To Appear In Proceedings of the Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 2016.

Object-Model Transfer in the General Video Game Domain

Alexander Braylan, Risto Miikkulainen
Department of Computer Science, The University of Texas at Austin

Abstract

A transfer learning approach is presented to address the challenge of training video game agents with limited data. The approach decomposes games into objects, learns object models, and transfers models from known games to unfamiliar games to guide learning. Experiments show that the approach improves prediction accuracy over a comparable control, leading to more efficient exploration. Training of game agents is thus accelerated by transferring object models from previously learned games.

Introduction

Reinforcement learning methods have achieved high levels of performance across a broad spectrum of games but often require large amounts of training data (Hausknecht et al. 2014; Mnih et al. 2015). Learning *forward models* of an agent's environment can reduce the amount of required training and improve overall performance and flexibility. A forward model is a function that predicts the future state of an environment from its current state. However, when the data used to train a model is sparse, noisy, or high-dimensional, the model is at risk of suffering from *generalization error* in predictions made outside of the data seen during training. For example, the first few frames of a new game an agent observes may not be sufficient to inform the agent about how the game will behave later on.

One field of research that may help address the problem of generalization error is *transfer learning* (Taylor and Stone 2009; Pan and Yang 2010), the reuse of knowledge and skills learned in *source* tasks to accelerate and improve performance in a different *target* task. Applied to video games, the idea is that an agent with ample experience playing various source games can learn a better model of a new target game by transferring and combining knowledge from the source games. This paper considers the role of this combined transferred knowledge in forming an *inductive bias* — an assumption that constrains the space of possible models, and a way to guard against generalization error (Mitchell 1980).

The first challenge in transfer learning is mapping between variables in the source and target environments. A second challenge is integrating and applying the transferred knowledge. This paper responds to both questions in the context of general video game playing. The key step taken toward the first challenge is to decompose game environments into collections of objects. In an object-oriented formulation of a game environment, objects belong to *object classes* exhibiting similar behaviors across a wide variety of games (Diuk, Cohen, and Littman 2008). The variables of an object class are interpreted in the same way regardless of the game, simplifying the question of variable mapping. The approach of this paper toward the second challenge is to construct *transfer ensembles* out of models transferred from source games and *scratch* models newly trained for the target game. Each transfer ensemble uses a weighted average of predictions from its constituent models to predict the behavior of a target object. The weights are calculated based on how well each constituent model describes the data observed for the target object class. An important final step is retraining the source models to better fit target data.

Experimental results show that agents using transfer ensembles as models of object classes generalize better than using scratch models. After observing small quantities of *in-sample* training data, transfer ensembles achieve greater accuracy than scratch models when predicting the behavior of objects in subsequent *out-of-sample* test data. Agents that use learned models to inform their actions in an exploration task are shown to perform better when using the transfer learning approach than when learning from scratch.

Altogether, the conclusion is that decomposing environments into objects and transferring object models across games is a promising approach for learning to play video games from small amounts of experience.

Background

This paper draws from research in general video game playing, model-based reinforcement learning, and transfer learning. Each is a broad field of research, so this section will review the topics most relevant to this work.

General Video Game AI

General Video Game AI (GVG-AI) is an open-source project that facilitates artificial intelligence research in general video game playing (Schaul 2013; Perez-Liebana et al. 2016). The GVG-AI project provides a framework for agents to interact with games and includes 60 games hand-coded in the Video Game Description Language (Ebner et al. 2013). The games are similar to games from the Atari 2600 console and other popular video games, including games inspired by Space Invaders, Frogger, Zelda, Lemmings, Seaquest, Sokoban, Dig Dug, Pacman, Star Fox, Plants vs. Zombies, among many others. Borrowing from several genres of video games presents agents with a wide diversity of challenges.

Additionally, there are a few advantages to using GVG-AI over an Atari emulator. GVG-AI objects can exhibit stochastic behavior. For Atari, stochasticity can so far only be added artificially to the initial game state or to the actions input by the player (Hausknecht and Stone 2015). Furthermore, each game in GVG-AI includes several levels with different initial conditions. These features allow for straightforward out-of-sample testing, crucial for measuring generalization error. Therefore, the experiments in this paper use the GVG-AI framework and games.

