NUS-ISS – Intelligent Software Agents
Self-learning Systems Project

Intelligent Game-playing Agent Using Advanced Reinforcement Learning Algorithms

Koh Sook Bing	(A0195413E)
Lim Chong Seng Hermann	(A0195392U)
Yeo Whye Chung Nelson	(A0195405A)

Table of Contents

Project Overview

Introduction

Reinforcement learning is one of the three main types of machine learning techniques, the other two being supervised and unsupervised learning. While supervised and unsupervised learning work with prepared datasets for the purpose of prediction or observing characteristics about the data, reinforcement learning interacts with dynamic environments for the purpose of decision-making on what actions to take upon the same environment, and learning from the desirability of outcomes from those actions. In supervised and unsupervised learning, output from the two techniques do not usually affect values in the input dataset or any new data received directly. While for reinforcement learning, actions taken will directly affect the state of the environment.

One major way in which both new and existing reinforcement learning techniques can be explored and tested is in video and computer games. Such games provide a controlled and easily replicable simulated environment with which a reinforcement learning agent can interact over many iterations and episodes within a short time-span and low budgetary cost compared to an actual real-time environment.

At its core, reinforcement learning agents use the immediate reward values of current and past actions to predict the reward values of future action sequences, with the aim of maximising long-term rewards. However, there can be situations where certain actions do not yield any direct reward, but would provide the agent with greater opportunities to earn greater rewards – contingent upon subsequent actions – than it otherwise could. Also, there can be contexts where the objective is to maximise rewards over multiple episodes. Many games can serve as examples of these situations, where it is preferable for an agent to forgo obtaining large rewards in favour of survival – the preferred action of human players - as longer playtimes offer a non-guaranteed potential for attaining even higher rewards.

Q-learning is one of the most popular reinforcement learning techniques in use today, with Deep Q-learning developed in 2014 as the popularity of deep learning grew. Since then, reinforcement learning research has continued and new techniques have since been developed.

Project Objectives

In this project, the team will study the gameplay performance of agents implemented with three reinforcement learning algorithms developed after the introduction of Deep Q-learning. The algorithms are:

- Trust Region Policy Optimisation (TRPO)
- Synchronous Advantage Actor Critic (A2C)
- Proximal Policy Optimisation (PPO)

This report will give a brief overview of each algorithm, discuss the observations made on trends and gameplay performance at different hyper-parameter configurations, and the comparative performance of optimised models for each algorithm.

Technical Approach

Game Description

The game used in this study is titled Space Invaders. First released as an arcade game during the 1970s, it is one of the earliest shooting games developed, and commonly considered to be the highest-grossing video game of all time. The gameplay mechanics is described as follows:

- The player controls a star-ship at the bottom of the screen, moving sideways or shooting upwards
- Five rows of enemy aliens start from the top, moving sideways back-and-forth and gradually advancing downwards, while firing random shots at the player
- The objective of the game is to shoot down all the aliens before they reach the player, while avoiding incoming shots to survive
- When the ship is hit, the player will lose one of three lives, after which the game is over
- Each alien killed awards points, with aliens further up the screen awarding more
- Once all the aliens are shot down, the player progresses to the next level with a new set of aliens moving at higher speeds
- Occasionally, a special alien ship will fly sideways across the top of the screen. A large amount of bonus points will be awarded if it is shot down
- The bottom of the screen, above the player, is also comprised of four “bunkers” which can shield the player from incoming shots, which will gradually disintegrate the shelter.

System Implementation

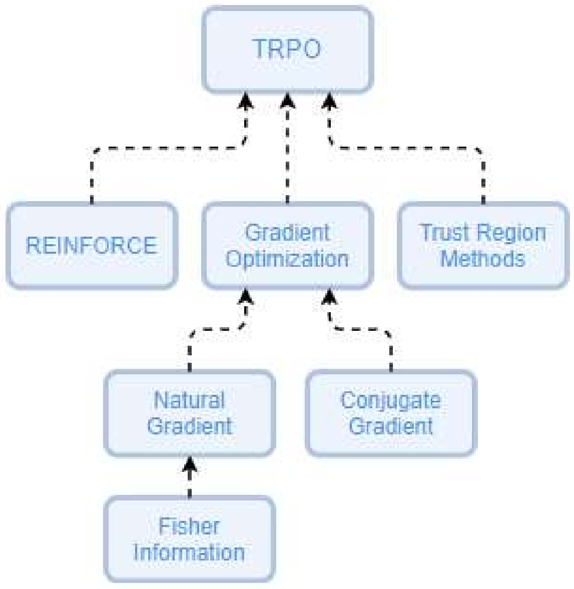
The system is implemented using two main Python libraries:

- gym-retro: Developed by OpenAI, this library supports emulation of Atari, NEC, Nintendo, and Sega game consoles, enabling creation of game environments for about 1,000 games, a much wider range than the classic gym library, which can only emulate Atari2600 games
- stable-baselines: This library is a fork from OpenAI’s baselines library, providing reusable implementations of several reinforcement learning algorithms for the purpose of simplifying the agent development process, so that developers can focus on environmental design and hyper-parameter optimisation. stable-baselines improves upon baselines with some major code refactoring and clean-ups, improved test coverage of the module’s codes, provision of better documentation, and implementation of several additional algorithms

Overview of Algorithms

Trust Region Policy Optimisation (TRPO)

TRPO is a scalable algorithm for optimising policies in reinforcement learning by gradient descent. Model-free algorithms such as policy gradient methods do not require access to a model of the environment and often enjoy better practical stability. Consequently, while straightforward to apply to new problems, they have trouble scaling to large, nonlinear policies. TRPO couples insights from reinforcement learning with optimisation theory to develop an algorithm which, under certain assumptions, provides guarantees for monotonic improvement. It is now commonly used as a strong baseline when developing new algorithms.



Policy Gradient

Policy gradient methods (e.g. TRPO) are a class of algorithms that allow us to directly optimise the parameters of a policy by gradient descent. The goal of reinforcement learning is to find an optimal behaviour strategy for the agent to obtain optimal rewards. The policy gradient methods target at modelling and optimising the policy directly.

Variance Reduction & Advantage Estimate

One major shortcoming of policy gradient methods is that the simplest instantiation of REINFORCE suffers from high variance in the gradients it computes. This results from the fact that rewards are sparse, we only visit a finite set of states, and that we only take one action at each state rather than try all actions. In order to properly scale our methods to harder problems, we need to reduce this variance.

Natural Gradient Descent

While gradient descent is able to solve many optimisation problems, it suffers from a basic problem – performance is dependent on the model's parameterisation. Natural gradient descent, on the other hand, is invariant to model parameterisation.

Conjugate Gradient

The conjugate gradient method (CG) is an iterative algorithm for finding approximate solutions to Ax=b, where A is a symmetric and positive-definite matrix (such as the Fisher information matrix). The method works by iteratively computing matrix-vector products Axi and is particularly well-suited for matrices with computationally tractable matrix-vector products.

Trust Region Methods

Trust region methods are a class of methods used in general optimisation problems to constrain the update size.

Synchronous Advantage Actor Critic (A2C)

The A2C algorithm generally makes use of 2 different algorithms to improve the learning process. The Actor is the component which decides on the action to take, while the Critic is the component that provides feedback for adjustments.

In common methods, learning and adjustments are usually done at the end of episode. In A2C algorithm, adjustments are made during steps within the episode, and not only at the end of it.

A2C works by having the Actor algorithm take an action based on a policy, and the Critic algorithm provide feedback on that action. The Actor algorithm will then use the feedback to improve the policy before taking the next action. At the same time, the Critic algorithm will also update itself to provide better feedback the next time round.

Advantages of the A2C algorithm include accelerated learning and reduced variances.

Proximal Policy Optimisation (PPO)

The PPO algorithm is developed by the OpenAI team, combining ideas like the use of multiple workers in the A2C implementation, as well as the trust region concept in TRPO, to produce a simpler, more easily implementable algorithm. The latest implementations integrate trust region updating with stochastic gradient descent in order to remove the complexity of adaptive KL penalty updating used in earlier PPO implementations.

In this way, PPO is able to retain the scalability and reliability of TRPO but with a much simpler implementation, as well as reducing the amount of effort needed to achieve good results via during hyper-parameter optimisation.

In this project, stable-baselines' PPO2 algorithm is used, which is a GPU-optimised variant.

Results and Evaluation

Introduction

In this section, the report looks at the hyper-parameter tuning process for each algorithm, discussing changes in gameplay performance and trends observed as hyper-parameter values were being tuned. Finally, models of the three optimised algorithms will be evaluated to compare their gameplay performance.

Methodology

When optimising the hyper-parameters, the agents are trained over 100,000 timesteps, and the performance was evaluated based on the trimmed average of scores achieved over 100 episodes, where outliers outside of 3 standard deviations are excluded. This is necessary to discourage the agent from focusing on the bonus spaceships in-game, which would possibly come at the expense of firing at the regular aliens approaching the player, the main objective of the game.

