UNIVERSAL AGENT FOR DISENTANGLING ENVIRONMENTS AND TASKS

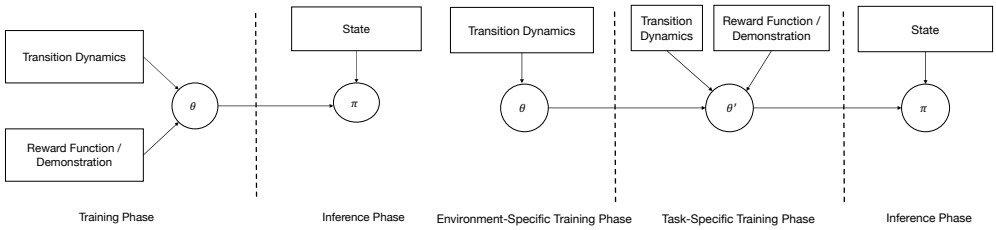
Anonymous authors
Paper under double-blind review

ABSTRACT

Recent state-of-the-art reinforcement learning algorithms are trained under the goal of excelling in one specific task. Hence, both environment and task specific knowledge are entangled into one framework. However, there are often scenarios where the environment (*e.g*. the physical world) is fixed while only the target task changes. Hence, borrowing the idea from hierarchical reinforcement learning, we propose a framework that disentangles task and environment specific knowledge by separating them into two units. The environment-specific unit handles how to move from one state to the target state; and the task-specific unit plans for the next target state given a specific task. The extensive results in simulators indicate that our method can efficiently separate and learn two independent units, and also adapt to a new task more efficiently than the state-of-the-art methods.

1 INTRODUCTION

Let's imagine ourselves learning how to play tennis for the first time. Even though we have never played tennis before, we already have a good understanding of agent and environment dynamics related to tennis. For example, we know how to move our arm from one position to another and that a ball will slow down and bounce back from the ground. Hence, we just need to learn the tennis specific knowledge (*e.g*. its game rule and a relationship between an arm control and a tennis racket). Just like this example, when we learn to complete a new task, we utilize the prior knowledge that is disentangled from the task and acquired over our lifetime.



(a) Specific Agent (b) Universal Agent

Figure 1: Our model disentangles environment-specific information (*e.g*. transition dynamics) and task-specific knowledge (*e.g*. task rewards) for training efficiency and interpretability.

From a reinforcement learning perspective, this brings a very interesting question – how can agents also obtain and utilize such disentangled prior knowledge about the environment? Most of today's deep reinforcement learning (DRL) models Mnih et al. (2016; 2015) are trained with entangled environment-specific knowledge (*e.g*. transition dynamics) and task-specific knowledge (*e.g*. rewards), as described in Figure 1a However, as described earlier, humans have an innate ability to obtain a good understanding about the environment dynamics, and utilize them in a newly given task. Motivated from this, we introduce a new scheme to disentangle the learning procedure of task-independent transition dynamics and task-specific rewards, as described in Figure 2. This will help an agent to adapt to a new task more efficiently and also provides an extra interpretability.

The idea of disentangling a model into two components can be related to hierarchical RL approaches Sutton et al. (1999); Parr & Russell (1998); Oh et al. (2017). However, to the best of our knowledge,

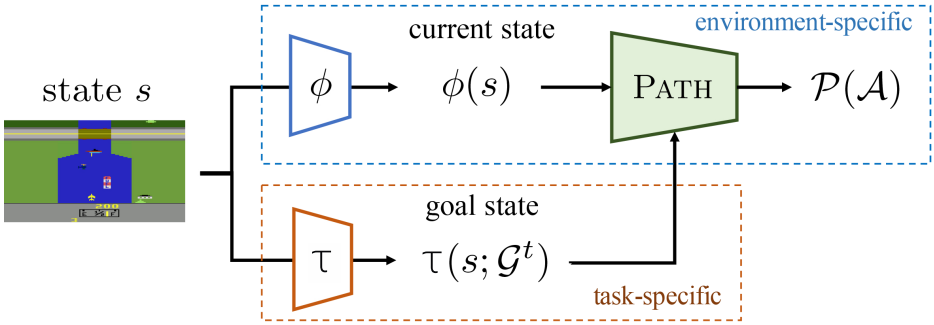


Figure 2: Proposed *universal agent*, which consists of three parts: a ϕ function mapping raw observation to feature space, a PATH function as an environment actor, and a τ function for future state planning.

there has not been a work that separating units by the natural criteria of environment and task specific knowledge, for the goal of transfer learning.

To this end, we introduce a model that consists of two major units: a PATH function and a goal generator. This is illustrated in Figure 2. The key intuition is as the following. PATH function handles the environment specific knowledge, and a goal function handles the task specific knowledge. To achieve this, we design (1) PATH function to learn how to move from one state to another – a lower-level controller, which is independent from the task and only depends on the environment, and (2) the goal function τ to determine the next state given a target task – a higher-level planner. Thus, PATH function can be shared across different tasks as long as it is under the same environment (*e.g.* the physical world).

We evaluate our method to answer the following two questions: (1) how a good PATH unit can benefit the task learning, and (2) how efficient is our model for learning a new task in the same environment. We analyze the behavior of our method on various environments including a maze world, Lunar Lander, and multiple Atari 2600 games. Our study shows that a good PATH unit can be trained, and our model has a faster convergence compared to the state-of-the-art method Mnih et al. (2016) in most of tasks, especially on transfer learning tasks.

In summary, we introduce a RL model with disentangled units for task-specific and environment-specific knowledge. We demonstrate in multiple environments that our method can obtain environment-specific knowledge, and this further enables an agent adapting to a new task within the same environment.

