A brief introduction to reinforcement learning

Akshay Lamba · Follow
Published in We've moved to freeCodeCamp.org/news
10 min read · Aug 27, 2018

Listen Share More

Photo by Daniel Cheung on Unsplash

Reinforcement Learning is an aspect of Machine learning where an agent learns to behave in an environment, by performing certain actions and observing the rewards/results which it get from those actions.

With the advancements in Robotics Arm Manipulation, Google Deep Mind beating a professional Alpha Go Player, and recently the OpenAI team beating a professional DOTA player, the field of reinforcement learning has really exploded in recent years.

Examples

In this article, we'll discuss:

- What reinforcement learning is and its nitty-gritty like rewards, tasks, etc
- 3 categorizations of reinforcement learning

What is Reinforcement Learning?

Let's start the explanation with an example — say there is a small baby who starts learning how to walk.

Let's divide this example into two parts:

1. Baby starts walking and successfully reaches the couch

Since the couch is the end goal, the baby and the parents are happy.

So, the baby is happy and receives appreciation from her parents. It’s positive — the baby feels good *(Positive Reward +n).*

2. Baby starts walking and falls due to some obstacle in between and gets bruised.

Ouch! The baby gets hurt and is in pain. It's negative — the baby cries *(Negative Reward -n)*.

That's how we humans learn — by trail and error. Reinforcement learning is conceptually the same, but is a computational approach to learn by actions.

Reinforcement Learning

Let's suppose that our reinforcement learning agent is learning to play Mario as a example. The reinforcement learning process can be modeled as an iterative loop that works as below:

- The RL Agent receives **state S^0** from the **environment** i.e. Mario
- Based on that **state S^0,** the RL agent takes an **action A^0**, say — our RL agent moves right. Initially, this is random.
- Now, the environment is in a new state **S^1** (new frame from Mario or the game engine)
- Environment gives some **reward R^1** to the RL agent. It probably gives a +1 because the agent is not dead yet.

This RL loop continues until we are dead or we reach our destination, and it continuously outputs a sequence of **state, action and reward.**

The basic aim of our RL agent is to maximize the reward.

Reward Maximization

The RL agent basically works on a hypothesis of reward maximization. **That’s why reinforcement learning should have best possible action in order to maximize the**

However, things don’t work in this way when summing up all the rewards.

Let us understand this, in detail:

Let us say our RL agent (Robotic mouse) is in a maze which contains **cheese, electricity shocks, and cats**. The goal is to eat the maximum amount of cheese before being eaten by the cat or getting an electricity shock.

It seems obvious to eat the cheese near us rather than the cheese close to the cat or the electricity shock, because the closer we are to the electricity shock or the cat, the danger of being dead increases. As a result, the reward near the cat or the electricity shock, even if it is bigger (more cheese), will be discounted. This is done because of the uncertainty factor.

It makes sense, right?

Discounting of rewards works like this:

We define a discount rate called **gamma**. It should be between 0 and 1. The larger the gamma, the smaller the discount and vice versa.

So, our cumulative expected (discounted) rewards is:

Cumulative expected rewards

Tasks and their types in reinforcement learning

A **task** is a single instance of a reinforcement learning problem. We basically have two types of tasks: **continuous and episodic.**

Continuous tasks

These are the types of tasks that continue forever. For instance, a RL agent that does automated Forex/Stock trading.

Photo by Chris Liverani on Unsplash

In this case, the agent has to learn how to choose the best actions and simultaneously interacts with the environment. There is no starting point and end state.

The RL agent has to keep running until we decide to manually stop it.

Episodic task

In this case, we have a starting point and an ending point **called the terminal state. This creates an episode**: a list of States (S), Actions (A), Rewards (R).

For example, playing a game of *counter strike*, where we shoot our opponents or we get killed by them.We shoot all of them and complete the episode or we are killed. So, there are only two cases for completing the episodes.

Exploration and exploitation trade off

There is an important concept of the exploration and exploitation trade off in reinforcement learning. Exploration is all about finding more information about an environment, whereas exploitation is exploiting already known information to maximize the rewards.

Real Life Example: Say you go to the same restaurant every day. You are basically **exploiting.** But on the other hand, if you search for new restaurant every time before going to any one of them, then it's **exploration**. Exploration is very important for the search of future rewards which might be higher than the near rewards.

