IEICE Transactions on Information and Systems,
vol.E95-D, no.10, pp.2426–2437, 2012.

Multi-Task Approach to Reinforcement Learning for Factored-State Markov Decision Problems

Jaak Simm
Tallinn University of Technology, Tallinn 152-8552, Estonia
jaak.simm@ttu.ee

Hirotaka Hachiya
Tokyo Institute of Technology, Japan.

Masashi Sugiyama
Tokyo Institute of Technology, Japan.
sugi@cs.titech.ac.jp
http://sugiyama-www.cs.titech.ac.jp/~sugi

Abstract

Reinforcement learning (RL) is a flexible framework for learning a decision rule in an unknown environment. However, a large number of samples are often required for finding a useful decision rule. To mitigate this problem, the concept of *transfer learning* has been employed to utilize knowledge obtained from similar RL tasks. However, most approaches developed so far are useful only in low-dimensional settings. In this paper, we propose a novel transfer learning idea that targets problems with high-dimensional states. Our idea is to transfer knowledge between *state factors* (e.g., interacting objects) within a single RL task. This allows the agent to learn the system dynamics of the target RL task with fewer data samples. The effectiveness of the proposed method is demonstrated through experiments.

Keywords

reinforcement learning, multi-task learning, transfer learning, factored state models, Markov decision process

1 Introduction

Recently, there is an increasing interest in methods of learning a decision rule in an unknown and stochastic environment. These methods have been investigated in the field of *Reinforcement Learning* (RL), where the environment is modeled as a Markov Decision Problem (MDP). RL methods have been applied to various domains, including robotics [12], and AI for computer games such as Tetris [5] and fighting games [4].

However, one of the main limiting factors for RL methods is that lots of samples are required for finding good decision rules. One of the approaches for mitigating this problem is to reuse data or previously found solutions from similar RL tasks (see Section 2 for the review). These approaches have been a focus of the research lately and are called *transfer learning* or *multi-task learning* methods for RL.

Current approaches to multi-task learning in RL focus on transferring solutions between several MDPs. However, such approaches do not scale to high-dimensional state spaces due to the *curse of dimensionality* since current approaches share knowledge between states of different RL tasks.

To tackle the high-dimensional RL tasks, we propose a novel usage of multi-task learning in RL: the transition models are shared between the *state factors* of a single MDP. An example of such sharing is a lake crossing problem where the agent has to navigate a boat through a lake full of floating objects, each object is represented as a state factor. The idea is to identify which state factors (objects) are similar to each other based on observed data and then share data between similar factors (objects). This allows the agent to transfer the knowledge learned from the interaction with one object to other similar objects. To accomplish that we propose a transfer learning method that uses a mixture-based approach (our implementation of the method is freely available[1]).

The contributions of our paper are the introduction of multi-task learning for state factors and a method for solving this setting. To our knowledge our paper is the first to introduce such an explicit multi-task learning between different state factors.

The rest of the paper is organized as follows. In the next section we review related research. In Section 3 we formally introduce the setting of RL with factored states. Our proposed multi-task learning ideas are introduced and formalized in Section 4. The performance of the proposed method is evaluated experimentally in Section 5. We discuss extensions and practical applications in Section 6 and finally conclude in Section 7.

2 Related Research

Transfer learning methods for RL can be categorized based on several different aspects [19]. One of the most important differentiating aspects is what is being shared between RL tasks, with most of the existing approaches transferring value functions, transition dynamics, instances or policies.

In the case of value function transfer, the value functions of previously solved RL tasks are transferred to the new task. A popular approach for transferring value functions is to use the previously found value functions as the initial solutions for the value function of the new RL task. These methods are called *starting-point methods* [17, 20]. A special case of such value-based transfer has been considered in [13], where it was assumed that all MDPs have exactly the same states, actions and state transition dynamics. This approach has been extended by [18] by using hypothesis testing to choose the most appropriate RL task and incrementally improve the chosen task. A hierarchical Bayesian approach was

[1]Please see `http://code.google.com/p/mt-factors/`.

proposed by [9], where value functions are shared by using Dirichlet Processes. This is a step forward from the starting-point methods, as the sharing is performed in a multi-task way, i.e., several tasks are learned simultaneously.

A similar approach to value function transfer is policy transfer, i.e., reusing the learned policies from previous tasks. [3] proposed a policy transfer method that biases a current policy in a new task towards the most similar previously learned policy. The idea is similar to starting-point methods, but is based on policy sharing and the weights of sharing are continuously updated based on new samples.

In the methods based on transition dynamics transfer, the transition and reward models are transferred from the solved RL tasks to new tasks. A model-based transfer method was proposed by [22] that uses hierarchical Bayesian clustering to estimate the prior probabilities of transition models of MDPs. If the model of the new MDP is similar to a previously encountered MDP, the data from the previous tasks are used to estimate the transition and reward models of the new task. Thus, the new task can be learned with fewer samples.

An approach similar to transition dynamics transfer is instance transfer. Instead of transferring the learned dynamics, [10] proposed to transfer samples from similar tasks to the new task. To avoid negative transfer, only the samples from the most similar tasks are transferred.

A field related to our work is reinforcement learning in factored-state MDPs where special structure at the state level is used to acquire better policies with fewer samples. Linear decomposition of the Q-function based on the state factor dependencies was proposed by [6] that allows more accurate learning of the Q-function. Instead of Q-function learning a popular approach is a model-based RL in factored-state MDPs, for example see [14]. We follow the latter approach, which will be described in detail in the next section. Our contribution to the previous factored-state RL works is that we consider a more general factored-state dependency that allows us to model more fine-grained dependencies between the factors (see Section 3.2). Additionally, as already mentioned, our work is the first we know of to consider sharing ideas in the context of state factors of an MDP.

3 Reinforcement Learning

The goal of RL is to learn optimal actions in an unknown and stochastic environment.

3.1 RL with Factored States

The environment is specified as an MDP, which is a state-space-based planning problem defined by $\mathcal{S}$, P_I, $\mathcal{A}$, P_T, R and γ. Here $\mathcal{S}$ denotes the set of states, $P_I(s)$ defines the initial state probability, $\mathcal{A}$ is the set of actions, and $0 \leq \gamma < 1$ is the discount factor. At each step the agent receives rewards defined by the function $R(s, a, s') \in \mathbb{R}$, where s, a, s' are, respectively, the current state, the action and the next state. The state transition function $P_T(s'|s, a)$ defines the conditional probability of the next state s' given the current state

s and action a. The state is assumed to satisfy the Markov property, i.e., the probability of next state s' only depends on s and a, and not the previous states or actions. The state is assumed to be factored into elements $s = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_F)$ where $\mathbf{x}_0$ is the factor for our agent and F is the number of non-agent factors (e.g., objects in the environment). Additionally, the state transition function is assumed to have a factorized form

$$P_T(s'|s,a) = \prod_{f=0}^{F} P_T(\mathbf{x}'_f|s,a).$$

The goal of RL here is to find a policy $\pi : \mathcal{S} \to P(\mathcal{A})$ that maximizes the expected discounted sum of future rewards, under the setup that the transition probability P_T is unknown. The discounted sum of future rewards is $\sum_{t=0}^{\infty} \gamma^t r_t$, where r_t is the reward received at step t. In this paper we assume the reward function R is known to the agent, but our multi-task sharing ideas can be applied also for learning the reward function.

In our paper we focus on model-based RL for finding the optimal policy. The model-based RL approach [16] consists of the following two steps:

1. Estimate the transition probabilities $P_T(s'|s,a)$ using transition data.
2. Use the estimated transition probabilities to plan actions, for example, by simulating dynamics few steps ahead and choosing the action giving the largest value. Another option is to find an optimal policy for the estimated transition model (e.g., by using a *dynamic programming*).

More specifically, the transition data consists of, possibly non-episodic[2], samples $\{(s_t, a_t, s'_t)\}_{t=1}^{T}$, where s_t and a_t correspond to the state and action of the t-th transition and s'_t is the the next state.

The performance of model-based RL is guaranteed by the Simulation lemma [7], which states that as the estimated transition probabilities get closer to the true probabilities, the value function of the estimated MDP is getting closer to that of the true MDP. This lemma also applies to the model-based transfer learning methods, i.e., if a model-based transfer learning method can learn a more accurate transition model it is expected to also obtain a more accurate value function and consequently, a better policy.