2 METHOD

Our goal in this paper is to introduce a reinforcement learning model that disentangles environment and task-specific knowledge. The advantage of such disentangling model is that we only need to learn task-specific knowledge when a goal of a task changes (*e.g.* the goal of bowling game changes from hitting 10 pins to 5 pins), or environment-specific knowledge when an environment of a task changes (*e.g.* an deceleration of a bowling ball changes in the game). This essentially is one form of generalization techniques across tasks or environments.

To this end, we introduce a modularized neural networks model – one unit handling a task-specific knowledge, and the other unit handling an environment-specific knowledge. For this, our model introduces two sub-functions: a goal function (τ) and a path function (PATH). Our goal function (τ) determines and generates the next target state for a given task (task-specific unit), and our path function (PATH) will provide the way to get to the target state (environment-specific unit). In this setup, we only have to change τ function when the goal of a task changes (*e.g.* hitting 5 pins instead of 10 pins) in the same environment, because the way to get from one state to another state stays the same within the same environment (*i.e.* PATH function can remain the same). On the other hand, we only need to change PATH function if the environment changes (*e.g.* faster deceleration of a bowling

ball) while the task is the same, because the way to go to one state from the current state changed (*e.g.* due to deceleration) while the goal of the task itself is the same.

2.1 PATH FUNCTION

We want to design our path function (PATH) to determine how to get to a state from another state, which represents environment-specific knowledge. Hence, we define a PATH function: $\text{PATH}(s, s') \rightarrow \mathcal{P}(\mathcal{A})$. Given a *(starting state, target state)* pair, PATH function outputs a probability distribution over action space for the first action to take at state s in order to reach state s'. For discrete action spaces $\mathcal{A} = \{a_1, a_2, \cdots, a_k\}$), the output is a categorical distribution, and for continuous action space: $\mathcal{A} = \mathbb{R}^k$, a multidimensional normal distribution with a spherical covariance is outputted Lillicrap et al. (2015). To generate a trajectory from s to s', one can iteratively apply the PATH function and obtain a Markov chain.

Given the state space $\mathcal{S}$, an alternative to train the PATH function is by sampling a starting state and a final state and teach the PATH function to generate the path from one to another using reward signal. Formally speaking, we define a series of games based on the original transition dynamics $s' = \text{Trans}(s, a)$[1], where s is the state and a is the action. For sampled state pair $(s, s') \in S' = \mathcal{S}^2$, we construct the following game instance $(S', \text{Trans}', \text{Reward}')$:

$$\text{Trans}'((s, s'), a) = (\text{Trans}(s, a), s'); \quad \text{Reward}'((s, s'), a) = \begin{cases} 1 & \text{if } \text{Trans}_e(s, a) = s' \\ -1 & \text{otherwise} \end{cases}.$$

We add the -1 term in reward function to enforce the actor to choose the shortest path getting from s to s'. PATH function can be learned with any RL algorithms on the defined game instances.

The state pair can be obtained by directly sampling from the state space (if known), or from agents' experience (requires the restoration of history states). We leave other approaches for the PATH training as future work.

2.2 GOAL FUNCTION

Now, we also want to design a goal function (τ), which determines what the goal state should be for a given task, and thus task-specific knowledge. We define our goal function for a given task: $\tau(s; \theta^g) \approx \phi(s')$ to compose a target state in feature space given the current state. PATH function is then inferred to get the primitive action to be taken.

Train with back-propagation The simplest way to train the τ function is by back-propagation through the PATH module. As in A3C, actor corporates with a task-specific critic network which maintains a value estimation for any state s. Fixing the parameters for PATH θ^p, the gradient for θ^g is computed as:

$$\nabla_{\theta^g} \log \pi(a|s; \theta^p, \theta^g) A(s, a; \theta^p, \theta^g, \theta^v) = \frac{\partial s'}{\partial \theta^g} \nabla_{s'} \log \text{PATH}(a|s, s'; \theta^p) A(s, a; \theta^p, \theta^g, \theta^v),$$

where $s' = \tau(s; \theta^g)$ and A is the advantage computed by $\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta^v) - V(s_t; \theta^v)$ and γ is the discount factor Mnih et al. (2016).

Reduce to a continuous control problem Another way to train the τ function is by viewing it as the response made by the agent and optimize it using the reward signal. Since the output lies on the manifold of $\phi(\mathcal{S})$, we can model it as a Gaussian distribution (as in continuous control RL). Any RL algorithms operated on continuous action space Schulman et al. (2015a;b); Lillicrap et al. (2015) can thus be applied to optimize the τ function.

A natural question would be: how to better choose the manifold of $\phi(\mathcal{S})$, which certainly affects the learning of τ? For example, can we use β-VAE Higgins et al. (2016) which enforces the distribution of ϕ's output to be close a Gaussian distribution to help? In the experiment section, we will study several

[1]For simplicity here, environmental randomness is considered as a part of the transition dynamics. *E.g.* assuming s contains the random seed of the environment.

choices of the $\phi(\mathcal{S})$ space including auto-encoder Hinton & Salakhutdinov (2006), beta-VAE Higgins et al. (2016), forward dynamics Chiappa et al. (2017), inverse dynamics (one-step PATH function) Pathak et al. (2017), and the feature space jointly learned with PATH function.

Thanks to the recent advances in low-bits neural networks Rastegari et al. (2016); Zhou et al. (2016), we can binarize the output coding of ϕ. Specifically, we first train ϕ with a $\tanh$ activation at the last layer, then binarize the output, and finally fine-tune the ϕ again. By doing so, the distribution of τ function becomes a binomial distribution.

