2019

ALE Transfer Learning

PROF MIKE ZYDA

CSCI 599: APPLIED MACHINE LEARNING FOR GAMES

ABHAY ATRISH | KUNAL KOTWANI | KUSHAL D'SOUZA | PRABHJOT SINGH

UNIVERSITY OF SOUTHERN CALIFORNIA | Los Angeles

Contents

Human learners appear to have inherent ways to transfer knowledge between tasks.

What we acquire as knowledge while learning about one task, we utilize in the same way to solve related tasks.

Know how to ride a motorbike -> Learn how to ride a car
Know how to play classic piano -> Learn how to play jazz piano
Know math and statistics -> Learn machine learning

The more related tasks are, the easier it is for us to transfer, or cross-utilize our knowledge.

The ability to act in multiple environments and transfer previous knowledge to new situations can be considered a critical aspect of any intelligent agent, and exploring it can help set the foundation for AGI - Artificial General Intelligence

1. Project Goal

The aim of our project is to train an autonomous agent to learn how to play an arcade-based game and identify the possible benefits of applying transfer learning to games within the **Arcade Learning Environment (ALE)** belonging to a similar genre.

Our first task is to research the various deep reinforcement learning models that are in use today and identify a game on which we would build and apply this model to learn optimal policies for the game using high level sensory inputs such as visual frames.

Upon the successful training of the model, we aim to use this pretrained model to train it on another game belonging to a similar genre to the originally trained game, to see if the gameplay learnt by first game can be adapted to the new game, with minimal training possible.

Finally, we will evaluate and report our findings to conclude any evidence of the benefits of transfer learning over training a deep RL model from scratch.

2. Background

In order to conduct our research for this project, we will train the agent to learn the optimal policy for the game using deep reinforcement learning.

2.1. Deep Reinforcement Learning

Reinforcement learning (RL) is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision-making problems in a wide range of fields in both natural and social sciences, and engineering.

A RL agent interacts with an environment over time. At each time step t, the agent receives a state s_t in a state space S and selects an action at from an action space A, following a policy $\pi(a_t|s_t)$, which is the agent's behavior, i.e., a mapping from state s_t to actions a_t, receives a scalar reward r_t, and transitions to the next state s_{t+1}, according to the environment dynamics, or model, for reward function $R(s, a)$ and state transition probability $P(s_{t+1}| s_t, a_t)$ respectively. In an episodic problem, this process continues until the agent reaches a terminal state and then it restarts. The return $R_t = P_\infty \gamma*k*r_{t+k}$ is the discounted, accumulated reward with the discount factor $\gamma \in (0, 1)$.

We obtain deep reinforcement learning (deep RL) methods when we use deep neural networks to approximate any of the following components of reinforcement learning: value function, $v\hat{}(s; \vartheta)$ or $q\hat{}(s, a; \vartheta)$, policy $\pi(a|s; \vartheta)$, and model (state transition function and reward function).

The game that we chose to train the source model on is Super Mario Bros, which is a platform game set in the Mushroom World. In this game, an actor called Mario can be controlled by the user. The game consists of multiple levels and Mario can jump, walk and run in each level.

The game also consists of other actors who are the enemies of Mario. The goal of the game is to get to the next level without dying. Mario can navigate to the end of a level by jumping over platforms, jumping to avoid holes and avoiding enemies. If Mario touches any enemy, this results in his death. Mario can also collect mushrooms within the game. Collecting red and white mushrooms make Mario bigger. Mario does not die if he touches his enemy when he is big. This results in him becoming smaller. The game has certain

other characteristics which are not important to the work we are doing on this research and are thus not described in this report. After learning the optimal policies for SuperMario, we apply transfer learning to a game in a similar genre as Mario.

