A Comparative Study between Motivated Learning and Reinforcement Learning

J. Graham and J. A. Starzyk
School of EECS
Ohio Univ., Athens, OH, USA
{jg193404, starzykj}@ohio.edu

Z. Ni and H. He
Electrical, Computer, and Biomed. Eng.
Univ. of Rhode Island, Kingston, RI, USA
{ni, he}@ele.uri.edu

T.-H. Teng and A.-H. Tan
School of Computer Engineering
Nanyang Technological Univ., Singapore
{thteng, asahtan}@ntu.edu.sg

Abstract— This paper analyzes advanced reinforcement learning techniques and compares some of them to motivated learning. Motivated learning is briefly discussed indicating its relation to reinforcement learning. A black box scenario for comparative analysis of learning efficiency in autonomous agents is developed and described. This is used to analyze selected algorithms. Reported results demonstrate that in the selected category of problems, motivated learning outperformed all reinforcement learning algorithms we compared with.

Keywords—motivated learning; reinforcement learning; goal creation; pain signals; desired resources.

I. Introduction and Background

For over 20 years, reinforcement learning (RL) has dominated machine learning techniques used in robots, multi-agent systems, genetic algorithms, swarm intelligence and optimal control systems. It uses cumulative rewards to learn proper behavior in an environment depending on the result of the action taken in a specific state of the environment.

One of the key assumptions in reinforcement learning is that a reward is provided if the action/state are the same as they were the last time the reward was given. However in dynamic environments, particularly with other agents present, this is not always the case.

A typical reinforcement learning algorithm assumes that the resources that an agent needs to perform its actions are available in sufficient quantities. This is not the case in resource sharing and resource competition situations, where there are not enough resources to perform all tasks, or where there is a competition between agents for the limited amount of resources.

In such problems, there is a need to coordinate tasks and allocate resources to various tasks [1]. Existing approaches that aim at multi-objective reinforcement learning [2], [3] assume stationary operating conditions. Existing task coordination strategies include resource sharing [4], learning of coordination [5] and the use of set strategies [6]. More recent works use learning approaches in non-stationary environment, coordinating agents to perform joint actions [7], [8], and performing coordination of work [9].

We argue that resource sharing and task coordination in a dynamic environment is solved very effectively using a motivated learning approach [10]. Motivated learning (ML) was developed by Starzyk [11], [12] to address problems of goal creation and goal management in embodied cognitive agents. ML yields an internal reward system that motivates an agent to act in a dynamically changing environment with limited resources and potential adversarial actions by other agents. In this work we compare our motivated learning approach with several reinforcement learning algorithms in a black box scenario. We describe how such a scenario is obtained and how it is represented in the simulation environment, and compare the obtained simulation results.

We limit ourselves to studying the problem of resource sharing and task coordination, and evaluate the agent's activities in terms of the average reward that the agent receives from the environment for its actions. An extension of the black box scenario is planned to include actions by other agents that can either cooperate or compete with the ML agent.

The rest of this paper is organized as follows. Section II discusses advanced RL algorithms and introduces ML. Section III compares ML and various implementations of RL to demonstrate that ML performs better than any of the tested RL algorithms. Section IV includes the conclusions and future work.

II. Advances in Reinforcement Learning

A. Advanced RL algorithms

Classical RL uses exact value functions and policies, and is therefore limited to problems with small numbers of states and possible actions. However, real world problems have sensory inputs with a potentially infinite number of states, thus, they require approximation of value functions and action policies to be effective.

Modern RL uses approximate value iteration, policy iteration, and policy search. For real-time applications non-stationary RL was considered [13]. Many authors consider variants of RL that include online algorithms, where data is collected during system operation. In addition, advanced RL algorithms also use policy gradient, actor-critic methods, and simulation-based policy iteration.

Offline algorithms like fitted Q-iteration and least-squares policy iteration use efficient approximation methods and can better exploit the sensory and reward data than online approximation algorithms such as gradient-based Q-learning and approximate SARSA. However, modern applications in robotics are increasingly relying on on-line learning even if the approximation provided may be not optimum. The data efficiency of online methods can be increased by various

means including stored transition samples and building a model in model-learning methods [14].

Bertsekas reviewed a number of techniques that use policy iteration and least-squares policy evaluation [15]. He compared various RL methods with respect to policy evaluation using temporal difference and aggregation. He also reviewed off-line algorithms with respect to the use of matrix inversion and iterative methods. In his work, he focused on convergence, policy oscillation and singularity of these techniques. The conclusions of his comparative study are tentative and regard selected aspects like regular behavior, better error bounds, and exploration-related difficulties.

Inverse reinforcement learning (IRL) is based on the imitation of expert actions to learn a reward function [16]. A model-free IRL discussed in [17] generalized imitation learning, minimizes the relative entropy between state-action trajectories. This is very useful in learning direct policy by observation in problems where learning the value function is non-trivial. IRL is useful in robotics and helps in understanding the choice of action by a demonstrator [18]. A review of IRL methods is provided in [19].

While RL had spectacular successes in many robotic applications, where the system could learn complex control functions to successfully operate in a stationary environment (like modeling human strategy in the ping pong game [18], or autonomous helicopter control [20]), we believe that RL methods can be further improved to consider special situations. According to Coelho et al. [21] RL methods tend to learn very slowly, which leads to their poor performance in dynamic environments. This is the type of environment where motivated learning can be very useful.

