However, the performance of a learning system strongly depends on the setting of some parameters such as the learning rates, the degree of exploration, or the time scale of evaluation. Such parameters are often called *metaparameters* as they influence the way the regular parameters, which characterize the system behaviour, are learnt. The appropriate setting of these metaparameters varies widely depending on the task and the environment. They usally have to be tuned by a human expert by trial and error. That's why machine learning systems learn well when their environement is changed.

Nevertheless, biological systems like the animals don't need a supervisor to adjust the metaparameters under an unknown environment: it seems to indicate that there exists in the brain the ability to change and tune its of metaparameters, some mechanism of *metalearning*. Indeed, we have some clues supporting the idea that some neuromodulators regulate the specific metaparameters of the brain (cf. [2] [3]).

Our goals are to study the influence of some metaparameters of reinforcement learning and to propose a metalearning algorithm to tune them. This paper presents in section 2 a theoretical part on reinforcement learning: we use a continous-time temporal difference (T.D.) learning algorithm wich allows the system to learn by itself from delayed rewards, in two versions, a normalized and a non-normalized one. We then explain why and how we can make a metaparameter evolve.

We consider in section 3 a particular task, a non-linear control task of swinging up a pendulum with limited torque. The program is first tested for different values of the metaparameter. Then we implement our evolving algorithm and observe the evolution, first for a fixed environment, and then in a dynamic environment (we change a physical parameter of the system periodically).

2 Reinforcement Learning

2.1 Continuous-time TD-learning

2.1.1 The non-normalized version

We will introduce here the model developped in [4]. For a more general introduction to Reinforcement Learning, please refer to [8]. We consider here the continuous-time deterministic system

$$\dot{x}\ (t) = \ f\ (x(t),\ u(t)), \tag{1}$$

where $x \in R^N$ is the state vector and $u \in R^M$ the action vector. The reward is given by

$$r(t) \ = \ r\ (x(t), u(t)). \tag{2}$$

The TD learning algorithm enables the system to learn by itself from delayed reward. The learner updates a value function, giving for each state a value indicating how it is to be in such a state. Our goal is to find a policy, a way of choosing an action given the actual state of the system, that maximizes the value function built with the cumulative rewards

$$V^{\ \mu}(x(t)) \ = \ \int_t^\infty e^{-\frac{s-t}{\tau}}\ r(x(s), u(s))\ ds \tag{3}$$

for any initial state $x(t)$, where τ is the time constant for discounting future rewards.

We have to build an estimate of the value function. If the current estimate of the value function is perfect, it should satisfy the consistency condition

$$\dot{V}(t) \ = \ \frac{1}{\tau}\, V(t) \ - \ r(t). \tag{4}$$

If it is not perfect, the estimate should be adjusted to decrease the inconsistency

$$\delta(t) \ = \ r(t) \ - \ \frac{1}{\tau}\, V(t) \ + \ \dot{V}(t). \tag{5}$$

This is the continuous-time counterpart of the TD error.

In the simulations below, the value function is represented by a normalized gaussian network with weight parameters w_c as

$$V(x(t)) = \sum_i \ w_{c_i}\ b_i(x(t)), \tag{6}$$

where $b_i(x(t)$ is the basis function.

The update follows $\Delta w_c \propto \mathit{error} \;\times\; \mathit{input}$, where the error is in fact TD error and we use an eligibility trace for the input. The eligibility trace is updated with

$$e_i(t+\Delta t) \;=\; e_i(t) \;+\; (b_i(x(t)) \;-\; e_i(t)) \;\times\; \frac{\Delta t}{\kappa}, \tag{7}$$

where Δt is the time step and κ rate for the eligibility trace update. Finally, the exact update equation for the value function is

$$w_{c_i}(t+\Delta t) \;=\; w_{c_i}(t) \;+\; \alpha_c \;\times\; \delta(t) \;\times\; e_i(t) \;\times\; \Delta t, \tag{8}$$

where α_c is the learning rate for w_c.

Now we have to improve the policy μ using its associated value function. We take the greedy policy with respect to the current value function

$$u(t) \;=\; \mu(x(t)) \;=\; arg \max_u \left[r(x(t),u) \;+\; \frac{\partial V(x)}{\partial x}\; f(x(t),u) \right], \tag{9}$$

which takes into account the reward function, the system dynamics, and the value gradient. Under the hypothesis 1) that the reward $r(x,u)$ can be written $r(x,u) = R(x) - S(u)$ when $S(u)$ is a convex function, and 2) that the input gain $\frac{\partial f(x,u)}{\partial u}$ is not dependent on u, the condition for the greedy action is given by

$$-S(u) + \frac{\partial V(x)}{\partial x}\frac{\partial f(x,u)}{\partial u} = 0. \tag{10}$$

By definition the cost function is $S(u) = c_c \int_0^u s^{-1}(\frac{v}{u_{\max}})dv$, where $s()$ is a monotonically increasing output function, c_c the control cost coefficient, and $u_{\max}$ the maximal amplitude of action output. We obtain the following greedy policy,

$$u \;=\; u_{\max} \;\times\; s\; \left(\frac{1}{c_c} \;\times\; \frac{\partial f(x,u)}{\partial u}^T \;\times\; \frac{\partial V(x)}{\partial x}^T \;+\; \sigma n\;(t) \right), \tag{11}$$

where $\sigma n(t)$ is the noise for exploration.

2.1.2 The normalized version

We will also use the normalized version of this algorithm studied in [5]. We just add a scaling factor $\frac{1}{\tau}$ to the definition of the value function (cf. equation 3),

$$V^{\mu}(x(t)) = \int_t^{\infty} \frac{1}{\tau} e^{-\frac{s-t}{\tau}} r(x(s), u(s))\, ds \tag{12}$$

This factor is used for normalization, as $\int_t^{\infty} \frac{1}{\tau} e^{-\frac{s-t}{\tau}} ds = 1$. In this way, whereas for the non-normalized version, the scaling of the value function depends on the setting of τ, for the normalized version, the scaling of the value function is the same as that of the reward, irrespectively of τ.

There are two other changes implied by the new definition of the value function, concerning the TD error (cf. equation 5),

$$\delta(t) = r(t) - V(t) + \tau \dot{V}(t) \tag{13}$$

and concerning the greedy policy (cf. equation 11),

$$u = u_{max} \times s\left(\frac{\tau}{c_c} \times \frac{\partial f(x,u)}{\partial u}^T \times \frac{\partial V(x)}{\partial x}^T + \sigma n(t)\right). \tag{14}$$

As one can see, there are a number of metaparameters to adjust in both versions, such as τ, σ, α_c and c_c: we will focus on the discount factor τ.

2.2 Adaptation of τ

2.2.1 Why is it necessary to change τ ?

As you will see in the simulation section, section 3, there are problems in the learning wether the setting of the discount factor τ is appropriate or not. When τ is too small, the system only takes into account immediate reward and cannot achieve a goal that requires long preparatory steps. When τ is too large, the learning system, TD error (cf 5 or 13), is more dependent on the temporal difference in the predicted future reward $\dot{V}$ rather than the actual reward r, which could be undesirable when V has not been well learnt. The system becomes chaotic and unpredictable. In both cases, the learning tends to be stuck in a suboptimal behaviour.

Besides, the appropriate setting of τ depends on the environment and the task. Thus we should find an automatic way of adjusting τ, so that the system can learn without the intervention of a human operator.

