Deep Reinforcement Learning for Atari Games

Garnet Liu Vikram Saraph Birol Senturk

December 14, 2018

Abstract

Model-free agents can perform very well on certain tasks, but they tend to have trouble when a greedy approach is not optimal, or precise planning is required for success. *Imagination-Augmented Agents for Deep Reinforcement Learning* is a paper which proposes to augment a model-free agent with an environment model that learns a game's or task's environment. We implement their algorithm in TensorFlow, and attempt to apply it to Pong, with some degree of success.

1 Introduction

In recent years, deep learning techniques have been successfully adapted and applied to problems in reinforcement learning. In this project, we survey and evaluate one such technique, called *imagination augmented agents*, or *I2A*. This technique was initially developed and published by Weber *et. al.* at DeepMind [5]. In their original work, the authors describe the novel I2A as one that combines aspects of *model-free* learning, in which an agent makes decisions based only on its state and reward observations, with *model-based* learning, where the agent attempts to learn about its environment in addition to a policy. In their work, the model-free agent is augmented with a model-based *imagination engine*, which makes predictions about future trajectories of the agent from its current state.

While model-free agents have seemingly more flexible architectures, in practice they do not generalize very well to different tasks, since they can exhibit poor performance. Incorporating model-based aspects allows one to provide additional fine-tuning to the network's architecture in a way that is specific of the agent's environment. Indeed, the authors of the original paper demonstrate a significant improvement in the agent when adding imagination to a standard model-free agent. They apply I2A to two different 2D games: MiniPacman, which is a simplified version of PacMan, and Sokoban, in which the agent must push blocks onto designated targets to solve each level.

In this work, we focus on applying I2A to Pong. First we provide a general background on the I2A architecture, in addition to the gaming environment we use to simulate Pong. Then we describe our implementation and design decisions we made when applying I2A to Pong, and conclude with results from our implementation and discussion on issues and potential future work.

2 Related Work

As with many machine learning models, the original work on the Actor-Critic algorithm long proceeded the recent explosion in deep learning literature [3]. Asynchronous Actor-Critic, or A3C [4], further builds upon A2C by parallelizing the work performed by the original algorithm in an asynchronous manner. A3C is thus capable of better exploiting modern multicore architectures, and is the backbone of the model-free component of I2A.

We implement I2A in TensorFlow. While there does not appear to be an official implementation of I2A by DeepMind, we referred to a PyTorch implementation found on GitHub [6] as a guideline for our implementation.

3 Background

In this section, we provide a high-level overview of the I2A architecture as described in the original work. See the Methodology section for more specifics on how we applied this architecture to our Pong use case.

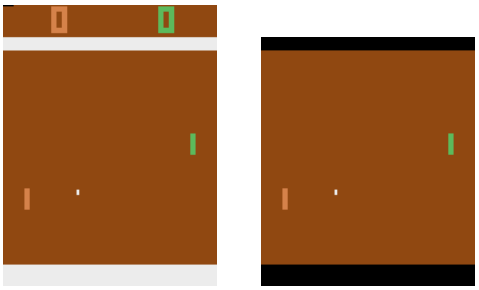
I2A unifies a model-free architecture and a model-based one. The model-based component of I2A is also referred to as the imagination engine, and consists of a sequence of *imagination cores* that collectively generate a predicted future trajectory. Each imagination core makes use of a pre-trained policy network and an environment model. The policy network predicts the best action to take given an input state. The environment model takes an input state and the action returned by the policy network, and

predicts the next state from taking the action. The predicted next state is fed from one imagination core to the subsequent one, so in this way, the output of the imagination cores is a sequence of predicted states (or trajectory). This trajectory provides additional context when learning a policy; together, the model-free policy and the imagine engine are combined to train a unified policy.

4 Game Environment

The Arcade Learning Environment (ALE) is used to simulate playing Atari games, so we use this to play and interact with Pong. ALE has been integrated into OpenAI Gym [2], which provides a standardized interface for taking actions and retrieving the resulting new state, reward, etc. The game's state is simply the RGB image of a frame at a given point in time, though many games have an alternative API that returns the RAM's state instead.

