University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Applied Mathematics

Simulation of a self-parking car using deep reinforcement learning

André Shumhei Kato

Final Essay

map 2010 — Capstone Project

Supervisor: Prof. Dr. Alexandre Megiorin Roma

São Paulo
2022

Esta seção é opcional e fica numa página separada;
ela pode ser usada para uma dedicatória ou epígrafe.

Agradecimentos

Study hard what interests you the most in the most
undisciplined, irreverent and original manner possible.
— Richard Feynman

Resumo

André Shumhei Kato. **Simulation of a self-parking car using deep reinforcement learning**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2022.

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave). Deve ser precedido da referência do documento. Texto texto. Texto texto.

Palavras-chave: Palavra-chave1. Palavra-chave2. Palavra-chave3.

Abstract

André Shumhei Kato. **Simulation of a self-parking car using deep reinforcement learning**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2022.

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa. De acordo com o Regimento da Pós-Graduação da USP (Artigo 99), deve ser redigido em inglês para fins de divulgação. É uma boa ideia usar o sítio www.grammarly.com na preparação de textos em inglês. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text. Text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text.

Keywords: Keyword1. Keyword2. Keyword3.

Lista de Abreviaturas

ML	Machine Learning
RL	Reinforcement Learning
MDP	Markov Decision Process
MRP	Markov Reward Process
PPO	Proximal Policy Optimization
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Contents

Appendixes

Annexes

Chapter 1

Preliminaries

1.1 Introduction

Reinforcement Learning is considered a subfield of Machine Learning, where learning occurs through an agent interacting with an environment. At each time step, the agent performs an action and the environment responds by producing a reward signal and transitioning to the next state. The goal of the agent is to maximize the total expected reward. Sutton and Barto, 2015

There are a lot of challenges that naturally arise from reinforcement learning problems that differ from the ones faced in classical machine learning. For example, balancing immediate rewards and future rewards: up to which point is it worth to sacrifice early rewards in exchange for bigger rewards in the future? An agent might be inclined to take actions that have been taken before because it has the knowledge of how much reward those actions will yield. Thus, limiting the agent to that specific set of actions and ultimately impairing it from further exploring the environment and possibly discovering new states and actions that could yield even more rewards. This is called the exploration-explotation dillema. Sutton and Barto, 2015

The focus of this work will be an application of the Proximal Policy Optimization (PPO) algorithm to train an agent able to park a car in a designated spot.

1.2 Motivation

With car crashes being more and more common, car manufacturers have started working on technologies that help mitigate crashes, ranging from simple proximity sensors that alarms the driver if a collision is imminent or stops the car by itself to fully fledged auto-driving systems. In the latter, automated parking is a key part in autonomous vehicle systems that allows cars to navigate through a parking lot completely unassisted.

This work aims to recreate the self-parking system inside a 3D virtual environment using deep reinforcement learning and studying how the algorithm performs in different parking situations.

Chapter 2

Introduction

2.1 Reinforcement Learning

Reinforcement learning is learning what to do - mapping situations to actions in order to maximize the reward received. The learner (or "agent") has no knowledge on what actions should be taken, as in many forms of machine learning. Instead, it must discover which ones yield the most reward by trying them and observing what happens. When taking an action, that action may affect not only the immediate reward but also the next reward and all the subsequent others. These two characteristics - trial-and-error search and immediate vs. delayed rewards are the most distinguishing aspects of a reinforcement learning problem.

The following examples illustrate how reinforcement learning concepts are applied in real life:

1. A chess player making a move. The move is informed both by planning - anticipating possible replies and counterreplies - and by intuititve judgements of what positions and moves are desirable.
2. A gazelle calf struggling to stand on its feet after being born and a few hours later being able to run.

Both examples involve an interaction between a decision-making agent and the environment, in which the agent seeks to achieve a goal, despite uncertainty about its environment. The agent's action may affect the future state of the environment, for example, the next chess move will affect the possible options and opportunities in the future. Taking the correct choice requires taking into account indirect and delayed consequences of actions, and thus requires planning.

Both examples have goals in which the agent can judge the progress towards it based on what it can sense directly. For example, the chess player could judge his progress by comparing his remaining pieces with the opponent's. The player also knows whether he wins.

In order to fully formulate a reinforcement learning problem, we need optimal control of a Markov Decision Process, which will be discussed later, but the basic elements are shown in the next section.