2.2. Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

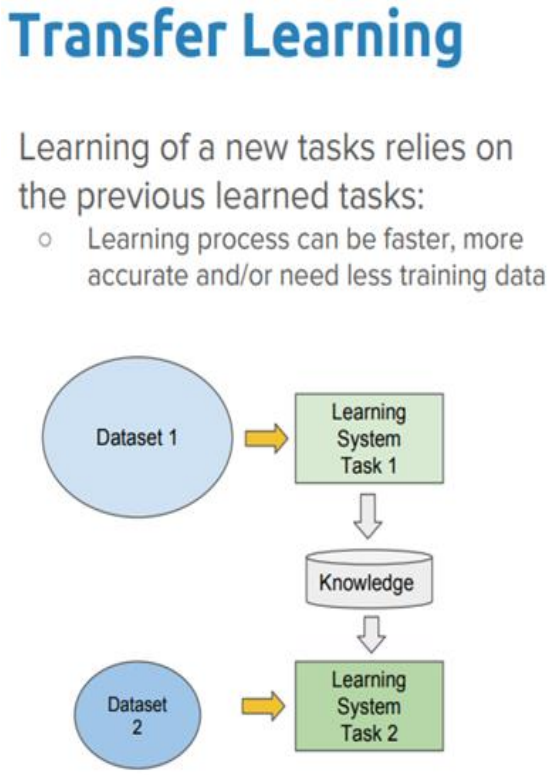

Figure 1. Transfer Learning Process

3. Prior Research

This section of the document will discuss the algorithm/model that we have chosen to work with after conducting research on various deep learning models used in the field today. Recently, researchers have achieved great success in games using deep reinforcement learning methods as well as transfer learning on games.

In their groundbreaking paper "Playing Atari with Deep Reinforcement learning", *Mnih et al.* developed a single Deep Q-Network (DQN) that can play multiple Atari games, in many cases surpassing human expert players. The model consists of convolutional layers followed by fully connected layers. The model takes in the raw image pixels of a game and outputs Q-values estimating future rewards.

"Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning", *Parisotto et al.* build on the work of *Mnih et al.* to explore transfer learning between Atari games. They developed the Actor-Mimic method that uses the guidance of many game-specific expert networks to train a single multitask policy network.

For transfer learning, they treat the multitask network as a DQN, and they transfer all the weights except the final Softmax layer to a new DQN.

With the recent breakthroughs in the actor critic networks, PPO has achieved significant popularity in the arcade learning domain, and we have decided to use the Proximal Policy Optimization (PPO) which is based on a type of algorithm known as Actor Critic.

Two of the traditional reinforcement learning methods are Value Based Methods and Policy Based methods.

3.1. Value-based Method

In value-based methods, we learn a mapping for every state action pair to a value. A value function is a prediction of the expected, cumulative, discounted, future reward; measuring the goodness of each state, or each state-action pair.

Using this mapping, we can choose an action that has the highest value at a given state. However, when we have action spaces that are continuous or stochastic, value-based methods can be computationally expensive and consume a large amount of memory.

3.2. Policy-based Method

Policy based methods work better in continuous and stochastic environments. A policy maps a state to an action, or, a distribution over actions, and policy optimization is to find an optimal mapping.

Value-based methods optimize value functions first, then derive optimal policies. Policy-based methods directly optimize an objective function, usually cumulative rewards. In policy-based methods, we update the policy without using an action-state-value mapping. Instead, policy-based methods use the total rewards earned in a given episode.

One issue that policy-based methods have is described in the next example. Suppose one takes actions A, B, C, D and E to reach the end of the episode and earn a high reward. However, even if acting C in this sequence leads to a negative reward, this action will be considered good because policy-based methods use the total reward obtained at the end of an episode.

3.3. Actor Critic Method

This is where the actor critic method comes into play. Actor-critic methods combine value function and policy. Here, instead of waiting till the end of the episode to get the total rewards and update the policy, we do an update at each step. A critic function can be used to approximate the value function so as to determine the action that provides the best value at a given state. Therefore, we have two neural networks one acting as the actor and the other as the critic.

The actor uses the critic as a value function approximator to update the policy at each timestep. At the same time, the critic updates its own value function at each timestep. The biggest advantage is parallel actors that employ different exploration policies to stabilize training, so that experience replay is not utilized, reducing memory usage.

