Reinforcement Learning: Transfer Learning Between Different Games

Student Name: Matthew Chapman

Supervisor Name: Dr Lawrence Mitchell

Submitted as part of the degree of BSc Natural Sciences to the

Board of Examiners in the Department of Computer Sciences, Durham University

April 7, 2021

Abstract —

Background — Reinforcement learning, concerned with how agents learn behaviour through trial-and-error interactions with a dynamic environment, has proven to be a successful technique of achieving at least human-level performance by machines. Transfer learning is the application of knowledge gained while solving one problem to solve a different but related problem, and is a technique that allows reinforcement learning agents to adapt to new environments.

Aims — The aim of this project is to investigate the usefulness of transfer learning in a modern reinforcement system, where environments are Atari video games. We seek to quantify the benefit of pre-training on a game on an agent's ability to learn to play another game with similar mechanics.

Method — The method is to implement a modern reinforcement learning algorithm and train it on two selected games. Once the agent is performing well in the environment, we save a video of it playing, and use the video as the basis of pre-training of another agent, which we then evaluate on the test game.

Results — Pre-training leads to the agent performing better, as measured by an improvement in the following three metrics: jump-start performance, accumulated reward, and final performance. The more pre-training information the agent has, the better it performs.

Conclusions — Transfer learning is a promising method to prepare agents for environments where it has limited opportunities to interact with and learn from. By training agents on similar environments, we can build confidence that the agent will perform successfully when evaluated on the test environment. This is particularly useful in fields such as autonomous driving, where the vehicle must be able to adapt to various changes in the environment.

Keywords — Artificial intelligence, machine learning, reinforcement learning, deep learning, transfer learning.

I INTRODUCTION (2-3 PAGES)

This project is about reinforcement learning and transfer learning. The project involves developing a reinforcement learning algorithm to learn to perform successfully in some environment. Additionally, the project involves investigating how transfer learning lets the algorithm store knowledge and apply it — to learn to perform successfully in a different but related environment.

A Background

A.1 Reinforcement learning

Reinforcement learning is the class of problems concerned with an agent learning behaviour through trial-and-error interactions with a dynamic environment (Kaelbling et al. 1996). An example of a problem is an aspiring tightrope walker (the agent) learning to maintain balance (the behaviour) while walking along a tightrope that contorts and wobbles under their weight (the dynamic environment). With each attempt and fall (the trial-and-error interactions), the walker learns how better to correct their balance, and adjusts their behaviour slightly for the next attempt. When the walker is able to maintain balance consistently over consecutive attempts, the desired behaviour is achieved, and so the learning task is complete. We say that the problem is solved and the reinforcement learning agent has learned to perform successfully in the environment.

There are algorithms that act as agents that solve reinforcement learning problems. These reinforcement learning algorithms can solve problems in physical settings, such as driving cars, or in virtual settings, such as playing games. We can treat these algorithms as functions that takes as input observations and outputs actions. Examples of observations are the video from a camera attached to a self-driving car or the positions of pieces on a chessboard in an online match. Examples of corresponding actions are to turn the steering wheel in one direction or to move a chess piece. The goal of the algorithm is to learn which actions are the best to take given some observations. How good an action given an observation is can be measured by how likely taking the action is to lead to the desired behaviour. For an algorithm that drives cars, the desired behaviour might be to drive safely, and so the algorithm would know to stop at a red traffic light. Whereas, for an algorithm that plays chess, the desired behaviour might be to win, and so the algorithm would know to take the opponent's king. The algorithm is learning a mapping, from actions and observations to values, to inform its decision-making. This mapping is initially unknown, but improves the more the algorithm interacts with its environment — the same way one gets better with practice at driving or playing chess. For relatively complex problems, there may not be an optimal solution, such as behaviour that guarantees no accidents or that always wins, and so the best the algorithm can do is to approximate an optimal solution.

