INDIAN INSTITUTE OF TECHNOLOGY, INDORE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNDERGRADUATE THESIS

P.E.A.R.L.

PLATFORM TO EXPLORE ALGORITHMS IN REINFORCEMENT LEARNING

---

by

DHRUV CHADHA AND SHASHWAT RAGHUVANSHI

DECEMBER 3, 2018

INDIAN INSTITUTE OF TECHNOLOGY, INDORE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNDERGRADUATE THESIS

P.E.A.R.L.

PLATFORM TO EXPLORE ALGORITHMS IN REINFORCEMENT LEARNING

Authors:
DHRUV CHADHA AND SHASHWAT RAGHUVANSHI

Supervisors:
DR. ARUNA TIWARI AND DR. KAPIL AHUJA

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Technology
in
Computer Science and Engineering

DECEMBER 3, 2018

Declaration of Authorship

We declare that this thesis and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a undergraduate degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by us jointly with others, we have made clear exactly what was done by others and what our contribution was.

Signatures (with Date):

Dhruv Chadha *Shashwat Raghuvanshi*

Declaration by BTP Guides

It is certified that the declaration of authorship made by the students is correct to the best of our knowledge.

Signatures (with Date):

Dr. Aruna Tiwari	*Dr. Kapil Ahuja*

"Don't watch the clock; do what it does. Keep going."

Sam Levenson

INDIAN INSTITUTE OF TECHNOLOGY, INDORE

Abstract

P.E.A.R.L.

Research in Reinforcement Learning has gained a lot of momentum, especially with the big firms investing tons of money in developing this field. There has been a scarcity of resources online that help to develop an intuitive understanding of the basic concepts involved. We present P.E.A.R.L., an online Do-It-Yourself platform for understanding the fundamental algorithms and concepts involved in Reinforcement Learning. It is designed to teach the fundamentals of the field in a very intuitive and visual manner, using real world examples and illustrations, and give hands-on experience of the algorithms involved. We have developed slideshows for every chapter, so that the learning can take place at user's comfortable pace. Along with chapters, we also provide them with code templates, which the user needs to complete as assignments. The platform is open-sourced, and is hosted online using GitHub Pages. This platform aims to spark interest in reinforcement learning in the mind of more students, be community driven to improve the content quality and coverage, and boost the research in the field.

Acknowledgements

We would like to thank our B.Tech. Project supervisors Dr. Aruna Tiwari and Dr. Kapil Ahuja for their constant guidance and support in structuring the project and their valuable feedback throughout the course of this project.

We are grateful to Mr. Rohit Agrawal who provided valuable guidance in shaping the project.

We are really grateful to the Institute for the opportunity to be exposed to systemic research especially Dr. Aruna Tiwari's Lab, who encouraged us to pursue this unconventional project.

Lastly, we offer our sincere thanks to everyone who helped us complete this project including our friends, who were a constant source of motivation and technical support, and some juniors, who helped us design the platform and various illustrations.

Contents

List of Figures

List of Abbreviations

ML	**M**achine **L**earning
RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
DP	**D**ynamic **P**rogramming
MC	**M**onte **C**arlo
TD	**T**emporal **D**ifference

Chapter 1

Introduction

1.1 Background

Reinforcement Learning (RL) is one of the major fields of machine learning, along with Supervised Learning, Unsupervised Learning etc.

RL is concerned with finding an optimal decision-making strategy, such that finally it achieves the required goal in a given environment. Human Intelligence has largely evolved through our continuous interaction and understanding of the environment. Our actions have consequences, both good and bad, and over time, with our continual interaction, our experiences form the basis of our understanding of what is desirable and what is potentially harmful.

RL aims to mimic this learner-environment interaction model, and solve a variety of difficult problems.

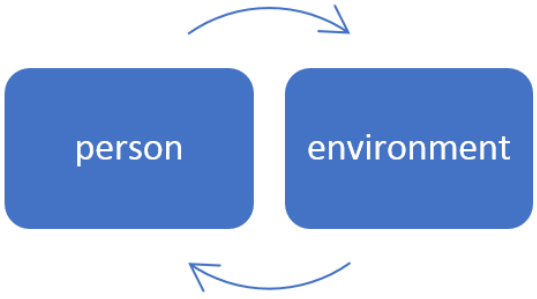

FIGURE 1.1: Human-Environment Interaction

Today, Reinforcement Learning finds applications in robotics, resources management, finance, autonomous vehicles, games etc. Recently, Google DeepMind's AlphaGo program (Silver et al., 2016) defeated the world champion in an extremely complex board game 'Go'. The program was based majorly on Deep Reinforcement

Learning (Mnih et al., 2013), a novel approach that combines Reinforcement Learning and Convolutional Neural Networks. Another huge achievement was in the area of resource management. DeepMind, which is really a front runner in RL research, applied its algorithms, that brought down Google's data center cooling cost by 40%. In the long term, there is a potential to apply this technology in other industrial settings, and help tackle climate change.

