

Reinforcement Learning: An introduction (Part 1/4)

Cédric Vandelaer

13–16 minutes



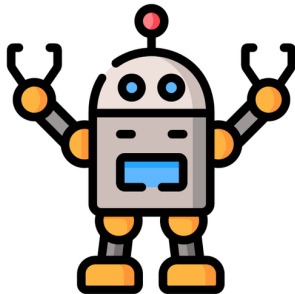
10 min read

Aug 20, 2022

Hi and welcome to the first part of a series on Reinforcement Learning.

Press enter or click to view image in full size

Let's explore
reinforcement learning!



If you somehow ended up here without having heard of Reinforcement Learning (RL) before, then let me summarize it as follows: “*RL is a general framework for training an artificial intelligence model to solve a certain task or goal*” ... or in layman’s terms, we make AI do cool things!

The goal of this blog series is to learn about RL and simultaneously explore some of the more recent research later on. We will start

from the very basics and work our way towards more advanced topics. Even if you have almost no prior programming and/or mathematics knowledge, you should be able to follow along pretty smoothly.

The first mini-series will be split into four parts:

- Part 1: What is Reinforcement learning?
- [Part 2](#): RL terminology and formal concepts
- [Part 3](#): The REINFORCE algorithm
- [Part 4](#): Implementing the REINFORCE algorithm

At the same time, this mini-series will be the introduction to future posts with increasing complexity. Feel free to skip to the next part if you are already familiar with the content.

Note: If you are reading this on a smartphone browser, you might not be able to view the subscripts. You can download the Medium app to mitigate this.

That sounds cool! ... but what can I do with RL?

Reinforcement learning is a framework to learn any task. In theory, RL can solve any problem that is phrased as a Markov Decision Process. We will explain what that means later on. For now, let’s have a look at some successful applications. If you enjoy these examples, please be sure to also check out the work from the original authors.

General

One of the videos I love showing is the hide and seek video from OpenAI. The video is a nice example of how RL can help us with finding novel solutions to problems, without explicitly programming tactics or solution methods.

Games

One of the early successes of Deep RL (which is combining RL with neural networks), was the ability to learn how to play Atari games, straight from pixels. Later on, researchers took it upon themselves to not only evaluate RL on the (relatively) simple Atari games, but rather evaluate it on the hardest competitive games out there. The assumption here is, that if RL can solve these complex games, it can also generalize to challenging real-world settings. As an example, this is Deepmind's AlphaStar taking on a pro-gamer in the game StarCraft 2.

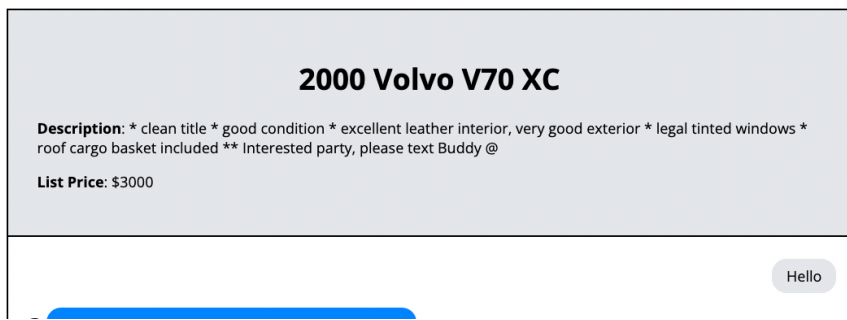
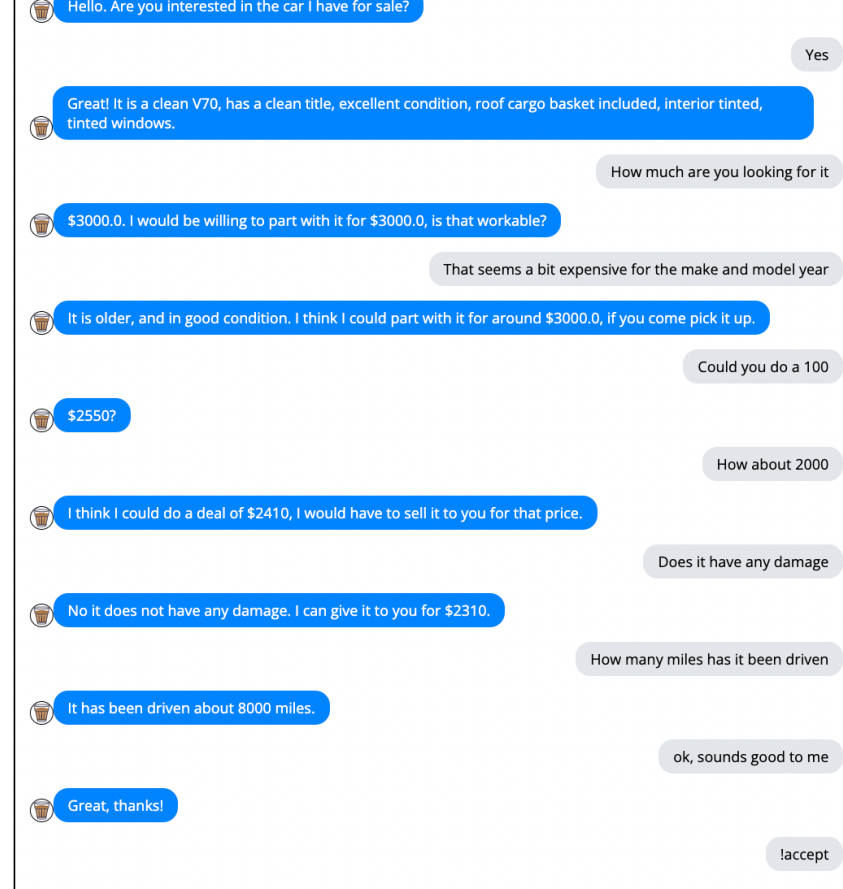
Robotics

