Transfer Learning as Representation Selection

Trung Nguyen-Thanh
Tomi Silander
Tze-Yun Leong
School of Computing, National University of Singapore

Abstract
An appropriate representation of the environment is often key to efficient problem solving. Consequently, it may be helpful for an agent to use different representations in different environments. In this paper, we study selecting and adapting multiple abstractions or representations of environments in reinforcement learning. We address the challenges of transfer learning in heterogeneous environments with varying tasks. We present a system that, through a sequence of tasks, learns a set of world representations to be used in future tasks. We demonstrate the jumpstart and faster convergence to near optimum effects of our system. We also discuss several important variants of our system and highlight assumptions under which these variants should improve the current system.

1. Introduction
In reinforcement learning (RL), an agent learns how to make sequential decisions through observing the environment. Agent behaves according to a reward-optimizing policy which suggests an action to be taken in a given state. The agent’s learned knowledge, however, is specific to a task in an environment. A small change in task or its environment may render the agent’s accumulated knowledge useless; costly re-learning from scratch is often needed.

Transfer learning techniques proposed to address this shortcoming often assume that the agent uses the same state representation for all tasks. This assumption may not work well in real-life applications. For example, many environmental cues that help an agent navigate through forest are simply missing when the agent tries to navigate at sea. To efficiently accomplish similar but varying tasks in different environments, the agent has to learn to focus attention on the crucial features of each different environment.

In this paper we study a setting where the agent encounters many environments with different state spaces, thus with different goal states. The distribution of state features may also differ between environments. To achieve good performance quickly, the agent tries to select a different simple representation for each environment. The agent, however, often does not know beforehand how effective or useful the knowledge transfer will be. Moreover, it may only have time to learn a simple, approximate model that can be used in a new task.

We propose a system that tries to transfer old knowledge, but at the same time evaluates new options to see if they work better. The transferable knowledge is expressed as a library of state abstractions that implement different foci of attention. In different domains, different state abstractions may perform well; new combinations of features may be needed in some domains. A main contribution of this paper is to introduce multi-abstraction transfer, or multiple ways to see the world, that we call views. The aim is to learn to select a proper view for a new task.

The rest of the paper is organized as follows. We will next introduce our system, and then discuss the related work. We will then demonstrate the capabilities of our method via a set of experiments before we conclude with some discussions and ideas for future work.

2. Method
In reinforcement learning a task environment is typically modeled as a Markov decision process (MDP). An MDP is defined by a tuple $(S, A, T, R)$, where $S$ is a set of states; $A$ is a set of actions; $T : S \times A \rightarrow P(S|S, A)$ is a transition function indicating the probability of a
transition to a state \( s' \) upon taking an action \( a \) at a state \( s \); \( R \) is a reward function indicating immediate expected reward after the state transition \( s \xrightarrow{a} s' \). The goal is then to find a policy \( \pi \) that assigns an action to each state so that the expected future reward (possibly giving higher weights to more immediate rewards) at each state is maximized (Sutton & Barto, 1998). In model-based RL the estimation of optimal policy is based on estimates of transition model \( T \) and reward model \( R \). In this paper we try to improve model-based RL.

The central idea of this work is to allow agent to entertain different abstractions of its sensory state space. Such views correspond to the agent’s decision to focus attention on only some features of the state in order to quickly approximate the state transition function. Successfully doing so leads to a jumpstart in learning and faster convergence to a near optimal policy. In this paper, we concentrate on approximating the transition model. The reward model can be learned in an analogous fashion.

In order to cope with multiple environments and multiple state spaces, we formalize the views through relocatable action models so that they are independent of the details of any particular task and ready for transfer.

### 2.1. Relocatable action models

Relocatable action model (RAM) (Sherstov & Stone, 2005) is a tuple \((S, A, \zeta, O, \kappa, \tau, \eta, R)\), where \( S, A, R \) have the same meaning as in standard MDP. \( \zeta \) is a set of classes of states in \( S \); \( O \) is a set of action outcomes; \( \kappa : S \to \zeta \) is a state classification function; \( \tau : \zeta \times A \to P(O \mid \zeta, A) \) determines the outcome distribution; \( \eta : S \times O \to S \) is a next state function that we assume to be known by the agent. While different environments often have different states, the outcomes of actions, i.e., the ways the actions change the states, are assumed to persist from environment to environment. For example, an attempt, i.e., an action, to move left in a grid world often results a state that is otherwise very similar to pre-action state, but the agent has moved one step left. The relative change in states, “moved left”, is called an outcome of an action.