A.2 Transfer learning

Transfer learning is the application of knowledge gained while solving one problem to solve a different but related problem (Sammut & Webb 2010). An example is an agent who has learned to walk a tightrope (solving one problem) applying their balancing ability (the knowledge gained) to learn to surf (a different but related problem). By reusing knowledge gained from solving past problems, it is expected that solving a different but related problem will be more efficient than it would be without the prior knowledge. As in the example, a tightrope walker should learn to

balance on a surfboard more easily and more quickly, due to their knowledge of balancing on a rope, than someone without the same acquired knowledge of balancing.

In transfer learning for reinforcement learning algorithms, the knowledge gained that can be applied is the agent's policy. The policy is the set of rules that determine an agent's behaviour (which can be informed by the mapping mentioned in the previous section). An example of a policy for an agent playing the video game Breakout might be to move the paddle randomly. Another, better policy might be to move the paddle in the direction of the projectile, so as to deflect it. The policy of interest, however, is one that the reinforcement learning algorithm developed itself while learning to play. Depending on the architecture of the algorithm, the policy could be a neural network, so that developing the policy equates to training the network. An example of transfer learning for reinforcement learning algorithms, then, could be to apply the trained neural network, or part of it, that was trained in the agent that learned to play Breakout, to a new agent that is learning to play a different but related game, such as Pong. In both games, players must move a paddle to deflect a projectile. The hope is that an abstraction of this concept has been learned in the network, which gets translated and used to make learning other games with similar features more efficient.

A.3 Context

Reinforcement learning algorithms have proven to be successful at achieving at least human-level performance in some tasks. A historical example is TD-Gammon: a program that achieved a level of play just slightly below that of the top human backgammon players of the time (). Another example that is more recent is AlphaGo: a program that is claimed to be arguably the strongest Go player in history (). In the future, an example might be self-driving cars. As computing power increases and new algorithms continue to be discovered (), reinforcement learning research will only grow in popularity, and so it is important to investigate it.

There is one drawback to existing reinforcement learning algorithms: they are most effective when there are no limits to the trial-and-error interactions an agent can make with its environment while learning. This is not an issue in virtual settings, but could cause problems in physical settings. For example, a computer chess program does not feel fatigue and is able to compute millions of games in minutes. On the other hand, an algorithm learning from scratch to drive a car will likely crash on its first run. For reasons such as not wanting to incur a repair cost or wasting time, it is preferable to be confident that a self-driving car has a reasonable ability to drive before testing it. To build confidence that a reinforcement learning algorithm will perform reasonably successfully in a real-world environment, we could first train the algorithm in a similar virtual environment, such as a simulation. Transfer learning is a key component of this process, since the algorithm's subsequent success depends critically on its ability to transfer the knowledge gained from the virtual domain and apply it to the real domain. For reasons such as this, transfer learning has become a crucial technique to build better reinforcement learning systems ().

B Aims

The research question proposed is as follows: *How much more efficiently do reinforcement learning agents with pre-training learn to play a different Atari game than those without?*. To address this research question, the objectives for this project were divided into three categories: minimum, intermediate, and advanced.

The minimum objectives were to establish a candidate reinforcement learning algorithm from the literature which would likely learn good policies for playing Atari games, and implement the algorithm minimally. This minimal algorithm should be used to train an agent in a relatively simple environment, such as CartPole (). In CartPole, the agent is required to prevent an upright pendulum attached to a cart from falling over, and is expected to *solve* the environment by getting an average reward of 195.0 over 100 consecutive trials. The purpose of this objective was to establish a strong understanding of reinforcement learning algorithms, and build a foundational model for the remainder of the project.

The intermediate objectives were to adapt the implemented reinforcement learning algorithm with a deep convolutional neural network and evaluate the benefit of pre-training by policy transfer. The adapted algorithm should be used to train two good agents for two Atari games, Breakout and Pong. In Breakout, the agent is required to break bricks with a ball and paddle (); whereas, in Pong, the agent is required to beat an opponent at table tennis with a ball and paddle (). Solving either environment requires maximising the final score. For Breakout and Pong, the trained agents should attain an average score that at least matches the popular benchmark of 401.2 and 18.9 respectively (). Then, the network weights from the trained Breakout agent should be used to initialise the network weights of a third agent learning Pong, or vice versa, and its performance evaluated against the other trained agent. The purpose of this objective was to understand how reinforcement learning algorithms can process pixels, and whether knowledge can be directly transferred across neural networks.

