Level-Up Learning: Transfer Learning-based Policy Generalization in Platforming Video Games

Joseph Kawiecki   John Ramthun

Abstract

Reinforcement learning (RL) has shown success in a wide variety of applications including game strategy, robotics, recommendation systems, healthcare, and more. However, most solutions for these applications require specific behavior policies for each narrow use. As RL continues to grow, this constraint of training-specific policies becomes a real hindrance. It would be beneficial for a more general approach to be able to transfer policies between similar experiences and tasks to other agents. In this paper, we investigate this problem further, focusing on the Proximal Policy Optimization (PPO) update method for video game playing with the Super Mario Franchise. We have reviewed various state-of-the-art methods for transfer learning, applied them to the PPO algorithm, and conducted experiments. Our results have shown that while beneficial for other algorithms, current approaches to transfer learning do not work well for PPO for video game play.

1. Introduction

Despite the success of RL, most methods have been traditionally developed with a specific application in mind. For example, training a robot arm to pick up a cup requires training a policy specific to that action, robot, and even environment. Such a policy is unlikely to perform well in the execution of a similar action, such as picking up a plate, on the same robot. Or that the original policy may not perform well when executed on a different robot of the same general type such as a different robot arm. Thus as the use cases and complexity of RL grow, learning to develop policies that are better able to generalize, and thus transfer prior knowledge, to similar yet new situations will be essential. Transferring prior knowledge can provide a variety of benefits such as more efficient development, generalization by building varied experiences, and likely better real-world performance results. In this paper, we explore, implement, and analyze a few methods in an attempt to accomplish task transfer learning utilizing the PPO update within the Super Mario video game franchise. Specifically, the end goal was to train an agent on a small set of levels and be able to learn enough to leverage those prior experiences to perform reasonably well on other, unseen levels. Specifically, we split our project into four overarching segments. The first was to train the agent and achieve reasonable performance on one level. Figure 1 shows training on Level 1-1. Second was to take the agent from one level and test its model on another. The third was to show the reasonable performance of the agent across multiple levels and of different worlds within the Super Mario Brothers game. Finally, the fourth step was to demonstrate the capable performance of the agent across multiple games of a similar style such as Super Mario Brothers to Super Mario World yet still within the Super Mario franchise. While we knew that given the timeframe, attaining some of the latter steps would be difficult and might be best suited for future works, we aimed to accomplish all that we could one step at a time. We chose to narrow our scope to the PPO algorithm specifically as it is very commonly used in a wide array of domains. However, we also conducted minor experimentation on other algorithms such as Deep Q-learning (DQN) and Advantage Actor-Critic (A2C). While the results for our methods produced mixed results, we view this as a good stepping stone for future work.

Figure 1. Agent (Mario) training on Level 1-1 within NES emulator.

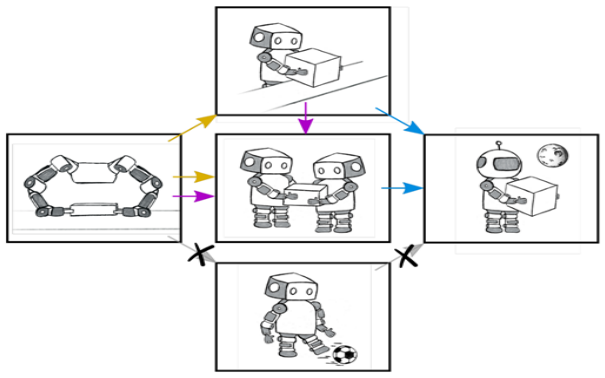

Figure 2. Illustration of transfer learning in robotics. A robot can store and leverage its experience performing a specific task in a particular environment to improve learning on a related task in a related environment. For example, the downward purple arrow portrays a robot using its gained knowledge from lifting a box to aid in teaching two robots to exchange a box with each other. This is known as task transfer as shown in purple. The blue arrows demonstrate environment transfer while the yellow arrows indicate agent transfer. A successful transfer usually at least requires some commonality between the source and target robot, task, and environments. The crossed-out arrows indicate a task is likely not transferable. A robot kicking a ball is likely not to have much useful experience in teaching a robot to carry a box in space.

The following paper further details various approaches to attempt transfer, experimentation, results, and finally conclusions. Our primary contributions are to build an adapted PPO algorithm to run within NES emulator games through OpenAI's Gym Retro [5], execute experiments for transfer learning with the PPO algorithm, and finally analyze results.

2. Background

2.1. Transfer Learning

Within video games, constructing a universal controller able to transfer knowledge and experiences can be categorized as environment, task, or agent transfer. Environment transfer is the changing of the environmental complexity or conditions where the task is being executed. A large focus within this domain is the transfer from simulation to real world and vice versa. Task transfer is the idea of transferring learned behavior from one robot to another and is often referred to as generalization. Agent transfer is similar to both yet involves transferring to different agents that may have different characteristics or features than the original. Figure 2 visually summarizes the domains of transfer learning within robotics.