RAM gains efficiency by assuming that many states have similar outcome distributions for actions. The states are partitioned into classes so that the states in a class share similar outcome distributions. Therefore, in RAM, the idea is not to estimate the state transition function \( P(s' \mid s, a) \) directly, but to first estimate the outcome distribution of an action, and predict the state transition using a deterministic function \( \eta \) that maps the outcome and the pre-action state \( s \) to the next state \( s' = \eta(s, o) \). \( \tau \) implies a transition model \( T \) via, using Iverson brackets,

\[
T(s, a, s'; \tau, \kappa) = P(s' \mid a, s) = \sum_{o \in O} \tau(\kappa(s), a, o) [\eta(s, o) = s'] \tag{1}
\]

We will define the classes using state variables that are applicable in all environments. Consequently, the elements \( \kappa, A, O, \tau \) may be naturally understood in all environments.

### 2.2. Views

A view defines a way to partition the states into classes based on state variables. When agent adopts a view, it also adopts the assumption that the outcomes of the actions in states belonging to any single class of a view follow the same distribution. Notice that this assumption may or may not hold in reality. It is an abstraction or simplification used by an agent, and this abstraction may be useful or harmful. The task is to learn to select good views to quickly approximate world transition models.

As we will deal with several environments with different state spaces \( S \), we assume that the states can be characterized (but not necessarily identified) by features, i.e., that there is a vector valued function \( \vec{f} \) that maps the states into feature vectors. The function \( \vec{f} \) is thought to be defined by the agent’s sensory, feature extraction and memory capabilities, thus it is available in all environments. As in typical reinforcement learning, we assume a set \( A \) of actions that the agent can perform. In this work we assume that all the actions can be attempted in all the states of all the environments. We also assume that the environments define an immediate reward function \( R : (S, A, S) \to \mathbb{R} \).

**Definition 1.** View is a pair \((\kappa, \tau)\), where \( \kappa : \vec{f}(S) \to \zeta \) is a state classification function and \( \tau : \zeta \times A \to P(O \mid \zeta, A) \) determines the outcome distribution.

The classifier \( \kappa \) maps the state \( s \), based on its features, to a class \( \kappa(\vec{f}(s)) \in \zeta \), where \( \zeta \) is the (finite) co-domain of \( \kappa \), i.e., classes. Since we consider one agent, the function \( \vec{f} \) is global, and we use a shorthand \( \kappa(s) \) for \( \kappa(\vec{f}(s)) \). Through experience, the agent updates the outcome model \( \tau \) of a view. In this paper we assume the state features to be discrete and only implement a simple form of \( \kappa \) that selects a subset of features from \( \vec{f}(s) \). The state partition \( \zeta \) corresponds then to just selecting some dimensions from the feature vector \( \vec{f}(s) \). This implements the idea of attention.
Table 1. Two different views. The view (a) classifies grids based on their surface to brick and non-brick classes. The view (b), instead, classifies states into three classes: water, soil, and other surfaces.

(a) IF surface(s)=brick THEN ζ=c1 ELSE ζ=c2
(b) IF surface(s)=water THEN ζ=c1 ELSEIF surface(s)=soil THEN ζ=c2 ELSE ζ=c3

<table>
<thead>
<tr>
<th>ζ</th>
<th>Outcome distributions for actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td><img src="image1.png" alt="Outcome distributions" /></td>
</tr>
<tr>
<td>c2</td>
<td><img src="image2.png" alt="Outcome distributions" /></td>
</tr>
<tr>
<td>c3</td>
<td><img src="image3.png" alt="Outcome distributions" /></td>
</tr>
</tbody>
</table>

Example

In our experiments we will consider an agent trying to find an path to the goal state in different grid worlds. Each grid world is different, but the grids have features such as the surface type (water, sand, grass, brick, and fire), and the wall configuration around the grid. These features of states can be detected in all grid worlds. Our agent can try four different actions (move up, down, left, or right), but the actions may sometimes yield surprising outcomes (moved up, down, left, right, didn’t move). The actual outcomes of actions stochastically depend on surface and wall configurations.

Now the agent may, for example, pay attention only to whether the surface of the grid is brick or not (view (a) in Table 1). It would then classify all the states into brick and non-brick, and assume that the outcome probabilities for actions only differ for these to classes. All the experience gained by attempting actions in any non-brick state could then be pooled together and also utilized in any other non-brick state, possibly even in other grid worlds. Whether this view is a good one will be judged later by our learning agent.

