

Learning Inter-Task Transferability in the Absence of Target Task Samples

Paper 652

ABSTRACT

In a reinforcement learning setting, the goal of transfer learning is to improve performance on a target task by re-using knowledge from one or more source tasks. A key problem in transfer learning is how to choose appropriate source tasks for a given target task. Current approaches typically require that the agent has some experience in the target domain, or that the target task is specified by a model (e.g., a Markov Decision Process) with known parameters. To address these limitations, this paper proposes a framework for selecting source tasks in the absence of a known model or target task samples. Instead, our approach uses meta-data (e.g., attribute-value pairs) associated with each task to learn the expected benefit of transfer given a source-target task pair. To test the method, we conducted a large-scale experiment in the Ms. Pac-Man domain in which an agent played over 170 million games spanning 192 variations of the task. The agent used vast amounts of experience about transfer learning in the domain to model the benefit (or detriment) of transferring knowledge from one task to another. Subsequently, the agent successfully selected appropriate source tasks for previously unseen target tasks.

Categories and Subject Descriptors

I.2.6 [Learning]: Miscellaneous

General Terms

Experimentations, Performance

Keywords

Single and multi-agent learning techniques, Transfer Learning, Reinforcement Learning

1. INTRODUCTION

Many learning tasks provide limited prior knowledge and minimal environmental feedback. Temporal difference methods have shown many successes in learning such tasks. However, in complex domains training the agent can be computationally expensive and may require large amounts of experiential data. Thus the goal of transfer learning is to re-use

previously learned knowledge when learning to solve a new task. After all, human beings (as well as other primates) rarely solve new problems from scratch but instead transfer learned skills to novel situations [17, 4, 12].

In a typical transfer learning scenario, the agent solves a *target* task by leveraging knowledge from 1 or more *source* tasks. An agent may transfer individual samples [27, 15, 14], a learned action-value function [28, 6], a policy [9, 8], or a model [18, 7]. Most current research in TL assumes that a good source task has already been identified [29]. The main limitation of the few existing approaches to source task selection is that they typically require the agent to already have some experience (e.g., training samples) in the target domain or for the target domain to be specified using a model (e.g., a Markov Decision Process) with known parameters.

To address these limitations, we propose a framework for selecting appropriate source tasks that uses meta-data – more specifically, attribute-value pairs – associated with each task. Our main hypothesis is that given parameters or attributes that describe two tasks, an agent may learn the benefits (or lack thereof) of transferring knowledge from one of them to the other. To test this hypothesis, we conducted a large-scale experiment in the Ms. Pac-Man domain in which the agent played over 170 million games spanning 192 variations of the task. For each source-target task pair, the agent measured the jump start in performance on the target task as a result of applying value-function transfer [30]. Subsequently, the agent learned a regression model of the transferability for any given pair and successfully used it to select appropriate source tasks for a new set of target tasks. To our knowledge, this is the largest computational experiment in transfer learning conducted to date.

The rest of the paper is organized as follows. Section 2 gives an overview of related works and discusses where this study falls in the landscape of the field. Section 3 provides background on temporal difference learning and value function transfer methods that were used in this paper. Section 4 describes the framework for learning and using an inter-task transferability model, while Section 5 describes the domain and the experiment that was conducted to test the framework. Finally, Section 6 summarizes the results, and discusses the limitations and implications of this work.

2. RELATED WORKS

In recent years, research in transfer learning (TL) has improved the performance of reinforcement learning methods by enabling them to re-use knowledge from one task to another (see [29] and [13] for a review). For example, an

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, Bordini, Elkind, Weiss, Yolum (eds.), May, 4–8, 2015, Istanbul, Turkey.

Copyright © 2015, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

agent may transfer individual samples [27, 15, 14], a learned action-value function [28, 6], a policy [9, 21, 8], or a model [18, 7]. In situations where the state and/or action spaces differ across tasks, an agent can learn an inter-task mapping [3] or use a hard-coded one provided by a human teacher [31]. Most TL methods assume that the source task has already been selected and that it is indeed a good source task for the target task [29]. In a more general case, however, an agent may have to choose appropriate source tasks on its own when faced with learning a complex novel task. This problem is referred to as *source task selection* and has received relatively little attention [29]. This problem is particularly important when some of the potential source tasks may be irrelevant to the target task, in which case the agent can suffer from negative transfer.

The few existing methods for source task selection typically assume that the agent has some experience (e.g., training samples) in the target task or that a model of the target task is available to the agent. For example, the method described by Lazaric *et al.* [15] enables an agent to select relevant samples from known source tasks by comparing them to samples collected in the target task. In their experiments, the agent was able to effectively choose samples from two source tasks to speed up learning on the target task. A different approach to the problem is that of Nguyen *et al.* [18] where instead of transferring samples, the method transfers learned expectation models of how the environment changes as a result of the agent’s actions. In that framework, the agent learns expectation models from a set of known source tasks and then dynamically identifies which of these models are useful when learning the new target task. Similarly, Perkins and Precup [19] describe an approach in which the agent learns reinforcement learning options on a set of source tasks and then uses them on a target task. While learning the target task, the agent estimated the value of known options by maintaining a belief about the target task’s identity with respect to the known tasks.

Another model-based approach to the problem is described by Ammar *et al.* [2]. The authors propose a novel similarity measure for Markov Decision Processes which is shown to be effective at selecting good source tasks for a target task. Like other model-based approaches for transfer learning, the method proposed by Ammar *et al.* requires that the agent has access to a good estimate of the target task’s MDP, which in practice may not always be available.