2.2. Reinforcement Learning Basics

RL is a subfield of machine learning in which an agent will learn to make decisions by interacting with an environment. As it relates to video games, this can be especially useful in teaching complex behaviors. A reinforcement learning problem is comprised of a learning agent and the environment in which it interacts. At a particular timestep t, an agent will observe its environmental state s_t, determine an action a_t, and receive feedback based on that action in the form of an environment reward r_t. Over time, the agents may learn a policy π that dictates a strategy mapping states to actions to achieve the optimal reward. Despite being widely used in practice, RL agents can have a high computational complexity to learn even simple behaviors. Thus it would be ideal to learn more generalized approaches to new tasks and environments.

2.3. Q-learning

Q-learning is a model-free, off-policy algorithm to find the optimal policy for an agent in a given environment. It does not require knowledge of the transition and reward functions and instead learns an optimal policy directly from trial and error. The $\mathcal{Q}$ component, indicating quality, quantifies how valuable an action to a particular state may be in terms of future estimated rewards. The $\mathcal{Q}$ update is shown in (1). Q-learning is off-policy in that it may evaluate and update a policy different from the policy used to take action. Specifically, the Q-learning algorithm will either select an action with the highest $\mathcal{Q}$ value or a random action based on ϵ probability. This allows exploration of the environment and potentially learning to new state-action-pairs that could lead to higher rewards.

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)] \tag{1}$$

2.4. Deep Q-learning

Despite the success of Q-learning, one of the main drawbacks is that it requires using a Q-table to store the $\mathcal{Q}$ values of the state-action mappings. Thus, for significant state spaces, this quickly becomes infeasible as the table grows exponentially. Thus one alternative is using deep neural networks for Q-learning known as Deep Q-learning (DQN). DQNs were introduced in 2013 by DeepMind to play Atari games [4].

DQNs build on Q-learning by using neural networks to approximate the Q-values for each state-action-pair essentially replacing the Q-table with a neural network. DQNs also utilize a replay memory buffer to store past experiences. Each experience is usually comprised of a state, action, reward, and next state. This replay buffer can average a behavior distribution over many previous states to smooth learning

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize target function $\mathcal{Q}$ with weights $\theta^- = \theta$
for episode = $1, M$ do
    Initialize $s_1 = x_1$, preprocess $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With $\Pr(\epsilon)$ select random action $a_t$
        else $a_t = \max_a \mathcal{Q}^*(\phi(s_t), a; \theta)$
        Execute $a_t$, observe reward $r_t$, image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$, preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random transition batch $(\phi_t, a_t, r_t, \phi_{t+1})$ from $\mathcal{D}$
        $y_j = \begin{cases} r_j & \text{step is } j+1 \\ r_j + \gamma \max_{a'} \hat{\mathcal{Q}}(\phi_{j+1}, a'; \theta) & \text{else} \end{cases}$
        Perform gradient descent update step on $(y_j - \mathcal{Q}(\phi_j, a_j; \theta))^2$ with respect to network parameters $\theta$
        Every C steps reset $\hat{\mathcal{Q}} = \mathcal{Q}$
    end for
end for
```

and avoid oscillations or divergence in parameters. In the DQN algorithm 1 below from the landmark paper [4], the authors use a static replay buffer size to store the N last experiences with equal importance. In their approach, the authors perform experiments on an Atari 2600 emulator. The DQN was shown to outperform previous approaches in six of seven games tested and beat human experts in three of seven.

2.5. Advantage Actor-Critic

Actor-critic methods are a family of update methods that use an actor to control how the agent behaves and a critic that measures how good the action taken is. While the actor tries some actions, the critic will provide feedback. The actor then updates its policy to improve on this feedback. Thus, the Actor-Critic combines policy (actor) that controls how good the agent is and a value (critic) function to measure how good the action taken is.

Advantage Actor-Critic (A2C) is one Actor-Critic method that uses an advantage function instead of the value function for the critic. An advantage function (2) calculates how much taking that action at that state is compared relative to the average value of the state. This means the advantage function determines how much extra reward we will get by taking this action at this state relative to the mean reward at the state.

$$A(s, a) = Q(s, a) - V(s) \tag{2}$$

2.6. PPO

Our project involved specifically studying task transfer learning following the Proximal Policy Optimization (PPO) algorithm [7]. The idea is to estimate an advantage score (2). This advantage score can then be combined with a ratio of old to new policies to determine the update. However, the main insight of PPO relative to previous policy update algorithms and why it is relatively stable is due to the clipping of this gradient update. Clipping the advantage and policy ratio multiplication ensures each particular update is not too drastic. PPO is widely used in practice offering a good balance between efficiently achieving an optimal policy with stability due to clipping. Algorithm 2 demonstrates PPO with a clipped gradient update.

2.7. Super Mario Bros as an MDP

Our project uses Super Mario Bros for the NES as the learning environment. The MDP (with reward) that represents this environment is as follows. The state space consists of each frame to be seen by the agent. Given the implementation, this will be every fourth frame generated by the emulator on every level for every trajectory. The action space is classified as a filtered "multi-binary" space by Gym Retro. The options within this space represent each button on an NES controller used by the game: 'A', 'B', 'Up', 'Down', 'Left', and 'Right'. The agent can press any combination of the 'A', 'B', and directional buttons, including none. The initial state distribution is the first frame of the spawn location at the beginning of each level. Thus, there are 32 possible initial states, with only eight having a probability greater than zero based on our implementation. The transition function is determined by the combination of states (frames) and action (controller inputs) pairs. The reward function is based on the agent's position, the game score, the game time, and whether the agent is using a warp pipe. We decided not to use lives in the reward function as the agent learned self-destructive policies that increased the reward. We discuss the reward in greater detail in the methodology section. Lastly, the discount factor is 0.99, as defined by Stable Baselines 3. Together, this MDP can represent every interaction between the agent and the game (environment).

3. Methodology

To show the results of transfer learning, we first establish a consistent and effective training environment. Next, we establish baseline agents that determine the expected performance without any transfer learning. We also train new agents using two multi-task learning approaches, giving an agent access to all of the test levels for shorter periods. Finally, we train select baseline models on new levels using policy transfer.

Algorithm 2 PPO with Clipped Gradient