Model-Based Reinforcement Learning

Reinforcement learning problems challenge agents to take actions in response to observations of an environment in order to accumulate rewards over time (Sutton and Barto 1998). In the most common case, the environment is formally a Markov decision process (MDP), which consists of a set of states, actions, and a probabilistic transition function. This function governs the distribution of subsequent states given every current state and action. *Model-based* reinforcement learning methods rely on an estimate of the transition function. In contrast to *model-free* methods, they have rich representations of the environmental dynamics. Such representations yield various benefits: data efficiency, better planning and exploration, and robustness against changes in the reward function (Atkeson and Santamaria 1997; Asmuth and Littman 2011). Most approaches to model learning for high-dimensional environments use *factored state* representations, learning approximate transition functions on a manageable number of features of the state space.

Factored-State Model Learning for Video Games Because video games are high-dimensional environments, the only approaches that learn models of video games use factored state representations. One approach to learning models of Atari games by Bellemare et al. (Bellemare, Veness, and Bowling 2013) predicts patches of pixels from neighboring patches using a compression algorithm, taking advantage of the tendency of game objects to depend only on nearby objects. Alternatively, a deep learning approach by Oh et al. (Oh et al. 2015) uses convolutional neural networks on pixel inputs from the entire game screen to predict future pixel values.

While some research exists on learning factored models to make the most out of few training samples (Degris, Sigaud, and Wuillemin 2006; Hester and Stone 2013; Jong and Stone 2007), both papers on model learning for video games focus on the scalability and power of the models rather than on sample efficiency. The neural networks used by Oh et al. trained on 500,000 frames per game, while the models in Bellemare et al. trained on 10 million frames. This paper investigates training on as few as 10 frames.

Object-Oriented Markov Decision Process An *Object-Oriented Markov Decision Process* (OO-MDP) is a factorization that exploits the object-oriented nature of many reinforcement learning problems by re-framing the environment as a collection of objects (Diuk, Cohen, and Littman 2008). Compared to the high dimensionality of the full game state, the object-oriented environment is represented only by the relatively few *attributes* of each object. These attributes include the object's internal state variables and variables representing relations with other objects. For example, geographic relationships are encoded by first-order propositions $on(o_1, o_2), touch^N(o_1, o_2), touch^E(o_1, o_2), etc.$ Each object belongs to an *object class*; all instances of the same object class are assumed to follow the same transition function, thus only one model is needed for each object class. The assumption that many of these object classes are similar over multiple games is one motivation for choosing the object-oriented factorization for transfer learning.

Transfer Learning

The transfer of models between tasks is related to the theory behind choosing a good inductive bias. When a learner can sample from multiple related tasks, an inductive bias that works well for several of those tasks can be expected to work well in other related tasks (Baxter 2000). For example, learning multiple related tasks at the same time with some shared learning parameters can be better than learning each task individually (Caruana 1997). Similarly, source knowledge can inform the selection of inductive bias in target tasks. Some such approaches involve the use of an ensemble model, a weighted combination of source models where the weights depend on how well each source model predicts the target data (Dai et al. 2007; Gao et al. 2008; Eaton and DesJardins 2009). This is the type of approach taken in this paper.

Approach

This section first presents a method for learning a forward model of the transition function of each object class from scratch in GVG-AI games. It then presents a transfer learning method for reusing scratch models to learn models more quickly for new objects in target games.

Learning Object Models from Scratch

A forward model F_j of an object class j is a function that generates a prediction $\hat{S}^i_t = F_j(S^i_{t-1})$ for the state S^i_t of object instance i (belonging to object class j), given its previous state S^i_{t-1}. The state S^i_{t-1} includes the object instance's internal variables as well as global state variables such as the player action A_{t-1}. Learning a model involves using observed data to alter the parameters of the model so as to improve its prediction accuracy. The three major decisions for specifying a model learner are on the model variables, the model's functional form, and the learning algorithm, described in detail in the rest of this subsection.