For dynamic environments, a typical reward function that is used in RL is augmented by an intrinsic reward that is only known to the agent. Several implementations of this concept were developed, including artificial curiosity proposed by Schmidhuber [22], intelligent adaptive curiosity developed by Oudeyer et al. [23], or intrinsic hierarchy of skills proposed by Barto et al. [24]. RL enhanced with curiosity based learning performed better than random exploration typically used to support RL. Also, active learning [25] using these ideas maximizes the expected information gain and improves the learning speed by concentrating exploration where there is greater uncertainty in the internal model. In active learning, a machine can achieve higher learning efficiency by actively choosing the data from which it learns. Motivated learning takes ideas of intrinsic motivation a step further, such that specific goal oriented motivations are developed and internally managed by the ML agent, giving it large autonomy and improved performance in a dynamic environment.

B. Motivated learning

In motivated learning (ML), a learning agent develops internal motivations and related goals based on its perceived successes or failures during its interaction with the environment. Internal motivations, created either by external rewards or other motivations, may dominate over the externally set goals (and rewards). Once created, the motivations are persistent and are responsible for all (except curiosity based) actions of the agent. In our implemented version of motivated learning, motivations are triggered by biases and internal pain signals that compete for the machine's attention, providing a natural mechanism for goal competition and scheduling.

ML is closely related to RL as it uses RL to learn how to satisfy its motivations. The developed bias system evaluates the importance of environmental stimuli to check how they relate to its goals. This allows the agent to self-supervise and self-organize its exploratory and motivated learning activities. We can represent our ML schema as shown on Fig. 1. This schema corresponds to the motivated hierarchical reinforcement learning MHRL(I) model [26], but it differs in how the system motivations are established.

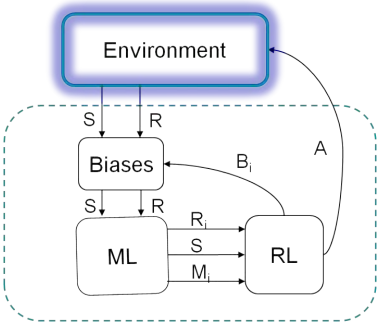

Fig. 1 ML agent interacting with the environment.

In the organization presented in Fig. 1, individual biases B_i are established depending on the success or failure of system response A generated by the RL block. ML sets individual goals providing the RL block with the sensory information S together with the intrinsic reward R_i and an internal motivation to act M_i.

In our motivated learning algorithm, bias signals are used to determine the level of pain or need associated with a particular concept (resource availability or an action performed by another agent). In this work we limit discussion to resource related motivations and learning. Biases indicate the likelihood of running out of resources needed or of facing a hostile action by another agent. There are several ways in which bias can be calculated.

If the ML agent needs to maintain a resource at a certain level, a bias signal that reflects how difficult it is to obtain this resource is used. The agent must first use the resource to learn if the resource is desired or undesired. The resource bias signal depends on the amount of the resource and its desired/undesired status as follows:

$$B(s_i) = -(1+\delta_i) * (lnA(s_i) + 1) + 2 * A(s_i) \quad (1)$$

where

$$A(s_i) = \frac{\varepsilon + R_c(s_i)}{\varepsilon + R_d(s_i)} \quad (2)$$

represents the availability of resource s_i, R_c is a current resource amount, and R_d is a desired resource amount established by the ML agent during learning. (The implementation used in this work simply sets R_d equal to the

initial environment state.) ε is a small positive number to prevent numerical overflow, and $\delta_i = 1$ when the resource is *desired*, $\delta_i = -1$ when it is *not desired*, and $\delta_i = 0$ otherwise (when the character of the resource is unknown).

Depending on the level of the bias signal and importance of a particular resource to the agent, ML generates an internal pain signal (equivalent to a negative reward). R_d is used as a normalizing factor for the resource level. If $R_c(s_i) = R_d(s_i)$ the corresponding resource pain is zero for a desirable resource, and for an undesirable resource the smaller the $R_c(s_i)$, the smaller the resource pain. Pain reaches significant levels when the agent lacks a desired resource or has too much of an undesired one.

Using the bias signal, the pain value related to this bias is obtained from:

$$P(s_i) = B(s_i) * w_{bp}(s_i) \quad (3)$$

where w_{bp} is the weight between bias and a given pain. This weight is computed incrementally based on pain change signals that resulted from the action taken as follows:

$$w_{bp} = \begin{cases} w_{bp} + \Delta_{b+} * (\alpha_b - w_{bp}) & if\ associated\ pain\ changed \\ w_{bp} * (1 - \Delta_{b+}) & if\ there\ was\ no\ change\ in\ pain \\ w_{bp} * (1 - \Delta_{b-}) & if associated\ percept\ was\ not\ used \end{cases} \quad (4)$$

where α_b sets the ceiling for w_{bp}, Δ_{b-} sets the potential rate of decline for w_{bp} weights, and Δ_{b+} sets the rate of potential increase for w_{bp} weights. The default values for these parameters are 0.5, 0.00001, and 0.2, respectively.

A ML agent's *motivations* are to reduce its pains below a threshold. Pain reduction in ML is equivalent to a reward in RL, but once a pain is reduced the agent does not need to act on it until it again increases above threshold. A current goal is selected based on a dominant pain signal, and it represents an intended action that the agent wants to perform.

The ML agent must choose which of the currently activated goals it must implement first. For clarity, let us define a goal as the intended action to reduce a specific pain, and an action as the motor action taken in an attempt to meet the goal. A simple ML agent chooses the goal with the greatest pain value and the best chance to reduce this pain. This simple approach can be improved as demonstrated in [27]. However, in the case of the Black Box scenario described in III-B, this is moot since the RL agents cannot use additional information needed (without appropriate modifications).

Once a goal is selected and an action performed, a RL mechanism is used to learn useful actions. As a result of this action a new intrinsic reward signal is obtained by the ML agent, which in turn generates a bias signal for or against a specific action or a resource used.