```
**Input:** initial policy parameters $\theta_0$, clipping threshold $\epsilon$
**for** $k = 0, 1, 2, ...$ **do**
    1. Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
    2. Estimate advantages $\hat{A}_t^{\pi\theta_k}$ using any advantage estimation algorithm
    3. Compute policy update
       $$\theta_{k+1} = \arg\max_{\theta} \mathbb{E}_{\tau\sim\pi_k}\left[\mathcal{L}_{\theta_k}^{CLIP}(\theta)\right]$$
    by taking K steps of (minibatch) SGD
    4. $\mathcal{L}_{\theta_k}^{CLIP}(\theta)$ is
       $$= \min\left(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}\right)$$
       $$r_t(\theta) = \frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}$$
**end for**
```

3.1. Getting Started

The environment used for training was created using OpenAI's Gym Retro, which allows users to create Gym environments of emulated games for RL. We use NES Super Mario Bros as our game of choice. We selected PPO to train our agents due to its performance in initial testing and wide general use. We implemented wrappers on top of the environment to make it easier to train the model. We use a custom time-limiting wrapper that limits the number of timesteps allowed per epoch. We also use Stable Baselines 3's MaxAndSkipEnv wrapper, which returns only every fourth frame, giving the agent more time between decisions. Then, we utilize the SubprocVecEnv function to allow multiple instances of our environment to run in parallel, accelerating training by the number of spawned processes. VecMonitor allows these multiple instances to be appropriately joined after 10 epochs to ensure the model learns from every instance. Brief experimentation revealed that the default model architecture for the Actor-Critic network was sufficient to learn in the above environment. We also applied a piecewise learning rate schedule that allows the model to finetune performance as exploration ends.

We modified Retro itself to make it more suitable for our testing. Firstly, the provided reward function was too simplistic. We could very clearly see that the model would learn to easily exploit the reward function and would not play like a human player. Falling into a certain pit on Level 1-1 twice would boost the reward by about 400 points. To close this exploit, we modified this line to only reward the agent for moving to the right and terminated episodes after losing one life. We did not punish the agent for moving left, as this may be required. To encourage the agent to behave more naturally, we also reward it based on the game score and punish it for each passing second. This should encourage the agent to collect coins and power-ups, jump on enemies, and find faster ways to complete levels. We also found that the agent would sometimes enter reward rooms full of coins. Since the agent knew to move to the right, it would maximize its reward in the room and become stuck. To counter this, we added an additional reward when the agent entered an L-shaped warp pipe, allowing the agent to proceed through the level. This last reward required adding the appropriate memory address and data format to Retro's list.

Finally, to support our brief multi-task learning approaches, we again modified Retro to support random stage selection. When enabled, the agent would complete an episode normally. Upon termination, the environment would provide the agent with an entirely new state space.

3.2. Base Models

Once we established our environment and model configurations, we could train agents to play the game. The first agent was trained on Level 1-1 to ensure that our decisions were viable. We consider Level 1-1 as a viable testing ground as it was designed to let human players learn how to play the game. As such, if an agent can learn to play Level 1-1, it should be able to understand any future mechanics of the game. Next, we trained several additional agents on other levels to establish baseline performance for various levels. Then, we trained a third type of agent with random level selection based on our modifications to Retro. Finally, we trained a fourth type of agent on all eight available levels in every episode. Please note that while the game itself has 32 levels, we only had access to save states for the first level of each world.

3.3. Transfer Learning