Model Variables: Object Class Attributes In addition to a visual display, GVG-AI reports a list of all objects in the

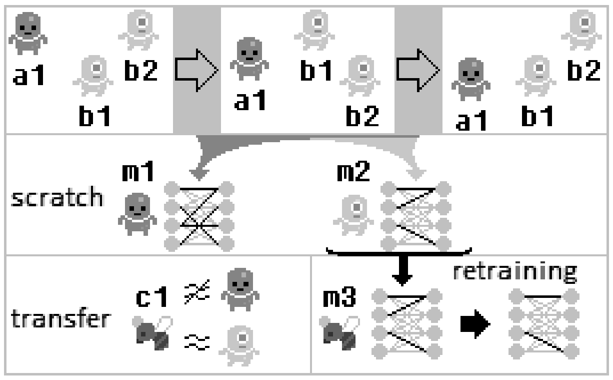

Figure 1: Simplified depiction of scratch and transfer approaches. The top row shows sequential game transitions containing object instance a1 of the dark creature class and b1 and b2 of the light creature class. The middle row shows data from each class being used to train their respective scratch models, m1 and m2. In the bottom row, observations of object instance c1 of the bumblebee class are used to train a transfer model m3 by adopting model m2 from the most similar class (the light creature) and retraining it. The scratch models serve both as stand-alone models to predict transitions and as building blocks for transfer models.

game at each frame. For each of these objects, it discloses the occupied *tile* – or x and y position – as well as a token representing the object's class used for grouping different instances of the same class within a game.

The above position and object class information are sufficient to extract a set of attributes that capture most of the observable object behaviors in GVG-AI games. The most common behaviors encountered include deterministic, stochastic, and player-controlled movement; death; and spawning of other objects on the same tile. Spawning is a novel extension of the OO-MDP formulation to capture the effect of a new object instance appearing in a game.

The predicted next state of an object instance g consists of the following *output* attributes:

- Directional movement (North/South/East/West) at time t, $M_t = \{m_t^N, m_t^S, m_t^E, m_t^W\}$;
- Whether the object is alive and on screen, e_t; and
- New spawns of other objects on the tile of this object, $N_t = \{n_t^{C(i)} : \text{spawn}_t(i), \text{on}_t(g,i)\}$.

To clarify how the spawn attributes are managed, the proposition $\text{spawn}_t(i)$ denotes whether an object i is a *spawn* – a newly observed object instance in a game – at time t. Every spawn observation is recorded as $n_t^{C(i)} = 1$ for every other object on the same tile as the spawn, with $C(i)$ denoting the object class of object i. For example, when a new bomb object appears on the same tile as an alien object, that alien object takes a value of 1 for the attribute n^{bomb}.

In addition to the above output attributes, the following *input* attributes account for factors upon which the predicted behaviors are conditioned:

- Directional movement at time $t-1$, $M_{t-1} = \{m_{t-1}^N, m_{t-1}^S, m_{t-1}^E, m_{t-1}^W\}$;
- Other objects touching the object $H_{t-1} = \{h_{t-1}^{D,C(i)} : \text{touch}_{t-1}^D(g,i)\}, D \in \{N, S, E, W, \text{ON}\}$;
- Whether the object was facing in the direction of its last movement, f_{t-1}; and
- Action input by the player, $A_{t-1} = \{a_{t-1}^{\text{NIL}}, a_{t-1}^{\text{UP}}, a_{t-1}^{\text{DOWN}}, a_{t-1}^{\text{LEFT}}, a_{t-1}^{\text{RIGHT}}, a_{t-1}^{\text{SHOOT}}\}$.

For example, whenever an object instance is adjacent or overlapping another object instance of a different class, its h attribute corresponding to the other object's class and relative position takes a value of 1.

In addition to object class models, termination models can be learned for predicting whether the game is won or lost at each frame from some global game variables. Termination models are not deeply explored in this paper but are helpful for experiments involving action selection. The two termination models, $P(\text{WIN}|X)$ and $P(\text{LOSE}|X)$, are conditioned on the following inputs:

- Existence of at least one live object instance of each class in the game, $X_{t-1} = \{x_{t-1}^j : \text{exists}_{t-1}(j)\}$.

Each x^j represents whether any instance of the game's object class j exists at all at the given time. This input is used because termination often coincides with the total disappearance of one of the game's object classes.

Functional Form and Learning Algorithm In a factored state model, the prediction $\hat{S}_t$ of the next state of an object is decomposed into predictions for each output variable $\hat{s}_t^k \in \hat{S}_t$. All of the specified object variables take values of either 0 or 1. The values of variables not observed by an object are 0 by default. The factored-state model produces a prediction between 0 and 1 for each output variable of an object instance. Each prediction represents the probability of the output variable taking a value of 1 given the observed input values. A logistic regression model is trained for each output variable of each object class using observations of all instances of the object class in a game, depicted in the first two thirds of Figure 1.