In contrast to existing methods for source task selection, this paper addresses the problem under the assumption that no samples from the target task are available. Instead, we consider the case where tasks are described using a fixed-length feature vector and thus, the agent is tasked with learning transferability across tasks using such meta-data. While most existing methods are evaluated only on a single target task, our empirical evaluation is conducted using a large-scale experiment in which the agent learns pairwise task transferability for a large number of tasks.

Another area of research that is relevant to this study is that of case based reasoning (CBR) [1, 16]. When faced with a new problem, CBR methods typically find similar problems (i.e., cases) that have already been solved in the past and re-use their solution. This is typically done by the use of a similarity function that can be used to identify relevant cases. The major limitation of applying CBR for source task selection is that it requires the similarity function to be

indicative of whether or not transferring from one task to another will result in positive or negative transfer. Therefore, while we evaluate the approach of using task descriptors to compute task similarity and select sources accordingly, the work here proposes that an agent can learn to directly predict the outcome of transfer from the task descriptors.

3. BACKGROUND

A Markov Decision Process (MDP) \mathcal{M} is defined by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where \mathcal{S} is the set of states, \mathcal{A} is the set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Pi(\mathcal{S})$ is a transition function that maps the probability of moving to a new state given an action and the current state, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a reward function that gives the immediate reward of taking an action in a state, and $\gamma \in [0, 1)$ is the discount factor.

At each step, the agent is able to observe its current state, and must choose an action according to its *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$. The goal of an RL agent is to learn an *optimal policy* π^* that maximizes the long-term expected sum of discounted rewards. One way to learn the optimal policy is to learn the optimal action-value function $Q^*(s, a)$, which gives the expected reward for taking action a in state s , and following policy π^* after:

$$Q^*(s, a) = \mathcal{R}(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|s, a) \max_a Q^*(s', a')$$

Common temporal difference methods for learning the action-value function include Q-learning [24, 33] and Sarsa [23]. The optimal policy is then to choose $\arg \max_a Q^*(s, a)$ in each state. These temporal-difference methods are especially useful for problems with large and continuous state spaces which are challenging for approaches that directly try to learn the MDP. In this paper, our experiments were conducted using the Sarsa algorithm [23]. We used Sarsa as a simple representative base learning algorithm, though in principle our methodology is equally applicable to any RL algorithm that learns a value-function.

Since the policy consists of taking the action with the highest action-value, transferring a policy is equivalent to transferring the action-value function. For example, if the function $Q^*(s, a)$ is represented using a parameterized function approximator, then value function transfer is achieved by using the parameters learned in a source task to initialize the function’s parameters in the target task. In other words, the agent starts learning the target task while acting under the policy learned in the source task. When a good source task is available, value function transfer has been shown to speed up learning by initializing the policy to something better than random exploration [28].

Common measures used to evaluate the result of transfer typically compare the learning trajectory on the target task after transfer with the trajectory that was produced by learning the target task from scratch [29]. In this work, we used the *jumpstart* measure to quantify transferability. This measure looks at the difference between the initial performance after transfer and the initial performance without transfer. Let $R^{baseline} \in \mathbb{R}^K$ be the reward curve after learning the target task for K episodes such that $r_k^{baseline} \in \mathbb{R}$ is the expected reward after learning for k episodes. Similarly, let $R^{transfer} \in \mathbb{R}^K$ be the reward curve for learning the target task after transferring a policy from the source task. The jump start metric can then be defined by:

$$\text{jumpstart}(m) = \frac{\sum_{k=1}^m (r_k^{\text{transfer}} - r_k^{\text{baseline}})}{m}$$

The parameter m determines the size of the temporal window which is used to compute the jump start after the onset of training on the target task. Other measures of transferability include asymptotic performance improvement (or detriment) as well as time-to-threshold [29]. Due to the large-scale nature of our experiment, we used the jump start measure defined above since computing it requires a relatively small number of training episodes on the target task.

4. MODELING TASK TRANSFERABILITY

In this section, we introduce the proposed framework for modeling inter-task transferability. The proposed framework described here is independent of the RL and TL methods that were described in the previous section.

4.1 Notation and Problem Formulation

Let \mathcal{T} be the set of possible tasks. Let $\mathcal{T}_{\text{source}} \subset \mathcal{T}$ be a set of tasks for which the agent has learned a policy and let $\mathcal{T}_{\text{target}} \subset \mathcal{T}$ be another set of tasks that represents the set of target tasks to be learned by the agent. For each task $T_i \in \mathcal{T}$, let $F_i \in \mathbb{R}^n$ be a feature descriptor for the task that is known to the agent.

Given a target task $T_j \in \mathcal{T}_{\text{target}}$, the goal of the agent is to select a task $T_i \in \mathcal{T}_{\text{source}}$ such that T_i serves as an effective source for learning T_j . Thus, given a task pair, T_i and T_j , let $B(T_i, T_j) \in \mathbb{R}$ denote the benefit of transferring the policy learned in T_i to the task T_j , where $B(T_i, T_j) > 0$ indicates positive transfer, while $B(T_i, T_j) < 0$ indicates negative transfer. In this work, the transfer benefit is estimated using the jump-start measure defined in Section 3, though in principle, other measures can be appropriate as well.

We assume that for each pair of source tasks (T_i, T_j) such that $T_i, T_j \in \mathcal{T}_{\text{source}}$, the agent has a reliable estimate for $B(T_i, T_j)$. Next, we describe how the agent can use these estimates to predict the expected transfer benefit between tasks in $\mathcal{T}_{\text{source}}$ and tasks in $\mathcal{T}_{\text{target}}$.

4.2 Predicting the Benefit of Transfer

Here, the task of the agent is to learn a function which, given two arbitrary tasks T_i and T_j from \mathcal{T} , can predict whether T_i is a good source task for T_j . More specifically, the function should produce the estimate $\hat{B}(T_i, T_j)$, i.e., the expected benefit of transferring from T_i to T_j . Since $B(T_i, T_j) \in \mathbb{R}$, a natural solution for modeling the transferability between two tasks is to train a regression model.

