DON'T FREEZE YOUR EMBEDDING: LESSONS FROM POLICY FINETUNING IN ENVIRONMENT TRANSFER

Anonymous authors
Paper under double-blind review

ABSTRACT

A common occurrence in reinforcement learning (RL) research is making use of a pretrained vision stack that converts image observations to latent vectors. Using a visual embedding in this way leaves open questions, though: should the vision stack be updated with the policy? In this work, we evaluate the effectiveness of such decisions in RL transfer settings. We introduce policy update formulations for use after pretraining in a different environment and analyze the performance of such formulations. Through this evaluation, we also detail emergent metrics of benchmark suites and present results on Atari and AndroidEnv.

1 INTRODUCTION

Imagine an agent completing a task when the background changes color. The underlying task is identical; only the observation space has changed. One might wonder: in this setting, might we want to update the embedding network while keeping the policy the same? This is counter to a common strategy of freezing the embedding while finetuning the policy head.

Much work in RL uses a pretrained visual embedding network, which may or may not be finetuned during policy learning. In this work, we aim to evaluate the effectiveness of these paths. To this end, we present a transfer framework that allows for evaluation of zero-shot generalization, embedding finetuning (frozen policy head), head finetuning (frozen embedding), and full policy finetuning. Comparing these approaches gives us a better understanding of how an agent makes use of the different parts of its policy network.

The effectiveness of these paths depends on environment similarity at transfer time. Exploiting this reasoning, we introduce environment metrics that evaluate an environment along multiple dimensions based on finetuning success. We present such metrics for Atari flavors in the Arcade Learning Environment (Machado et al., 2018) and task variants in AndroidEnv (Toyama et al., 2021).

2 RELATED WORK

2.1 MULTITASK TESTBEDS IN RL

In recent years, researchers have identified a growing need for better evaluation in RL (Machado et al., 2018; Henderson et al., 2018). One such dimension is a generalization component of RL testbeds. Machado et al. (2018) examined determinism in Atari environments and proposed the addition of sticky actions and variants on Atari games they call flavors. Others have created RL testbeds with generalization in mind using procedural generation of environments (Chevalier-Boisvert et al., 2018; Cobbe et al., 2020). Multitask benchmarks have also cropped up in robot learning (Yu et al., 2020b). In our work, we make use of the Arcade Learning Environment as formulated by Machado et al. (2018). We also introduce task variants for transfer in AndroidEnv (Toyama et al., 2021).

2.2 MULTITASK METHODS

An entire body of research is dedicated to multitask learning, which approaches ranging from metalearning (Finn et al., 2017) to gradient surgery (Yu et al., 2020a). Many such approaches are not necessarily specific to RL and can be applied to a range of supervised tasks like vision. Hierarchical

approaches that can exploit common structures across tasks are also well-suited to multitask learning (Sutton et al., 1999). Our work, in contrast, does not optimize for multiple tasks at the same time; tasks are seen sequentially, and there is a clear target task in mind at any given point in training.

2.3 Task Transfer as an Evaluation Protocol

Some recent work has created evaluation protocols for task transfer. Farebrother et al. (2018) looks at DQN generalization with experiments on pretraining and transfer between Atari game flavors. For consistency, we use the same set of flavors in our Atari experiments, though our focus is on types of policy updates rather than generalization techniques like dropout and regularization.

Parisi et al. (2021) also crafts a transfer evaluation framework in RL. The framework involves pretraining a self-supervised exploration policy which is used to bias actions of a task policy trained from scratch at transfer time. The framework is implemented in 2D navigation settings (MiniGrid and AI Habitat). Our work, in contrast, trains a single policy using extrinsic rewards across pretraining and transfer, and we present results in more dynamic settings (Atari and AndroidEnv).

3 Method

Our approach has two primary components: first, a transfer framework for training a policy (Section 3.1), and second, a policy update formulation used after transfer (Section 3.2).