2.3 TRAINING DETAILS

Reinforcement learning model We implemented batched A3C as our base model following same settings as original paper Mnih et al. (2016). The discount factor is chosen as $\gamma = 0.99$ for all our experiments. We use a convolutional neural network (CNN) structure (2-4 layers depending on the task) instead of LSTM as feature extractor for ϕ and τ, while PATH function is a 3-layer MLP. To preserve historical information, we concatenate four consecutive frames in channel dimension as input to our network (as state s). The critic network shares feature with τ (they are both task-specific) and has two extra fully-connected layers for value estimation.

Curriculum learning for Path function The game defined in Section 2.1 is a game with very sparse reward. To accelerate the training process, we employ curriculum learning to help. In particular, the max distance between starting state and final state is initialized as one at the beginning of the training process. After every K iterations of updating, the max distance is increased by one. This max distance can be either accurate (by finding the shortest path from s to s'), heuristic (*e.g.*, Manhattan distance on a 2D maze), or determined by agents' experience (see experiments).

Compared to other pre-training methods or hierarchical RL methods, the proposed *universal agents* shows following two main advantages:

1. PATH function can be obtained without task specification. With only exploration (*e.g.*, naive random walk or curiosity-driven RL Pathak et al. (2017)), PATH function can be trained using the experience.
2. PATH function encodes no information about the task. When jointly learned on one or more tasks, PATH function shows better generalization to new tasks.

3 EXPERIMENTS

We begin our experiments with a small environment: Lava World. Rich ablation experiments are conduced to study the choice of feature space, optimization strategy and its knowledge transfer ability. We further extends our result to a more realistic environment: Atari 2600 games. More experiments like continuous control problem or imitation learning would be shown in the appendix.

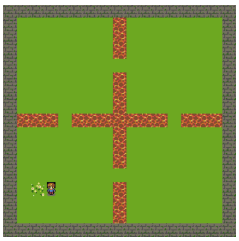


Figure 3: Lava World

3.1 LAVA WORLD

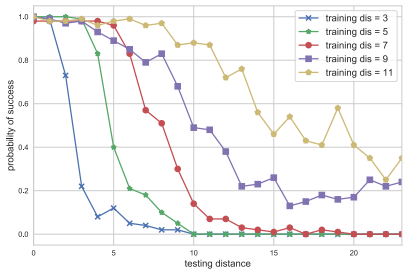
Lava world (Figure 3) is a famous 2D maze in reinforcement learning. The agent moves (in 4 directions) in the maze. The whole map is 15×15 and divided into 4 rooms. The grids with lava are obstacles to the agent. Therefore, to go from one room to another, the agent is forced to find the door connecting two rooms and go through it, which is harder than random generated 2D maze.

Path function The set of states in lava world is definite, we directly use randomly generated starting state and final state to train the path function. During this training phase, the agent understands how to move itself as well as how to go from one room to another.

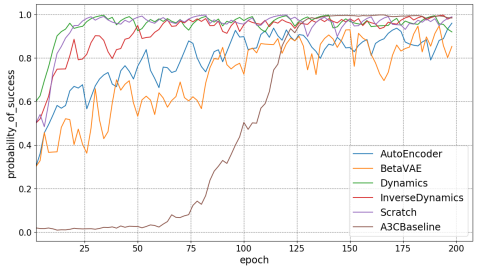
We test the generalization ability of our PATH function by setting max distance between the starting state and the final state during training but testing it on harder instances (the distance is longer). Figure 4a shows the result. When trained with very simple instances (distance is less than 7), the

PATH function shows very limited generalization ability. When the training distance is increased to 7, it begins to generalize. This phenomenon was also witnessed in other experiments: too short training distance leads to pool generalization.

Task: Reachability In this task, agents is asked to move to the target position presented on the map as a special grid. In all experiments, agents are provided with dense reward: when the agent is getting closer to the target position, it can receive positive reward. Shown in figure 4b, *universal agent* demonstrates faster learning speed. The learned τ function is well interpretable. We provide visualization of the goal state composed by the agent in the appendix A.



(a) Generalization analysis of PATH function.



(b) The rate of successfully reaching the final position in game Reachability. The universal agent model is trained through back-propagation.

Feature space and training strategy We now go deeper into the choice of feature space of ϕ and the training strategy for τ.

We explored following feature spaces for the state encoder ϕ:

- AutoEncoder: Encode the input state with an information bottleneck proposed in Hinton & Salakhutdinov (2006).
- β-VAE: Encode the input state with a disentangled coding by best compressing the information. We adjusted β such that the information is reduced as much as possible under the constraint that the reconstruction is perfect. From another point of view, β-VAE also enforces the feature space to be close to a Gaussian distribution. Higgins et al. (2016).
- Forward dynamics: Similar to Chiappa et al. (2017), we trained an neural network to approximate the dynamics of the environment: $p(s_{t+1}|s_t, a_t)$.
- Inverse dynamics: Used by Pathak et al. (2017), inverse dynamics is an one-step PATH function: $p(a_t|s_t, s_{t+1})$.
- No pre-training. The feature space is jointly learned with PATH function.

We first train PATH function for certain ϕ space with sufficient amount of data. Shown in Figure 5, autoencoder and β-VAE perform worse than other feature spaces due to their ignorance of the underlying environment's dynamics. The interesting "odd-even" pattern, we hypothesis, is attributed to the feature space for pixel-level reconstruction.