Let $F_i = [f_{i,1}, f_{i,2}, \dots, f_{i,n}]$ and $F_j = [f_{j,1}, f_{j,2}, \dots, f_{j,n}]$ be the features for a pair of tasks (T_i, T_j) . To train a regression model on task pairs, a third feature vector is computed, X^{ij} , such that it captures some aspects of how the two feature vectors F_i and F_j are related. The feature vector X^{ij} was computed such that each element x_k^{ij} is defined by:

$$x_k^{ij} = \frac{f_{i,k} - f_{j,k}}{\max(f_{i,k}, \epsilon)}$$

where ϵ is a very small number to avoid divisions by 0. In other words, the vector represents the change along the n -dimensional features space relative to the feature values of the first task in the pair.¹The function that computes how two tasks are related was designed to be sensitive to the order of the tasks in the pair since preliminary experiments suggested that task transferability is not always symmetric.

Given this representation and a dataset $\{X^{ij}\}_{T_i, T_j \in \mathcal{T}_{\text{source}}}$, a regression model M is trained such that:

$$M(X^{ij}) \approx B(T_i, T_j)$$

Once trained on pairs of tasks from $\mathcal{T}_{\text{source}}$, the regression model is subsequently used to select source tasks for the tasks in $\mathcal{T}_{\text{target}}$. Given a target task T_j , the task $T_i \in \mathcal{T}_{\text{source}}$ that maximizes $M(X^{ij})$ is selected as the source task. Next, we describe the performance measures that were used to evaluate the framework proposed here.

4.3 Evaluation

4.3.1 Performance Measures

For each target $T_j \in \mathcal{T}_{\text{target}}$, the best possible source task is defined by:

$$T^* = \arg \max_{T_i \in \mathcal{T}_{\text{source}}} B(T_i, T_j)$$

Let T_i be the source task selected by the model. To compare the model's choice for a source task to the optimal source task, we define the *loss* as:

$$\text{loss}(T_i) = B(T^*, T_j) - B(T_i, T_j)$$

We also evaluated the ranking of source tasks induced by the regression model. For a given target task T_j , let $R_j = [T_{\{1\}}, T_{\{2\}}, \dots, T_{\{P\}}]$ be the ranked list of source task according to the learned regression model, i.e., $\hat{B}(T_{\{k\}}, T_j) \geq \hat{B}(T_{\{k+1\}}, T_j)$. For each position k in the ranking, let $\text{rel}_k = B(T_{\{k\}}, T_j)$ be a measure of the relevance of the result at that position. A common measure to evaluate the quality of a ranking is the Discounted Cumulative Gain (DCG) [11]:

$$\text{DCG}_p(R_j) = \text{rel}_1 + \sum_{k=2}^p \frac{\text{rel}_k}{\log_2(k)}$$

where $p \leq P$. The normalized DCG (NDCG) is computed by $\frac{\text{DCG}(R_j)}{\text{DCG}(R_j^{\text{best}})}$ where R_j^{best} is the true (i.e., best possible) ranking of source tasks. A normalized DCG of 1.0 would indicate a perfect ranking.

4.3.2 Baseline Comparison

For a baseline comparison, we consider the naive approach of selecting the most similar task according to the feature vectors used to describe the tasks. In other words, given target task T_j , the naive method would select the source task T_i that minimizes the squared distance between F_i and F_j , i.e., $L_2(F_i, F_j)$. The baseline approach does not perform any learning but nevertheless, we hypothesize that it will perform better than randomly selecting a source task.

5. EXPERIMENTS AND RESULTS

To evaluate the proposed framework, we conducted a large-scale experiment in the Ms. Pac-Man domain. The following subsections describe the domain, the experimental methodology, and the results of the experiment.

¹Other representations for the vector X^{ij} were explored as well, including raw difference (i.e., $f_{i,k} - f_{j,k}$) as well as ratio (i.e., $f_{i,k}/f_{j,k}$). Representations that captured the absolute or squared distance between F_i and F_j did not perform as well as they were not sensitive to the order of the tasks in the pair.

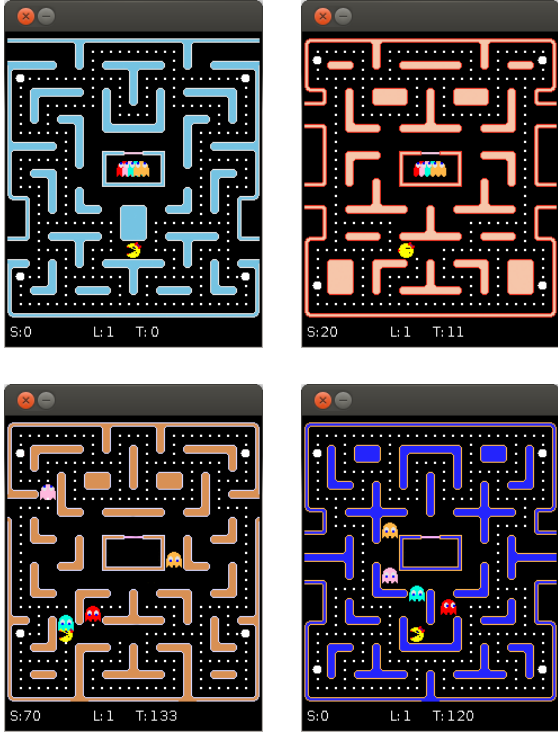


Figure 1: Screen shots of the game Ms. Pac-Man. In our experiments, the agent played 192 variations of the task, spanning 4 different mazes, shown above. The top-left image shows a sample configuration at the start of a game.

