Transferring Models in Hybrid Reinforcement Learning Agents

Anestis Fachantidis, Ioannis Partalas, Grigorios Tsoumakas, Ioannis Vlahavas

Transferring Models in Hybrid Reinforcement Learning Agents

Anestis Fachantidis, Ioannis Partalas, Grigorios Tsoumakas, Ioannis Vlahavas

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{afa,partalas,greg,vlahavas}@csd.auth.gr

Abstract. The main objective of transfer learning is to reuse knowledge acquired in a previous learned task, in order to enhance the learning procedure in a new and more complex task. Transfer learning comprises a suitable solution for speeding up the learning procedure in Reinforcement Learning tasks. In this work, we propose a novel method for transferring models to a hybrid reinforcement learning agent. The models of the transition and reward functions of a source task, will be transferred to a relevant but different target task. The learning algorithm of the target task's agent takes a hybrid approach, implementing both model-free and model-based learning, in order to fully exploit the presence of a model. The empirical evaluation, of the proposed approach, demonstrated significant results and performance improvements in the 3D Mountain Car task, by successfully using the models generated from the standard 2D Mountain Car.

1 Introduction

Reinforcement Learning (RL) is popular for its ability to deal with complex problems using limited feedback. However, RL algorithms often require a considerable amount of training, especially when they are applied to problems with very large and/or continuous state spaces.

Besides this, data efficiency is another important factor. Data efficiency refers to the ability of an agent to compute a good policy with less data. In the case of RL less data means less interaction with the environment which is specially important in real world applications such as robotics, where much interaction with the environment needs time and can be costly. Two of the most important approaches to speed up the learning process and boost data efficiency are Transfer Learning and Model - Based RL algorithms.

Whereas direct RL algorithms compute for long periods before finding good policies *Model-based algorithms* learn an explicit model of the environment and at the same time, the value function which determines the agent's policy. The model generated from this learning process can either be used for generating simulated experiences or for planning.

Transfer learning refers to the process of using knowledge that has been acquired in a previous learned task, the *source task*, in order to enhance the learning procedure in a new and more complex task, the *target task*. The more similar those two tasks are, the easier it is to transfer knowledge between them.

The key insight of our approach is to achieve maximum data efficiency and knowledge exploitation by combining transfer learning with model based learning. We do so by adopting a hybrid Dyna-like approach for our agent, in order to maximize the use of the model. The proposed approach is dubbed ***T**ransfer learning **i**n **M**odel-based **R**einforcement **L**earning **A**gents* (TiMRLA). The empirical evaluation, of the proposed approach, demonstrated significant results and performance improvements in the 3D Mountain Car task, by successfully using the models generated from the standard 2D Mountain Car.

2 Learning the model of a source task

In order to capture the dynamics of the source task environment we have to learn models of its transition and reward functions. Our main concern here is to achieve a balanced presence of the state space in the training sample and maximize the accuracy of the learned models.

To gather experiences and generate the source's task model, we use the model-based RL algorithm *Model Based Policy Improvement* (MBPI) [4]. MBPI is a model-based learning algorithm that can learn, on-line, the model of a task from real, gathered, experiences. Thereafter, MBPI uses that model to generate simulated experiences and learns directly from them using Sarsa.

In continuous domains, a value approximation technique is necessary in order to approximate the value of a state using nearby state values. We use the Cerebral Model Articulation Controller (CMAC) [1], a well-known tile coding function approximation scheme.

Our specific choice of the MBPI model-based learning algorithm in the source task, should not be considered as a limitation of the method as every other model-based RL algorithm can be equally considered, as long as it can provide us with models of the reward and transition functions of the task. The simulated experiences, in the form of a tuple $< s, a, r, s' >$, form a training set that will be used to learn the transition and reward functions of the task.

In our specific implementation, we use two cascade neural networks to learn the models of these functions. Cascade neural networks were chosen as the characteristics of this type of networks were considered suitable for the RL scheme [7].

Next, the agent of the source task starts to learn and progressively creates the requested models. Every e episodes MPBI learns a new model of the transition and reward functions, based on all the gathered experiences so far. In our implementation these new models are kept only if their Validation mean square error (MSE) is less than that of the previously generated models. We choose Validation MSE and not Train MSE to avoid overfitting and test the ability of the model to predict from unseen data, such as those in the validation data set.

3 Transfer Learning with TiMRLA

The TiMRLA algorithm (see Algorithm 3.3) consists of two learning mechanisms which are set up to be compatible. This hybrid-learning approach is followed, as explained later, in order to make maximum use of the source's task model.