III. Comparision to Reinforcement Learning

A. Compared algorithms

In this section, we compare our Motivated Learning algorithm with several reinforcement learning algorithms, including classical methods like Q-learning, SARSA(λ), and hierarchical reinforcement learning but also more advanced algorithms like NFQ, Explauto, and TD-FALCON. For fair comparison, we set the reinforcement learning algorithms in the same environment and with the same parameters and testing conditions. Please note that we have attempted to optimize the parameters of the tested algorithms to make the comparison as fair as possible. None of the algorithms have any pre-encoded knowledge of the environment, and each of them receives the same information. This includes the ML, which was given no hidden parameters or information about the environment. The reinforcement learning algorithms are detailed as follows:

Q-learning: Q-learning is one of the traditional reinforcement learning algorithms and can be used to acquire optimal control strategies from delayed rewards. The agent usually has no prior knowledge of the effect of its actions on the environment (e.g. it has no model to work with). There are several variations of Q-Learning, such as Delayed Q-learning, Greedy GQ, etc. Here, we implement the Q-learning algorithm from [28] with the discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.7$.

SARSA(λ): The SARSA algorithm considers the transitions from state-action pair to state-action pair, and is a standard example of a temporal difference (TD) learning algorithm. SARSA(λ) combines the Monte Carlo and dynamic programming ideas [29]. It updates reward estimates based on the other learned estimates, without waiting for a final outcome. It differs from the original SARSA with the inclusion of the trace decay (λ) parameter, which affects the distribution of reward. (Larger λ leads to a larger proportion of the reward credit being given to more distant states when there are multiple cycles between rewards.) The discount factor, learning rate, and trace decay parameters are set to $\gamma = 0.9$, $\alpha = 0.4$, and $\lambda = 0.9$.

Hierarchical RL: We used a variant of the MAXQ algorithm from [30] adopted to implement the hierarchical reinforcement learning (HRL) algorithm, where the goals are split into subgoals and subtasks. The size of the state space is reduced and the learning efficiency is improved. In HRL, the agent constructs a set of policies that need to be considered during reinforcement learning. The MAXQ value function is assumed to represent the value function of any given hierarchy. The target Markov decision process (MDP) is decomposed into smaller MDPs The parameters for the algorithm were set as $\gamma = 0.9$, and $\alpha = 0.4$.

Dyna-Q+: The Dyna-Q+ algorithm includes direct reinforcement learning, model learning and planning [29]. It is believed that Dyna-Q+ can deal with changing environments better than other RL algorithms. It generates an action based on direct reinforcement learning and then updates both a Q

table as well as its model. During the planning process, the Q-planning algorithm randomly chooses samples from state-action pairs that have already been visited. In this case, the model will never be queried with a pair about which it has no information. The parameters are set the same as those in Q-learning, with discount factor $\gamma = 0.9$ and learning rate $\alpha = 0.7$.

Explauto: Explauto [31] is a framework for the implementation for active and online sensorimotor learning algorithms. Explauto cognitive architecture is composed of a the sensorimotor model, which iteratively learns forward and inverse models from experience, and an interest model which makes the choices about where to explore in the environment. In our tests with Explauto we used its built-in DiscreteProgress interest model configured to match our environment with other parameters left at their default values.

TD-FALCON: This algorithm [32] incorporates a temporal difference algorithm with a family of self-organizing neural networks known as the Fusion Architecture for Learning and Cognition (FALCON). (TD-FALCON has some similarities to SARSA.) Based on the Adaptive Resonance Theory and using evaluative feedback from the environment, TD-FALCON works by learning the value functions of the state action space estimated using Q-Learning. The learned value functions are then used to determine the effective actions.

NFQ RL: NFQ (neural fitted Q iteration) is a batch RL learning FQI (fitted Q iteration) method [33]. FQ implements a dynamic scaling heuristic that can be seamlessly integrated into neural batch RL algorithms, which use a fixed set of a priori-known transition samples, e.g. offline learning. Fitted Q-iteration can be viewed as approximate value iteration applied to action-value functions.

B. Black Box scenario

To compare the aforementioned algorithms, we designed a "black box" environment that would present state and reward information to the RL or ML algorithm and receive a response in the form of an action to take. This action would then be presented to the environment, which would adjust itself accordingly and respond with a reward value ranging from 0-1 depending on the action and the current state of the environment. We show the results of the comparison of these algorithms in the next section.

The environment is an 8-level hierarchy of "resources" that depend on each other for restoration, similar in structure to the basic scenario presented in [10]. In this scenario, there is a single "primitive" need, which can be resolved by the correct action, which consumes a specific resource. This specific resource gets depleted over time and needs yet another action/resource combination to restore, and so on up to the "top" level of the resource hierarchy, which is not depleted.

In the "black box" scenario each *potential* primitive reward R_p that can be received from the environment, increases gradually after each iteration until it reaches its maximum level R_{mp}

$$R_p(i) = min(i_{0p} * R_{rp}, R_{mp}) \quad (5)$$

where i is current iterative step (time elapsed), i_{0p} is the iterative step from the last time R_p was awarded to the agent, and R_{rp} is the rate of change of this primitive reward. After the reward is received, i_{0p} is set to 0. R_p is only awarded to the agent if it performs a beneficial action to reduce the primitive need. If plotted, the potential reward function would look somewhat like a sawtooth plot with peaks varying in range from 0 to R_{mp}, with the exception that once the line reached R_{mp} it would remain there until it dropped due to the reward being given.

In the experiment, $R_{mp} = 1$ and $R_{rp} = S_{Rate}/S_{inp}$, where S_{inp} is the initial value of the primitive reward generating resource, and S_{Rate} is its rate of decline. We varied S_{Rate} for the tested algorithms to see how they perform under varying levels of pressure. For instance, if $S_{inp} = 40$ and $S_{Rate} = 1$, it will take 40 iterations for the resource to deplete (and make the maximum reward available). Higher values of S_{Rate} will mean it will take less time for the resource to be depleted. However, since the resources are connected in a hierarchy, it will also mean that the agent will have to learn the next resource in the chain more quickly, hence the aforementioned increase in pressure.

