SYNOPSYS® (https://www.synopsys.com)

What is Reinforcement Learning?

By **Piyush Verma** (https://www.linkedin.com/in/piyush-verma-profile/) and **Stelios Diamantidis** (https://www.linkedin.com/in/stelix/) | Last Updated: April 27, 2021

Table of Contents

How RL works

Examples of RL

Benefits of RL

Challenges with RL

RL vs. Supervised Learning

Future of RL

Reinforcement Learning and Synopsys

Definition

Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. This optimal behavior is learned through interactions with the environment and observations of how it responds, similar to children exploring the world around them and learning the actions that help them achieve a goal.

A system that is based on trial and error to get the best reward/success.

In the absence of a supervisor, the learner must independently discover the sequence of actions that maximize the reward. This discovery process is akin to a trial-and-error search. The quality of actions is measured by not just the immediate reward they return, but also the delayed reward they might fetch. As it can learn the actions that result in eventual success in an unseen environment without the help of a supervisor, reinforcement learning is a very powerful algorithm.

Reinforcement Learning Explained in 90 Seconds | Synopsys

How Does Reinforcement Learning Work?

The Reinforcement Learning problem involves an agent exploring an unknown environment to achieve a goal. RL is based on the hypothesis that all goals can be described by the maximization of expected cumulative reward. The agent must learn to sense and perturb the state of the environment using its actions to derive maximal reward. The formal framework for RL borrows from the problem of optimal control of Markov Decision Processes (https://en.wikipedia.org/wiki/Markov_decision_process) (MDP).

The main elements of an RL system are:

1. The agent or the learner
2. The environment the agent interacts with
3. The policy that the agent follows to take actions
4. The reward signal that the agent observes upon taking actions

The framework of an RL system!

A useful abstraction of the reward signal is the value function, which faithfully captures the 'goodness' of a state. While the reward signal represents the immediate benefit of being in a certain state, the value function captures the cumulative reward that is expected to be collected from that state on, going into the future. The objective of an RL algorithm is to discover the action policy that maximizes the average value that it can extract from every state of the system.

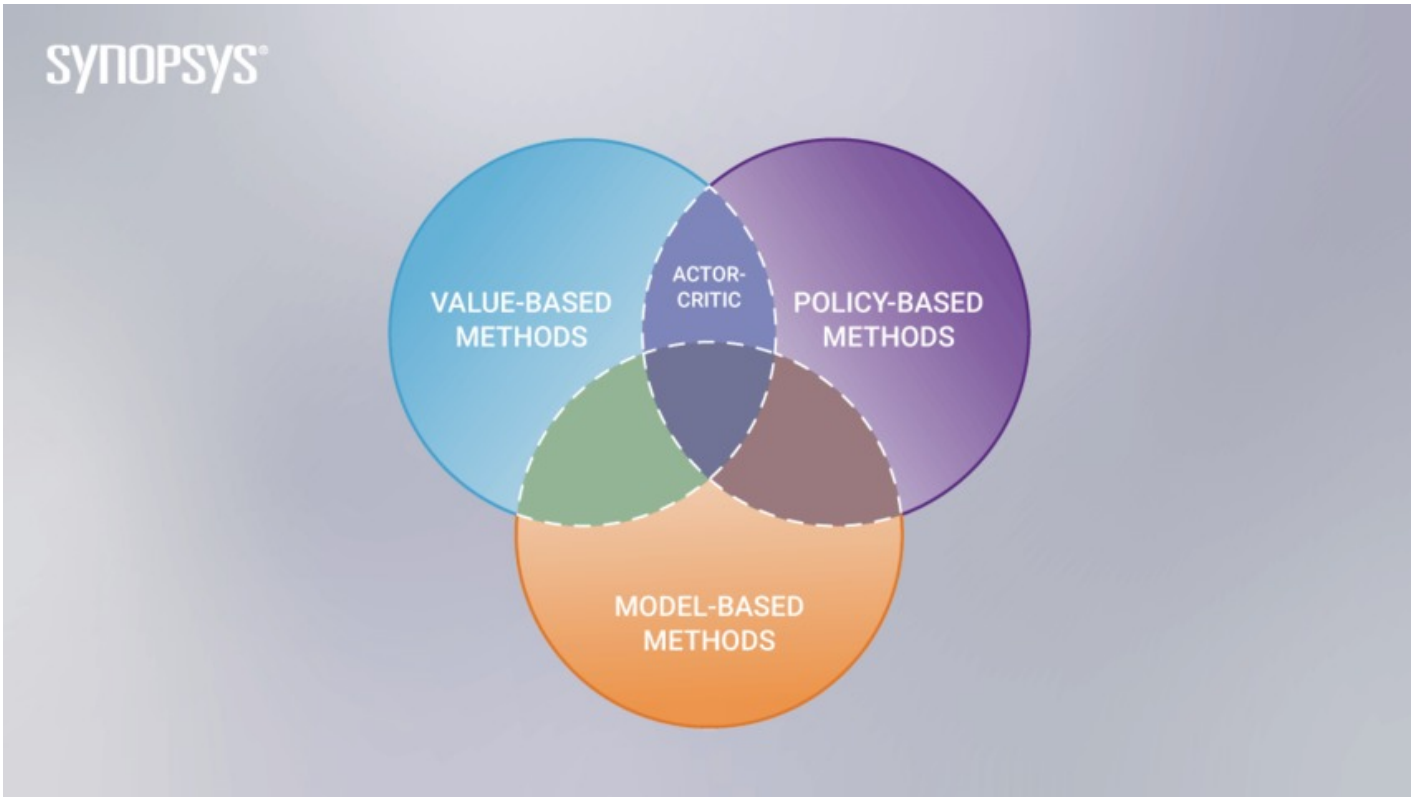
Value and reward functions are how RL is using to improve and determine its success rate over time!