3.4. Proximal Policy Optimization

In policy-based methods, the update involves taking a gradient ascent step in order to maximize the rewards obtained. This ascent step is regulated by a step size. The issues with this are twofold, a small

step size leading to long training times and a large step size leading to high variability. This is where Proximal Policy Optimization (PPO) methods help.

PPO alternates between data sampling and optimization, with added benefits of stability and reliability and with the goal of simpler implementation, better generalization, and better empirical sample complexity. PPO brings in greater stability during the update step performed on training the actor by limiting the size of the update.

4. Methodology

The overall architecture for the project has been summarized below.

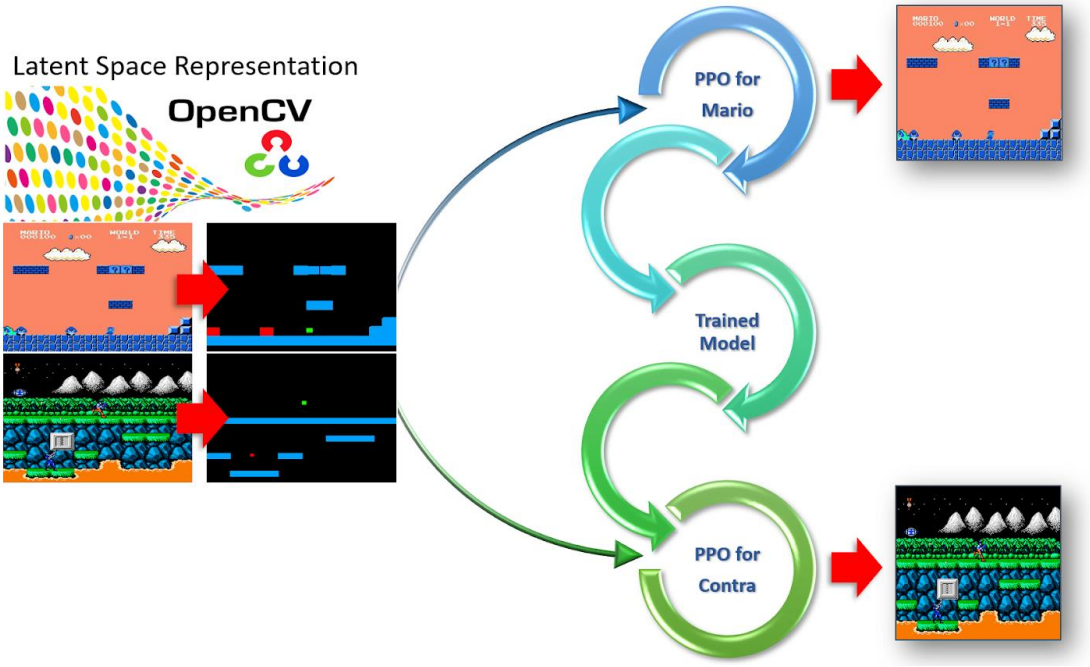

Figure 2. Overall Architecture for ALE Transfer Learning

The first step in our model pipeline was to determine how we would consume the game data to be fed as an input to our model. The options we had were either providing the raw pixel data as input to the network or providing the game state with the values of different features (Location of Mario, Location of on-screen obstacles, Location of enemies, etc.) as input to the network.

4.1. Latent Space Representation

We decided to provide a down-sampled form of the raw pixel data from each frame as the input to our model.

During the down-sampling process, we use OpenCV in order to locate items that are on the currently visible frame. We color each of the items (Mario, Enemies, Ground) that are of interest to our model using separate colors. We then redraw these items on a black canvas keeping their positions in place. This redrawn image is fed as an input to the model.

The reason for performing this down-sampling on the input frames are twofold:

1. Firstly, down-sampling and redrawing items of interest on a black canvas will help the model focus on only the items of interest and ignore any noise that would have occurred while providing the original frame as input.
2. Secondly, when transferring this model to be trained for the second game, we will perform the same down-sampling of the frames from the second game. This ensures that both games are converted to a command representation wherein features such as the enemies in both the games have the same color. We hope that this will enable us to train the model during the transfer learning process in a shorter time period.