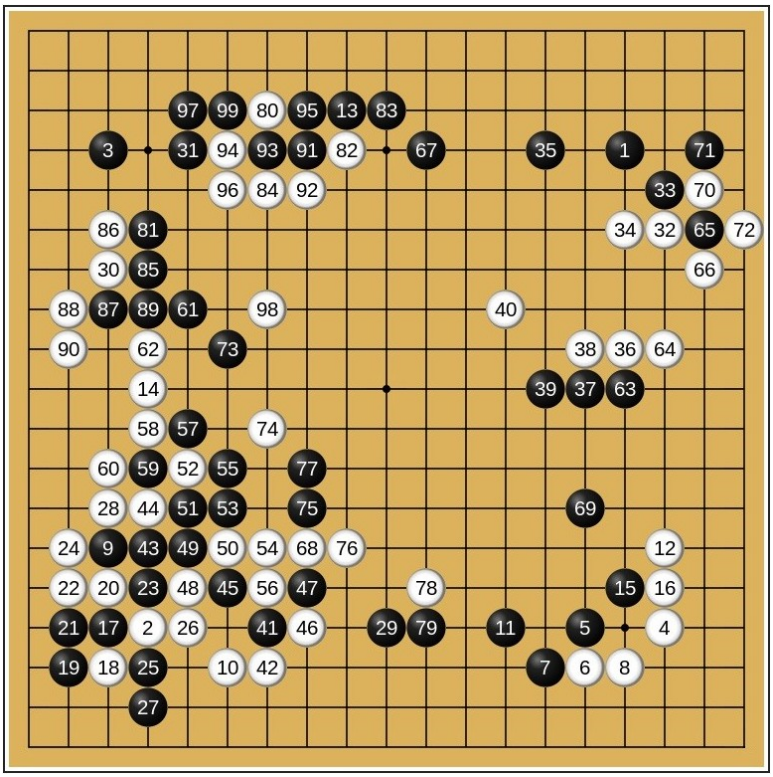

FIGURE 1.2: First 99 moves, 2nd match between AlphaGo and Lee Sedol (Image source - theDiagonal)

1.2 Related Work

With applications of such large impact, RL needs to be given much more focus than already is. There has been a lot of efforts recently online, that aim to improve understanding of the field.

OpenAI Gym (Brockman et al., 2016), which is an open-source library, is a toolkit for developing and comparing reinforcement learning algorithms. It has a growing collection of RL problems, that people can solve, improve and compare results upon.

DeepMind Lab (Beattie et al., 2016) is a first-person 3D game platform designed for research and development of general artificial intelligence and machine learning systems.

UC Berkeley's *The Pac-Man Projects* aims at introducing starters to a variety of algorithms to play Pacman, which include Machine Learning, Graph Theory, Game Theory, Probabilistic Inference etc. It has video lectures from their lab sessions, along with code templates that the Learner has to fill and execute.

Top MOOCs like *Udacity Deep Reinforcement Learning Nanodegree* Program is also an extensive course that has video lectures and uses OpenAI's Gym environment for practical purposes.

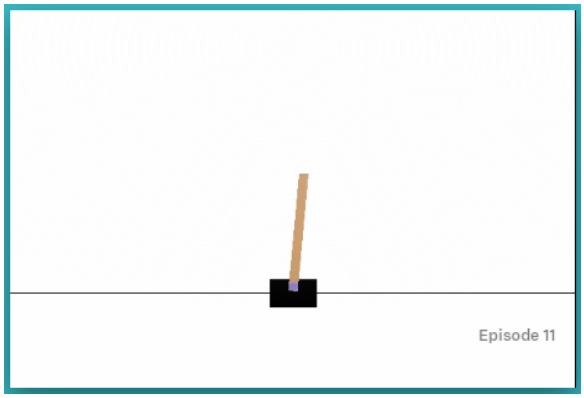

FIGURE 1.3: Classic CartPole problem in OpenAI Gym
Image source - OpenAI Gym

1.3 Motivation

The motivation to develop a new platform arises because of the obstacles that a new learner faces on the platforms mentioned above.

OpenAI Gym and DeepMind Lab aim to provide standardization of environments used in publications, and also provide better benchmarks. There is not study material to get beginners started.

The Pac-Man Projects introduces RL on a very basic level, and doesn't cover the many of the concepts that build up to the modern popular RL algorithms.

Online Nanodegree Programs are usually very expensive, and are hence inaccessible to many students. Udacity's Nanodegree Program on Deep RL is extremely expensive, costing approximately Rs. 50000.

The P.E.A.R.L. platform we have created aims to overcome the above shortcomings in the following manner-

1. It would provide coding assignments which would help the users visualize the core concepts of Reinforcement Learning instead of an application, like Pacman, or CartPole.

2. It would also provide a theoretical background so that the users have the necessary understanding to implement the algorithms.

3. It would be completely open-source, so it will be accessible to everyone on the internet, without any charges.

Chapter 2

Literature Survey

In this chapter, we will explore the theoretical basis of our learning platform.

First we will introduce RL as a subset of AI, and compare it with supervised and unsupervised learning. Later, we will provide the theory of RL that we have covered through our learning platform. It will begin with the definition of the elements of RL. Then we will discuss the mathematical formulations in RL, and finally, we will briefly discuss the core algorithms to solve RL problems.

2.1 Basics

Reinforcement Learning addresses the question of how an autonomous agent (one that learns) that senses its current environment, can learn to choose and perform optimal actions to achieve its goals.

It does so by modelling the learning task in the form of a Markov Decision Process (MDP).

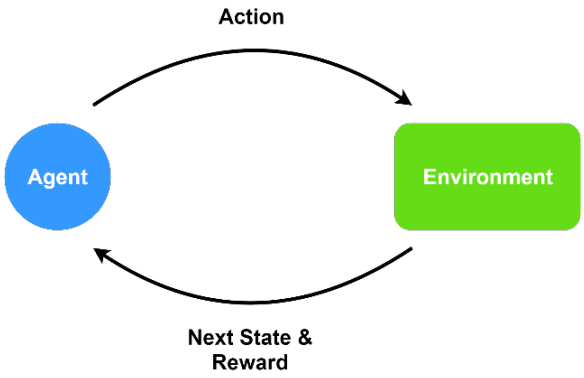

FIGURE 2.1: Agent-Environment interaction in a MDP

For every action the agent takes, there is a corresponding change in the environments, which is either desirable or undesirable. On repeated interaction, the agent gradually learns the behavior of the environment, and consequently adjusts and optimizes its own actions so as to maximise the desirability and achieve the goal.