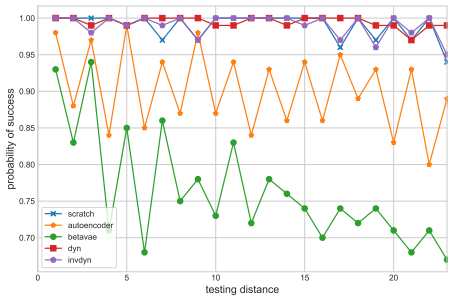
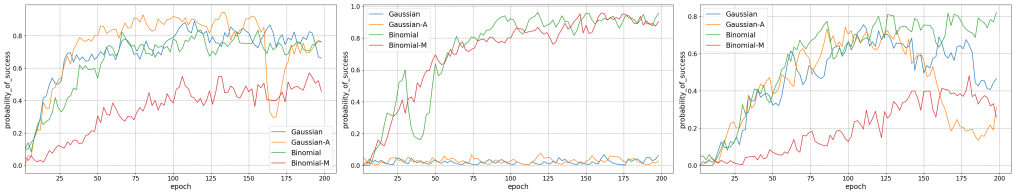


Figure 5: The quality of path functions learned over different feature spaces.

For all feature space mentioned, we model the output distribution of τ function as multi-dimensional Gaussian distribution with a scalar variance σ^2. For their binarized variants, we model the distribution as binomial distribution. We also investigate two variants of the distribution modeling: Gaussian-Additive ($\phi(t) \approx \phi(s) + N(\mu, \sigma^2)$) and Binomial-Multiplicative ($\phi(t) = \phi(s) * \text{Binomial}(logits)$).

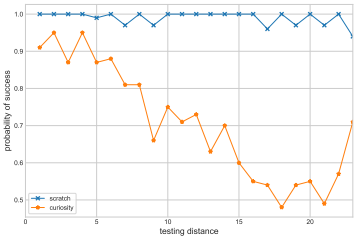
We use proximal policy optimization (PPO) Schulman et al. (2017) for the distributional τ learning. Shown in figure 6, Gaussian and Gaussian-A have similar behavior while Binomial is typically a



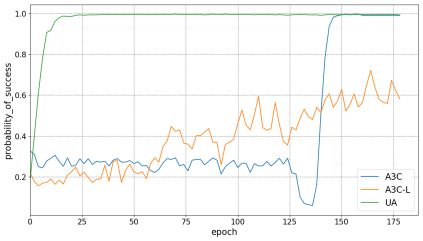
(a) Dynamics (b) Inverse Dynamics (c) β-VAE

Figure 6: Different feature spaces affect the learning of τ function

better choice than Binomial-B, probably because currently $\tau(s)$ does not depends on $\phi(s)$, which means the target state space composer has no information about the current state representation $\phi(s)$. Gaussian models performs well on all feature spaces with pixel-level information (*e.g.*, β-VAE and dynamics), while fails in the feature space with more high-level, discrete information (inverse dynamics). The hyper-parameters for the distributional models are generally more difficult to adjust, but might be useful in robotics settings where the input space itself is usually a continuous space (*e.g.*, joints of a robot). However, it shows no clear advantage over A3C shown in figure 4b on the visual input games.



(a) PATH function learned using curiosity-driven RL without any reward.



(b) The probability of successfully finish the task during testing time in game Taxi. We only measure the probability of agents finishing the whole task.

Knowledge transfer via PATH function We study two types of knowledge transfer: knowledge from exploration and knowledge from task. We implemented Pathak et al. (2017) as a no-reward RL explorer in unknown environments. The experience of the agents are collected for the PATH training. In no-reward exploration setting, PATH function is trained from scratch without pre-trained feature space. Since we can not exactly get the distance between two states in this setting (assuming we have no knowledge about the state space), when picking the starting state and the final state from the experience pool, we set the limit the number of steps that the explorer used as 15. With same number of frames experienced ($\sim$ 32M frames), the PATH function can reach comparable accuracy as the original curriculum learning correspondent. We compare the performance of two models in Figure 7a. Their performances on the reachability tasks are almost identical.

To study the transfer across multiple tasks, we define another task in the environment: Taxi. The agent now acts as a taxi driver, who first goes to one position to pick up the passenger (shown on the map), and then goes to another place. The target position will be provided to the agent by two one-hot vectors representing the (x, y) coordinates after our agent picks up the passenger. In order to complete the task, the agent learns how to interpret the task representation (one-hot vectors).

We compare the performance of *universal agent* with both A3C baseline trained from scratch as well as A3C model pre-trained for reachability game (A3C-L) in Figure 7b. Even through the first phase of the task is exactly the same as reachability game, *universal agent* excels in the task interpretation and outperforms both baselines.

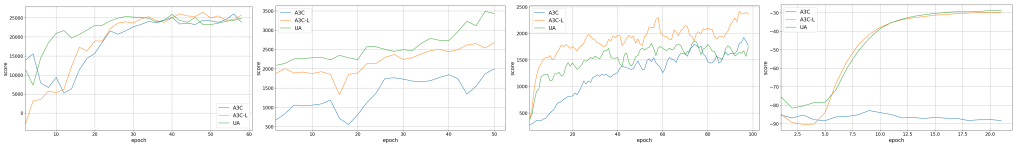
3.2 ATARI 2600 GAMES

We tested our framework on a subset of Atari 2600 games, a challenging RL testbed that presents agents with a high-dimensional visual input (210×160 RGB frames) and a diverse set of tasks which are even difficult for human players.

Train the path function A critical feature of Atari games is the difficulty of exploration, which is attributed to the complexity of the environment as well as the intricate operation. We use state restoration for the PATH learning, starting state and targeting state are sampled from agents' experience performing tasks or exploration.