Reinforcement learning models are either based on a particular policy to follow determined rules or are model-free to explore different situations using pure trial and error.

RL algorithms can be broadly categorized as model-free and model-based. Model-free algorithms do not build an explicit model of the environment, or more rigorously, the MDP. They are closer to trial-and-error algorithms that run experiments with the environment using actions and derive the optimal policy from it directly. Model-free algorithms are either value-based or policy-based. Value-based algorithms consider optimal policy to be a direct result of estimating the value function of every state accurately. Using a recursive relation described by the Bellman equation (https://en.wikipedia.org/wiki/Bellman_equation), the agent interacts with the environment to sample trajectories of states and rewards. Given enough trajectories, the value function of the MDP can be estimated. Once the value function is known, discovering the optimal policy is simply a matter of acting greedily with respect to the value function at every state of the process. Some popular value-based algorithms are SARSA (https://en.wikipedia.org/wiki/State%E2%80%93action%E2%80%93reward%E2%80%93state learning (https://en.wikipedia.org/wiki/Q-learning). Policy-based algorithms, on the other hand, directly estimate the optimal policy without modeling the value function. By parametrizing the policy directly using learnable weights, they render the learning problem into an explicit optimization problem. Like value-based algorithms, the agent samples

trajectories of states and rewards; however, this information is used to explicitly improve the policy by maximizing the average value function across all states. Popular policy-based RL algorithms include Monte Carlo policy gradient (REINFORCE) and deterministic policy gradient (DPG). Policy-based approaches suffer from a high variance which manifests as instabilities during the training process. Value-based approaches, though more stable, are not suitable to model continuous action spaces. One of the most powerful RL algorithms, called the actor-critic algorithm, is built by combining the value-based and policy-based approaches. In this algorithm, both the policy (actor) and the value function (critic) are parametrized to enable effective use of training data with stable convergence.

Algorithms examples for reinforcement learning



Model-based RL algorithms build a model of the environment by sampling the states, taking actions, and observing the rewards. For every state and a possible action, the model predicts the expected reward and the expected future state. While the former is a regression problem, the latter is a density estimation problem. Given a model of the environment, the RL agent can plan its actions without directly interacting with the environment. This is like a thought experiment that a human might run when trying to solve a problem. When the process of planning is interweaved with the process of policy estimation, the RL agent's ability to learn.

Examples of Reinforcement Learning

It is very useful where you can use trial and error to maximize output while trying out many situations.