Solving tasks in simulations and video games is one thing, but what about real life? Another popular field where RL is often applied (or at least holds great promise), is robotics. Robotics are significantly harder than simulations for various reasons. Think for example about the time it takes to repeatedly make a robot try out a certain action. Or think about the safety requirements involved for robotics. In the example below, you can see how the ANYmal robot from the Robotics System Lab in Zürich learned to recover from a fall.

Real world examples

RL can be applied to many other domains than the ones I just mentioned. Advertising, finance, healthcare, ... just to name a few. As a final example, I present a goal-oriented chatbot, trained to negotiate about sales (source: <https://siddharthverma314.github.io/research/chai-acl-2022/>).

Press enter or click to view image in full size



RL: The basics

A description of the RL framework is as follows: We have an agent that tries to solve a task in a certain environment. The concept of agent should be taken very broadly here, an agent can be a robot, a chatbot, a virtual character, etc. . At every timestep t , the agent needs to choose an action a . After this action it might receive a reward r and we get a new observation of its state s . The new state can be determined both by the action of the agent and also by the environment the agent is operating in.

The RL problem is trying to maximize the cumulative reward the agent gets over time.

Press enter or click to view image in full size

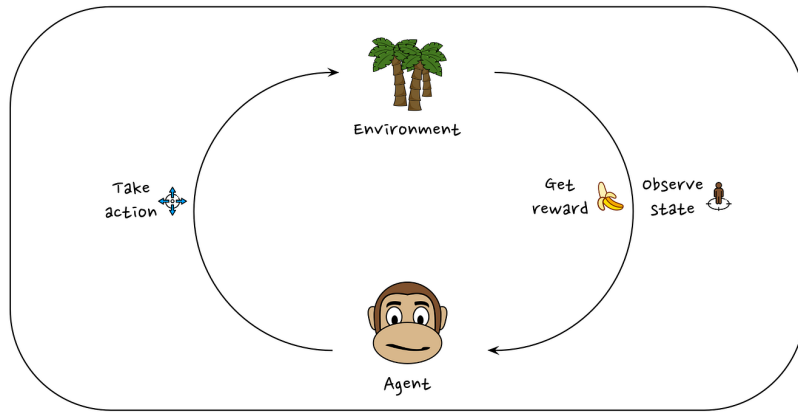
Get Cédric Vandelaer's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The likely reason for this intersection of domains, is that reinforcement learning is the study of a fundamental problem. **It is essentially the science of decision taking.** In these series, we will be looking at it from the umbrella of computer science and machine learning.