The advanced objectives were to develop a novel architecture and workflow for pre-training an agent with multiple policies, and evaluate the effectiveness of this approach. The architecture would be a generative model, and would be required to learn from 10 agents trained on 10 different Atari games using the reinforcement learning algorithm implemented earlier. The generative model would then be used to train an agent to play an 11th Atari game, and its performance evaluated against an agent without pre-training. The purpose of this objective was to extend the current state of the art of transfer learning in reinforcement learning, and provide a solution to one of OpenAI's *unsolved problems* ().

C Achievements

We achieved the minimum and intermediate objectives of the project. We developed a minimal PPO algorithm that learned to perform successfully in the CartPole environment. By introducing a convolutional neural network (CNN) to the policy and value networks, the algorithm was able to learn to perform successfully in the environments Pong and Breakout. By reusing the weights in the network of one of the agents, another agent was trained and its performance analysed. We observed there to indeed be a benefit of pre-training. The greater the amount of data, the greater the benefit there was, as measured by three metrics: jump-start performance, accumulated rewards, and final performance. The advanced objective was not able to be completed due to the challenges brought about by external limiting factors, such as Covid-19.

II RELATED WORK (3 PAGES)

There have been several academic achievements in developing reinforcement learning algorithms that address challenging sequential decision-making problems, with potential for real-world applications. For an agent to learn a good behaviour, the agent has to make decisions in an environment to optimise a given notion of cumulative rewards. We focus on existing solutions that achieve this by value-based methods or policy gradient methods. Other approaches include those that combine the two methods, or are model-based.

A Value-based methods

One of the simplest and most popular value-based algorithms to define a policy is Q-learning. The basic version of Q-learning keeps a lookup table of values with one entry for every state-action pair (François-Lavet et al. 2018). In order to learn the optimal Q-value function, the Q-learning algorithm makes use of the Bellman equation for the Q-value function, which has a unique solution. This idea was expanded to involve approximations of the Q-values (Gordon 1995), and Q-values parameterised with a neural network where the parameters are updated (Riedmiller 2005). More recent approaches use deep neural networks as function approximators (Mnih et al. 2015), culminating in the deep Q-learning algorithm (DQN). Many improvements made to the DQN algorithm, incorporating dueling network architectures (), double Q-learning (), prioritised experience replays (), which were all combined to form Rainbow DQN (). Traditionally, the optimal policy was found by finding an exact solution; whereas, modern approaches approximate the optimal policy by using states as input to neural networks.

One limitation of value-based approaches is that these types of algorithms are not well-suited to deal with large action spaces. In the Atari game Gravitar which has 18 actions, a DQN agent achieves a mean score of 306.7 (compared to 2672 by a human) (Mnih et al. 2015).

B Policy gradient methods

Policy gradient methods optimise a performance objective (typically the expected cumulative reward) by finding a good policy thanks to variants of stochastic gradient ascent with respect to the policy parameters (François-Lavet et al. 2018). Basic approaches include using the expected finite-horizon undiscounted return as the performance objective (), and reward-to-go policy gradient, which reinforces actions only on rewards obtained after taking the action () — the latter method makes more sense than the former, which reinforces actions based on *all* rewards ever obtained. Approaches which build on this include those that use the infinite-horizon discounted return as the performance objective, and incorporate an advantage function to describe how much better or worse an action is than other actions on average (relative to the current policy) (). More advanced methods use special constraints, expressed in terms of KL-Divergence, on how close the new and old policies are allowed to be (Schulman et al. 2015), use Taylor expansions to make estimates (Schulman et al. 2015), and take mutiple steps of minibatch stochastic gradient ascent (Schulman et al. 2017). Most of these approaches have an actor-critic architecture, where the actor is the policy and the critic to estimates how good the actor's choices are.

One limitation of policy gradient methods is that it explores by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure. Over the course of training, the policy typically becomes progressively less random, as the update rule encourages it to exploit rewards that

it has already found. This may cause the policy to get trapped in local optima (). Even seemingly small differences in parameter space can have very large differences in performance—so a single bad step can collapse the policy performance. This makes it dangerous to use large step sizes with vanilla policy gradients, thus hurting its sample efficiency. TRPO nicely avoids this kind of collapse, and tends to quickly and monotonically improve performance.