2.2 Elements of a Reinforcement Learning Problem

Apart from the agent and environment, there are other subelements of a reinforcement learning system: a *policy*, a *reward signal*, a *value function* and, optionally, a *model of the environment.*

A *policy* is a mapping from perceived states to actions to be taken when in those states, that is, a policy is what defines the agent's behavior. Policies can be deterministic, being a simple function or a lookup table, or stochastic, with probabilities associated with each action.

A *reward signal* is what defines the goal in a reinforcement learning problem. At each time step, the environment sends a reward signal, which is nothing but a number. The agent's goal is to maximize reward received over the long run. In biological systems, rewards are analogous to feeling pleasure or pain. The reward received depends on the agent's current action and on the state of the environment. The agent cannot alter this process in any way, but it can influence it through its actions. The reward signal is the primary basis for altering a policy. If an action selected by the policy yields a low reward, then the policy may be changed to select another action in that situation in the future.

A *value function* tells us how much reward the agent should expect to receive over the future by starting in a specific state. While reward signals indicates whether a state is immediately desirable or not, the value function estimates the long-term desirability of that state by taking into account the states that are likely to follow the current one. For example, a low reward state might be followed by a high reward state or vice versa. It's values we are most concerned with when making and evaluating decisions - we seek for actions that yield maximum value, not reward. Note that this is equivalent to maximizing reward over the long run. The major problem with value is that it can be hard to estimate - rewards are given directly by the environment, but values must be re-estimated at each iteration from sequences of observations an agent makes over its lifetime. The most important component of almost all reinforcement learning algorithms is value function estimation.

A *model of the environment* is something that mimics the behavior of the environment itself. More generally, that allows for inferences about how the environment will respond to certain actions. For example, for a given pair of state and action, the model might predict how the environment will respond to that action in that particular state. Models are particularly useful for *planning*, which is deciding on a course of action by considering possible future situations. Methods that use models and planning are called *model-based* methods, as opposed to *model-free* methods that rely on trial-and-error to learn about the environment.

2.3 Agent-Environment Interface

The learner and decision-maker agent and the thing it interacts with, comprising of everything outside the *agent*, is defined as the *environment*. The agent and the environment interact continually, with the agent choosing actions and the environment responding to those actions, presenting new situations to the agent and rewarding it.

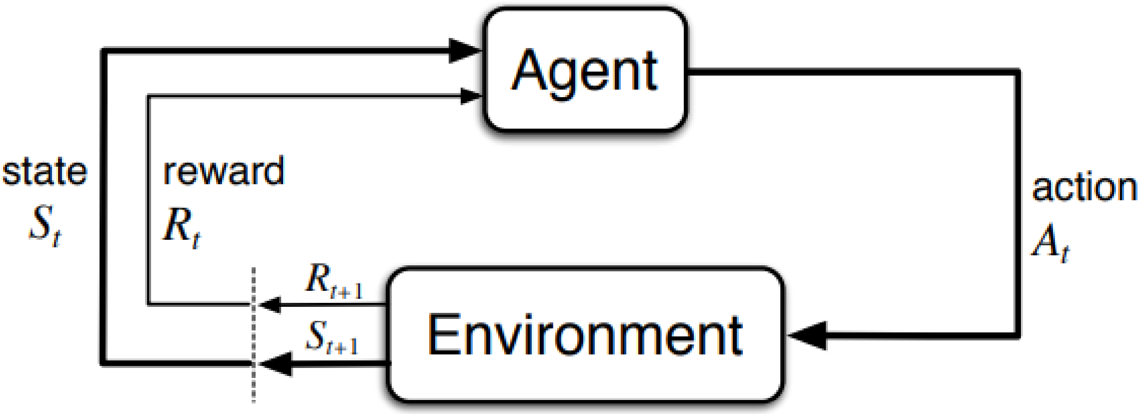

Figure 2.1: *Representation of the agent-environment interface from* Sutton *and* Barto*, 2015.*

In other words, at each time step $t = 0, 1, 2, \dots$, the agent receives some representation of the environment's state $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states, and selects an action $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}$ is the set of all possible actions and $\mathcal{A}(S_t)$ the set of all possible actions in state S_t. At the next time step $t + 1$, the agent receives a reward $R_{t+1} \in \mathbb{R}$ and a new state S_{t+1}.

The action the agent takes is sampled from the agent's policy, denoted π, with $\pi(a \mid s)$ being the probability of taking action a when in state s. π is just an ordinary function defining a probability distribution over $a \in \mathcal{A}(s)$. Reinforcement learning methods specify how the policy is changed as the agent gathers more experience.