We focus on the case where the state transitions probabilities P_T are defined by a parametric model[3], i.e.,

$$P_T(s'|s,a,\beta) = \prod_{f=0}^{F} P_T(\mathbf{x}'_f|s,a,\beta_f),$$

where $\beta = (\beta_0, \beta_1, \ldots, \beta_F)$ are the unknown parameters for each factor. The learning of the transition probabilities is, thus, simplified to the learning of these parameters.

[2]Non-episodic means that there is no requirement that the next state of the t-th transition sample (i.e., s'_t) has to be equal to the starting state of the $(t+1)$-th transition sample (i.e., s_{t+1}).

[3]It should be noted that non-parametric extension is possible using the standard kernel trick.

3.2 Restricted Transition Model

An important and very useful aspect of factored state models is the possibility to restrict the interactions between different state factors. A standard approach is to use global restrictions, by employing dynamic Bayesian networks [8]. For example, a global restrictions allow us to specify that the next state of factor $\mathbf{x}_0$ only depends on the current value of factor $\mathbf{x}_2$ and does not depend on factor $\mathbf{x}_1$.

In this paper we adopt state-level restriction for the factors, which is slightly more flexible than the global restriction. Namely, at every state s the next state of a factor $\mathbf{x}_f$ is only affected by a subset of factors denoted by $g_f(s) \subseteq \{0, \ldots, F\}$. Thus, in contrast to the global model, the dependencies between factors can vary according to state s. We assume that the functions $g_f(s)$ for all $f \in \{0, \ldots, F\}$ are known to the agent (as explained below, this assumption is realistic). Note that global restrictions are a special case of state-level restrictions and the reason why we use state-level restrictions is to make the factored state model (and our proposed transfer learning idea) applicable to more general RL problems.

To illustrate the state-level restriction, consider the lake crossing problem where the agent controls a boat that has to get from one shore to another while avoiding collision with floating objects in the lake. In this example, the next value of the agent factor (boat), $\mathbf{x}'_0$, is affected by its own current value ($\mathbf{x}_0$) and only by the states of few floating objects close to it. Floating objects that are currently far away are not affecting the boat. Such dependency can be captured by a dependency function $g_0(s)$ that acts like a nearest neighbor function by only selecting the closest factor within a certain radius to the boat factor $\mathbf{x}_0$. Notice that such a dependency cannot be represented by using global restrictions, because all floating objects can, at certain states, interact with the boat factor (and also with other factors). In that case the boat factor would need to depend on all factors, losing one of the main advantages of the factored state model.

To formally introduce the state-level restrictions, let $\mathbf{x}_\mathbf{h} = (\mathbf{x}_{h_1}, \ldots, \mathbf{x}_{h_H})$ be the state factors chosen by an H-dimensional vector $\mathbf{h}$ with $h_i \in \{0, \ldots, F\}$. The restricted transition model for $\mathbf{x}_f$ can be expressed, for all $f \in \{0, \ldots, F\}$, as

$$P_T(\mathbf{x}'_f | s, a, \beta_f) = P_T(\mathbf{x}'_f | a, \mathbf{x}_\mathbf{h}, \beta_{f,\mathbf{h}}), \tag{1}$$

where $\mathbf{h} = g_f(s)$. The parameter $\beta_{f,\mathbf{h}}$ specifies the model for the states where the factor $\mathbf{x}_f$ is affected by factors $\mathbf{x}_\mathbf{h}$. For example, the parameter $\beta_{0,(0,3,4)}$ specifies the model for the boat factor $\mathbf{x}_0$ in the states where the next state of the boat is affected by itself and objects $\mathbf{x}_3$ and $\mathbf{x}_4$, i.e., in states s where $g_0(s) = (0, 3, 4)$.

An example of parametric form of (1), which can be used as a transition model for the lake crossing problem, is a multivariate Gaussian distribution

$$P_T(\mathbf{x}'_f | a, \mathbf{x}_\mathbf{h}, \beta_{f,\mathbf{h}}) = \mathcal{N}(\mathbf{x}_f + \mathbf{w}_{f,\mathbf{h}}^\top \phi(\mathbf{x}_\mathbf{h}, a), \mathbf{\Sigma}_{f,\mathbf{h}}), \tag{2}$$

where $\mathcal{N}(a, b)$ denotes the multivariate Gaussian distribution with mean a and covariance b. Parameters in the transition model are $\beta_{f,\mathbf{h}} = (\mathbf{w}_{f,\mathbf{h}}, \mathbf{\Sigma}_{f,\mathbf{h}})$ where $\mathbf{w}_{f,\mathbf{h}}$ and $\mathbf{\Sigma}_{f,\mathbf{h}}$ affect the mean and the covariance of the Gaussian distribution, respectively; $\phi(\mathbf{x}_\mathbf{h}, a)$ is a

feature vector. By adding $\mathbf{x}_f$ to the mean of the Gaussian distribution, $\mathbf{w}_{f,\mathbf{h}}$ estimates the mean change in the state factor $\mathbf{x}_f$. Modeling the change is useful for continuous dynamics domains such as the above lake crossing task. If $\phi(\mathbf{x_h}, a) \in \mathbb{R}^B$ and the dimension of factor $\mathbf{x}_f$ is C, then the parameters are $\mathbf{w}_{f,\mathbf{h}} \in \mathbb{R}^{B\times C}$ and $\mathbf{\Sigma}_{f,\mathbf{h}} \in \mathbb{R}^{C\times C}$.

3.3 Estimating the Parameters of the Transition Model

The parameters β have to be learned from data. A natural approach is to maximize the logarithm of a posteriori of the observed transition data (MAP). Let $S = \{s_n\}_{n=1}^N$, $A = \{a_n\}_{n=1}^N$ and $S' = \{s'_n\}_{n=1}^N$ be the observed states, actions and next states, respectively, where N is the number of observed transitions. Then the logarithm of a posteriori is given by

$$\begin{aligned} L(\beta) &= \log P(S'|S, A, \beta)P(\beta) \\ &= \sum_{n=1}^{N} \log P(s'_n|s_n, a_n, \beta) + \log P(\beta), \\ &= \sum_{n=1}^{N}\sum_{f=0}^{F} \log P(\mathbf{x}'_{n,f}|s_n, a_n, \beta_f) + \log P(\beta), \end{aligned} \tag{3}$$

where $P(\beta)$ is the prior of the transition model and double indexed $\mathbf{x}$ denotes a particular factor of that state, i.e., $s_n = (x_{n,0}, \ldots, x_{n,F})$ and $s'_n = (\mathbf{x}'_{n,0}, \ldots, \mathbf{x}'_{n,F})$. For multinomial and Gaussian models (as the one specified in (2)), the maximization of $L(\beta)$ is a convex optimization problem and thus the solution can be effectively computed by standard maximum likelihood approaches [1].

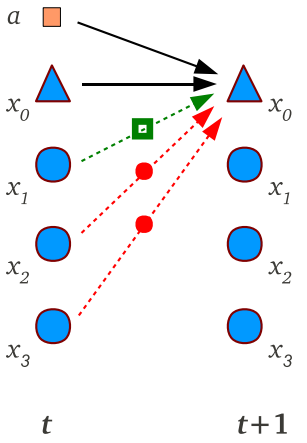
However, learning such models in complicated domains still demands large amounts of data to get an accurate estimate, because the number of parameters (β) needed to be estimated is very large. We now introduce the multi-task sharing method for the state factors, which tries to improve the accuracy of the estimation of β.

4 Multi-Task Learning for State Factors

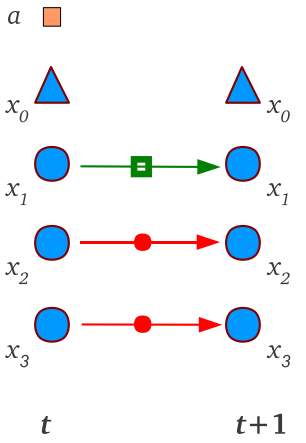
In this section, we introduce our multi-task sharing method for the state factors.