Any real-world problem where an agent must interact with an uncertain environment to meet a specific goal is a potential application of RL. Here are a few RL success stories:

- **Robotics.** Robots with pre-programmed behavior are useful in structured environments, such as the assembly line of an automobile manufacturing plant, where the task is repetitive in nature. In the real world, where the response of the environment to the behavior of the robot is uncertain, pre-programming accurate actions is nearly impossible. In such scenarios, RL provides an efficient way to build general-purpose robots. It has been successfully applied to robotic path planning, where a robot must find a short, smooth, and navigable path between two locations, void of collisions and compatible with the dynamics of the robot.
- **AlphaGo.** One of the most complex strategic games is a 3,000-year-old Chinese board game called Go. Its complexity stems from the fact that there are 10^270 possible board combinations, several orders of magnitude more than the game of chess. In 2016, an RL-based Go agent called AlphaGo defeated the greatest human Go player. Much like a human player, it learned by experience, playing thousands of games with professional players. The latest RL-based Go agent has the capability to learn by playing against itself, an advantage that the human player doesn't have.
- **Autonomous Driving.** An autonomous driving system must perform multiple perception and planning tasks in an uncertain environment. Some specific tasks where RL finds application include vehicle path planning and motion prediction. Vehicle path planning requires several low and high-level policies to make decisions over varying temporal and spatial scales. Motion prediction is the task of predicting the movement of pedestrians and other vehicles, to understand how the situation might develop based on the current state of the environment.

Benefits of Reinforcement Learning

Reinforcement learning is applicable to a wide range of complex problems that cannot be tackled with other machine learning algorithms. RL is closer to artificial general intelligence (AGI), as it possesses the ability to seek a long-term goal while exploring various possibilities autonomously. Some of the benefits of RL include:

- **Focuses on the problem as a whole.** Conventional machine learning algorithms are designed to excel at specific subtasks, without a notion of the big picture. RL, on the other hand, doesn't divide the problem into subproblems; it directly works to maximize the long-term reward. It has an obvious purpose, understands the goal, and is capable of trading off short-term rewards for long-term benefits.
- **Does not need a separate data collection step.** In RL, training data is obtained via the direct interaction of the agent with the environment. Training data is the learning agent's experience, not a separate collection of data that has to be fed to the algorithm. This significantly reduces the burden on the supervisor in charge of the training process.

Very resourceful, adaptive, and great at the overall task!

- **Works in dynamic, uncertain environments.** RL algorithms are inherently adaptive and built to respond to changes in the environment. In RL, time matters and the experience that the agent collects is not independently and identically distributed (i.i.d.), unlike conventional machine learning algorithms. Since the dimension of time is deeply buried in the mechanics of RL, the learning is inherently adaptive.

Challenges with Reinforcement Learning

While RL algorithms have been successful in solving complex problems in diverse simulated environments, their adoption in the real world has been slow. Here are some of the challenges that have made their uptake difficult:

- **RL agent needs extensive experience.** RL methods autonomously generate training data by interacting with the environment. Thus, the rate of data collection is limited by the dynamics of the environment. Environments with high latency slow down the learning curve. Furthermore, in complex environments with high-dimensional state spaces, extensive exploration is needed before a good solution can be found.
- **Delayed rewards.** The learning agent can trade off short-term rewards for long-term gains. While this foundational principle makes RL useful, it also makes it difficult for the agent to discover the optimal policy. This is especially true in environments where the outcome is unknown until a large number of sequential actions are taken. In this scenario, assigning credit to a previous action for the final outcome is challenging and can introduce large variance during training. The game of chess is a relevant example here, where the outcome of the game is unknown until both players have made all their moves.
- **Lack of interpretability.** Once an RL agent has learned the optimal policy and is deployed in the environment, it takes actions based on its experience. To an external observer, the reason for these actions might not be obvious. This lack of interpretability interferes with the development of trust between the agent and the observer. If an observer could explain the actions that the RL agent tasks, it would help him in understanding the problem better and discovering limitations of the model, especially in high-risk environments.