Trust Region Policy Optimisation (TRPO)

For the TRPO algorithm, the hyper-parameters of Policy Model and Discount Factor were explored and optimised. For Policy Model, we compared between the MlpPolicy and CnnPolicy models available from the stable-baselines library. stable-baselines also offer LSTM variants of these models, but they are not explored in this project due to existing issues flagged by the library's developers regarding implementation of LSTM policies with TRPO.

	MlpPolicy	CnnPolicy
Average Score	375.5	114.4

As we can see above, MlpPolicy has a much higher performance than CnnPolicy, at 375.5 vs 114.4. Using MlpPolicy, the agent is further optimised based on Discount Factor.

Discount Factor	Average Score
0.0	216.6
0.1	317.1
0.2	118.5
0.3	182.8
0.4	172.0
0.5	131.6
0.6	211.1
0.7	177.9
0.8	375.6
0.9	185.3
1.0	170.2

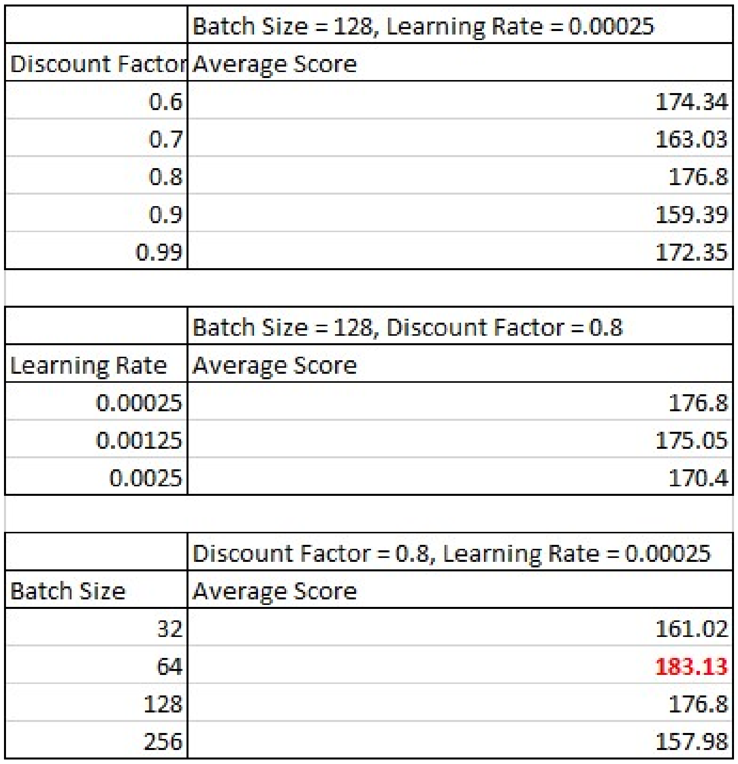
From the above scores, we can see that peak performance is achieved when Discount Factor is 0.8. Performance drops sharply to less than 200 when the Discount Factor is tuned to other values, with the exception of 0.1 where an average score of 317.1 is achieved. Nonetheless, this is still significantly below the average of 375.6 achieved at Discount Factor 0.8.

Based on the above, it is concluded that the optimal hyper-parameters for TRPO are:

- MlpPolicy model
- Discount Factor 0.8

Synchronous Advantage Actor Critic (A2C)

Results of experimentation:



	Batch Size = 128, Learning Rate = 0.00025
Discount Factor	Average Score
0.6	174.34
0.7	163.03
0.8	176.8
0.9	159.39
0.99	172.35

	Batch Size = 128, Discount Factor = 0.8
Learning Rate	Average Score
0.00025	176.8
0.00125	175.05
0.0025	170.4

	Discount Factor = 0.8, Learning Rate = 0.00025
Batch Size	Average Score
32	161.02
64	183.13
128	176.8
256	157.98

In using the A2C algorithm for the Space Invaders game reinforcement model, the hyper-parameters used for tuning were the Discount Factor, Learning Rate, and Batch Size. The results of the experiments are listed in the tables shown above.

Experimenting with several values of the Discount Factor, we picked the best performing value of 0.8, which gave an average score of 176.8 over 100 episodes.

Keeping the Discount Factor at 0.8, we initially used a Learning Rate value of 0.00025. After increasing the value for a number of tests, the result was that as the Learning rate value increased the average score decreased. Therefor e we picked the Learning Rate value of 0.00025 as the best performing option.

For Batch Size, we used an initial value of 128. Increasing the Batch Size value seemed to decrease the performance, therefore we experimented with lowering the Batch Size value. We found that with a Batch Size value of 64, we got an average score of 183.13, the highest average score we have gotten. Lowering the Batch Size Value even further seemed to decrease the score, so we picked 64 as the optimal Batch Size value.