2.2.2 Algorithm 1 for changing τ (non-normalized version)

We first consider an adaptation algorithm for the non-normalized version. We define the average absolute value of δ over a trial as

$$E[|\delta|] \;=\; \frac{1}{T}\int_0^T \mid \delta(t) \mid dt. \tag{15}$$

where T is the duration of a trial. We also define the variance of δ over a trial as

$$V(\delta) \;=\; \frac{1}{T}\int_0^T (\delta(t) \;-\; \bar{\delta})^2 \, dt, \tag{16}$$

where $\bar{\delta}$ is the average of δ over a trial. These variables are helpful in deciding wether τ is too large or too small. By observing the behaviours of system variables with different setting of τ (cf Fig. 1 and 2), we found that $E[|\delta|]$, measuring the reward prediction error, becomes small when the system improves and succeeds in its task: we decide to increase τ in this case so that the system can still evolve. On the other side, we use $V(\delta)$ as an indicator of the instability of the system. As you will see in 3.2, the variance $V(\delta)$ is high when τ is high. Thus, we propose the following algorithm for the non-normalized version,

```
if (V(δ) > θ_v)
    τ = τ − Δ_v / i
elseif (E[|δ|] < θ_e)
    τ = τ + Δ_e / i
else
    τ = τ
end
```

where i is the number of the current episode, θ_v and θ_e are respectively the thresholds for the variance and the average value, Δ_v and Δ_e are respectively the steps for the variance and the average value.

We tested different versions, for example by changing the conditions, their order and the thresholds, and the one presented here seems the most appropriate. One can also notice the decreasing step $\frac{\Delta}{i}$: we first used a constant step, but it was too messy and we couln't see any clear convergence.

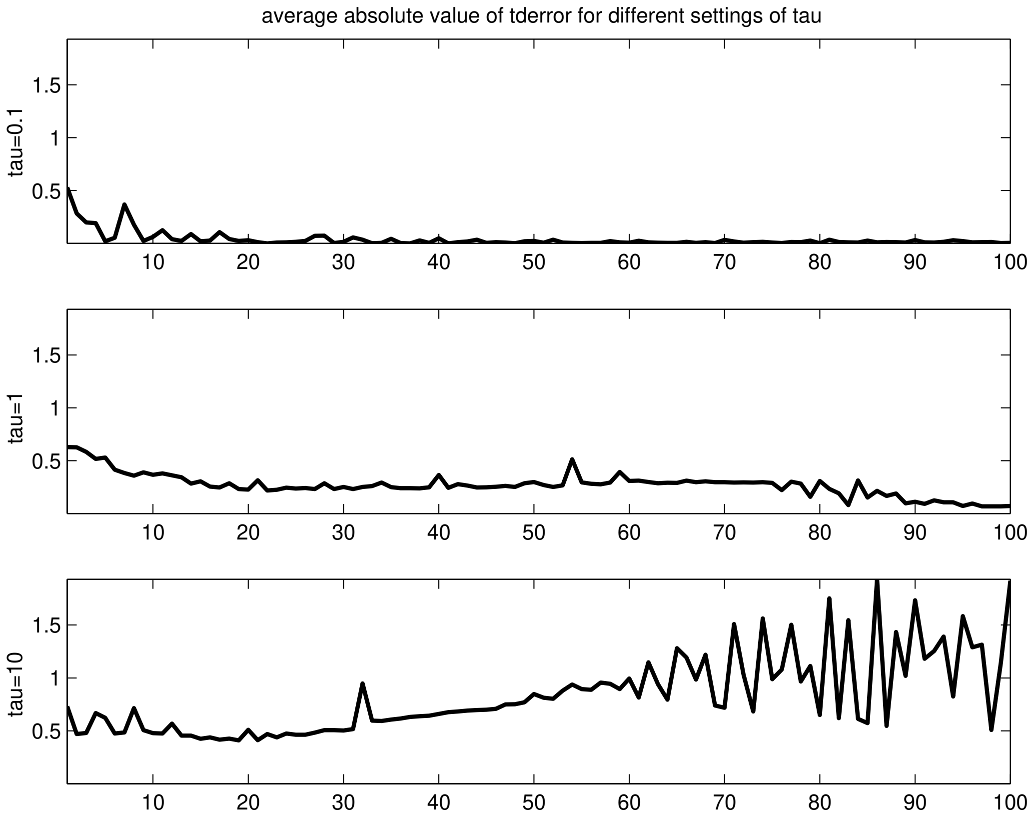

Figure 1: $E[|\delta|]$ evolves differently with the episodes of simulation for different settings of τ.

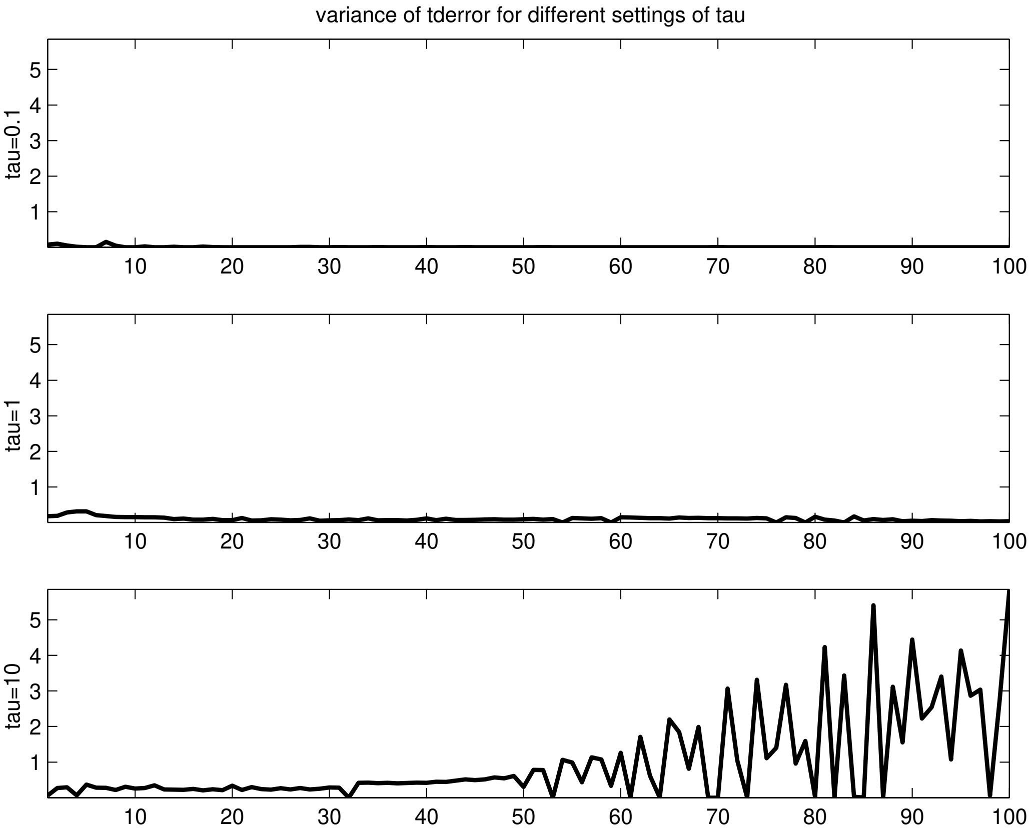

Figure 2: $V(\delta)$ evolves differently with the episodes of simulation for different settings of τ.

2.2.3 Algorithm 2 for changing τ (normalized version)

We then considered the normalized version. We first tried algorithm 1, presented in the preceding part, but the results were really bad, even though we tried different thresholds and steps. Therefore we looked for another algorithm, and we used different features of the system. The first was used previously, $E[|\delta|]$, the average absolute value of δ over a trial, and the second is the average reward over a trial,

$$E[r] = \frac{1}{T}\int_0^T r(x(t), u(t))\, dt. \tag{17}$$

As we said previously, $E[|\delta|]$ measures the reward prediction error, and $E[r]$ the current performance of the system.

The idea of the algorithm is explained by the figure 3. At the beginning, the reward is very low and the error is high, and the goal is that the reward becomes close to 1 and the error as low as possible. In these areas, τ is not evolving. The arrows represent the path the system is expected to follow. First, $E[|\delta|]$ remains to be high, and we decrease τ to make reward prediction easier. The reward is still relatively low and well-predicted ($E[|\delta|]$ small), then the system can be stuck in a suboptimal behaviour. To prevent this, we also decrease τ. The system can be oscillating between these 2 areas for some time (double-arrow), till it succeeds in jumping to the other increasing area. Afterwards, the goal is reached very quickly.