3.1 Training

Training consists of 2 phases. First, pretrain an RL agent in the default version of an environment. Second, put the agent in a different flavor of the environment while training with modified policy updates (described in Section 3.2). This sequence of events yields a number of evaluation points, from random to zero-shot generalization to finetuning. Such evaluation is beneficial for understanding the performance not only of the agent but also of the environment itself. We use IMPALA (Espeholt et al., 2018) for our RL agent, which enables the decoupling of action and evaluation.

3.2 Policy Updates

We apply a set of modified policy updates during transfer time. As depicted in Figure 1, we split the IMPALA policy network into 2 parts: an embedding and a policy head. The embedding is a deep stack of convolutional layers which takes in an image observation and outputs a 256-dimensional vector. The head represents the temporal and action aspects of the agent, consisting of an LSTM (Hochreiter & Schmidhuber, 1997) and fully connected layer before producing a policy and value function. The architecture is based on the large network from Espeholt et al. (2018), which contains 15 convolutional layers and 1.6 million parameters. Our only change is the addition of a fully-connected layer to the head after the LSTM.

At transfer, we activate our modified policy updates, which independently learn or freeze each model component. This yields 4 formulations (freeze or not freeze head, freeze or not freeze embedding), one of which is trivial (freeze entire policy, zero-shot evaluation).

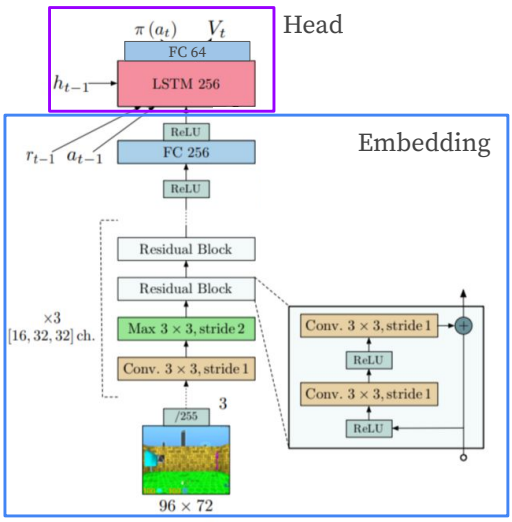

Figure 1: **IMPALA policy network components (Espeholt et al., 2018).** At transfer time, we independently update or freeze these components.

4 EXPERIMENTS

In our experiments, we set out to answer 2 overarching questions:

- How does freezing the policy embedding affect performance, on both the current environment flavor and the pretrained one?
- What kind of generalization are Atari game flavors and AndroidEnv variants testing?

The first question is agent-centric, while the second is environment-centric. In this section, we present experiments tackling these areas on Atari (Section 4.1) and AndroidEnv (Section 4.2). All experiments are run with 5 random seeds, and plots show mean and confidence across these seeds.

4.1 ATARI EXPERIMENTS

In our Atari experiments, we use the set of games and flavors from Farebrother et al. (2018), comprised of 9 transfer tasks across 4 games (Figure 2). Experiments on the game Freeway, in line with prior work, did not yield significant learning due to the sparsity of the reward, so we focus our analysis on the 3 remaining games. We pretrain the agent on the default game flavor (mode 0 difficulty 0, abbreviated m0d0) for 50M steps, followed by an additional 50M steps of training in another flavor.

