chizuru4rogue: Deep Learning for Deep Dungeon Crawling

Dylan G. Drescher

B.Sc. Computer Science - University of Bath

2022 - 2023

Abstract

In this article we introduce chizuru4rogue, which is a computer program designed to play the video game Rogue, a famous role-playing game that inspired the creation of the "roguelike" video game genre. Rogue offers a unique problem to solve, requiring a player to solve a partially observable, randomly generated levels. chizuru4rouge utilises a *** to explore levels in Rogue, collect gold and reach the goal.

TensorFlow will be used as a framework to implement the reinforcement learning agent. TensorFlow is a Python library that provides tools to streamline development of deep learning models.

Declaration of Access

This Dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Contents

List of Figures

List of Tables

Acknowledgements

To-do

1 Introduction

TODO introduction goes here lmao

2 Literature Review

2.1 Fundamentals

The fundamentals of reinforcement learning and many fundamental algorithms is explained in detail by Sutton & Barto (2018). The core idea behind reinforcement learning algorithms is that an agent performs *actions* on an *environment* by deriving what it should do from its *policy*, which is a mapping from states to actions. Once the agent performs an action, it receives the new game state as well as a *reward* signal, telling the agent how good its choice was.

The purpose of rewards is for the agent to constantly estimate a *value function*. This function tells the agent either how profitable being in a certain state and following its policy is, or how profitable taking a certain action then following its policy is. The theory is that the agent should aim to maximise its reward over the long term.

One of the most well-known reinforcement learning algorithms is the Q-learning algorithm (Sutton & Barto 2018, Chapter 6.5). In this algorithm, the agent keeps track of a table, mapping state-action pairs to its value. When the agent reaches a certain state, it consults its Q-table to determine the most valuable action to take.

2.2 Deep Learning

While the Q-learning algorithm can solve simple problem domains sufficiently, when it comes to more complex domains that don't have fully observable states such as Atari games, the amount of resources it takes to run the algorithm can become extremely large. The way that Mnih et al. (2015) chose to solve this is to use a convolutional neural network to approximate the Q-learning action-value functions, through an algorithm the authors call "Deep Q-network".

The Deep Q-network in their writing was shown to play several Atari games to a superhuman level, most notably Video Pinball and Boxing. A similar algorithm involving neural networks was employed by Silver et al. (2016) in their development of the AlphaGo system, an agent that was found to beat human grandmaster players in the game of Go. The authors used a convolutional neural network alongside "policy gradient" reinforcement learning, where

While the DQN algorithm by itself is serviceable for simpler problem domains such as Atari, there are better methods to tackle more challenging

domains. When trying to create an agent that plays the online game Dota 2, Berner et al. (2019) used a Long Short-term Memory network.

LSTMs were first defined by Hochreiter & Schmidhuber (1997) and improved upon in later works. An LSTM is an extension of the "recurrent" neural network, where nodes use feedback connections to allow the network to "remember" information in the long term. This solves the problem that traditional neural networks have, where they can't store information that can be useful to them in the long term.

2.3 Exploring Rogue

The first notable instance of a program being developed to play Rogue was by Mauldin et al. (1983), where they created "ROG-O-MATIC", an expert system that plays Rogue. An expert system, as stated by Jackson (1986) in their book's introduction, "is a computing system capable of representing and reasoning about some knowledge-rich domain". Essentially, these systems aim to emulate a human expert in a particular domain and their decision-making. While expert systems are artificial intelligence, they make no use of machine learning to learn and adapt to situations, they follow what instructions have been programmed within them and are designed to rigidly solve one problem domain. ROG-O-MATIC provides a good yardstick to measure the performance of the agent we will be creating.

An interface for machine learning agents to play Rogue has been created, called Rogueinabox (Asperti et al. (2017)). Rogueinabox is a framework that allow developers to create agents that interface with the game Rogue. In the Rogueinabox article, the authors ran a Deep Q-learning agent on the game for testing. They simplified the problem domain to have the agent only consider exploring the dungeon to find the stairs, without fighting or collecting items. Their agent performed reasonably well accounting dungeon exploration alone, however, the aim of our agent is to reach the Amulet of Yendor and clear the game, which is difficult if the player does not fight monsters and gets stronger.

The initial agent proposed in the original Rogueinabox paper was further improved upon (Asperti et al. (2018)). The problem domain was still simplified to only consider getting to the exit stairs alone. While the previous implementation employed a DQN, the agent in the improvement implemented an A3C algorithm as a base, rather than a DQN. The A3C algorithm in the improvement was partitioned, meaning the sample space is *partitioned* into a set of situations. This allows the different agents that run simultaneously to learn from different situations to build a common cumulative reward. It also involves the work by Jaderberg et al. (2016). The A3C algorithm is first defined by Mnih et al. (2015). It is an asynchronous algorithm that aims to optimise a policy and estimate a value function by training multiple actors in parallel.