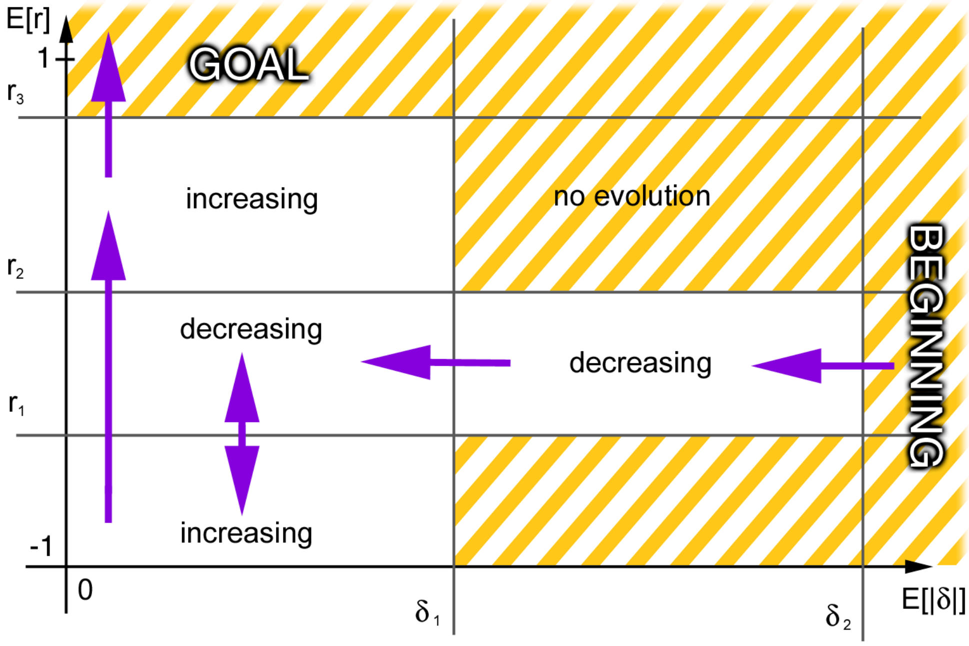

Figure 3: Explanation of algorithm 2. The arrows show the way the learning should follow. In the orange zone, τ doesn't evolve.

We can write algorithm 2 the following way,

if $(E[r] > r_3$ and $E[|\delta|] < \delta_2)$
  if $E[|\delta|] < \delta_1$ and $(E[r] > r_2$ or $E[r] < r_1)$
    $\tau = \tau + \Delta_{inc}$
  elseif $(\tau > 0)$
    $\tau = \tau \times \Delta_{dec}$
  else
    $\tau = \tau$
  end
end

We use a multiplicative decrease in τ to quickly reset a suboptimal behaviour.

3 simulation

3.1 Our system

The simulated system is a pendulum, of which physical parameters are given in the figure 4.

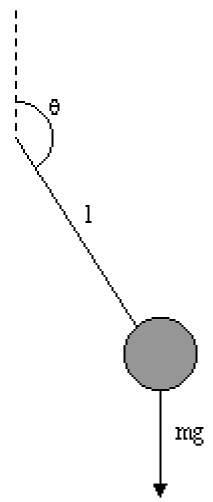

Figure 4: Control of a pendulum with a limited torque. The physical parameters are (in parenthesis are the chosen default settings of these values): mass m ($1\ kg$), length l ($1\ m$), gravitation g ($9.8\ m.s^{-2}$), friction factor μ ($0.01\ N.s.rad^{-1}$), the maximum value for the torque u_{max} ($5.0\ N$).

Thus set, the dynamics are given by,

$$ml^2\dot{\omega} = -\mu\omega + mgl\sin\theta + u,\ \omega = \dot{\theta} \tag{18}$$

The state of the pendulum is characterized by a two-dimensional state vector: $x = (\ \theta\ ,\ \dot{\theta}\ )$. The initial state is given by $x = (\ \theta(0)\ ,\ 0\ )$, where $\theta(0)$ is selected randomly in $[\ -\pi\ ;\ +\pi\ ]$.

Besides, to make the task more realistic, there is a control cost coefficient ($c_c = 0.1$).

In fact, the task is non trivial because of the limited torque: thus, if u_{max} is smaller than the maximal load torque mgl, it cannot succeed in reaching the top directly. It needs several swings to reach the top, and also to decelerate to stay at the top. Moreover, one trial lasts only $T = 20$ seconds.

The physical systems were simulated by a fourth order Runge-Kutta method, and the learning dynamics by an Euler method, both with the time step, Δt, of 0.01 second.

As we wrote it in section 2, the greedy policy we used is

$$u = u_{max} \times s\left(\frac{1}{c_c} \times \frac{\partial f(x, u)}{\partial u}^T \times \frac{\partial V(x)}{\partial x}^T + \sigma n(t)\right), \quad (19)$$

where $s()$ is a monotonically increasing output function : we chose here $s(x) = \tanh(x)$.

The value function was implemented by a normalized gaussian network with 15 $\times$ 15 basis functions to cover the two-dimensional state space. The reward was given by the height of the pendulum, say $R(x) = \cos(\theta)$.

You can have an idea of the typical behaviours of the system in figure 5.

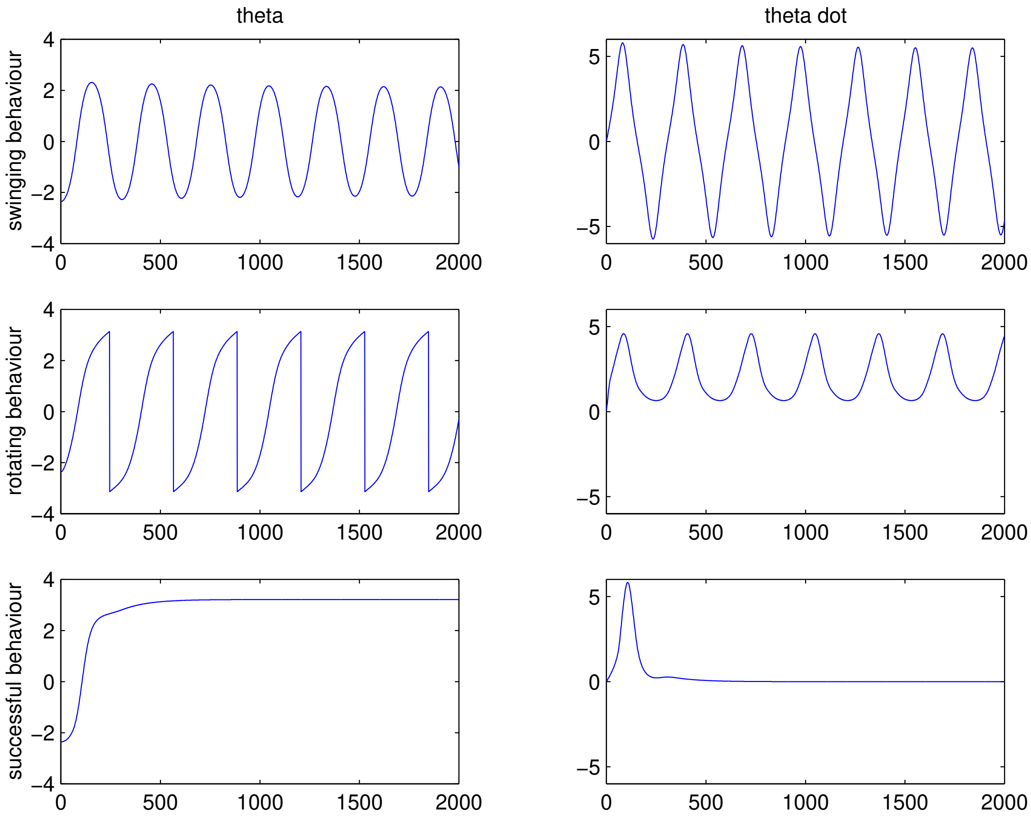

Figure 5: Typical Behaviours after 200 trials ($\alpha_c = 0.2$). The setting for the swinging behaviour was $\tau = 0.1$, for the successful behaviour $\tau = 1$ and for the rotating behaviour $\tau = 10$.