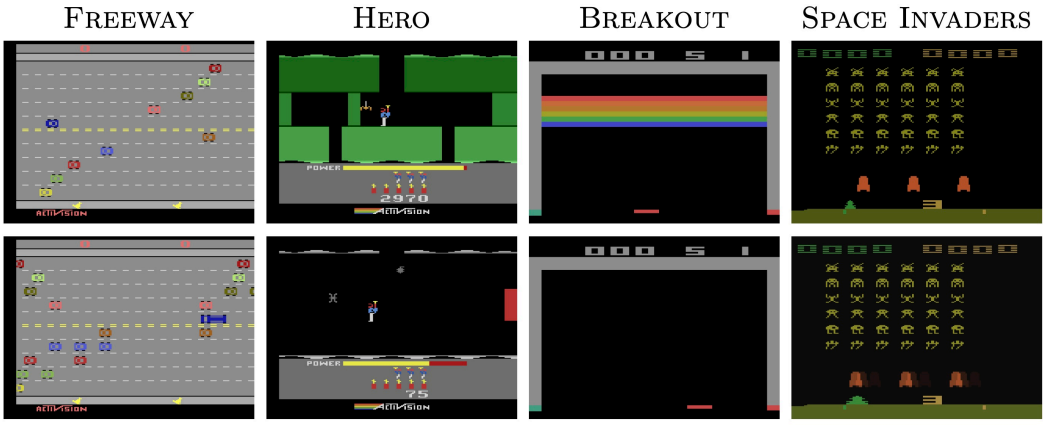

Figure 2: **Example Atari game flavors.** Visualizations borrowed from Farebrother et al. (2018). The top row shows default flavors used for pretraining and the bottom row shows variants seen during transfer: Freeway's mode 1 adds buses, more vehicles, and increases velocity; Hero's mode 1 starts the agent at level 5; Breakout's mode 12 hides all bricks unless the ball has recently collided with a brick; Space Invader's difficulty 1 widens the agent's spaceship and oscillates the red shields.

4.1.1 ATARI TRANSFER RESULTS

Figure 3 shows reward curves after transfer to a new Atari flavor. Updating the entire policy performs best while freezing the entire policy shows the fixed performance after pretraining. The most interesting finding from these results is that across the 3 Space Invaders flavor transfers, freezing the policy head performs *better* than freezing the embedding. Freezing the embedding is commonly seen in RL, especially in settings where we expect the observation space to be similar. From Figure 2 right, Space Invaders difficulties 0 and 1 appear to have similar observations (only the size of the agent differs). This makes it even more surprising that updating the embedding is so crucial to performance (*especially* in difficulty 1 transfer, Figure 3 bottom left). From this result, we hypothesize that the embedding is learning task-centric representations and conclude that RL researchers should be more wary of freezing an embedding network.

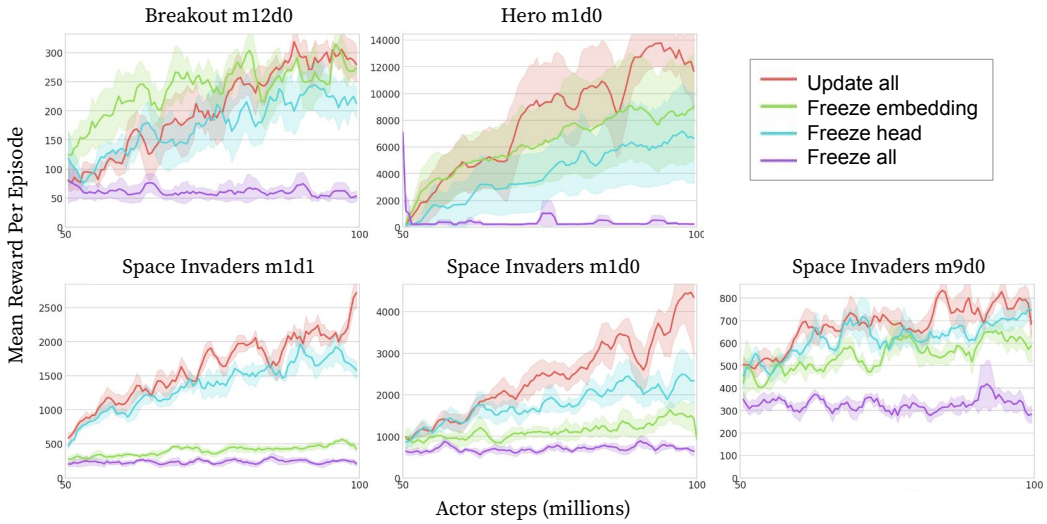

Figure 3: **Atari transfer reward curves.** Mean episode reward in flavor 2 after 50M actor steps of pretraining in flavor 1. Unsurprisingly, updating the entire policy (red) performs best. Surprisingly, freezing the embedding (green) yields lower performance than freezing the policy head (blue) across the 3 Space Invaders flavors. Freezing the entire policy (purple), by definition, does not improve.

