Reinforcement-Learning Approach Guidelines for Energy Management

Yohann Rioual, Johann Laurent, Jean-Philippe Diguet

Reinforcement-Learning Approach Guidelines for Energy Management

Yohann Rioual*, Johann Laurent[2], Jean-Philippe Diguet[1]

[2]UMR CNRS 6285 – Lab-STICC, University of Bretagne Sud, F-56100 Lorient, France,
johann.laurent@univ-ubs.fr

[1]UMR CNRS 6285 – Lab-STICC, University of Bretagne Sud, F-56100 Lorient, France,
jean-philippe.diguet@univ-ubs.fr

*corresponding author: Yohann Rioual

Address:

University of Bretagne Sud

UMR CNRS 6285 – Lab-STICC

Saint-Maudé Street

F-56100 Lorient, France

Office : +33 (0)2 97 87 45 60

Fax : +33 (0)2 97 87 45 27

Email : yohann.rioual@univ-ubs.fr

Date of Receiving: **to be completed by the Editor**

Date of Acceptance: **to be completed by the Editor**

Reinforcement-Learning Approach Guidelines for Energy Management

Yohann Rioual, Johann Laurent, Jean-Philippe Diguet

Abstract– IoT and autonomous systems are in charge of an increasing number of sensing, processing and communications tasks. These systems may be equipped with energy harvesting devices. Nevertheless, the energy harvested is uncertain and variable, which makes it difficult to manage the energy in these systems. Reinforcement learning algorithms can handle such uncertainties, however selecting the adapted algorithm is a difficult problem. Many algorithms are available and each has its own advantages and drawbacks. In this paper, we try to provide an overview of different approaches to help designer to determine the most appropriate algorithm according to its application and system. We focus on Q-learning, a popular reinforcement learning algorithm and several of these variants. The approach of Q-learning is based on the use of look up table, however some algorithms use a neural network approach. We compare different variants of Q-learning for the energy management of a sensor node. We show that depending on the desired performance and the constraints inherent in the application of the node, the appropriate approach changes.

Keywords– Energy management, reinforcement learning, neural network, Q-learning

1 INTRODUCTION

Interest for diffuse and automatic observation of the complex physical and biological phenomena in various areas is on the rise. Thus, sensor nodes are becoming more complex in order to meet an increased need for accurate environmental observations. These nodes include sensors, processing capabilities and communication unit. Nevertheless adding new sensors increase the processing and communication needs, which results in an increase of energy consumption (Figure 1). However the battery's technologies are not evolving fast enough to meet the need of energy. So, these systems often include harvesting capabilities.

These harvesting capabilities rely on the energy the system can harvest in its environment. Different technologies exist using human motion, heat or solar radiation, and the choice of a technology depends mostly of the context, but also of the energy the node needs to harvest. The solar panel technology is the most efficient way to harvest energy (up to $15\ \mathrm{mW/cm^2}$). The harvesting capability of a solar panel depends of ageing of the components [1] and of the weather. Therefore, the harvested energy greatly varies over the day, and increases the uncertainty of such systems. Reinforcement Learning (RL) can handle such uncertainty and so is a promising candidate method to provide the sensor nodes with the ability to adapt according to the available energy.

The interest in RL has arisen with the success in various domains such as Atari game, robotic control or energy management. Nevertheless, there is a multitude of reinforcement algorithms and all of them are not suitable for implementation on embedded systems such as sensor node. Challenges and hindrances to overcome are numerous. Moreover, these algorithms may require a significant amount of computing capabilities and memory. There are two main approaches used, one using a lot of memory which stores information in a look up table and another one which trains a neural network with the information using a lot of processing. The choice of the approach to select when designing an energy management for an application is not obvious. Thus, in this paper, we provide a comparison of different RL approaches to help designers to choose the most appropriate one for a specific application. This paper addresses the question of the algorithm choice for a given but classical real case. We illustrate the comparison with the use case of a marine buoy equipped with solar panels to complement its battery.

This paper compares several versions of a well-known reinforcement learning algorithm, which uses either look up table or neural network. The rest of the paper is organized as follows : Section 2 provides an overview of the background and state of the art works on the use of reinforcement learning algorithms in the field of energy management, we also present our use case: the marine buoy. Section 3 presents the different algorithms and the performances of each on energy management of our system. Section 4 presents a comparison of the different algorithms. And finally, Section 5 concludes the paper by summarizing the relevant features of this work.

2 OVERVIEW

Reinforcement Learning (RL) [2] is a framework to optimize the behaviour of an agent, or a controller that interacts with its environment. First, this section presents the RL, then, related works from the state of the art using RL algorithms to optimize the energy consumption of cyber-physical systems are presented. Finally, our use case, a marine buoy, is presented.

2.1 Background on Reinforcement Learning

RL is a formal framework that models the problem of sequential decisions, in which an agent learns how to take better decisions by interacting with its environment (Figure 2). When the agent performs an action, it receives as a feedback the new state of its environment and a reward signal, encoding the information on the quality of the transition. The agent's objective is to maximize its long term reward.

RL algorithms can be used to solve optimization problems that can be formulated as Markov Decision Process (MDP). An MDP is a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ is a set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1[$ are the probabilities of state transitions and $\mathcal{R}$ is a reward signal.

The agent follows a policy $\pi : \mathcal{S} \times \mathcal{A}$ which determines the agent's behaviour. An optimal policy π^* maximize the long term reward, i.e. :

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi) = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} \left(\gamma^t \times \mathcal{R}(s_t, \pi_t(s_t))\right)\right] \tag{1}$$

where t represents a time step and $0 < \gamma < 1$ is a discount factor. In the RL framework, the aim is to learn an optimal policy when the model, $\mathcal{T}$ and the reward $\mathcal{R}$, is unknown.

The discount factor γ determines the importance of future rewards. A factor of 0 makes the agent myopic by considering only current rewards, while a factor close to 1 implies more distant rewards. If the discount factor is close to 1, without a terminal state, or if the agent never reaches one, all environment histories become infinitely long, and Q-values with additive, undiscounted rewards generally become infinite.

2.1.1 ϵ-greedy Policy

A major issue in RL is the dilemma between exploration and exploitation. Exploration chooses an action randomly in the system to find out the utility of that action. Whereas exploitation deals with the actions which have been chosen based on the previously learned utility of this action. We encourage the algorithm to explore more during the beginning of the training period, and then slowly decrease the exploration to exploit the learned informations. With sufficient training, the algorithm should converge to its optimal policy. However, acting greedily before convergence may lead to sub-optimal policies because the agent would not have had the opportunity to sample state-actions pairs that might have led to higher returns. In order to avoid this, we follow an ϵ-greedy policy. This means that the actions are chosen greedily most of the time, but with probability ϵ, a random exploratory action is selected.