5.1 The Ms. Pac Man Domain

The framework for learning task transferability was evaluated using the Ms. Pac-Man domain, shown in Figure 1. The goal of the Ms. Pac-Man agent is to traverse a maze and earn points by eating edible items such as pills, while avoiding ghosts. The game typically starts with a large number of pills, four power pills located near each corner, and four ghosts that are initially placed in a lair that is inaccessible to Ms. Pac-Man. Shortly after the game starts, the ghosts leave their lair and may either chase Ms. Pac-Man or move about randomly. If a ghost catches Ms. Pac-Man, the game is over (we did not model the number of lives that are typically available to a human player). Whenever the agent eats one of the four power pills, the ghosts themselves become edible by Ms. Pac-Man for a short amount of time and their speed is reduced. If a ghost is eaten during that time, Ms. Pac-Man earns points and the ghost is sent back to the lair for a fixed amount of time, after which it starts to operate as normal. The agent’s action space consists of four actions, up, down, left, and right, though not every action is available in every state. Ms. Pac-Man eats pills, power pills and ghosts (when edible) whenever she gets within a small distance threshold of the object. Table 1 lists the rewards Ms. Pac-Man can get for different events in the game. The game ends when all the pills are gone, Ms. Pac-Man is eaten by a ghost, or 2000 time steps pass.²

²An original game play video (not associated with this work) could be found at http://youtu.be/c4n_6NFYvLY at the time of

Table 1: The Reward Structure of the Ms. Pac-Man Domain

Event	Reward (points)
Ms. Pac-Man eats a pill	10
Ms. Pac-Man eats a power pill	50
Ms. Pac-Man eats a ghost	200
Ms. Pac-Man eats an additional ghost while they are still edible	Apply a multiplier of 2 to the usual reward for each additional ghost that is eaten
Ms. Pac-Man is eaten by a ghost	Game Over

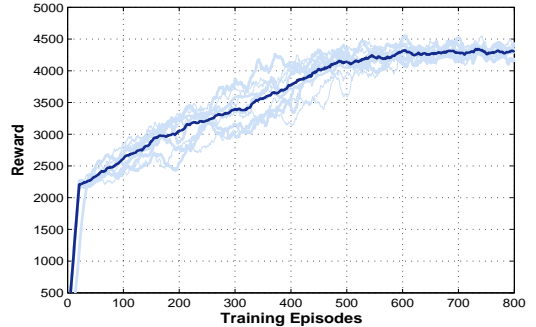


Figure 2: An example baseline test for one of the 192 tasks. The dark line indicates the reward averaged after 10 different runs (shown as the lighter lines), each starting with a different random seed. In this example, the policy converged after about 700 episodes.

In our experiments, we used the Ms. Pac-Man implementation described by Taylor *et al.* [26]. The raw state space of the game is highly dimensional and also specific to each maze, thus making it unsuitable for learning. Therefore, in practice the state space in the Ms. Pac-Man game is typically represented by a set of local features that are ego-centric with respect to Ms. Pac-Man’s position on the board (see [22, 5, 25] for a representative sample of approaches). In this work, we used 7 heavily-engineered features defined in [26]. These features calculate properties such as the safety of junctions, and scores for the amount of pills and ghosts that could potentially be eaten along a certain direction. The agent learned the game using the Sarsa RL algorithm [23]. The action-value function was represented by a simple linear function approximator over those 7 features.

5.2 Experimental Methodology

We generated 192 variations of the Ms. Pac-Man task by varying several parameters that dictate the dynamics of the game:

- **Maze:** each game was played on one of four different mazes, shown in Figure 1.
- **Number of ghosts:** the number of ghosts present in the game was varied from 1 to 4.
- **Ghost slowdown:** when Ms. Pac-Man eats a power pill, the ghosts become edible and their movement

writing this paper.

Table 2: The task features that were known to the agent

Feature	Description
<i>number-of-ghosts</i>	The number of ghosts in the game (1 to 4)
<i>ghost-slowdown</i>	The amount of speed reduction that the ghost undergoes when Ms. Pac-Man eats a power pill. The values ranged from 1 to 4.
<i>ghost-type</i>	The behavior of the ghosts. There are three possible values: <i>Random</i> , <i>Standard</i> , and <i>Chaser</i> .
<i>num-nodes</i>	The number of nodes in the maze graph
<i>num-pills</i>	The number of regular pills in the maze
<i>distance-to-ghost</i>	The distance between Ms. Pac-Man and the ghosts at the start of the game
<i>distance-power</i>	The average distance between power pills
<i>distance-lair</i>	The average distance between the ghost lair and the power pills
<i>junctions-between-junctions</i>	The average number of junctions that lie on the shortest path between any pair of junctions
<i>eccentricity</i>	The average eccentricity of nodes in the graph. The eccentricity for a node u is defined as $e(u) = \max\{d(u, v) : v \in V\}$ where d is the shortest-path function for a pair of nodes and V is the total set of nodes in the graph.
<i>eccentricity-junction</i>	The average eccentricity of junctions (i.e., nodes with more than 2 neighbors). The eccentricity for a junction node u is defined as $e(u) = \max\{d(u, v) : v \in J\}$ where $J \subset V$ is the set of nodes that are junctions.
<i>graph-diameter</i>	The diameter of the graph is defined as $diam(G) = \max\{e(u) u \in V\}$.
<i>num-nodes-d2</i>	Number of nodes with 2 neighbors
<i>num-nodes-d3</i>	Number of nodes with 3 neighbors
<i>num-nodes-d4</i>	Number of nodes with 4 neighbors

speed is reduced. The *ghost-slowdown* parameter specified the amount of speed reduction and varied from 1 to 4, in increments of 1. When the Ghost slowdown is set to n , then the ghosts remain stationary every n^{th} game step when they are edible. Thus, a higher value makes the ghosts move faster, while a value of 1 makes them stop moving completely.