3.1 Model-based learning

TiMRLA implements model based learning by generating mental (non-real) experiences from a model and directly learning from them with Sarsa (policy improvement) (line 30). This model can either be a source task model thus implementing transfer learning, or the new target task model being trained while learning in the target task. In the beginning of each episode the two models compete for being the current episode's "Model" (line 7). The choice is based on their error on predicting the afterstates from the current agent's state, this error is calculated as the euclidean distance between a predicted state and the one experienced (line 23).

3.2 Direct learning

Direct learning is implemented using Sarsa(λ) accompanied with eligibility traces for temporal-credit assignment. This is a model-free approach but in our implementation is accompanied by a model-based mechanism, *Action Determinant*, that assists the model-free learning process. This approach could be considered closer to that of Reward Shaping [3]. Reward Shaping is a methodology in which directly altering and adjusting the reward an agent receives, can improve its performance by biasing certain state-action values according to some prior knowledge. A simple, in form, quantity called Action Determinant is being calculated in every step (line 17). This quantity is calculated for each action α and consists of two parts. The first, CMAC.$V(s'_{model})$ is according to our model, the value of the state following our current state if action α was to be applied. The second part of the quantity, r_{model} is the reward predicted from our model for the specific state-action pair (s, α). For each action $\alpha \in A$ the sum of the two quantities is summed with the respective $Q(s, a)$ value thus creating $|A|$ new values. These values will be used for action selection with real experiences. So, when being greedy, the agent will make a more informed choice of actions based on the CMAC weights of the predicted afterstates and on the predicted reward.

3.3 Transfer Learning

Learning in the source task provided us with the best, in terms of Validation MSE, Neural Networks. These three networks are used by TiMRLA to predict next states, whether they are terminal, and their rewards.

We describe the flow of the proposed algorithm as it executes in the target task. In lines 1-3 some basic initializations are done. The weights in the CMAC,

representing our value function $Q(s, a)$, are all set to zero and the initial model to be used is the Source Model. In the beginning of each episode we obtain the start state and we select an action (ϵ-greedy) with respect to the policy. Next, takes place the process of choosing the model that will be used (lines 7-9) in the specific episode. In lines 12-13 we obtain for each step, the current state and its reward. We then optionally calculate the quantities for the Action Determinant by predicting the next state and its value, according to our function approximator as also the reward. This is calculated for each action and passed as a parameter to the function that implements the ϵ-greedy action selection (line 19). Next model-free learning with Sarsa(λ) takes place, with its update rule using an added "CMAC" notation to emphasize the fact that these values come from the approximation of the CMAC.

Next, in lines 21-24 and for each even numbered step (not in every step, for robustness) the two models (target and source) are queried for the new state s' (already known) based on the preceding state-action pair (s, a). We then compute and compare two euclidean distances $d(s', s'_{target})$ and $d(s', s'_{source})$ where the second argument in the distances, represents the state predictions of the two models. The distances are stored in a tabular form with a window of 1000 distances and the two mean errors are calculated for that window. As mentioned before, at the start of each episode, based on this averaged error (distance), TiMRLA chooses to use the target or the source task model (the one with the less error).

In the next lines the flow continues with the MBPI algorithm for policy improvement using generated experiences from the model.

4 Experiments

4.1 Generalized Mountain Car

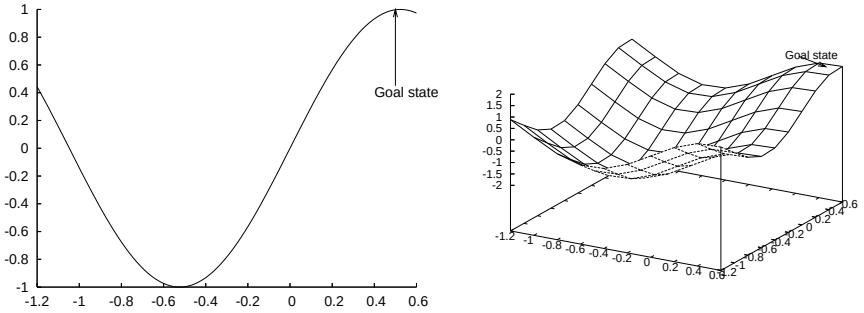
The mountain car domain was originally proposed by [6] and extended by [8]. In the standard 2D task an underpowered car must be driven up to a hill. The state of the environment is described by two continuous variables, the horizontal position $x \in [-1.2, 0.6]$, and velocity $v_x \in [-0.007, 0.007]$.The actions are {Neutral, Left and Right} which modify the velocity by $-0.001, 0$ and 0.001 respectively. At each time step, an amount of $-0.0025 * \cos 3x$ is added to the velocity, in order to take gravity into account.