A heuristic used to calculate the probability of exploration at any point of time is given in [3] :

$$\epsilon = \min(\epsilon_{max}, \epsilon_{min} + k \times (S_{max} - S)/S_{max}) \tag{2}$$

where ϵ_{max} and ϵ_{min} denote upper and lower boundaries for the exploration factor, respectively. S_{max} represents the maximum number of states and S represents the current number of states already known. At each time step, the system calculates ϵ and generates a random number in the interval $[0, 1]$. If the selected random number is less than or equal to ϵ, the system chooses a uniformly random task (exploration), otherwise it chooses the best task using Q-values (exploitation).

2.2 Related work

Reinforcement learning approach has been applied in a variety of schemes such as routing, resource management or dynamic channel selection in wireless networks. Indeed RL allows systems to observe, learn and respond to its complex and dynamic operating environment in an efficient manner. According to the problem, different algorithms are used : Q-learning, TD(λ), SARSA(λ), Deep Q-learning, temporal difference algorithm.

The most popular RL algorithm in the literature is the Q-learning. It has been applied successfully in various applications. In the context of radio optimization [4] proposes an adaptive media access control (MAC) protocol. In this case, it adapts the communication of a sensor node depending on the number of data to be transmitted and the energy available. [5] proposes on-line power management technique with Q-learning for peripheral devices and microprocessor. The idea is to adapt the energy consumption according to the workload with no prior information since devices have different operating behaviours and performance evaluation. Their techniques adjust on the fly the energy consumption and take into account uncertainties that emanate from hardware and application characteristics. The use of Q-learning algorithm does not limit to the optimization of the energy consumption. [6] presents a prediction algorithm for the energy generation from harvesters. Since embedded systems have limited-energy source. However, inevitable energy depletion will eventually disturb the system's operation. Solar energy is the most effective environmental energy for energy harvesting because of its high energy intensity, nevertheless it comes from a non-controllable source,

the sun. The proposed method predicts the system's energy that will harvest to adjust the energy consumption accordingly.

The algorithm can also be distributed among several agents to allow them to cooperate, to compete or something in between. [7] proposes a distributed Q-learning to deal with the problem of aggregation of interference generated by multiple cognitive radio at the passive digital TV primary receivers. The authors tested this approach with a look up table to store the expected reward for each possible action or a neural network version which compute the expected reward. The comparison shows that according the application, a designer might prefer using more memory with a look up table or do more computation with a neural network.

When an agent performs an action with Q-learning, it receives a reward that evaluates the efficiency of the action. However an agent may receive a reward only after performing a sequence of actions, assigning credit to the appropriate state-action pairs becomes an issue. To resolve this, different algorithms introduce a memory variable for each state-action pair called the eligibility trace. During each epoch, the eligibility trace for all state-action pairs decays by $\gamma\lambda$ where λ, $0 < \lambda < 1$, is a parameter that allows to specify the strength with which Q-values of early state-action pairs are updated as a consequence of the final reward. The authors of [8] use this approach to propose on-line power management technique for peripheral devices and microprocessor. The idea is to adapt the energy consumption according to the workload with no prior information. Indeed devices have different operating behaviours and performance evaluation. Their technique adjusts on the fly the energy consumption and takes into account uncertainties that emanate from hardware and application characteristics. Moreover, the authors add a workload prediction based on on-line Bayes network to improve the performance of their algorithm. In order to maximize a node's lifetime, the node is equipped with one or more energy harvesting devices, enabling the nodes to be entirely powered by the energy harvested in their environments.

Energy Neutral Operation (ENO) is a mode of operation where the energy consumption of the node is always at most the energy than the energy harvested from the environment. In order to achieve energy neutral operation, energy optimisation methods need to fulfil the energy neutrality constraints while maximising performance. In [9], the author uses an other RL algorithm, SARSA(λ), to achieve a ENO power management of a sensor node in a monitoring application. It uses the previously harvested energy, which is stored in the energy buffer (battery, capacitor, . . .) and the weather prediction to determine the different possible actions in order to achieve a ENO. It achieves less than 6% root mean square deviation from ENO as compared to more than 23% deviation that occurs when using other approaches, and a reduction by almost 4 of the difference between the energy collected and the energy consumed.

Some different approaches use neural networks to reduce the energy consumption. For instance, [10] presents a method to reduce the energy consumption of the front-end radio using neural networks. It proposes a real-time channel-adaptive system which is able to change its power consumption according to the desired level of Quality of Service. The key objective of learning based adaptation is to determine the optimum tuning knob combinations for the front-end for every channel's state on-the-fly. The experiments show up to 2.5 times savings in energy consumption compared to a worst-case design method. And [11] uses the Deep Q-learning, a neural version of the popular Q-learning to take routing decision for packets in underwater acoustic sensor network (UASN) in different topologies. By combining this approach with hybrid broadcast and unicast communication mechanisms, the author achieves highest energy efficiency which results an improving lifetime for the network by $34 - 36\%$ compared to others approaches without reduction of the latency.

There are plenty other approaches using RL algorithms, which do not use neural network or Q-learning. In [12], the author propose an algorithm using temporal difference learning with linear function approximation. This algorithm called RLMan dynamically adapts energy management policy to time-varying environment. The author successfully applied it to the Pow Wow platform [13], a wireless sensor node with harvesting capabilities. The algorithm has been tested with two different sources of energy, indoor light and outdoor wind. The average throughput is 70% higher than with state-of-the-art algorithms which it is compared. And [14] applied RL algorithm to device-to-device communication for cellular networks. Device-to-device communications help to improve cellular networks by reusing the spectrum resources. Nevertheless, it provides interferences in the system while reusing the resources. The authors propose an adaptive power allocation method using a bandit arm solver and provide an efficient interference management system. The

D2D throughput increased by 28% as compared with the distributed reinforcement learning.

RL is widely used to optimize the energy consumption. There are different algorithms available and different approaches (look up table or neural network). However, the main challenge remains to find which approach to use and how to define the MDP. In this the paper we explore this challenge while considering a marine buoy case study presented in the following section.