- **Ghost type:** the ghosts behaved according to one of three different modes: *Standard*, *Random*, and *Chaser*

The three different ghost behaviors are as follows: (1) *Standard ghosts* chase Ms. Pac-Man 80% of the time and move randomly the other 20%. When Ms. Pac-Man eats a power pill, the ghosts start moving away from the agent and eventually revert to their original behavior once they are no longer edible; (2) *Random ghosts* choose a random direction when reaching a junction 100% of the time. This

makes it easier for Ms. Pac-Man to avoid them, but harder for Ms. Pac-Man to catch ghosts after eating a power pill. (3) *Chaser ghosts* have the same behavior as the *Standard ghosts* when inedible. However, after Ms. Pac-Man eats a power pill, they continue moving towards Ms. Pac-Man instead of fleeing. This makes it easy for Ms. Pac-Man to learn to eat ghosts (sometimes also too easy, since Ms. Pac-Man can learn to just stay in place and let the ghosts come to it, which does not transfer well to the normal setting).

Varying the four parameters resulted in $4 \times 4 \times 4 \times 3 = 192$ versions of the game. These 192 tasks constituted the full set of tasks \mathcal{T} . To compute transferability for all pairs of tasks, the agent first learned to play each task from scratch for 2,500 episodes (the number of total episodes was chosen such that the agent’s policy converged on each of the 192 tasks). Each episode consisted of playing a full game of Ms. Pac-Man. After each episode, the policy was frozen and the agent played an additional 10 games to compute a reliable estimate for the expected reward at each point during training. This procedure was repeated 10 times for each task in order to account for the stochastic nature of the domain. Thus, the agent played a total of $192 \times 2,500 \times (1 + 10) \times 10 = 50,800,000$ games to compute the baseline performance reward curves. Figure 2 shows an example baseline test for one of the 192 tasks. The bold line indicates the average reward curve from the 10 different runs.

Once the baseline curves were computed, the benefit of transfer was estimated for all task pairs. To do so, for each of the 36,672 pairs of tasks (T_i, T_j) in \mathcal{T} , the agent learned on task T_j for 30 episodes starting with the policy learned on task T_i (i.e., the agent transferred the policy from source task T_i to target task T_j). This process was repeated 10 times for each pair, such that in each run, a different one of the 10 policies computed during the baseline run was used as a starting point. Thus the agent played $36,672 \times 30 \times (1 + 10) \times 10 = 120,101,760$ games. The average reward with transfer and the average baseline reward over the first 30 episodes were then used to compute the jump start measure. The jump start measure requires a parameter m that denotes the size of the temporal window (in terms of number of episodes) to be used when averaging the rewards (see Section 3). We computed the jump start measure for $m = 1, 3, 5, 10, 15$, and the maximum, 30.

All told, to compute both the baseline reward curves as well as the transfer reward curves, the agent had to play over 170 million games. This type of an experiment would be next to impossible on a single computer and therefore, we used our department’s Condor Cluster system [10]. A learning episode typically took about 0.5 – 0.75 seconds, though this duration could vary depending on the cluster machine being used. Based on logged data, the experiment took over 2,300 hours of compute time spread over 192 individual machines. We believe that this is the largest computational experiment in transfer learning to date.

The framework for learning task transferability proposed in Section 4.2 requires that the agent has access to a real valued feature vectors that describes each task. Table 2 shows the task features that were used in our experiments. All of the features, except for *ghost-type*, are numeric. The *ghost-type* feature was originally nominal and therefore was converted into 3 different binary features, one for each type of ghost behavior. Thus, $F_i \in \mathbb{R}^{17}$. The features that were used to describe the tasks corresponded to the parameters used to

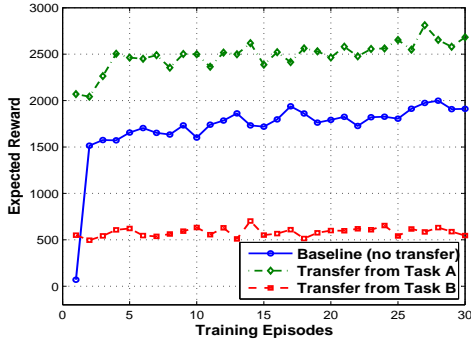


Figure 3: An example transfer result for a given target task and two potential source tasks. Task A is clearly the better source task, resulting in a large positive transfer.

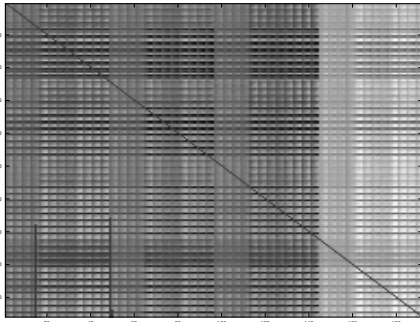


Figure 4: An example transferability matrix computed for each pair of the 192 tasks considered in our experiments. In this matrix, the entry at i, j amounts to the resulting $jumpstart(30)$ measure after transferring the policy learned on task T_i to task T_j . Light values indicate high jump start while dark values indicate low (possibly negative) jump start.

generate the tasks, as well as graph-based features induced by the maze in each task. The features were not specifically selected or tuned to maximize performance. The graph-based features included domain specific attributes (e.g., the distance between Ms. Pac-Man’s starting position and the Ghosts’ lair) as well as general graph-based features such as *eccentricity* and a histogram of the nodes’ degrees (the last three features in the Table 2).

