A Tensor Factorization Approach to Generalization in Multi-Agent Reinforcement Learning

Stefano Bromuri
Department of Business Information Systems
University of Applied Sciences Western Switzerland
Switzerland, 3 Technopole, 3960, Sierre
Email: stefano.bromuri@hevs.ch

***Abstract*—Reinforcement learning (RL) and multi-agent reinforcement learning (MARL) are disciplines concerned with defining automatically the behaviour of an agent, or a set of interacting agents, by means of reward mechanisms coming from the environment. An important research issue in the context of RL and MARL is the definition of approaches to combine the knowledge of multiple learning agents to improve the overall performance of the multi-agent system (MAS). This paper illustrates how to improve RL and MARL algorithms by utilizing results from multi-linear algebra such as tensors and tensor factorizations. In particular, the focus is on showing how to modify existing algorithms from literature to include a tensor factorization step applied to the Q-Tables learned by the individual agents to generalize the knowledge about the actions performed in the environment. The modified algorithms are then evaluated in three RL and MARL scenarios against their unmodified version to show the benefits of the tensor factorization step.**

***Keywords*-Software Agents; Learning Systems; Algorithms; Dynamic Programming;**

I. Introduction

In Reinforcement learning (RL) [1], an agent interacts with an environment to maximize its reward in the long-term. To achieve this, RL requires a precise definition of states, actions and rewards. As stated in [1], the learner is not told what to do, it rather has to learn what are the best actions to perform according to the rewards provided by the environment in which it is situated. Furthermore, multi-agent RL (MARL) [2] is concerned with the problem of combining multiple reinforcement learners that interact in a common environment. An interesting issue of RL systems is the one of *generalization* [3]. Generalization is the problem of defining an approximation function to use experience and the perceptions that the agents have of the environment, to bias in a fruitful way the search for an optimal solution. In the MARL case, the generalization problem is also concerned with combining the knowledge of multiple agents about the environment.

To the best of the author's knowledge, an approach to generalization in RL involving multi-linear algebra and tensor factorization has never been attempted before. Tensor factor models have been previously applied in chemometrics [4], face recognition [5], gait recognition [6], and many other applications, see [7] for a review. Recently, the interest on multi-way factor models, such as Tucker's decomposition [8], has been revived due to their capability to consider the natural representation of spatio-temporal data. In particular, in those problems where it is possible to have a tensorial representation, multi-way factor models allow to simplify the problem at hand by projecting the raw multidimensional data to a low dimensional space that then can be analyzed with standard techniques to infer properties about the original problem. The contribution of this paper is not on defining new algorithms for RL or MARL. The contribution of this paper resides in showing how to apply tensor factorization in single and multi-agent settings, improving existing RL algorithms by introducing a further learning step to allow the agents to generalize a model of their actions and bias their learning in the environment.

To illustrate the effectiveness of this approach, existing RL and MARL algorithms are compared with their version modified with a tensor factorization step, in well known RL and MARL scenarios such as the Cart Pole, DYNA maze [9] and Robot Rescue [10] scenarios. The rest of this paper is structured as follows: Section II introduces the relevant background for this paper; Section III illustrates how to add tensor factorization to RL algorithms in multi-agent settings; Section IV evaluates the proposed approach in three different scenarios, showing the benefits of including the tensor factorization step; Section V puts this work in comparison with other relevant contributions in the RL area; Finally, Section VI concludes this paper and draws the lines for future works.

II. Background

A. Reinforcement Learning

Reinforcement learning can be defined in terms of Markov Decision Processes (MDPs).

Definition 1. *(Markov Decision Process) A MDP is a tuple* $\langle S, A, p, r \rangle$*, for which* S *is a set of environment states,* A *is a set of possible actions the agent can perform,* p *is the probability of moving from one state to another given an*

action that can be expressed as $p : S \times A \times S \to \mathbb{R}$ *, and* r *is the reward function associated to the transition that can be expressed as* $r : S \times A \times S \to \mathbb{R}$.

The behavior of an agent in its environment is defined by its action selection policy π, which is a function $\pi : S \to A$. A policy is stationary if it does not evolve in time.

$$V^{\pi}(s) = E\left\{\sum_{t=0}^{\infty} \gamma^k r_{k+1} \middle| s_0 = s, \pi\right\} \quad (1)$$

$$V^{*}(s) = E\left\{\sum_{t=0}^{\infty} \gamma^k r_{k+1} \middle| s_0 = s, \pi^*\right\} \quad (2)$$

Equation 1 reports the discounted reward V^{π} expected at state s, where $\gamma \in (0, 1]$ represents a discount factor that forces the recent rewards to be more important than the old ones, while equation 2 reports the optimal reward V^* at state s. The agent's goal is to find a policy π that maximizes this reward. In other words, the agent should optimize the long term reward, by only considering the short term rewards, so to asymptotically reach the optimal reward function. One algorithm achieving this is Q-Learning [11], shown in Fig. 1.

1: Initialize $Q(s, a)$, $\forall s \in S, a \in A$
2: **while** s' is not terminal **do**
3: $s \leftarrow current(nonterminal)state$
4: $a \leftarrow \pi(s)$
5: execute a, observe s' and the reward r
6: $Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t)+$
7: $\alpha_t \left[r(s_t, a_t) + \gamma \widehat{V}_t(s_{t+1}) - Q_t(s_t, a_t)\right]$
8: **end while**

Figure 1. The Q-Learning Algorithm.

where $\widehat{V}_t(s_{t+1}) = max_a[Q_t(s_{t+1}, a)]$ is the optimal expected reward estimate $V^*(s_{t+1})$.