4.1 Basic Idea

Before introducing the general idea, let us again come back to the lake crossing example. Suppose there are two types of floating objects in the lake: seaweed and plastic bottles. In such case it would be beneficial to share data between the floating objects of the same type. Firstly, the same type of floating objects should have similar effect to the boat when they come into contact with it. Secondly, the same type of floating objects should also move in a similar way in the lake.



(a) Sharing idea 1: factors $\mathbf{x}_2$ and $\mathbf{x}_3$ have similar effect on agent factor ($\mathbf{x}_0$).



(b) Sharing idea 2: factors $\mathbf{x}_2$ and $\mathbf{x}_3$ have similar transition dynamics.

Figure 1: Multi-task learning ideas for state factors, showing links from the time step t to the next time step $t+1$.

Our multi-task learning approach tries to achieve these two types of sharing, which can be expressed in a general way as follows:

- **Sharing idea 1**: If two factors $\mathbf{x}_f$ and $\mathbf{x}'_f$ are similar, then their effect to the agent factor and other factors is similar, as illustrated in Figure 1(a).
- **Sharing idea 2**: If two factors $\mathbf{x}_f$ and $\mathbf{x}'_f$ are similar, then their transition models are similar, i.e., β_f and $\beta_{f'}$ are similar, as illustrated in Figure 1(b).

Note that the agent does not know in advance which factors are similar and this grouping has to be also learned from the data. If the grouping is incorrect, wrong transfer can occur between the factors, increasing the model error. To avoid such incorrect transfer we will next introduce a mixture model that allows us to learn the grouping that best explains the data.

4.2 Mixture Model for State Factors

For accomplishing the two multi-task learning goals mentioned above we propose to use a mixture model for the non-agent factors, i.e., $\mathbf{x}_1, \ldots, \mathbf{x}_F$. Every non-agent factor $\mathbf{x}_f$ is assumed to belong to one mixture component, denoted by $c_f \in \{1, \ldots, C\}$. Thus, each component represents similar types of objects (for example, in the case of the lake crossing task there are two components, one for seaweed and the other for plastic bottles).

The main idea is that the factors in the same component have the same transition dynamics, which we can express by replacing Eq.(1) by

$$P_T(\mathbf{x}'_f | s, a, \beta^{MT}, \mathbf{c}) = P_T(\mathbf{x}'_f | a, \mathbf{x_h}, \beta^{MT}_{c_f, \mathbf{c_h}})$$

where $\mathbf{h} = g_f(s)$, $\mathbf{c} = \{c_0, c_1, \ldots, c_F\}$, and $\mathbf{c_h} = (c_{h_1}, \ldots, c_{h_{\dim(h)}})$ denotes the components of factors $\mathbf{x_h}$.[4] The crucial difference to (1) is that we now have introduced multi-task transition model β^{MT} that uses the same transition model for factors belonging to the same component. This means that $\beta^{MT}_{c,\mathbf{k}}$ specifies the transition model for all factors belonging to the mixture component c when they are affected by factors belonging to components specified in vector $\mathbf{k}$.

To illustrate the idea in the lake crossing problem, assume objects $\mathbf{x}_1$ and $\mathbf{x}_2$ (e.g., floating seaweed) belong to component 1 and factors $\mathbf{x}_3$ and $\mathbf{x}_4$ (e.g., plastic bottles) belong to component 2. Then the dynamics of object $\mathbf{x}_1$ when affected only by object $\mathbf{x}_3$ is the same as the dynamics of the object $\mathbf{x}_2$ when affected by $\mathbf{x}_4$, both dynamics specified by the parameter $\beta^{MT}_{1,(1,2)}$.

With respect to the difficulty of the learning task, the number of parameters in β^{MT} is greatly reduced compared to β. For example, consider models where factors are affected at most by only one other factor. Then the number of parameters in β is $O(F^2)$, whereas in β^{MT} there are $O(C^2)$ parameters. Multi-task learning can be very useful if C is significantly smaller than F.

Analogously to Eq.(3) we now define a likelihood for the multi-task learning model

$$\begin{aligned} L(\beta^{MT}) &= \log P(S'|S, A, \beta^{MT}) P(\beta^{MT}) \\ &= \log \sum_{\mathbf{c}} P(\mathbf{c}, S'|S, A, \beta^{MT}) + \log P(\beta^{MT}). \end{aligned}$$

Next we will discuss how to maximize the likelihood by using EM method.

4.3 EM Method for the Mixture of State Factors

As $\mathbf{c}$ are unknown a standard approach to finding the maximum likelihood solutions to the mixture models is to treat $\mathbf{c}$ as latent variables and employ the Expectation-Maximization (EM) algorithm [1]. In our case, the main steps iterated by the EM algorithm are to evaluate $P(\mathbf{c}|S, S', A, \beta^{MT^{old}})$ for each $\mathbf{c}$ and then to find a new β^{MT} that maximizes

$$\sum_{\mathbf{c}} P(\mathbf{c}|S, S', A, \beta^{MT^{old}}) \log P(\mathbf{c}, S'|S, A, \beta^{MT}).$$

The number of possible values for $\mathbf{c}$ is exponential to the number of factors. However, full enumeration is usually not necessary (e.g., in the case of a mixture of Gaussians [1]), because the latent variables (c_f) are conditionally independent. Unfortunately, in the mixture of state factors this is not the case, i.e.,

$$P(\mathbf{c}|S, S', A, \beta^{MT}) \neq \prod_{f=1}^{F} P(c_f|S, S', A, \beta^{MT}),$$

[4] To simplify the notation we have added a pseudo component for the agent factor $\mathbf{x}_0$, fixed as $c_0 = 0$.

because a single transition can be affected by several c_f. A simple example of this is when a factor f is affected by factor f', then the multi-task transition probability is $P(\mathbf{x}'_f|\mathbf{x}_{(f,f')}, a, \beta^{MT}_{c_f,(c_f,c_{f'})})$, depending on both c_f and $c_{f'}$.

Therefore, we employ an extended EM approach where evaluating $P(\mathbf{c}|S, S', A, \beta^{MT^{old}})$ is replaced by sampling $\mathbf{c}$ from it. This is possible by using Gibbs sampling, as we will detail below. The maximization step is then carried out over the sampled $\mathbf{c}$. The drawback of this approach is that the sampling is computationally expensive. To overcome this problem we will exploit the fact that there are training samples which only depend on a single latent variable c_f. The idea is to first use only those samples, allowing us to carry out EM without sampling and then switch to a heavier sampling-based approach. We describe this in Section 4.4.

We now outline the steps of the proposed EM method and leave the details of the derivation to the A:

1. Initialize transition models β^{MT}. One option is to initialize randomly. A more efficient way will be discussed in Section 4.4.

2. E-step: Draw K samples of $\mathbf{c}$ from distribution

$$P(\mathbf{c}|S', S, A, \beta^{MT})$$

   using Gibbs sampling, i.e., drawing sequentially each c_f from $P(c_f|\mathbf{c}_{-f}, S', S, A, \beta^{MT})$ where $\mathbf{c}_{-f}$ is $\mathbf{c}$ without c_f. We will denote the K sampled values by $\{\boldsymbol{\zeta}_k\}_{k=1}^K$.

3. M-step: Find the parameter β^{MT} that maximizes

$$\frac{1}{K}\sum_{k=1}^{K} \log P_T(S'|S, A, \beta^{MT}, \mathbf{c} = \boldsymbol{\zeta}_k) + \log P(\beta^{MT}).$$

4. If the log likelihood or parameters have converged, stop; otherwise, go to step 2.

The Algorithm 1 gives a general pseudo code description of the proposed method. The derivation of the algorithm for Gaussian transition dynamics is discussed in detail in B. The inputs to the algorithm are the transition data (S, S', A) and parameters for the number of components (C) and regularizer for the Gaussian model (λ). In the experiments we will choose the values of the two parameters using holdout validation, i.e., we train the model using two thirds of the data and using the rest to validate. Choices for parameters are $C \in \{2, 4, 6\}$ and $\lambda \in \{10^0, 10^{-1}, 10^{-2}\}$. Note that the number of possible $\mathbf{c}''$ in the M-step of Algorithm 1 grows exponentially with respect to its length. In experiments we limit the length of $\mathbf{c}''$ to three. This allows us to model interactions with up to two affecting state factors (the first element of $\mathbf{c}''$ is always the component of the state factor that is being affected).