A logistic regression output is a sigmoidal function of its weighted inputs, taking the form $\hat{s}_t^k \sim \frac{1}{1+e^{-WS_{t-1}}}$. The weight vector W consists of coefficients to the input variables and an intercept term which are trained through gradient descent. The gradient descent algorithm iteratively decreases a cost computed from the values of observed S_t and predicted $\hat{S}_t$. Weights are gradually changed in the direction of the partial derivative of this cost with respect to the weights so as to reduce the cost. The cross entropy error $E_t = -\sum_k (s_t^k \ln \hat{s}_t^k + (1-s_t^k)\ln(1-\hat{s}_t^k))$ is used as the cost function to ensure convergence near a global minimum. During gradient descent training, data points are presented in random order at each iteration to avoid biasing the learned model.

Object Model Transfer

The transfer learning approach in this paper relies on a simple and intuitive assumption: Some object classes encountered in a target should behave similarly to other object classes previously encountered in sources. Therefore, when reasoning about an unknown target object, knowledge of previously seen similar source objects can help constrain and shape the distribution of predictions for the target object's behavior. The measure of similarity depends on what is known about the target object, what is known about the source objects, and the ability to establish relationships between attributes of the different objects. This assumption forms an inductive bias which should help trained models generalize better to unseen target data.

The bottom third of Figure 1 is a sketch of how this approach uses source models to train transfer models. The following example serves to illustrate more tangibly how object class models can be transferred.

An Illustrative Example of Walls and Avatars In the game Chase, the player-controlled avatar must chase goats by moving freely in four directions except when blocked by wall objects. These movement rules for the avatar are common in several other games, such as the game Escape. In Escape, the avatar is again moved in four directions by the player and is blocked by walls. The Escape avatar can also push away box objects and disappear through hole objects. A transfer-learning agent who has played many games of Chase but has only seen a few frames of Escape should be able to reuse specific knowledge from Chase to make more accurate predictions about Escape than a total novice.

Upon encountering a wall for the first time in Escape, a novice agent with no Chase experience would have low certainty on the outcome of an attempt to move the avatar into the wall. In contrast, a transfer learning agent could notice some similar behavior between the Chase and Escape avatars — such as how the player inputs move both of them in similar directions — and reason that the interaction with the wall is also likely to be the same in both games.

The transfer learning method presented in this paper produces models that make predictions as described above when the source is Chase and the target is Escape. However, transferring object class models is not always so simple for all sources and targets. The following subsections explain additional challenges encountered and how they are addressed.

Source Knowledge Selection One objective for a transfer learning algorithm is an ability to choose automatically what knowledge to transfer from a potentially large pool of source models. Transferred knowledge may harm rather than improve performance in the target task, an outcome called *negative transfer* (Taylor and Stone 2009). In order to reduce negative transfer, the transfer learning algorithm may select its sources according to their expected contribution to performance.

This paper uses a measure of one-frame forward prediction accuracy to evaluate learned models, both for guiding source selection and for overall evaluation. Prediction accuracy of an object class model F on object transition data $\mathbb{S} = \{S_1, S_2, ...S_T\}$ is calculated as $\text{accuracy}(F, \mathbb{S}) = \frac{1}{T}\sum_{t=2}^{T} \text{equal}(S^t, F(S^{t-1}))$, where $\text{equal}(S, \hat{S}) = 1$ if for all output attributes k, $s_k = \text{round}(\hat{s}_k)$, and 0 otherwise. This measure can also serve to evaluate the goodness of fit of a source model F^{SRC} to target data $\mathbb{S}^{\text{TRG}}$, referred to in this paper simply as the *fitness* of SRC to TRG, = $\text{accuracy}(F^{\text{SRC}}, \mathbb{S}^{\text{TRG}})$. These accuracy measures range from 0 to 1, with higher values denoting better models.

The fitness measure serves to estimate which source models are likely to transfer well to target object classes. A source selection algorithm might additionally use other measures such as the visual similarity of the icon used to represent the object or the frequency at which the object class model successfully transfers to other object classes. Such additional measures could further improve performance on source selection but are left to future research.