In particular, as reported in [2], [11], for equation 3

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t)+ \alpha_t \left[r(s_t, a_t) + \gamma \widehat{V}_t(s_{t+1}) - Q_t(s_t, a_t)\right] \quad (3)$$

to converge to a global optimum, the following conditions must be satisfied: a) explicit, distinct values of the Q-function are stored and updated for each state-action pair; b) the sum $\sum_{t=0}^{\infty} \alpha_t(s, a)$ is infinite and the sum $\sum_{t=0}^{\infty} \alpha_t^2(s, a)$ converges; c) all the state-action pairs are visited infinitely often.

The first condition is usually satisfied by implementing a lookup table (Q-Table). The second condition is satisfied if the learning rate $\alpha_t(s, a) \in (0, 1]$ as also reported in [11]. The third condition is generally satisfied by applying a policy where the probability of a random action selection is $\epsilon > 0$ and the probability of selecting the best action is $1 - \epsilon$. An interesting variation of Q-Learning is provided by the SARSA algorithm [12]. SARSA is very similar to Q-Learning except that to update the Q-Table, the following equation is used:

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t)+ \alpha_t \left[r(s_t, a_t) + \gamma Q_{t+1}(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\right] \quad (4)$$

The fact that in equation 4 no max_a function is used allows SARSA to converge faster to a solution than Q-Learning if the number of actions to perform is large. Another important concept in RL is the one of *eligibility trace*. As reported in [1], an eligibility trace is an additional memory variable associated to each state taking into consideration the frequency with which each state is visited. An eligibility trace can be expressed as shown in equation 5.

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t & \text{if } s \neq s_t \\ \gamma\lambda e_t + 1 & \text{if } s = s_t \end{cases} \quad (5)$$

Where γ represents a discount factor and $\lambda \in [0, 1]$ a decaying factor for the trace defining how much the strength of a choice in the past should be discounted. The eligibility trace can then be used to define the SARSA(λ) algorithm [1] as shown in Fig. 2.

1: Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$, $\forall s \in S, a \in A$
2: **while** s' is not terminal **do**
3: $s \leftarrow current(nonterminal)state$
4: $a \leftarrow \pi(s)$
5: execute a, observe s' and the reward r
6: $\delta \leftarrow r + \gamma Q_{t+1}(s', a') - Q_t(s, a)$
7: $e_{t+1}(s, a) \leftarrow e_t(s, a) + 1$
8: **for** all $s \in S$ **do**
9: $Q_{t+1}(s, a) \leftarrow +\alpha\delta e(s, a)$
10: $e_{t+1}(s, a) \leftarrow \gamma\lambda e_t(s, a)$
11: **end for**
12: $s \leftarrow s'$
13: **end while**

Figure 2. The SARSA(λ) algorithm.

In such an algorithm, at each moment the current δ is assigned to each state according to its eligibility trace. The use of eligibility traces allows for a faster convergence to a global optimum than Q-Learning or SARSA alone, but this comes at the computational cost of keeping the information about past visited states.

B. Tensors and Tucker's Decomposition

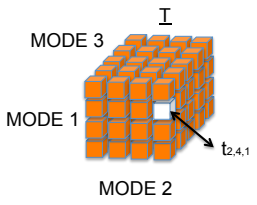

Figure 3. A tensor $\underline{T} \in \mathbb{R}^{4\times4\times5}$.

In what follows capital underlined letters ($\underline{T}$) represent tensors, while capital letters represent matrices (M), and

lower case letters represent the elements of a matrix or of a tensor (y_{ijk}).

As reported in [13], a tensor is defined as:

Definition 2. *(Tensor) Let $I_1, I_2, \ldots, I_N \in \mathbb{N}$ denote index upper bounds. A tensor $Y \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ of order N is an N-way array where elements $y_{i_1,i_2,\ldots,i_n}$ are indexed by $i_n \in 1, 2, ..., I_n$ for $1 \leq n \leq N$.*

Tensors are generalization of vectors and matrices, for example third order tensors have three modes or indexes as shown in Fig. 3. In particular, as also defined in [13], the concept of *tensor slice* is specified as:

Definition 3. *(Tensor Slice) A tensor slice is a two-dimensional section (fragment) of a tensor, obtained by fixing all indexes except for two indexes.*

Tucker's decomposition [8] is of particular importance for this paper. Tucker's decomposition is a form of higher-order principal component analysis (PCA), which removes correlation across the modes of the tensor rather than across the rows of a matrix, like PCA does. It achieves this by decomposing a tensor into a core tensor multiplied to a matrix along each mode, as shown in Fig. 4. Considering the three-way case, with a tensor $\underline{T} \in \mathbb{R}^{X \times Y \times Z}$, Tucker's decomposition is defined as:

$$\underline{T} \approx \underline{C} \times_1 E \times_2 F \times_3 G = \sum_{i=0}^{I} \sum_{j=0}^{J} \sum_{k=0}^{K} c_{ijk} e_i \circ f_j \circ g_k \quad (6)$$

where I, J and K are the components of the projection matrices E,F, G, $E \in \mathbb{R}^{X \times I}$, $F \in \mathbb{R}^{Y \times J}$, $G \in \mathbb{R}^{Z \times K}$ and $\circ$ is the outer product. Such components have to be decided according to the best model that represent the data set, usually depending on the application at hand.