4.1.2 Atari Environment Flavor Metrics

Comparing the policy update approaches yields a general framework for environment difficulty and similarity scores. We compare episode rewards from 5 points: 1) trained from scratch on target flavor for 50M actor steps, 2) random policy, 3) pretrain then freeze the embedding, 4) pretrain then freeze the head, and 5) pretrain then update everything. In Table 1, we display 2-5), all normalized by 1). Freeway agents rarely yield nonzero rewards, so the Freeway metrics do not shed much light.

The normalized random policy reward is a metric for environment difficulty. Random policies on the Breakout and Hero flavors receive 2% of the rewards of the policy trained on the target; random policies on Space Invaders receive 10-30% (suggesting that Space Invaders is an easier game).

The normalized pretrain then update everything ('update all') is a metric for flavor similarity – if the flavors are more similar, then pretraining should provide a boost in target performance. Pretraining provides a 2-4x boost in performance over training from scratch, with the exception of Hero Mode 2 – here pretraining actually *hurts* performance.

The normalized metrics for pretraining and freezing either the embedding or head tell us *how* two environments are similar or different. There are 2 distinct findings here. First, as seen on Hero mode 1, freezing the embedding is twice as effective as freezing the head. Second, on Space Invaders difficulty 1, freezing the *head* is three times as effective as freezing the embedding.

We present these metrics not only to illustrate our particular results but also as a general framework for evaluating multitask and transfer benchmarks. These metrics could simply be computed once upon release of a benchmark to show what kinds of challenges it presents.

Table 1: Atari environment metrics

	Breakout	**Freeway**			**Hero**		**Space Invaders**		
	m12d0	m1d1	m1d0	m4d0	m1d0	m2d0	m1d1	m1d0	m9d0
Random policy	0.026	0.0	0.0	0.0	0.020	0.024	0.119	0.141	0.282
Freeze embed	2.974	0.0	0.0	0.0	3.017	0.892	0.524	1.508	1.111
Freeze head	2.846	0.0	0.0	NaN	1.631	0.654	1.845	2.641	1.404
Update all	4.418	0.0	0.0	0.0	4.188	0.814	2.923	3.967	1.644

4.2 ANDROIDENV EXPERIMENTS

We also evaluate our policy learning strategies and evaluation framework on AndroidEnv, a new testbed that is challenging for RL due to its lack of determinism and complex action space. We present results on the Catch task as depicted in Figure 4. We introduce 4 variants of this task as flavors. We use an action wrapper that discretizes the pointer location into a 9x6 grid, as done in prior work. The image observations are downsampled by 4x in x and y for a resolution of 120x80. We run IMPALA with 5 actors.

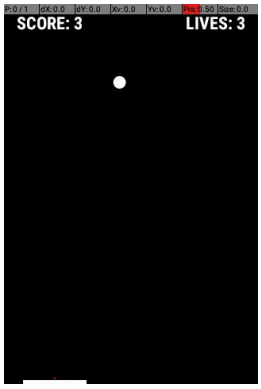

Figure 4: **AndroidEnv Catch.** We introduce 4 variants: large and small ball, small paddle, and red background.

4.2.1 ANDROIDENV TRANSFER RESULTS

Figure 5 shows the reward curves during transfer to Catch variants after 5M steps of pretraining in the default flavor. Reinforcing a finding from the Atari experiments, on the large ball variant (Figure 5 left), freezing the embedding performs *significantly worse* than freezing the policy head or updating the policy altogether. In fact, freezing the policy head does not seem to degrade performance at all.

Other variants do not show as much distinction between policy update strategies. We hypothesize that this is due to the simplicity of the Catch task and the binary nature of its reward. In future work, we hope to expand these experiments to more nuanced tasks in AndroidEnv.

