Evaluation of deep reinforcement learning and its application through a case study in computer games

by

­John Gregg-Webster

Progress Report



Department of

Computer and Information Sciences

November, 2023

CONTENTS

Contents

[1 Aims and Objectives 1](#_Toc2277)

[2 Summary of Related Work 2](#_Toc2278)

[3 Project Specification 4](#_Toc2279)

[4 Project Plan including summary of progress 6](#_Toc2280)

[5 Approach to Development 7](#_Toc2281)

[5.1 Methodology 7](#_Toc2282)

[5.2 Design **Error! Bookmark not defined.**](#_Toc2283)

[5.3 Implementation and Technology ideas 7](#_Toc2284)

[5.4 Testing and Evaluation plans 7](#_Toc2285)

ii

1 AIMS AND OBJECTIVES

# Aims and Objectives

The objective of the project is to produce insights about the strengths and weaknesses of various deep reinforcement learning approaches in the context of a particular task. To this end, the project objectives are to;

* Implement a simple computer game.
* Implement, by the use of deep reinforcement learning, agents capable of playing the game.
* Implement software to empirically measure and compare the effectiveness of those agents.
* Implement software to visualize the experimental results.
* Produce a written report presenting the results.

2 SUMMARY OF RELATED WORK

# Summary of Related Work

Reinforcement learning is one of three commonly accepted branches of machine learning; differentiated from supervised, and unsupervised learning. In reinforcement learning, the system is trained to perform a desired input-output mapping by applying a reward signal that acts to reinforce or weaken the system’s association between a given input and output.

Reinforcement learning is of particular interest because of it allows for the creation of control systems in novel environments, where the details of the optimal policy are not known, or where the data necessary to apply supervised learning is not available.

In value-based methods the agent learns to estimate the value of the actions it can take in a given state, and the associated policy can be enacted by choosing the action with the highest estimated value. Under a policy-based approach the agent learns to produce a probability distribution of actions at each state, which can used to enact the learned policy [1]. A third class of agents use actor-critic methods, which combine of both approaches [2].

Early approaches were generally tabular. That is, they achieve the desired input-output mapping by constructing a look-up table containing every possible input state, which is held in memory. Tabular methods are backed by theoretical proofs of their ability to converge to optimal policies [3, 4]. Despite this useful property, such methods are not typically practical due to the large state-spaces of real-world machine-learning problems. The associated table would be so large that storing it and updating it during training would be computationally infeasible.

In contrast, modern approaches typically use neural networks to approximate the mappings described by tabular methods. While perceptrons have been proven to converge to the optimal policy over time [5], neural networks in general are not guaranteed to converge, or even to improve. Though in practice they often do.

The use of neural networks with hidden layers to solve reinforcement learning problems is known as deep reinforcement learning, and in the last decade it has been applied to achieve human-level performance in complex control problems [6, 7].

Modern RL encompasses a diverse variety of approaches, and there are a variety of overlapping categories by which reinforcement learning algorithms can be classified. The taxonomy of those approaches is of interest if they are to be meaningfully compared.

Table 1: Summary of RL Methods

|  |  |  |
| --- | --- | --- |
| Class | Technique | Notes |
| Value-Based | Deep Q-Learning. | Off-policy temporal difference method. |
| SARSA | On-policy temporal difference method. |
| Policy-Based | REINFORCE | Runs a full episode, uses the future returns of each decision to improve the policy by gradient ascent. |
| Proximal Policy Optimisation |  |
| Trust Region Policy Optimisation |  |
| Actor-Critic | Advantage Actor-Critic |  |
| Adversarial Advantage Actor-Critic |  |

3 PROJECT SPECIFICATION

# Project Specification

**The Problem.**

The goal of the project is to gather empirical data about the effectiveness of various reinforcement learning approaches, when applied to particular game-playing tasks (described below).

**The Game Environment.**

The environment is a turn-based game, where the player is tasked with navigating a maze. The maze is grid-based, composed of empty tiles through which the player’s avatar can move, and solid squares that it cannot enter. The player gains score by navigating the maze to collect coins.

* Each coin is placed in a random unoccupied square with equal probability.
* A set number of coins are placed when the game starts.
* An additional coin is placed each time a coin is collected.

Each game round lasts for a configurable number of turns, after which the round ends and the player’s score (equal to the number of coins collected) is displayed.