There are several working implementation of Tucker's decomposition, in particular the Tucker3 algorithm, as provided in the tensor toolbox [14], is based on an alternating least square approach to compute the best Tucker's decomposition given the desired dimensions of the projection matrices, that have to be entered as an input.

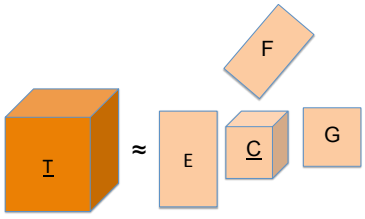

Figure 4. Tucker's decomposition.

The fact that Tucker's decomposition behaves like a higher order PCA implies that the sub-spaces contain the factor loadings, which represent the correlation coefficients between the variables in the tensor modes and the factors. The squared factor loadings represent the percentage of variance in that indicator variable explained by the factor. In other words, the intuition behind using Tucker's decomposition in RL is that the Q-Tables of one agent at each episode and the Q-Tables of different agents are correlated between each other as they represent different aspects of the same problem. Introducing a tensor factorization step, by considering the Q-Tables as slices of a tensor, will help to remove such correlation amongst the Q-Tables, returning tensor sub-spaces, where the columns of such sub-spaces are factor loadings characterizing states, actions, episodes, agents which are going to be used to bias the RL algorithms.

III. Combining Tensor Factorization with Q-Learning Algorithms

In what follows, the author shows how to include the information obtained through Tucker's decomposition in the Q-Learning algorithm, as it is the simplest one for illustration purposes. Later in this Section, the same approach is applied to more complex algorithms.

Init $\underline{MultiQ}(s, a, ag)$ $\forall s \in S, a \in A, ag \in Agents$
Init $\omega(a) = 1$ $\forall a \in A$
Set max number of episodes, $maxepisodes$
Set number of Q-Tables to factorize, $nrqtables$
Init $\underline{MQT}(ep, ag, s, a), ep \in E, |E| = nrqtables, \forall s \in S, a \in A, ag \in Agents$ ▷ a 4-way tensor
for $i \leftarrow 1, maxepisodes$ **do**
  for $j \leftarrow 1, |Agents|$ **do**
    $s_t \leftarrow currentstate$
    $a_t \leftarrow \epsilon greedy(s)$
    execute a, observe s' and the reward r
    $Q_t \leftarrow \underline{MultiQ}(:, :, j)$
    $Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha_t \omega_t(a)[r(s_t, a_t) +$
      $\gamma max_a[Q_t(s_{t+1}, a)] - Q_t(s_t, a_t)]$
    $\underline{MultiQ}(:, :, j) \leftarrow Q_{t+1}$
    $\underline{MQT}(:, :, mod(i, nrqtables), j) \leftarrow Q_{t+1}$
  end for
  if $mod(i, nrqtables)$==0 **then**
    $\underline{MQT} \approx \underline{Core} \times_1 S_p \times_2 A_p \times_3 E_p \times_4 Ag_p$ ▷ Tucker
    $\omega_t \leftarrow \|A_p\|_{row} / \|A_p\|$
    $indexmax \leftarrow max(\|Ag_p\|_{row})$ ▷ max variance explained.
    $Q_{max} \leftarrow \underline{MultiQ}(:, :, indexmax)$
    $\forall ag \in Agents, \underline{MultiQ}(:, :, ag) \leftarrow Q_{max}$
    reinitialize $\underline{MQT}$
  end if
end for

Figure 5. Multi Agent Q-Learning with 4-way tensor factorization.

Fig. 5 shows the modified version of Q-Learning that uses an additional tensor factorization step. The $\underline{MultiQ}$ tensor simply contains the current Q-Tables of all the agents. The tensor factorization step consists in storing at each episode the Q-Tables for each agent involved in the computation within a 4-way tensor $\underline{MQT} \in \mathbb{R}^{S \times A \times nrqtables \times nragents}$ where the dimensions are the number of states, number of actions, number of Q-Tables and number of agents on which Tucker's decomposition should be performed. After the factorization, the $\underline{MQT}$ tensor is emptied to accept new Q-Tables produced during the learning of the agents.

The decomposition of $\underline{MQT}$ produces four sub-spaces in terms of matrices: S_p as the states subspace, A_p as the actions subspace, E_p as the episodes subspace and Ag_p as the agent sub-space.

The actions sub-space, the matrix A_p, represents in each row an action and in each column the action factor loadings. It is then possible to calculate the Frobenius norm of each row of A_p, where the Frobenius norm for a vector or a matrix is defined as: $\|A\|_{\mathrm{F}} = \sqrt{\sum_{i,j} |a_{ij}|^2}$

representing the contribution of each action in the sub-space, subdivided by the Frobenius norm of the whole action subspace A_p, in order to have each action represented with a number in the range between [0,1]. The obtained values are stored in the ω vector. The ω vector is then used to tune the learning rate α according to the estimation of the contribution of each action. Furthermore, being the elements of ω in the range between [0,1], and being α between [0,1], their product is always between [0,1] and the convergence of this algorithm is ensured by the already discussed convergence constraints of Q-Learning.

