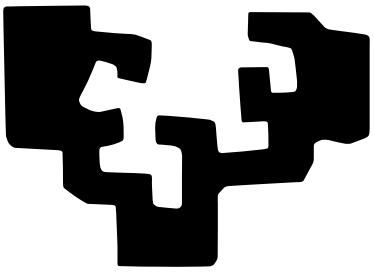
Advanced Techniques in Artificial Intelligence

Counting to Explore and Generalize in Text-based Games

Author:
Xabier Benavides Canta
David Sandström Daparte

October 6, 2019

eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Abstract

Among this report, we will cover the most important points extracted from the here linked document, used as main document for the investigation.

This document proposes a Reinforcement Learning (RL) model with count-based exploration mechanisms that help discovering good policies in text-based game environments. The main improvement of the model we propose is that it can be **generalized** to unseen environments.

First, we will explain all the important concepts that are needed to understand the agents, and then we will explain the main aspects of the proposed model and the experiments used to test it.

As the last part, we will cover as well the most important sections of the implementation that is provided with the document previously mentioned.

List of Contents

1 Introduction

Text based games are complex and interactive simulations that use natural language to describe the state of the world. Because of that, the player only has a partial observability of the environment, and has to deal with ambiguous and incomplete information while trying to achieve the goal of the game.

There are many different text games, but in this report we will work with a text based version of the "chain experiment". In other words, the game will be composed of a certain set of nodes connected by paths creating a chain with some distractor nodes that are off-chain (Without creating any cycle). Our agent will start in one of the edges of the chain, and it's goal will be to find a coin that is in the other edge, ideally without revisiting the off-chain "dead end" nodes.

To simplify our game, the agent will work with two word text commands. Each command is a (**verb**, **object**) pair, and the allowed verbs and objects are the following:

- **Verb**: go, take.
- **Object**: coin, south, north, east, west.

However, not all of the verb-object combinations are understood by the game. The only commands that are actually useful for the agent are: **go north**, **go east**, **go south**, **go west** and **take coin**.

In order to apply a Reinforcement Learning (**RL**) algorithm to solve this game, we propose a Deep Q-Network (**DQN**) based model. To encourage exploration, we use a count-based reinforcement that assigns a reward derived from the count of state visitations during learning. In contrast to other approaches, our model is able to learn policies that are useful to solve **unseen games**.

2 Explanation of the main Concepts

To fully understand the entire model that we are presenting, it is necessary to understand first a few concepts. This concepts will be very summarised, since they are not the main target of this report.

2.1 Used Neural Network models

1. **MLP**: this is a type of Artificial Neural Networks that consists in, at least, three layers of nodes:
   (a) One input layer.
   (b) One hidden layer.
   (c) One output layer.

   This model can grow, by simply adding more hidden layers. For the learning process, this model uses a **back propagation** algorithm.

2. **LSTM**: this is a type of Recurrent Neural Networks (RNNs). This type of Neural Networks are meant to process sequences of data, instead of isolated data instances.

   In order to process sequences of data, the neurons not only have connections to the next layer, but they also have connections in a "temporary order", so that neuron k of layer a is not only connected to the neurons in layer $a + 1$ but to the neuron $k + 1$ in layer a.

2.2 Markov Decision Process (MDP)

Each and every state that the environment goes through is a consequence of the previous state that the environment was in, that turns out to be a consequence of the previous one. This property will be satisfied up to the initial state.

This previous condition exposes a very important problem: we can not store all that information even if the environment turns to be small. To resolve this, we assume that each state follows a **Markov Property**.

Each state depends only on the previous state and the transition from the previous state to the current one. As we can see in the following example:

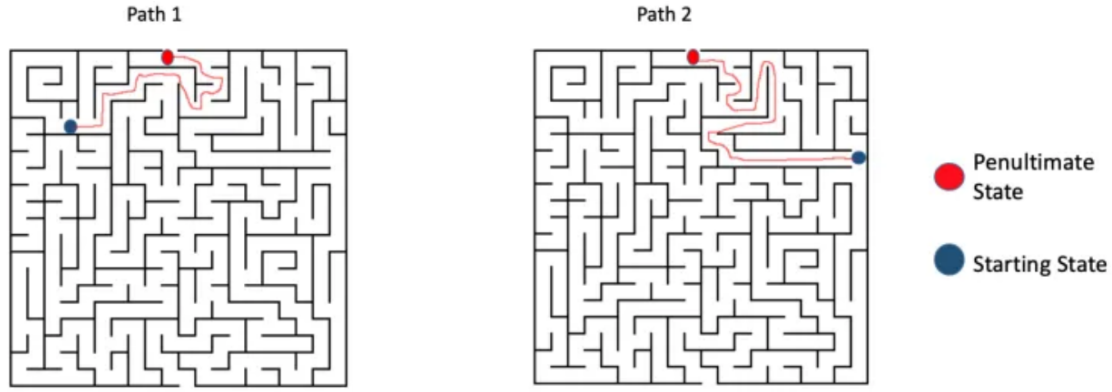


Figure 1: Example of Markov Property.

In both scenarios we have different starting points and therefore different ways to reach the solution. Anyhow, the way to the exit is going up for any of them just looking at the red state regardless of the way used to reach that state.

2.2.1 POMDP

A **Partially Observable Markov Decision Process** (**POMDP**) is a subtype of the Markov Decision Process. A POMDP follows the Markov Property just like a regular MDP, but the main difference is that the agent doesn't have access to the complete underlying state. Instead, the model needs to maintain a probability distribution over the set of possible states.

The text game we are going to work with is an example of a POMDP.

2.3 Deep Q-Learning

The concept of **Q-Learning** consists in a technique used in **Reinforcement Learning**. This algorithm works well in any finite **MDP**, and its goal is to learn policies that guide the agent through the environment by searching for the best possible action in a given state.

In order to select the next action, the agent tries to maximize the total reward over all the possible successive steps starting from the current state. The **Q-value** names the function that returns the reward used to provide reinforcement, and can be interpreted as the "quality" of an action in a certain state.

Deep Q-Learning is a technique used when our agent has to deal with an environment large enough that we can not save all the states and actions at the same time. This way, instead of calculating the Q-values for all the possible actions and states, we will approximate them as a neural network model.

3 Model Explanation

In this report, we work with two different model architectures: **LSTM-DQN** and **LSTM-DRQN**. Both of the models are based in a Deep Q-Learning approach.

The LSTM-DQN model processes the current observation using two MLPs to calculate the Q-values of all the verbs and objects independently. The average of the result of each verb and each object gives the Q-value for the composed actions. As the LSTM-DQN is not able to remember the previous actions or observations during a game, we concatenate the previous command to the current observation before processing it to improve the performance of the model.

The LSTM-DRQN recursive model solves the limitation of the LSTM-DQN model by replacing the two MLPs by an LSTM cell. This LSTM cell takes the observation together with the history information from the previous step of the game. Then, it produces the information of the current step, and feeds it into two MLPs to obtain the Q-values in the same way as in LSTM-DQN architecture. The information of the current step is also passed forward to the next step of the game.

To promote the exploration in the RL agent, we use a reward system that is based on counting the number of times a state has been visited before. We propose two different count-based strategies: A cumulative counting bonus and a episodic discovery bonus.

- The **cumulative counting bonus** reward is defined as $r^{+}(o_t) = \beta \cdot n(o_t)^{-1/3}$, where $n(o_t)$ is the number of times the agent has seen o_t since the ***beginning of the training***, and β is the bonus coefficient.
- The **episodic discovery bonus** reward is defined as $r^{++}(o_t) = \begin{cases} \beta & \text{if } n(o_t) = 1 \\ 0.0 & \text{otherwise} \end{cases}$, where $n(o_t)$ is the number of times the agent has seen o_t in the ***current episode***. Like in the previous strategy, β is the bonus coefficient.

In short, we are going to work with two model architectures (DQN and DRQN) and two reward systems to determine if any of them is helpful to teach the agent how to generalize its behaviour to unseen environments.

4 Experiment

From now on, we will use the following notation to refer to the different types of RL models that we are going to test:

- **LSTM-DQN+** (LSTM-DQN with cumulative counting bonus).
- **LSTM-DQN++** (LSTM-DQN with episodic discovery bonus).
- **LSTM-DRQN+** (LSTM-DRQN with cumulative counting bonus).
- **LSTM-DRQN++** (LSTM-DRQN with episodic discovery bonus).