Each episode starts with the car at the bottom of the hill and ends when x becomes greater than 0.5. At each time step, the agent selects among the three available actions and receives a reward of -1. The objective is to move the car to the goal state, as fast as possible. The main difficulty in this environment is that the force of gravity is greater than the force that the engine is capable to provide. Therefore, the agent controlling the underpowered car must move back and forth, in order to exploit the effect of gravity to its favour and manage to reach the goal state.

The 3D mountain car extends the 2D task by adding an extra spatial dimension. The 3D task was originally proposed in [12]. The state is composed by

Algorithm 1 Transferring Models in Reinforcement Learning Agents - TiMRLA

$Q \leftarrow Q_0$ $D \leftarrow 0$
$Model \leftarrow SourceModel$
repeat for each episode
  $s \leftarrow StartStateFromEnvironment()$
  $\alpha \leftarrow Select\_e\_GreedyAction(Q, ActionDeterminant[])$
  if $TargetModel.error < SourceModel.Error$ **then**
    $Model \leftarrow TargetModel$
  end if
until end
repeat for each step
  $s'_{mdl} \leftarrow nextStateFromEnvironment$
  $r \leftarrow reward$
  for each action α in A **do**
    $s'_{mdl} \leftarrow Model.PredictState(s', \alpha)$
    $r_{mdl} \leftarrow Model.PredictReward(s', \alpha)$
    $ActionDeterminant[\alpha] \leftarrow \text{CMAC}.V(s'_{mdl}) + r_{mdl}$
  end for
  $\alpha' \leftarrow Select\_e\_GreedyAction()$
  $CMAC.Q(s, \alpha) \leftarrow CMAC.Q(s, \alpha) + a(r + \gamma CMAC.Q(s', \alpha') - CMAC.Q(s, \alpha))$
  $s'_{trg} \leftarrow TargetModel.PredictState(s, a)$
  $s'_{src} \leftarrow SourceModel.PredictState(s, a)$
  $SourceModel.Error \leftarrow SourceModel.Error + d(s'_{src}, s')^2$
  $TargetModel.Error \leftarrow TargetModel.Error + d(s'_{trg}, s')^2$
  $s \leftarrow s'; \alpha \leftarrow a'$
  if $NonTrainingPeriodElpased = false$ **then**
    $D \leftarrow D \cup (s, \alpha, r, s')$
  else
    $LearnTargetModel(D)$
    $ImprovePolicy(Q)$ //Using MBPI in our case
  end if
until s' terminal



(a) (b)

Fig. 1. (a) The Mountain Car 2D task. (b) The Mountain Car 3D task.

four continuous variables, the coordinates in space x, and $y \in [-1.2, 0.6]$, as well as the velocities v_x and $v_y \in [-0.07.0.07]$. The available actions are {Neutral, West, East, South, Nourth}. The Neutral action has no impact on the velocity of the car. The West and East actions add -0.001 and 0.001 to v_x respectively, while South and North add -0.001 and 0.001 to v_y respectively. Additionally, in order to simulate the effect of gravity, at each time step, $-0.0025 * \cos 3x$ and $-0.0025 * \cos 3y$ is added to v_x and v_y respectively. Each episode starts with the car at the bottom of the basin. The goal state is reached when $x \geq 0.5$ and $y \geq 0.5$. At each time step, the agent receives a reward of -1. In this work we use the mountain car software[1] that is based on version 3.0 of the RL-Glue library[2] [9].

4.2 Setup

After thorough experimentation we concluded to a set of parameters for the MBPI agent in the source task. We set ϵ to 0.5 decaying by a factor of 0.99, the step-size parameter α to 0.2 and γ, the discounting factor, to 1 as this task is an episodic one.Also, depth d of the MBPI algorithm was set to 1 as [4] show that a greater depth decreases the overall learning performance.

As the standard MBPI algorithms specifies, after every e episodes, training of the networks takes place based on the set of experiences D gathered so far. We set e to 40 episodes. The neural networks produced are compared with the previously used, based on their MSE, finally keeping the best of them.