We first chose 8 typical Atari games to validate the effectiveness of PATH function in the appendix D. Also, we provide analysis on knowledge transfer from exploration in the appendix E.

Knowledge transfer from other tasks *Universal agent* is capable of efficient transfer learning. We further define some new tasks for four of the Atari games. The task specifications are included in the appendix F. In Figure 8, we demonstrated how our *universal agent* can benefit from the pre-trained PATH function in original task. For a fair comparison, in A3C-L, we load all the weights (except the last fully-connected layers for policy and value) from the model trained on original task, while A3C represents A3C trained from scratch.

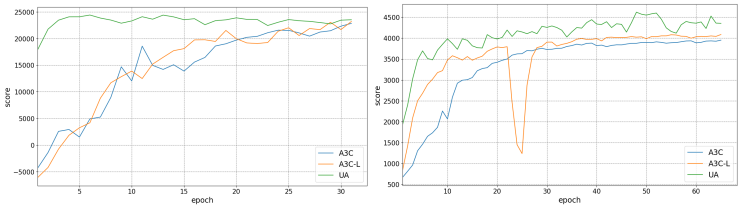


(a) JourneyEscape-new (b) Riverraid-new (c) UpNDown-new (d) Alien-new

Figure 8: Comparison between A3C, A3C with pre-trained model on the original task, and UA with pre-trained PATH function.

In both JourneyEscape-new and Riverraid-new, UA with pre-trained PATH function outperforms A3C baselines. We also show two more games where UA fails to perform very well. In game UpNDown-new, A3C succeeds in finding a bug in the game, and use it to escape from other cars. While performing slightly worse than A3C, UA chooses to jump over other cars to avoid being hit, which is more reasonable and rational. In Alien-new, both UA and A3C-L perform better than A3C trained from scratch. The new task has more sparse reward than other tasks, which brings difficulty for the τ function to learn.

Task transfer from multiple sources We demonstrate the ability of knowledge transfer from multiple tasks in two Atari games. A set of pre-trained A3C agents for different tasks in same environments are employed to collect experiences for our agent in the PATH function training phase. The task specifications can be found in the appendix. The learning procedure corresponds to humans' life-long learning from multiple tasks in the same environment. With this prior knowledge, agents are trained on completely new tasks with PATH function given and fixed. Results for A3C-S and A3C-L are also provided as baseline models.



(a) JourneyEscape-new (b) Riverraid-new

Figure 9: Comparison between A3C, A3C with pre-trained model on one of source tasks, and UA with pre-trained PATH function from multiple source tasks.

In these two environments, *universal agent* shows faster convergence and better performance than a3c with pre-trained model on another tasks.

4 RELATED WORKS AND DISCUSSION

Multi-task learning and transfer learning Multi-task learning and transfer learning Devin et al. (2016); Andreas et al. (2016); Taylor & Stone (2009) provide approaches to transfer knowledge among multiple agents. The methods include the decomposition of value function or task, and direct multi-task learning where agents learn several tasks jointly. Contrary to them, *universal agent* obtains the environment specific knowledge without any specific task supervision.

Hierarchical reinforcement learning In standard hierarchical RL (HRL) Sutton et al. (1999); Hernandez-Gardiol & Mahadevan (2001); Kulkarni et al. (2016); Fox et al. (2017) a set of subtasks are defined by a hyper-controller. Low-level controllers receive subtask signals from the hyper-controller and output primitive actions. Both low-level controllers and the hyper-controller (sometimes called meta-controller) learn from a specific task. Contrary to them, the proposed PATH function can be trained without supervision and provides a more general interface for potential tasks. Unlike Schaul et al. (2015); Vezhnevets et al. (2017), the τ function does not perform option selection but directly composes target state since a general-purpose PATH function is utilized. We do not follow typical HRL methods where once the subtask is selected, the low-level controller will be executed for multiple time steps. In *universal agent*, for simplicity now, τ function plans the future state at every time step. We leave its adaption to HRL frameworks as a future work.

Instruction-based reinforcement learning In Oh et al. (2017), instruction are provided to agents as task description. Trained with the ability to interpret instructions, agents can perform zero-shot generalization on new tasks. However, the performance of this method is still restricted by the set of tasks the agents are trained with. *Universal agent* addresses the generalization among multiple tasks by a general PATH function.

Path function PATH function is an instance of *universal value function approximator* Schaul et al. (2015). While a set of works use the prediction of future state as knowledge representation for RL agents Chiappa et al. (2017); Dosovitskiy & Koltun (2016) (*i.e.*, trying to approximate the transition dynamics $p(s_{t+1}|s_t, a_t)$), the proposed PATH function can be viewed as an inverse and a high-level abstract version of the approximated transition dynamics. From another point of view, PATH function is a variant of the feasibility function which measures the reachability from one state to another. An important property of PATH function is the ignorance of knowledge about any specific tasks. Therefore, PATH function can be shared among different agents working on different tasks but in same environment.

Imitation learning Imitation learning Hester et al. (2017); Duan et al. (2017) considers the problem of deriving a policy from examples provided by a teacher. Contrary to Reinforcement Learning (RL) where policy derived from experience, it provides an intuitive way to communicate task information as human. A common solution is to performs supervised learning from observations to actions Pomerleau (1989); Ross et al. (2011). However, pure supervised learning suffers from the lack of knowledge about domain dynamics, which leads to bad generalization when the environment is complex Hester et al. (2017). We show in the appendix how we incorporate prior knowledge of environment into an imitation learning framework and improve its generalization and robustness.