2.3 Marine buoy case study

Monitoring the marine environment is an important and difficult task. Sea wave height, water temperature, atmospheric pressure, and wind speed used for many practical applications, are usually obtained from three sources: buoy measurements, model calculations, and ship observations. Compared to other data acquisition methods, buoy measurements are the most reliable and readily data source available continuously for years. Therefore, in this paper, we consider a marine buoy near the coast communicating with a base station. The buoy is equipped with two sensors, an anemometer 3D and atmospheric sensor, to monitor environmental conditions. The sensors have different energetic behaviours (Table 1) and we want the buoy to be deployed as long as possible. In order to make it simple, data are sent immediately without processing. Since solar radiation is the most effective environmental energy for harvesting, the buoy is equipped with solar panels. To avoid collision with boats, a beacon light flash every 4 seconds and during 0.5 seconds when the brightness is low.

The aim of the use case is both to extend the buoy's lifetime and to maximize the measurement done by the sensors. We minimize the set of actions that the system can take to keep it simple. So, the buoy can adjust the sampling frequency of its sensors in order to preserve the battery's energy. Thus, we use a RL algorithm to control the sampling frequency of the sensors and to choose the correct frequency in order to preserve the battery while maximizing the performance. The measured data are transmitted immediately, there is no retention. Obviously, this approach is suboptimal [15]. The idea of this paper is to give a guideline to select the appropriate RL approach for the energy management of a sensor node. The MDP presented here needs some adaptation to be in accordance for the desired performance in the specification of your application.

We define our MDP as follows : a state space $\mathcal{S}$ of 10 states (for each 10% increment of the battery charge), the set of actions $\mathcal{A}$ is the different operating modes our sensors are allowed to choose (i.e. a sampling frequency (Hz) in $[0.1, 0.2, \cdots, 0.9, 1]$). The reward function indicates what kind of behaviour best serves our objective. In our RL model, the reward awarded at the end of an episode depends on the residual battery energy and the sampling frequency of our sensors during the episode. We reward our system using a simple function :

$$\eta \times \mathcal{N}(\text{frequency}) + (1 - \eta) \times \mathcal{N}(\text{battery charge}) \tag{3}$$

where $\mathcal{N}$ is the function that normalizes the values. $\eta \in [0, 1]$ is a parameter which balances the importance of the battery charge and the sampling frequency. It's really important for the reward function to be deterministic and bounded, otherwise the algorithms will never converge.

We conduct several simulations for three different algorithms, Q-learning, Dyna Q-learning and Deep Q-learning. The principle of these algorithms is the subject of the next section. The production from solar panels is estimated using the surface power coming from sun radiations. It uses state-of-the-art model for photovoltaic panel [16]. The photovoltaic panel's temperature is influenced by the solar irradiation, the ambient temperature and the wind velocity. Climate data come from the meteorological station at Lorient, France. It includes an archiving of wind (angle, speed) and ambient temperature. A sunshine model is also implemented to estimate the surface power of the sun radiations. It depends on the date and node position on the globe. All the details can be found in [17]. Figure 3 shows the evolution of the energy the solar panels harvested during 3 weeks in summer.

The following section presents the reinforcement learning algorithms we use for the energy management of our node and the results of simulations.

3 REINFORCEMENT LEARNING ALGORITHMS FOR ENERGY MANAGEMENT

The choice of the correct RL algorithm to use is a challenging problem. Indeed, there are many algorithms available and it is not easy to find the best for a target application. In this section, we explore different algorithms with a look up table or with a neural network, and we try to propose a methodology to guide the designer choice. Here, we investigate only the field of energy management.

3.1 Algorithms using a look up table

When we use an algorithm with a look up table, we use this table to store the expected future reward for the selected action in a certain state (Figure 4). These values are called Q-values. In this paper, we present 2 algorithms which use this approach to learn, the Q-learning and the Dyna Q-learning.

3.1.1 Q-learning

The Q-learning (Algorithm 1) is a well-known and popular RL algorithm [18] due to its ease of implementation and to its convergence which is mathematically proven. In this section, we first present the algorithm and in a second time the results achieved on the energy management for our case study.

The Q-learning reinforcement function is based on the value iteration algorithm. The aim of this algorithm is to optimize the evaluation function for each state-action pair (s, a) in order to deduce an optimal control strategy. For this purpose, Q-learning uses the temporal difference principle [19] to update the evaluation function. The temporal difference corresponds to the difference between two successive estimates of a couple's gain expectation.

The Q-value associated with the state s_t and the action a in the look up table is updated with the following equation :

$$Q(s_t, a) = Q(s_t, a) + \alpha \left(r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a) \right) \tag{4}$$

In equation 4, the term $r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a)$ corresponds to the difference between the new and the old estimation of $Q(s_t, a)$, this is the temporal difference.

Learning rate α : The learning rate α determines how fast the new information will surpass the old one. A factor of 0 would not teach anything to the agent, whereas a factor of 1 would only teach to the agent with the latest information. In our work, we decrease slowly the learning rate α in such a way that it reflects the degree to which a state-action pair has been chosen in the recent past. It is calculated as:

$$\alpha = \max \left(\frac{\zeta}{visited(s, a)}, 0.1 \right) \tag{5}$$

where ζ is a positive constant and $visited(s, a)$ represents the visited state-action pairs so far [20].

A good optimization requires to choose a correct value for the discount factor γ and for the duration of each action. We already have a heuristic (Equation (5)) to set up α. We set γ to 0.8 to consider the expected reward to a near future. Indeed, the deployment length is a critical limitation for a monitoring application, so we anticipate in the long term. And after numerous simulations, we choose to iterate the algorithm to take a new action every 30 minutes. This delay avoids waking up the system too many times. This value is dependent of the application and requires a good knowledge of both the algorithm and the application.

Results

We simulate a 3 week deployment of our buoy near to Lorient, Bretagne, France. Figure 5 shows the evolution of the battery charge and the sampling frequency of the sensors. At the beginning, the agent has no prior knowledge of its environment and takes random actions to become aware of it. After few iteration, we observe a daily variation in the sampling frequency, it corresponds to the evolution of the harvested energy

during the day. At the end of the simulation, the agent still does not know perfectly its environment but the daily variation are more well-defined.

The agent succeeds to adapt itself to the daily variation of the harvested energy without prior knowledge about it. However, one problem with this algorithm is that we have to store the Q-value of each pair (s, a), and it can only be uses for discrete state space and discrete action space. In addition, its time of convergence is difficult to estimate and make it unusable for applications with time constraints. To improve the convergence's time of the Q-learning algorithm, there are different usable solutions. In the following section, we present the Dyna Q-learning and the results of a simulation with identical parameters. The backbone of Dyna Q-learning algorithm is the Q-learning to which few adjustments are made. And one advantage is the adjustment of its complexity according to the computing capacity available.