The idea is that an action explaining a lot of variance is going to have a bigger coefficient in ω vector, implying that the agent will learn faster when selecting that action. Furthermore, there is also the need to update such ω vector as it may happen that during the learning the importance of each action changes according to the topology, behavior or state of the environment.

The use of a 4-way tensor to store the experience of multiple agents allows for an evaluation of the action weights in the ω vector by considering the perspective of multiple agents. This algorithm also implies further parameters in input to the Q-Learning algorithm that have to be empirically determined according to the application: the number of successive Q-Tables per agent necessary to periodically recalculate the ω vector, the number of agents involved in the simulation and the dimensions of Tucker's decomposition. The exact number of Q-Tables necessary to include in the tensor factorization to have a good estimation of ω is dependent on the properties of the application at hand and the same can be said for the dimensions of the sub-spaces of Tucker's decomposition.

In addition to calculating ω, it is possible to calculate each agent contribution to the variance in the agent subspace Ag_p. Remembering that Tucker's decomposition removes correlation, the idea is that if the Q-Table of an agent explains more variance than other agents in the Ag_p subspace, it means that its Q-Table is less noisy, and it explains better how to reach the solution than the other Q-Tables. Consequently, its Q-Table, represented in the algorithm in Fig. 5 as Q_{max}, should be the one used for to the next cycle by all the agents. There are other possibilities to use the sub-spaces of the tensor factorization, but a complete analysis of all these possibilities is a matter of future work. This Section continues by showing how the tensor factorization step can be applied to existing RL algorithms.

A. Modifying the DYNA Architecture

RL and MARL algorithms can work without an MDP model, but having a model can speed up learning, by biasing the exploration. A learned model can also be useful to do more efficient value updating. In general, model learning is useful to estimate the dynamics of the environment. The DYNA architecture [9] implements a way to use a model to improve the experience of the agent, by interleaving Q-Learning with extra updates using a model, which is constantly updated too. The result is that DYNA needs less interactions with the environment, as it reuses experience to update the Q-Table more often.

initialize $Q(s,a)$ and $Model(s,a)$ arbitrarily $\forall s \in S, a \in A$
for $n \leftarrow 1$, $maxepisodes$ **do**
  $s_t \leftarrow S$ current state
  $a_t \leftarrow \epsilon$-greedy(s,Q)
  execute a, observe s' and the reward r
  $Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t)+$
    $\alpha_t \left[r(s_t,a_t) + \gamma \widehat{V}_t(s_{t+1}) - Q_t(s_t,a_t) \right]$
  $Model(s_t,a_t) \leftarrow s'_t, r$
  for $i \leftarrow 1$, n **do**
    $a_t \leftarrow h_a(s_t,H)$ ▷ Learning worst plans during planning
    if $s_t, a_t \notin Model$ **then**
      $s_t \leftarrow$ randomly selected observed state
      $a_t \leftarrow$ random, previously selected action from s
    end if
    $s'_t, a_t \leftarrow Model(s_t,a_t)$
    $Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t)+$
      $\alpha_t \left[r(s_t,a_t) + \gamma \widehat{V}_t(s_{t+1}) - Q_t(s_t,a_t) \right]$
  end for
end for

Figure 6. The DYNA-H algorithm, adapted from [15].

As shown in Fig. 6, the DYNA-H algorithm, defined by Santos et al. [15], follows the DYNA architecture by including a model of the environment, and adding to DYNA a learning step in the planning phase using the function $h_a(s,H)$. The $h_a(s,H)$ function uses an heuristic function H to sample the worst plans in the model, which leads to get the worst simulated rewards allowing the agent to converge quickly to a good solution by avoiding these actions at execution time, as reported in [15]. In this paper, the DYNA-H algorithm is extended with an additional tensor factorization step used as a generalization mechanism to estimate the learning rate associated to each action available to the agents. Fig. 7 shows the DYNA-HT algorithm. Except for the fact that each agent has its own environment model stored in the $\underline{MModel}$ tensor, and that DYNA-HT uses the same $h_a(s,H)$ planning function of DYNA-H, the same considerations discussed for the algorithm in Fig. 5 apply for the ω vector of DYNA-HT.

B. Modifying SARSA(λ)

This Section illustrates how the SARSA(λ) algorithm presented in Section II has been modified to include a tensor factorization step.