The agent's interaction/experience in broken down into a series of episodes. In each episode, the agent starts from an initial state, and the interaction proceeds until the environment reaches a terminal state.

2.2 Comparison with other paradigms of ML

2.2.1 Reinforcement Learning vs. Supervised Learning

Supervised Learning involves learning using training set of correct behaviour examples.
However, Reinforcement Learning involves learning using practical experience and feedback from environment.

2.2.2 Reinforcement Learning vs. Unsupervised Learning

Unsupervised Learning aims at finding structure in the data and extracting useful features.
However, Reinforcement Learning aims at finding an optimal sequence of decisions to reach a desired goal.

2.3 Elements of Reinforcement Learning

2.3.1 Agent

Agent is the learning entity that seeks to achieve a goal by performing actions in its environment and observing their consequences.

2.3.2 Environment

It is place where the agent performs actions. The agent's actions influence the environment's properties. Generally speaking, anything that cannot be changed arbitrarily by the agent is considered to be outside it, and thus part of its environment.

2.3.3 State

State is a signal conveying to the agent some sense of "how the environment is" at a particular time. It is used to distinguish between two 'configurations' of the environment. Each state also has associated set of actions possible in that state.

2.3.4 Reward

Whenever the agent interacts with the environment and changes its state, the agent receives a feedback signal, called a reinforcement or a 'reward'. When the reward is positive, it is an achievement to be in that state, and when it is negative, it is a punishment to be in that state.

2.3.5 Policy

Policy is a mapping from states to probabilities of selecting each possible action. It can be both deterministic and stochastic. It determines what action the agent will take when he is present at a particular state. Reinforcement learning methods specify how the agent's policy is changed as a result of its experience.

2.3.6 Environment Model

A model mimics the behavior of the environment and allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward.

2.4 Mathematical Formulations

We will now discuss the formalization of a decision-making problem in the form of a Markov Decision Process, and the functions/quantities that need to be defined in order to solve the problem.

2.4.1 Markov Decision Process

A Markov Decision Process (MDP) is a classical formalization of sequential decision making where actions have long term consequences. The MDP framework proposes that any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment: states, actions and rewards.

Mathematically, an MDP comprises of 4 sets -

1. States (S)
2. Actions (A_s)
3. Transition Probability ($P^a_{ss'}$)
4. Transition Reward ($R^a_{ss'}$)

2.4.2 Transition Graph

The dynamics of a finite MDP can be summarized using a transition graph. There is a state node for each possible state and an action node for each state–action pair. Starting in state 's' and taking action 'a' moves you along the line from state node 's' to action node (s, a). Then the environment responds with a transition to the next state's node via one of the arrows leaving action node (s, a).

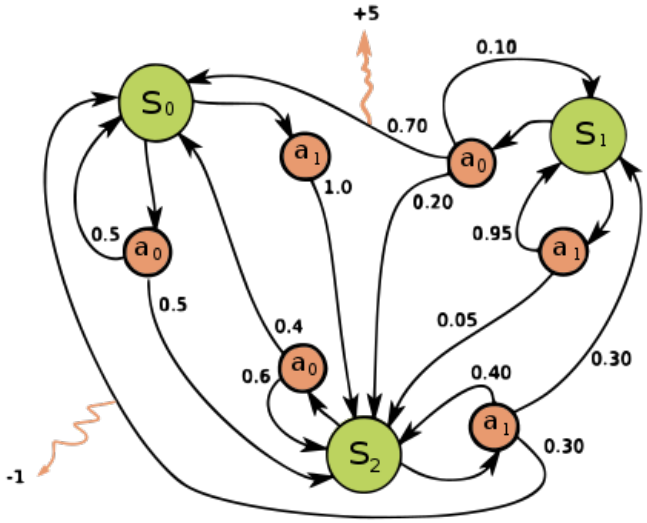

FIGURE 2.2: A transition graph representing an MDP
(Image by waldoalvarez / CC BY-SA 4.0

2.4.3 Return

The aim of a Reinforcement Learning agent is to select actions so as to maximize the cumulative discounted rewards it receives over the long run. Mathematically, this quantity is called Return(G).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2.1}$$

where $0 \leq \gamma \leq 1$ is called the discount rate.

The infinite sum has a finite value if $\gamma < 1$ and the reward sequence $\{R_k\}$ is bounded. The discount rate determines the level of farsightedness of the agent. If $\gamma = 0$, the agent is "myopic" in being concerned only with maximizing immediate rewards. As γ approaches 1, the agent becomes more farsighted.

2.4.4 Value Function

Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Unlike rewards, which are basically given directly by the environment, values must be estimated and re-estimated from the observations

an agent makes over its entire lifetime. Both state nodes and action nodes have value functions, namely state value function and action value function.

The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. For MDPs, we can define v_π formally as -

$$v_\pi(s) = E_\pi[G_t|S_t = s] \tag{2.2}$$

Similarly, we define the value of taking action s in state s under a policy π, denoted $q_\pi(s, a)$, as the expected return starting from s, taking the action a, and thereafter following policy π -

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{2.3}$$

2.5 Solving an MDP

Value functions help in evaluating the desirability of the states/actions. Once we have these values, it is very simple to find an optimal policy. We will now look at the equations that help us calculate these values.