3.1.2 Dyna Q-learning

The Dyna Q-learning (Algorithm 2) [21] is a variant of the Q-learning, which also uses a table to store the Q-values. One drawback with Q-learning algorithm is its slow Q-values convergence, so the Dyna Q-learning is proposed to accelerate this convergence. To achieve this objective, Dyna Q-Learning uses a partial and deterministic model of the environment to learn using the previous experience. The last transition and the associated reward are stored in memory for each visited state. The algorithm uses this model to optimize the evaluation function at each episode or even independently during a break in the decision making process. It selects a state s already visited and chooses an action a already performed, and then it uses the transition s_{t+1} and the reward r stored in the memory to update the Q-value $Q(s, a)$. The number of updates per sampling period is noted N.

We use the same parameters α, γ and time slot as the Q-learning algorithm. Increasing the value of N reduces the learning phase up to a certain limit. However it increases the computation as well which results in an increase of the energy consumption. We choose to set N to 10, this value was determined after a series of experiments to improve the convergence rate; higher values did not accelerated the learning as much.

Results

We simulate the Dyna Q-learning in the same conditions as the Q-learning during 3 weeks. Figure 6 shows the evolution of the battery charge and the sampling frequency of the sensors. At the beginning of the simulation, the agent has no prior information about its environment and takes random actions as we can see with high variation in the sampling frequency. Around day 13, the sampling frequency follows the battery behaviour during the charge and discharge. At day 18 the sampling frequency reached the maximum at 1 Hz while the battery is fully charged. We can consider that the value converged around day 13.

The results show that the Dyna Q-learning algorithm increases the convergence of the Q-value by 17% in this example. The agent can take better decision about its sampling frequency to adapt itself the evolution of the battery load. However if the convergence of the Q-values is accelerated, the memory requirement increases as well. Indeed we need a look up table ($\mathcal{A} \times \mathcal{S}$) for the Q-values and 2 others of the same size to store the transition and the associated reward, and the problem is that embedded systems have often small memory capacity. To reduce the use of memory, another approach consists in computing the different Q-values possible using a neural network. The memory only stores the weight of neurons allowing larger MDP.

3.2 Algorithms using neural network

Creating and updating a Q-table is not efficient at all for large environments, it's not scalable. The idea in this case is to create a neural network that will approximate, given a state, the different Q-values for each action (Figure 7). In this section, we present the Deep Q-learning (Algorithm 3) [22], a version of Q-learning using neural networks.

3.2.1 Deep Q-learning

The Deep Q-learning uses a neural network to take advantage of their ability to generalize the learning. Instead of storing the Q-value for each state-action pair in a look up table, a neural network takes as an input the state and for each possible action computes the expected reward (Figure 7). We take the biggest Q-value of this output to find our best action. Then after the episode, we update the neural network with the obtained reward. Equation (6) shows how the algorithm adjusts the network's weights Δw using a gradient descent algorithm.

$$\underbrace{\Delta w}_{\substack{\text{Change in} \\ \text{weights}}} = \alpha[\underbrace{(R + \gamma \max_{a'} \hat{Q}(s_{t+1}, a', w))}_{\substack{\text{Maximum possible Q-value} \\ \text{for the next state}}} - \underbrace{\hat{Q}(s, a, w)}_{\substack{\text{Current} \\ \text{predicted} \\ \text{value}}}] \underbrace{\nabla_w \hat{Q}(s, a, w)}_{\substack{\text{Gradient of our} \\ \text{current predicted} \\ \text{Q-value}}} \tag{6}$$

The change in weights depends on the difference between the predicted value for the current state, the highest value for the next state and the gradient of the predicted value $\nabla_w \hat{Q}(s, a, w)$.

Experience replay helps to avoid forgetting previous experiences and reduces correlation between the experiences. Because of high correlation between actions and states, the weights of the network are highly variable. At each interaction with the environment, we receive a tuple (state, action, reward, new state) and use it to learn the best action to take. The problem is that this information is obtained sequentially and the network tends to forget the previous experience since it overwrites them with new experiences. With the experience replay, we decrease this problem by storing the previous experiences in a replay memory while interacting with the environment, thus we sample a small batch of experiences to feed our neural network.

A neural network does not need to explore all state-action pairs. The network can find good solution without exploring the states by generalizing its knowledge. However the Deep Q-learning needs different experiences to avoid overfitting. The overfitting appears when the network fails to reliably predict future observation.

This algorithm has parameters to adjust as the previous ones. The learning rate α determines how fast the new experience replaces the old ones, and a neural network is more sensitive than a look up table to this parameter evolution since it impacts all the weights and so all the computed reward. So we set α to 0.1, which is a balance between network stability and convergence speed. The value of α is determined after several experiments. The discount factor γ still represents the importance given to the future reward over the immediate one and we set it to the same value 0.8. We tested different hyperparameters for our neural network and finally, we selected a neural network composed with 1 input layer neuron then 1 hidden layer of 20 neurons and finally 10 neurons in the output layer. An overly complex or simple network will not be efficient at all. The activation function is a rectifier (Equation (7)). We store in a memory the last 10 experiences in order to use it as a batch.

$$f(x) = x^+ = max(0, x) \tag{7}$$

Results

As for the previous algorithms, we simulated the system over a period of 3 weeks. Figure 8 displays the evolution of the battery load and the sampling frequency of the sensors. At the beginning the sampling frequency is low and increases after day 17. Nevertheless we start to observe the daily variation around day 4. At the end the sampling frequency increases and the daily variation disappears. The agent loses the information it learned at the beginning.

The Deep Q-learning algorithm achieves an energy management using less memory than the Q-learning. However it forgets over the time the daily variation information. A way to improve the learning is to store only relevant information in the memory, but the question is how to determine which information is relevant.

The following section presents a comparison of the three presented algorithms to help designers to determine the most appropriate for their applications.

4 COMPARATIVE RESULTS

We simulated three different reinforcement learning algorithms using the same decision process. The goal of each algorithm was to manage the energy of the same marine buoy equipped with solar panels. The variation in the harvested energy during the day makes the energy management challenging. They succeed to adapt the sampling frequency to this daily variation. These algorithms use Q-values to determine the best action to take and the convergence of the Q-values is difficult to anticipate. The algorithms have probably not reached the optimal policy but provide good enough decision to preserve the system's battery.