3. View Transfer (VT) Algorithm

Let W be a set of all views we are considering. We assume that no two views share the same state classification function κ. In practice W may be very large in which case we would probably have to sample this set, but for now we assume that we consider all the views in W.

The key steps in our approach are presented in Algorithm 1. In prose, the VT algorithm receives as input the view set W, and a goodness profile Φ of the views in W. For each new task, agent selects the best view w* according to Φ and derives the corresponding initial policy for a new task. w*, however, may not be the best suited view for the task. Observed transitions after each action (or a few actions) are used to score all the views as a fundamental step for view re-selection later. Observed transitions are also used to adjust the parameters τ of all the views w in W. The view w* will be switched out and replaced with a view w+ if w+ scores at least d points higher than the w*. This process is repeated until policy π converges or when a stopping condition is satisfied. After completing the task, in order to achieve knowledge transfer, knowledge learned in the task about the views W must be updated to the knowledge base. Detail of this step is discussed in section 3.3.

One of the characteristics of VT is to treat exploitation as exploration. Apart from small ε-greedy stochasticity, the agent does not try to explore the new task, but just tries to exploit and adapt the views which it has learnt earlier. This is clearly a dangerous bias that may lead to suboptimal policies. However, any behaviour in a new environment leads to unexplored states, thus an element of exploration is partially preserved. Our results demonstrate the pros and cons of this bias.

3.1. Learning view parameters

A view w = (κw, τw) can be seen as consisting of a structure defined by the co-domain ζw of κw, and a quantitative part τw that defines the outcome probabilities of actions. A natural way to learn τ is via Bayesian updating of outcome probabilities. We assume that in a view w for a class ζw we have a parameter vector θ(ζw,a) such that θ(ζw,a) = P(o|ζw,a,θ). These parameter vectors are assumed to have a Dirichlet distribution Dir(α(ζw,a)) with α(ζw,a) = (α1(ζw,a), ..., α|a|). For notational convenience we aggregate the parameters α^w = {α(ζw,a)|a ∈ A} and α^w = {α^w|ζw ∈ ζw}.

When acting under any policy π, we take action a in
a state \( s \), which produces an outcome \( o_j \). The resulting pair \((s, o_j)\) may now be interpreted as a pair \((\kappa^w(s), o_j)\) in any view \( w \). Each view \( w \) is then updated by increasing the hyperparameters \( \alpha_w \) by one. We denote the result of updating hyperparameters \( \alpha_w \) from initial values \( \alpha_0^w \) using a sequence \( D = D_1^N = ((s_1,a_1,o_1),\ldots,(s_N,a_N,o_N)) \) of actions and their outcomes, with \( \alpha^w(D, \alpha_0^w) \).

Since the Bayesian parameter updating described above is consistent and since \( \epsilon \)-greedy exploration guarantees that asymptotically every (state,action)-pair is experienced infinitely many times, we have a following result.

**Theorem 1.** Within a task with state space \( S \) and a true transition function \( T^* \), if \( \exists w \in W \) with classifier \( \kappa \) such that \( \exists \tau^\kappa \forall (s,a,s') T(s,a,s';\tau,\kappa) = T^*(s,a,s') \), then VT will asymptotically converge to an optimal policy.

A corollary of the theorem above is that if we consider a “view” in which each concrete state forms a class of its own, this concrete view allows us (trivially) to express the true transition function and, consequently, to reach the optimal policy. In this paper, we consider only the transition model, and assume that the reward model is known. That the agent would asymptotically choose the view that allows the correct transition model to be expressed is a property of our view scoring method.

### 3.2. Scoring views

The optimal policy is the one that at each state picks the action yielding the highest expected reward. To calculate this expectation, the agent needs to know the transition probabilities for all actions at each state. While exact knowledge of all transition probabilities is unrealistic in practice, the ability to estimate these probabilities is important for estimating the optimal policy.

Each view yields an approximation of the true transition function via equation 1. We will measure the approximation ability of a view using a log-score which is known to be a proper score for predictors, thus the score is maximized if the predictor assigns probability \( p \) for outcomes that in reality happen with probability \( p \).