5 CONCLUSION

We present a new reinforcement learning scheme that disentangles the learning procedure of task-independent transition dynamics and task-specific rewards. The main advantage of this is efficiency of task adaptation and interpretability. For this we simply introduce two major units: a PATH function and a goal generator. Our study shows that a good PATH unit can be trained, and our model outperforms the state-of-the-art method Mnih et al. (2016) in most of tasks, especially on transfer learning tasks.

REFERENCES

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *arXiv preprint arXiv:1611.01796*, 2016.

Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *arXiv preprint arXiv:1609.07088*, 2016.

Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*, 2017.

Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

Natalia Hernandez-Gardiol and Sridhar Mahadevan. Hierarchical memory-based reinforcement learning. *Advances in Neural Information Processing Systems*, pp. 1047–1053, 2001.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, et al. Learning from demonstrations for real world reinforcement learning. *arXiv preprint arXiv:1704.03732*, 2017.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 3675–3683, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL `http://dx.doi.org/10.1038/nature14236`.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Communicating hierarchical neural controllers for learning zero-shot task generalization, 2017.

Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pp. 1043–1049, Cambridge, MA, USA, 1998. MIT Press. ISBN 0-262-10076-2. URL `http://dl.acm.org/citation.cfm?id=302528.302894`.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*, 2017.

Dean A Pomerleau. Alvinn, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, pp. 6, 2011.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1312–1320, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Supplementary Materials for Universal Agent

A INTERPRETABLE GOAL STATE IN REACHABLITY GAME

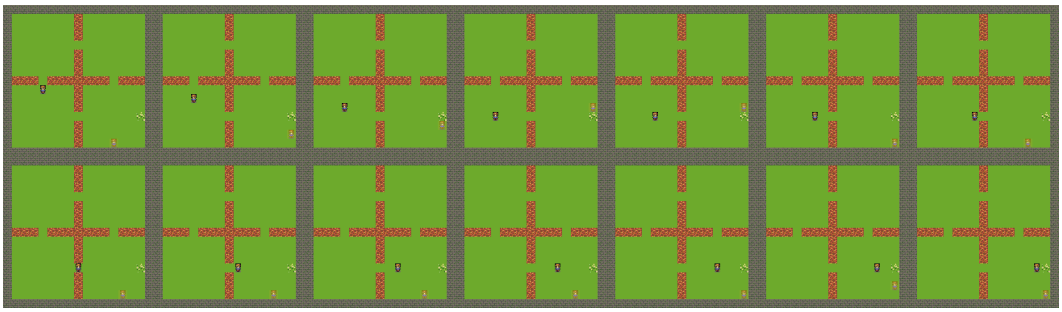


Figure 10: Visualization of the goal state composed by universal agent

Given the feature space of all states, we can visualize the goal state generated by τfunction by link it to the nearest real state in feature space. The shadow agent in Figure 10 shows the corresponding goal position when the agent is at its real position. The *universal agent* model can thus dramatically increase the interpretability of the agent's play in certain games.

B IMITATION LEARNING

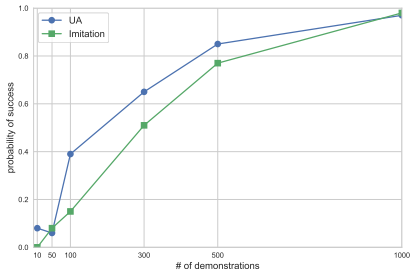
In imitation learning (IL), task supervision is provided by the demonstration coming from either human experts or other pre-trained agents. A straightforward way is to minimize $\mathbb{E}_{(s,a)\in\mathcal{D}} \log \pi(a|s;\theta)$. Given a set of demonstration plays (sequences of state-action pair $\mathcal{D}$), we can add extra supervision signal on *Universal Agent* to make it better understand the goal.

$$\mathcal{L}' = \mathcal{L} + \alpha\, \mathbb{E}_{\text{demo}} \text{KL}[\tau(s_t)|s_{t+k}],$$

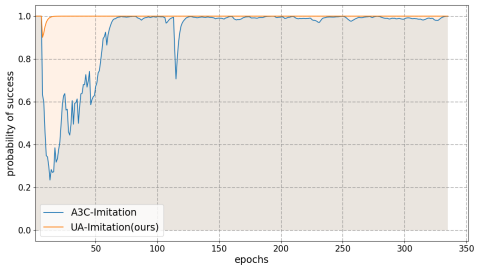
where s_t, s_{t+k} are two sampled state from a trajectory in $\mathcal{D}$. k is a planning horizon factor.

While the primitive action policy was obtained in environment-specific training phase, agents are forced to understand how to plan future state (what should happen) instead of primitive actions (what should be done). We show empirically that *universal agent* demonstrate better generalization and robustness over pure imitation learning.

In settings where the demonstration serves as a complement for reward function, we train our agents with both supervision. Figure 11b shows the performance.



(a) The probability of successfully reaching the final position during testing time in game Reachability by pure imitation learning.



(b) The probability of successfully reaching the final position during testing time in game Reachability by imitation learning + reinforcement learning.

When choosing a suitable reward function for a specific task is hard (*e.g.*, it suffers from very sparse rewards), we hope our agent has the ability to generalize from small set of demonstration and being able to perform the task by imitation learning. *Universal agent* also fits this setting. Instead of training agent with primitive actions given every state, it learns the planning (target state) directly. As introduced in Section 2.2, two variants of imitation learning are addressed: pure learning from demonstration and integrating demonstration into standard RL. In reachability task, each demonstration play contains one of the shortest paths from starting position to target position.