2.5.1 Bellman Equations

They express a relationship between the value of a node (state/state-action) and the values of its successor nodes.

Bellman Equation for v_π

$$\begin{aligned} v_\pi(s) &= E_\pi[G_t|S_t = s] && (2.4)\\ &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] && (2.5)\\ &= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1}|S_{t+1} = s']|S_t = s] && (2.6)\\ &= E_\pi[R_{t+1} + \gamma v_\pi(s')|S_t = s] && (2.7) \end{aligned}$$

Bellman Equation for q_π

$$
\begin{aligned}
q_\pi(s,a) &= E_\pi[G_t|S_t = s, A_t = a] && (2.8)\\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] && (2.9)\\
&= E_\pi[R_{t+1} + \gamma E_\pi[G_{t+1}|S_{t+1} = s', A_{t+1} = a']|S_t = s, A_t = a] && (2.10)\\
&= E_\pi[R_{t+1} + \gamma q_\pi(s', a')|S_t = s, A_t = a] && (2.11)
\end{aligned}
$$

2.5.2 Optimal Policy

A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$. In an MDP, there is always at least one policy which is better than or equal to all other policies. This policy is called the optimal policy. There can be multiple optimal policies for the same MDP.

2.5.3 Optimal Value Functions

Optimal State-Value Function

Optimal policies share the same State-Value function, called the Optimal State-Value function, denoted by v_*, and defined as $v_*(s) = \max_\pi v_\pi(s)$ for all $s \in S$. This function assign to each state, the largest expected return achievable by any policy, starting from that state.

Optimal Action-Value Function

Optimal policies also share the same Action-Value function, called the Optimal Action-Value Function, denoted by q_*, and defined as $q_*(s,a) = \max_\pi q_\pi(s,a)$ for all $s \in S$ and $a \in A(s)$. Like the Optimal State-Value Function, the Optimal Action-Value Function assigns to each state–action pair, the largest expected return achievable by any policy, after taking the particular action from the particular state.

2.5.4 Bellman Optimality Equations

They express the relationship between the Optimal State-Value Function and the Optimal Action-Value Function.

Bellman Optimality Equation for v_*

$$\begin{aligned} v_*(s) &= \max_a q_*(s,a) && (2.12)\\ &= \max_a E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] && (2.13) \end{aligned}$$

Bellman Optimality Equation for q_*

$$\begin{aligned} q_*(s,a) &= E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] && (2.14)\\ &= \max_a E[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] && (2.15) \end{aligned}$$

2.5.5 Solving the Bellman Optimality Equations

Each Bellman Optimality Equation is actually a system of non-linear equations. If the MDP is finite, we can find a unique solution of the equations, which is independent of any policy.

There exists a Bellman Optimality Equation for each state, so if there are n states, then there are n equations in n unknowns. If the dynamics p of the environment are known, then in principle one can solve this system of equations for v^* using any one of a variety of methods for solving systems of nonlinear equations.

Once one has v^*/q^*, it is relatively easy to determine an optimal policy. For each state s, there will be one or more actions at which the maximum is obtained in the Bellman Optimality Equations. Any policy that assigns nonzero probability only to these actions is an optimal policy.

So, the simplest way to find v^*/q^* values for all states/state-action pairs is by solving the system of non-linear equations. However, this solution relies on at least three assumptions that are rarely true in practice:

1. We accurately know the dynamics of the environment

2. We have enough computational resources to complete the computation of the solution

3. The Markov property

Hence, we take the help of iterative methods like Dynamic Programming, Monte-Carlo Methods, Temporal Difference Learning etc. to approximate the solution of the system of equations.

The idea behind iterative methods is to start with an initial approximation to the solution (which does not have to be accurate at all), and to change this approximation in several steps till it converges to the true solution. Once the approximation is sufficiently accurate, it is taken to be the solution to the system.

2.6 Dynamic Programming

These classical DP algorithms assume that a perfect model of the environment as an MDP is given i.e. all states, action, probability values, rewards etc. are known, and are computationally expensive. But these algorithms provide the foundations for all the future algorithms.

2.6.1 Idea

DP algorithms are obtained by turning Bellman equations such as these into assignments, that is, into update rules for improving approximations of the desired value functions.

2.6.2 Efficiency of DP

A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is k^n. In this sense, DP is exponentially faster than any direct search in policy space could be.

2.7 Monte Carlo methods

Monte Carlo methods learn from experience and don't require complete knowledge of the environment's dynamics.

2.7.1 Idea

These methods only need sample sequences of S, A and R, gathered from repeated interaction with the environment. The value functions are evaluated from these sample sequences, using the Law Of Large Numbers.

2.8 Temporal Difference Learning

TD Learning combines the ideas of Monte Carlo methods with those of dynamic programming. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome.

2.8.1 Idea

Both TD and Monte Carlo methods use experience to solve the prediction problem. Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is G_t known completely), TD methods need to wait only until the next time step. At time $t + 1$, they immediately form an 'approximate target', and make a useful update.

2.9 Current Advancements

Q-Learning (Watkins and Dayan, 1992) was one of the major breakthroughs in the field of machine learning. This algorithm helps the agent learn to make optimal decisions in the environment. Over the years, Q-Learning has been further developed and extended to Double Q-Learning, Hierarchical Q-Learning etc., with one of the recent

major advancements being Deep Q-Learning. Since Deep Q-Learning, all the major Q-Learning improvements have also been applied to Deep Q-Learning, like Double Deep Q-Learning (Hasselt, Guez, and Silver, 2015) etc. A different field of research named Inverse Reinforcement Learning (Russell, 1998) also exists in which the optimal actions are already given by an expert, the agent needs to observe, and learn to replicate the results, just like an apprentice.