Init $MultiQ(s,a,ag)$, $\underline{MModel}(s,a,ag)\ \forall s \in S, a \in A, ag \in Agents$
Init $\overline{\omega(a) = 1}\ \forall a \in A$
Set max number of episodes, $maxepisodes$
Set number of Q-Tables to factorize, $nrqtables$
Init $\underline{MQT}(ep,ag,s,a), ep \in E, |E| = nrqt, \forall s \in S, a \in A, ag \in Agents$
for $n \leftarrow 1, maxepisodes$ **do**
  for $j \leftarrow 1, |Agents|$ **do**
    $Q_j, t \leftarrow \underline{MultiQ}(:,:,j)$
    $Model_j \leftarrow \underline{MModel}(:,:,j)$
    $s_j \leftarrow S$ current state
    $a_j \leftarrow \epsilon$-greedy(s, Q_j)
    execute a, observe s' and the reward r
    $Q_{j,t+1}(s_{j,t}, a_{j,t}) \leftarrow Q_{j,t}(s_{j,t}, a_{j,t}) +$
      $\alpha_t \omega_t(a) \left[r(s_{j,t}, a_{j,t}) + \gamma \widehat{V}_{j,t}(s_{j,t+1}) - Q_{j,t}(s_{j,t}, a_{j,t}) \right]$
    $Model_j(s,a) \leftarrow s', r$
    for $n \leftarrow 1$, n **do**
      $a \leftarrow h_a(s, H)$ ▷ Learning worst plans during planning
      if $s, a \notin Model_j$ **then**
        $s \leftarrow$ randomly selected observed state
        $a \leftarrow$ random, previously selected action from s
      end if
      $s', a \leftarrow Model_j(s,a)$
      $Q_{j,t+1}(s_{j,t}, a_{j,t}) \leftarrow Q_{j,t}(s_{j,t}, a_{j,t}) +$
        $\alpha_t \omega_t(a) \left[r(s_{j,t}, a_{j,t}) + \gamma \widehat{V}_{j,t}(s_{j,t+1}) - Q_{j,t}(s_{j,t}, a_{j,t}) \right]$
    end for
    $\underline{MultiQ}(:,:,j) \leftarrow Q_{j,t+1}$
    $\underline{MModel}(:,:,j) \leftarrow Model_j$
    $\underline{MQT}(mod(n, nrqtables), j, :, :) \leftarrow Q_{j,t+1}$
    if $mod(n, nrqt)$==0 **then**
      $\underline{MQT} \approx \underline{Core} \times_1 S_p \times_2 A_p \times_3 E_p \times_4 Ag_p$ ▷ Tucker
      $\omega_t \leftarrow \|A_p\|_{row} / \|A_p\|$
      $indexmax \leftarrow max(\|Ag_p\|_{row})$
      $Q_{max} \leftarrow \underline{MultiQ}(:,:,indexmax)$ ▷ max variance explained.
      $\forall ag \in Agents, \underline{MultiQ}(:,:,ag) \leftarrow Q_{max}$
      reinitialize $\underline{MQT}$
    end if
  end for
end for

Figure 7. The Multi-Agent DYNA-HT algorithm.

Fig. 8 shows TSARSA(λ). With respect to SARSA(λ), the algorithm in Fig. 8 presents the following changes: a) the $\underline{MultiQ}$ tensor simply contains the Q-Tables of every agent at a certain episode; b) the $\underline{MultiE}$ tensor simply contains the eligibility trace of every different agent at a certain episode; c) since TSARSA(λ) is going to be used for control problems, the Q-Tables of every agent are periodically updated to the value of the Q-Table of the agent that can explain the least variance in the tensor agent sub-space. The last change is necessary as it implies selecting the agent with the noisiest Q-Table in the tensor, meaning that it explored the largest number of states with respect to the other agents, and consequently it is more likely to find a good sequence of actions to repeat in the next cycle. In particular, concerning the convergence of the algorithm TSARSA(λ), the focus is on tuning the learning rate α keeping it in the range between (0,1], to ensure no convergence problems.

IV. Evaluation

To evaluate the effects and scalability of the tensor factorization step, three scenarios of growing complexity are considered. The first scenario is the Cart Pole scenario, the second one is DYNA maze defined by Sutton et al. in [9], while the third one is the MARL cooperative robot rescue

Init $MultiQ(s,a,ag)\ \forall s \in S, a \in A, ag \in Agents$
Init $\underline{MultiE}(s,a,ag) = 0, \forall s \in S, a \in A, ag \in Agents$
Init $\omega(a) = 1\ \forall a \in A$
Set max number of episodes, $maxepisodes$
Set number of Q-Tables to factorize, $nrqtables$
Init $\underline{MQT}(ep,ag,s,a), ep \in E, |E| = nrqtables, \forall s \in S, a \in A, ag \in Agents$ ▷ a 4-way tensor
for $i \leftarrow 1, maxepisodes$ **do**
  for $j \leftarrow 1, |Agents|$ **do**
    $Q_{j,t}(:,:) \leftarrow \underline{MultiQ}(:,:,j)$
    $e_{j,t}(:,:) \leftarrow \underline{MultiE}(:,:,j)$
    $\delta \leftarrow r + \gamma Q_{j,t+1}(s', a') - Q_{j,t}(s,a)$
    $e_{j,t+1}(s,a) \leftarrow e_{j,t}(s,a) + 1$
    for all $s \in S$ **do**
      $Q_{j,t+1}(s,a) \leftarrow +\alpha \omega_t \delta e_{j,t}(s,a)$
      $e_{j,t+1}(s,a) \leftarrow \gamma \lambda e_{j,t}(s,a)$
    end for
    $\underline{MultiQ}(:,:,j) \leftarrow Q_{j,t+1}$
    $\underline{MultiE}(:,:,j) \leftarrow e_j, t+1$
    $\underline{MQT}(:,:,mod(i, nrqt), j) \leftarrow Q_{t+1}$
  end for
  if $mod(i, nrqtables)$==0 **then**
    $\underline{MQT} \approx \underline{Core} \times_1 S_p \times_2 A_p \times_3 E_p \times_4 Ag_p$ ▷ Tucker
    $\omega_t \leftarrow \|A_p\|_{row} / \|A_p\|$
    $indexmin \leftarrow min(\|Ag_p\|_{row})$ ▷ min variance explained.
    $Q_{min} \leftarrow \underline{MultiQ}(:,:,indexmin)$
    $e_{min} \leftarrow \underline{MultiE}(:,:,indexmin)$
    $\forall ag \in Agents, \underline{MultiQ}(:,:,ag) \leftarrow Q_{min}$
    $\forall ag \in Agents, \underline{MultiE}(:,:,ag) \leftarrow e_{min}$
    reinitialize $\underline{MQT}$
  end if
end for