To perform transfer learning, we trained the first agent (trained on 1-1) again on select different levels using the same hyperparameters. We also modified the hyperparameters to see if a slower learning rate would affect performance.

We then trained a second set of agents. For the first, we assigned each process to one level, such that all eight levels were being played at one time. This would allow the model to consider a wider breadth of states. The agent would need to learn a general policy to succeed at any level. For the second, we modified the environment to change the starting state to a new level after each episode terminates. Since the model is run in parallel, each instance receives a level at random (sampled with replacement) and uses that for the next episode. This training also continues until the set limit.

4. Literature Review

4.1. Actor-Mimic

This work [6] published in 2016 presented an early attempt at transfer learning for Deep Reinforcement Learning (DRL). Specifically, the authors develop a method called Actor-Mimic that learns a single policy network for a set of distinct tasks by using guidance from several expert teaching policies. These expert policies are derived from a set of expert DQNs trained for the specific task. Actor-Mimic works by using a combination of a policy regression and a feature regression objective. The policy regression feature simply aggregates the Q-values of the expert predictions via a softmax function. The idea is that the student policy can focus on mimicking a particular action that was chosen by the guiding expert at each state. The feature regression objective is a further loss function to pressure the Actor-Mimic to compute features that predict the expert. Together, they are used to train a single, multi-task policy network for a small set of source games to play near expert levels. Once at this level, the authors demonstrate that Actor-Mimic can be used to pre-train the weights of a new DQN on a novel task and achieve improved performance versus random initialization. Specifically, of the seven game tasks evaluated in the Atari emulator, three of seven showed increased learning once pre-trained with a near-expert Actor-Mimic.

In terms of our project, we used a standard DRL model architecture that did not make use of expert policies. In a sense, our expert policy was successful training for the one level. However, we did implement pretraining for transferring learned task knowledge to new levels.

4.2. Schema Networks

In this work [3], the authors employ an approach to transfer learning by learning relationships. Specifically, the authors propose a method called Schema Networks that aims to learn the cause-and-effect relationship through backward reasoning. The method also uses object representations, the idea of detection of a bounded figure against an unbounded background, to help deal with uncertainty in causation. Combined, Schema Networks show promising results when evaluated on video games with slight variations in visual graphics. This is in contrast to other DRL approaches, such as A3C that do not perform well in such scenarios. The main idea behind the learning of the cause and effect relationship, or schema, is premised on backward iterating factor graphs with boolean logic relations to connect them.

While Schema Networks do show promising results, we decided to not go in this direction for the project. This was due to a few reasons including that the model architecture is complicated, it requires entity-based state representations to leverage the object representation, and is ideal for multi-task transfer. Particularly for the entity-based representation, this involved augmenting the input channels based on the particular objects in the game, further adding to complexity and data issues. During the author's experimentation, for example, this involved adding 30 additional channels to the input indicating the position of the game paddle parts with each channel corresponding to a single pixel.

4.3. Transfer Learning via Image-to-Image Translation

In this work [1], the authors demonstrate that methods at the time for transfer learning fail even due to small visual alterations. Within the domain of video gameplay, the authors add small, strictly visual (in that they don't impact the game flow) components such as rectangles or lines to the background. While this might be annoying if anything to a human player, these small changes rendered their agents unable to play anymore.