The goal of the agent in the model-based learning is to be able to generate samples of state transitions from the current state and possible actions. For that we propose a

Algorithm 1 Algorithm for finding the solution for the mixture models.

1: **procedure** LEARNFACTORMT(S, S', A, C, λ)
2: $i = 0$
3: Initialize β^{MT^i}
4: **repeat**
5: $i \leftarrow i + 1$
6: $\boldsymbol{\zeta}^i \leftarrow$ ESTEP(S,S',A, C, $\beta^{MT^{i-1}}$)
7: $\beta^{MT^i} \leftarrow$ MSTEP(S,S',A, C, $\boldsymbol{\zeta}^i$, λ)
8: **until** $\|\beta^{MT^i} - \beta^{MT^{i-1}}\|$ becomes small
9: **end procedure**
10:
11: **procedure** ESTEP(S, S', A, C, K, β^{MT})
12: Initialize $\mathbf{c}$ randomly from 1 to C
13: **for** k in 1..K **do**
14: For each f in turn sample c_f from
15: $c_f \sim P(c_f|\mathbf{c}_{-f}, S', S, A, \beta^{MT})$
16: $\boldsymbol{\zeta}_{k,f} \leftarrow c_f$
17: **end for**
18: **return** $\boldsymbol{\zeta}$
19: **end procedure**
20:
21: **procedure** MSTEP(S, S', A, C, $\boldsymbol{\zeta}$, λ)
22: For each $c', \mathbf{c}''$:
23: Loop through $\boldsymbol{\zeta}$ and (S, S', A)
24: Collect all $(\mathbf{x}_{n,\mathbf{h}}, \mathbf{x}'_{n,f}, a_n, \boldsymbol{\zeta}_k)$ such that:
25: $\boldsymbol{\zeta}_{k,f} = c'$ and $\boldsymbol{\zeta}_{k,\mathbf{h}} = \mathbf{c}''$ with $g_f(s) = \mathbf{h}$.
26: Using collected data find $\beta^{MT}_{c',\mathbf{c}''}$ maximizing
27: $\sum_{data} \log P_T(\mathbf{x}'_{n,f}|a_n, \mathbf{x}_{n,\mathbf{h}}, \beta^{MT}_{c',\mathbf{c}''}) + \log P(\beta^{MT}; \lambda)$
28: End for
29: **return** β^{MT}
30: **end procedure**

following approach using the estimated transition model β^{MT} and component assignments ξ. Firstly, the agent samples k uniformly from $\{1, \ldots, K\}$, specifying the component assignment ξ_k for factors. Then, the transition is generated according to $P_T(\mathbf{x}'_f|\mathbf{s}, a, \mathbf{c} = \xi_k)$, where we assume that we can generate samples from transitions model. For example, this can be easily done for Gaussian and multinomial dynamics. See section 5.2 for the details of model-based policy used in the experiments.

Algorithm 1 does not specify how to initialize β^{MT} (line 3 in the algorithm). A straightforward option is to initialize β^{MT} randomly, but that often results in lots of iterations before convergence. Below we discuss a better initialization option that often speeds up the convergence.

4.4 Speeding Up EM Method

The proposed method above can be slow due to sampling in E-step. Sampling is required in E-step because c_f for different factors $\mathbf{x}_f$ are not independent given the data. Here, our idea is to initialize β^{MT} by only using a subset of data where c_f are conditionally independent. If c_f are conditionally independent the E-step can be carried out analytically.

This initialization is expected to converge close to the main method solution if there is low dependency between c_f; thus, only requiring few iterations of slow sampling-based E-steps.

This analytically approximated E-step is formally expressed for c_f by

$$\begin{aligned} & P(c_f = c' | S', S, A, \beta^{MT}) \\ & = \frac{P(S' | c_f = c', S, A, \beta^{MT})}{\sum_{c''=1}^{M} P(S' | c_f = c'', S, A, \beta^{MT})} \\ & \approx \frac{\prod_{(n,h) \in I(f)} P(\mathbf{x}'_{n,h} | c_f = c', \mathbf{s}_n, a_n, \beta^{MT})}{\sum_{c''=1}^{M} \prod_{(n,h) \in I(f)} P(\mathbf{x}'_{n,h} | c_f = c'', \mathbf{s}_n, a_n, \beta^{MT})}, \end{aligned} \tag{4}$$

where $I(f)$ contains pairs of indices of a transition sample and a factor which do not depend on any latent variables except c_f and c_0 (which is just a constant).

Note that in the M-step we will also only use the subset of the data specified by $I(\cdot)$, resulting in an EM procedure that is just based on the subset of data. See C for the details of the M-step.

5 Experimental Results

We test our proposed method in a lake crossing task where the agent has to control a boat from one shore to another and has to avoid hitting the shores, which terminates its run. The difficulty lies in the fact that the lake contains floating objects that interact with the boat, e.g., light garbage and floating lake weed. Although it is a toy problem it has several challenging aspects that are present in real-world problems. Firstly, the state space is very large, with 41 objects and 5 features for each object; the state has more than 200 dimensions. Thus, applying standard value function estimation methods such as *fitted Q-learning* [2] is not feasible. Secondly, as the objects are interacting when they get close (i.e., $g_f(s)$ is a nearest neighbor function with a fixed threshold), the agent has to plan its route carefully in the lake to avoid or control collisions, requiring accurate estimation of the dynamics of the objects.

5.1 Setup

We now outline the details of the lake crossing tasks. The goal of the agent is to guide the boat from the start state to the goal state, see Figure 2. The environment is simulated using continuous state dynamics, which we adopted from [10] where the environment

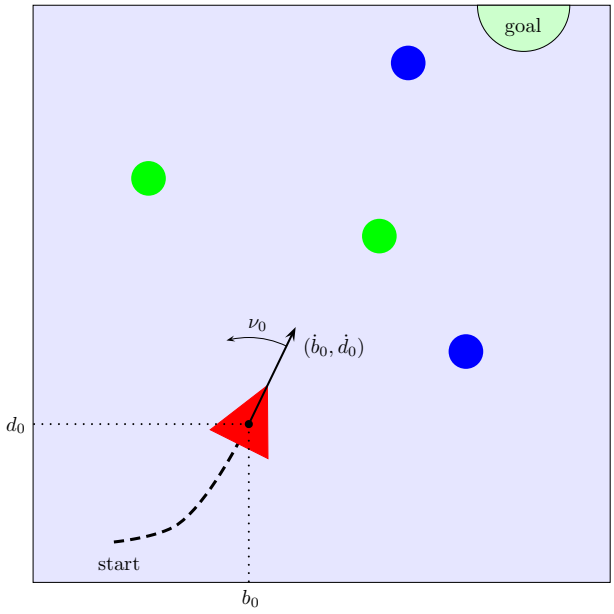


Figure 2: Example of a lake crossing problem with a boat factor ($\mathbf{x}_0 = (b_0, d_0, \dot{b}_0, \dot{d}_0, \nu_0)$) and 4 floating objects of two types (the actual setup contained 40 objects of 4 types).

consisted only of one 5-dimensional factor, the boat. In our setup we adopt the dynamics of the boat to all objects, i.e., factor $\mathbf{x}_f \in \mathbb{R}^5$:

$$\mathbf{x}_f = (b_f, d_f, \dot{b}_f, \dot{d}_f, \nu_f),$$

where $b_f, d_f \in [0, 200]$ are the location coordinates on the lake, $\dot{b}_f, \dot{d}_f \in [-5, 5]$ are the speeds of the object along the respective coordinates, and $\nu_f \in [-\pi/6, \pi/6]$ is the rotation speed (angles per time step). If the distance between two objects in the lake is less than 5 they are interacting, i.e., if

$$\|(b_f, d_f) - (b_{f'}, d_{f'})\|_2 < 5, \tag{5}$$

then $\mathbf{x}_f$ is affected by $\mathbf{x}_{f'}$ or formally $f' \in g_f(s)$. At the state s the dynamics for non-agent factor f is

$$\mathbf{x}'_f = \begin{bmatrix} b_f + \dot{b}_f \\ d_f + \dot{d}_f \\ \sum_{f'=0}^{F} \omega_{f'} (\dot{b}_{f'} \cos \nu_{f'} - \dot{d}_{f'} \sin \nu_{f'}) \\ \sum_{f'=0}^{F} \omega_{f'} (\dot{b}_{f'} \sin \nu_{f'} + \dot{d}_{f'} \cos \nu_{f'}) \\ \nu_f \end{bmatrix} + \boldsymbol{\epsilon}_f,$$

where $\boldsymbol{\epsilon}_f$ is 5-dimensional Gaussian noise with zero mean and the variable $\omega_{f'}$ shows the strength of the effect from object $\mathbf{x}_{f'}$ to object $\mathbf{x}_f$. The sum of weights $\omega_{f'}$ is equal to 1. For objects $\mathbf{x}_{f'}$ that do not satisfy (5) the weight is 0 (thus, they have no effect on $\mathbf{x}_f$)

Table 1: Values used in the lake crossing experiments.

