Analyzing Robustness of Deep RL Models

Arjun Krishna (ak4471), Ethan Wolfe (ew2740), Ali Al-Keldi (ata2160), Harald Gerhardsen (hg2637)

Introduction

Over the last decade, tremendous strides have been made in the field of Deep Reinforcement Learning (DRL). This is due to breakthroughs in research, technologies and methods of Artificial Neural Networks. Multiple types of networks have been used on Reinforcement Learning (RL) domains with huge success: in 2013, DeepMind introduced the first agent that acted directly on high-dimensional input data in terms of frames from Atari 2600 games, using Convolutional Neural Networks; in 2015, DeepMind published the DQN agent, one of the first agents to perform at or beyond human-level across a variety of games [1]; Transformer models have been used to outperform the state-of-the-art agents on similar, as well as novel and more complex, domains [2][3].

In this project, we aim to study the robustness and performance of some published DRL agents. We will implement the DQN, the Swin Transformer, and the Decision Transformer to play video games, and their generalization performance will be compared. If possible we would like to make modifications to the games, train the networks on a specific version of the game, and evaluate the performance and robustness on other versions. Another method of evaluating out networks would be to train them on a specific level of the game (e.g. Super Mario Level 1-1), and then test them using different levels of the game (e.g. Super Mario Level 1-2).

Related Works

One of the earliest, and most successful, models in the field of DRL was the **Deep Q-Network (DQN)** [1] introduced by DeepMind in 2015. This agent took frames directly from games (in the Atari 2600 domain) as input, and with a convolutional neural network decided on actions to take in the game. This was a major milestone in both Deep Learning and in Reinforcement Learning, since the agent outperformed all previous state-of-the-art, and even played at the same level or better as a professional human game-tester in over half of all games. The robustness of the agent was thoroughly assessed - the same architecture and set of hyperparameters were used across all games. Further extensions to this agent has been made, including large-scale parallelization [5].

Decision Transformers were proposed in [2] by Chen et al. They provide an architecture that casts the problem of Reinforcement Learning as conditional sequence modeling. Decision Transformer diverges from the traditional paradigm of RL, which is typically concerned with Q-value approximation through either dynamic programming and Bellman updates or with gradient computations in the case of Deep Neural Networks. Instead, the Decision Transformer samples data through random trajectories, and feeds entire trajectories as sequential data to the transformer model.

Swin Transformer was originally published by Liu et al. [4] as a hierarchical transformer model that could better capture long-term dependencies between different regions in visual data. Meng et al. [3] successfully used a Swin transformer in DRL setting to play video games. Their model outperforms that of the Decision Transformer on the majority of games in the

Arcade Learning Environment, and it does so while returning to the more conventional method of gradient computation and Q-value estimation.

Methods & Algorithms

We will implement three different models, each of which takes a different approach to reinforcement learning, and analyze their performance in the context of robustness. To implement these models, we will utilize existing pytorch libraries such as https://github.com/microsoft/Swin-Transformer and https://github.com/kzl/decision-transformer. These implementations will be tested primarily on the Super Mario Bros game, using the OpenAI Gym platform [6] to control the character.

One method of testing robustness is by making small adjustments to the game environment, such as moving the goal object or increasing the number of obstacles, and seeing if the model fails to pursue the goal effectively after the distribution shift. We will do this in our implementation by testing our models on a different set of levels, which will have different patterns, environments, and enemies. If possible, we may also make manual changes to the levels ourselves to test the model's robustness against specific obstacles / challenges. The general goal of the model will still be to move to the right as far as possible. Super Mario Bros. works well as an environment for testing robustness due to the variety in levels and obstacles provided.

We will measure the performance of each model by measuring the average distance the model travels to the right of the starting position, for each of the test levels. From this, we will perform a thorough analysis of the factors of each model that may lead to better/worse generalization across the test level set.

Extensions

We can utilize reinforcement agent visualization methods, such as saliency maps, to facilitate visual understanding of exactly what the agent is attending to as it progresses through the game. Example work in this space https://arxiv.org/abs/1711.00138. We can extend this approach to other games such as Atari games using the Arcade Learning Environment [5], as well as continuous and motor-control tasks.

References

1. Human-level control through deep reinforcement learning
    a. https://www.nature.com/articles/nature14236
2. Decision Transformer: Reinforcement Learning via Sequence Modeling:
    a. https://arxiv.org/pdf/2106.01345.pdf
3. Deep Reinforcement Learning with Swin Transformer
    a. https://arxiv.org/abs/2206.15269
4. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows
    a. https://arxiv.org/abs/2103.14030
5. Asynchronous Methods for Deep Reinforcement Learning
    a. https://arxiv.org/abs/1602.01783
6. Atari emulator
    a. https://github.com/mgbellemare/Arcade-Learning-Environment

7. Open AI Gym, Super Mario Bros
    a. https://pypi.org/project/gym-super-mario-bros/
8. Nintendo Emulators
    a. https://github.com/albarji/deeprl-snes
    b. https://github.com/freshwater/NES-Solve