As our methodology allows for different state and action variables between the source and the target task, we used manual inter-task mappings. For our experimental domain this is successfully done in [11]. Manual mapping of the different actions and states should not be seen as a limitation of the method, as there exist methods that are able to automatically learn these mappings [12]. Table 1 presents the inter-task mapping that we used in our experiments.

For the model-based learning part of the algorithm, we got the best results by setting the model-based policy improvement period to 1, which means that policy improvement takes place in every episode. The iterations n were set to 2000, and e and the step-size parameter a, was set to 0 (greedy) and 0.07 respectively. The above values of the parameters are those who gave us the best results.

Finally, for the optional part of our algorithm that specifies the Action Determinant, we run the experiments for both versions of TiMRLA, with and without Action Determinant (TiMRLA+AD).

Both versions of the TiMRLA agent were compared with the standard Sarsa(λ) with a CMAC function approximator. In order to isolate and evaluate the effect of transferring the source model, the two agents had exactly the same parameters for learning from real experiences, with ϵ set to 0.5 , a to 0.2 and γ to 1.

[1] Available at http://library.rl-community.org/
[2] Available at http://glue.rl-community.org/

3D action	2D action	3D variable	2D variable
Neutral	Neutral	$\mathbf{x}$	$\mathbf{x}$
North	Right	$\mathbf{v_x}$	$\mathbf{v_x}$
East	Right	$\mathbf{y}$	$\mathbf{x}$
South	Left	$\mathbf{v_y}$	$\mathbf{v_x}$
West	Left		

Table 1. Inter-task Mappings used by TiMRLA to construct target task instances from source task data.

5 Results and Discussion

Figure 2 presents the average cumulative reward that is obtained by each algorithm against the learning episodes.

The graphs were produced from 20 independent trials per method and with a smoothing-averaging window of 20 episodes for clarity purposes. They show a clear performance gain of the proposed algorithm TiMRLA over the standard model-free agent, which is also statistically significant at $p < 0.05$ (one-tail t test). Its performance demonstrates the positive effect of transferring a model from a source task and also that this model can still assist learning even after many episodes.

Clearly, TiMRLA+AD, shows a more unstable performance having a slight performance gain at the start that decreases gradually after 300 episodes. Its average reward difference with the standard model-free agent is still statistically significant at $p<0.05$ but for the first 400 episodes. This feature for model based assisting in the action selection, although promising, needs major adjustments such as proper decaying of its influence that will allow the method to positively influence learning in the beginning and gradually decrease its effect.

Furthermore we calculated the mean predictive accuracy of the mountain car 2D model. The mean euclidean distance of a real afterstate s' to the predicted one was 0.018 for the state variables x and y.

For the simulated experiences, it is important to note that the best results were obtained with policy improvement taking place in every episode. This shows us that the models succeeded in producing real-like experiences that were positively influencing the learning algorithm just like complementary steps. Although that, the tuned value for the step-size parameter was 0.07 being less than that used for the real experiences which was set to 0.2. This can be seen as a reasonable consequence of using approximate models since they are error prone and can produce at times, errors that could negatively alter the learning procedure.

Finally, an interesting finding is that the best results were obtained with the simulated actions ϵ parameter set to zero. We hypothesize that the model-based learning part, favours greedy action selection because the agent explores anyway by just being placed at random start states. This sub-optimal placing is the exploration mechanism in the agent's mental experiences, and he balances by implementing greedy action selection.

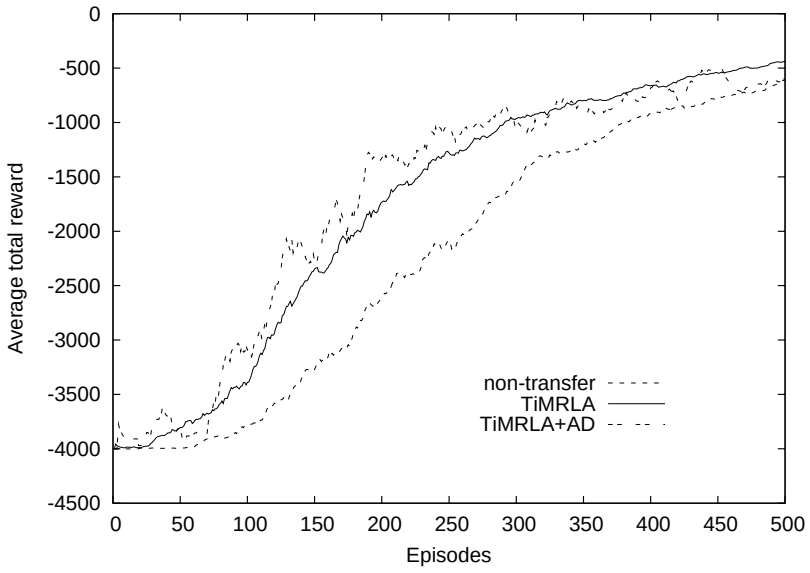


Fig. 2. Average reward in a period of 500 episodes. The curves are smoothed with a window of 20 episodes