Target Model as Ensemble of Source Models The algorithm for transferring object class models is described in detail in this subsection. The algorithm starts with several trained object class models from source games. Then it observes some frames in a new target game. Some of the source models should predict later observations of the target game objects more accurately than new models trained from scratch on the observations made so far. Specifically, the assumption is that source models with high fitness to the target data should be more useful than those with low fitness. Therefore, for each object class in the target game, the algorithm builds a *transfer ensemble* out of both the pool of source models and the new scratch target model. The basic ensemble used is a forward model that makes predictions based on the weighted sum of its constituent forward models' predictions. Each of its constituents is assigned a weight as follows:

1. The scratch target model gets a nominal weight of 1.
2. Each source model j gets a nominal weight equal to $(b_j - a/2)$, where b_j is the source model's fitness and a is the scratch accuracy. Subtracting a portion of the scratch accuracy increases the relative strength of weights given to fitter source models.
3. Source models with non-positive weights are dropped.
4. The final weights are normalized by the sum.
5. The source models are *retrained* by adjusting their internal coefficients through the same gradient descent method used for their initial training, minimizing prediction error on the new target data while leaving intact the parts of the source models uninformed by target data.

The transfer ensemble is expected to predict target objects in out-of-sample data better than the scratch target model alone because of the inductive bias — the ensemble is biased toward models that work in other games. Retraining improves the accuracy of the transfer ensembles and reduces the number of cases and severity of negative transfer.

Experiments and Results

Out of the 60 GVG-AI games, 30 were used for exploratory testing and tuning of the system, while the other 30 were withheld for experiments. The reason for this division was to prevent bias from corrupting the results of the experiments.

The first experiments test the generalization ability of transfer ensembles compared to models learned from scratch. The hypothesis is that, after observing a small amount of in-sample training data from target games, transfer ensembles achieve higher accuracy on out-of-sample target data than scratch models. Initially, source models are learned from scratch using 500 frames of each source game. Then, for each of the 30 target games, scratch and transfer models are trained on 10 frames of target data. Each transfer model is an ensemble composed of object models from one of three disjoint sets of six randomly selected source games. The target game is never in the set of source games used for transfer. The ensemble is built according to the method described in the section above, using target training data both to calculate each source model's fitness score and to retrain the models. After the 10 frames of training, 100 out-of-sample testing frames are produced from a different level of the target game, and accuracy is measured for each object class model produced by the scratch and transfer methods. All player actions are selected randomly.

The main measure of success is the outperformance in forward prediction accuracy of transfer models over scratch models in the testing frames for each object class in each target game. To reject the null hypothesis of the differences being due to chance, a t-statistic is used to compute a one-sided p-value. A total of 500 object class models from 30 games are tested.

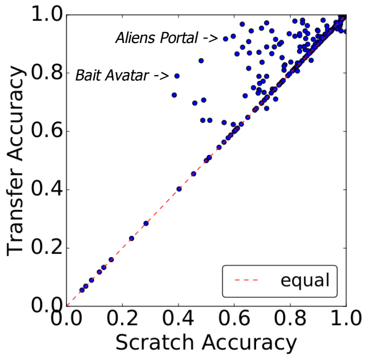

Figure 2: Test accuracy for scratch models versus transfer models. Points represent object classes, which come from all of the target games. Points appearing on the line or in the top-left half of the plot indicate transfer does as well or better than scratch. Transfer outperforms scratch for many object classes, such as the avatar in Bait and the portal in Aliens, and never significantly reduces accuracy.

Table 1 shows that the average increase in accuracy is statistically significant, and it can be concluded that the transfer ensemble approach for learning models of object classes is sound. Figure 2 displays how the transfer ensemble models compare in out-of-sample accuracy against models trained from scratch. Each dot represents one object class model; scratch and transfer perform equally when a dot falls on the line, transfer outperforms when a dot is in the upper-left, and scratch outperforms when a dot is in the lower-right.

Table 1: Mean accuracy μ of scratch (s) and transfer (t) models for each experimental setup with training frames T and source set S. Also reported are the mean accuracy μ^a for models of the avatar – usually the most important object – and t-statistics for differences in object class model accuracy between transfer and scratch. Improvement in accuracy from using transfer is statistically significant.