The choice of the RL algorithm to use is not straightforward and depends on application requirements, computation capabilities and available memory. Moreover, there are numerous different embedded systems and their capabilities in memory and processing vary a lot. Each algorithm has its own advantages and so, the choice of the trade-off will depend on the context.

In order to provide designers with guidelines, we compare the algorithms using four criteria : the computation requirement, the memory needs, the learning speed and the stability of the algorithm. The computation requirement is important for an embedded system. Indeed, these systems have limited computing capacity and often time constraint applications. Moreover, they consume energy to compute the Q-value and it would be counter-productive to consume more energy or to take more processing time for the energy management algorithm than for the main application. The Q-learning algorithm is the less processing-hungry algorithm since it computes only a value at each iteration. The Deep Q-learning computes all the Q-values at each iteration and the training of the neural network requires processing too.

Memory usage is another parameter to consider since the memory available on a micro-controller unit is often low (few kB). The neural network approach needs less memory than a look up table for large environment since it stores only the weights of the neurons, whereas the look up table stores all the Q-values. Moreover it is possible to further reduce the memory usage of a neural network by reducing the number of neurons but we decrease the accuracy of the computed Q-values.

The learning speed requirement depends on the application. Reinforcement learning approaches can only be used when the system can make errors safely and learn. However, some applications need to take good decisions quickly after the deployment of the node. The Deep Q-learning surpasses other algorithms on the learning speed. In fact, the neural networks have the property to generalise the learning, which makes the knowledge of the Q-values in one state usable in a different state. A solution to have the benefits of a good convergence speed while using a look up table, is to implement a Dyna Q-learning and then disable the model of the environment when the learning rate α is low.

The stability depends on the impact that new information can have on the algorithm. The stability of the neural network approach is lower, indeed, with each network update all future Q-values are modified. While a new experience with Q-learning only changes one Q-value and only once. With the Dyna Q-learning, this Q-value can be modified several times before repeating the experiment with the use of a partial model. This criterion becomes important when the learning ends because a bad experience can modify the agent's behaviour, which is why the value of α is modified as the exploration progresses, reducing this risk.

These different parameters (Figure 9) should give a designer the guideline to choose the most appropriate algorithm for the energy management of the system he is developing. After choosing the algorithm, the designer must find the correct parameters for his application. These parameters are found either with the designer's experience or empirically.

5 CONCLUSION AND PERSPECTIVES

More and more algorithms of energy management use a Reinforcement Learning approach, since it allows a system to adapt dynamically to changes in its environment. Nevertheless, there is numerous existing algorithms using different approaches. In this paper, we select and compare three RL algorithms with an energy management of the same marine buoy, Q-learning, Dyna Q-learning and Deep Q-learning. Based on our experiments, we propose a classification to help designers to select an appropriate solution depending on their system's constraints. These criteria are Learning Speed, Stability as well as Storage and Computation

requirements. The comparison shows that each approach has its own advantages and drawbacks. And the choice of the approach depends on the application and the trade-off between computation and memory use.

The energy management used in this paper is quite simple. The only possible action of the agent is to modify the sensors sampling frequency, the data are transmitted immediately that is a sub-optimal solution. An improvement would be to propose a more complex energy management algorithm with several different actions, to change the sampling frequency but also the transmission parameters. Future works aim to implement the improved algorithms on a real marine buoy and test them with the real world conditions in order to validate the simulation results.

REFERENCES

[1] V. Sharma and S. Chandel, "Performance and degradation analysis for long term reliability of solar photovoltaic systems: a review," *Renewable and Sustainable Energy Reviews*, vol. 27, pp. 753–767, 2013.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT press, 2018.

[3] M. I. Khan and B. Rinner, "Energy-aware task scheduling in wireless sensor networks based on cooperative reinforcement learning," in *Communications Workshops (ICC), 2014 IEEE International Conference on*, pp. 871–877, IEEE, 2014.

[4] K. J. Prabuchandran, S. K. Meena, and S. Bhatnagar, "Q-Learning Based Energy Management Policies for a Single Sensor Node with Finite Buffer," *IEEE Wireless Communications Letters*, vol. 2, no. 1, pp. 82–85, 2013.

[5] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving Autonomous Power Management using Reinforcement Learning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 18, no. 2, pp. 1–32, 2013.

[6] S. Kosunalp, "A new energy prediction algorithm for energy-harvesting wireless sensor networks with Q-Learning," *IEEE Access*, vol. 4, pp. 5755–5763, 2016.

[7] A. Galindo-Serrano and L. Giupponi, "Distributed Q-learning for aggregated interference control in cognitive radio networks," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 4, pp. 1823–1834, 2010.

[8] Y. Wang, Q. Xie, A. Ammari, and M. Pedram, "Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification," in *Proceedings of the 48th Design Automation Conference*, DAC '11, (New York, NY, USA), pp. 41–46, ACM, 2011.

[9] S. Shresthamali, M. Kondo, and H. Nakamura, "Adaptive Power Management in Solar Energy Harvesting Sensor Node Using Reinforcement Learning," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 5s, pp. 1–21, 2017.

[10] D. Banerjee, S. Sen, and A. Chatterjee, "Self learning analog/mixed-signal/RF systems: Dynamic adaptation to workload and environmental uncertainties," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 59–64, IEEE Press, 2015.

[11] Y. Su, R. Fan, X. Fu, and Z. Jin, "Dqelr: An adaptive deep q-network-based energy-and latency-aware routing protocol design for underwater acoustic sensor networks," *IEEE Access*, vol. 7, pp. 9091–9104, 2019.

[12] F. A. Aoudia, M. Gautier, and O. Berder, "Learning to Survive : Achieving Energy Neutrality in Wireless Sensor Networks Using Reinforcement Learning," may 2017.

[13] O. Berder and O. Sentieys, "Powwow: Power optimized hardware/software framework for wireless motes," in *23th International Conference on Architecture of Computing Systems 2010*, pp. 1–5, VDE, 2010.

[14] M. I. Khan, M. M. Alam, and Y. Le Moullec, "A multi-armed bandit solver method for adaptive power allocation in device-to-device communication," *8th International Conference on Ambient Systems, Networks and Technologies (ANT 2018)*, no. February, 2018.

[15] Y. Huang, W. Yu, C. Osewold, and A. Garcia-Ortiz, "Analysis of pkf: A communication cost reduction scheme for wireless sensor networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 2, pp. 843–856, 2015.