More formally, let \( D = \{d_1, d_2, \ldots, d_N\} \) be a set of outcomes \( d = (s,a,o) \) observed while operating on a task environment.

\[
Score(w, D) = \log P(D|w) = \sum_{i=1}^{|D|} \log P(d_i|D_1^{i-1}, w),
\]

where \( P(d_i|D_1^{i-1}, w) = P(o|s,a,\alpha_o^w(D_1^{i-1}, o_0^w)) = \int \theta_o^{(\kappa^w(s), a)} P(\theta_o^{(\kappa^w(s), a)} | D_1^{i-1}, o_0^w) d\theta_o^{(\kappa^w(s), a)} = E[\theta_o^{(\kappa^w(s), a)}], \)

and expectation is taken assuming

\[
\theta_o^{(\kappa^w(s), a)} \sim Dir(\alpha_o^w(D_1^{i-1}, o_0^w)).
\]

This means that after observing an event \( \kappa(s) \xrightarrow{a} o_j \), but before adapting the views, we score a view \( w \) based on the expected value of its \( \theta_o^{\kappa(s)} \). More specifically, the view \( w \) is granted \( \log P(o_j|\kappa^w(s), a) = \log \tau^w(\kappa^w(s), a_i, o_j) \) points for its outcome prediction performance. Computational complexity to update a view score is, therefore, \( O(1) \).

### 3.3. Transferring Views

For transfer purposes, we need to maintain a goodness profile, and the quantitative part of every view. A goodness profile can be expressed as a probability distribution over views parametrized by a vector \( \Phi \), such that \( \Phi_i = P(w_i|\Phi) \). Based on the success of different views after completing a task, the goodness profile is updated. To sequentially update the goodness profile after each task, we do not fix the goodness profile parameters \( \Phi \), but rather maintain a distribution over them, \( \Phi \sim Dir(\beta) \), where \( \beta = (\beta_1, \ldots, \beta_{|W|}) \). Initially, before the first task, we set \( \beta = 1 \), which corresponds to the uniform distribution for \( \Phi \). After each task, each view \( w \) has a score \( S^w \) that is a log-likelihood of data. We will then take the actual likelihoods, normalize them to get a distribution \( p \propto (e^{S^w_1}, \ldots, e^{S^w_{|W|}}) \).
and then update the profile distribution: \( \beta \leftarrow \beta + p \).

For the next task, the most promising view can now be selected simply by selecting the view \( w_i \) with the largest \( \beta_i \).

After each task, the quantitative parts of the views (see section 3.1) are updated as well. More specifically, after transferring the the parameters \( \alpha_{0w}^w \) to the new task and subsequently obtaining posterior \( \alpha_{nw}^w(D, \alpha_{0w}^w) \), we want to update the parameters of the successful views for future transfer. We do so by again turning the scores from the task to probabilities by \( p \propto (e^{S_{w_1}}, \ldots, e^{S_{w_i}}) \), and then updating \( \alpha_{new}^w \leftarrow p_{w\alpha}(D, \alpha_{0w}^w) \).

4. Related Work

Transfer learning has recently been an active research topic and many transfer learning methods have been proposed to transfer various forms of knowledge in RL. A recent survey by Taylor and Stone offers a relatively comprehensive exhibition of recent work (Taylor & Stone, 2009). In the following we describe some of the previous research that has interesting links to our work.

Abstraction selection has been applied in solving several issues in RL domains. Konidaris and Barto (Konidaris & Barto, 2009) focused on selecting the best abstraction to assist agent’s skill learning. Van et al. (Van Seijen et al., 2008) studied using multiple representations together to solve a RL problem. However, none of these studies solve the problem of transferring knowledge in heterogeneous environments.

Taylor and Stone (Taylor & Stone, 2007) studied transferring knowledge based on a fixed representation (of value function) to the new task in which another representation is used. The work does not deal with selecting a representation for the task.

Atkeson and Santamaria (Atkeson & Santamaria, 1997) suggested to use locally weighted transfer learning (LWT) to adapt previously learned transition models into a new situation. This study is among the very few that actually consider transferring the transition model to a new task (Taylor & Stone, 2009). While their work is conducted in continuous state space using fixed state similarity measure, it can be adapted to a discrete case. Doing so corresponds to adopting a fixed single view. We will compare our work with this approach in our experiments. This approach could also be extended to be compatible with our work by learning a library of state similarity measures and developing a method to choose among those similarities for each task.