Press enter or click to view image in full size

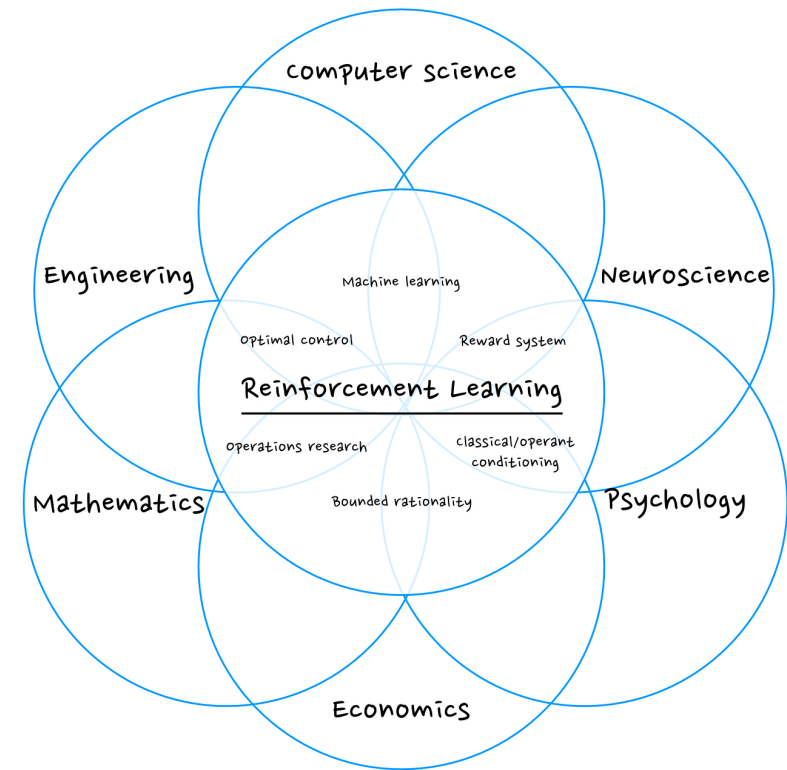


Imagine our **agent** is a monkey and the task we want the monkey to solve, is to pick up as many bananas as possible. **At every timestep**, the monkey needs to decide to **take an action**. The actions could be to step towards the tree, grab something, climb, ... Perhaps the **reward** at every timestep can be defined as the number of bananas the monkey got at that timestep. After every action, the monkey will also be in a new **state**. Maybe we define the state of the monkey as its position in the world. So when the monkey takes a step, the state at the next timestep would be the coordinates of the monkey at the next timestep. We are now searching for the **optimal behavior, the best sequence of actions the monkey can take**, to maximize the cumulative number of bananas it will get.

How does RL fit in the bigger picture?

You might look at this framework and think: *“Hey, isn’t that exactly what people are already studying in the field of ...?”*. And in fact, you might be right.

In other domains like engineering or mathematics, people have often been studying the same problems with different names and methods. Or in fields like neuroscience and psychology, some similarities can be found in the way our brain “rewards” us by releasing dopamine.



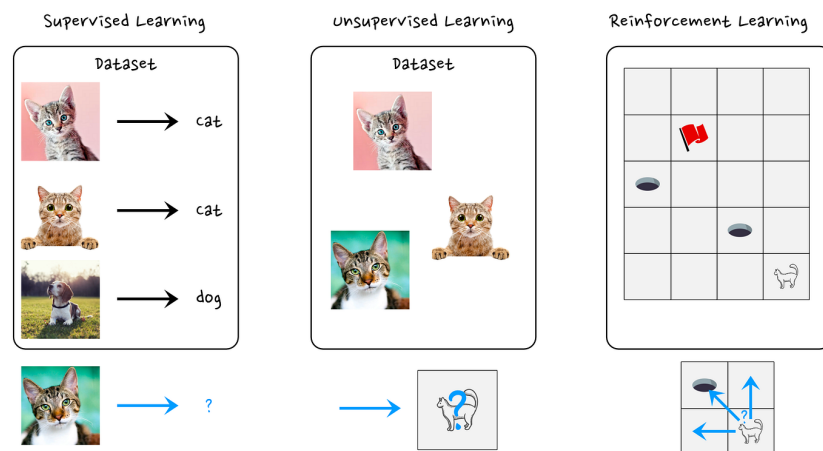
Content taken from David Silverman's course on RL, published by Deepmind

This general applicability is also what makes RL so interesting to me personally. RL is one of the potential technologies that could get us closer to general AI: an AI system that can solve any task, in contrast to a narrow set of tasks. There are also other technologies (e.g. big language models, graph neural networks, ...) that are

making some strides in this regard, but the problem statement of RL in particular seems to be the most ambitious.