The received rewards are averaged and normalized using

$$Ave\ R_{Norm}(i) = \frac{\frac{1}{i}\sum_{p=1}^{n}\sum_{k=1}^{i} R_p(i)}{\sum_{p=1}^{n} R_{rp}} \quad (6)$$

where *i* indicates the current time step, *n* is the number of primitive resources, and R_{rp} is the rate at which the rewarding resource *p* is depleted.

C. Reinforcement Learning Results

In the following tests, unless otherwise stated, each plot shows the results of 25 averaged tests, each test running for 10,000 iterations. As expected, for all the algorithms, a greater S_{Rate} yields lower performance, since the algorithm has less time to explore and is under more pressure to perform. This is observed most noticeably in the Q-Learning, SARSA and HRL algorithms.

In Fig. 2, we observe that the ML algorithm performs well (and maintains near perfect performance until it hits an S_{Rate} of about 16, at which point its performance starts to fall off.

This is because at this point it began to have difficulty maintaining the primitive resource, while also maintaining the other resources. It simply ran out of time to perform the required actions due to the high rate of decrease in the primitive resource and the associated higher demand on higher level resources, and its reward capability suffered as a result. However, ML is able to perform with higher S_{Rate} and greater overall reward than any of the other algorithms tested here.

Figs. 3-6, present the results from the Q-Learning, SARSA, HRL and Dyna-Q+ algorithms. Note, that beyond the first few hundred iterations, the results for each algorithm are similar to each other. This is likely because all three

algorithms share the same basic process and have difficulty performing properly once the known solution toward generating their reward is no longer available and they have to rely on random attempts.

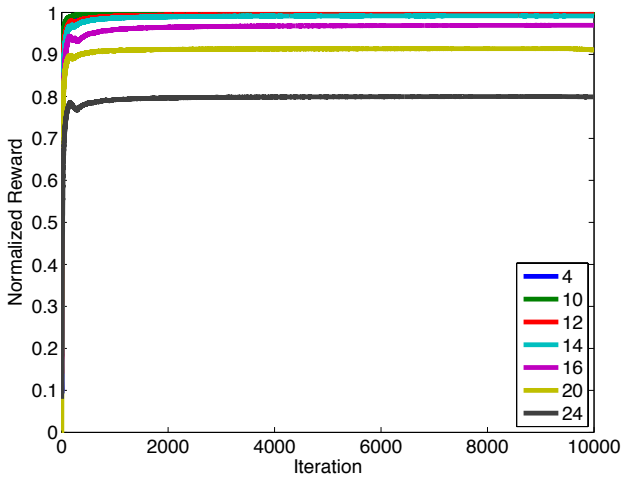

Fig. 2. Combined Results from the ML algorithm for different values of S_{Rate}.

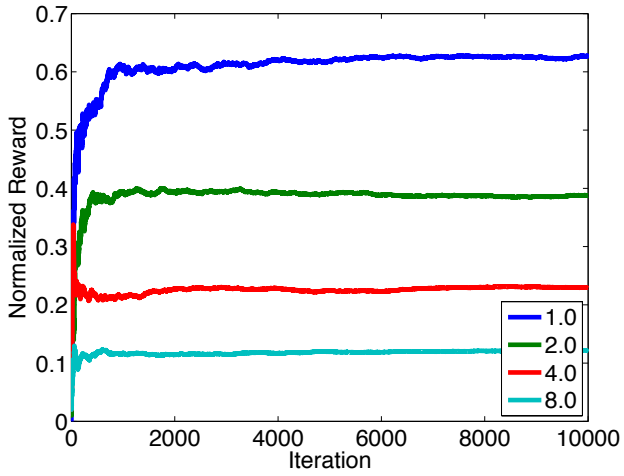

Fig. 3. Combined Results from the Q-Learning algorithm for different S_{Rate}.

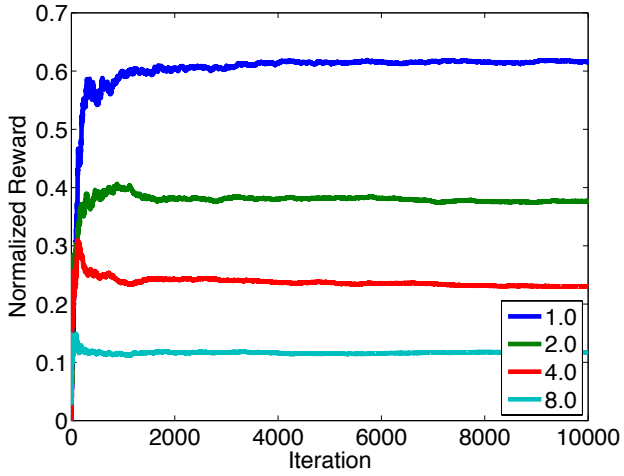

Fig. 4. Combined Results from the SARSA algorithm for different S_{Rate}.

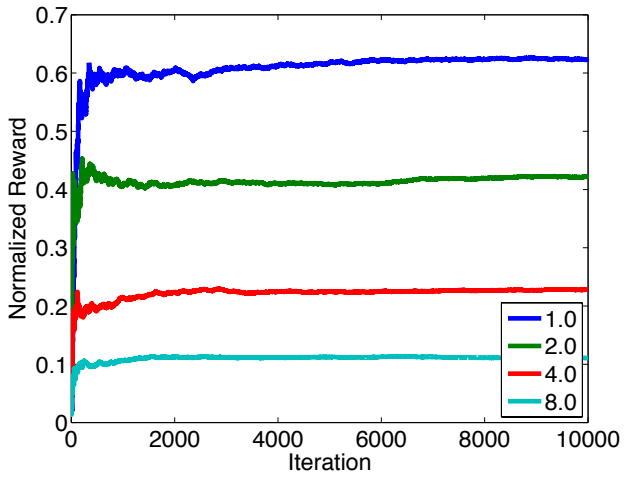

Fig. 5. Combined Results from the HRL algorithm for different values of S_{Rate}.