Multiple models have previously been used to guide behaviour in non-stationary environments (Doya et al., 2002) (Silva et al., 2006). Wilson et al. (Wilson et al., 2007) retained multiple MDP models of previous tasks as prior knowledge to sample model for a new task. Unlike our work, these studies use one common state representation.

Sharma et al. (Sharma et al., 2007) built a library of state cases, abstracted actions to perform in those cases, and extracted their utilities to transfer between tasks. Such a case-base approach corresponds to a single, though evolving, representation of the environment. Fernández et al. (Fernández et al., 2010) transferred a library of policies learned in previous tasks to bias exploration in new tasks. The method requires the tasks to have the same state space, otherwise a state mapping strategy is needed.

5. Experiments

To demonstrate the efficacy of our view transfer (VT) approach, we consider a grid-based robot navigation domain. Each cell in the world has surface of one of the five materials, sand, soil, water, brick, and fire. In addition, there may be walls between cells. Surface and walls determine the stochastic dynamics of the world. The environment structure is available to the agent, but the dynamics model is not. The agent’s goal is to reach any exit door in the world consuming as little energy as possible. The agent can perform four actions (move up, down, left, right) which will lead it to one of the four states around it or leave it to its current state if it bumps into a wall. To perform an action, agent will spend 0.01 units of energy. It loses 1 unit if falling into a fire, but gains 1 unit when successfully reaching an exit door. A task ends when agent reaches a terminal state, i.e., any exit door or fire. This is a cost minimization problem. We used the term “reward” as cost, so instead of maximizing reward, we minimize reward.

We design sixteen tasks for the purpose. Each task has different state spaces, and terminal states. The distributions of the surface materials differ between environments. It is therefore important for the agent to focus on certain important features to efficiently learn the transition model in each environment. In our experiments, we generate all the environment transition models from one dynamics model which entails that every different combination of cell surfaces and walls around will lead to a different transition dynamics model at the cell. However, this (correct) view is
removed from agent’s view set to simulate the limited perception or processing capability of the agent.

We manually design a set of fourteen simple views for the agent to use. In each view, the states are grouped into classes based on different combinations of cell materials and surrounding walls. Examples of two such views are shown in Table 1.

For each view, our agent initially assumes the outcome distributions for all (class, action)-pairs to be equally probable. We represent effects of the actions as five outcomes (moved up, left, down, right, did not move). In the beginning, all the views are also assigned the same prior probability of goodness. When acting in an environment, the agent maintains certain inertia of views and only switches from \( v_x \) to \( v_y \) if score of \( v_x \) is \( d = \log(100) \) larger than score of \( v_y \). Since the agent transfers knowledge, it does not try to very actively explore the new environments. However, we allow little \( \epsilon \)-greedy exploration with \( \epsilon = 0.05 \).

5.1. Results

We present empirical results showing that VT can successfully transfer views that speed-up the learning process in various environments. Specifically, we show that a) selecting views using the proposed scoring metric works, b) our method achieves a good jump start, and c) that our method also performs well in the long run.

We conducted leave-one-out cross-validation with sixteen different tasks. In each scenario the agent was first allowed to experience fifteen tasks; over 100 episodes in each, and it was then tested on the remaining one task. We selected representative patterns to describe the results. The results are averaged over 50 runs.

View Selection vs. Random View

Our first experiment tests whether the view selection mechanism works well. Figure 1(a) shows how different views lead to different policies with different accumulated rewards over 100 episodes. It also includes the best view for the task, which VT tries to find among views. As seen an inappropriate view may quickly result in a reward that is even less than the reward of a “concrete view” in which all the states form their own class. We can also see that our view selection mechanism (VT) picks good views, and very quickly achieve high accumulated reward. However, sometimes there are views that perform better than our initial selection. In those cases we observe that our agent realized this fact only after some episodes, and even if it switched to them, it never caught them in the accumulative reward sense. For instance, at around episode 80, we see that VT lose to the best view, because VT explores better solution, and temporarily switches to the non-optimal views. We observe that the agent switch back to the optimal view shortly after that, but in the term of accumulated reward we will still see the gap as in Figure 1(a).

We also estimate the expected performance of an agent (AVG) that randomly selects views. The view selection algorithm of VT always achieves better performance than AVG in all test cases. Due to view switching, VT sometimes performs better than any single view. This behaviour is similar in spirit to smoothing, in which the agent is trained using increasingly complicated tasks. In our case, the agent first uses simple but rough approximations of the transfer function because such approximations can be quickly learned. After gathering more evidence, it then switches to more accurate and complicated views/approximations.