T	S	μ_s	μ_t	μ_s^a	μ_t^a	t	p
10	1	0.90	0.92	0.74	0.80	3.53	**<1%**
10	2	0.92	0.94	0.75	0.82	3.46	**<1%**
10	3	0.91	0.92	0.78	0.85	3.70	**<1%**

These graphs show that improvement is consistent across many games with rare occurrences of negative transfer.

As shown in Table 1, the average difference between scratch and transfer performance is only about two percent. However, this average difference understates the significance of the improvement. Many object classes are easy enough to model that scratch achieves perfect accuracy. For example, wall objects never move and are modeled with perfect accuracy by scratch in all the tested target games. More important is the improvement in object classes that are hard to model, such as the avatar, which behaves with varying complexity depending on the game. Transfer achieves higher accuracy of about seven percent for the avatar models. Furthermore, Table 1 shows that the improvement is consistent using all three sets of source games. Overall, these results strongly support the hypothesis that the transfer method of this paper leads to improved out-of-sample accuracy for many object classes, with very little negative transfer.

The final experiments test how well scratch and transfer models perform relative to each other and relative to a random action-taking agent on the task of exploring the environment. Agents perform this task on three levels of the game Labyrinth, in which the agent must guide the avatar to reach a destination through maze-like levels containing a few spike tiles fatal to the avatar.

First, agents are given either 10, 50, or 100 frames of training before being evaluated on a fresh 500 frames. If the avatar dies or reaches the goal at any time, the game is restarted with the avatar in its original position. In all setups, the transfer agent is built using an ensemble of six random source games other than Labyrinth, with 500 frames of training for each source, and retrained on the 10/50/100 frames of target data. Termination models are trained in addition to object class models for both scratch and transfer in source and target games in order to help predict death.

After training, agents go through a testing phase of another 500 frames. During this phase, agents use their forward models to choose one-step actions most likely to take them to novel or least-recently visited states. Their decision-making works as follows. The agent remembers each unique game state it visits and the time frame at which it was last visited. For each action the agent predicts the next game state by using its object class models to predict the next state of each game object. Model outputs are treated as probabilities of setting the corresponding object variables to one.

Treating forward predictions as probabilistic samples in this way helps agents avoid getting stuck. The value of each action considered at each frame by an agent is calculated as $1 - \frac{t_{\hat{S}}}{t_A}$, where t_A is the current time frame of the game and $t_{\hat{S}}$ is the last visited time frame of the predicted next state ($t_{\hat{S}} = 0$ if the state has never been visited). If the agent predicts death the value is -1. At each frame the agent chooses whichever action has the maximum value. At the end of the testing phase, the total number of unique states visited are counted and used as the metric of evaluation. After the testing phase, an additional 500-frame phase is run to measure prediction accuracy as in the previous experiments. During this phase, all agents take random actions rather than informed ones, in order to ensure fair comparison. The purpose is to determine the relationship between model accuracy and actual performance on an important task requiring action selection. Results are averaged over five experiments on each of the three levels of Labyrinth and each of the three training setups.

Table 2: Average improvement in accuracy (Acc) and exploration (Exp) over random actions, by level and training size (N), for the scratch (S) and transfer (T) approaches. Transfer outperforms scratch in exploration even when they are tied for accuracy, as in the results of the 100-frame training scenarios. The conclusion is that transfer leads to better accuracy and exploration performance.

Map	N	Acc_S	Acc_T	Exp_{S-R}	Exp_{T-R}
L0	10	0.71	0.86	-3.2	45.8
L1	10	0.77	0.84	4.4	24.4
L2	10	0.69	0.92	2.4	12.6
L0	50	0.84	0.93	12.4	27.8
L1	50	0.89	0.91	-3.6	32.8
L2	50	0.88	0.94	3.4	13.4
L0	100	0.95	0.86	41.8	39.2
L1	100	0.91	0.9	12.2	26.6
L2	100	0.93	0.95	10.6	25.6

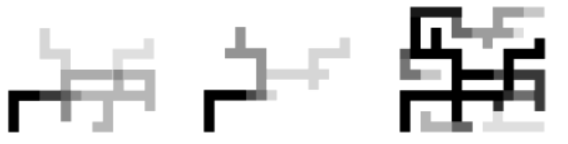
(a) Random agent (b) Scratch agent (c) Transfer agent

Figure 3: Trajectory maps of avatar during test phase, five runs overlaid for Level 0, agents trained on 10 frames. The maps show more space explored by transfer agents.