[16] H. Bellia, R. Youcef, and M. Fatima, "A detailed modeling of photovoltaic module using matlab," *NRIAG Journal of Astronomy and Geophysics*, vol. 3, no. 1, pp. 53–61, 2014.

[17] C. Piedallu and J.-C. Gégout, "Multiscale computation of solar radiation for predictive vegetation modelling," *Annals of forest science*, vol. 64, no. 8, pp. 899–909, 2007.

[18] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, may 1992.

[19] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[20] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Heuristically accelerated q–learning: A new approach to speed up reinforcement learning," in *Advances in Artificial Intelligence – SBIA 2004*, pp. 245–254, Springer Berlin Heidelberg, 2004.

[21] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

6 FIGURES AND TABLES

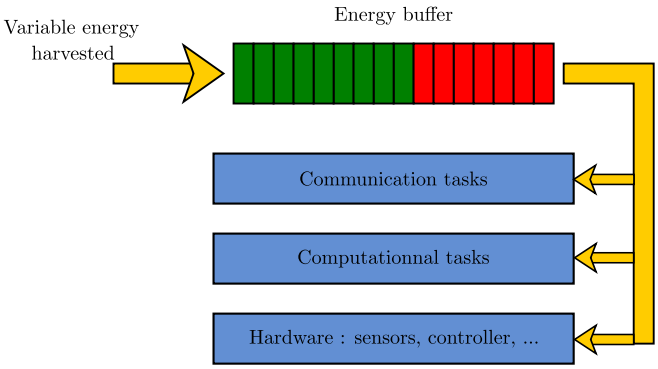

Figure 1: Energy consuming task of a sensor node

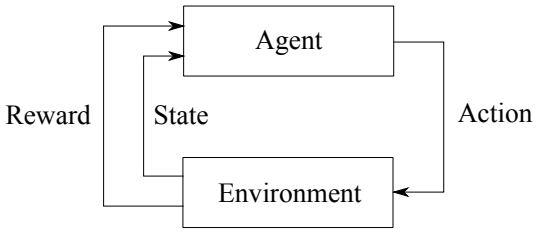

Figure 2: Interaction between an agent and its environment

Table 1: Buoy components

Components	**Characteristic**
Anemometer 3D	WindMaster HS
Atmospheric sensor	YOUNG61302L
Processor	Cortex-M4 MCU
Transmitter	CC1000
Harvester	**Power**
Solar panel	2×10 W
Battery capacity	5200 mA

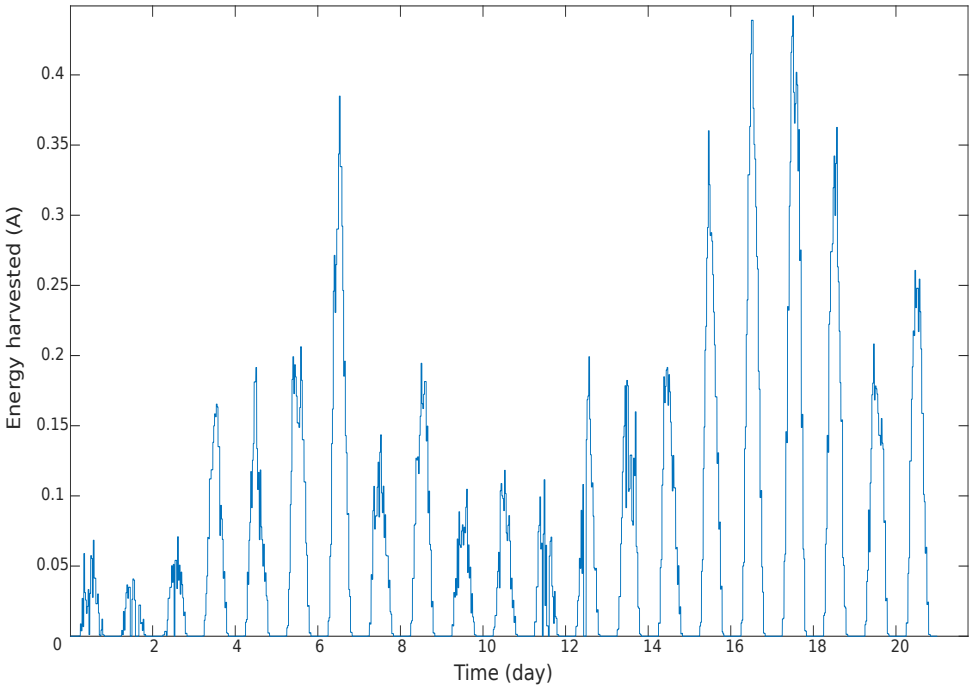

Figure 3: Evolution of the energy harvested in simulation between 07/01/2017 and 07/21/2017

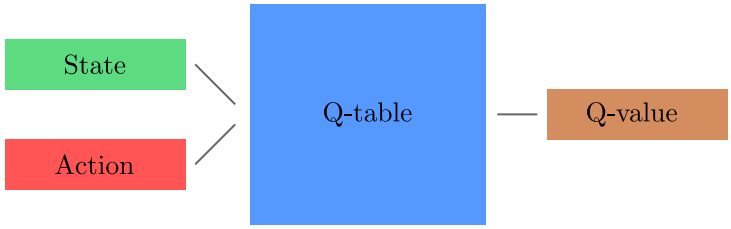

Figure 4: Overview of how a look up table works

Algorithm 1 Q-learning

Initialize $Q(s, a)$ arbitrarily
The agent observes the initial state s_0
for each decision epochs **do**
Choose a from s using policy derived from Q
Take action a, observe the new state s' and the associated reward r
$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s, a) \right)$
$s \leftarrow s_{t+1}$
end for