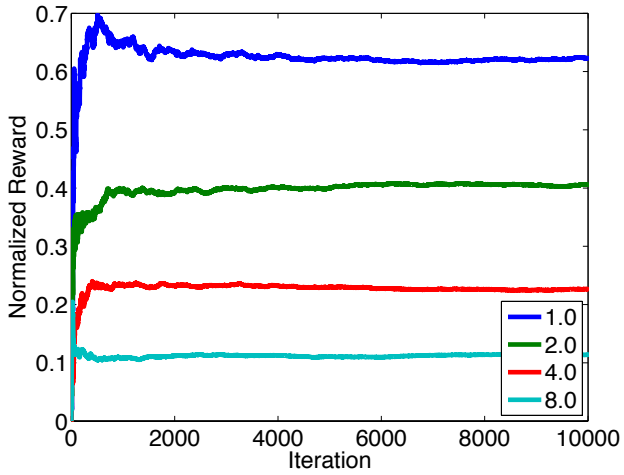

Fig. 6. Combined Results from the Dyna-Q+ algorithm for different values of S_{Rate}.

Fig. 7 shows the results from the Explauto algorithm, which performs similarly (although slightly worse), to the Q-Learning, SARSA, HRL and Dyna-Q+ algorithms. This is likely due to it being more oriented toward sensorimotor based RL learning rather than the type of RL hierarchy we are dealing with here.

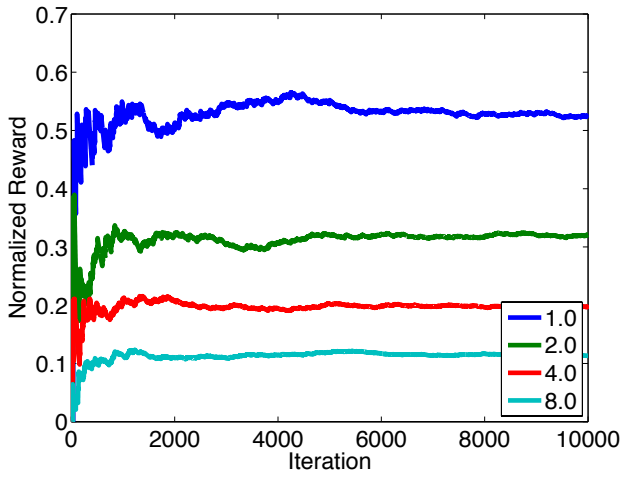

Fig. 7. Combined Results from the Explauto algorithm for different S_{Rate}.

Fig. 8 shows the results from an implementation of TD-FALCON in our environment. TD-FALCON is actually one of the better performing algorithms in our tests. Like the preceding algorithms, its performance tends to decrease with the increase of S_{Rate}. Only the NFQ algorithm seems to exceed its performance, and only then at higher values of S_{Rate}.

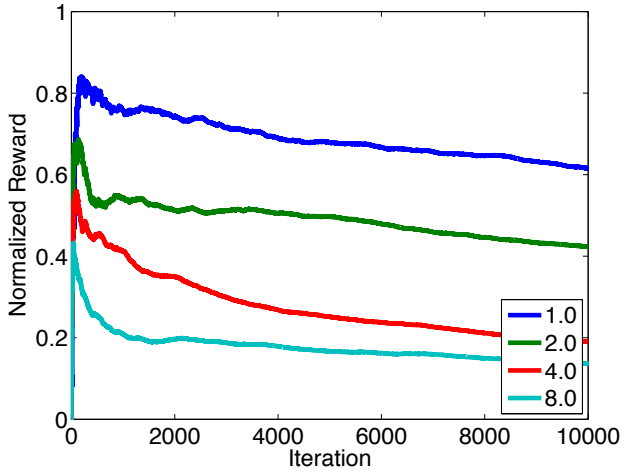

Fig. 8. Combined Results from the TD-FALCON algorithm for different values of S_{Rate}.

Interestingly, the NFQ algorithm tends to perform better than (most of) the other RL methods when the S_{Rate} is increasing (see Fig. 9). However, the algorithm's performance is not very consistent. The reason for this inconsistency is likely the oscillation observed in Fig. 10 among the individual runs and the large confidence interval of ±0.123. The confidence intervals of Q-Learning, SARSA, HRL, Dyna-Q+, TD-FALCON, Explauto and ML for $S_{Rate} = 8.0$ are 0.0034, 0.0035, 0.0034, 0.0031, 0.0326, 0.0026 and 0.0013, respectively.

Fig. 10 gives an 'inside' view of the NFQ algorithm's results by showing performance results for all 25 runs. It can be observed that the algorithm appears to oscillate between good and bad performance (or two bands), while appearing to gradually improve overall.

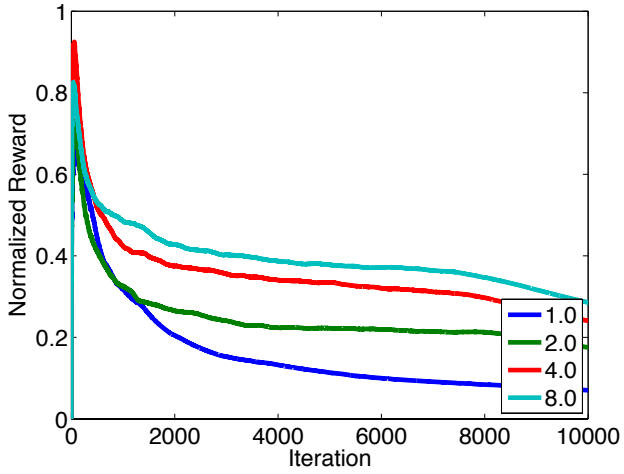

Fig. 9. Combined Results from the NFQ algorithm for different values of S_{Rate}.