Table 2 shows how scratch and transfer agents perform in exploring three levels of Labyrinth given 10, 50, and 100 initial frames of training. Figure 3 shows an example of the avatars' trajectories. The agents trained from scratch on only ten frames of the game are not highly accurate in out-of-sample experience and struggle to perform better than random exploration. In contrast, the transfer agents are more accurate, supporting the results of the previous experiments, and are also able to explore much more efficiently.

As the number of training frames increases to 100, scratch models catch up in accuracy to transfer models. Interestingly, the transfer agents still explore more efficiently on average than the scratch agents, despite not being any more accurate. One possible explanation for the outperformance unexplained by accuracy is that the transfer agent may be particularly more accurate in the more important predictions. For example, if the avatar dies from contact from a spike tile, the agent must restart from the original position. The transfer agent may more accurately predict this deadly interaction than a scratch agent when no spike traps appear during training because the transfer agent is composed of some source models that predict death from contact with foreign objects. Being able to avoid death allows the transfer agent to keep exploring while the scratch agent has to start over. This advantage from using transfer is underestimated by the average accuracy measure, which is diluted by other predicted behaviors.

Discussion and Future Work

The methods explored in this paper - object-oriented factorization, transfer ensembles, and model retraining - help improve the sample efficiency of agents learning GVG-AI games. In these experiments, transfer-learning agents were more accurate than scratch agents when predicting future states. They were also more efficient at exploration, which is a widely useful ability for learning games. Future work will investigate the ultimate task of maximizing score, which is outside the scope of this paper because it requires the integration of planning and value approximation methods.

GVG-AI games contain diverse challenges that test how well the learning approach generalizes across games. Crucially, its games also have stochastic behaviors and multiple levels, which test how well agents generalize across experiences. However, there are some challenges that are not covered by the GVG-AI domain, and an important path for future work is to improve the robustness of this approach by adapting it to other domains. For example, using transfer to reduce generalization error could be useful in domains with noisy or high-dimensional observation spaces.

Conclusion

This paper demonstrated a model-based transfer learning approach for training video game agents from very little data. The approach constructs an ensemble out of source object models and uses the limited target data both to choose the ensemble weights and to retrain the final model. Although both scratch and transfer models achieve global minima in prediction errors during training, experiments showed consistently higher out-of-sample performance for transfer models across diverse GVG-AI games. Transfer agents showed particular improvement in modeling important objects such as avatars, which was useful for more quickly exploring unfamiliar game maps. Artificial agents can use this approach to accelerate early-stage learning and quickly adapt to novel situations.

References

Asmuth, J., and Littman, M. L. 2011. Learning is planning: Near bayes-optimal reinforcement learning via monte-carlo tree search. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*.

Atkeson, C. G., and Santamaria, J. C. 1997. A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*. Citeseer.

Baxter, J. 2000. A model of inductive bias learning. *J. Artif. Intell. Res.(JAIR)* 12:149–198.

Bellemare, M.; Veness, J.; and Bowling, M. 2013. Bayesian learning of recursively factored environments. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 1211–1219.

Caruana, R. 1997. Multitask learning. *Machine learning* 28(1):41–75.

Dai, W.; Yang, Q.; Xue, G.-R.; and Yu, Y. 2007. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, 193–200. ACM.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd international conference on Machine learning*, 257–264. ACM.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, 240–247. ACM.

Eaton, E., and DesJardins, M. 2009. Set-based boosting for instance-level transfer. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, 422–428. IEEE.

Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a video game description language.

Gao, J.; Fan, W.; Jiang, J.; and Han, J. 2008. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 283–291. ACM.

Hausknecht, M., and Stone, P. 2015. The impact of determinism on learning atari 2600 games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Hausknecht, M.; Lehman, J.; Miikkulainen, R.; and Stone, P. 2014. A neuroevolution approach to general atari game playing. *Computational Intelligence and AI in Games, IEEE Transactions on* 6(4):355–366.

Hester, T., and Stone, P. 2013. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning* 90(3):385–429.

Jong, N. K., and Stone, P. 2007. Model-based function approximation in reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 95. ACM.

Mitchell, T. M. 1980. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, 2845–2853.

Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on* 22(10):1345–1359.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Schaul, T. 2013. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8. IEEE.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research* 10:1633–1685.