In the above game, our robotic mouse can have a good amount of small cheese (+0.5 each). But at the top of the maze there is a big sum of cheese (+100). So, if we only focus on the nearest reward, our robotic mouse will never reach the big sum of cheese — it will just exploit.

But if the robotic mouse does a little bit of exploration, it can find the big reward i.e. the big cheese.

This is the basic concept of the **exploration and exploitation trade-off.**

Approaches to Reinforcement Learning

Let us now understand the approaches to solving reinforcement learning problems. Basically there are 3 approaches, but we will only take 2 major approaches in this article:

1. Policy-based approach

In policy-based reinforcement learning, we have a policy which we need to optimize. The policy basically defines how the agent behaves:

We learn a policy function which helps us in mapping each state to the best action.

Getting deep into policies, we further divide policies into two types:

- **Deterministic**: a policy at a given state(s) will always return the same action(a). **It means, it is pre-mapped as S=(s) ➡ A=(a).**
- **Stochastic:** It gives a distribution of probability over different actions**. i.e Stochastic Policy ➡ p(A = a | S = s)**

2. Value Based

In value-based RL, the goal of the agent is to optimize the value function *V(s)* which is defined as a function that tells us the maximum expected future reward the agent shall get at each state.

The value of each state is the total amount of the reward an RL agent can expect to collect over the future, from a particular state.

The agent will use the above value function to select which state to choose at each step. The agent will always take the state with the biggest value.

In the below example, we see that at each step, we will take the biggest value to achieve our goal: 1 ➡ 3 ➡ 4 ➡ **6** so on…

Maze

The game of Pong — An Intuitive case study

Let us take a real life example of playing pong. This case study will just introduce you to the Intuition of **How reinforcement Learning Works**. We will not get into details in this example, but in the next article we will certainly dig deeper.

Suppose we teach our RL agent to play the game of Pong.

Basically, we feed in the game frames (new states) to the RL algorithm and let the algorithm decide where to go up or down. This network is said to be a **policy network,** which we will discuss in our next article.

The method used to train this Algorithm is called the **policy gradient.** We feed random frames from the game engine, and the algorithm produces a random output which gives a reward and this is fed back to the algorithm/network. This is an **iterative process.**

We will discuss **policy gradients** in the next Article with greater details.

Environment = Game Engine and Agent = RL Agent

In the context of the game, the score board acts as a reward or feed back to the agent. Whenever the agent tends to score +1, it understands that the action taken by it was good enough at that state.

Now we will train the agent to play the pong game. To start, we will feed in a bunch of game frame **(states)** to the network/algorithm and let the algorithm decide the action.The Initial actions of the agent will obviously be bad, but our agent can sometimes be lucky enough to score a point and this might be a random event. But due to this lucky random event, it receives a reward and this helps the agent to understand that the series of actions were good enough to fetch a reward.

Results during the training

So, in the future, the agent is likely to take the actions which will fetch a reward over an action which will not. Intuitively, the RL agent is leaning to play the game.

Source: OLEGIF.com

Limitations

During the training of the agent, when an agent loses an episode, then the algorithm will discard or lower the likelyhood of taking all the series of actions which existed in this episode.

Red Demarcation Shows all the action Taken in a losing episode

But if the agent was performing **well** from the start of the episode, but just due to the last 2 actions the agent lost the game, it does not make sense to discard all the actions. Rather it makes sense if we just remove the last 2 actions which resulted in the loss.

Green Demarcation shows all the action which where correct and Red Demarcation are the action Which Should be removed.

This is called the **Credit Assignment Problem.** This problem arises because of a **sparse reward setting.** That is, instead of getting a reward at every step, we get the reward at the end of the episode. So, it's on the agent to learn which actions were correct and which actual action led to losing the game.

So, due to this sparse reward setting in RL, the algorithm is very sample-inefficient. This means that huge training examples have to be fed in, in order to train the agent. But the fact is that sparse reward settings fail in many circumstance due to the complexity of the environment.

So, there is something called **rewards shaping** which is used to solve this. But again, rewards shaping also suffers from some limitation as we need to design a custom reward function for every game.