Another way of situating RL, is by looking at how it compares to other learning paradigms. Within the field of machine learning, people often distinguish between **Supervised learning**, **Unsupervised learning** and **Reinforcement learning**.

Press enter or click to view image in full size



When it comes to **Supervised learning**, we are essentially trying to learn a function, a mapping from X to Y . Our dataset consists of samples X and during training we provide the AI system with labels Y of what these samples should map to. Our AI model is successful when it correctly predicts a label y given an (unseen) sample x .

As an example, say we have a dataset consisting of cat and dog images. Each sample (image) has a label, stating whether the image contains a "dog" or a "cat". The goal of our supervised model, is now to learn how to map a dog-image to the label "dog" and a cat-image to the label "cat". After it has learned this mapping, the hope is that this AI model can repeat this process for new images that it has not seen before.

In an **Unsupervised learning** context, we no longer provide any labels Y . So the task for the AI system now becomes learning some general statistics about the dataset. We could for example give an

AI the task to generate a new sample, similar to the ones seen in the dataset. In this case the model would be considered successful if it manages to correctly learn the interesting characteristics of a dataset.

Reinforcement Learning is rather different from the previous paradigms. In the case of RL, we consider an agent that is actively interacting with an environment. Through its interactions, it is possible that it may influence the environment that it operates in. The "dataset" we need to consider here, are the actions our agent took and the accumulated rewards it got by taking those actions. An added difficulty here is that our dataset is non-static. Say that our agent acts in a certain way, we can then collect some data of the actions our agent took and we can try to optimize these (e.g. do more of the actions that led to a successful result). But as a result of this optimization, we have now changed the behavior of this agent, and thus we will need to collect new data to see how well our agent fares now.

If RL is so great, then why isn't everyone using RL?

After reading all this, you might be wondering why people aren't using RL to solve all imaginable problems. The truth is that even though the field has made a lot of advancements in the last few years, there are still a few fundamental problems to be solved. Progress is being made all the time, but to give you an idea of what you might encounter, I'll list a few common ones.

Sample (in-)efficiency

It is generally known that RL is very sample-inefficient. We regard a "sample" as an interaction with the environment. RL needs a lot of samples/interactions to be able to solve a task. In this sense, RL is very inefficient compared to humans, for example, it doesn't take a human dozens of hours to learn how to play an Atari game.

This sample-efficiency can in part be explained by the fact that

humans can leverage a lot of their previous knowledge (priors) when they encounter a new task. A human can for example reuse some of the knowledge and skills of previous games and/or concepts they already acquired from other experiences throughout their life. An RL-agent in contrast, starts the learning process without any assumptions.

Another thing to mention is that leveraging knowledge from previous tasks is also an active research topic. I'll just put one example (out of many) here to give you an idea.

- [Deepmind Gato](#)
- [Google Jump-Start RL](#)



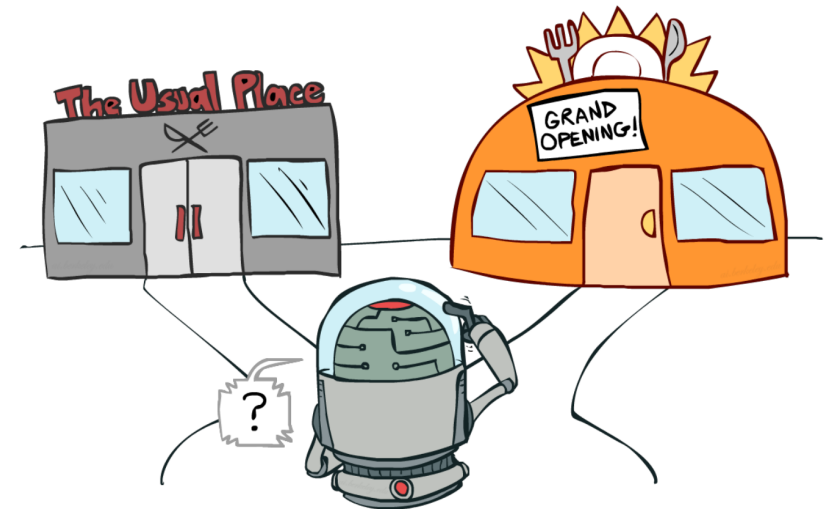
The exploration-exploitation trade-off

While the previous problem sounds more like an engineering effort (it's not), the exploration-exploitation trade-off seems more fundamental. Whenever we train an RL-agent, the agent will need some time to explore, it needs to take some actions that it hasn't taken before, in order to discover how to solve the problem. On the other hand, we can't let the agent always take random actions, because these random actions might lead to nothing. Sometimes