Thus the authors present a solution based on separating the visual and dynamics transfer. For visual transfer, they have developed a method to transfer images between tasks. This is accomplished by building a visual mapping between domains via Generative Adversarial Networks (GANs) [2].

This ties in closely with the robustness of the discriminator network when the level theme changes in games. Super Mario Bros has five level themes: normal, night, underground, underwater, and castle. Each theme has unique color palettes and mechanics. What human players might consider a minor difference, such as the color of the background changing from blue to black between normal and night levels, can be incapacitating for an agent. Separating visual recognition tasks from the actor-network may provide significant gains, as the policy may perform well in the new state space.

4.4. Transfer Learning in Deep Reinforcement Learning: A Survey

This paper [8] presents a recent survey of common methods concerning transfer reinforcement learning (TRL). The rise of RL has led to the rise of TRL to improve the efficiency and effectiveness of the learning process. In this paper, the authors present the need for TRL, analyze a few approaches, and compare them as well as their uses. In terms of approaches, the authors come up with a few broad categories largely based on what knowledge is transferred.

First is reward shaping. This refers to the group of approaches that focus on transferring knowledge by controlling the reward function of the target domain. Specifically, these methods utilize exterior knowledge to reconstruct the reward distribution of a target domain. This reconstructed distribution can then be used to guide an agent's policy update. The reconstruction will usually include the environment reward as well as a reward-shaping function to provide extra support in guidance. This allows for extra rewards to be in addition to the environment reward to favor particularly beneficial state-action pairs along a target trajectory.

Next is learning from demonstration that makes use of external demonstrations to transfer knowledge. The idea is for the demonstrations to contain some valuable knowledge that the target domain can leverage to learn from the source. These methods vary in their use cases for an offline or online approach. In an offline approach, demonstrations can be used for pre-training or warmup, for better parameter initialization. In the online setting, the demonstrations can be used directly to guide an agent's action to ensure efficient data exploration. Most approaches combine the offline training approach with the online methods. One limiting factor is that the demonstrations are largely dependent on the particular framework such as Q-learning, gradient-based methods, etc.

There are also policy transfer methods. The external transfer knowledge here is a set of pre-trained policies from one or multiple source domains. Oftentimes, there is a many-to-one approach involving training a set of N teacher policies on N source domains that a single student policy can leverage to learn. Within policy transfer, most approaches fall under policy distillation or policy reuse. Policy distillation refers to the scenario above in applying knowledge from multiple teacher policies to one student policy. Policy reuse is the direct reuse of a policy from source to target. The target policy is then assigned a weighted combination of the source domain policies that are set based on the expected performance gain in the target domain.

Many of the methods presented here are a bit advanced for our work. Our transfer learning attempts stem from pre-training and policy transfer. By pre-training, we provide our agent with weights that work for Level 1-1, which was designed to let human players learn the basic mechanics. Rather than expecting a player to learn from scratch further on, there is a steady learning process as progress is made. Pre-training accomplishes a similar goal. We also tested a one-to-one policy transfer, with the hope that the discriminator network learns only from the given level, rather than having a potentially poor initialization based on previous knowledge.