Before image data is fed into any part of the network, it is normalized in a way that is more convenient for training. In particular, the top of the image is removed, since it is only used to display the players' scores, pixels of which are unlikely to be useful features for learning to play the game. In addition, the top and bottom border color is changed from white to black, since it is the same color as the ball. The idea is to help the network distinguish between the two types of pixels. See Figure 1 below for a frame and its normalization.



(a) Original image (b) After normalization

Figure 1: Pong screenshots before and after normalization.

The OpenAI Gym API provides rewards and states, so we work with this data when training our model. Actions are taken with the `step` function.

5 Methodology

The model is trained in three phases. First, a model-free agent is trained by playing the game and using A2C to learn. Once the model-free agent is trained, its policy is used to train the environment model. The environment is trained by using the model-free agent's policy to play the game; by doing so it learns the environment. In the following, we outline the structure of our TensorFlow program.

5.1 Model-free agent

ALE provides two different state spaces for playing Atari games, RAM content and the rendered pixels. We decided to use the pixel output, and stacked the current frame on top of the previous one to give the network some directional information which would in principle make our model-free agent easier to train.

While the original paper used an A3C (Asynchronous Actor-Critic) architecture as its model-free policy network, we decided to use the simpler A2C architecture. The main difference is that A3C uses independent workers to update a shared policy network asynchronously, while A2C updates its weights sequentially. A2C networks are harder to train, but easier to implement. A2C contains an Actor that takes in an observation and produces a probability distribution over possible actions, and a Critic that also takes in an observation and assigns a numeric value to the observation which is then used to determine how advantageous a certain action is given the observed state. While these can be two separate networks, we decided to use one network with two heads instead. Input first goes through a stack of three convolutional layers and a fully connected layer that forms the trunk. The output of this network is fed into separate Actor and Critic components which are two fully-connected layers and one output layer each.

5.2 Environment model

The computation graph of the environment model's network begins by using the policy described in the previous section to take an action. Given an action and current state, recall that the environment model predicts the next state and reward obtained. In general, the environment model's architecture should be designed for the specific game in mind; however, for this project we used a modified version of the Mini-Pacman environment model.

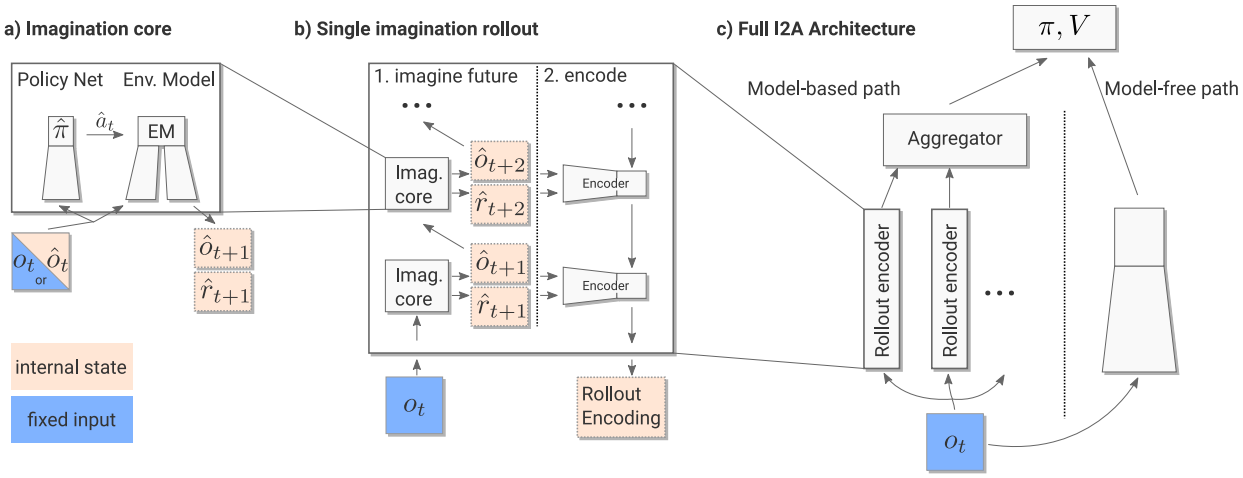


Figure 2: I2A architecture

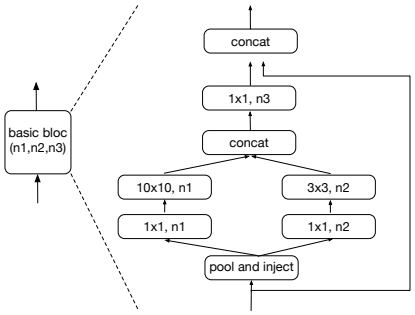


Figure 3: Basic Block, or a building block of the environment model's architecture