C Application of transfer learning

D Generative models

III SOLUTION (5 PAGES)

A Specification

In order to develop a reinforcement learning system that can successfully operate in its environment, the system must satisfy certain requirements. First, the system would initialise an environment and the agent in it. Second, a reinforcement learning algorithm would control the agent and interact with the environment by making observations and taking actions. Third, the environment would respond with a reward for each action taken by the agent. Finally, the algorithm would process the observations, actions, and rewards to iteratively improve the agent's behaviour.

Additionally, there is another set of requirements for developing a system to pre-train reinforcement learning agents. First, the trajectories from one or more reinforcement learning agents operating in different environments would be generated. Second, a generative model would learn the agents' behaviour by predicting the actions they took. Finally,

B Design

- The algorithm chosen to be implemented was proximal policy optimisation (PPO).
- The pseudocode for this algorithm is as follows.
- For the simple version, the algorithm uses a neural network function approximator.
- For the Atari version, the algorithm uses a convolutional neural network.
- The architectural diagram for the convolutional neural network is as follows.

C Implementation issues

- The original algorithm runs 8 agents in parallel. We only did one due to hardware issues. The architecture was also slightly different.

D Tools used

- Rather than implement everything from scratch, we built on frameworks to allow focus on algorithm design.
- OpenAI Gym, arcade learning environment
- Adam optimiser
- PyTorch

E Verification and validation

- Verification was done by ...
  - Do implementations work there? ...
  - What do I do to judge the outcome/success?
  - Try to answer whether transfer learning is generalisable
- Validation was done by ...

F Testing

[Bridging paragraph]

- Testing was done by ...
  - reproduce on simple problems

IV RESULTS (3 PAGES)

[Bridging paragraph]

A Evaluation method

- The evaluation methods adopted were ...

B Experimental settings

- These were the experimental settings for each experiment carried out: ...

C Results

- The results generated by the software were ...

V EVALUATION (3 PAGES)

[Bridging paragraph]

A Suitability of the approach (more SE, maybe exclude?)

- The approach was/was not suitable because ...
- Was it a good idea to use PyTorch, etc.?

B Strengths and limitations of the algorithm

- The strengths of the algorithm were ...
- The limitations of the algorithm were ...
- The lessons learnt were ...

C Project organisation

- The project was organised as well as you would expect in a global pandemic ...

VI CONCLUSIONS (1 PAGE)

A Project overview

- The project was to ...

B Main findings

- The main findings were as follows: ...
- The conclusions from these findings were ...

C Further work

- The project can be extended by ...

()

References

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G. & Pineau, J. (2018), 'An introduction to deep reinforcement learning', *CoRR* **abs/1811.12560**.
URL: *http://arxiv.org/abs/1811.12560*

Gordon, G. J. (1995), Stable fitted reinforcement learning, *in* 'Proceedings of the 8th International Conference on Neural Information Processing Systems', NIPS'95, MIT Press, Cambridge, MA, USA, p. 1052–1058.

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996), 'Reinforcement learning: A survey', *Journal of Artificial Intelligence Research* **4**, 237–285.
URL: *https://doi.org/10.1613/jair.301*

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**(7540), 529–533.
URL: *https://doi.org/10.1038/nature14236*

Riedmiller, M. (2005), Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method, *in* J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge & L. Torgo, eds, 'Machine Learning: ECML 2005', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 317–328.

Sammut, C. & Webb, G. I., eds (2010), *Encyclopedia of Machine Learning*, Springer US.
URL: *https://doi.org/10.1007/978-0-387-30164-8*

Schulman, J., Levine, S., Moritz, P., Jordan, M. I. & Abbeel, P. (2015), 'Trust region policy optimization', *CoRR* **abs/1502.05477**.
URL: *http://arxiv.org/abs/1502.05477*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017), 'Proximal policy optimization algorithms', *CoRR* **abs/1707.06347**.
URL: *http://arxiv.org/abs/1707.06347*