5. Outcomes

Through initial testing, we considered three model types for our agent: DQN, PPO, and A2C. In our initial environment, using the gym-super-mario-bros library, we found that DQN-based agents were the only options that seemed to perform remotely well. After more than 10 hours of training a PPO-based agent, the agent learned effectively nothing other than the basic controls. Agents would never stop jumping and would continually find pits to fall in or enemies to run into. We also found that training a DQN-based agent for longer than 30 minutes resulted in increasingly worse performance. As such, we decided to restart training using OpenAI's Gym Retro. However, the default state space is of the type "multi-binary," which allows agents to provide multiple, binary inputs at a time. In other words, the agent can press multiple buttons on a virtual controller at the same time. Stable Baselines 3's implementation of DQN does not allow for multi-binary inputs, so we excluded DQN from further training.

In the Retro-based environment, we tested both PPO and A2C-based agents. We found that on average, PPO-based agents performed the best across various training times, so we selected PPO for all further testing.

The next set of testing was to select all parameters and hyperparameters. We started with the configuration designed by ClarityCoders, who found that the default PPO implementation by Stable Baselines 3, a learning rate of 3e-5, selecting every fourth frame, and a maximum of 4500 time steps per epoch performed quite well. We performed basic A/B experimentation to validate each decision independently. Firstly, we considered various learning rates from 1e-3 to 1e-8 and found that 3e-5 was the optimal learning rate for this environment. Secondly, we tested different numbers of skipped frames: 1 (no frames are skipped), 2 (every other frame), and 4 (every fourth frame). We saw that the agent could adapt to various numbers of skipped frames, but it learned significantly faster when every fourth frame was used. Third, we tested the limit of time steps per epoch. We tested 2000, 3000, 4000, and 4500 time steps. We found that a lower limit of time steps makes initial training more productive, but limits the amount of learning that a model can do. A high limit makes initial training slow, which makes the agent less effective. Setting the limit to 3000 achieved a strong balance of initial and later performance, which was a clear choice. Fourthly, we wanted to test the network architecture of the Actor-Critic network, so we added one additional layer to both subnetworks (Stable Baselines 3's default architecture is two shared layers of 64 neurons, followed by two split networks of two-layer, 64-neuron MLPs) to see if that would allow the agent to learn deeper connections between states and actions. However, we found no meaningful result besides increased training time, so we left the network in the default configuration.

5.1. Base Model Experiments

After establishing our environment, model, and training configurations, we created five base agents for comparison. Each agent was trained on a different level: 1-1, 2-1, 3-

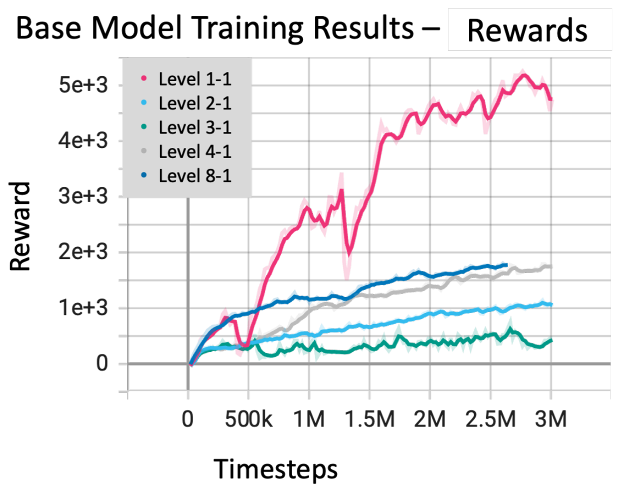

Figure 3. Base Model Training: Rewards

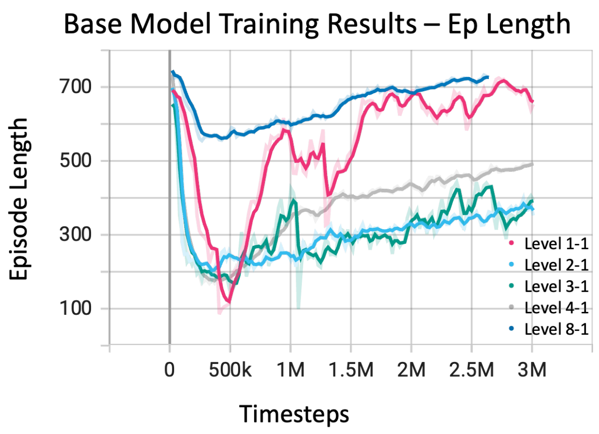

Figure 4. Base Model Training: Episode Lengths

1, 4-1, and 8-1. We consider the agent trained on 1-1 to be our base model for our transfer learning experiments. That is to say, we transferred the policy from this model since it exhibited the best performance of any model. For each experiment, we take the mean reward from each of the vectorized environments after 10 epochs.

After three million time steps, our base model for 1-1 obtained a reward of approximately 4500. The maximum reward seen was 5254. The learning was not consistently improving, but the reward clearly trended upward. The base models for 2-1, 4-1, and 8-1 also trend upwards, albeit at a slower rate. The base model for 3-1 sees nearly constant performance throughout the training process. Of note, the episode length started at 3000 steps (the plot divides the y-axis by 4 due to limitations of implementation) and quickly dropped toward zero, before recovering and steadily trending upward. This implies that the model spends the first few epochs exploring and learning the actions before beginning to learn a proper policy. The drop to zero signifies an encounter with the first danger in a level. 3-1 likely saw poor performance due to the introduction of both night levels with a dark background instead of the standard light blue, bouncing enemies, and hammer bros which throw projectiles. These are a significant departure from 1-1 and 2-1, which are much simpler levels by comparison.