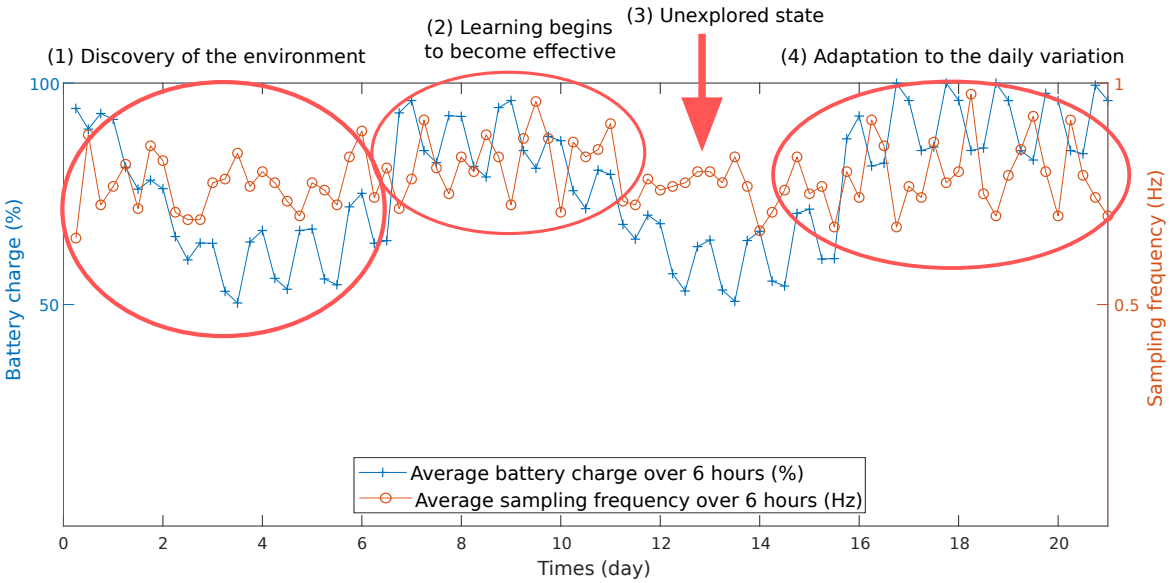

Figure 5: Evolution of the battery charge and sampling frequency of the sensors using the Q-learning algorithm

Algorithm 2 Dyna Q-learning

Initialize $Q(s, a)$ arbitrarily
The agent observes the initial state s_0
for each decision epochs **do**
  Choose a from s using policy derived from Q
  Take action a, observe the new state s_{t+1} and the associated reward r
  $Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s,a) \right)$
  $m(s,a) \leftarrow s_{t+1}, r$
  for $i = 1$ **to** N **do**
    $s \leftarrow$ random visited state
    $a \leftarrow$ random visited action
    $s_{t+1}, r \leftarrow m(s,a)$
    $Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s,a) \right)$
  end for
end for

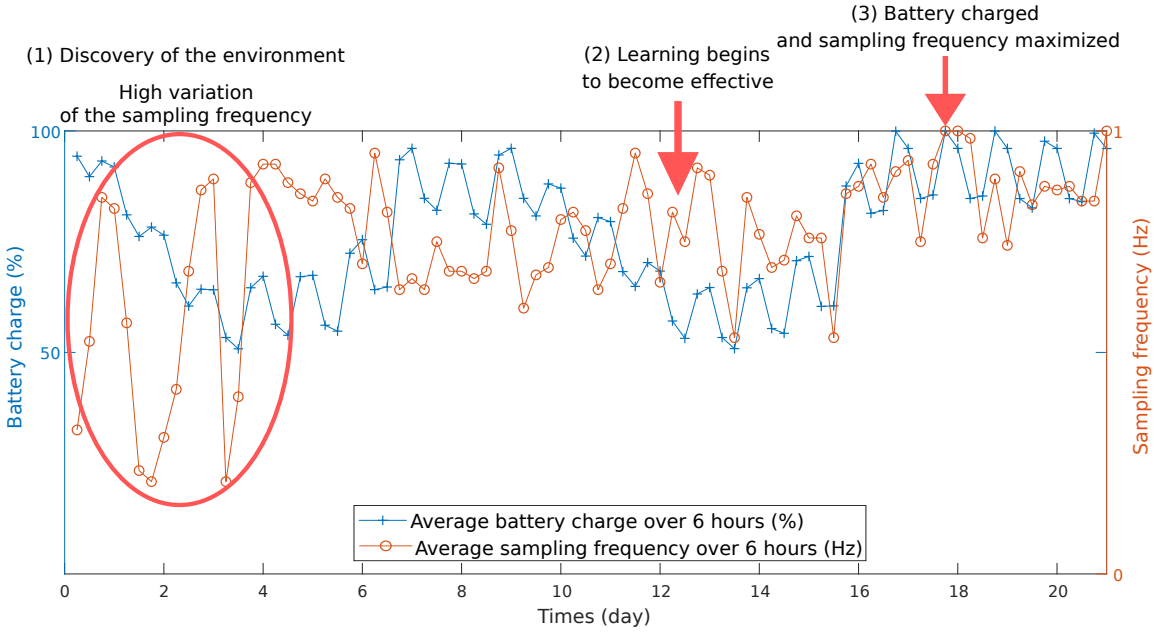

Figure 6: Evolution of the battery charge and sampling frequency of the sensors using the Dyna Q-learning algorithm

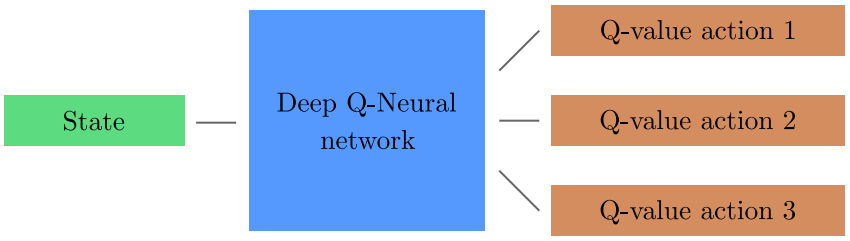

Figure 7: Deep Q-learning

Algorithm 3 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $\mathcal{N}$
Initialize action-value function Q with random weights
for each decision epochs **do**
Initialise sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
With probability ϵ select a random action a_t
otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
Execute action a_t and observe reward r_t
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a_{t+1}} Q(\phi_{j+1}, a_{t+1}; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
end for