Recall that the state returned by the OpenAI Gym API is just an RGB image, and due to normalization described in the Game Environment section, there are only five distinct kinds of pixels: border pixels (black), background pixels (brown), ball pixels (white), player pixels (green), and opponent pixels (orange). Therefore, each pixel in a given state is first mapped to a corresponding index (1 - 5). Then, the action taken by the model-free agent is converted to a one-hot representation, and concatenated to the state data. This encodes all state and action information that the environment model needs to begin training.

Next, the input data is sent through a series of convolutional layers, consisting of higher level logical units called "basic blocks", as described in the paper. The first layer of a basic block is *pool-and-inject*, which applies a maxpool and concatenates the result to the input. The result is then passed through two parallel convolutional layers, which are concatenated again for the basic block's final output.

After the basic block units, there is an additional convolutional layer and fully-connected layer for the state and reward predictions. Figure 3 is the diagram of basic block as included in the original paper.

5.3 I2A

I2A combines the knowledge learned by the models from the previous two steps. Like the model-free A2C, I2A learns a policy, but it also incorporates information from the environment model.

We describe the forward pass of the architecture. Recall that I2A depends on a pre-trained model-free agent and environment model. Together, these will form the imagination cores. The forward pass is fed a batch of states, and the imagination engine uses these states to predict a sequence of predicted future states. In more detail, the model-free agent picks an action, and gives this action to the environment model, which predicts a next state. This next state is iteratively passed back into the model-free agent to pick another action, and the process continues, producing a sequence of states, or a *rollout* as referred to in the paper. One or more rollouts may be computed. Each rollout is then passed to a *rollout encoder*, which is a GRU cell. The results of the encoder is finally concatenated to the output of the model-free path, which uses the collective information to pick an action. In this way, I2A learns a policy. See Figure 2 for the schematic diagram used in the original paper.

6 Results

Here we discuss some results obtained from running components of the I2A network.

6.1 Model-free agent

A2C was very difficult to train for our current task. The network converges to a suicide strategy, both for Pong, and even when applied to Qbert, another Atari game. See the discussion section for why this may be.

Average loss is increasing even though the average time discounted rewards is also increasing. The network tends to converge into a losing strategy that has very low critic loss and quickly abandons accidental discoveries of high reward strategies.

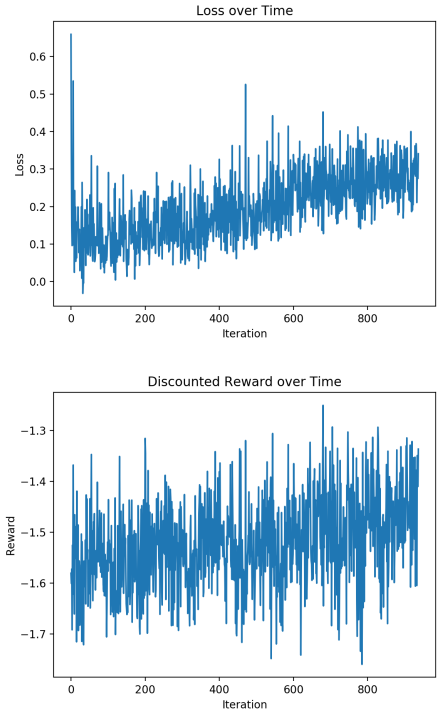
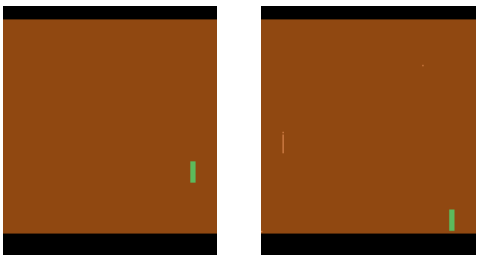


Figure 4: Loss and discounted reward while training the model-free agent.