5.2. Transfer Learning Experiments

For our first transfer learning experiment, we trained our Level 1-1 base model on Level 4-1. We trained another agent using direct policy transfer, rather than continuing to train the model. We found that both methods of transfer learning showed strong benefits early on, but the base model caught

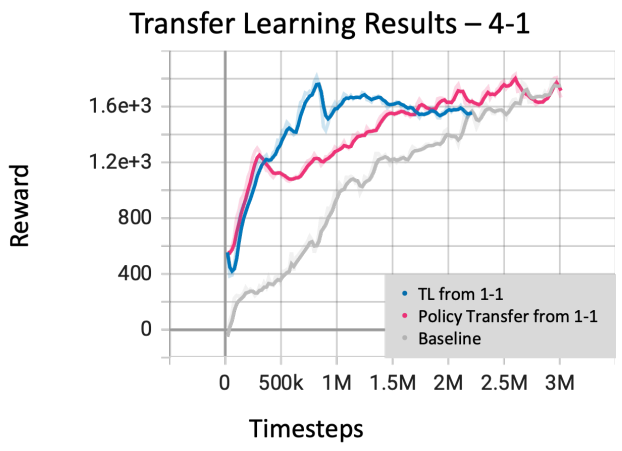

Figure 5. Transfer Learning Rewards: Level 4-1

up by the end of the training. A likely explanation would be that the models with transferred knowledge are not given additional time to explore further. Of note, the base model for 4-1 never reached the time limit. This likely means that the transferred policies waste some time. However, since the performance is still strong, we consider this a success. We followed a similar procedure for Level 8-1, but only with continuing training on our model from 1-1 (this is different from the model trained on 1-1 and 4-1). We found that the model with transfer learning performed better overall by about 300 points.

Our second transfer learning experiment draws from multi-task learning. We train a brand new agent by randomly changing the level after each episode terminates. To compensate slightly, we allow the agent five million time steps

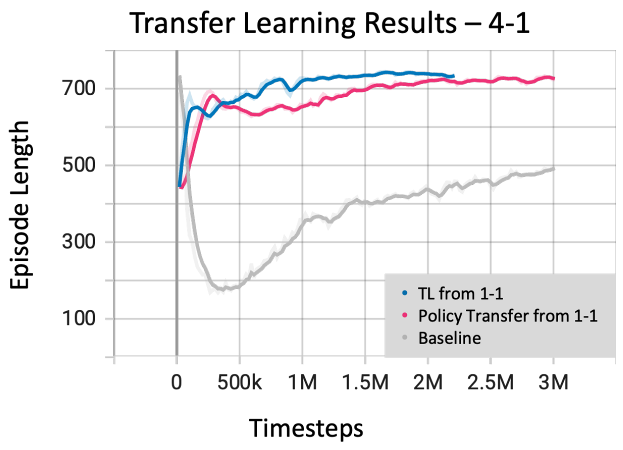

Figure 6. Transfer Learning Episode Lengths: Level 4-1

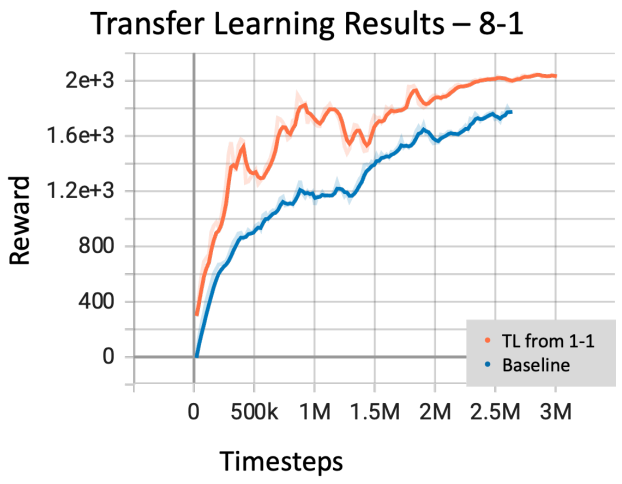

Figure 7. Transfer Learning Rewards: Level 8-1

total. This allows the agent to see all of the levels for a shorter time but should force it to develop a general policy. However, the training results indicate this was not a good approach. The maximum reward was approximately 530 and had a maximum episode length of 715, smaller than any previous training run. There are a few potential reasons for this. First, the agent does not have a lot of time to see any particular level. Thus, it cannot develop a policy for that level. Second, the agent is trying to generalize the policy too quickly. There is a strong chance that the agent simply enters the next level with a weak policy that fits the previous level, which never allows it to improve. Third, the levels might be too different for this approach to be effective. Since the performance for 3-1 is so much lower, the agent could begin to fail every time it is placed at that level. Therefore, the agent would struggle to develop a general policy when it cannot develop a single policy for this level.

Our final transfer learning experiment aims to resolve the limitations of the previous methods. Instead, we reduce the number of parallel environments to eight, one for each state, and run all eight levels in parallel. We found similarly poor performance, so we tried two other learning rates (3e-3 and 3e-7) with no success. As such, we believe these two experiments require fundamentally different training and/or model designs.

6. Discussion

We find that Level 1-1 has the best performance of every model. We expect this as the level is designed to be a learning space for new human players. As training proceeds, we see that the episode length increases. Since the maximum episode length is 750 (which equates to 3000 due to internal scaling within Retro), we expect the reward could continue to grow higher if we allowed the agent to continue learning. However, we made the time trade-off to limit training to three million time steps. We found this was enough to demonstrate stellar performance while also keeping training runs manageable.