Variable	Value
Discount factor (γ)	0.98
Goal location	$(150, 200)$
Goal radius	15
Initial state of boat ($\mathbf{x}_0$)	$(20, 5, 0, 0.5, 0)$
Inertia of boat (I_0)	0.05
Inertia of type 1 objects	0.01
Inertia of type 2 objects	0.04
Inertia of type 3 objects	0.3
Inertia of type 4 objects	0.6
Goal reward	+10
Reaching the shore reward	−1

and for others it depends on their inertia $I_{f'}$ and their strength of interaction $\rho_{f,f'} \in [0, 1]$ (for the factor $\mathbf{x}_f$ we have $\rho_{f,f} = 1$):

$$\omega_{f'} = \begin{cases} 0 & \text{if (5) is false,} \\ \rho_{f,f'} I_{f'}/C & \text{otherwise,} \end{cases}$$

where I_f is the inertia of object $\mathbf{x}_f$ (proportional to its mass) and C is a normalization constant for making $\sum_{f'=0}^{F} \omega_{f'} = 1$.

For the agent factor $\mathbf{x}'_0$ we add an additional action-dependent effect

$$\left[0, 0, \frac{\dot{b}_0}{v_0}(a_1 - v_0)I_0, \frac{\dot{d}_0}{v_0}(a_1 - v_0)I_0, (a_2 - \nu_0)\frac{v_0}{v_{max}}\right]^\top,$$

where $v_0 = \|(\dot{b}_0, \dot{d}_0)\|_2$ is the speed of the agent factor $\mathbf{x}_0$, v_{max} is the maximum speed (equal to 5), $a_1 \in [0, v_{max}]$ and $a_2 \in [-\pi/6, \pi/6]$ are actions by the agent, respectively, controlling the movement and rotation speeds. At each step the agent chooses (a_1, a_2) to be one of the following 5 actions: $(0, 0)$, $(2, 0)$, $(5, 0)$, $(1.0, -\pi/6)$, $(1.0, -\pi/6)$.

There are 4 types of floating objects differing by their inertia (I_f), specified in Table 1 together with other values used in the experiments. There are in total 10 objects of each type and the initial state of the objects is chosen randomly.

5.2 Model-Based Policy

As model-free approaches cannot be used for the lake crossing problem, due to the high-dimensionality of the state space, we compare our proposed method (MT-factors) to the case where no multi-task learning is used, i.e., a transition model is learned for each object separately. For both approaches we use the following look-ahead strategy:

1. Estimate transition model P_T.

2. Estimate the Q-function from data where the boat is not affected by other objects, i.e., the case of an empty lake. In such a case the learning of the Q-function is feasible, since the problem has only 5-dimensional state ($\mathbf{x}_0$) and 5 possible actions. For learning the Q-function we employ Fitted Q-learning [2] with the Extra Trees method[5].

3. To find the best action at state s, we look ahead 5 steps by generating the next states according to the estimated P_T and then record its value using the Q-function found in step 2. We repeat this 4 times and return the action that has the highest average value.

This approach uses the Q-function from no interaction model as a general guide. By looking ahead the agent can plan the possible interactions with the floating objects and should be able to achieve good performance if its learned model of transition dynamics is accurate.

As the underlying model for the transition dynamics we use the parametric Gaussian model given in Eq.(2) with a quadratic function for the features, which can be expressed as $\phi(\mathbf{x_h}, a) = \mathbf{z} \otimes \mathbf{z}$ with $\mathbf{z} = (1, \mathbf{x_h}^\top, a)^\top$ and $\otimes$ is the tensor product (we removed the double entries that are from the symmetry).

To apply the MT-factors in the case of Gaussian dynamics the maximization (line 27 in Algorithm 1) in the M-step becomes ridge regression and in the E-step we have to use (2) to calculate the transition probabilities. The details are given in B.

5.3 Compared Methods

We compare the performance of MT-factors to other approaches. Firstly, we adopt the standard model-based approach for factored-state environments to our setup, which is essentially equivalent to learning the transition dynamics of each factor separately. We call it ST-factors, i.e., single-task learning for factored-state RL. Secondly, we compare against Fitted-Q that uses just the boat factor without interaction data (Plain Fitted-Q). Applying Fitted-Q to learn the whole state-space of more than 200 dimensions would require intractable amounts of training data, because the full Q-function would require representing very complex interaction effects between the boat factor and each floating object.

In general, more suitable approach for learning the Q-function in factored-state environments is to exploit the factor dependencies and separate the Q-function into linear components as proposed by [6]. However, in our setting this approach would not be successful because dependencies between factors are not global but based on state-level restrictions, i.e., all factors can be, at some point, dependent on each other. Therefore, efficient linear decomposition of the whole Q-function cannot be made without very serious losses in accuracy, e.g., if we decompose the Q-function into a sum of several Q-functions

[5]Specifically, we generated 500,000 samples from the estimated no interactions model, i.e., the samples where the boat $\mathbf{x}_0$ is affected only by itself ($g_0(s) = (0)$). The Fitted Q-learning was run 40 iteration with 15 trees. The parameter determining the leaf size in Extra Trees was chosen by using holdout validation.

where each Q-function depends only on a single factor it would be impossible to represent any effects of interactions. Thus, we do not compare against this approach.

5.4 Results

To clearly see the effects of the learned model on the performance we used the same randomly collected data as input for each method. The data was sampled uniformly over the state and action space (note that model-based methods can use non-episodic data as input). The found policies were then ran on the environment for 100 episodes to measure the expected value and success rate. Each episode started from the initial state and ended in a terminal state or after 200 steps, whichever came first. This was repeated 10 times for each data size (1000, 2000, 4000, 8000 and 12000 samples) to find mean and standard deviations for all the methods.

Figure 3 shows the performances of our proposed method (MT-factors), ST-factors and Plain Fitted-Q. In the case of sample sizes 2000 and higher MT-factors has (statistically) better performance at 1% statistical significance (using t-test) compared to ST-factors and Plain Fitted-Q.

To see how well the policies are working we also compared the success rate of the agent, which is depicted in Figure 4. The success rate shows the proportion of runs where the agent is able to reach the goal state.

We also computed the performances of random policy (Random) and the policy that uses the true transition model (True-Model), which are not depicted in the figures. The value of the random policy was -0.3 with success rate of 0% and True-Model[6] had value 1.8 with success rate of 95%. This means that having accurate transition model is crucial for good performance. None of the methods could reach success rate of the 95% of the True-Model, which shows that the lake crossing problem is challenging. Nevertheless, our proposed method outperformed the non-sharing approaches.

To investigate how MT-factors is grouping together similar factors we measured the frequency that two factors belong to the same component, which is depicted in Figure 5. Factors belonging to the same type were assigned to the same component with frequency 1.0, meaning that MT-factors was able to detect the similar factors (for reference, factors 1–10 were type 1, 11–20 type 2, 21–30 type 3 and 31–40 type 4). However, factors of type 3 and type 4 factors were almost always assigned to the same component, which is probably caused by the fact that their dynamics are similar due to similar inertia (0.3 and 0.6, respectively). On the other hand, more distinct types, like type 1 and type 4, are never grouped together, showing that MT-factors avoids sharing data between too distinct types.

6 Discussion

In this section, we discuss extensions and practical applications of the proposed method.

[6]True-Model used the Fitted-Q value function that was based on 12000 training samples.