2.4 Goals and Rewards

The goal of an agent is formalized in terms of a reward signal $R_t \in \mathbb{R}$ that is passed to the agent by the environment. The agent's goal is to maximize the total amount of rewards it receives, which means maximizing the cumulative reward. Consider the sequence of rewards received after time step t: $R_{t+1}, R_{t+2}, \dots$. What we seek to maximize is the *expected return* G_t, which is a function of the sequence of rewards. The simplest case is the sum of all of them:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

where T denotes the final time step. The notion of a final time step comes naturally when the agent-environment interaction can be breaked down into subsequences, which we call *episodes* or *trials*. For example, plays of a game, trips through a maze or any sort of repeated interactions can be considered episodes. Each episode ends in a special state called *terminal state*, followed by a reset to a standard state or to a sample of a distribution of starting states.

However, not all agent-environment interactions can be breaked naturally into episodes, instead, it could go on without limit. That is, with abuse of notation, $T = \infty$ and the return G could easily be infinite as well. Thus, the concept of *discounting* arises, where we introduce a *discounting factor* γ, $0 \leq \gamma \geq 0$, to weigh immediate and future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Then, the sum is finite as long as $\gamma < 1$ and the reward sequence is bounded. Also note that if $\gamma = 0$, the agent is only concerned with immediate rewards and the action A_t will be chosen in such a way to maximize only R_{t+1}. If γ is close to 1, then the future rewards will be taken into account more strongly.

According to Sutton and Barto, 2015, although formulating goals in terms of reward signals appears to be limiting, in practice, it has proved to be flexible and widely applicable. For example, to make a robot learn how to escape from a maze, the reward is often -1 for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible.

While designing how rewards should be given to the agent, it's crucial that it's done in such a way maximizing also makes the agent achieve the goal. That is, the reward signal must not be used to impart any prior knowledge to the agent about how to achieve the goal. For example, in a chess game, rewards should be given when the agent wins the game, and not when accomplishing subgoals, such as taking enemy pieces. If the agent gets rewarded for achieving subgoals, it might find a way to maximize the rewards by only taking enemy pieces and without actually winning. All in all, the reward signal is our way of communicating to the agent what needs to be done, not how to do it.

Chapter 3

Mark

Appendix A

Código-fonte e pseudocódigo

Com a *package* listings, programas podem ser inseridos diretamente no arquivo, como feito no caso do Programa **??**, ou importados de um arquivo externo com o comando \lstinputlisting, como no caso do Programa A.1.

Program A.1 Máximo divisor comum (arquivo importado).

```
FUNCTION euclid($a$, $b$)  ▷ The g.c.d. of a and b
    $r \leftarrow a \bmod b$
    while $r \neq 0$  ▷ We have the answer if r is 0
        $a \leftarrow b$
        $b \leftarrow r$
        $r \leftarrow a \bmod b$
    end
    return $b$  ▷ The g.c.d. is b
end
```

Trechos de código curtos (menores que uma página) podem ou não ser incluídos como *floats*; trechos longos necessariamente incluem quebras de página e, portanto, não podem ser *floats*. Com *floats*, a legenda e as linhas separadoras são colocadas pelo comando \begin{program}; sem eles, utilize o ambiente programruledcaption (atenção para a colocação do comando \label{}, dentro da legenda), como no Programa A.2[1]:

Program A.2 Máximo divisor comum (em português).

```
FUNCAO euclides($a$, $b$)  ▷ O máximo divisor comum de a e b
    $r \leftarrow a \bmod b$
enquanto $r \neq 0$  ▷ Atingimos a resposta se r é zero
        $a \leftarrow b$
        $b \leftarrow r$
```

cont ⟶

[1] listings oferece alguns recursos próprios para a definição de *floats* e legendas, mas neste modelo não os utilizamos.

```
    ⟶ cont
            $r \leftarrow a$ mod $b$
        fim
    devolva b  ▷ O máximo divisor comum é b
    fim
```

Além do suporte às várias linguagens incluídas em listings, este modelo traz uma extensão para permitir o uso de pseudocódigo, útil para a descrição de algoritmos em alto nível. Ela oferece diversos recursos:

- Comentários seguem o padrão de C++ (`//` e `/* ... */`), mas o delimitador é impresso como "▷".
- ":=", "<>", "<=", ">=" e "!=" são substituídos pelo símbolo matemático adequado.
- É possível acrescentar palavras-chave além de "if", "and" etc. com a opção "morekeywords={pchave1,pchave2}" (para um trecho de código específico) ou com o comando \lstset{morekeywords={pchave1,pchave2}} (como comando de configuração geral).
- É possível usar pequenos trechos de código, como nomes de variáveis, dentro de um parágrafo normal com \lstinline{blah}.
- "$...$" ativa o modo matemático em qualquer lugar.
- Outros comandos LaTeX funcionam apenas em comentários; fora, a linguagem simula alguns pré-definidos (\textit{}, \texttt{} etc.).
- O comando \label também funciona em comentários; a referência correspondente (\ref) indica o número da linha de código. Se quiser usá-lo numa linha sem comentários, use `///` \label{blah}; "`///`" funciona como `//`, permitindo a inserção de comandos LaTeX, mas não imprime o delimitador (▷).
- Para suspender a formatação automática, use \noparse{blah}.
- Para forçar a formatação de um texto como função, identificador, palavra-chave ou comentário, use \func{blah}, \id{blah}, \kw{blah} ou \comment{blah}.
- Palavras-chave dentro de comentários não são formatadas automaticamente; se necessário, use \func{}, \id{} etc. ou comandos LaTeX padrão.
- As palavras "Program", "Procedure" e "Function" têm formatação especial e fazem a palavra seguinte ser formatada como função. Funções em outros lugares *não* são detectadas automaticamente; use \func{}, a opção "functions={func1,func2}" ou o comando "\lstset{functions={func1,func2}}" para que elas sejam detectadas.
- Além de funções, palavras-chave, strings, comentários e identificadores, há "specialidentifiers". Você pode usá-los com \specialid{blah}, com a opção "specialidentifiers={id1,id2}" ou com o comando "\lstset{specialidentifiers={id1,id2}}".

Annex A

Perguntas frequentes sobre o modelo[2]

- **Não consigo decorar tantos comandos!**
  Use a colinha que é distribuída juntamente com este modelo (gitlab.com/ccsl-usp/modelo-latex/raw/master/pre-compilados/colinha.pdf?inline=false).

- **Por que tantos arquivos?**
  O preâmbulo LaTeX deste modelo é muito longo; as partes que normalmente não precisam ser modificadas foram colocadas no diretório `extras`, juntamente com alguns arquivos acessórios.

- **Estou tendo problemas com caracteres acentuados.**
  Versões modernas de LaTeX usam UTF-8, mas arquivos antigos podem usar outras codificações (como ISO-8859-1, também conhecido como latin1 ou Windows-1252). Nesses casos, use \usepackage[latin1]{inputenc} no preâmbulo do documento. Você também pode representar os caracteres acentuados usando comandos LaTeX: \'a para á, \c{c} para cedilha etc., independentemente da codificação usada no texto[3].

- **Aparece uma folha em branco entre os capítulos.**
  Essa característica foi colocada propositalmente, dado que todo capítulo deve (ou deveria) começar em uma página de numeração ímpar (lado direito do documento). Se quiser mudar esse comportamento, acrescente "openany" como opção da classe, i.e., \documentclass[openany,...]{book}.

- **É possível resumir o nome das seções/capítulos que aparece no topo das páginas e no sumário?**
  Sim, usando a sintaxe \section[mini-titulo]{titulo enorme}. Isso é especialmente útil nas legendas (*captions*) das figuras e tabelas, que muitas vezes são demasiadamente longas para a lista de figuras/tabelas.

[2] Esta seção não é de fato um anexo, mas sim um apêndice; ela foi definida desta forma apenas para servir como exemplo de anexo.

[3] Você pode consultar os comandos desse tipo mais comuns em en.wikibooks.org/wiki/LaTeX/Special_Characters. Observe que a dica sobre o pingo do i *não* é mais válida atualmente; basta usar \'i.

- **Existe algum programa para gerenciar referências em formato bibtex?**
  Sim, há vários. Uma opção bem comum é o JabRef; outra é usar Zotero ou Mendeley e exportar os dados deles no formato .bib.
- **Posso usar pacotes LaTeX adicionais aos sugeridos?**
  Com certeza! Você pode modificar os arquivos o quanto desejar, o modelo serve só como uma ajuda inicial para o seu trabalho.
- **Como faço para usar o Makefile (comando make) no Windows?**
  Lembre-se que a ferramenta recomendada para compilação do documento é o latexmk, então você não precisa do make. Mas, se quiser usá-lo, você pode instalar o MSYS2 (www.msys2.org) ou o Windows Subsystem for Linux (procure as versões de Linux disponíveis na Microsoft Store). Se você pretende usar algum dos editores sugeridos, é possível deixar a compilação a cargo deles, também dispensando o make.

References

[Sutton and Barto 2015] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 2015 (cit. on pp. 1, 5, 6).

Index