For example, using the legend as a guide, run 10 is the highest of the lower performing band of runs, however, run 12 is the 2nd worst run of all of them. Run 16 has the highest average reward at the 10,000th iteration. So while time does seem to bring improvement to the performance, it is not very consistent.

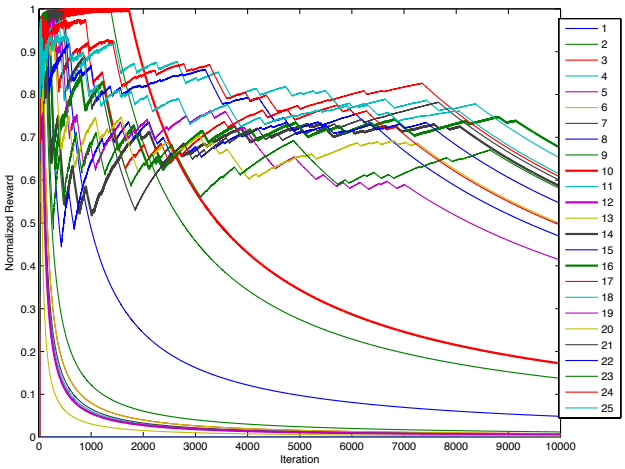

Fig. 10. Individual results of the NFQ algorithm with $S_{Rate} = 8$.

In Fig. 11 we directly compare our Motivated Learning algorithm against the reinforcement learning algorithms. TD-FALCON is better than Dyna-Q+, Dyna-Q+ performs slightly better than HRL, SARSA, or Q-learning, and Explauto with NFQ are the worst. However, ML outperforms all of them. ML appears better suited to operate in the hierarchically structured environment we provided due to the way it creates additional needs.

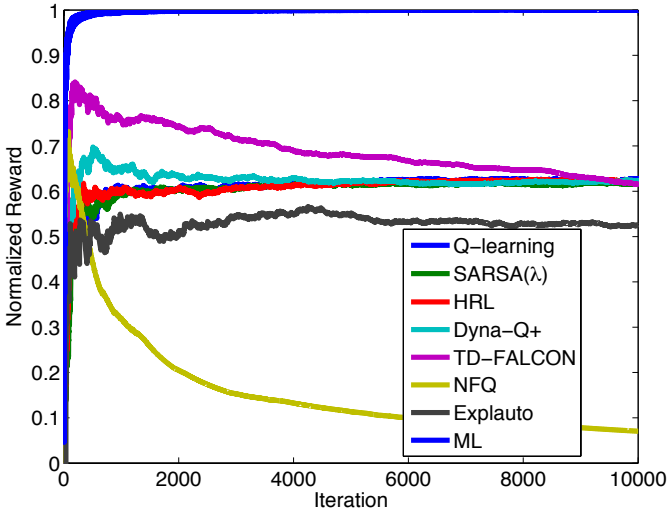

Fig. 11. Comparison of Reinforcement Learning algorithms' average reward performance to Motivated Learning with $S_{Rate} = 1.0$.

Figs. 12-14 compare the algorithms using rate value S_{Rate} set at 2,4, and 8, respectively. We can observe that as the rate of change of the primitive reward S_{Rate} increases, the advantage of TD-FALCON over other RL algorithms diminishes, while NFQ performance improves.

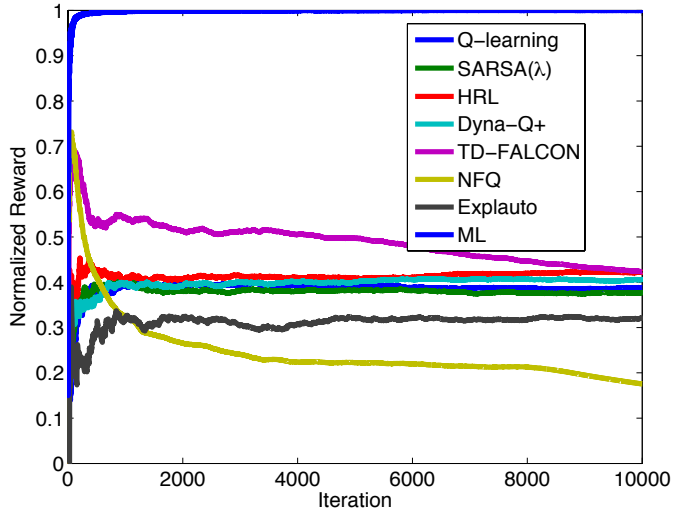

Fig. 12. Comparison of Reinforcement Learning algorithms' average reward performance to Motivated Learning with $S_{Rate} = 2.0$.

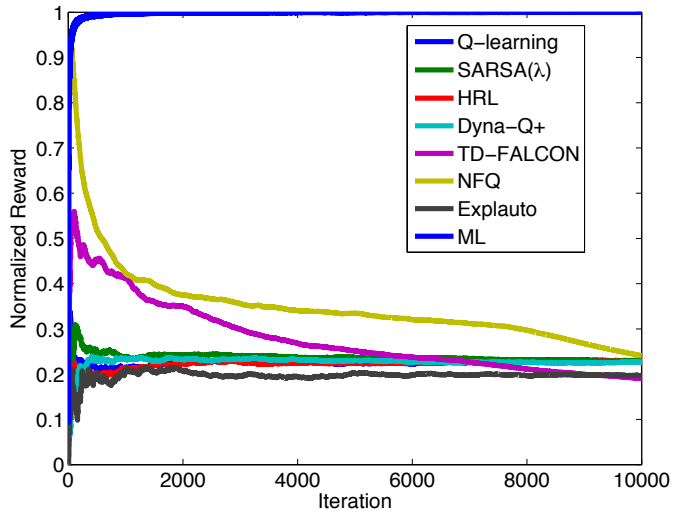

Fig. 13. Comparison of Reinforcement Learning algorithms' average reward performance to Motivated Learning with $S_{Rate} = 4.0$.