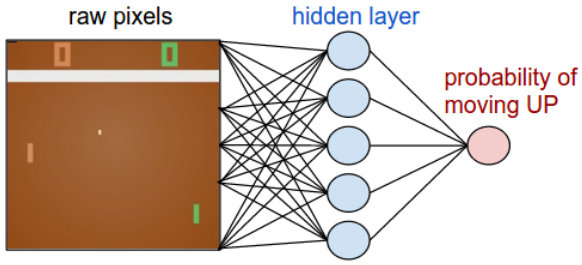

FIGURE 2.3: A simple visualisation of Deep Q-Learning
Image source - Andrej Karpathy blog

Chapter 3

Design Analysis and Content

In this chapter, we shall discuss the design decisions that we took for building the platform, and the content that we have developed, including the study material, coding assignments and the platform as a whole.

3.1 Study Material

The material is greatly influenced from Richard S. Sutton and Andrew G. Barto's book – Reinforcement Learning: An Introduction (Sutton and Barto, 2017).

We found that the content in the book was slightly difficult/complicated for a beginner to understand. So, we decided to develop our own material, using the book as a reference, to make the core ideas and intuitions presented in the book accessible to an even wider audience. But the topics are ordered majorly in accordance with the book.

3.2 Slideshows

Slideshows have helped us to provide a more visual and intuitive way of conveying the basic principles involved in RL.

Every slide has limited number of lines for explaining the concepts, and an elegant design, so the learner doesn't get overwhelmed with information and lose interest. The explanations are also accompanied with diagrams and illustrations time and again to make them easy to visualize.

Additionally, the transitions that are possible with slideshows, help to control where the user's eyes will focus at various moments throughout the presentation. This comes

in handy when large diagrams or concepts need to be broken down into various slides, for easy explanation.

3.3 Coding Assignments

Similar to UC Berkeley's strategy, we decided to accompany most chapters with relevant coding tasks to complete and execute.

We didn't want learners to struggle with writing unnecessary code like graphics and other helper functions. Their focus should be restricted to understanding and implementing the core algorithms. Hence, the assignments are python programs with proper graphics so that one can visually see the running code.

However, unlike The Pac-Man Projects, or OpenAI's gym environment, our coding assignments do not show real world examples, like a game of Pacman, or an agent trying to balance a pole on a cart. Instead, our visual output is more inclined towards showing the fundamental result of the algorithms, i.e. how an algorithm is actually working on an MDP, and finding out an optimal decision-making strategy using the algorithm.

3.4 Website

P.E.A.R.L. is an open-source learning platform, to encourage community contribution.

In case anyone finds any shortcomings or improper explanations, they can simply raise an issue on GitHub, where the platform has been hosted. They can also resolve issues and create pull requests. This way, the community will be able to strengthen the platform and create a near-perfect learning environment. This will help in maintaining quality content.

Chapter 4

Implementation

We will now discuss the stages of development of the slideshows, coding assignments and the website involved in the platform.

4.1 Evolution of slideshows

1. Understanding theory of RL in depth from the book “Reinforcement Learning: An Introduction”.
2. Developing and filtering content into word documents, to make it more intuitive and easier to understand.
3. Creation of slides from word documents and adding illustrations and making slides more elegant.

4.2 Evolution of coding assignments

1. Understanding useful libraries like matplotlib, numpy etc. in python.
2. Designing a code structure to represent an MDP and its various features and functionalities.
3. Rapid prototyping of a working code, with essential functionalities and graphics.
4. Optimising code and adding assignments.

4.3 Evolution of website

1. Studying the design of various similar websites like Coursera, Slideshare, Sway, OpenAI Gym, The Pac-Man Projects.

2. Developing a basic user interface, providing coding assignments as a download link and designing a separate page for the slideshows.

3. Adding final touches and hosting on GitHub.

Chapter 5

Conclusion and Future Work

Recent advances in the field of RL has led to a great deal of excitement in the area, and we wish to encourage more students to research in this field.

Hence, we have created a platform that focuses only on Reinforcement Learning. It provides the fundamentals, along with coding assignments for the users to apply the gained knowledge. It is completely open-source and free of cost, which enables collaboration with other users, who can point out and correct any mistakes at any time and make the platform more robust.

The course is completely self-paced, and structured in a manner so as to keep the learner engaged, along with lots of visual aids incorporated in the form of slideshows.

The open-source nature of the platform gives it the flexibility to be extended in the future in various ways by the online community and students -

1. Extension of the course – As research continues in the field and novel approaches are discovered, more chapters can be added to explain the latest advancements.

2. Improvement of the explanations – We want the explanations to be crystal clear. In case people find any ambiguities and problems in understanding any part of any chapter, it can be improved by further clarifying the explanations and adding real worlds examples.

3. Adding more visual aids – We hope to add more visual aids that help in better understanding the topics and algorithms, like drawings and animations.

4. Extension to other areas of Machine Learning – If successful, this endeavor can be extended to supervised learning, unsupervised learning and other areas of Machine Learning.

Appendix A

User Manual

A.1 Navigation

Our platform is hosted online on the url https://pearl-ai.github.io, and consists of a single page, with multiple secions.