The four RL models were tested in the text game explained in the introduction of this report. The game has three different modes: **easy** (No distractor nodes along the chain), **medium** (Each node

along the chain has one distractor node) and **hard** (Each node along the chain has two distractor nodes). In addition, we use difficulty levels to indicate the optimal path's length of a game.

1. **Training**

   In order to train the models, we randomly generated game sets that contained a random number of level 10 games for each game mode (Easy, medium and hard).

   We could notice that when the games became harder or more training games were provided, the episodic bonus had an advantage over the cumulative bonus, and recurrence became more crucial for memorizing the game distribution.

2. **Testing**

   In order to check if the pre-trained models could be generalized to unseen games, we tested the models in a test set composed of ten random level games for each difficulty mode. It's important to emphasize that all the games in the test set were unseen games.

   To simplify, the tests were performed using only the models that had produced better results during training: **LSTM-DQN++** and **LSTM-DRQN++**. At test time, the verb and noun actions were selected based on the argmax of the Q-values computed during training. These were the results of our tests:

   - First, we noticed that both models were able to generalize well on unseen easy games. However, the **LSTM-DRQN++** performed better than **LSTM-DQN++** when trained in fewer games.
   - When testing on hard mode games, we discovered that the two models seem to suffer from overfitting. In order to avoid this phenomenon, we generated a validation set that contained ten hard games of level 10.
   - Using the validation set to avoid overfitting, we noticed that the **LSTM-DQN++** model was able to perform near perfectly in unseen hard games. In fact, the agent learned a general strategy that didn't need any information beyond the previous command to solve unseen hard games.

5 Implementation Review

The implementation that we are going to review can be downloaded from the following repository. To run this code we will need to install the python library *textworld* which, by the current time, is only available for Mac and Linux. Check this link for compatibility. This implementation is a very complete one, consisting of the following components:

1. A script that allows us to create a new world for our agents. This world can be generated with the desired difficulty.
2. An implementation for the agent based on the **LSTM-DQN** model.
3. An implementation for the agent based on the **LSTM-DRQN** model.

Besides those, it is possible to find other python scripts in the repository, allowing us for example to delete the duplicate elements among the train and test sets.

5.1 Tests Performed

After trying to execute all the possible options given by this implementation we obtained the following results:

1. After a lot of compatibility problems with the libraries, many of them related to the libraries *gym* and *gym_textworld*, we were finally able to make the world creating script run. We were only able to make it run in a Linux platform.

   As a result of that, we could generate a "world" consisting on a JSON file and other two ones. The bad news are that we were not able to use this generated world at all.

2. We tried both model implementations of the agent but we ran into similar problems as with the previous part. There were missing library elements that we could not find or that simply didn't exist in the versions of the libraries that we had access to.

Apart of that, the code was very hard to read because there wasn't almost any comment all along the implementation.

6 Conclusion

In conclusion, Deep Q-learning models that use count-based approaches show very promising results when tested on text games. It's particularly interesting that, in contrast to other similar works, the previously described agents are able to learn policies that can be generalised to unseen games.

However, it's clear that there is still lots of work to be done in this field. Although the results seem to be very good, it's important to notice that the used game environment was very simple, so we still don't know if this approach is useful in more complex tasks.

Regardless of whether the described models are useful in real life problems or not, we think that the reinforcement learning perspective is vitally important for decision-making agents. Because of that, we strongly believe that future studies will continue creating new reinforcement techniques for this type of agents, and hopefully, they will be able to solve much more difficult tasks than simple text games.

References

[1] Xingdi Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, Adam Trischler
Counting to Explore and Generalize in Text-based Games.
Available in this link.

[2] FloydHub
An introduction to Q-Learning: Reinforcement Learning
Available in this link.

[3] Analytics Vidhya
A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python
Available in this link.

[4] Mohammad Ashraf
Reinforcement Learning Demystified: Markov Decision Processes.
Available in this link.

[5] Chris Nicholson
A Beginner's Guide to Multilayer Perceptrons (MLP)
Available in this link.

[6] Christopher Olah
Understanding LSTM Networks
Available in this link.

[7] Prithviraj Ammanabrolu, Mark O. Riedl
Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning.
Available in this link.

[8] Annual Reviews
Reinforcement Learning and Episodic Memory in Humans and Animals: An Integrative Framework.
Available in this link.

[9] Xingdi (Eric) Yuan
TextWorld-Coin-Collector
Available in this link.