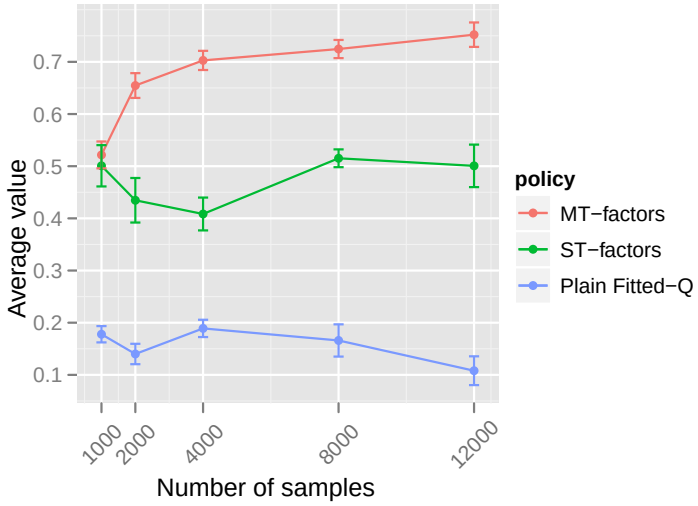


Figure 3: Values of the policies in the lake crossing experiment showing the mean and the standard deviation over 10 trials. In each trial the value (sum of discounted returns) was calculated as the average of 100 runs.

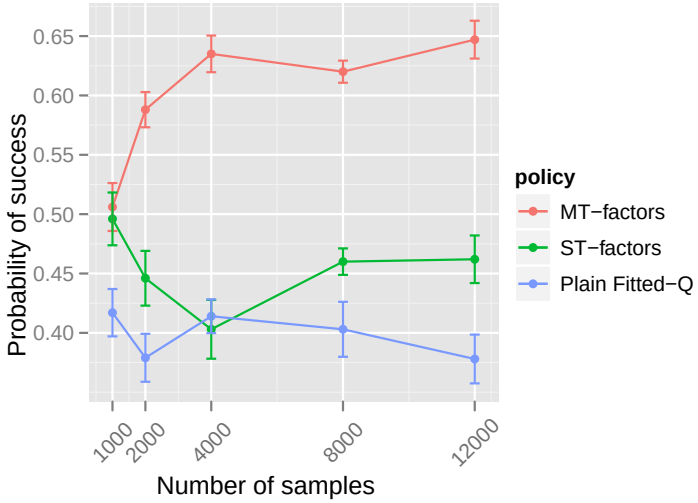


Figure 4: Success rate of the policies in the lake crossing experiment showing the mean and the standard deviation over 10 trials. In each trial the success rate was calculated from 100 runs.

6.1 Applications

MT-factors is suitable for applications where there is interaction between several factors that have hidden characteristics. An example is a dialog agent that can observe people chatting and has a goal to chat with them. In terms of the MT-factors framework a text

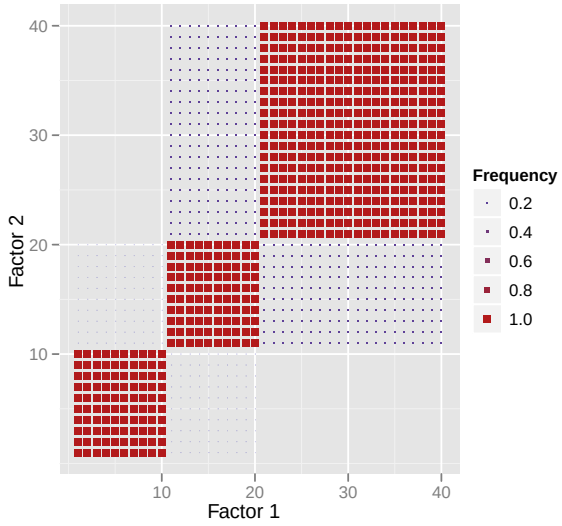


Figure 5: Averaged frequency of two factors belonging to the same component with 8000 training samples, averaged over 100 runs. No dot means that the two factors were never assigned to the same component.

by each chatter f can be considered as factor x_f and we model how two chatters interact by modelling the probabilities of $P(x'_f|x_f, x_{f'}, \beta_{f,f'})$. This model allows each chatter f to have a different style for each other chatter f', which is often the case as people choose styles depending on the characteristics of the other person (like mood, personality). This is the key reason why MT-factors is needed if we want to share between similar chatting styles. In terms of mixture components, the model for the text of chatter f depends, in addition to its own component c_f, to the component of his/her chatting parter $c_{f'}$.

This kind of situation also arises in modelling the strategy of opponents in games where players adjust their policies depending on their opponent type. An example of such games is poker where players can observe an opponent's play. Our proposed MT-factors can be adopted for creating AI players that quickly adapt to the strategy of their opponent by sharing between players that have similar style.

6.2 Exploration and Exploitation

Up to now we have considered the case where the agent has access to random samples from the environment. But often instead it is required to collect useful samples while incurring as minimal loss as possible. This is called the exploration-exploitation dilemma. As MT-factors is a model-based method we can use a exploration bonus approach [21]. The idea is to give an explicit bonus to states where there are agent and factor interactions that have very few samples: the fewer data points we have (implying a less accurate model) the bigger the exploration bonus. In more general terms, the exploration bonus is given for visiting factor interactions whose parameters of the transition model (β^{MT}) are not

yet accurately estimated.

However, we leave the exact details of the exploration method for future research.

6.3 Structural Prior Knowledge

Similarly to other factored-state based approaches MT-factors requires additional prior structural knowledge, i.e., the factorization of the state space and dependency information, which is specified by $g_f(\mathbf{s})$. Thus, currently our proposed method can be applied to tasks where such prior information is readily available.

There have been several proposals for learning the structural knowledge in factored-state RL, for example see [15] and [11]. These approaches remove the need for prior structural knowledge by finding global dependencies between the factors. However, regarding MT-factors the structure learning is still an open problem as the dependencies allowed by $g_f(\mathbf{s})$ are more complex. This is also left as a future work.

7 Conclusions

We proposed multi-task learning for the state factors that is able to learn transition probabilities more accurately by sharing data between similar factors. This was demonstrated in the challenging lake crossing problem where the MT-factors method outperformed standard factored-state RL learning approaches. We proposed applications of MT-factors in real-world domains such as learning dialog agents and adaptable AIs for games.

Acknowledgments

JS was supported by MEXT Scholarship, MS was supported by MEXT KAKENHI 23120004, and HH was supported by the FIRST Program.

References

[1] C.M. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer, 2006.

[2] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," Journal of Machine Learning Research, vol.6, no.1, pp.503–556, 2005.

[3] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, pp.720–727, ACM, 2006.

[4] T. Graepel, R. Herbrich, and J. Gold, "Learning to fight," Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education, 2004.

[5] A. Gro, J. Friedland, and F. Schwenker, "Learning to play tetris applying reinforcement learning methods.," ESANN, pp.131–136, 2008.

[6] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, "Efficient solution algorithms for factored MDPs," Journal of Artificial Intelligence Research (JAIR), vol.19, pp.399–468, 2003.

[7] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," Machine Learning, vol.49, no.2-3, pp.209–232, 2002.

[8] M.J. Kearns and D. Koller, "Efficient reinforcement learning in factored MDPs," IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, San Francisco, CA, USA, pp.740–747, Morgan Kaufmann Publishers Inc., 1999.

[9] A. Lazaric and M. Ghavamzadeh, "Bayesian Multi-Task Reinforcement Learning," ICML '10: Proceedings of the 28nd International Conference on Machine learning, 2010.

[10] A. Lazaric, M. Restelli, and A. Bonarini, "Transfer of samples in batch reinforcement learning," ICML '08: Proceedings of the 25th international conference on Machine learning, New York, NY, USA, pp.544–551, ACM, 2008.

[11] L. Li, M.L. Littman, and T.J. Walsh, "Knows What It Knows: a framework for self-aware learning," ICML 08 Proceedings of the 25th International Conference on Machine Learning, pp.568–575, ACM, 2008.

[12] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS), Karlsruhe, Germany, September 2003.

[13] S.P. Singh, "Transfer of learning by composing solutions of elemental sequential tasks," Machine Learning, vol.8, pp.323–339, 1992.