6.2 Imagination

The environment model had moderate success at learning to predict the RGB image of a next state. Typically, when looking at frames that the model "imagines", one sees a pattern where the model first learns shape, then learns colors. For example, the model is very quick to learn the border and background boundaries in just a few iterations, even though in the imagined image the background may be green or the border white. It is slower to learn the player, opponent, and ball pixels, though they do eventually appear after several dozen steps of training the environment.

See Figure 5 for examples produced by the imagination core. In the figure on the left, only the player is visible, while on the right, some pixels of the opponent are visible. The ball is also visible, though it is more difficult to see, and is not in the correct color, since the model has not yet learned the ball's color.



(a) Player appears (b) Opponent appears

Figure 5: Sample imagined images. The model is quick to learn the background pixels, but takes longer to learn the player, opponent, and ball pixels.

7 Discussion

In this section, we discuss the issues encountered when studying I2A and implementing an architecture for Pong, along with potential solutions that could be implemented in future work.

7.1 Model-free agent

The model-free agent is an integral part of the whole I2A architecture, as it not only is one of the two primary decision pathways alongside the aggregated rollouts, but it is also used while training the environment model and the rollout encoder. However, as seen in the Results section, A2C did not appear to perform very well when learning Pong. There has been previous academic discussion on A2C being a poor choice to learn and play a game like Pong [1].

There are many states that are similar to one another, yet only a small subset of these states yield rewards, so there is a high critic loss attached to actions that return rewards, meaning the network avoids repeating strategies that were successful. On top of this, state values get harder to learn as the network plays better, since a bad agent consistently gets negative rewards with low variance, while variance of the expected reward increases as the agent gets better and better.

We attempted several approaches to alleviate these issues. One thing that was tried was to add an entropy loss to the loss function, given by the equation:

$$entropyFactor * actProbs * \log(actProbs)$$

Entropy incentivizes exploration instead of converging to a one-hot probability configuration. While the addition of entropy did help, it hampered the network's ability to precisely adjust the player's position, losing points even when the player is mostly in the correct position.

Another issue we encountered was that policies that were overall better sometimes performed badly given the randomness in the outcome and how rare the rewards are. This results in good strategies being abandoned quickly. Batches of 8 games were played before each update to reduce this effect. Larger batches would help even further, but the size of GCP instance's VRAM prevented us from increasing this batch size. A better way to solve this problem may be to use A3C, which has many worker processes that each play individual games, which would mean frequent updates to the network as well as many games with similar policies to get an average effectiveness of the policy.

7.2 Environment

As previously stated, the environment model used for Pong is a modification of the one used by MiniPacman. Between the environments of MiniPacman and Sokoban, we believe that the former is better suited to Pong since its pool-and-inject layer considers global information about the environment. However, in a more thorough study of I2A applied to Pong, one should further consider how the environment can be best tuned specifically to Pong. It is possible there is a significantly different environment architecture that is performs better than what is used in this work.

Another critical difference between Pong and MiniPacman is the difference in size between the RGB images of the two games. MiniPacman is quite small, even by Atari 2600's standard, and is represented as a 15 x 19 grid world. By contrast, Atari's Pong has a 210 x 160 pixel representation, which is roughly two orders of magnitude larger than MiniPacman. Therefore, it is possible that using larger or deeper networks than the one used in MiniPacman would result in much better performance. Alternatively, one could design a Pong game from scratch that uses substantially fewer pixels, since the game has very low complexity. Unfortunately, the original paper did not provide any specifications on the hardware used for training their models.

8 Conclusion

In this work, we attempted a TensorFlow implementation of I2A. We were able implement the model-free agent and the environment model, and get an imagination core working to produce imagined images. In future work, it should be considered that A2C may not be the best model-free agent to use for a game such as Pong, and it may be better to use a Deep Q-Network instead. Nonetheless, we were able to demonstrate some success with the components of I2A that were implemented.

References

[1] Samuel Arzt, Gerhard Mitterlechner, Markus Tatzgern, and Thomas Stutz. Deep reinforcement learning of an agent in a modern 3d video game. In *Visual Learning and Embodied Agents in Simulation Environments, ECCV Workshop*, 2018.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000.

[4] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[5] Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5690–5701. Curran Associates, Inc., 2017.

[6] Dulat Yerzat. Building agents with imagination, 2018.