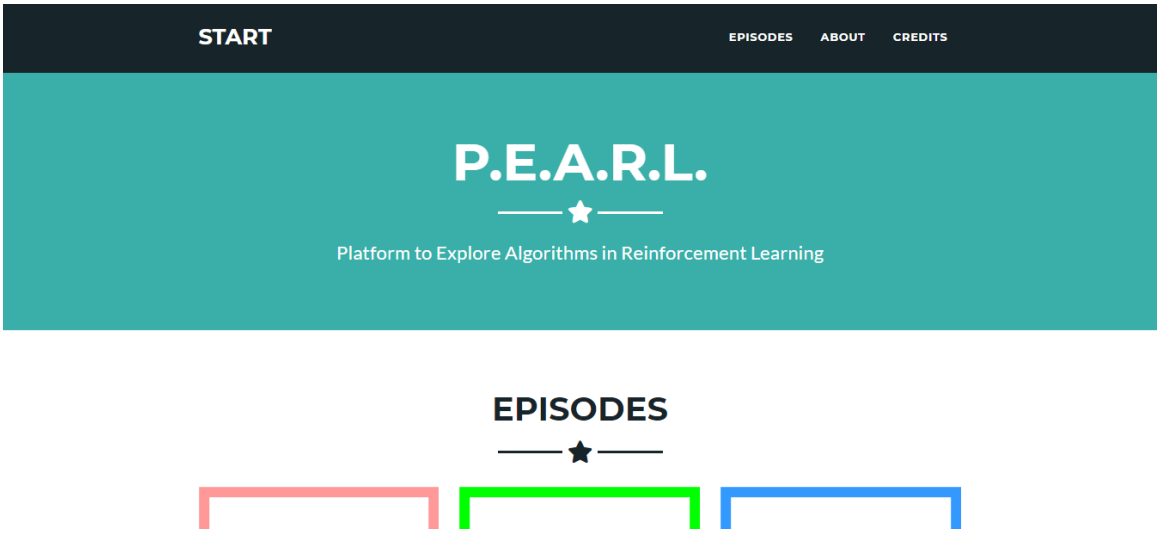

FIGURE A.1: P.E.A.R.L. Home Page

The chapters are present under the section named Episodes, which consists of all the chapters present on the website.

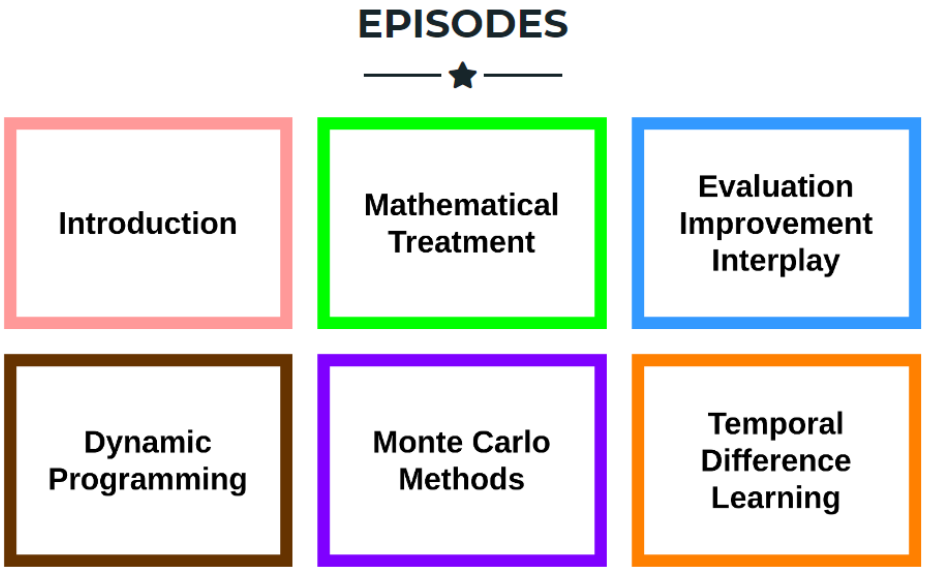

FIGURE A.2: Episodes section on the platform

A.2 Content

Clicking any chapter in this section would open its description. At the end of the description are links to the slides and coding assignment for the chapter.

A.2.1 Slides

On clicking the Slide button, a new web page opens up for viewing the presentation. The presentation can also be expanded to full screen mode for better experience.

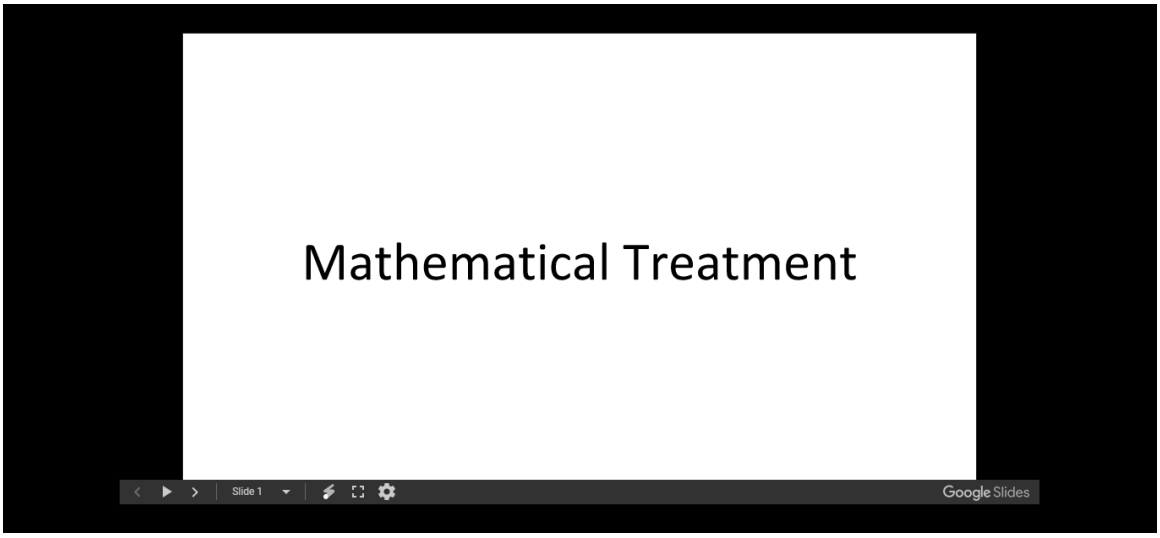

FIGURE A.3: Separate page for viewing presentation

A.2.2 Coding Assignments

On clicking the Code button, a zip file containing the coding assignment is downloaded on the user's machine.