[14] A.L. Strehl, "Model-based reinforcement learning in factored-state MDPs," Approximate Dynamic Programming and Reinforcement Learning 2007 ADPRL 2007 IEEE International Symposium on, no.Adprl, pp.103–110, 2007.

[15] A.L. Strehl, C. Diuk, and M.L. Littman, "Efficient structure learning in factored-state MDPs," Proceedings Of The National Conference On Artificial Intelligence, pp.645–650, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[16] R.S. Sutton and A.G. Barto, Reinforcement Learning, MIT Press, Cambridge, MA, USA, 1998.

[17] F. Tanaka and M. Yamamura, "Multitask reinforcement learning on the distribution of MDPs," Computational Intelligence in Robotics and Automation, 2003, pp.1108–1113, July 2003.

[18] T. Taniguchi and T. Sawaragi, "Incremental acquisition of behaviors and signs based on a reinforcement learning schemata model and a spike timing-dependent plasticity network," Advanced Robotics, vol.21, no.10, pp.1177–1199, 2007.

[19] M.E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," Journal of Machine Learning Research, vol.10, no.1, pp.1633–1685, 2009.

[20] M.E. Taylor, P. Stone, and Y. Liu, "Value functions for RL-based behavior transfer: A comparative study," Proceedings of the Twentieth National Conference on Artificial Intelligence, pp.880–885, July 2005.

[21] M.A. Wiering and J. Schmidhuber, "Efficient model-based exploration," Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior: From Animals to Animats 6, ed. J.A. Meyer and S.W. Wilson, pp.223–228, MIT Press/Bradford Books, 1998.

[22] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: a hierarchical Bayesian approach," ICML '07: Proceedings of the 24th International Conference on Machine learning, New York, NY, USA, pp.1015–1022, ACM, 2007.

A Details of EM Algorithm

Here, we explain the details of the EM algorithm for the state factors (see Section 4.2).

The EM method constists of following steps:

1. Initialize transition models β^{MT}.
2. E-step: find the expectation of the latent variable $\mathbf{c}$ given the data and current value of β^{MT}, i.e., $P(\mathbf{c}|S', S, A, \beta^{MT})$.
3. M-step: find β^{MT} maximizing the expected log-likelihood

$$\max_{\beta^{MT}} \mathbb{E}_{\mathbf{c}} \log P(\mathbf{c}, S'|S, A, \beta^{MT}) P(\beta^{MT}), \tag{6}$$

   where $\mathbb{E}_{\mathbf{c}}$ denotes the expectation over $\mathbf{c}$ found in E-step, i.e., $\mathbf{c} \sim P(\mathbf{c}|S', S, A, \beta^{MT})$.
4. If the log-likelihood or parameters have converged, stop, otherwise, go to step 2.

A.1 E-Step

Usually in the E-step of the mixture models it is possible to analytically evaluate $P(\mathbf{c}|S', S, A, \beta^{MT})$ by using the fact that latent variables are conditionally independent. However, in the case of the mixture of state factors the variables $\mathbf{c}_f$ and $\mathbf{c}_{f'}$ are not conditionally independent, because the model allows interaction between several non-agent factors. In such situation a standard approach is to use sampling of $\mathbf{c}$ from $P(\mathbf{c}|S', S, A, \beta^{MT})$ in E-step.

In our case a straightforward option for sampling is to use Gibbs sampling, i.e., sampling each c_f in turn from the conditional distribution $P(c_f|\mathbf{c}_{-f}, S', S, A, \beta^{MT})$, where $\mathbf{c}_{-f}$ is $\mathbf{c}$ without c_f. The conditional distribution is

$$\begin{aligned} &P(c_f = c'|\mathbf{c}_{-f}, S', S, A, \beta^{MT}) \\ &= \frac{P(S'|c_f = c', \mathbf{c}_{-f}, S, A, \beta^{MT})}{\sum_{c''=1}^{C} P(S'|c_f = c'', \mathbf{c}_{-f}, S, A, \beta^{MT})}, \end{aligned} \tag{7}$$

where

$$\begin{aligned} &P(S'|c_f = c', \mathbf{c}_{-f}, S, A, \beta^{MT}) \\ &= \prod_{n=1}^{N} \prod_{f=0}^{F} P_T(\mathbf{x}'_{n,f}|s_n, a_n, c_f = c', \mathbf{c}_{-f}, \beta^{MT}). \end{aligned}$$

This product can be easily calculated because the elements in the product are just transition probabilities when transition models are β^{MT} and the component assignments of the factors are given by c_f and $\mathbf{c}_{-f}$.

A.2 M-Step

Given K samples of $\mathbf{c}$ from E-step, denoted by $\{\boldsymbol{\zeta}^k\}_{k=1}^K$, we want to find solution to (6). We express

$$\begin{aligned} &P(\mathbf{c}, S'|S, A, \beta^{MT}) \\ &\quad = P_T(S'|S, A, \beta^{MT}, \mathbf{c})P(\mathbf{c}) \\ &\quad = \frac{1}{C^F} P_T(S'|S, A, \beta^{MT}, \mathbf{c}) \\ &\quad = \frac{1}{C^F} \sum_{\mathbf{c}'} \delta_{\mathbf{c},\mathbf{c}'} P_T(S'|S, A, \beta^{MT}, \mathbf{c}') \\ &\quad = \frac{1}{C^F} \prod_{\mathbf{c}'} P_T(S'|S, A, \beta^{MT}, \mathbf{c}')^{\delta_{\mathbf{c},\mathbf{c}'}}, \end{aligned} \tag{8}$$

where $\delta_{\mathbf{c},\mathbf{c}'}$ is Kronecker's delta, i.e., equal to 1 if and only if $\mathbf{c}$ and $\mathbf{c}'$ are equal.

The product form in (8) allows us to easily express the expectation of its logarithm as follows

$$\begin{aligned}
&\mathbb{E}_{\mathbf{c}} \log P(\mathbf{c}, S'|S, A, \beta^{MT}) \\
&= \mathbb{E}_{\mathbf{c}} \log \frac{1}{C^F} \prod_{\mathbf{c}'} P_T(S'|S, A, \beta^{MT}, \mathbf{c}')^{\delta_{\mathbf{c},\mathbf{c}'}} \\
&= \mathbb{E}_{\mathbf{c}} \left[\sum_{\mathbf{c}'} \delta_{\mathbf{c},\mathbf{c}'} \log P_T(S'|S, A, \beta^{MT}, \mathbf{c}') + \log \frac{1}{C^F} \right] \\
&= \sum_{\mathbf{c}'} \mathbb{E}_{\mathbf{c}}[\delta_{\mathbf{c},\mathbf{c}'}] \log P_T(S'|S, A, \beta^{MT}, \mathbf{c}') + \text{const.}, \qquad (9)
\end{aligned}$$

where the summation is over all possible assignments to $\mathbf{c}'$.

When summing over all $\mathbf{c}'$ the number of summation elements is exponential with respect to the number of factors F. However, since we have only K samples of $\mathbf{c}$ from E-step, denoted by $\{\boldsymbol{\zeta}_1, \ldots, \boldsymbol{\zeta}_K\}$ we can use them to approximate (9) as follows,

$$\begin{aligned}
&\sum_{\mathbf{c}'} \mathbb{E}_{\mathbf{c}}[\delta_{\mathbf{c},\mathbf{c}'}] \log P_T(S'|S, A, \beta^{MT}, \mathbf{c}') \\
&\approx \frac{1}{K} \sum_{k=1}^{K} \log P_T(S'|S, A, \beta^{MT}, \mathbf{c} = \boldsymbol{\zeta}_k) \\
&= \frac{1}{K} \sum_{k=1}^{K} \sum_{n=1}^{N} \sum_{f=0}^{F} \log P_T(x'_{n,f}|s_n, a_n, \beta^{MT}, \mathbf{c} = \boldsymbol{\zeta}_k). \qquad (10)
\end{aligned}$$

The maximization of (10) together with a standard prior $P(\beta^{MT})$ with respect to β^{MT} is tractable for multinomial models and for Gaussian models. For the details of the latter see B.