Figure 8. TSARSA(λ): SARSA(λ) with a Tensor Factorization Step.

scenario. As already mentioned, the purpose of this paper is not, by any means, to create the best performing algorithm to solve a problem in RL, it is rather to show that tensor factorization approaches can be useful to generalize about the accumulated experience of one or more agents in order to tune the existing algorithm with a further learning step.

A. The Cart Pole Scenario

The Cart Pole scenario is a control problem where a mass is put on top of a massless rod attached to a cart that can move horizontally to balance the mass on the top. In particular, the environment is discretized to have 110 states and the agents can perform 21 different actions to balance the cart. The reward is defined in function of the angle described by the rod. If the angle is below 45 degrees, then the reward is positive, while it is negative otherwise. To evaluate the generalization capabilities of the tensor factorization step, SARSA(λ) is compared against its modified version TSARSA(λ). The parameters used for the evaluation are a discount factor γ of 0.95, a learning rate α of 1, a decay factor λ of 0.95 for the eligibility trace, an initial probability of random action selection ϵ equal to 0.001. For the tensor factorization, the update frequency of the ω vector was set to 10 episodes, and the chosen dimensions for the tensor sub-spaces obtained with Tucker's decomposition were: 10 columns for the state space, 10 columns for the action space, 9 columns for the episode space and 1 column for the agent space. The cumulative number of steps for which the agents can balance the cart

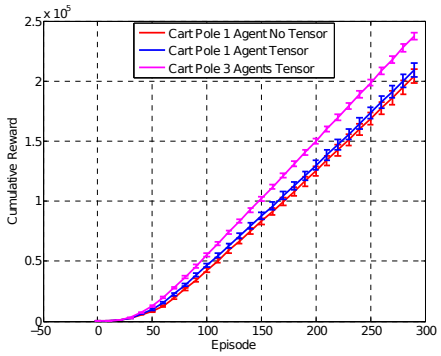

Figure 9. Cumulative Reward Curves.

pole was taken as an evaluation measure, averaging the results over 120 iterations. As shown in Fig. 9, the one agent TSARSA(λ) already succeeds in improving the basic SARSA(λ) algorithm, converging to an equilibrium faster than SARSA(λ), which is reflected in a higher cumulative number of steps at the end of the simulation.

Fig. 9 shows also that introducing multiple agents improves the results with respect to the single agent case, where to compute the curve we averaged the cumulative rewards of the agents. In particular for the multi-agent case multiple agents run concurrently and every 10 episodes their Q-Tables were factorized using the 4-way tensor factorization previously described. In the multi-agent case there are two aspects contributing: firstly, as in the single agent case, the ω vector, but this time using the perspective of many agents sharing the same goal; secondly, in the multi-agent case, the best performer is selected according to its contribution to the variance of 4-way tensor and its Q-table is used as the base point for the next episodes.

B. The DYNA Maze Scenario

This scenario uses randomly generated DYNA maze worlds, with 39×36 cells, where the goal is set to be in the [28,34] cell. In the DYNA maze world an agent has to find the exit in a maze. The agent does not know the maze in advance and the goal is to find the optimal path between the starting point and the exit. In particular an agent can move up, down, left or right, while the environment can have tiles that represent obstacles. The reward is set to be -1 for every tile that is not the goal. Every different world in the simulation is generated using a Gaussian probability distribution to decide where the walls are situated, with mean 0 and deviation 0.3. The parameters used for the evaluation are a discount factor γ of 0.95, a learning rate α of 0.1, a a probability of random action selection ϵ equal to 0.1, and an update frequency of the ω vector of 10 episodes. The dimensions chosen for the sub-spaces of Tucker's decomposition were: 15 columns for the state space, 3 columns for the action space, 9 columns for the episode space and 1 column for the agent space.

The left part of Fig. 10 shows the path found by DYNA-H to reach the goal in a randomly generated DYNA maze world. What is interesting is that DYNA-H manages to find a good direction towards the goal with an acceptable number of steps.

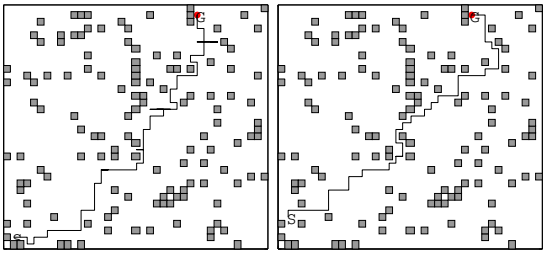

Figure 10. Single Agent Path finding with DYNA-H (on the left) and Single Agent Path finding with DYNA-HT (on the right).