3.2 Different settings of the metaparameters

We first simply simulated our system with different settings of the metaparameters to find the more appropriate settings so that the learning of the system is successful and quick.

3.2.1 Non-normalized version

For each value of τ (0.01, 0.1, 1, 10, 100), we made 10 series of 100 trials. One way to evaluate the global performance of a series of trials is to measure the time spent by the pendulum up ($\mid \theta \mid < \frac{\pi}{4}$), which we denote as "time up".

We picture only 7 series to make the figures clearer, for the values of 0.01, 0.1, 1, 10 and 100 (see Fig. 6). It seems at first sight that the best value for τ is above 1. Indeed, under 0.2, the system doesn't succeed at all: the best it can do is swinging between 2 maximum positions. It improves then to the value of 1. Afterwards, the system seems to become better and better, but when we have a closer look at the results, we realize that it also becomes a lot more chaotic: for example, when the system succeeds once, that doesn't imply that he will succeed the next time, whereas it is usually the case for $\tau = 1$ or 2.

Thus, when we plot the variance of the 10 series for each τ, it becomes larger when τ increases above 2. Then it seems that the best value is around 1.

We can have a look at other evidences of success, like for example the value function (see Fig. 7). These figures confirm that for the smaller values of τ, below 0.5, the system only learns that the best values are for the bottom position when θ is small. Secondly, there are tiny differences between the value function for $\tau = 1$ and for $\tau = 10$ or 100: for $\tau = 10$ or 100, there are fewer states with intermediate values (green, yellow) between the extrema. Thus, when computing the next action to take, the system has fewer chances to reach these intermediate states, and can fail all of a sudden. Indeed when you have a look at the trials themselves, you cannot see why the system can succeed once and fail the following time: as we said, the system is more chaotic.

For the large values of τ, the system is in fact learning too slowly: we can correct this by increasing the learning rate α_c or making the experiment run a longer time. We let the system evolve for a longer time with $\tau = 10$ and during 200 and 500 trials (see Fig. 8). We clearly see that the system is improving, and that the value function looks more like the one for $\tau = 1$, and maybe even better. And when we look at the trials themselves, above 100

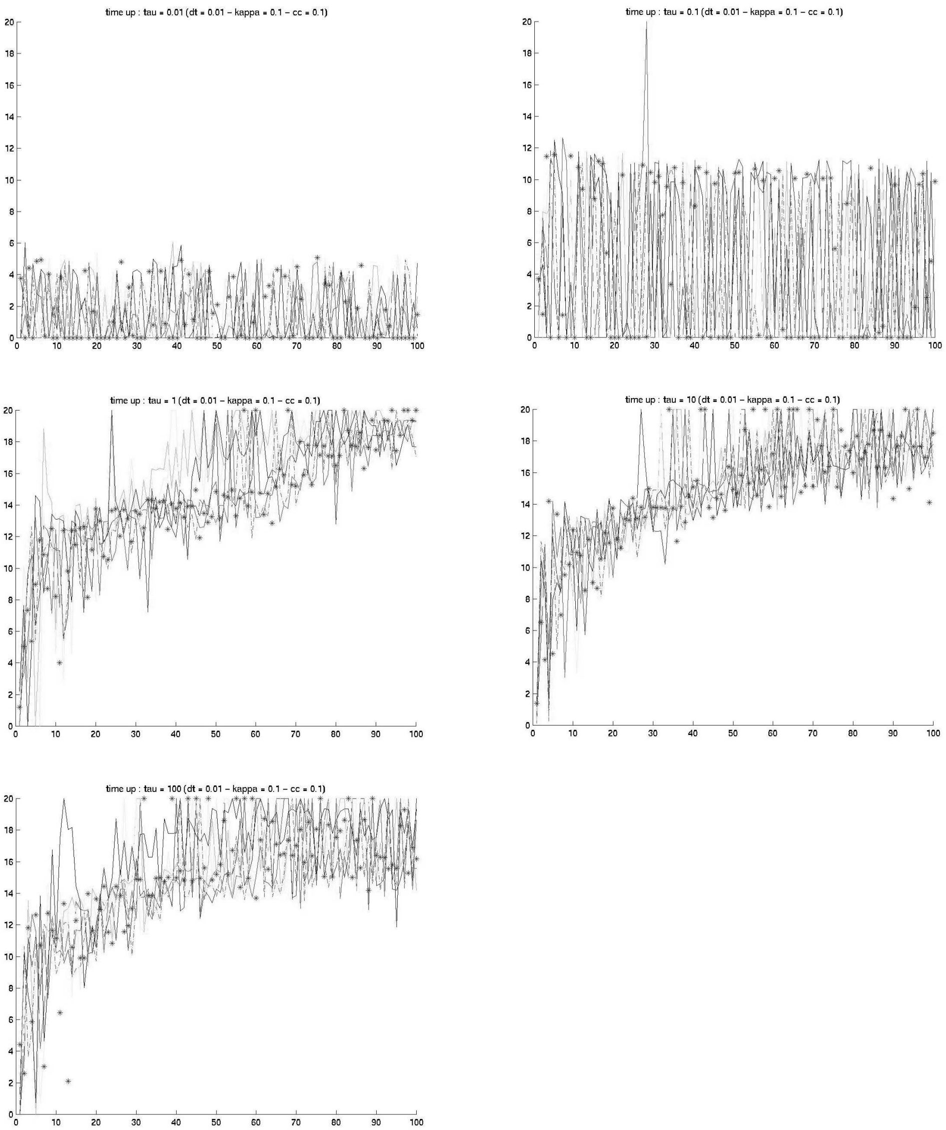

Figure 6: Time up for different values of τ. These episodes lasted 100 trials of 20 secondes.

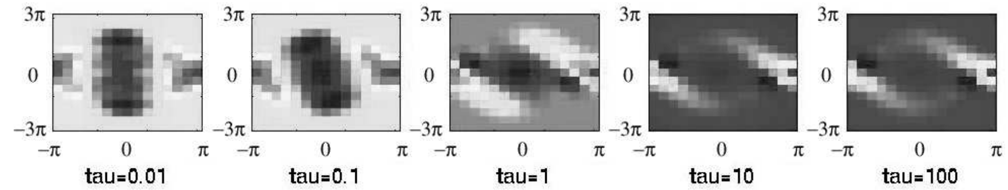

Figure 7: Value Function V(x), implemented by a 15 × 15 network for $\tau = 0.01, 0.1, 1, 10, 100$ (axis x: θ ; axis y: $\dot{\theta}$). The higher values are on the sides, in the tails, the smaller in the center.

or 150 trials they are successful, and not chaotic anymore. We can conclude that in fact the system was just learning too slowly.

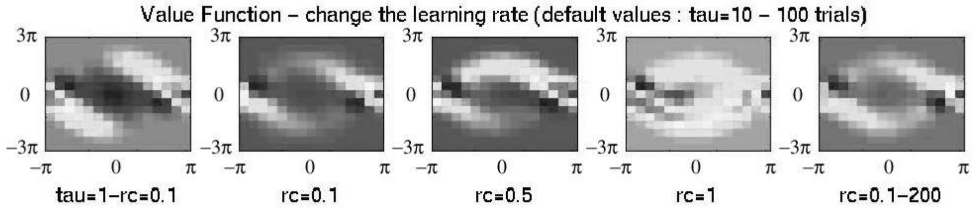

Figure 8: Value Function V(x). The default values were $\tau = 10$ and 100 trials. The first function is the reference, the three following ones show its evolution for different learning rate α_c and the last one lasted 200 trials with the default α_c (α_c is also noted as r_c).