In our experiments, we explored two different implementations for the regression model M described in section 4.2: 1) Linear Regression, and 2) M5 Model trees [20]. Linear Regression was selected due to its simplicity, while the M5 Model tree was selected as it is able to handle non-linear problems. Both implementations can be found in the WEKA machine learning library [34]. The WEKA implementation uses a modified version of the original tree induction algorithm, called M5P [32] which added pruning as a part of the training stage.³

5.3 Results

³All source code and implementations that were used to conduct this experiment will be made publicly available upon publication.

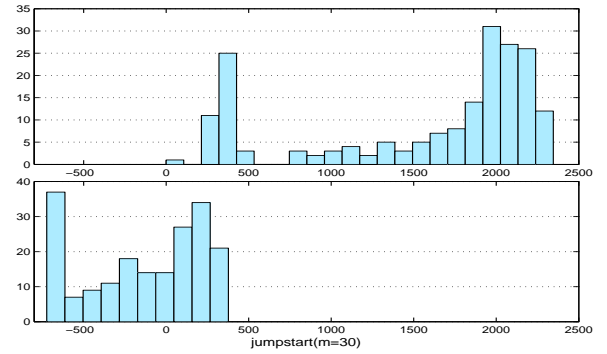


Figure 5: Histograms of the jump start measures for two randomly chosen target tasks (i.e., a histogram over the values in a given column of the transferability matrix). For the first target task (top histogram), virtually all source task result in positive transfer, while for the second, there are a large number of source tasks that induce negative transfer.

5.3.1 The Transferability Matrix

Figure 3 shows an example transfer result for a target task and two different source tasks. In this case, transferring the policy from one of the source task to the target task results in positive transfer, while the other source task induces negative transfer. Figure 4 shows the whole transferability matrix computed for the set of 192 tasks considered in our experiments. In this example, each entry contains the expected benefit of transfer according to the $jumpstart(30)$ measure for each pair of tasks (in other words, the jump start was computed over the first 30 training episodes on the target task). White values indicate high jump start while black values indicate low (possibly negative) jump start.

The order of the columns and rows of the matrix is not random but rather, the entries are sorted first according to the *maze*, then *ghost-type*, then *ghost-slowdown*, and then finally, *number-of-ghosts*. The last 1/4 set of columns in the matrix appear brighter than the rest because those tasks were much more likely to benefit from transfer. These tasks corresponded to tasks with the fourth maze, which proved to be much more difficult for the agent than the other three mazes. The grid-like pattern shows that transfer is not random and hence, we hypothesized that the parameters that define the tasks may be useful in predicting the benefit of transfer across tasks.

Figure 5 shows a histogram of the jump start measures for two randomly chosen target tasks (i.e., a histogram over the values in a given column of the transferability matrix). Even though the shapes of the histograms are similar, one of the target tasks is much more likely to benefit from transfer. For the first target task (top histogram), virtually all source tasks result in positive transfer. For the second target task, however, there are a large number of source tasks that induce negative transfer, which further motivates the need for effective source task selection.

5.3.2 Regression Model Performance

The performance of the regression model used to estimate transferability was evaluated using 10-fold cross validation at the task level. In other words, during each run, the tasks were split into 10 sets such that 9 of these formed the set

Table 3: Regression Model Performance measured by Correlation Coefficient

Transferability Measure	Linear Regression	M5P Model Tree
$jumpstart(m=1)$	0.54	0.74
$jumpstart(m=3)$	0.64	0.85
$jumpstart(m=5)$	0.65	0.87
$jumpstart(m=10)$	0.66	0.87
$jumpstart(m=15)$	0.65	0.86
$jumpstart(m=30)$	0.61	0.83

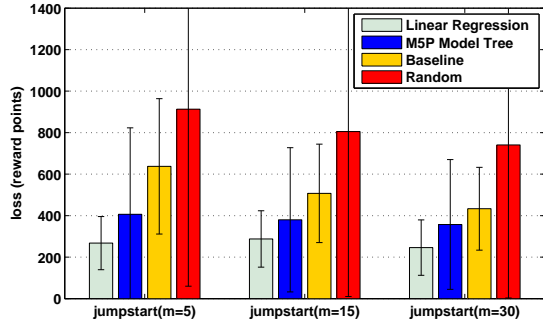


Figure 6: Source Task Selection *loss* for three transferability measures. The two regression models were compared with the baseline source task selection model and with random source task selection.

\mathcal{T}_{source} while the remaining fold was considered as the set of target tasks \mathcal{T}_{target} . The regression model was trained on all pairs of tasks (T_i, T_j) such that $T_i, T_j \in \mathcal{T}_{source}$ and then tested on all pairs of tasks induced by the cross product of $\mathcal{T}_{source} \times \mathcal{T}_{target}$.

Table 3 shows the performance of the two regression algorithms that were used to predict the $jumpstart(m)$ measure for different values of m , the size of the temporal window used to compute the jump start. The results are reported in terms of the Correlation Coefficient (CC) between the actual and the predicted values. These results show that the difficulty of modeling task transferability depends on the measures used to estimate the benefit of transfer. For example, modeling the jump start after just 1 training episode on the target task is more difficult than modeling the jump start after 10 episodes on the target task. Overall, the CCs are high enough that we expect the ranking induced by the regression models to be useful for source task selection.

5.3.3 Source Task Ranking and Selection

Next, the framework for source task selection proposed in this paper was evaluated in terms of the expected *loss*, i.e., if the agent selects the source task that maximizes the expected transferability according to the regression model, how much worse does it do compared to selecting the optimal source task that it has already learned. Figure 6 shows the result of this test for two different regression algorithms, as well as the baseline approach. In addition, as a sanity check we computed the loss when randomly selecting a source task.