The right part of Fig. 10 shows an example of the effect of applying tensor factorization in DYNA-HT. In the same environment used for DYNA-H, the single agent DYNA-HT requires only 83 steps to reach the goal, while DYNA-H requires 90 steps.

As expected, the tensor factorization step allows the agent to produce a generalization about the weight of the actions according to the experience of the agent. When three DYNA-HT agents are used to estimate the path, the obtained path has 59 steps. The use of multiple agents allows the DYNA-HT to combine the knowledge of many agents and generalize better about the weight of the actions in the simulation, ending up with a better result than the single agent case.

Fig. 11 shows the cumulative cost of finding the path to reach a fixed goal in the DYNA maze world for both DYNA-H and DYNA-HT. Such a test was conducted randomizing the generation of 10 different mazes, summing the steps of the path found in each of the mazes, and then averaging over 25 iterations. The evaluation of DYNA-HT includes also the multi-agent case. For the multi-agent case, to compute the curve we also average the steps of the paths found by the different agents.

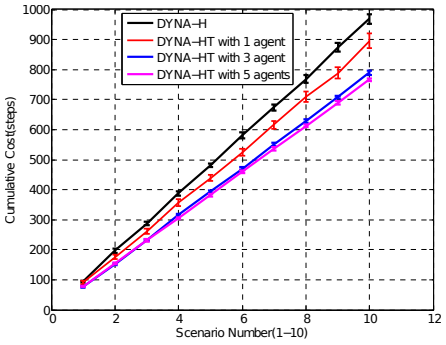

Figure 11. Cumulative Cost of DYNA-H vs DYNA-HT.

Fig 11 shows that the introduction of multiple agents sharing their Q-Tables in the 4-way tensor for the estimation

of the ω vector, allows the algorithm to find better solutions than in the single agent case.

C. The Robot Rescue Scenario

The Robot Rescue Scenario is taken from the MARL toolbox [10], and it involves two agents that must coordinate their actions to bring a rectangular object to a destination. Fig. 12 shows as schematic of this scenario. This scenario involves a 8×8 grid world. The parameters used for the evaluation are a discount factor γ of 0.95, a learning rate α of 0.1, a probability of random action selection ϵ equal to 0.1, a trace decaying factor λ of 0.95, an update frequency of the ω vector of 10 episodes, and the dimensions chosen for the sub-spaces of Tucker's decomposition were: 50 columns for the state space, 10 columns for the action space, 9 columns for the episode space and 1 column for the agent space.

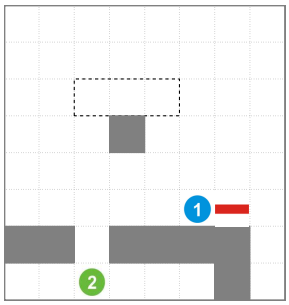

Figure 12. The Robot Rescue Scenario.

The fact that this scenario involves two agents that coordinate between each other implies a quite big state space as the agents can perform joint actions to bring the block to the destination, resulting in a Q-Table of 921600 elements. From the computational perspective, tensor factorization methods scale up quite well [16], consequently, the application of the tensor factorization step does not imply a big computational load on the algorithm even for large MARL problems. In particular, this scenario is already multi-agent, so in this case the Team-Q [17] algorithm, available in the MARL toolbox, is directly compared with its modified version Team-TQ. Team-Q uses a plain Q-Learning approach with eligibility traces.

Fig. 13 reports the cumulative costs of both algorithms. The cumulative cost of Team-TQ is improved with respect to Team-Q. The result implies that the use of a tensor factorization step happens to be effective in big MDPs, where there are many states upon which the agents apply their policies.

V. Related Work

To the best of the author's knowledge the use of tensor factorization in RL and MARL is a novel idea. However, modifying the agents' learning rate by means of generalizations or function approximations is not a new idea. In [18],

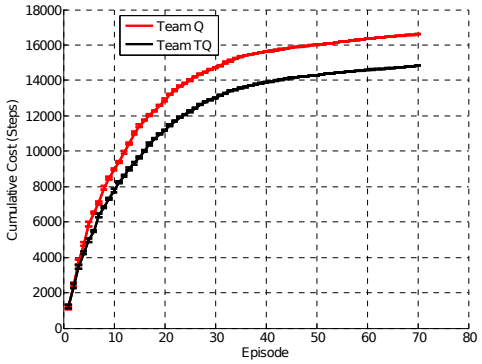

Figure 13. Cumulative Cost Curve in the Robot Rescue Scenario.

Bowling and Veloso present the WOLF algorithm, a MARL algorithm involving variable learning rates. In WOLF, the learning rate is adapted according to the performance of the agent. The main difference between the tensor factorization step and [18] is that the former estimates the learning rate for each action according to the explained variance, while in the latter the learning rates follow a well defined function that requires the notion of winning or losing.

In [19] da Motta and Anderson present a RL algorithm that uses a local radial basis function to extract features in terms of adjusting errors to tune the behavior or the RL algorithm. The approach presented in this paper follows a similar idea to the one of da Motta and Anderson, but rather than performing an estimation on the states, the tensor factoriation performs an estimation on the variance explained by each agent and each action in a tensor of Q-Tables.