The assignments are named in the following manner -

Assignment_<assignment-number>_<chapter-short-name>.py.

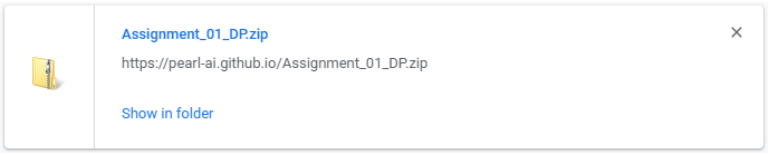

FIGURE A.4: Downloaded Dynamic Programming chapter assignment

Running the code requires Python 3, along with matplotlib and numpy libraries to complete the assignment. To install these libraries, the user can use the standard installation methods present online, namely the standard pip installation, or the anaconda installation.

Each assignment consists of empty functions that need to be filled by the user in order to run the code. This would open a window showing the output simulation of the algorithm that the user has written. Instructions to fill the empty code would be available in both the slides and the code to guide the user.

The code will be evaluated against the correct answer to check the accuracy and code execution time will be measured. Using all these factors, the performance of the user's code will be displayed in the form of a final score.

Appendix B

Pseudocodes

B.1 Dynamic Programming

We have included two important DP algorithms - Policy Iteration and Value Iteration. Both these algorithms involve interleaving of two important processes - Policy Evaluation and Policy Improvement. Let's look at the two processes before understanding the algorithms.

1. Iterative Policy Evaluation

   We start by assigning an arbitrary state-value to each of the states. Each successive approximation is obtained by using the Bellman equation for v_π as an update rule, until the value converges

   $$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(s')|S_t = s] \tag{B.1}$$

   for all states. This algorithm is called iterative policy evaluation. Formally, iterative policy evaluation converges only to the limit, but in practice it must be halted short of this.

2. Policy Improvement

   For policy improvement, we simply compute for every state, the action that has the maximum $q(s, a)$ value, and select that action for further computations.

B.1.1 Policy Iteration

Once a policy, π, has been improved using v_π to yield a better policy, π', we can then compute $v_{\pi'}$ and improve it again to yield an even better π''. We thus obtain a sequence of monotonically improving policies and value functions -

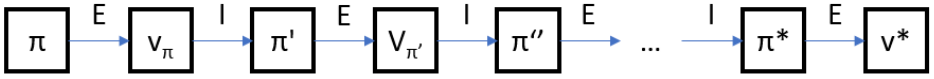

FIGURE B.1: Evaluation-Improvement cycle in Policy Iteration

This way of finding the optimal policy is called policy iteration. Note that each policy evaluation is itself an iterative computation.

Pseudocode

1. Initialization

   $V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$

2. Policy Evaluation

   Repeat:

   $\Delta = 0$

   For each $s \in S$:

   $v = V(s)$

   $V(s) = \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

   until $\Delta < \theta$ $(a\ small\ positive\ number)$

3. Policy Improvement

   $policy\_stable = true$

   For each $s \in S$:

   $old\_action = \pi(s)$

   $\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

   If $old\_action \neq \pi(s)$, then $policy\_stable = false$

   $\Delta = max(\Delta, |v - V(s)|)$

   If $policy\_stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2.

B.1.2 Value Iteration

The policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special case is when policy evaluation is stopped after just one sweep (one update of each state). This special case is called value iteration.

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement. It is obtained simply by turning the Bellman optimality equation into an update rule.

Pseudocode

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \, \epsilon \, S^+$)
Repeat:
$\Delta = 0$
For each $s \, \epsilon \, S$:
$v = V(s)$
$V(s) = max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
$\Delta = max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ $(a \; small \; positive \; number)$
Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

B.2 Monte Carlo methods

We cover two important MC algorithms - On-Policy MC Control and Off-Policy MC Control. Policy Evaluation and Policy Improvement are present in these as well, but use a slightly different approach due to non-availability of environment transition information.

1. Policy Evaluation using Monte Carlo approach

   In MC, the state values alone are not sufficient to determine the policy as we don't have a model of the environment like DP, and are unaware of the environment

probabilities. One must explicitly estimate the value of each action in order for the values to be useful in suggesting a policy. To achieve this, we consider the policy evaluation problem for action values.

Since the action value is nothing but the expected value of the return, we can apply the law of large numbers to find this value from experience. For each of the experienced episode (sequence of S, A and R), we can average over all the returns observed after visits to that action node.

A state–action pair (s, a) is said to be visited in an episode if ever the state s is visited and action a is taken in it.

This evaluation step converges to the true expected values as the number of visits to each state–action pair approaches infinity.

2. Policy Improvement

   For policy improvement, we again simply compute for every state, the action that has the maximum $q(s, a)$ value, and select that action for further computations.