Huge amounts of interaction with the environment

Takes time to provide consistent results

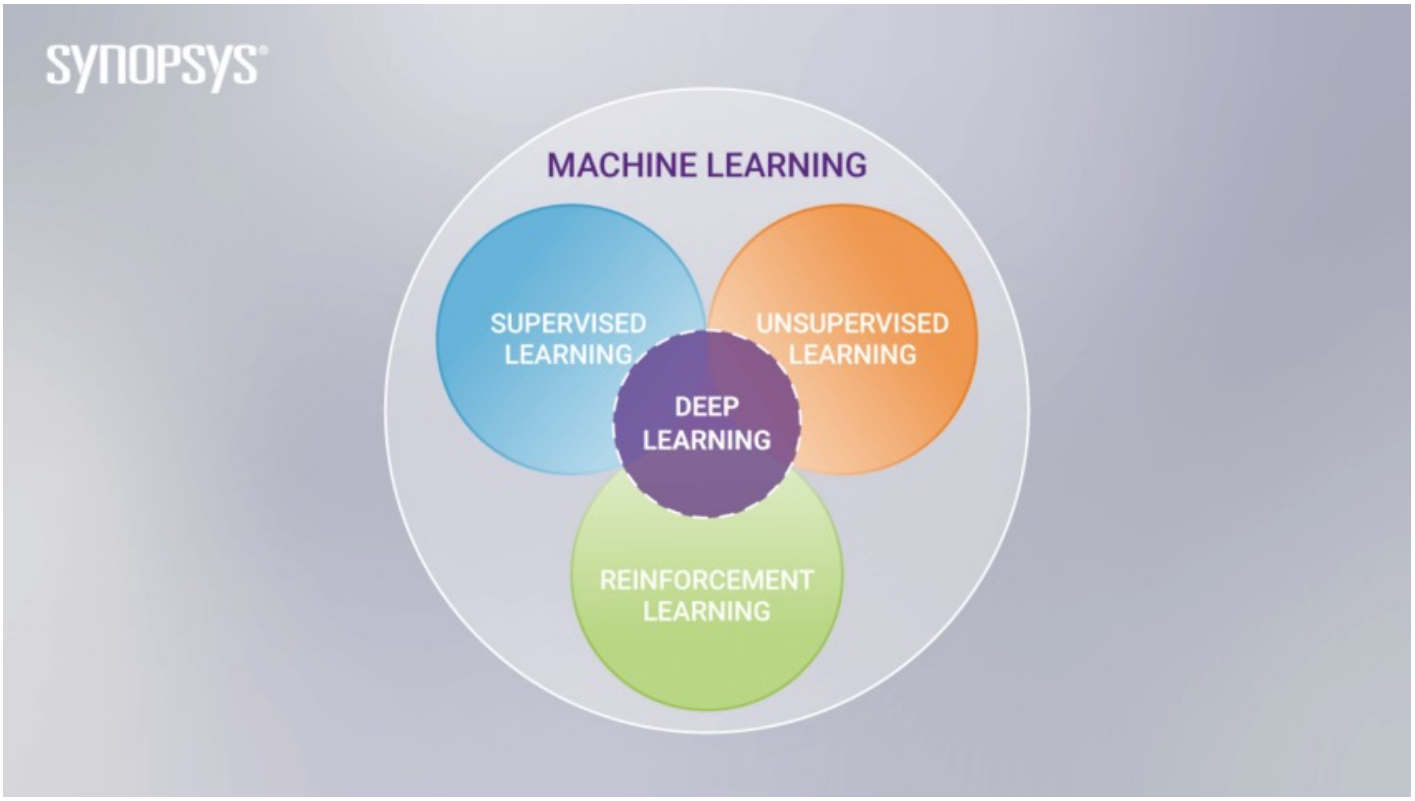
RL Agent is hard to interpret which could a problem in business cases.

Reinforcement Learning vs. Supervised Learning

Supervised learning is a paradigm of machine learning that requires a knowledgeable supervisor to curate a labelled dataset and feed it to the learning algorithm. The supervisor is responsible for collecting this training data – a set of examples such as images, text snippets, or audio clips, each with a specification that assigns the example to a specific class. In the RL setting, this training dataset would look like a set of situations and actions, each with a 'goodness' label attached to it. The core function of a supervised learning algorithm is to extrapolate and generalize, to make predictions for examples that are not included in the training dataset.

Only works based on new situations rather than labeled data!

RL is a separate paradigm of machine learning. RL does not require a supervisor or a pre-labelled dataset; instead, it acquires training data in the form of experience by interacting with the environment and observing its response. This crucial difference makes RL feasible in complex environments where it is impractical to separately curate labelled training data that is representative of all the situations that the agent would encounter. The only approach that is likely to work in these situations is where the generation of training data is autonomous and integrated into the learning algorithm itself, much like RL.