In [20] Hester and Stone define the RL-DT algorithm combining reinforcement learning with decision trees as a generalization step to learn a model of the environment. The decision tree is used to predict, given the current state and action of the agent, the next state and reward received by the agent. The results shown in [20], demonstrate that the combination of decisions trees with reinforcement learning allows the agents to take better informed decisions about the exploration of the environment, when compared to standard model based algorithms. The tensor factorization step proposed in this paper for generalization purposes is not model based, it uses the accumulated knowledge of the agents to create a factorized tensor in an unsupervised manner. RL-DT uses supervised machine learning to classify the states.

The Heuristically Accelerated Minimax-Q (HAMMQ) is presented by Bianchi et al. in [21] for zero sum Markov games. HAMMQ introduce a heuristic function that influences the choice of the action by taking into consideration both the actions of the agent and the actions of its opponent in the environment. The tensor factorization step proposed does not use heuristic evaluations of the states, it estimates the contribution of each action to the variance of the Q-Tables in the tensor. A possibility is to define a heuristic

function to decide which action to take according to the variance explained in the states, rather than by the actions as currently done in this paper.

VI. Conclusion and Future Work

This paper presented a generalization step based on tensor factorization for RL and MARL scenarios. This step is based on Tucker's decomposition to estimate the weight of each action to bias the learning rates of the agents in the environment. Convergence is ensured as the learning rate is kept in the range $(0, 1]$. Existing algorithms modified with tensor factorization were compared with their original version in three scenarios of growing complexity, showing that tensor factorization is a viable approach to improve RL and MARL algorithms, but showing also that the approach can scale up with the size of the problem.

The author's hope is that the ideas presented in this paper will foster new research directions at the intersection between multi-linear algebra and reinforcement learning, allowing researcher to improve existing algorithms, but also to define new algorithms that use efficiently the information held by the sub-spaces obtained through the tensor factorization step.

Acknowledgement

This work was partially supported by the FP7 287841 COMMODITY12 project and by the Hasler Stiftung 10020 MONDAINE project.

References

[1] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[2] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," in *Innovations in Multi-Agent Systems and Applications – 1*, ser. Studies in Computational Intelligence, D. Srinivasan and L. Jain, Eds. Berlin, Germany: Springer, 2010, vol. 310, ch. 7, pp. 183–221.

[3] C. H. C. Ribeiro, "A Tutorial on Reinforcement Learning Techniques," in *Supervised Learning Track Tutorials of the 1999 International Joint Conference on Neuronal Networks*. Washington: INNS Press, 1999.

[4] E. Acar, R. Bro, and B. Schmidt, "New exploratory clustering tool," *Journal of Chemometrics*, vol. 22, no. 1, pp. 91–100, 2008.

[5] M. A. O. Vasilescu and D. Terzopoulos, "Multilinear analysis of image ensembles: Tensorfaces," in *ECCV (1)*, ser. Lecture Notes in Computer Science, A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, Eds., vol. 2350. Springer, 2002, pp. 447–460.

[6] D. T. D. Tao, X. L. X. Li, X. W. X. Wu, and S. J. Maybank, "General Tensor Discriminant Analysis and Gabor Features for Gait Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 10, pp. 1700–1715, 2007.

[7] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[8] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, pp. 279–311, 1966c.

[9] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *SIGART Bull.*, vol. 2, no. 4, pp. 160–163, Jul. 1991. [Online]. Available: http://doi.acm.org/10.1145/122344.122377

[10] L. Buşoniu, "The marl toolbox," visited in May 2012. [Online]. Available: http://busoniu.net/repository.php

[11] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[12] R. S. Sutton, "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding," in *NIPS*, D. S. Touretzky, M. Mozer, and M. E. Hasselmo, Eds. MIT Press, 1995, pp. 1038–1044.

[13] A. Cichocki, R. Zdunek, A. H. Phan, and S. ichi Amari, *Nonnegative Matrix and Tensor Factorizations - Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley, 2009.

[14] T. G. K. Brett W. Bader *et al.*, "Matlab tensor toolbox version 2.5," Available online, January 2012. [Online]. Available: http://www.sandia.gov/ tgkolda/TensorToolbox/

[15] M. Santos, J. A. M. H., and V. Lopez, "Dyna-H: a heuristic planning reinforcement learning algorithm applied to role-playing-game strategy decision systems," *CoRR*, vol. abs/1101.4003, 2011.

[16] E. Acar, D. M. Dunlavy, T. G. Kolda, and M. Mørup, "Scalable Tensor Factorizations for Incomplete Data," *Chemometrics and Intelligent Laboratory Systems*, vol. 106, no. 1, pp. 41–56, March 2011.

[17] M. L. Littman, "Value-function reinforcement learning in Markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55 – 66, 2001.

[18] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215 – 250, 2002.

[19] A. da Motta Salles Barreto and C. W. Anderson, "Restricted gradient-descent algorithm for value-function approximation in reinforcement learning," *Artificial Intelligence*, vol. 172, no. 45, pp. 454 – 482, 2008.

[20] T. Hester and P. Stone, "Generalized model learning for reinforcement learning in factored domains," in *AAMAS (2)*, C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, Eds. IFAAMAS, 2009, pp. 717–724.

[21] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Heuristic Selection of Actions in Multiagent Reinforcement Learning," in *IJCAI*, M. M. Veloso, Ed., 2007, pp. 690–695.