Closing Note

Today, reinforcement learning is an exciting field of study. Major developments has been made in the field, of which deep reinforcement learning is one.

We will cover deep reinforcement learning in our upcoming articles. This article covers a lot of concepts. Please take your own time to understand the basic concepts of reinforcement learning.

But, I would like to mention that reinforcement is not a secret black box. Whatever advancements we are seeing today in the field of reinforcement learning are a result of bright minds working day and night on specific applications.

Next time we'll work on a Q-learning agent and also cover some more basic stuff in reinforcement learning.

Until, then enjoy AI 😉…

> ***Important :** This article is 1st part of Deep Reinforcement Learning series, The Complete series shall be available both on Text Readable forms on Medium and in Video explanatory Form on my channel on YouTube.*

For deep and more Intuitive understanding of reinforcement learning, I would recommend that you watch the below video:

Source

Subscribe to my YouTube channel For more AI videos : **ADL** .

If you liked my article, please click the 👏 as I remain motivated to write stuffs and Please follow me on Medium &

If you have any questions, please let me know in a comment below or **Twitter**.
Subscribe to my YouTube Channel For More Tech videos : **ADL** .

Machine Learning · Reinforcement Learning · Deep Learning · Artificial Intelligence · Tech

Follow

Written by Akshay Lamba

516 Followers · Writer for We've moved to freeCodeCamp.org/news

Startup, Web,Mobile Dev, AI, ML ..... :) Instagram : https://www.instagram.com/buildlikelamba # Linkedin : https://www.linkedin.com/in/iamadl/

More from Akshay Lamba and We’ve moved to freeCodeCamp.org/news

Akshay Lamba in We’ve moved to freeCodeCamp.org/news

Get to know TensorFlow.js in 7 minutes

And learn how you can run ML/DL models directly in the browser

7 min read · Jul 26, 2018

-- 4

TK in We’ve moved to freeCodeCamp.org/news

Learning Python: From Zero to Hero

This post was originally published at TK's Blog.

11 min read · Sep 30, 2017

-- 136

Peter Gleeson in We've moved to freeCodeCamp.org/news

An A-Z of useful Python tricks

Python is one of the world's most popular, in-demand programming languages

9 min read · Aug 28, 2018

-- 57

Akshay Lamba in Towards Data Science

Introduction to ML5.js

A Beginner's Friendly Machine Learning for the Web.

5 min read · Aug 8, 2018

-- 1

See all from Akshay Lamba

See all from We've moved to freeCodeCamp.org/news

Recommended from Medium

Waleed Mousa in Artificial Intelligence in Plain English

Building a Tic-Tac-Toe Game with Reinforcement Learning in Python: A Step-by-Step Tutorial

Welcome to this step-by-step tutorial on how to build a Tic-Tac-Toe game using reinforcement learning in Python. In this tutorial, we will…

9 min read · Mar 13

-- 1

Mehul Gupta in Data Science in your pocket

Training OpenAI gym environments using REINFORCE algorithm in reinforcement learning

Policy gradient methods explained with codes

8 min read · Mar 26

--

Lists

Predictive Modeling w/ Python

18 stories · 258 saves

Natural Language Processing

494 stories · 128 saves

AI Regulation

6 stories · 76 saves

Practical Guides to Machine Learning

10 stories · 269 saves

Wouter van Heeswijk, PhD in Towards Data Science

Proximal Policy Optimization (PPO) Explained

The journey from REINFORCE to the go-to algorithm in continuous control

· 13 min read · Nov 29, 2022

-- 2

Renu Khandelwal

Unlocking the Secrets of Actor-Critic Reinforcement Learning: A Beginner's Guide

Understanding Actor-Critic Mechanisms, Different Flavors of Actor-Critic Algorithms, and a Simple Implementation in PyTorch

✦ · 6 min read · Feb 21

-- 1

cyber girl

How I Made $100k On My First Indie Game

Here's the transcript:

5 min read · Aug 4

--

Ming-Hao Hsu

[RL] E-MAPP: Efficient Multi-Agent Reinforcement Learning with Parallel Program Guidance…

Paper Link: E-MAPP: Efficient Multi-Agent Reinforcement Learning with Parallel Program Guidance

3 min read · Jul 21

--

See more recommendations