If the project timeframe allows it, the following additional features will be implemented;

* Enemy agents within that hunt the player, ending the game on contact.
* Random generation of mazes.
* Functional Requirements.
  + The user can initialize the environment, optionally providing settings.
    - Random seed.
    - Round duration.
    - Number of coins.
    - Number of hostile agents.
    - Rendering on/off (for AI training).
  + The player can act within the environment.
    - Move up.
    - Move down.
    - Move left.
    - Move right.
* Non-Functional Requirements.
  + Should perform well.
  + Should be deterministic with respect to a given random seed.
  + Should be playable by humans.

If the project timeframe allows it, multiple environments may be designed and implemented. Though the precise details of other potential environments are out of scope at this time. This could be accomplished by creating multiple sets of agents each capable of playing a single game, or by creating agents capable of playing multiple games.

**Training System.**

* Functional Requirements.
  + The user can train an agent.
  + The user can set the parameters of the training.
    - Number of Episodes.
  + The user can set the hyperparameters of the agent.

**Evaluation System.**

* Functional Requirements.
  + The user can set the parameters of the evaluation system.
    - Agents to evaluate.
    - Number of Episodes.
  + The user can evaluate an agent (or set of agents).
* Non-Functional Requirements.
  + Evaluations should be deterministic with respect to a given random seed.

**Visualisation System.**

* Functional Requirements.
  + The user can visualize a provided evaluation dataset.

4 PROJECT PLAN INCLUDING SUMMARY OF PROGRESS

# Project Plan including summary of progress

A review of the existing work on the topic has been performed, and several prototype models have been produced simpler control problems (a multi-arm bandit environment, and the OpenAI-gym’s cartpole environment).

Moving forward, the current plan is to begin work on the environment concurrent with evaluation tools, then begin implementing agents, as depicted on the chart below.

A graph with colorful rectangles

Description automatically generated

Gantt chart of predicted project timeline.

5 APPROACH TO DEVELOPMENT

# Approach to Development

## Methodology

AGILE principles will be followed throughout development. Many techniques typically branded as AGILE are intended for use in team environments (stand-ups, appointment of scrum masters), so due to the nature of the project this will mainly manifest in the use of sprints. During each sprint, the development team progresses through all three stages of; design, implementation, testing, with the goal of implementing a particular feature. The structure of the project is convenient for this approach as the implementation of each agent is a self-contained goal, appropriate in scope for a single sprint.

## Implementation and Technology ideas

The solution will be implemented in Python as it is the industry standard for machine learning. TensorFlow will be used due to the developer‘s familiarity with the library.

## Testing and Evaluation plans

Due to the nature of the software, the robustness of the environment is of particular importance. Automated agents are capable of finding and exploiting implementation errors and edge cases, which could render the game unexpectedly easier and make it more difficult to interpret the results. It is especially difficult to determine whether this has occurred, as it is not feasible to manually inspect training episodes for unexpected behaviour.

The implementation should feature comprehensive unit testing of all its components. The environment should be broken down into components that can be unit tested, in addition to integration testing.

A test system will be also used to evaluate whether the agents have been implemented correctly, by verifying that agents achieve the expected level of competence at the target task. Lack of improvement, or failure to score better than a random policy, though not sufficient to prove the existence of an implementation error, could indicate the presence of errors. It seems likely that features of the test system will overlap with the evaluation and visualisation system required for the final report.

REFERENCES

# References

|  |  |
| --- | --- |
| [1] | R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Second Edition, MIT Press, 2018. |
| [2] | V. Konda and J. Tsitsiklis, “Actor-Critic Algorithms,” *Advances in neural information processing systems,* vol. 12, 1999. |
| [3] | P. Dayan and C. J. Watkins, “Q-Learning,” *Machine Learning,* vol. 8, pp. 279-292, 1992. |
| [4] | R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning,* vol. 8, p. 229–256, 1992. |
| [5] | A. Novikoff, “On Convergence Proofs For Perceptrons,” *Proceedings of the Symposium on the Mathematical Theory of Automata,* vol. 12, no. 1, pp. 615-622, 1963. |
| [6] | M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” *Proceedings of the AAAI conference on artificial intelligence,* vol. 32, no. 1, 2017. |
| [7] | V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv preprint arXiv:1312.5602,* 2013. |