B.2.1 On-Policy Monte Carlo Control

In the improvement step of MC, we find the argmax action amongst all actions from a state, but we don't take the greedy action each time. Most of the time we choose an action that has maximal estimated action value, while sometimes (with probability ϵ) we select a random action out of all possible actions in that state. Such a policy is known as an ϵ-greedy policy.

Pseudocode

Initialize for all $s \in S, a \in A(s)$:

$Q(s, a) = arbitrary$

$Returns(s, a) = empty\_list$

$\pi(a|s) = an\ arbitrary\ \varepsilon\_soft\ policy$

Repeat forever:

1. Generate a policy using π.

2. For each pair s,a appearing in the episode:

   G = the return that follows the first occurrence of s,a

   Append G to $Returns(s, a)$

   $Q(s, a) = average(Returns(s, a))$

3. For each s in the episode:

   $A^* = argmax_a(Q(s, a))$

   For all $a \in A(s)$:

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \varepsilon/|A(s)|, & \text{if } a = A^* \\ \varepsilon/|A(s)|, & \text{if } a \neq A^* \end{cases}$$

B.2.2 Off-Policy Monte Carlo Control

To learn about the optimal policy (not near optimal) while exploring, a straightforward approach is to use two policies, one that is learned about, called the target policy, and one that is more exploratory and is used to generate behaviour, called the behaviour policy. In this case we say that learning is from data "off" the target policy, and the overall process is termed off-policy learning.

Pseudocode

Initialize for all $s \in S$, $a \in A(s)$:

$Q(s, a)$ = arbitrary

$C(s, a) = 0$

$\pi(s) = \arg\max_a Q(S_t, a)$ (with ties broken consistently)

Repeat forever:

b = any soft policy

Generate an episode using b:

$S_0,\ A_0,\ R_1,\ ...,\ S_{T-1},\ A_{T-1},\ R_T,\ S_T$

$G = 0$

$W = 1$

For $t = T - 1,\ T - 2,$ ... down to 0:

$G = \gamma G + R_{t+1}$

$C(S_t, A_t) = C(S_t, A_t) + W$
$Q(S_t, A_t) = Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
$\pi(S_t) = \arg\max_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit For loop
$W = W\frac{1}{b(A_t|S_t)}$

B.3 Temporal Difference Learning

The simplest TD method makes the update -

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{B.2}$$

immediately on transition to S_{t+1} and receiving R_{t+1}. This TD method is called TD(0), or one-step TD.

We cover two TD learning algorithms –

1. On-Policy TD Control using SARSA algorithm
2. Off-Policy TD Control using Q-Learning algorithm

B.3.1 Sarsa algorithm

We consider transitions from state–action pair to state–action pair, and learn the values of state–action pairs. The theorems assuring the convergence of state values under TD(0) also apply to the corresponding algorithm for action values:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{B.3}$$

This rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state–action pair to the next. This quintuple gives rise to the name *Sarsa* for the algorithm

Pseudocode

Initialize $Q(s, a)$, for all $s \in S, a \in A(s)$, arbitrarily, and $Q(terminal\_state, \cdot) = 0$
Repeat (for each episode):

Initialize S

Choose A from S using policy derived from Q (e.g., ϵ*_greedy*)

Repeat (for each step of episode):

Take action A, observe R, S'

Choose A' from S' using policy derived from Q (e.g., ϵ*_greedy*)

$Q(S,A) = Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$

$S = S'; A = A';$

until S is terminal

B.3.2 Q-Learning algorithm

In this case, the learned action-value function, Q, directly approximates q^* , the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs.

$$Q(S_t,A_t) = Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{B.4}$$

Pseudocode

Initialize $Q(s,a)$, for all $s \in S, a \in A(s)$, arbitrarily, and $Q(terminal\_state,\cdot) = 0$

Repeat (for each episode):

Initialize S

Repeat (for each step of episode):

Choose A from S using policy derived from Q (e.g., ϵ*_greedy*)

Take action A, observe R, S'

$Q(S,A) = Q(S,A) + \alpha[R + \gamma\, max_a Q(S',a) - Q(S,A)]$

$S = S'$

until S is terminal

Bibliography

Abbeel, Pieter and Dan Klein. *The Pac-Man Projects*. URL: http://ai.berkeley.edu/project_overview.html.

Beattie, Charles et al. (2016). "Google DeepMind, DeepMind Lab". In: URL: https://arxiv.org/abs/1612.03801.

Brockman, Greg et al. (2016). "OpenAI Gym". In: URL: https://arxiv.org/abs/1606.01540.

Hasselt, Hado van, Arthur Guez, and David Silver (2015). "Deep Reinforcement Learning with Double Q-learning". In: *AAAI*. URL: https://arxiv.org/abs/1509.06461.

Mnih, Volodymyr et al. (2013). "Playing Atari with Deep Reinforcement Learning". In: *NIPS Deep Learning Workshop*. URL: https://arxiv.org/abs/1312.5602.

Russell, Stuart (1998). "Learning agents for uncertain environments". In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory* COLT' 98, pp. 101–103. URL: https://dl.acm.org/citation.cfm?id=279964.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–489. URL: https://www.nature.com/articles/nature16961.

Sutton, Richard S. and Andrew G. Barto (2017). *Reinforcement Learning: An Introduction*. 2nd ed. The MIT Press, Cambridge, MA. URL: http://incompleteideas.net/book/the-book-2nd.html.

Udacity Deep Reinforcement Learning Nanodegree. URL: https://in.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893.

Watkins, Christopher J. C. H. and Peter Dayan (1992). "Q-learning". In: *Springer* 8.3-4, pp. 279–292. URL: https://link.springer.com/article/10.1007/BF00992698.