Multi-views vs. One-view, and Non-transfer

We also want to see if maintaining several views really helps to improve jumpstart. We compare our VT with LWT as discussed in 4. The discrete version of LWT corresponds to using the single view that has proven to work best in earlier tasks. From Figure 1(b) we see that our selection method outperforms LWT in this test task. However, when LWT is truly the best view for a task, our method also quickly picks that view, thus we never lose much but often gain a lot (see also Table 2).

To compare the jumpstarts of VT and non-transfer approaches, we study prioritized sweeping (PS) with full backup. PS is a popular RL algorithm, known as less data and less time method (Moore & Atkeson, 1993). As seen in Figure 1(b), VT achieves higher jump-start than PS. A reason is that PS does not use any prior-knowledge, thus its reward for the first episode is achieved with a random strategy.

The average total rewards after the first episode in all 16 test tasks are listed in Table 2. From the table we can see that VT always gains higher reward than PS and PR. Compared to LWT, VT loses in some scenarios, but not much.

Asymptotic Performance of VT

To study the asymptotic performance of VT, we compare it with popular RL methods, prioritized sweeping (PS), and RMAX. We ran each test 50 times for 1000 episodes, and measured the average cumulative
reward. We also compare VT with full certainty equivalence (CE) RL since it corresponds to the VT method with the concrete view. Notice that such a view is specific to the environment and cannot be transferred.

VT outperforms PS, RMAX, and CE in all cases. PS appears to learn faster than RMAX and CE. In 1000 episodes, the RMAX, however, seems to converge faster than the other two, but it still does so later than our method. Figure 1(c) shows that our agent can utilize views to gain higher reward from the beginning and that it can maintain that quality later when PS, RMAX, or CE converge. We also observe that RMAX tends to approach our agent’s performance after about 300 episodes.

6. Discussion and Conclusion

In this work, the views are transferred with a relatively strong preference for appropriate outcome distributions. It would be possible to also consider not transferring the outcome model at all, but just transferring the state classification $\kappa$. The success of such an approach depends on the stability of action outcomes in different environments. It would also be possible to consider environments as generative mechanisms for outcome probabilities and then fit the hyperparameters of the Dirichlet distribution to these probabilities, so that the robustness of parameters would be automatically, and separately assessed for different views.

We also limited ourselves to the situation where the world will be changed only when agent takes an action. In general there may be other agencies changing the world, and the next state cannot easily be expressed as a combination of a small set of outcomes and deterministic next state function $\eta$. Adjustments to the RAM theory are needed to accommodate dynamic worlds.

In the current study we used our expertise to select a set of views. In more realistic setting the state features span a large space of views and considering them all may not be possible. Some kind of learning of good views is needed. Our scoring function makes this possible. We score the views based on their prediction capability of action outcomes while traditionally the success in RL is measured by the reward. On one hand, our choice is natural given our interest in representation of the environment. Furthermore, this avoids costly learning of policies for each view. Currently, the views are based on environmental cues only. It would be most natural to base them on different goals too.

We have used Bayesian machinery in several learning tasks, as is commonly done in model based learning. However, our interest is mainly in the situation where the true transition model is outside of our model family, thus no convergence to the truth is possible even asymptotically. This questions the appropriateness of the selected theoretical framework. We may defend the selection by interpreting the marginal likelihoods used as prequential scores (Dawid, 1984) and interpreting Bayesian updating as a method for universal modeling (Rissanen, 1984).

In conclusion, we have studied the idea of transferring multiple state abstractions in reinforcement learning in
Transfer Learning as Representation Selection

a situation where task environments differ from each other. Not surprisingly, state abstraction gives the agent a jumpstart in a new task compared to non-abstracting methods. When the environments are different, the combination of learning multiple views and dynamically selecting the ones that seem to apply to current environment yields a system that can learn a good policy faster in a new environment. Compared to learning just a single good view and using it in all occasions, our system never loses much but sometimes gains a lot.

There are lot of possibilities for related studies using the framework we have introduced. We wish that our work would incite more studies in this area.

Acknowledgements

This research is partially supported by a GAMBIT grant "Tools for Creating Intelligent Game Agents", no. R-252-000-398-490 from the Media Development Authority and an Academic Research Grant no. T1 251RES1005 from the Ministry of Education in Singapore.

References