The system needs more time to learn with $\tau = 10$ than with $\tau = 1$. But there is another parameter that surely controls the speed of learning, the learning rate of the value function α_c (see Fig. 8).

Indeed, for $\alpha_c = 0.5$ (default value 0.1), the system with $\tau = 10$ learns as quickly as the system with $\tau = 1$ and $\alpha_c = 0.1$ (the value function has a good shape, and the trials are successful). However, we have to warn that there are dangers in increasing the learning rate α_c, as shown by the value function created with $\alpha_c \geq 1$: the system learns quickly how to rotate, but not to stop at the top. The increasing of the learning rate α_c is one solution, but one has really to pay attention not to increase its value too much.

3.2.2 Normalized version

One could think that the learning of the system in the normalized version is comparable to that in the non-normalized version, but, due to the scaling of the value function, the behaviour of the system according to the setting of the metaparameters is different. For example, for $\tau = 10$, the system gets much more easily stuck in a suboptimal behaviour.

Let's have a look at the time-up for different values of τ (Fig. 9), at the value functions (Fig. 10) and at the ratio of the different final behaviours (Fig. 11).

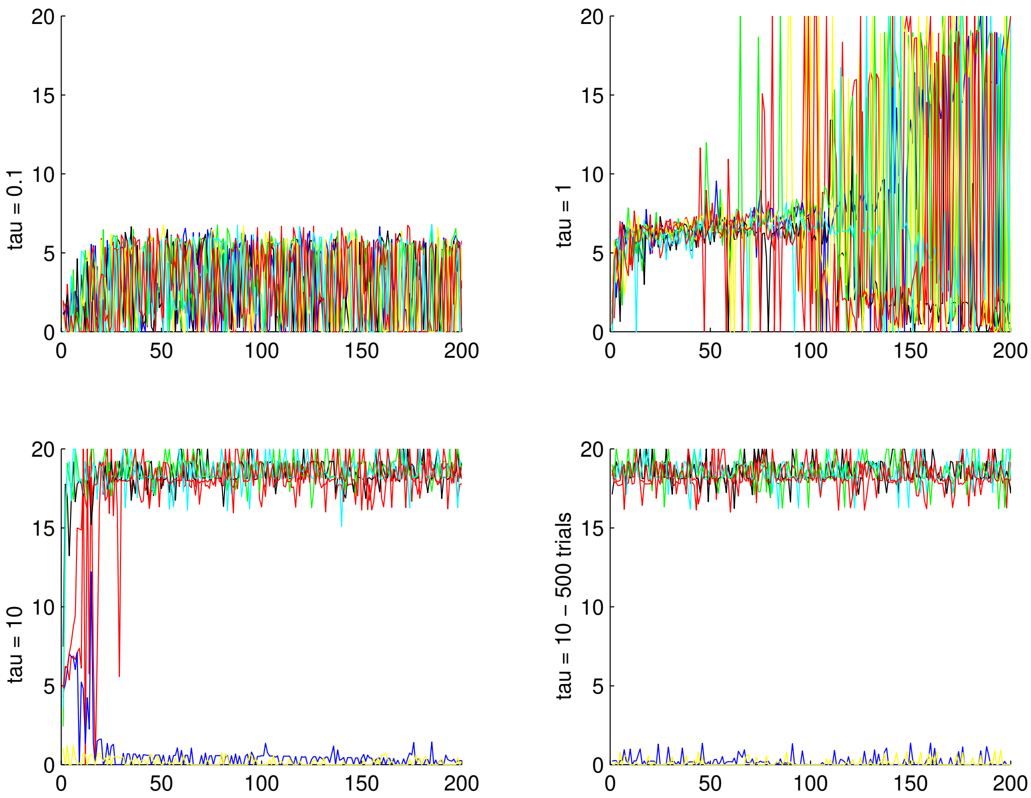

Figure 9: Time up for different values of τ, for the normalized version ($\alpha_c = 0.2$, 200 trials, except for the last, 500 trials).

The first remark is that the value of 1 still seems to be the more appropriate setting of τ for the best learning. The second is that the learning rate α_c should be a little higher than for the non-normalized version, around 0.2.

For the other values of τ, the results for a low one are the same as for the non-normalized version. However, for the high values of τ, around 10, the behaviours are different: the system is more willing to get stuck in a suboptimal behaviour, particularly the rotating one. Even when we increase the length of the experience, or when we increase the learning rate, the success rate doesn't get larger. This difference can account for the problems

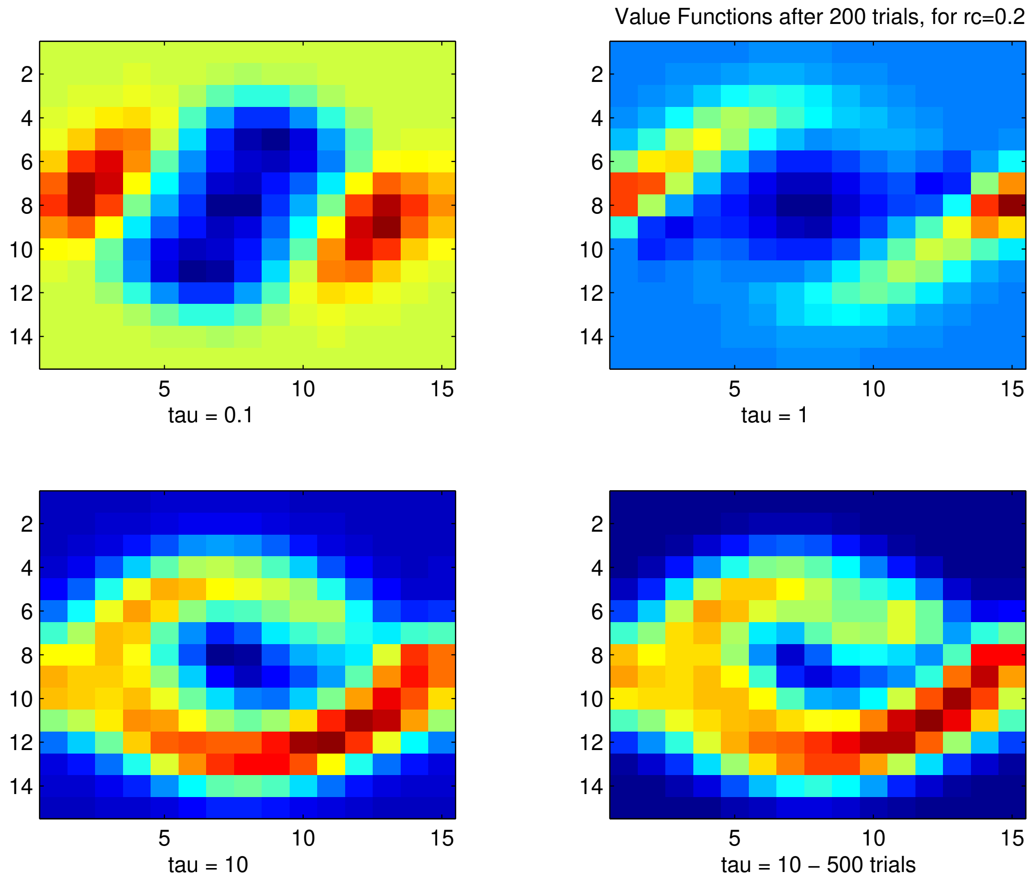

Figure 10: Value Functions for different values of τ, for the normalized version ($\alpha_c = 0.2$, 200 trials, except for the last, 500 trials).