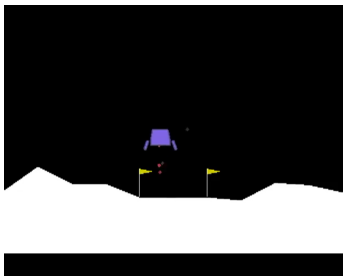
For pure imitation learning, Figure 11a shows the empirical analysis of agents' performance in the game with respect to the number of demonstration plays. With limited number of demonstration plays, *universal agent* interprets the task and shows better generalization than agents trained from scratch. Figure 11b illustrates consistent results for imitation learning + RL.

C LUNAR LANDER

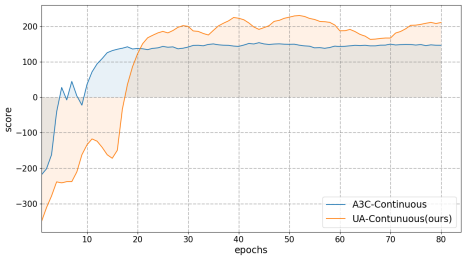
We choose Lunar Lander (Figure 12a as environment to evaluate *universal agent* in continuous control setting Lillicrap et al. (2015); Schulman et al. (2015a;b). In this environment, the agent controls a spacecraft with three engines (main, left and right) to land safely on the moon. Different from discrete control version where the agent can only choose the fire each engine or not, in continuous control, an engine can be fired partly, which enables more meticulous control. The reward is determined by whether the agent land successfully as well as the fuel consumed. The randomness of the environment comes from the configuration of the land as well as initial position and velocity of the lander. The baseline model is the continuous control version of A3C as in Mnih et al. (2016).

In lunar lander, the state space is rather difficult to explore by random actions. Thus we make use of a pre-trained A3C model for the landing task and other heuristic agents for imitation learning (described later) to provide demonstration plays for our agent in the PATH function training phase. To ensure the generality of learned PATH function, a large noise is added to the action performed by the demonstrators ($N(0, 0.6)$ in our setting, compared to the valid value interval $[-1, 1]$).

Once the PATH function is obtained, the agent is trained with A3C. Figure 12b shows the comparison between *universal agent* and A3C trained from scratch.

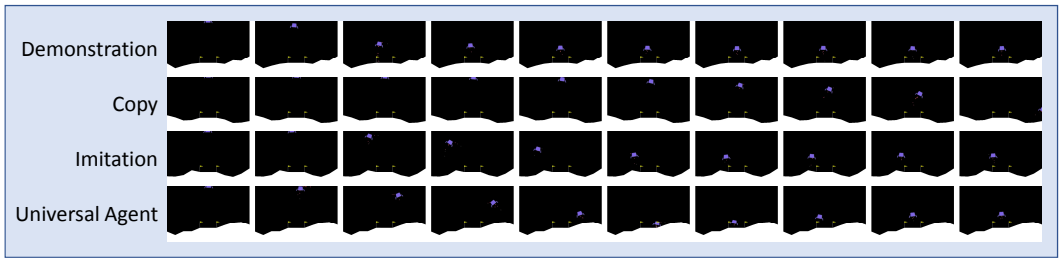


(a) Lunar Lander

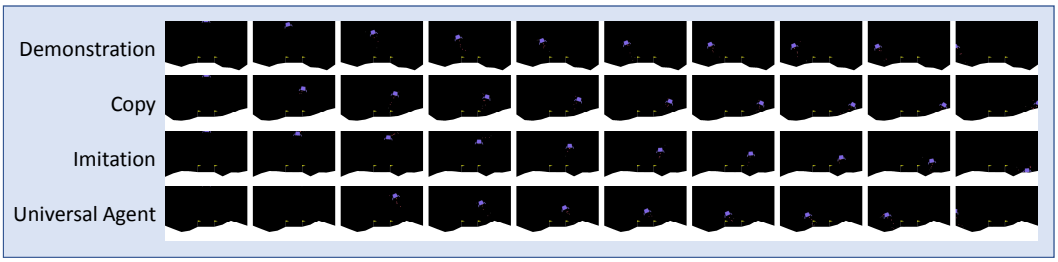


(b) Performance (scores) in game lunar lander during test time.

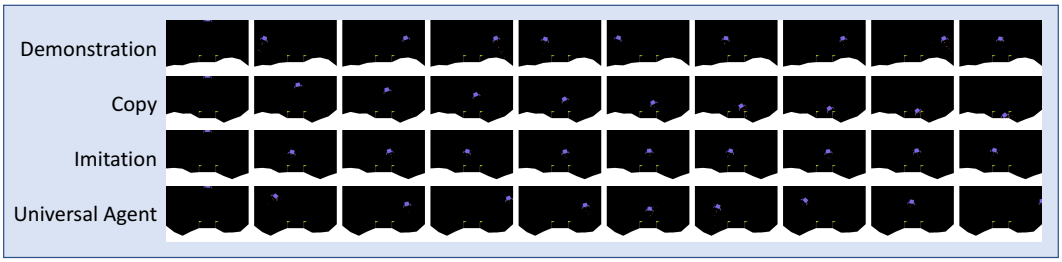
The Lunar Lander is a typical task where reward function is hard to design, especially for special tasks. To show the generalization ability, we design three new tasks for imitation learning: hover in the center, fly away from left-middle, and swing near the center. We provide the agent with a small set of demonstration plays (1 for all tasks) generated by heuristic players. The imitation learning baseline is a CNN model trained by minimizing the KL-divergence between policy $\pi(a'|s)$ with one-hot policy a for all demonstration pairs (s, a). Shown in Figure 13, *universal agent* with pre-trained PATH function outperforms pure imitation learning model in generalization and robustness.