Since RL does not require a supervisor, it is important to point out that RL is not the same as unsupervised learning, yet another paradigm of machine learning. In unsupervised learning, the training data is not labelled, and the objective is to uncover the hidden structure in the data. A knowledge of this hidden structure lets the model group similar examples or estimate the distribution function that generated the examples. Uncovering this hidden structure does not solve the RL problem, which is to maximize the reward at the end of a trajectory. However, the knowledge of a hidden structure in the agent's experience can help speed up the learning process.

RL is completely different to

A challenge that is unique to RL algorithms is the trade-off between exploration and exploitation. This trade-off doesn't arise in either supervised or unsupervised machine learning. An RL agent must strike a careful balance between exploiting its past experience and exploring the unknown states of the environment. The right balance would lead the agent to discover the optimal policy that yields maximal reward. If the agent continues to exploit the past experience only, it is likely to get stuck in a local minima and produce a sub-optimal policy. On the other hand, if the agent continues to explore without exploiting, it might never find a good policy.

What is Design Space Optimization?

DSO is a novel approach to searching large design spaces enabled by recent advancements in machine-learning.

DSO Overview (https://www.synopsys.com/glossary/what-is-design-space-optimization.html)

What's the Future of Reinforcement Learning?

In recent years, significant progress has been made in the area of deep reinforcement learning. Deep reinforcement learning uses deep neural networks to model the value function (value-based) or the agent's policy (policy-based) or both (actor-critic). Prior to the widespread success of deep neural networks, complex features had to be engineered to train an RL algorithm. This meant reduced learning capacity, limiting the scope of RL to simple environments. With deep learning, models can be built using millions of trainable weights, freeing the user from tedious feature engineering. Relevant features are generated automatically during the training process, allowing the agent to learn optimal policies in complex environments.

Traditionally, RL is applied to one task at a time. Each task is learned by a separate RL agent, and these agents do not share knowledge. This makes learning complex behaviors, such as driving a car, inefficient and slow. Problems that share a common information source, have related underlying structure, and are interdependent can get a huge performance boost by allowing multiple agents to work together. Multiple agents can share the same representation of the system by training them simultaneously, allowing improvements in the performance of one agent to be leveraged by another. A3C (Asynchronous Advantage Actor-Critic) is an exciting development in this area, where related tasks are learned concurrently by multiple agents. This multi-task learning scenario is driving RL closer to AGI, where a meta-agent learns how to learn, making problem-solving more autonomous than ever before.

Limits of reinforcement learning are that some agents are inefficient and slow while they will continue to be a great part of AI in the future.

Reinforcement Learning and Synopsys

Synopsys taps into reinforcement learning for its DSO.ai™ (Design Space Optimization (https://www.synopsys.com/glossary/what-is-design-space-optimization.html) AI) solution, which is the semiconductor industry's first autonomous artificial intelligence application for chip design. Inspired by DeepMind's AlphaZero that mastered complex games like chess or Go, DSO.ai uses RL technology to search for optimization targets in very large solution spaces of chip design. DSO.ai revolutionizes chip design by massively

scaling exploration of options in design workflows while automating less consequential decisions, allowing SoC teams to operate at expert levels and significantly amplifying overall throughput.

Continue Reading

Solution

Synopsys DSO.ai™

The world's first autonomous AI application for chip design.

Blog

How is Hardware Driving AI Innovation?

Discover what hardware is critical to enabling the real-time responses.

News

DSO.ai Design Space Optimization System Named "Innovative Product of the Year"

Webinar

Enabling Next-Generation SoC Design with Machine Learning-Driven Implementation

Related Term

What is an AI Accelerator?

Learn more about AI Accelerators, including how they work and their

benefits.

Related Term

What is Glitch Power?

Learn more about Glitch Power, including how it works and its benefits.

LEGAL

Privacy (/company/legal/privacy-policy.html)

Trademarks & Brands (/company/legal/trademarks-brands.html)

Software Integrity Agreements (/company/legal/software-integrity.html)

CORPORATE

About Us (/company.html)

Careers (/careers.html)

CSR Report (/company/corporate-social-responsibility.html)

Inclusion & Diversity (/careers/inclusion-diversity.html#present)

Investor Relations (/company/investor-relations.html)

Contact Us (/company/contact-synopsys.html)

FOLLOW

(https://www.synopsys.com/ai/what-is-reinforcement-learning.html#:~:text=The Reinforcement Learning problem involves,maximization of expected c… 15/15