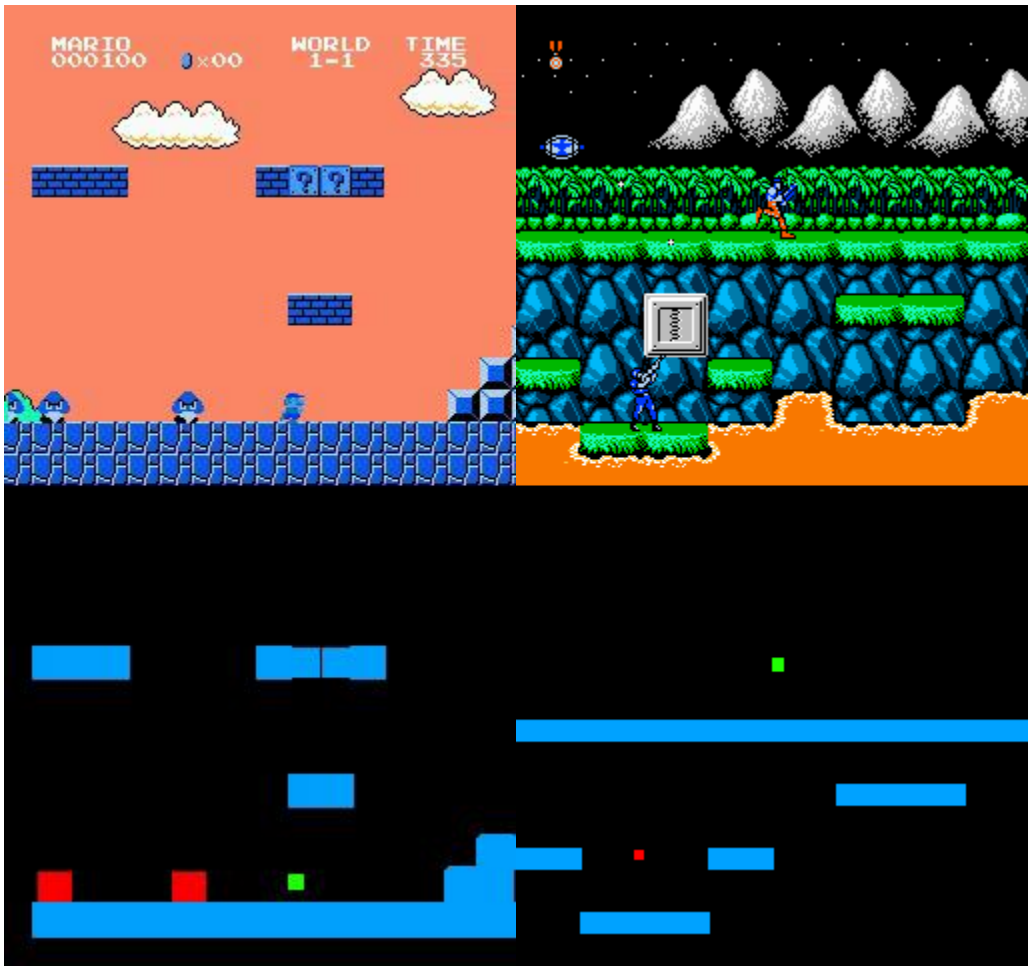

Figure 3. Latent Representation for SuperMario and Contra

4.2. Environment

In the proposed architecture, the game environment has to be simulated to perform actions and obtain the frames for training. The project uses OpenAI's Gym for environment simulation of NES games. Like the DeepMind paper, we preprocess the raw frames with different wrappers to augment the learning process.

1. **Stochastic Frame Skip** - Helps to limit the frames that flow out of Gym for training.
2. **Time Limit** - Limits the time for a continuous environment to a limited step.
3. **Warp Frame** - Applies the latent visual representation and downscales the frames.