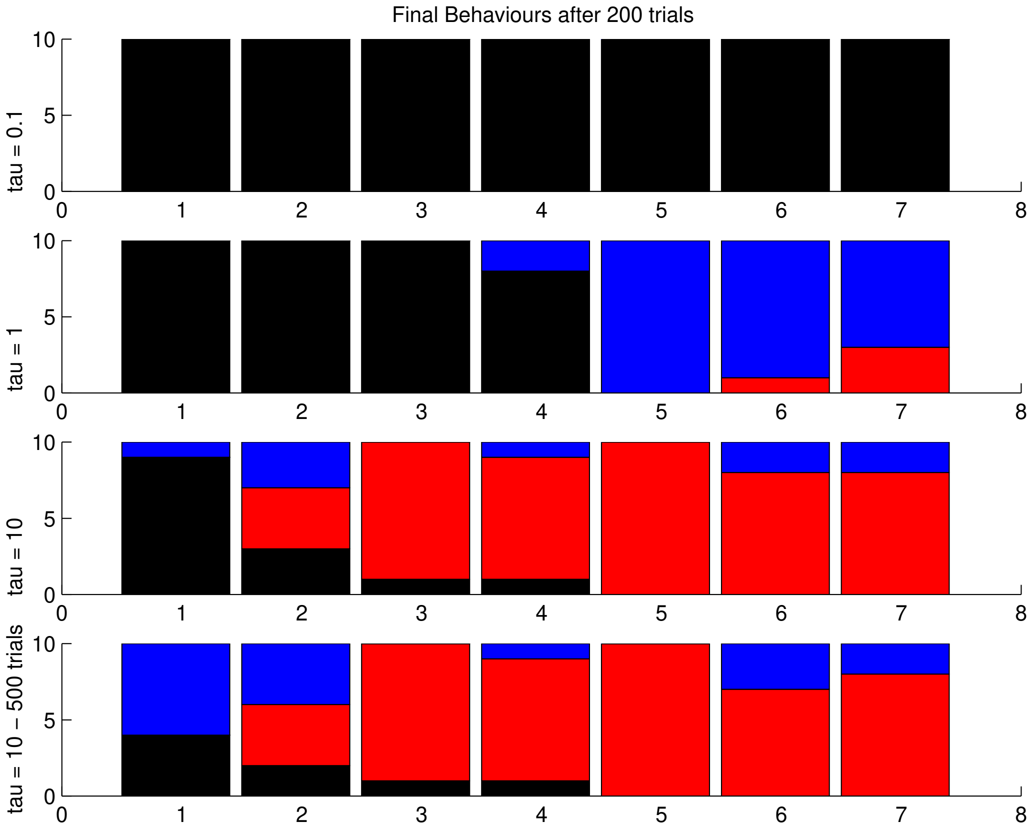

Figure 11: Ratio of the final behaviours for different values of τ, for the normalized version (200 trials, except for the last, 500 trials). In black is the ratio of the swinging behaviour, in lighter gray the successful behaviour and in the lightest the rotating behaviour. The values on the x axis correspond to the learning rate (1 is 0.01, 2 is 0.02, 3 is 0.05, 4 is 0.1, 5 is 0.2, 6 is 0.5 and 7 is 1).

we encountered when we tried to use algorithm 1 in the normalized version.

3.3 Evolution of τ

3.3.1 Non-normalized version

Now that we have an idea of the relationship between τ and the speed of learning, we can implement the algorithm 1 in the non-normalized version, explained in section 2.2.2. We can see the evolution for different initial values of τ and different lengths of the pendulum in the figures 12, 13 and 14.

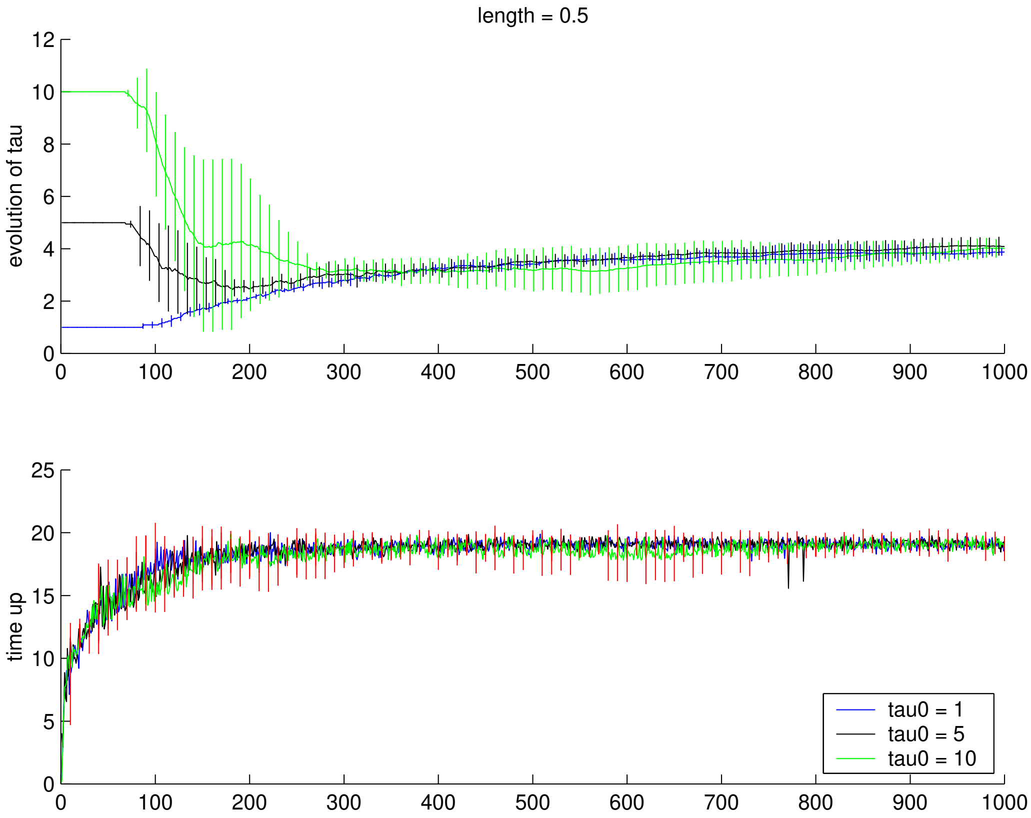

Figure 12: Evolution of τ and time up for the corresponding trials. We made 5 experiences for each setting of the initial value of tau and the length of the pendulum ($\tau = 1, 5$ and 10 - $length = 0.5$), lasting 1000 trials. We changed the maximal value of the torque as it directly depends on the length so that the goal-task remains as hard for the system: $u_{max} = 2.5$, 5 and 10, respectively for $length = 0.5$, 1 and 2 (the ratio is constant). We plotted for each setting only the mean curve, and the vertical lines are the square root of the variance.