(a) Lunar Lander extra task: Hover



(b) Lunar Lander extra task: Fly-away from left-middle

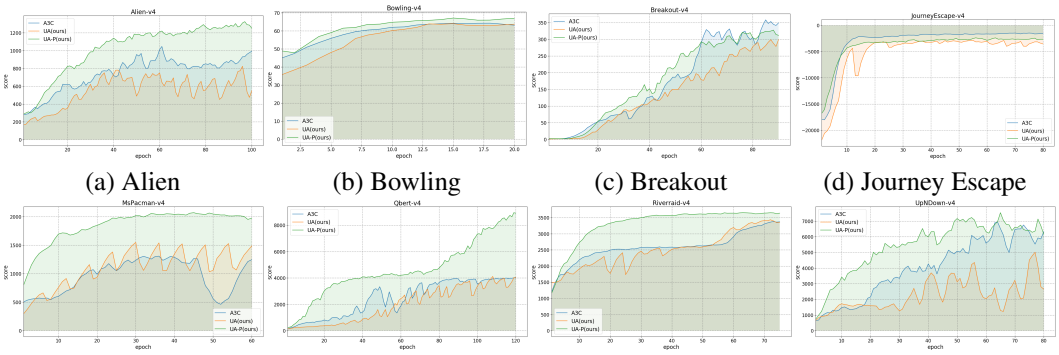


(c) Lunar Lander extra task: Swing

Figure 13: Learn from only demonstration in Lunar Lander. “Copy” represents agents which directly copy demonstration’s actions as baseline.

D JOINT TRAINING OF PATH FUNCTION AND τ FUNCTION IN ATARI GAMES

In particular, we modify the training pipeline for A3C and include two phases: PATH training and τ training, which are performed alternately. At first, with no knowledge about the state space, agents gather experience by random actions, which is further used for the training of PATH function. Then, with PATH function fixed, $\tau$is trained for the task. After a fixed number of iterations, since the agent has explored more in the state space, PATH function is trained again based on all experiences collected during the exploitation. And the max number of steps for PATH function is chosen as 15 for all experiments.



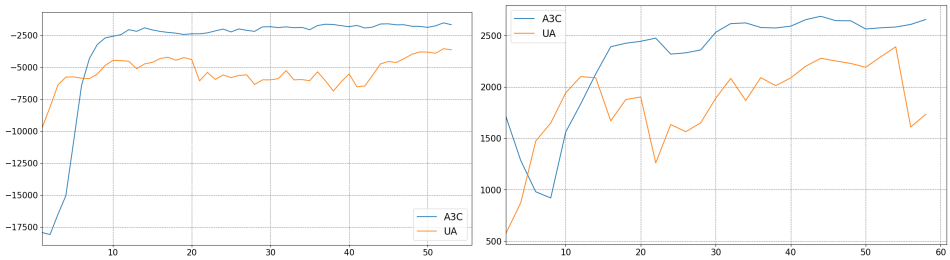
(a) Alien (b) Bowling (c) Breakout (d) Journey Escape

(e) MsPacman (f) Qbert (g) Riverraid (h) UpNDown

Figure 14: Comparison between A3C, jointly trained UA, and UA with pre-trained PATH function. With a pre-trained PATH function, a faster convergence speed in universal agent is witnessed.

E KNOWLEDGE TRANSFER FROM EXPLORATION IN ATARI GAMES

We studies how *universal agent* can benefit from a PATH function trained by experience of pure exploration without any reward. Similar to Lava World experiments, we employ curiosity-driven model to collect experience. With PATH function learned during exploration, the agents showed faster



(a) JourneyEscape (b) Riverraid

learning speed on the given task. However, their performance (score) is typically worse than the A3C model. In both tasks, we find that the agents are able to control themselves, but fail to effectively avoid colliding with objects which will lower the score. In path training, because of the absence of reward signal, the PATH function fails to learn how to avoid certain "bad" objects.

F SPEC OF NEW TASKS DEFINED OVER ATARI GAMES

New tasks for task transfer from single source

[1] Riverraid: Shooting boat will now get penalty, instead of getting reward.

[2] UpNDown: Live longer, but try not to squash other cars (in original game, squashing cars gets reward).

[3] Alien: Try to find the guard and dies (in original game, you need to avoid meeting any guards).

tasks for task transfer from multiple sources

[1.1] Riverraid-bad-boat: Shooting boat will now get penalty, instead of getting reward.

[1.2] Riverraid-bad-helicopter: Shooting helicopter will now get penalty, instead of getting reward.

[1.3] Riverraid-bad-fuel: Shooting fuel will now get penalty, instead of getting reward.

[1.4] Riverraid-original: Original task.

[2.1] JourneyEscape-good-heart: Touching red-heart shaped objects will get reward, instead of get penalty.

[2.2] JourneyEscape-good-ball: Touching yellow-ball shaped objects will get reward, instead of get penalty.

[2.3] JourneyEscape-good-heart: Touching head shaped objects will get reward, instead of get penalty.

[2.4] JourneyEscape-original: Original task.

We choose [1.1]Riverraid-bad-boat and [2.1]JourneyEscape-good-heart as our target task, and PATH function is pre-trained with demonstration from other three tasks. The τ function is fine-tuned in the task [1.3] Riverraid-bad-fuel and [2.4] JourneyEscape-original. And we choose these two tasks as source task for a3c. Then we fine-tuned τ function in target task and get the result.