Although NFQ seems to perform better that other RL algorithms if the resource decline rate is higher than 4, we can observe its gradual decline with the increasing number of iterations. Fig. 15 shows the comparison between ML, NFQ, TD-FALCON, and Q-Learning methods for $S_{Rate} = 4.0$ with 20,000 iterations. We see that both advanced RL methods eventually fall below Q-learning, which indicates that in a longer run they are not capable of maintaining their advantage over other RL algorithms in this testing scenario. However, the ML algorithm was able to continue operating without declining performance

With these results we have shown that the Motivated Learning algorithm performs favorably against several common reinforcement learning algorithms. The results support our assertion that ML outperforms RL in complex environments, particularly, when an agent needs to discover the relations between several different "resources" and is only provided feedback in terms of the environment state and a single "reward" signal.

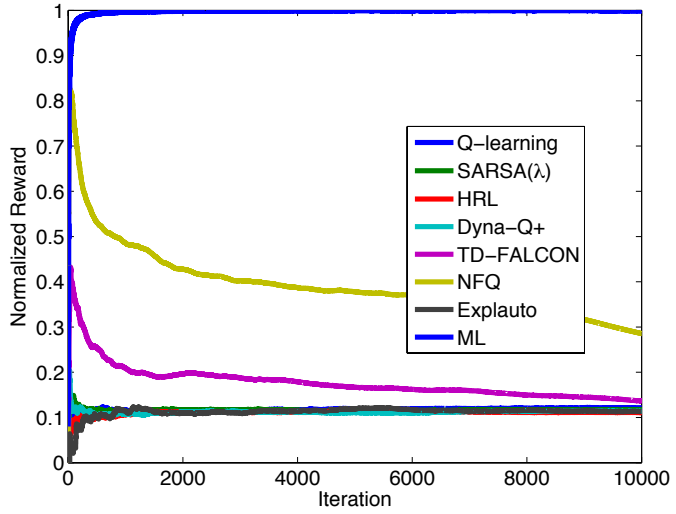

Fig. 14. Comparison of Reinforcement Learning algorithms' average reward performance to Motivated Learning with $S_{Rate} = 8.0$.

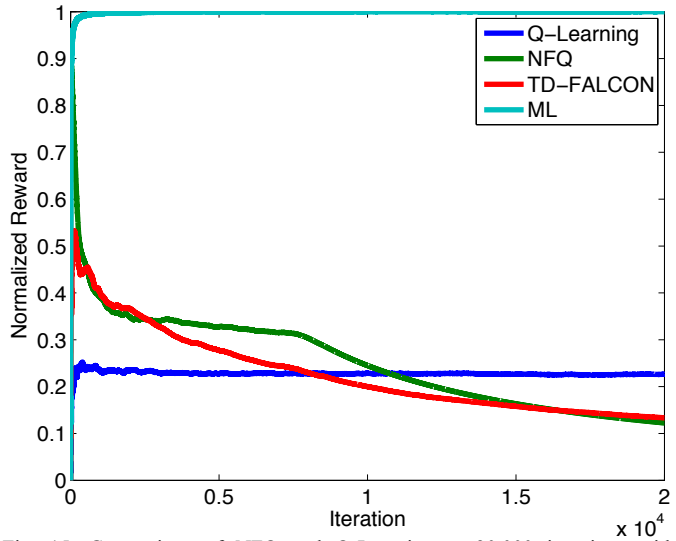

Fig. 15. Comparison of NFQ and Q-Learning at 20,000 iterations with S_{Rate}4.0.

The Black box scenario is open to anyone wishing to conduct their own experiments at:

http://ncn.wsiz.rzeszow.pl/autonomous-learning-challenge/

Please feel free to run the experiment yourself and let us know if your results are better than what we reported.

IV. Conclusion

In this paper we argued that a Motivated Learning approach is able to effectively solve resource sharing and task coordination in a dynamic environment. This is because a ML agent's internal reward system motivates it to act effectively in a dynamically changing environment with limited resources and respond to potential adversarial actions by other agents. We compared our motivated learning approach with several RL algorithms using a black box scenario. We obtained results in the form of reward values provided to the agents. Using different resource settings we showed that our ML algorithm was able to perform well even when running into time constraints.

We plan to extend the black box scenario's complexity by including actions by other agents (that can either cooperate or compete with the agent under test). Another addition to the scenario may be the concept of "space" in order to find objects in the environment. While our ML implementations handle actions by other agents as well as finding objects in the environment, both features were not included in the black box scenario. This is to keep the black box scenario as compatible as possible with the various RL algorithms.

Acknowledgements: We would like to acknowledge C. Moulin-Frier, and M. Riedmiller for their aid at helping generate data using their reinforcement learning algorithms. This research was supported by The National Science Centre, grant No. 2011/03/B/ST7/02518.

REFERENCES

[1] T.-H. Teng, A.-H. Tan, J. A. Starzyk, Y.-S. Tan and L.-N. Teow, "Integrating Motivated Learning and k-Winner-Take-All to coordinate Multi-Agent Reinforcement Learning," in Proceedings of IAT, pp. 190-197, Warsaw, August 2014.

[2] J. Perez, C. Germain-Renaud, B. Kegl, C. Loomis, "Multi-Objective Reinforcement Learning for Responsive Grids," Journal of Grid Computing, vol. 8, no. 3, pp. 473-492, 2010.