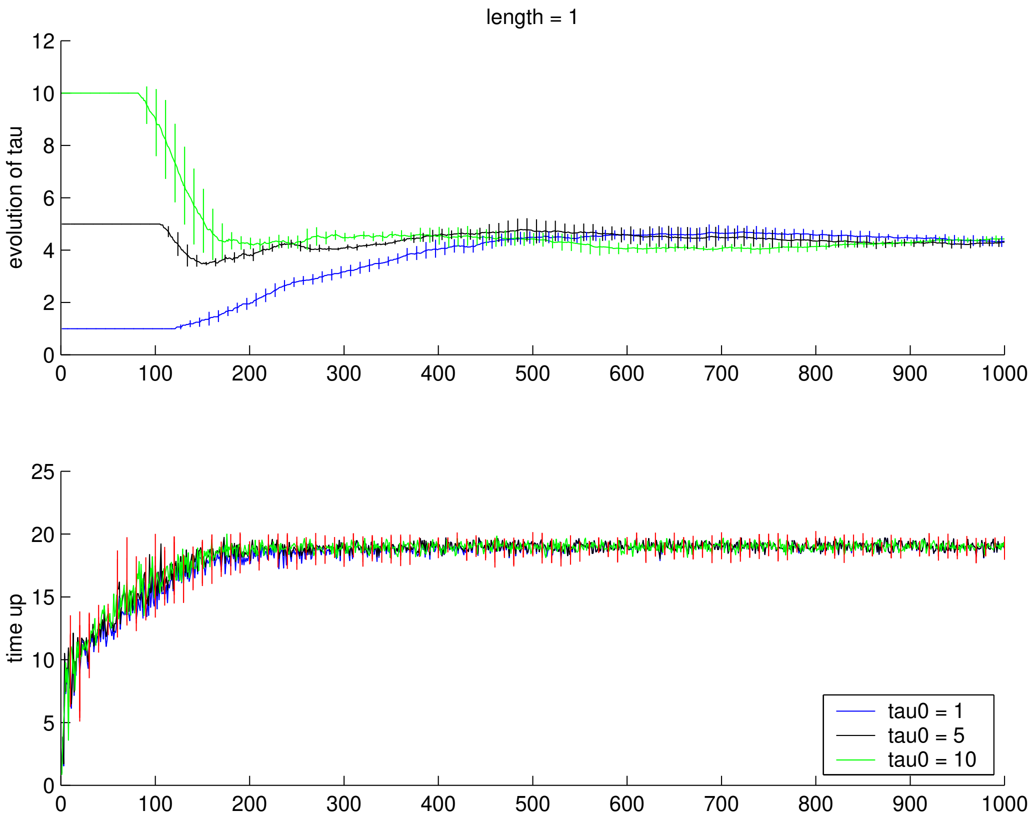

Figure 13: Evolution of τ and time up for the corresponding trials ($\tau = 1, 5$ and 10 - $length = 1$).

τ_0 \ length	0.5	1	2
1	3.88 $\pm$ 0.14	4.34 $\pm$ 0.21	5.51 $\pm$ 0.31
5	4.08 $\pm$ 0.34	4.31 $\pm$ 0.19	4.75 $\pm$ 0.15
10	4.01 $\pm$ 0.14	4.42 $\pm$ 0.14	4.73 $\pm$ 0.27

Table 1: final values of τ, according to the length and the initial value τ_0

Looking at the table 1, the first observation is that the final value doesn't depend a lot on the initial value of τ: so there seems to be convergence to a value characterizing the physical parameters of the system, at least the length of the pendulum. Besides, the convergence goes along with the success of the system, emphasized by the values of the so-called time up.

The second observation is that the limit value is different from one setting of the length to another: the higher the length, the higher the final value of τ. However, there is no direct proportionnality between the length and the

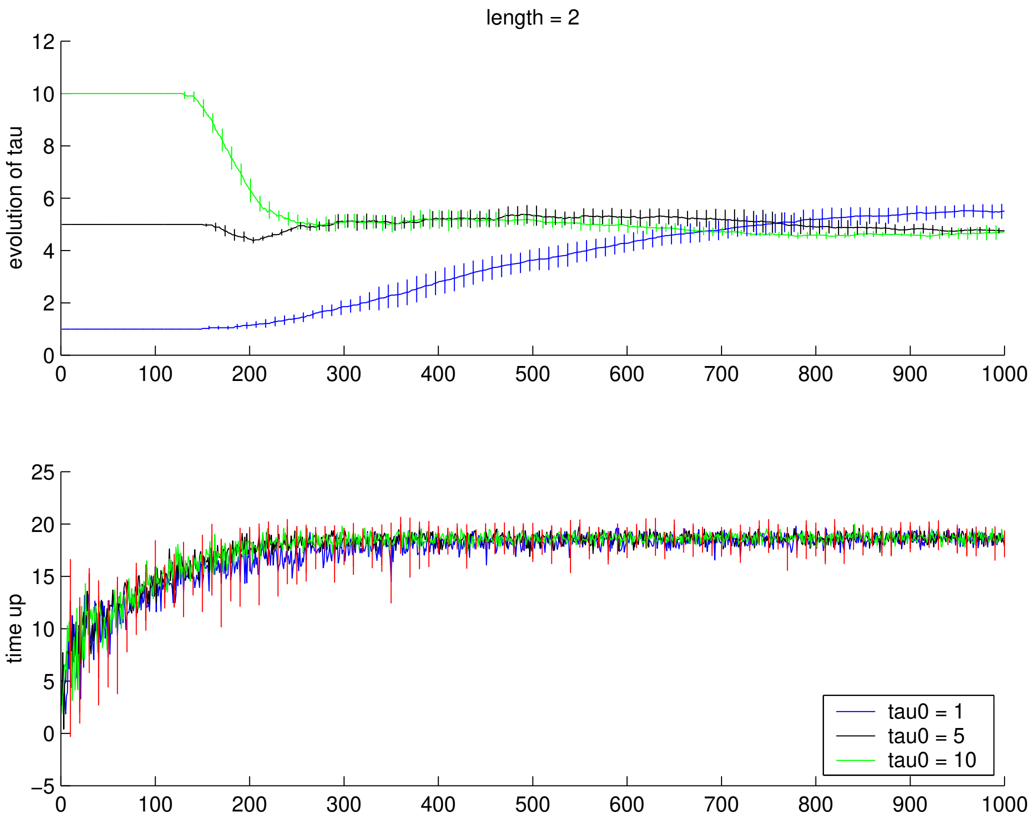

Figure 14: Evolution of τ and time up for the corresponding trials ($\tau = 1, 5$ and 10 - $length = 2$).

final value.

3.3.2 Normalized version

In the case of the normalized version, we implemented the algorithm 2 , explained in section 2.2.3. We made similar figures to the non-normalized case.

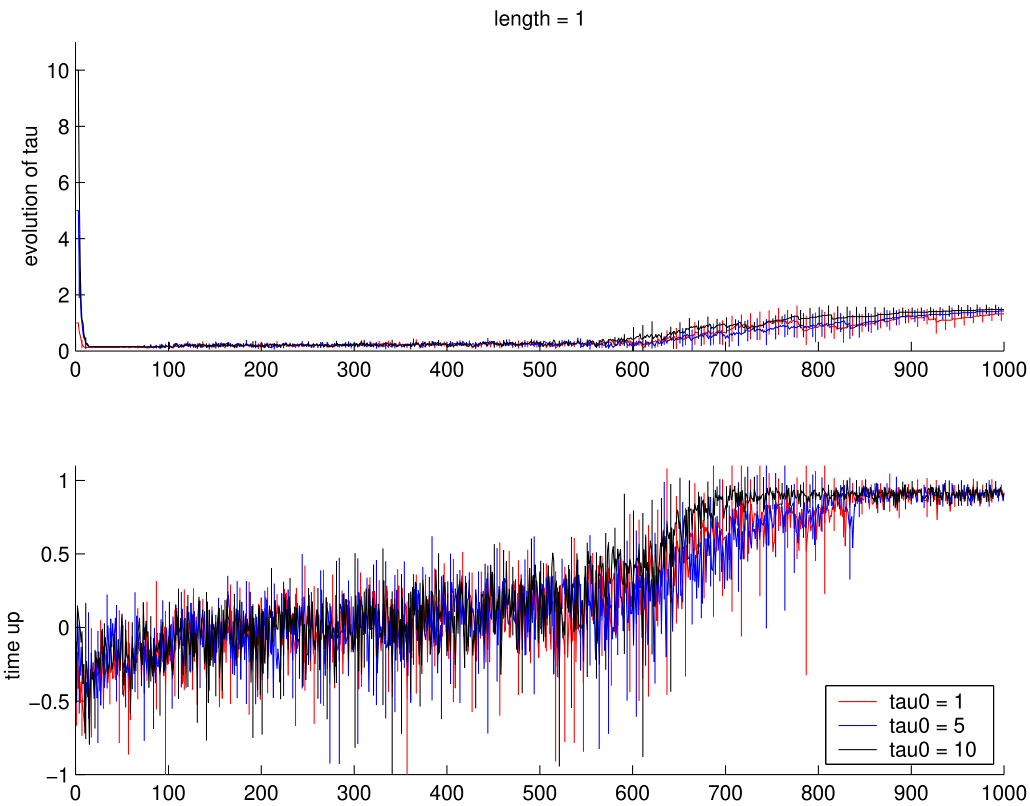

Figure 15: Evolution of τ and time up for the corresponding trials. We made 5 experiences for each setting of the initial value of tau and the length of the pendulum ($\tau = 1, 5$ and 10 - $length = 1$), lasting 1000 trials. We changed the maximal value of the torque as it directly depends on the length so that the goal-task remains as hard for the system: $u_{max} = 2.5$, 5 and 10, respectively for $length = 1$ and 2 (the ratio is constant). We plotted for each setting only the mean curve, and the vertical lines are the square root of the variance.