Based on several experiments, we decided that the optimal hyper-parameters to use for our A2C algorithm to be:

- Discount Factor: 0.8
- Learning Rate: 0.00025
- Batch Size: 64

Proximal Policy Optimisation (PPO)

For PPO, we looked at four hyper-parameters: Policy Model, Discount Factor, Batch Size, and Learning Rates. For the six available Policy Models implemented in the stable-baselines library, the average performances are shown as follows:

Policy Model	Average Score
MlpPolicy	181.0
MlpLstmPolicy	371.2
MlpLnLstmPolicy	379.8
CnnPolicy	172.7
CnnLstmPolicy	119.3
CnnLnLstmPolicy	130.4

LSTM variants of the MLP policy models performed much better than the other policy models, scoring above 370 while the other models scored less than 200.

Discount Factor	Average Score
0.0	177.8
0.1	399.3
0.2	165.1
0.3	113.7
0.4	320.7
0.5	316.3
0.6	138.2
0.7	181.8
0.8	385.1
0.9	123.5
1.0	145.3

When optimising the Discount Factor, it can be observed that the values fluctuate between various highs and lows as the Discount Factor changes in value. The highest values are achieved when Discount Factor is 0.1 and 0.8, while a slightly lower high occurs at 0.4-0.5. Performance at other values are low, with an average score below 200.

Batch Size	Average Score
64	113.4
128	399.3
256	350.8
512	330.4
1024	202.9

At Batch Size 64, the agent performs very poorly, achieving a score of only 113.4. Performance peaks at 399.3 when Batch Size is 128, and gradually declines at higher values.

Learning Rate	Average Score
0.1	0.0
0.01	360.3
0.001	387.2
0.0001	191.4
0.00001	206.5

The Learning Rate of 0.1 is too large, and the agent was unable to learn anything at all, resulting in an average score of 0. At 0.001, the agent achieved the best average score of 387.2. At lower values, the learning rate dropped sharply, indicating that the subsequent Learning Rates are too small and the agent was learning too slowly.

The optimal hyper-parameters for PPO was concluded to be:

- Policy Model: MlpLnLstm
- Discount Factor: 0.1
- Batch Size: 128
- Learning Rate: 0.001

Final Evaluation

After tuning the hyper-parameters, the three optimised algorithms are trained over 5,000,000 timesteps, with the agents' performances evaluated per million timesteps. The resultant average and highest scores are shown below:

Timesteps Trained	TRPO		A2C		PPO	
	Average	Top	Average	Top	Average	Top
1,000,000	398.5	680	174.9	520	295.3	810
2,000,000	341.8	540	156.3	520	312.2	590
3,000,000	335.5	790	175.4	540	219.7	380
4,000,000	336.6	760	168.3	380	326.4	650
5,000,000	333.8	590	188.6	520	110.2	290

From the above results, we can observe that A2C performs significantly worser than TRPO and PPO at every stage of evaluation, and TRPO also outperforms PPO at most times. While performance of TRPO and A2C are relatively consistent across each of evaluation, PPO's performance drops sharply after 4,000,000 training timesteps.

Conclusion

In this study of different reinforcement learning algorithms, the team gained a more in-depth understanding of the algorithms, and also better analytical skills in observation and experimentation. Through using a methodical approach of hyper-parameter tuning, the team managed to find optimise the search for optimal settings, by focusing on good performers.

Bibliography

https://www.freecodecamp.org/news/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d/

https://medium.com/deeplearningmadeeasy/advantage-actor-critic-a2c-implementation-944e98616b

https://arxiv.org/abs/1602.01783

https://medium.com/@jonathan_hui/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

https://arxiv.org/abs/1707.06347

https://arxiv.org/abs/1502.05477

Annex A: Setup Instructions

This project was carried out in a Windows environment, and the instructions are based on this environment, unless otherwise stated.

1) Install MPI for Windows (https://www.microsoft.com/en-us/download/details.aspx?id=57467) or Open MPI for other operating systems (https://www.open-mpi.org/)
2) Install Python 3.6 and the following packages (as listed in requirements.txt):
   a) gym-retro==0.8.0
   b) stable_baselines==2.10.0
   c) mpi4py==3.0.3
   d) tensorflow==1.15.3
   e) tensorflow-gpu==1.15.3
3) Import the Space Invaders' game into gym-retro. The game's ROM file ("Space Invaders (USA).sfc") is provided within the project folder. In the command line, enter the following:

   ```
   python -m retro.import /path-to-project-folder/
   ```