B EM Algorithm in the Case of Gaussian Models

Here are the detailed steps of the EM procedure for the Gaussian model, expressed in Eq.(2). Consider a state s and one of its factors $\mathbf{x}_f$ being affected by factors $\mathbf{x}_\mathbf{h}$ where $\mathbf{h} = g_f(s)$. Then for all possible components of $\mathbf{x}_f$, denoted by c_f, and all possible components of the affecting factors $\mathbf{x}_\mathbf{h}$, denoted by $\mathbf{c}_\mathbf{h}$, we have parameters $\beta^{MT}_{c_f,\mathbf{c}_\mathbf{h}} = (\mathbf{w}_{c_f,\mathbf{c}_\mathbf{h}}, \boldsymbol{\Sigma}_{c_f,\mathbf{c}_\mathbf{h}})$ that express the distribution of the next state of the factor $\mathbf{x}_f$ with the Gaussian $\mathcal{N}(\mathbf{x}_f + \mathbf{w}^\top_{c_f,\mathbf{c}_\mathbf{h}} \phi(\mathbf{x}_\mathbf{h}, a), \boldsymbol{\Sigma}_{c_f,\mathbf{c}_\mathbf{h}})$.

For clarity we recall that the unknown parameters of the Gaussian are $\mathbf{w}_{c_f,\mathbf{c}_\mathbf{h}} \in \mathbb{R}^{B\times C}$ and $\boldsymbol{\Sigma}_{c_f,\mathbf{c}_\mathbf{h}} \in \mathbb{R}^{C\times C}$ (see Eq.(2) for details).

B.1 E-Step

Given the current model for β^{MT} we carry out Gibbs sampling of c_f as expressed in (7) with

$$\begin{aligned} &P_T(\mathbf{x}'_{n,f}|s_n, a_n, c_f = c', \mathbf{c}_{-f}, \beta^{MT}) \\ &= \mathcal{N}(x_{n,f} + \mathbf{w}^\top_{c_f,\mathbf{c_h}}\phi(x_{n,\mathbf{h}}, a_n), \mathbf{\Sigma}_{c_f,\mathbf{c_h}}), \end{aligned}$$

where $\mathbf{c}$ denotes the combined c_f and $\mathbf{c}_{-f}$.

B.2 M-Step

Maximizing the log likelihood given the K samples $\{\boldsymbol{\zeta}_k\}_{k=1}^K$ of $\mathbf{c}$ from E-step. The maximization of β^{MT} can be carried out separately for each $\beta^{MT}_{c',\mathbf{c}''}$. For that we define $J(c', \mathbf{c}'') \mapsto (n, f, k)$ to return indices of all points $\mathbf{x}_{n,f}$ whose transition is affected by $\beta^{MT}_{c',\mathbf{c}''}$ when $\mathbf{c} = \boldsymbol{\zeta}_k$.

The part that depends on $\beta^{MT}_{c',\mathbf{c}''}$ in (10) can be written as

$$\frac{1}{K} \sum_{(k,n,f)\in J(c',\mathbf{c}'')} \log P_T(x'_{n,f}|s_n, a_n, \beta^{MT}_{c',\mathbf{c}''}, \mathbf{c} = \boldsymbol{\zeta}_k), \tag{11}$$

i.e., everything else in (10) is constant with respect to $\beta^{MT}_{c',\mathbf{c}''}$. Adding the log of prior $\log P_T(\beta^{MT}_{c',\mathbf{c}''})$ to (11) we can find the maximum a priori estimator (MAP) of the parameters $\mathbf{w}_{c',\mathbf{c}''}$ and $\mathbf{\Sigma}_{c',\mathbf{c}''}$ by the standard least-squares approach. Assuming a prior for the mean,

$$\log P(\mathbf{w}_{c',\mathbf{c}''}) = -\lambda\|\mathbf{w}_{c',\mathbf{c}''}\|^2,$$

where $\lambda \in [0, \infty)$, we get MAP solutions:

$$\begin{aligned} \mathbf{w}_{c',\mathbf{c}''} =& (\mathbf{\Phi}^\top\mathbf{\Phi} + K\lambda\mathbf{I})^{-1}\mathbf{\Phi}^\top\Delta\mathbf{x}, \\ \mathbf{\Sigma}_{c',\mathbf{c}''} =& \frac{1}{L_{c',\mathbf{c}''}}(\Delta\mathbf{x} - \mathbf{\Phi}\mathbf{w}_{c',\mathbf{c}''})^\top(\Delta\mathbf{x} - \mathbf{\Phi}\mathbf{w}_{c',\mathbf{c}''}), \end{aligned}$$

where $\mathbf{I}$ is the identity matrix. $\mathbf{\Phi}$ is a matrix where the i-th row is $\phi(x_{n,\mathbf{h}})^\top$ with $\mathbf{x}_n$ corresponding to i-th triplet $(n, f, k) \in J(c', \mathbf{c}'')$; similarly, the i-th row in matrix $\Delta\mathbf{x}$ corresponds to i-th $(\mathbf{x}'_{n,f} - x_{n,f})^\top$. $L_{c',\mathbf{c}''}$ denotes the size of $J(c', \mathbf{c}'')$ (thus, equal to the number of rows in both matrices). Note that $\mathbf{\Phi}$ and $\Delta\mathbf{x}$ are different for different c' and $\mathbf{c}''$, and for better readability we have dropped the indices of the matrices.

C Details of Initialization of EM Algorithm

Here we describe the derivation of initialization of β^{MT} in the EM algorithm.

C.1 M-Step

As in Appendix A.2 we try to maximize the expected log-likelihood. However, in this case we only use the data that depends on a single non-agent factor. This data is specified by index functions $I(f)$ for every $f \in \{1, \ldots, F\}$. Let $\bar{S}$, $\bar{S}'$ and $\bar{A}$ be the unions of these datasets, respectively, for starting state, next state and action. We want to maximize

$$\begin{aligned}&\mathbb{E}_{\mathbf{c}} \log P(\mathbf{c}, \bar{S}' | \bar{S}, \bar{A}, \beta^{MT}) \\ &= \mathbb{E}_{\mathbf{c}} \log P(\bar{S}' | \bar{S}, \bar{A}, \beta^{MT}, \mathbf{c}) + \text{const.}\end{aligned}$$

with respect to β^{MT}.

Due to the conditional independence of c_f we have

$$\begin{aligned}&P(\bar{S}' | \bar{S}, \bar{A}, \beta^{MT}, \mathbf{c}) \\ &= \prod_{f=1}^{F} \prod_{(n,h) \in I(f)} P_T(x_{n,h} | s_n, a_n, \beta^{MT}, \mathbf{c}) \\ &= \prod_{f=1}^{F} \prod_{(n,h) \in I(f)} P_T(x_{n,h} | s_n, a_n, \beta^{MT}, c_f) \\ &= \prod_{f=1}^{F} \prod_{(n,h) \in I(f)} \sum_{c'_f=1}^{C} \delta_{c_f, c'_f} P_T(x_{n,h} | s_n, a_n, \beta^{MT}, c'_f) \\ &= \prod_{f=1}^{F} \prod_{(n,h) \in I(f)} \prod_{c'_f=1}^{C} P_T(x_{n,h} | s_n, a_n, \beta^{MT}, c'_f)^{\delta_{c_f, c'_f}}.\end{aligned}$$

Analogously to the derivation of (9), we now get

$$\begin{aligned}&\mathbb{E}_{\mathbf{c}} \log P(\bar{S}' | \bar{S}, \bar{A}, \beta^{MT}, \mathbf{c}) \\ &= \mathbb{E}_{\mathbf{c}} \sum_{f=1}^{F} \sum_{(n,h) \in I(f)} \sum_{c'_f=1}^{C} \delta_{c_f, c'_f} \\ &\qquad \cdot \log P_T(x_{n,h} | s_n, a_n, \beta^{MT}, c'_f) \\ &= \sum_{f=1}^{F} \sum_{c'_f=1}^{C} \sum_{(n,h) \in I(f)} \mathbb{E}_{\mathbf{c}}[\delta_{c_f, c'_f}] \\ &\qquad \cdot \log P_T(x_{n,h} | s_n, a_n, \beta^{MT}, c'_f),\end{aligned}$$

where $\mathbb{E}_{\mathbf{c}}[\delta_{c_f, c'_f}]$ is the expectation calculated in the analytic E-step, Eq.(4).