We can make the same remarks as for the non-normalized version. Looking at the table 2, the first observation is that the final value doesn't depend a lot on the initial value of τ: so there seems to be convergence to a value

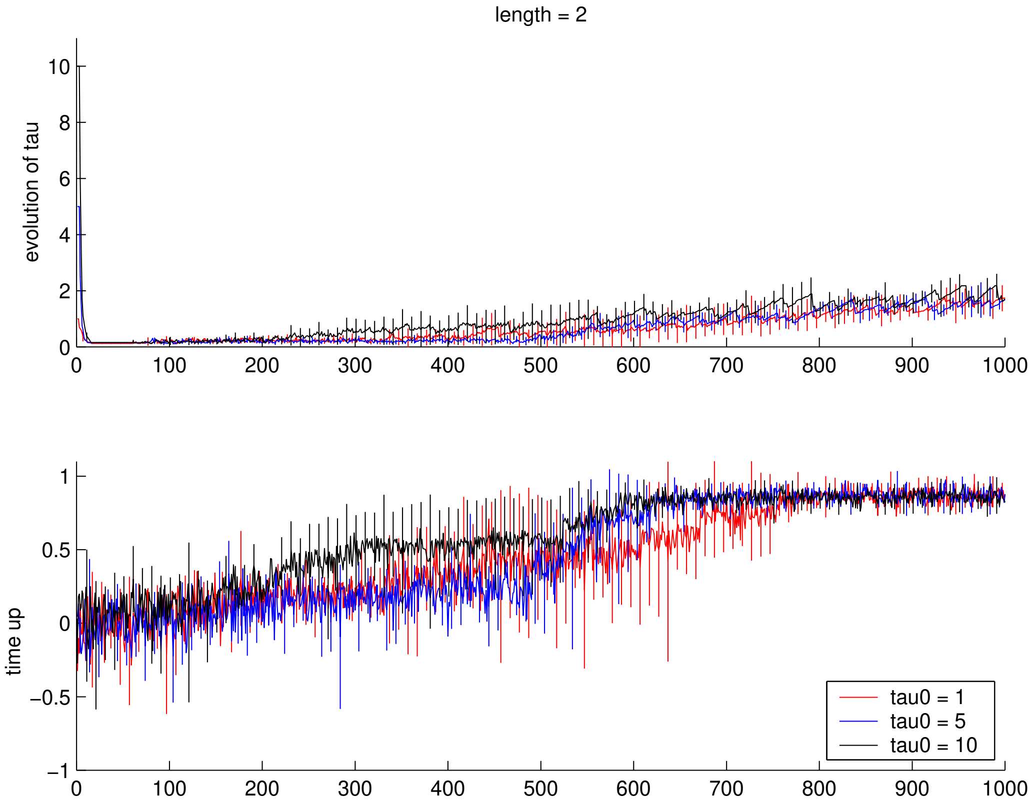

Figure 16: Evolution of τ and time up for the corresponding trials ($\tau = 1, 5$ and 10 - $length = 2$).

$\tau_0 \setminus$ length	1	2
1	1.32 ± 0.06	1.71 ± 0.25
5	1.42 ± 0.03	1.59 ± 0.04
10	1.48 ± 0.03	1.78 ± 0.10

Table 2: final values of τ, according to the length and the initial value τ_0

characterizing the physical parameters of the system, at least the length of the pendulum. Besides, the convergence goes along with the success of the system.

The second observation is that the limit value is different from one setting of the length to another: the higher the length, the higher the final value of τ. However, there is no direct proportionnality between the length and the final value.

4 Conclusion

4.1 Related Work

We didn't find many related works, concerning the adaptation of the meta-parameters of Reinforcement Learning. However, we can present at least one interesting works.

Richard S. Sutton developped a new algorithm, the Incremental Delta-Bar-Delta algorithm (IDBD), described in [9], for the learning of appropriate biases based on previous learning experience. This algorithm is also studied and extended in [7]. What's interesting in this algorithm is that it automatically adjust metaparameters, the learning rate parameters. He also shows that the system performs better with IDBD, which finds the optimal values of the learning rates. However, the algorithm was developped for the case of a simple, linear learning system, the least mean square or delta rule with a separate learning rate parameter for each input. When we tried to adapt it to our system, the result weren't satisfying. And we had to look for another method.

4.2 Discussion

As we said in the introduction, metaparameters such as τ have to be tuned to each environment. When changing the environment, systems are often unsuccessful because their metaparameters aren't adjusted.

We think that, according to the results, we have shown here that the parameter τ is controlling, at least partially, the speed of learning of the system. We also showed that we can use the fact that the system is unsuccessful to change the metaparameter τ, what will then improve the learning and so on. We tested two evolving algorithm can be a solution, at least for the evolution of the discount reward factor. The tuning of the metaparameters in reinfocement learning is a complicated problem, but we hope our methods can be generalized, or at least pave the path for future work. What can we say for other metaparameters ?

For the tuning of the noise, we think that it should be high when the system has to explore all its possible states to find the optimal solution, that is to say at the beginning of the learning and when the envionment changes, and low the rest of the time as it always makes the action a little inaccurate. We could for example make the amplitude of the noise decrease with the number of episodes, and set it high when a sudden change in the performances of the system is detected, then decreasing it again.

Another kind of important metaparameters is the learning rate, for example the learning rate α_c of the value function we changed in section 3. As we saw it, there must be a high bound for its value. Besides, we think that we could have the same kind of algorithm as the one used for τ, as they both influence the learning of speed, but we haven't made any tests.

As stated in [2] and [3], many hypotheses can be made about a relationship between the metaparameters and the neuromodulators, based on a large body of neurobiological data and computational modelling studies. For example, the serotonergic system seems to control the discounting factor τ, and the acetylcholine the learning rate. Thus, our neuromodulators, and our emotions consequently, could control our ability to learn and to adjust to our environement, playing the role of a metalearning system.

References

[1] Dimitri P. Bertsekas and John N. Tsitsiklis, *Neuro-dynamic Programming*, Athena Scientific, 1996.

[2] Kenji Doya, *What are the computations of the cerebellum, the basal ganglia and the cerebral cortex ?*, Neural Networks 12 (1999) 961-974.

[3] Kenji Doya, *Metalearning, neuromodulation, and emotion*, Affective Minds, Chapter 10, 2000.

[4] Kenji Doya, *Reinforcement Learning in Continuous Time and Space*, Neural Computations 12, 219-245 (2000).

[5] Kenji Doya, *Temporal Difference Learning in Continuous Time and Space*, in Advances in Neural Information Processing Systems 8, pages 1073-1079, MIT Press (1996).

[6] Stefan Schaal and Christopher G. Atkeson, *Constructive Incremental Learning from Only Local Information*, Neural Computation 10, 2047-2084 (1998).

[7] Nicolas Schweighofer and Michael A. Arbib, *A Model of Cerebellar Metaplasticity*, Learning and Memory 4, 421-428, 1998.

[8] Richard S. Sutton and Andrew G. Bartow, *Reinforcement learning, an introduction*, The MIT Press, 1998.

[9] Richard S. Sutton, *Adapting Bias by Gradient Descent: An Incremental Version of Delta-Bar-Delta*, Proceedings AAAI-92.