The remaining levels tested had much lower performance (over 2000 points less). Since this was the same agent playing, we would expect that the level of difficulty has increased. This was a reasonable assumption as the game was designed in a way such that new mechanics are progressively introduced as the game proceeds. However, some levels saw much lower than expected performance, like 3-1. Level 3-1 is the first night level, which replaces the light blue sky in the background with black. Since all other objects remain the same color, this cannot be the reason. Instead, we suspect the Hammer Bros enemies halfway through the level are to blame. This is a new enemy that is tricky to avoid and/or defeat for new players. We believe that the agent is continually running into projectile hammers at this point in the stage and cannot find a way past them. It is possible the reward function does not reward the agent for being patient. However, it can be argued that the agent would discover that the reward significantly increases with a little patience.

The results of the transfer learning approaches are the most surprising. Early testing exhibited poor performance in the long run for all agents with previous knowledge of another level. However, after the full tuning process, we found that the agents with transferred knowledge did not need much exploration at the beginning of training to increase their reward. However, these agents quickly approached a ceiling on episode length. We suspect that the agents got stuck somewhere in Level 4-1 and failed to find an effective

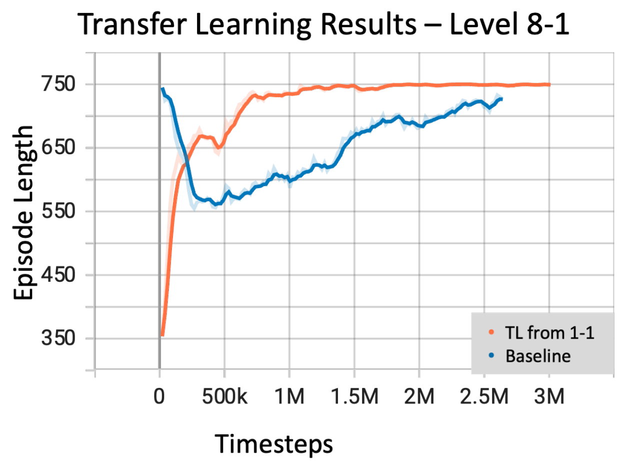

Figure 8. Transfer Learning Episode Lengths: Level 8-1

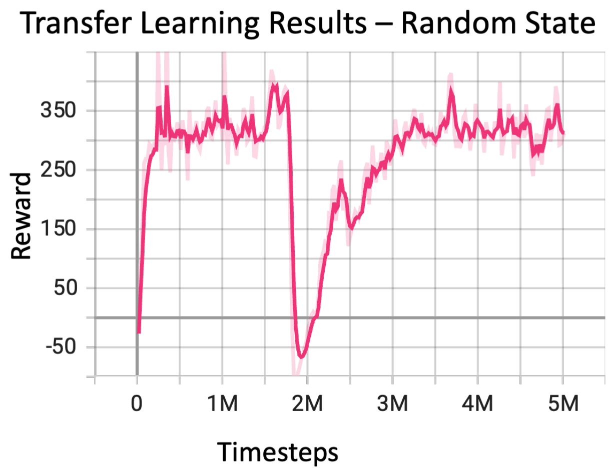

Figure 9. Transfer Learning Rewards: Random Level

solution, hence the performance also hit a ceiling. For Level 8-1, we see performance increasing throughout the entire training performance, despite hitting the per-epoch step limit. We suspect that in this case, the model can constantly optimize its path through the level. Future work in this environment would be related to identifying points at which the models begin to overfit. In preliminary testing, we trained an agent that was capable of reaching the halfway point of all eight levels, but it could not finish any of them. We are still unaware of what caused this model to generalize so well, but we do know that it was trained for a much shorter period (approximately 500k time steps).

The other transfer learning-based approaches performed mediocre at best, with the random state method showing a significant drop in performance after around 2 million steps. We suspect that these environments may be viable, but only for other training strategies. Since each level is reasonably different, we expected that multi-task learning or meta-learning-based approaches would produce more robust agents. However, since our agents were not able to beat every level we trained them on, we expect other issues need to be resolved.

7. Conclusion

We examined transfer learning-based approaches to generalizing agent policies in the context of platforming video games. We summarized three RL algorithms that are capable of such a task and introduced Super Mario Bros as an MDP. We also discussed our base models and transfer learning methodologies. We examined previous approaches to task transfer specifically in the context of RL. We found that pre-training and policy transfer allowed for boosts in initial performance, but found that this advantage is not held over time. To continue this work, we suggest multi-task and meta-learning for generalizing the policy further. We hoped to demonstrate an agent playing different games but found that to be unachievable at this time. Using multi-task learning would allow an agent to see better performance in more than one game by default.

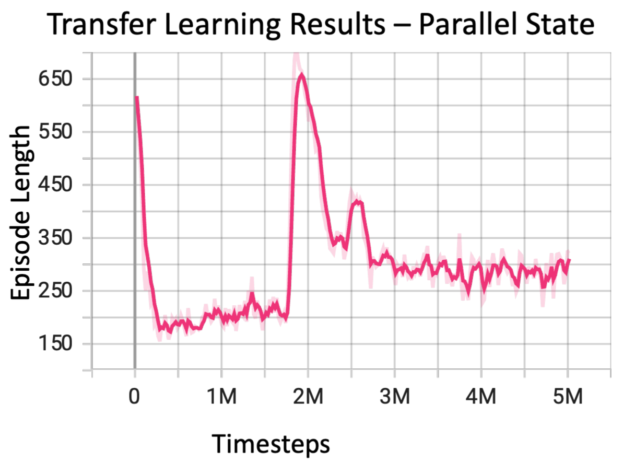

Figure 10. Transfer Learning Episode Lengths: Random Level

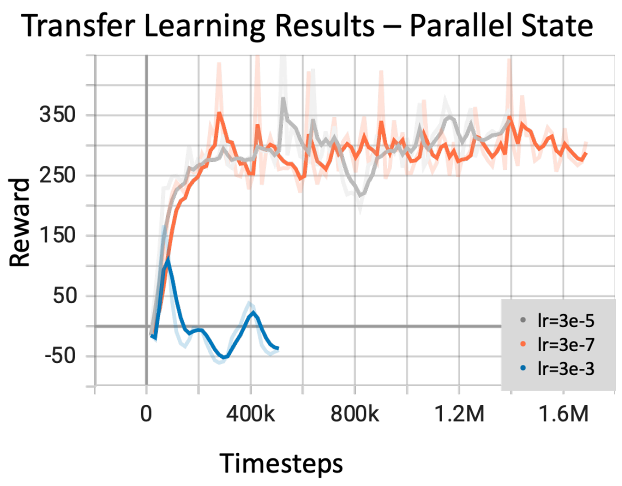

Figure 11. Transfer Learning Rewards: Levels in Parallel

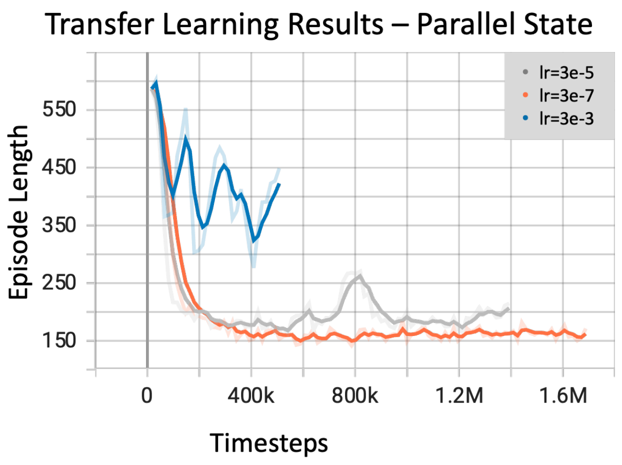

Figure 12. Transfer Learning Episode Lengths: Levels in Parallel

References

[1] S. Gamrian and Y. Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *International conference on machine learning*, pages 2063–2072. PMLR, 2019.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[3] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *International conference on machine learning*, pages 1809–1818. PMLR, 2017.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

[6] E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[8] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.