we want the agent to leverage what it has already learned to try and optimize further. This is the exploration-exploitation trade-off, we want an automated way to strike a good balance between letting the agent explore and taking actions for which it already knows what they will lead to.

For a lot of problems, it is quite possible that the agent gets stuck in a local optimum.

Press enter or click to view image in full size



The exploration-exploitation trade-off sounds very much tractable at first, but it turns out to be one of the hardest problems for RL to solve. To give you an idea about how hard this problem is: The problem was originally considered by the scientists of the allied forces, but was suggested to be dropped over to Germany, because it was deemed so intractable that they wanted the German scientists to also waste their time on it.

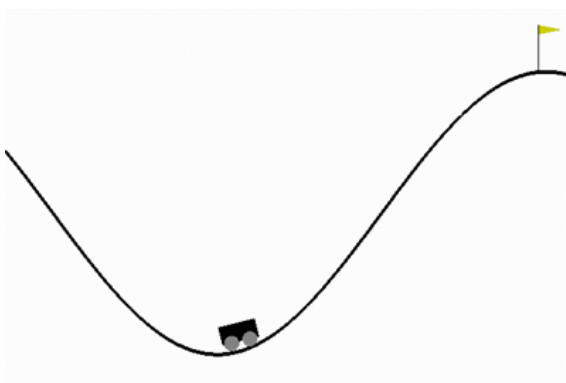
No silver bullet for this problem has been found and there are many people working on various solutions. I will leave a link here to a previously proposed solution called "curiosity-driven exploration" which I found particularly interesting.

[Curiosity-driven exploration by Self-supervised Prediction](#)

The sparse-reward problem

Another rather fundamental problem, is the so called Sparse-reward problem. As the name implies, this problem occurs when our RL-agent receives so little rewards, that it actually gets no feedback on how it should improve.

Imagine for example this mountain car. The agent needs to move the car left and right, such that it gets enough momentum to reach the top. Initially though, the agent doesn't know that it needs to move the car back and forth to reach the top. If we only give our agent a reward (a positive feedback signal) when we have reached the flag, it might not ever get a positive feedback signal, simply because it might never reach the flag by taking random actions (exploration).



Something commonly done to counteract this problem, is by “reward shaping”. We will modify the reward such that the agent gets more feedback signals to learn from. In case of the mountain car, we could for example also give the agent a positive reward based on the speed or altitude it achieves. However, reward shaping is not a scalable solution. Luckily other solutions are being sought after.

I'm leaving another example here which I think is an interesting (but not generally applicable approach) for counteracting sparse rewards.

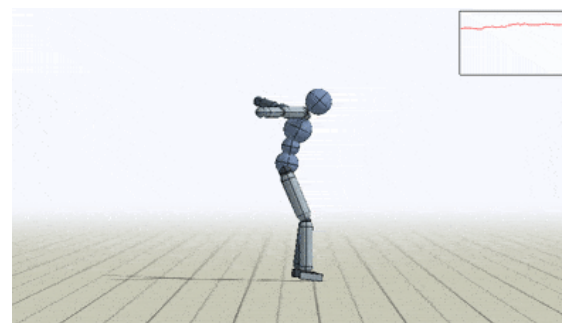
[Hindsight Experience Replay](#)

The aforementioned problems are just some prominent examples,

but there are in fact many more problems that are being looked into by some of the brightest minds out there. The good news is that regular and steady progress is being made.

Conclusion

Phew, that was a lot of information in a very short period, but you made it through! This first introduction should give you a good idea of what RL is all about.



In the [next article](#), we will start formalizing some of the ideas and concepts which we have briefly touched upon in this post, as a preparation for getting our hands dirty and implementing these ideas ourselves.