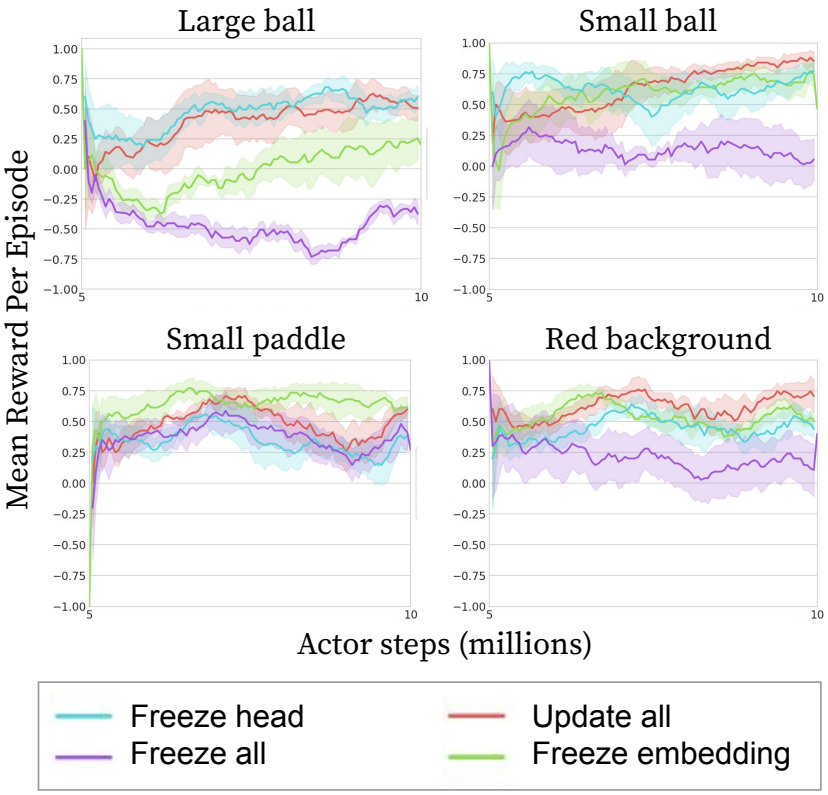

Figure 5: **AndroidEnv transfer reward curves.** Mean episode reward in during training in a Catch variant after 5M actor steps of pretraining in the default environment. The large ball results mirror Atari experiments: freezing the embedding yields lower performance than freezing the policy head.

4.2.2 ANDROIDENV ENVIRONMENT FLAVOR METRICS

Table 2 shows the metrics for AndroidEnv across the 4 Catch flavors. Note that since the catch rewards can be positive and negative, we do not normalize the metrics as we did for Atari flavors. Each number is the mean episode reward for that strategy, averaged over 100 episodes from that point in training. 'Freeze all' shows the initial performance of the pretrained policy right after transfer, which gives us a similarity metric: 'small paddle' does almost as all initially as after finetuning (the other flavors yield lower initial performance). Updating all policy parameters performs well except in the large ball flavor, where freezing the head surprisingly performs the best.

Table 2: AndroidEnv environment metrics

	Large ball	Small ball	Small paddle	Red background
Freeze embed	0.348	0.768	0.664	0.432
Freeze head	0.524	0.832	0.436	0.436
Update all	0.328	0.784	0.636	0.688
Freeze all	-0.444	0.064	0.444	0.100

5 DISCUSSION

Our approach has two goals: first, to analyze agent effectiveness in cases where part of the policy is frozen, and second, to analyze the underlying environment testbeds. Notably, our results show that freezing a policy embedding sometimes performs *worse* than freezing the policy head, opening doors for future work to explore the role of a policy embedding. A clear takeaway from this result is that researchers should think twice before using a pretrained embedding without updating it in concert with the rest of the policy. Next, on the environment-centric side of our analysis, we presented a collection of metrics to analyze environment flavors, their difficulties, and their similarities to one another. Such metrics hold promise for designers of future benchmarks or anyone hoping to gain a better understanding of existing environments. In a field struggling with sample efficiency as we move towards more complex applications, RL researchers must increase focus on generalization and finetuning to make these problems tractable; our work makes simple steps towards these goals.

REFERENCES

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pp. 1407–1416. PMLR, 2018.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Simone Parisi, Victoria Dean, Deepak Pathak, and Abhinav Gupta. Interesting object, curious agent: Learning task-agnostic exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020a.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100. PMLR, 2020b.