2.4 Exploring Other Roguelikes

Rogue is not the only roguelike that has been explored with machine learning. NetHack is one of the most popular games that has been explored with neural networks. NetHack is a roguelike game created in 1987 and is still being updated to this day, with a small but dedicated player-base.

SkillHack (Matthews et al. (2022)). NLE (Küttler et al. (2020)). An article by Izumiya & Simo-Serra (2021) explores how to involve the item inventory in the neural network system of a deep reinforcement learning agent with an attention-based approach. It is attention based as the system calculates a score for each item in an inventory using an "attention function"

3 Concepts

3.1 Reinforcement Learning Concepts

3.2 Rogue Concepts

Rogue is a 1980 role-playing computer game inspired by text-based adventure games and tabletop role-playing games like Dungeons and Dragons that led to the creation of "roguelikes" - games that are based off of the core gameplay of Rogue. Roguelike games are mainly characterised by challenging, turn based hack and slash gameplay, procedurally generated levels and permanent character death.

3.2.1 Objective

In Rogue, your objective is to descend the Dungeon of Doom to slay monsters, collect gold coins, retrieve the Amulet of Yendor and escape the dungeon with it alive. The game is turn based, which means the player can spend as long as they want thinking their next move before the game processes the environment. Figure 1 depicts an example screenshot of the game.

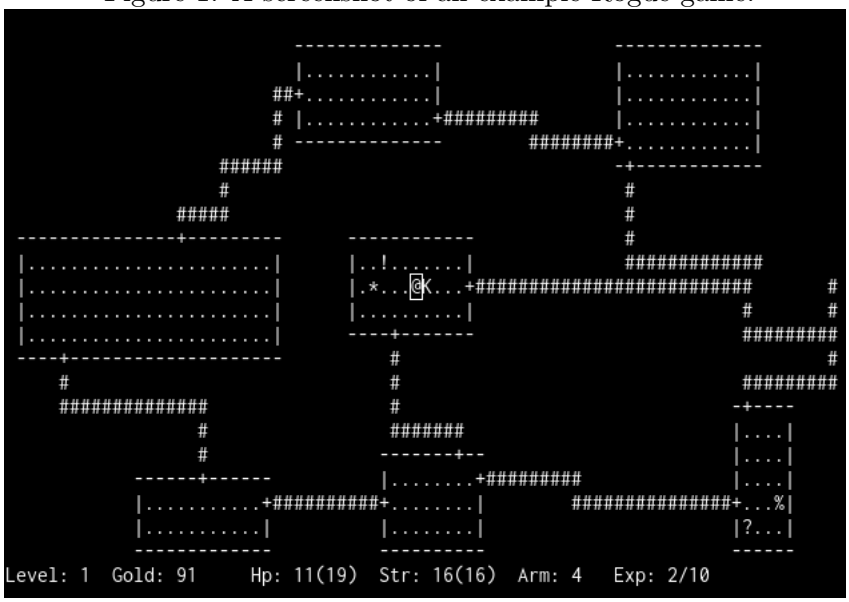
3.2.2 Environment

Every floor of the dungeon is a randomly generated level consisting of several rooms connected with corridors. Rooms sometimes generate empty, but they may also generate populated with several items or enemies. When the player starts a new run, the player is placed in dungeon level 1 with some food, a mace, basic armour, a bow and arrows.

Rogue's environment is partially observable. The dungeon configuration is initially obscured to the player, revealing itself as the player moves around. In addition, enemies on the map will only be shown to the player if the enemy is within the player character's line of sight.

The game tracks several things that are always shown to the player:

Figure 1: A screenshot of an example Rogue game.



- **Level** denotes the current dungeon level.
- **HP** (Hit Points) represents how much damage the player can take before death. The number in brackets is the player's maximum HP.
- **Str** (Strength) represents how strong the player is. The number in brackets is the player's maximum strength.
- **Gold** is how many gold coins the player has collected.
- **Arm** (Armour) is the player's current armour rating. Higher is better.
- **Exp** shows the player's experience level and total experience points. When the player earns enough experience points, the player's experience level increases, increasing the player's maximum HP.

3.2.3 Items

There are a wide variety of items the player can use, such as potions, scrolls, weapons and armour. Some items need to be identified before the player knows what it will do. This can either be done by using a scroll of identify, or by blindly using or wearing the item, which may be risky. Some potions have negative effects such as the potion of poison, and rings may be "cursed". Cursed rings may not be removed once equipped, and they reduce the player's stats. Curses can be removed with the scroll of remove curse.