4. **Clip Reward Env** - Clips the rewards to +1, -1.
5. **Frame Stack** - Stacks multiple frames to create a buffer for delta calculation.
6. **Scaled Float Frame** - Normalizes the frame's pixel intensity values between 0 and 1

Figure 4. Wrapper-based frame pre-processing

We defined custom rewards for the Mario environment to:

1. Reward the agent to move right.
2. Punish the agent for not taking any action for a long time.
3. Punish the agent if they die.

These are all defined as a part of the environment for SuperMarioBros to enable an efficient learning process.

4.3. Deep Learning Model (PPO) architecture

The project utilizes the PPO algorithm with an Actor-Critic architecture to implement reinforcement learning. The training is performed on the frames from the Gym environment preprocessed using the custom wrappers.

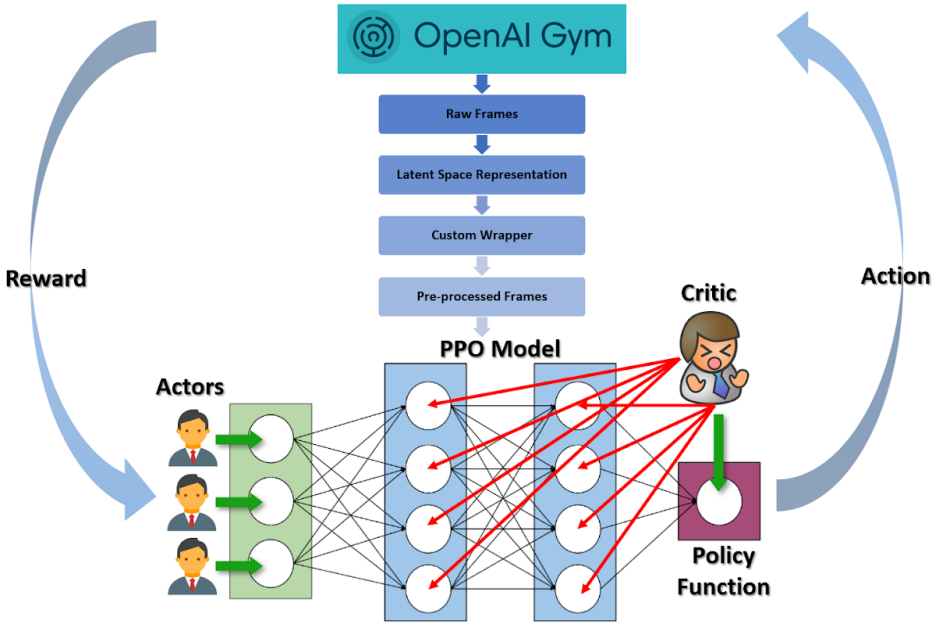

Figure 5. PPO Architecture for ALE

The parameters for policy and value function can be shared in a neural network, and advantage function can be estimated to reduce variance of policy parameters estimation. Multiple parallel actors explore the environment in different states performing actions, while the critic updates the policy network from their observations.

The model itself will output a function that will inform us of the best action to take from a given state. We will then use the computed parameters of this model as a starting point when training this network on the second game.

We aim to identify whether converting both games to a common representation and then transferring the weight from the model of the first game to be used as a starting point while training the model of the second game will lead to a quicker training process. We also hope that the rewards obtained during training over different epochs is stable.

5. References

1. Dipanjan (DJ) Sarkar. "A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning." 17 Nov. 2018, towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a.
2. Asawa, Chaitanya et al. “Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance.” (2017).
3. Mnih, Volodymyr et al. “Playing Atari with Deep Reinforcement Learning.” ArXiv abs/1312.5602 (2013).
4. Parisotto, Emilio et al. “Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning.” CoRR abs/1511.06342 (2015).
5. Tan, Chuanqi et al. “A Survey on Deep Transfer Learning.” ArXiv abs/1808.01974 (2018).
6. Schulman, John et al. “Proximal Policy Optimization Algorithms.” ArXiv abs/1707.06347 (2017).
7. Mnih, Volodymyr et al. “Asynchronous Methods for Deep Reinforcement Learning.” ICML (2016).