As we expected, the baseline approach which selects a source task based on task similarity in the task feature space

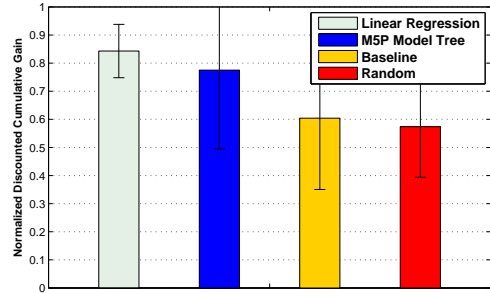


Figure 7: Evaluation of source task ranking using the learned regression model and the baseline case-based reasoning approach. The ranking was evaluated using the Normalized Discounted Cumulative Gain (DCG_p) and the $jumpstart(m=5)$ measure (the results were similar for the remaining values of m used in this study). The value for p , the number of elements to be considered in the ranking (starting at position 1) was set to 20.

performs better than randomly selecting a source task. Furthermore, the proposed method for learning task transferability substantially outperforms the baseline approach. While the Linear Regression (LR) model performed worse in terms of Correlation Coefficient when compared to the M5P Tree (M5P), the top source task selected when using LR tended to be a better source task than the one selected by M5P. The results so far were computed when approximately 172 tasks (i.e., 9 out of 10 folds) were available for training the regression model. An important question is whether performance would suffer as the training set becomes smaller. To obtain an answer, the number of tasks used to train the model was varied from 2 to 30 and we found that the expected loss converges after about 20 tasks (i.e., 400 pairs) are available for learning the regression.

The quality of the rankings were further evaluated using the Normalized Discounted Cumulative Gain measure. The results of this test are shown in Figure 7. Overall, LR performed the best. These results conclusively show that inter-task transferability can be learned even without samples or models of the target task. In particular, when faced with a new target task, a single good source task can be selected for transfer. These results naturally raise the question of whether it is possible to chain together multiple such source tasks sequentially to do even better. We examine that question next.

5.3.4 Multi-stage Transfer

In this section, we explore whether we can chain together a sequence of tasks $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_{target}$, such that learning T_1 makes it “easier” to learn T_2 , which makes it “easier” to learn T_3 , and so on. For simplicity, consider two stage transfer: we are looking for source tasks T_1 and T_2 such that transferring from $T_1 \rightarrow T_2 \rightarrow T_{target}$ gives better performance than training directly on T_{target} or any of the one-stage transfers $T_1 \rightarrow T_{target}$ and $T_2 \rightarrow T_{target}$.

Candidates for the tasks T_1 and T_2 can be determined recursively using the transferability matrix. We simply look at the column corresponding to the target task, and select the row (i.e. source task) that provides the best transfer.

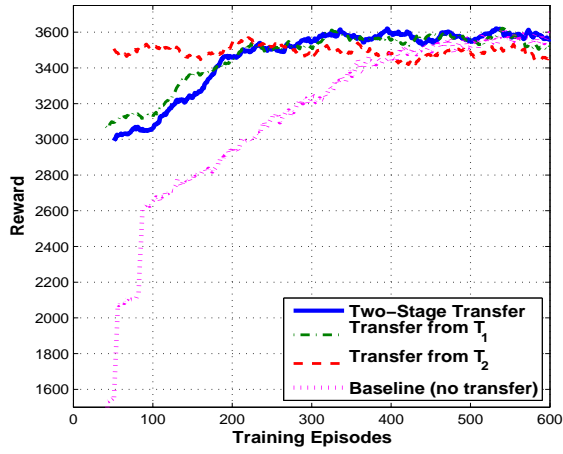


Figure 8: Performance on a target task using one and two-stage transfer. Note that the transfer curves are offset to reflect time spent training in their source tasks. In this example, all methods of transfer result in jump start but there is no benefit of two-stage transfer relative to single-stage transfer.

The selected task then becomes the column for the next recursive stage.

A key question that we have not addressed so far is how to decide how many episodes to spend on each source task. Training on each of the subtasks T_1 and T_2 until convergence before transferring to T_{target} would be equivalent to just training on T_2 until convergence and performing single stage transfer to T_{target} . Therefore, in this preliminary test, we used a heuristic approach based on the intuition that an agent should train on a source task until additional training does not improve performance on the target.⁴

We hand-selected several of the more challenging tasks to serve as T_{target} . The results for one such target task are shown in Figure 8. All methods of transfer resulted in a jump start, but there was no benefit to using two stage transfer over single stage transfer. The results were similar for the other target tasks and overall, we were not able to find a two-stage transfer that was significantly better than its one-stage counterpart. Our hypothesis is that value function transfer is not suitable for two-stage transfer, since single stage transfer already initializes the policy in some area of the search space, and adding more stages does not noticeably refine this area. We leave for future work whether alternative RL and TL methods would facilitate finding a better two-stage transfer result.

⁴ We define the target performance to be the total reward accumulated by the agent on the target task, for a fixed number of episodes (i.e. the area under the learning curve). Let A_{base} be the total reward accumulated by training directly on the target task without using transfer, and let $A_{transfer}^x$ be the total reward accumulated on the target task after training on the source task for x episodes, and using value function transfer. We used an incremental approach where the agent trained on the source task for 10 episodes, and used this to compute $A_{transfer}^x$. If the difference ($A_{transfer}^x - A_{base}$) was positive and increased, the agent trained on the source for 10 more episodes. This process was repeated until the difference no longer increased, at which point training on the source task was halted.

