Action Embedding for Transfer Reinforcement Learning

Yu Chen$^1$ *, Yingfeng Chen$^1$ *, Zhipeng Hu$^2$, Tianpei Yang$^3$, Changjie Fan$^1$, Yang Yu$^4$, Jianye Hao$^3$

$^1$NetEase Fuxi AI Lab
$^2$Zhejiang University
$^3$Tianjin University
$^4$Nanjing University

Presenters: Emma Yu, Stefan Ivanovic, Bochao Li
Agenda

- Background: Transfer Reinforcement Learning
- Related Works
- Motivation
- TRACE
- Experiments
- Future Work
Transfer Reinforcement Learning

- An agent exists in an environment with states and takes actions from a set of actions in order to maximize its reward.
- The agent learns to select which action to take depending on the state.
- One can apply transfer learning to share information in the model across multiple environments.
Related Works

● Same state and action space
  ○ Policy distillation: [Teh et al., 2017]
  ○ Meta-learning/advanced training method: [Finn et al., 2017], [Barreto et al., 2019], [Ma et al., 2018]

● Different state and action space
  ○ State space mapping: [Gupta et al., 2017], [Wulfmeier et al., 2017], [Wan et al., 2020], [Liu et al., 2019], [Raiman et al., 2019]

● Inter-task mapping
  ○ [Taylor et al., 2007], [Ammar et al., 2015], and [Zhang et al., 2021]

● Action embedding
  ○ [Whitney et al., 2020] and [Jain et al., 2020]
Motivation

- The motivating example in this paper is video games in which different characters can take different actions in the same environment.
- Each character may have many different actions they can take such as different attacks and abilities, however, these actions share underlying similarities.
- For example, many actions for many characters may be similar to “do a small attack quickly” or “do a large but costly attack”.

![Image of abilities in video games]
Problem Definition

● Given an Markov Decision Process, the goal of the agent is to find an optimal policy $\pi^*$ that maximizes the expected discounted return $R = \sum_{t=0}^{\infty} \gamma^t r_t$

● Consider the transfer problem between a source MDP $M_S = (S_S, A_S, T_S, R_S, \gamma_S)$ and a target MDP $M_T = (S_T, A_T, T_T, R_T, \gamma_T)$
  ○ Assume different state and action spaces, but similarities in the reward functions and the transition functions
  ○ Tying this back into the video game example:
    ■ The source and target MDP correspond to two different roles in an MMORPG, with the same goal of defeating the boss and the same reward of the result of the battle (whether it be a win, a loss, or a tie).
Action Embeddings

- The purpose of actions is to effect the environment and gain rewards via affecting the environment.
- Action embeddings should be sufficiently descriptive that one can determine the effect on the environment of an action based on its embedding.
- Therefore, a transition model is used to predict the next state given the action embedding and the current state. This model is then trained together with the action embedding vectors.
Transition Model Details

- The transition model is split into two functions.
- The first function inputs the state and action embedding and then outputs a mean and variance for a latent representation of the state action pair.
- A latent variable is then generated from a Gaussian with that mean and variance. The second function inputs the state, action embedding, and latent variable and outputs the predicted next state.
- This approach is inspired by variational autoencoders and allows for randomness in the predicted state.

\[
\mathcal{L}(\theta^D, W^{ae}) = \mathbb{E}_{s_t, a_t, s_{t+1}} \left[ \| \tilde{s}_{t+1} - s_{t+1} \|^2_2 \right] + \beta D_{KL}(\mathcal{N}(\mu_t, \sigma_t) \parallel \mathcal{N}(0, I))
\]
Training Policy and Selecting Actions

- The policy is trained to input states and output vectors in the space of action embeddings.
- The actual action selected is the action whose embedding is closest to the policies output vector.
- An actor critic system is used to learn the expected value for different actions and thus to learn which actions to take.
TRACE Transfer

- **Same-Domain Transfer:**
  - Same state, but different action spaces
  - Allows for direct transfer of transfer policy and fine-tuning target task

- **Cross-Domain Transfer:**
  - Different state and action spaces, so transition model cannot be reused
  - Or different reward
Experiments

This paper implements three experiments:

- Simple gridworld game (same domain)
- Mujoco and Roboschool build-in control problem (same domain and cross domain)
- Real World video game experiment. (more practical cross domain problem)
Experiments

- TRACE: no transfer on target task
- TRACE-P only transfer policy model
- TRACE-T only transfer transition model
- TRACE-PT paper’s method, transfer both.
- SAC direct soft actor-critic in target task.
- BT directly apply source task model to target task
- MIKT [Wan et al., 2020] state transition transfer

Figure 4: Experiment results on gridworld environments. The solid lines denote our method, and the dashed lines represent the compared methods. The shaded areas are bootstrapped 95% confidence intervals.
Experiments: N-Step Gridworld

(a) task $n = 1$ (source task $n = 3$)   (b) task $n = 2$ (source task $n = 1$)   (c) task $n = 3$ (source task $n = 2$)

Figure 4: Experiment results on gridworld environments. The solid lines denote our method, and the dashed lines represent the compared methods. The shaded areas are bootstrapped 95% confidence intervals.

- TRACE perform worst on Gridworld, and TRACE-T is nothing better than that (transition model is easy to learn for simple problem)
- other algorithms works well on all three problems, but for 2->3 case, there is larger gap
- old algorithm doesn’t capture action embedding information, and it is easy to transfer from larger action space to smaller one
- compare with TRACE-P, TRACE-PT have lower variance.
Experiments: N-Step Gridworld

- Actions embeddings: $e(↑↑←) + e(↑←→) - e(←→←) ≈ e(↑↑↑)$
- $e(↑) + e(→) ≈ e(↑→) ≈ e(→↑)$
Experiments: Mujoco and Roboschool

- mujoco r-roboschool P- pendulum DP-double pendulum, discrete action space

Roboschool is more complex task. (more state, action, reward information)

- same domain transfer: all algorithm works relatively well. And smaller gap in roboschool. (TRACE is better at observing task environment information)
- cross domain: TRACE-type algorithms perform much better
- (for cross domain, simply transfer or station embedding is not enough)
- again easy to learn from larger action space to smaller one
Experiments: Commercial Game Combat

NetEase 1v1 combat where the agent can have different skill sets to fight a build-in opponent

- Two classes: *She Shou* (SS) and *Fang Shi* (FS)

Figure 6: PCA projections of learned action embeddings. Each dot represents an action embedding of skill in the game.
Experiments: Commercial Game Combat

- Transfer in this commercial game is actually easier than the pendulum problem. (because the combat game have similar reward function).
Summary

- Learn meaningful action embeddings by training a transition model
- Train RL policies with action embeddings
- Quick adaptation of policy
- Significantly improves sample efficiency, even with more challenging transfer tasks (different state and action spaces)
Future Work

- Extend method to continuous action spaces
- Align the state embeddings with additional restrictions
- Transfer reinforcement learning in meta-learning
- More real world application of transfer reinforcement learning
Q&A

Thanks for listening!