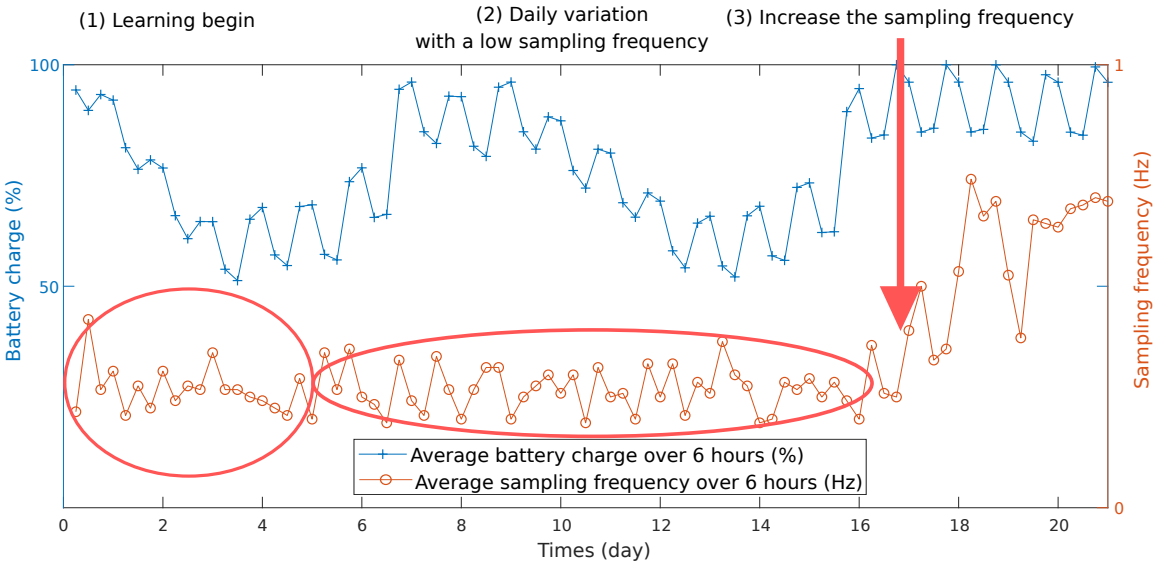

Figure 8: Evolution of the battery charge and sampling frequency of the sensors using the Deep Q-learning algorithm

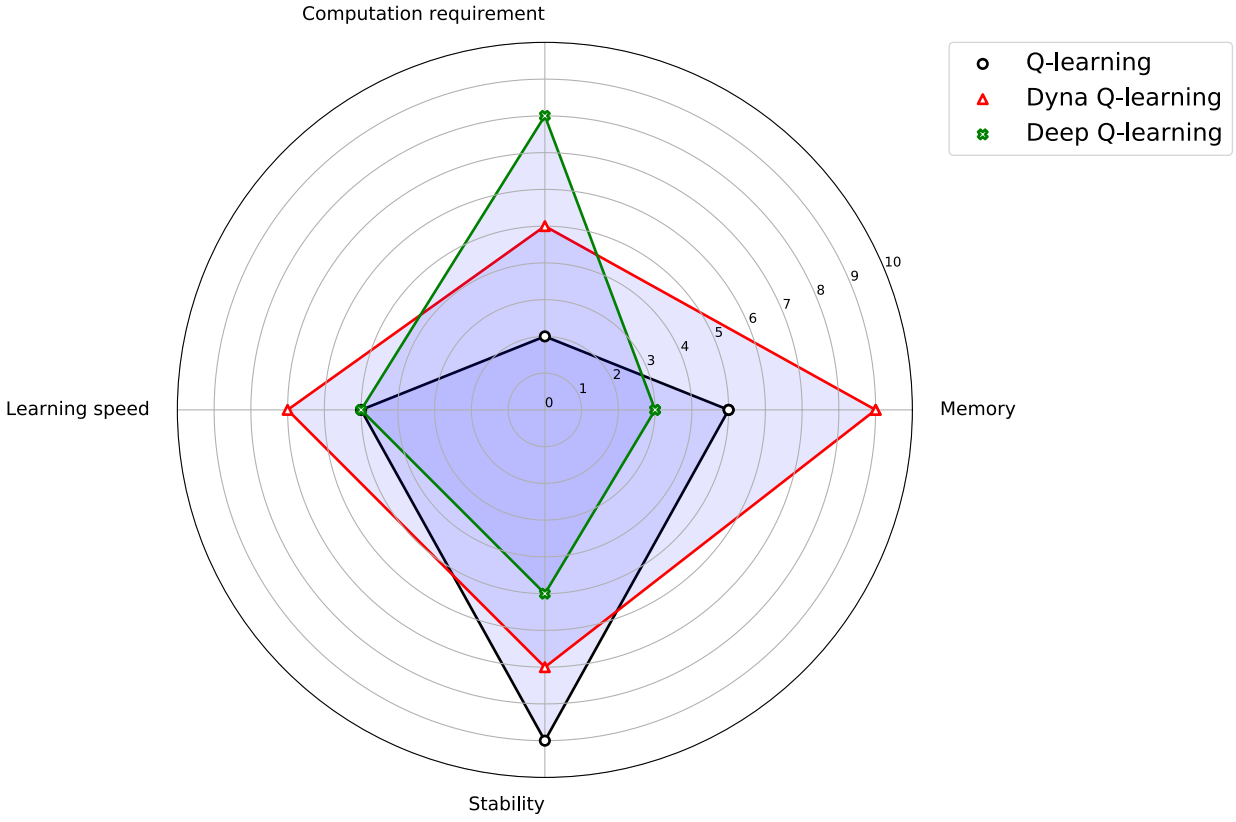

Figure 9: Algorithm comparison

7 BIBLIOGRAPHY

Yohann Rioual received B.S. and M.S. degrees in Electronics from the University of South Brittany, Lorient, France. He is currently working towards the Ph.D. degree at the Lab-STICC (Laboratoire des Sciences et Techniques de l'Information, de la Communicationet de la Connaissance), University Bretagne Sud, Lorient. His research interests include power/energy optimizations and reinforcement learning.

Johann Laurent is an Associate Professor at the University Bretagne Sud and works at the CNRS Lab-STICC (Laboratoiredes Sciences et Techniques de l'Information, de la Communication et de la Connaissance). His research interests include software consumption estimation and power characterization for complex architectures like SoCs. He received a Ph.D. in electronics from the University Bretagne Sud, France, in 2002 and his Habilitation à Diriger les Recherches in 2015.

Jean-Philippe Diguet is a CNRS director of research at Lab-STICC, Lorient/Brest, France. He received the Ph.D. degree from Rennes University (France) in 1996. In 1997, he has been a visitor researcher at IMEC (Belgium). He has been an associate professor at UBS University (France) until 2002. In 2003, he co-funded the dixip company in the domain of wireless embedded systems. Since 2004 he is a CNRS researcher at Lab-STICC, where he has been heading the MOCS team until 2016. He has been a visitor researcher at the University of Queensland, Australia in 2010 and an invited Prof. at Tohoku University, Japan in Nov. 2014 and May 2019, and at Univ. of São Paulo, Brazil, in Nov. 2016. His current work focuses on various aspects of embedded system design: Designs and Tools for NoC-based MPSoC architectures including memory-based computing, Self-adaptivity for uncertain environments as autonomous vehicles and Design of dedicated hardware accelerators.