6. CONCLUSION AND FUTURE WORK

This paper proposed a framework for source task selection in settings where neither samples from the target task, nor a model of the task, are available to the learning agent. Instead, the agent used task descriptors (i.e., a low-dimensional feature vector describing some aspects of the task) to learn the expected benefit of transfer, i.e., transferability, between source tasks and target tasks. The framework was evaluated using a large-scale experiment in which the agent learned to play 192 variations of the Ms. Pac-Man game. To test our framework, the agent played over 170 million games, making this, to the best of our knowledge, the largest computational experiment in transfer learning conducted so far. Our results show that an agent can indeed learn to predict the transferability for an arbitrary pair of source-target tasks, provided training pairs for which the benefit (or detriment) of transfer is known. The learned transferability model was then used to effectively select relevant source tasks that improve the agent’s learning performance on a given target task.

There are several limitations and open questions that need to be considered for future work. While efficiency was not addressed in this paper, in practice generating a large set of data using every source-target pair is expensive. We found that only a small fraction of the source tasks are needed for the source task selection loss to converge and we believe that simple active learning frameworks can further reduce the number of task pairs needed to learn the transferability model. In our experiments, the agent learned the source tasks for the maximum amount of allowed time but we have also found that policies can successfully be transferred even when there is only limited exploration in the source task. Therefore, efficiency may also be improved if the agent can autonomously decide when to stop learning a source task and transfer the policy to a target task.

Evaluating efficiency in a setting like this poses additional challenges as it depends strongly on the number of potential target tasks to be solved by the agent in the future. To get a strong transfer result, the time (e.g., number of episodes) spent training on source tasks needs to be taken into account when comparing the performance with training without transfer. When there is only one target task, the amount of learning spent on source tasks is bounded by the amount of time required to learn the target task from scratch. Most recent frameworks for evaluating transfer assume that there is indeed only one target task [29] and therefore, there is a need to identify good measures for quantifying strong transfer results in the setting where the set of target tasks is large and potentially larger than the set of source tasks.

One aspect of the framework that makes it applicable in a wide variety of settings is that it is agnostic with respect to the reinforcement learning algorithm or transferring learning method being used. At the same time, this property limits the potential for deeper theoretical analysis. Another open question is whether a similar methodology can be used to discover and subsequently model two-stage transfer, i.e., situations in which the agent learns multiple source tasks in a precise order such that learning on the target task is improved. A follow-up result from this study is that two-stage transfer sequences are rare or perhaps non-existent in the task space we considered. Future work will examine whether this is a limitation of the domain, or a limitation of the transfer learning method (i.e., value-function transfer).

7. REFERENCES

- [1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
- [2] H. B. Ammar, E. Eaton, M. E. Taylor, D. C. Mocanu, K. Driessens, G. Weiss, and K. Tuyls. An automated measure of MDP similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.
- [3] H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems, 2012.
- [4] J. Annett and J. Sparrow. Transfer of training: a review of research and practical implications. *Programmed Learning and Educational Technology*, 22(2):116–124, 1985.
- [5] P. Burrow and S. M. Lucas. Evolution versus temporal difference learning for learning to play Ms. Pac-Man. In *IEEE Symposium on Computational Intelligence and Games*, pages 53–60. IEEE, 2009.
- [6] K. C. Chatzidimitriou, I. Partalas, P. A. Mitkas, and I. Vlahavas. Transferring evolved reservoir features in reinforcement learning tasks. In *Recent Advances in Reinforcement Learning*, pages 213–224. Springer, 2012.
- [7] A. Fachantidis, I. Partalas, G. Tsoumakas, and I. Vlahavas. Transferring task models in reinforcement learning agents. *Neurocomputing*, 107:23–32, 2013.
- [8] F. Fernández, J. García, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.
- [9] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM, 2006.
- [10] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.
- [11] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.
- [12] N. Kornell, L. K. Son, and H. S. Terrace. Transfer of metacognitive skills and hint seeking in monkeys. *Psychological Science*, 18(1):64–71, 2007.
- [13] A. Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.
- [14] A. Lazaric and M. Restelli. Transfer from multiple MDPs. In *Advances in Neural Information Processing Systems*, pages 1746–1754, 2011.
- [15] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551. ACM, 2008.
- [16] R. Lopez De Mantaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T. Cox, K. Forbus, et al. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(03):215–240, 2005.
- [17] G. B. Nallan, M.-B. Brown, C. Edmonds, V. Gillham, K. Kowalewski, and J. S. Miller. Transfer effects in feature-positive and feature-negative learning by adult humans. *The American Journal of Psychology*, pages 417–429, 1981.
- [18] T. Nguyen, T. Silander, and T. Y. Leong. Transferring expectations in model-based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2555–2563, 2012.
- [19] T. J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. In *Technical Report*. University of Massachusetts, 1999.
- [20] R. J. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.
- [21] J. Ramon, K. Driessens, and T. Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Machine Learning: ECML 2007*, pages 699–707. Springer, 2007.
- [22] D. Robles and S. M. Lucas. A simple tree search method for playing Ms. Pac-Man. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 249–255. IEEE, 2009.
- [23] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.
- [24] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.
- [25] I. Szita and A. Lőrincz. Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *J. Artif. Intell. Res. (JAIR)*, 30:659–684, 2007.
- [26] M. E. Taylor, N. Carboni, A. Fachantidis, I. Vlahavas, and L. Torrey. Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1):45–63, 2014.
- [27] M. E. Taylor, N. K. Jong, and P. Stone. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 488–505. Springer, 2008.
- [28] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59. ACM, 2005.
- [29] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.
- [30] M. E. Taylor, P. Stone, and Y. Liu. Value functions for rl-based behavior transfer: A comparative study. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 880. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.
- [31] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 37. ACM, 2007.
- [32] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.
- [33] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.
- [34] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufman, San Francisco, 2nd edition, 2005.