6 Related Work

This section presents related work in transfer learning and contrasts it with the proposed approach. For a broader review of transfer learning in reinforcement learning domains the reader can refer to a comprehensive survey of the field [10].

Fernadez and Veloso [2] propose an approach that reuses past policies from solved source tasks to speed up learning in the target task. A restriction of this approach is that the tasks (source and target) must have the same action and state space. Our method allows the state and action spaces to be different between the target and the source task.

Advice based methods have been proposed in [13, 15]. Taylor and Stone [13] proposed a method that uses a list of learned rules from the source task as advice, to guide the learning procedure in the target task. The authors introduce three different utilization schemes of the advice. Furthermore, Torrey et al. [15] export rules in first-logic order and translate them into advices.

Taylor et al. [14] proposed a method, named *Transfer via inter-task mappings*, that initializes the weights of the target task, using the learned weights of the source task, by utilizing mapping functions. The method depends strongly on the function approximation scheme that is used in both tasks, as the function that transfers the weights is altered according to the approximation method.

The transfer of samples have also been proposed in [11, 5]. Lazaric et al. [5] proposed a method that uses samples gathered in the source task, in order to

train a batch reinforcement learning algorithm in the target task. The state and action spaces are not allowed to differ between the two tasks. Taylor et al. [11] present an algorithm for transferring samples in model-based algorithms. The procedure of transfer is accomplished on-line, that is, when the agent interacts with the environment and the state-action space can change.

7 Conclusions and Future Work

We examined the viability and benefits of transferring the full model of a task to a different but relevant task. The tasks are allowed to have different actions and state variables by constructing and using the right inter-task mappings. The use of cascade neural networks in an RL scheme showed promising results by providing us with accurate enough models of the transition and reward function of the the source task.

The source's task model was successfully used from a novel algorithm capable of fully exploiting its presence. The proposed algorithm is equipped with two learning mechanisms (hybrid) one for model-free learning and one for model-based. Empirical results demonstrate a significant performance improvement when compared to the "no-transfer" case. The model of a source task, created with supervised learning, can be efficiently used in a standard model-free learning algorithm either through a model-based add-on module, or directly, with approaches such as that of Reward Shaping. Moreover concurrent learning with the two mechanisms can be compatible as also feasible and efficient.

Proposals for future work include the evaluation of the method with the use of different model-based algorithms and the improving, upon our design, of the model-free learning mechanism with the use of more sophisticated forms of Reward Shaping. Also, the development of new hybrid learning algorithms may further demonstrate the advantages of this approach. Finally, a lot of work has to be done in the field of trustworthy and informative evaluation metrics, in the context of transfer learning for Reinforcement Learning.

References

1. J.S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, 97:220, 1975.
2. F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *5th international joint conference on Autonomous agents and multiagent systems*, pages 720–727, 2006.
3. Vijaykumar Gullapalli, , Vijaykumar Gullapalli, and Andrew G. Barto. Shaping as a method for accelerating reinforcement learning. In *In Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 554–559. IEEE, 1992.
4. Shivaram Kalyanakrishnan, Peter Stone, and Yaxin Liu. Model-based reinforcement learning in a complex domain, 2008.
5. Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Transfer of samples in batch reinforcement learning. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 544–551, 2008.

6. Andrew Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Machine Learning: Proceedings of the Eighth International Conference*, 1991.
7. Franois Rivest and Doina Precup. Combining td-learning with cascade-correlation networks. 2003.
8. Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158, 1996.
9. Brian Tanner and Adam White. Rl-glue: Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, 2010.
10. Matthew Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.
11. Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, volume 5212, pages 488–505, September 2008.
12. Matthew E. Taylor, Gregory Kuhlmann, and Peter Stone. Autonomous transfer for reinforcement learning. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 283–290, 2008.
13. Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 879–886, 2007.
14. Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8:2125–2167, 2007.
15. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Skill acquisition via transfer learning and advice taking. In *17 th European Conference on Machine Learning*, pages 425–436, 2005.