3.2.4 Combat

As the player navigates around the dungeon, they will encounter enemies of increasing difficulty. Enemies in the game will try to harm the player by attacking and reducing the player's HP. If the player's HP reaches 0, the player loses the game.

The player can attack enemies by moving into them. This will make the player hit the enemy with their equipped weapon. Each weapon in Rogue deals a different amount of damage, so it is important to find and equip stronger weapons.

If the player defeats an enemy, they are granted "experience points". When the player earns enough experience points to increase their player level, their HP and Strength increases, making them stronger.

4 Network Architecture

The objective of the neural network for Chizuru is to take in the observed dungeon map, player status, recent message and inventory as inputs and return an action that will maximise the expected reward as output as if it were maximising an action-value function.

4.1 State

We use the following information to represent game state:

- The player's status - HP, strength, EXP and other attributes.
- The game map, a 21 x 79 array of ASCII characters.
- A 9 x 9 crop of the map centred around the player
- The inventory of items.
- The items the player is equipped with

4.2 Action

Every action in Rogue is available to the player from the start of the game. Actions can be divided into basic actions and actions that utilise an inventory item, depending on the action and the item type. For example, the "eat" action can be used to make the player eat something in the inventory. If the player attempts to use an action on an item where it wouldn't make sense, the action fails and a message is displayed to the player, such as 'Ugh, you would get ill if you ate that'.

When the player uses an action that utilises an item, the game will await the player to input a key. Every item in the player's inventory maps to one

key on the keyboard. The player may input * in order to see what legal items they may choose and their corresponding key. Additionally, the player may see what the item to keyboard mapping is by viewing their inventory with the `i` key at any other point during the game.

4.3 Policy Optimisation

Our goal was to find an optimal policy that maximises the chance that the agent can successfully reach the 26th dungeon level and get the Amulet of Yendor.

4.4 State Representation

The agent will use Rogueinabox to interface with the game. Rogueinabox is a program

4.5 Reward Representation

4.6 Neural Network

4.7 Hyperparameters

5 Implementation

5.1 Language

The agent will be implemented in Python. Python is one of the most popular languages used to model neural networks due to the large amount of artificial intelligence related libraries that are available for the language. The main library we will be using is TensorFlow, a library that streamlines the creation of machine learning models in Python by providing the programmer with tools to construct models, visualise data and more.

6 Agent Training and Investigation

6.1 Evaluation

During our training of the agent, we measured the agent's performance with the following criteria after every run:

- The final score the agent achieved
- The deepest dungeon level the agent entered

7 Conclusion and Future work

chizuru4rogue achieves its goal on being an improvement to Asperti et al. (2018)'s simple navigation by being able to use items in game and fight monsters.

For future developments of the model, we plan to use *** to *** because ***

8 Reflection

References

Asperti, A., Cortesi, D. & Sovrano, F. (2018), 'Crawling in rogue's dungeons with (partitioned) a3c'.

Asperti, A., De Pieri, C. & Pedrini, G. (2017), 'Rogueinabox: An environment for roguelike learning', *International Journal of Computers* **2**, 146–154.

Berner, C., Brockman, G., Chan, B., Cheung, V. et al. (2019), 'Dota 2 with large scale deep reinforcement learning'.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Izumiya, K. & Simo-Serra, E. (2021), Inventory management with attention-based meta actions, *in* '2021 IEEE Conference on Games (CoG)', IEEE, pp. 1–8.

Jackson, P. (1986), *Introduction to expert systems*, Addison-Wesley Pub. Co., Reading, MA.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D. & Kavukcuoglu, K. (2016), 'Reinforcement learning with unsupervised auxiliary tasks', *arXiv preprint arXiv:1611.05397* .

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E. & Rocktäschel, T. (2020), 'The nethack learning environment', *Advances in Neural Information Processing Systems* **33**, 7671–7684.

Matthews, M., Samvelyan, M., Parker-Holder, J., Grefenstette, E. & Rocktäschel (2022), 'Hierarchical kickstarting for skill transfer in reinforcement learning'.

Mauldin, M., Jacobson, G., Appel, A. & Hamey, L. (1983), 'ROG-O-MATIC: A belligerent expert system'.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A. et al. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**, 529–533.

Silver, D., Huang, A., Maddison, C. et al. (2016), 'Mastering the game of go with deep neural networks and tree search', *Nature* **529**, 484–489.

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement learning: an introduction*, MIT Press.

A Methods

B Results

C Data