[3] C.-K. Ngai and H.-C. Yung, "A Multiple Goal Reinforcement Learning Method for Complex Vehicle Overtaking Maneuvers, IEEE Trans. on Intelligent Transportation Syst., vol. 12, no. 2, pp. 509-522, June 2011.

[4] S. Sen and M. Sekaran, "Multi-agent coordination with learning classifier systems," Adaptation and Learning in Multi-agent systems, pp. 218- 233, 1996.

[5] H.-L. Guo and Y. Meng, "Distributed Reinforcement Learning for Coordinate Multi-Robot Foraging," Journal of Intelligent Robotic Syst. vol. 60, pp. 531-551, 2010.

[6] V. Lesser, et all. "Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework," Autonomous Agents and Multi-Agent Systems, vol. 9, pp. 87-143, 2004.

[7] A. Burkov and B. Chaib-Draa, "Adaptive Play Q-Learning with Initial Heuristic Approximation," in Proc. of ICRA, 2007, pp. 1749-1754.

[8] F. S. Melo and M. Veloso, "Learning of coordination: exploiting sparse interactions in multi-agent systems," in Proc. of AAMAS, 2009, pp. 773-780.

[9] C. B. Excelente-Toledo and N. R. Jennings, "Using reinforcement learning to coordinate better," Computational Intelligence, vol. 21, no. 3, pp. 217-245, 2005.

[10] J. A. Starzyk, J. T. Graham, P. Raif, and A-H.Tan, "Motivated Learning for Autonomous Robots Development", Cognitive Science Research, v.14, no.1, 2012, pp. 10-25.

[11] J. A. Starzyk, Motivated Learning for Computational Intelligence, in Computational Modeling and Simulation of Intellect: Current State and Future Perspectives, IGI Publishing, ch.11, pp. 265-292, 2011.

[12] J. A. Starzyk, "Motivation in Embodied Intelligence," in Frontiers in Robotics, Automation and Control, I-Tech Education and Publishing, Oct. 2008, pp. 83-110.

[13] I. Szita, B. Takacs, and A. Lorincz, "ε-MDPs: Learning in varying environ-ments", J. of Machine Learning Research, vol. 3, pp. 145-174, 2002.

[14] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," in Advances in Neural Information Processing Systems 19, B. Scholkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 1–8.

[15] D. P. Bertsekas, "Approximate policy iteration: A survey and some new methods," Massachusetts Institute of Technology, Cambridge, US, Tech. Rep. LIDS 2833, July 2010.

[16] P. Abbeel and A. Y. Ng,"Apprenticeship Learning via Inverse Reinforcement Learning", in Proc. of the Twenty-first Int. Conf. on Machine Learning (ICML'04), pp. 1-8, 2004.

[17] A. Boularias, J. Kober, and J. Peters, "Relative entropy inverse reinforcement learning," in Proceedings of the Artificial Intelligences and Statistics (AISTATS), pp. 20- 27, 2011.

[18] K. Muelling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," The Int. Journal of Robotics Research, vol. 32, no. 3, pp. 263 -279, 2013.

[19] S. Zhifei and E. Joo, "A survey of inverse reinforcement learning techniques," International Journal of Intelligent Computing and Cybernetics, vol. 5, no. 3, pp. 293-311, 2012.

[20] J. A. Bagnell and J. C. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods" in IEEE Int. Conf. on Robotics and Automation (ICRA), 2001.

[21] Coelho, J. A., Araujo, E. G., Huber, M., and Grupen, R. A., Dynamical categories and control policy selection.Proceedings of IEEE International Symposium on Intelligent Control, 1998, pp. 459–464.

[22] J. Schmidhuber, "Curious model-building control systems," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, pp. 1458–1463, Singapore, vol. 2, 1991.

[23] Oudeyer, P-Y., Kaplan, F., & Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. IEEE Transactions on Evolutionary Computation, 11, 265–286.

[24] A. Barto, A. Singh, S. & Chentanez, N, "Intrinsically motivated learning of hierarchical collections of skills," in *Proc. 3rd Int. Conf. Development Learn*., pp. 112–119, San Diego,CA, 2004.

[25] M. Hasenjager and H. Ritter, "Active Learning in Neural Networks". Berlin, Germany: Physica-Verlag GmbH, Physica-Verlag Studies In Fuzziness and Soft Computing Series, pp. 137–169, 2002.

[26] K. Merrick and M. L. Maher, Motivated Reinforcement Learning: Curious Characters for Multiuser Games. Berlin: Springer, 2009.

[27] J. Graham, J.A. Starzyk, D. Jachyra, "Opportunistic Behavior in Motivated Learning Agents", to appear in *IEEE Trans on Neural networks and Learning Systems*, 2015.

[28] T. M. Mitchell, "Machine Learning", New York, NY, USA: McGraw-Hill, 1997.

[29] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA, 1998.

[30] T. G. Dietterich, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", Journal of Artificial Intelligence Research, vol. 13, 227-303, 2000.

[31] C. Moulin-Frier, P. Rouanet, P.-Y. Oudeyer, "Explauto: an open-source Python library to study autonomous exploration in developmental robotics," in *Proc. International Conference on Development and Learning*, Genova, Italy, 2014.

[32] A.-H. Tan, N. Lu and X. Dan, "Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback," IEEE Trans. Neural Networks, vol. 19, no. 2, pp. 230-244, Feb, 2008.

[33] T. Gabel, C. Lutz, M. Riedmiller, "Improved Neural Fitted Q Iteration Applied to a Novel Computer Gaming and Learning Benchmark," in *Proc. IEEE Symp. on Approximate Dynamic Programming and Reinforcement Learning*, Paris, 2011, pp. 279-286.